



YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

# **Computer Programming and Inquiry-Based Experimentation in Science Education**

Zacharia C. Zacharias, Ph.D.

## **Table Of Contents**

Introduction.....	4
Unit 1: Inquiry-Based Physics Projects In Optics.....	5
Make Your Own Camera .....	6
Make Your Own Animated Movie .....	10
Invisible Animals .....	15
Are There Artificial Colors In Your Food? .....	21
Save Yourself From A Sunburn: A Scientific Way To Determine The Most Effective Suntan Lotion.....	27
Measuring The Diameter Of A Hair With A Laser .....	33
Unit 2: Inquiry-Based Physics Projects In Acoustics .....	39
Make Your Own Guitar .....	40
Make Your Own Bottle Organ.....	44
What Is The Difference Between Noise And Music?.....	49
Waves On A String? .....	53
Make Your Own Laser Show.....	59
Unit 3: Computer Programming In Science Education .....	65
Spreadsheets As A Scientific Tool: Using Spreadsheets To Analyze Multiloop Circuits	66
How To Create Your Own Webpages: An Introduction To Html .....	78
How To Create Your Own Computer Programs: An Introduction To C.....	91
Create Your Own Simulations!.....	116
Create Your Own Computer Games!.....	131
Create Your Own Windows Processes System Messages: An Introduction To Visual Basic.....	147
Learn How To Create Simulations By Using A Mapping Language .....	172
Using Spreadsheet Features As A Mapping Language To Simulate Science Concepts.	189
Create Your Own Computer Programs: An Introduction To Pascal .....	205



## **Introduction**

Science is perhaps unique as a subject in the curriculum of schools all over the world. This uniqueness results from the variety of materials and experiments necessary for its effective teaching. If it is to be learned effectively science must be experienced. “It must be learned and not learned about.”

Believing that science and scientific method of problem solving should play a significant role in science education, Lucent Technologies through a Science Grand Program, Commitment to High School Science Education, offers this book in the hope that will assist students in their efforts to understand everyday life phenomena related to optics and acoustics and how to use computer programming as an educational tool. It is also being offered with the hope that science educators can use it as a resource book for experiments in optics, acoustics, and computer programming. The underlying philosophy is that the science is most effectively taught and learned when students follow an inquiry-based experimental procedure.

The main criterion for the selection of the particular topics, introduced in this book, was to include experiments that interest students and challenge them at the same time.

We believe that we provided a broad and balanced range of ideas, skills and contexts through which students can construct scientific knowledge. However, no claim that this book is complete is made. But it is hoped that these pages will serve as a stimulus to both students and teachers to define their own science problems and then find the answers to their own questions through inquiry-based experimental procedure.

Zacharias C. Zacharia



*Computer Programming And Inquiry-Based Experimentation In Science Education.*  
*Zacharias C. Zacharia*

## **Unit 1: Inquiry-Based Physics Projects in Optics**

## **Unit 1: Optics - Images**

### **Project 1**

## **Make Your Own Camera**

Photos are indisputably a major part of our life. People everywhere in the world treasure them because of their personal meaning or use them in order to support their ideas, projects, stories etc. However, have you ever wondered how can moments or things related to our lives be captured on a piece of paper? Do you know what a camera is? Do you know what a film is? Why do we use film? How many times did you wonder how a camera works? Follow us through the process of creating your own camera and find out for yourself the answers to these questions or the ones that were troubling you so far!

### **Degree of Difficulty**

Experimental: Easy

Conceptual: Easy

### **Objectives**

Completion of the activities should enable you:

- to construct your own Pinhole Camera
- to explain the reasons for each decision you make regarding the construction of the camera (i.e. the size of the hole)
- to explain how a “clear” image can be formed
- to compare the image with the real object (i.e. size)

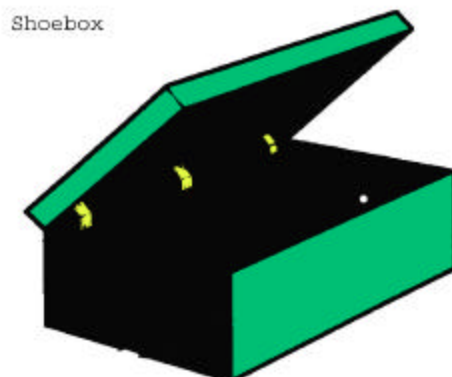
**Materials:** Shoebox, black paint or black paper, scissors, greaseproof paper (with grids if possible), stiff cardboard, pins, scotch-tape, meter stick, small convex lens (its size must be about 1 cm and its focal length can be anything).

### **Procedure**

#### **Constructing the camera**

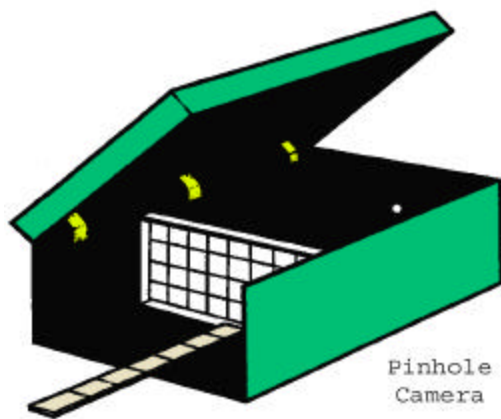
1. Cut one of the small sides out of the shoebox, but not the lid (see figure 1).
2. Paint the inside of the box and the lid black, or attached black paper along all sides of both the box and the lid (see figure 1).
3. Take the other small side of the box (the one that you did not cut out) and make a pinhole somewhere in the center (see figure 1).

**Figure 1**



4. Hinge the lid on the shoebox with scotch-tape (see figure 1). Have in mind that the lid will be opened several times in this experiment. Thus, it must easily open and close without destroying the other parts of the camera (cut the corners so you can easily raise the lid).
5. Place a piece of greaseproof paper (with one cm grids) on a stiff cardboard or wooden frame, so it can fit inside the box (see figure 2). The dimensions of the frame must be slightly less than the width and the height of the box. The frame must be put inside the box and it must be removable. If you have problems fixing the screen in parallel position with the small side of the shoebox, use some dark clay to fasten the two lower corners of the screen on the box. Another way of fixing the screen and being removable at the same time is to stick it on a small ruler (the ruler's length must not be more than half of your shoebox's length – otherwise it will be bothering your face when trying to see through it), (see figure 2). Can you think of another way of fixing the screen but at the same time being able to remove it without opening the lid? Include a description and an explanation of your idea in your final report.

**Figure 2**



(Smith and Holloway, 1985, p. 22)

6. The final setup must be: eye – greaseproof paper - pinhole - object.

## Activities

1. Point the camera to something bright and look at the picture. Draw the object and the picture/image. How does the image relate to the object? Is it larger or smaller? Is it upside down or right side up?
2. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).
  - How does that relationship change as we change the distance between the camera and the object? Measure the image sizes (how many grids does the image area cover?), as the distance between the camera and the object increases or decreases and plot these on a graph.
  - Does the brightness of the picture change, as the distance between the camera and the object increases or decreases?
3. Try making two more holes close to the first pinhole. Do you see any changes regarding the picture/image that is being formed on the screen (all three holes must be opened)? Try making one of the two additional holes bigger and the other smaller than the initial hole. By covering each time two of the holes, explain how the size of the uncovered pinhole affects what you see on the screen.
4. Move to a darkened room, where transparencies of various letters are projected on a big screen. Project one of the transparencies and before using your pinhole camera predict what you will see on the screen of your camera (Draw a sketch of your prediction). Does your prediction agree with what you actually see on your camera's screen? (This activity must be done in school, with help from your science teacher)
5. Look only in your pinhole camera and try to determine the word your teacher projects on the wall.
6. Make a hole of 1cm in diameter and place a convex lens in front of it. Find a screen position for a clear image. Can you find more than one point where you can get a

clear image? Follow once more steps 1 through 5. How do your results compare with the pinhole camera you have at first? Explain.

7. Visit the following website ([http://www.exploratorium.edu/light\\_walk/camera\\_todo.html](http://www.exploratorium.edu/light_walk/camera_todo.html)). The site will give you specific information on how to construct a different version of a pinhole camera. The advantage of this camera is that you can take pictures with it (special film must be purchased). Why can your shoebox camera not be used for taking pictures? In order to answer this question you will have to investigate what are the films made of (try both your school library and the internet).
8. Design and perform experiments to answer the following questions, regarding the camera you constructed in the previous activity. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion.
  - How does the time you leave your camera's hole open affect the quality of your pictures?
  - Does the brightness of the light surrounding the object you are taking the picture of, affects the quality of your pictures? (For this experiment, it is suggested to take the picture of the same object at different times of the day, i.e. early in the morning, at noon, in the afternoon, at night. Your source of light will be the sun or use a light with a dimmer switch or use a 3-way bulb).

## **Final Project/Report**

You have been asked to write the manual of the pinhole camera you have constructed at activity 7, by a major camera manufacturer. You have the freedom to make your own selections concerning the content of the manual. Do not forget to summarize all the experiments and the activities you have performed in this project. In addition, have in mind the manual must be descriptive and understandable by people that have never used a pinhole camera before. Therefore, sections about how to build the pinhole camera, how to use the camera, and how/why the camera works are strongly suggested to be included within your manual. Use of diagrams or pictures are strongly suggested. A camera manual would be a good point to start your research from. Have fun!

## **Reference**

Smith, G. and Holloway, G. (1985). *40 Science Activities*. London: Macmillan Education.

## Unit 1: Optics – Light & Color

### Project 2

## Make Your Own Animated Movie

Have you ever wondered how an animated movie is created? How do pictures become alive in front of our eyes? Follow us through the process of creating your own animated movie, by using a very simple device (stroboscope) that you will build yourself and find out the answers to these questions or the ones that were troubling you so far!

### Degree of Difficulty

Experimental: Easy

Conceptual: Moderate

### Objectives

Completion of the activities should enable you:

- to construct your own stroboscope
- to understand the principles underlying the stroboscope
- to be able to produce sets of pictures (flick-book figures) showing physical phenomena or physics laws
- to be able to explain the practical applications of a stroboscope

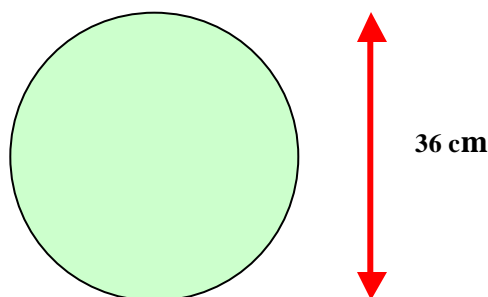
**Materials:** scissors, stiff cardboard, pins, scotch-tape, meter stick, a wooden handle (a piece of wood - 1.5cm x 2cm x 25cm), mirror, bolt, 2 washers, two nuts, one lock nut, permanent marker, drill.

### Procedure

#### Constructing the stroboscope

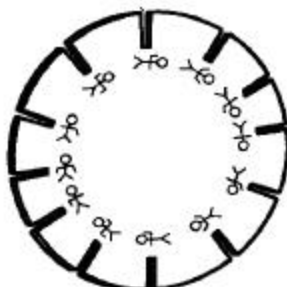
1. Take the stiff cardboard and cut a disk with a diameter of 36cm (see figure 1).

**Figure 1**



2. Mark off about twelve equal spaces along the periphery of the disk.
3. Cut a slit about 2x3cm (starting from the periphery – 3cm along the radius) for each one of the marked points of step 2 (see figure 2).
4. Draw a set of flick book pictures, one under every slit (about 1cm underneath the slit), with their head facing right (see figure 2).

**Figure 2**



5. Make a single hole at the center of the disk, equal to the diameter of the bolt that is available (You will use the bolt to attach the disk to the wooden handle).
6. Take the wooden handle (1.5cm x 2cm x 25cm) and drill a hole, equal to the diameter of the bolt that is available, at one of its ends.
7. Attach the center of the disk (after you make the slits and draw the flick book pictures) to the wooden handle (the disk will be parallel to the handle). First, insert the bolt into the wooden handle hole and fasten it with a nut. Second, pass the bolt that is already on the handle through the center hole of the disk. Third, fasten the disk with a second nut and lock the second nut with the use of a third nut (“lock nut”), (see figure 3).

**Figure 3**





8. Put your device (stroboscope) between one of your eyes and a mirror (the mirror does not have to be of particular size, but must be big enough for you to see clearly the flick-book pictures in it) with the revolving figures facing the mirror.
9. Hold with one hand the wooden handle and with the other one spin the disk. Look through the slits of the stroboscope at the pictures in the mirror, as the disk is rotating.

## Activities

1. Look through the slits of the stroboscope at the pictures in the mirror, as the disk is rotating. Change both the rates and the direction of rotation. What do you observe?
2. How does this phenomenon relate to slow motion photography or speeded up photography? (Physics books and internet would be helpful)
3. Draw a new set of flick book pictures, one under every slit (about 1cm underneath the slit), with their head now facing left instead of right. Do you observe any differences? How about if you draw the top towards the edge?
4. Use your stroboscope to observe a spinning wheel of a bicycle.
  - What do you observe when rotate your stroboscope in the same direction as the spinning wheel?
  - What do you observe when rotate your stroboscope in the opposite direction as the spinning wheel?
  - What physical value relates to the rotation rate of your stroboscope?
5. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).
  - Can you make the spinning wheel of a bicycle seem to stop turning? Can you make it seem to turn in the opposite direction it rotates (backwards)?
  - How does the angular velocity of the spinning wheel relate to the angular velocity of your stroboscope in each one of these two cases? We use the term *angular velocity* to refer to the rate with which the rotating body (i.e. wheel) is rotating. If

you have more questions regarding this physical value, ask your science teacher to help you.

- Is it possible to make the wheel seem to spin faster than it actually is?
6. Follow steps 1 to 4, and 8 to create a new stroboscobe. However, instead of drawing flick book pictures on the disk, divide one of its surfaces in three equal areas/pieces (it will look like a pie graph). Paint the first piece cyan, the second magenta, and the third yellow. Put your stroboscope between one of your eyes and a mirror. Look through one of the slits at one of the colored pieces.
    - Try to rotate the two wheels with the same angular velocity (at this point you will probably need someone to rotate the two wheels for you). What do you see? Are you able to see all colors? If no, how can you see the rest of the colors? Explain your reasoning and report any mathematical calculations you might make.
    - What if the disk is divided in more than 3 equal pieces? Will it take the same time as before to see a second colored area? Explain your reasoning and report any mathematical calculations you might make.
    - What do you see when the angular frequency of your stroboscope is much less than the angular velocity of the colored spinning wheel? Explain your reasoning. Change the colors of your colored wheel. Do you observe any differences? Explain.
  7. Create a second spinning wheel for your stroboscope, where you present a physical phenomenon. For example, show how a feather and a stone will move relatively to each other, when you drop them from a certain height. First, assume that the air resistance is negligible and then show how the two items will move with the presence of air resistance. You are encouraged to create more flick-book pictures for your own ideas.
  8. Create another spinning wheel for your stroboscope, where you use more than one color to color parts of the item you are sketching. For this part you have to be systematic. For example, start with one stationary item painted with one color, then use two colors (one every other picture), then use three colors (one every two pictures) etc. Make sure that each time you are adding a color you make the corresponding observation by rotating your stroboscope. It is suggested to create a two-column table where you report the colors you used for your picture and your observation.
  9. Design a set of flick-book pictures where you present a stationary wheel spinning.

## **Final Project/Report**

You have been asked to write the manual of the stroboscope, by a major company. You have the freedom to make your own selections concerning the content of the manual. Do not forget to summarize all the experiments and the activities you have



**YSAP**<sup>®</sup>  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

performed in this project. Have in mind the manual must be descriptive and understandable by people that have never used a stroboscope before. Therefore, sections about how to build the stroboscope, how to use the stroboscope, and how/why the stroboscope works are strongly suggested to be included within your manual. Use of diagrams or pictures are strongly suggested. At the end of your manual give examples of possible applications of the stroboscope in our everyday life.

## **Reference**

Smith, G. and Holloway, G. (1985). *40 Science Activities*. London: Macmillan Education.

## **Unit 1: Optics – Reflection & Refraction**

### **Project 3**

### **Invisible Animals**

Did you know that colorless animals can become invisible in water? Do you want to know how they do it? Follow this experiment and see not only the magic surrounding these animals, but also learn how to make items become invisible yourself!

### **Degree of Difficulty**

Experimental: Easy

Conceptual: Moderate

### **Objectives**

Completion of the activities should enable you:

- to understand Snell's law
- to explain the phenomenon of how colorless animals can “disappear” in water.
- to simulate the phenomenon by using a test tube made of glass and different solutions, such as, corn syrup, sugar solution etc.

**Materials:** clear corn syrup, water, graph paper, a semi-cylindrical petri dish, protractor, ruler, laser pen or light box, calculator, polar graph paper.

### **Part A: Procedure**

#### **Snell's Law**

1. Place the protractor on the graph paper and mark a point around its semi-circular side at  $15^{\circ}$ ,  $30^{\circ}$ ,  $45^{\circ}$ ,  $60^{\circ}$ ,  $75^{\circ}$ ,  $90^{\circ}$ ,  $105^{\circ}$ ,  $120^{\circ}$ ,  $135^{\circ}$ ,  $150^{\circ}$ , and  $165^{\circ}$ . Make sure that you mark a line at the center of the protractor (mid-point of line at  $0^{\circ}$ ).
2. Connect each point marked on the circumference of the protractor with the point at its center. The line at  $90^{\circ}$  is called “the normal.”
3. After you draw the lines, remove the protractor and place the dish on the paper. Make sure that the center of the flat side of the petri dish and the already marked point of the protractor's center match.
4. Fill the petri dish with water.

5. Place the laser pen or light box about two inches away from the petri dish and aim a beam of light from the laser pen at the curved side of the dish. Examine what happens when the beam strikes the dish at different angles. Starting from the normal try to match the beam coming out from the laser pen with the already marked lines on the graph paper. Sketch various arrangements of the dish and beam, both before it passes through the dish (*incident beam*) and after it passes through (*refracted beam*). In some cases you will see no refracted beam, but a reflected beam. This phenomenon is called *total internal reflection* and the angle between the reflected beam and the normal is called *angle of reflection*.
6. Place the petri dish of water on a sheet of polar graph paper. The center of the flat side of the dish must be positioned at the center of the paper. Arrange the light beam and dish as you did in step 5. Concentrate on the angle of incidence (angle between the normal and the incidence beam) and the angle of refraction (angle between the normal and the refracted beam). Note that when the beam of incidence matches with the normal, the angle of incidence is  $0^0$ . Use angles of incidence ranging from  $0^0$  to  $90^0$  in increments of  $5^0$ . Measure the angle of refraction for each case with the protractor and use your calculator to get the sine of an angle. Record your data in the following table:

**Table 1**

Angle of Incidence	Angle of Refraction	Sine of Angle of Incidence	Sine of Angle of Refraction

7. Make a graph of Sine of Angle of Incidence versus Sine of Angle of Refraction. Should your graph pass through the origin? Determine the slope of the graph and write an equation that relates the slope, Angle of Incidence, and Angle of Refraction.
8. Repeat steps 6 and 7, by changing the content of the petri dish. Use material, such as glass (for this one you will need a petri dish of solid glass), alcohol, corn syrup, vegetable oil, corn oil, salt water (try different concentrations), sugar and water (try different concentrations), and soda and air. Try it with different things in the tank and dish - water in tank, air in dish; air in tank, water in dish; etc for all fluids. You could try material of your own preference, as well. Record your data in the following table:

**Table 2**

Solution	Slope of the graph


9. Get a relatively big tank (i.e. ripple tank), which allows you to repeat steps 6 to 8, and put some water in it. Make sure that you use a waterproof laser pen before you put it in the tank. The level of water must not surpass the petri shell's height. Once, again repeat steps 6 to 8 and record your data in an identical table as the one presented at step 8.
10. Compare the two tables and explain any differences.

## Activities

1. At step 3 of the procedure we mention that “after you draw the lines, remove the protractor and place the dish on the paper. Make sure that the center of the flat side of the petri dish and the already marked point of the protractor's center match.” Would this also work with a circular petri dish, if the center of the dish is placed on the mark?
2. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).
  - Is the angle of reflection the same for all the substances you tested? Could this experiment be used as a method for identifying substances or materials? Explain.
  - Can a total internal reflection occur when a light beam in water is incident on air?
  - Does total internal reflection occur for a specific angle or for a range of angles? Give the angle or the range of angles for each substance you tested at step 8.

The slope of the graph of Sine of Angle of Incidence versus Sine of Angle of Refraction from step 8 is called the index of refraction of water with respect to air. The relationship between the Sine of Angle of Incidence versus Sine of Angle of Refraction is called the law of refraction or Snell's Law:

$$\text{index of refraction of inside stuff} \times \text{Angle of Refraction} = \text{index of refraction of outside stuff} \times \text{Angle of Incidence}$$

Given Snell's Law and that the **index of refraction of air** is equal to 1, interpret the slopes of the graphs you obtained at step 9.

## Part B: Procedure

### “Invisibility”

1. Place the petri dish on a sheet of polar graph paper. The center of the flat side of the dish must be positioned at the center of the paper and the line at  $90^0$  must be perpendicular to it.
2. Set the laser pen so that the angle of incidence is  $45^0$ . In this part of the experiment the angle of incidence will be kept constant. Therefore, it will be a good idea to fasten the laser pen at  $45^0$  with some masking tape.
3. Fill the semi-cylindrical plastic dish with clear corn syrup and pass the laser light through it. Measure the angle of refraction. Repeat the same process for different solutions by diluting corn syrup with water. The solutions should have different concentrations (90% of corn syrup to 10% of water, 80% of corn syrup to 20% of water, 70% of corn syrup to 30% of water ... 10% of corn syrup to 90% of water, and 100% water). Record your data in the following table:

**Table 3**

<b>Solution</b>	<b>Angle of incidence <math>T_i</math></b>	<b>Angle of refraction <math>T_r</math></b>	<b>Sine of Angle of Incidence</b>	<b>Sine of Angle of Refraction</b>	<b>Refractive index <math>n</math></b>

4. Describe how does the refractive index of corn syrup change as water is added.
5. Repeat steps 1 to 4 for other solutions (i.e. salt and water, and sugar and water), including different types of oils (corn oil, vegetable oil, olive oil). Do not mix water with oil. Why mixing water with oil will not work? However, if you want to dissolve them use ethanol.



6. Describe which of the substances give you the same visual result as the corn syrup.

## Activities

1. Take two glass test tubes. Fill two thirds of the first one with corn syrup and the other with water. Place both in a small beaker half filled with corn syrup. Explain what you observe by using concepts from the experiments you conducted in Part A and B.
2. Notice that the corn syrup test tube seems to disappear (see figure 1), while the test tube of water is clearly visible. Select two other substances that you used at step 5 of Part B to make one of the test tubes disappear.

**Figure 1**



(Sheppard, 2000)

3. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).
- Could the substance that is in the test tube that seems to disappear be different from the substance that is in the beaker?



- Could the test tubes be of different material than glass?
4. Explain how do you think colorless animals can “disappear” in water?

### **Final Project/Report**

Write a report summarizing all the experiments and the activities of both Part A and B. In addition, research on animals that can “disappear” in water. Does their method of becoming “invisible” agree with the prediction you made at the fourth activity of Part B.

### **Reference**

Sheppard, K. (2000). Index of refraction. *Scientific American*, Teacher’s kit, p. 6.

## **Unit 1: Optics – Spectrophotometry**

### **Project 4**

## **Are There Artificial Colors in Your Food?**

Do you want to know whether your food contains artificial color? Do you want to know if that color is not harmful to your health? Follow this experiment and see how a dye or dyes contributing to the color of a substance can be identified spectrophotometrically. Despite the fact that the term might sound complex, for the purpose of this experiment the process you will follow is very simple. Spectrophotometry involves the photometric (the measurement of the interaction of radiant energy with matter) comparison between parts of the spectra (ultraviolet and visible). The experiment is separated in two parts. The first part aims to familiarize you with the operation of the Spectrometer. The second part describes a simple method you can follow in order to identify dye(s) that might be in your favorite candy!

### **Degree of Difficulty**

Experimental: Moderate

Conceptual: Moderate to Difficult

### **Objectives**

Completion of the activities should enable you:

- to understand the function of the spectrometer 20D.
- to set the spectrometer to a specific wavelength.
- to extract the dye(s) from a substance and identify the dye using a spectrophotometer.

**Materials for Part A:** Spectrophotometer 20D, 2 cuvettes, Chalk, Water, Kimwipes. The Educational Absorption Spectra Kit is available from: Milton Roy Company, 820 Linden Avenue, Rochester, NY 14625, Phone: 800-654-9955 (Catalog # 333135, List Price: \$62.00)

**Materials for Part B:** candies, cuvettes, spectrophotometer, test tubes, distilled water.

### **Part A: Procedure**

#### **The digital spectrometer 20D**

1. Turn on the Spectrometer (turn the left front knob clockwise).
2. Turn the large knob (wavelength control knob) on the top slowly. What do you observe on the display? What does the number represent?
3. Adjust the wavelength to 450 nm (nm corresponds to *nanometers*, which is wavelength's unit of measurement – 1 nanometer =  $1 \times 10^{-9}$  meters). Place a cuvette with white chalk (or a white piece of plastic/rubber) into the *sample compartment* (if you have difficulty finding it, ask your teacher or use the manual). Look into the sample compartment and describe what you see. What do you observe as you increase the wavelength? How about when you decrease the wavelength?
4. Without removing the chalk, slowly turn the front right knob (Transmittance control knob) both towards right and towards left. What do you observe?
5. Set the wavelength to the lowest possible setting without removing the chalk in your cuvette. Increase the wavelength until you see light appearing on the chalk. This is the *lowest visible wavelength*. What light's wavelength is immediately beneath the lowest visible wavelength?
6. Remove the chalk cuvette and close the lid on the sample compartment. Turn the left front knob (Zero control knob) until the data value is 0.0. (The minus sign may flash on and off this is normal.)
7. Pour deionized water in a cuvette with to within 2 cm of the top (the water is called a *blank* because it does not have anything dissolved in it). Make sure that the bottom of the cuvette is clean. Why is it important to keep the cuvette clean? [The manufacturer, Milton Roy, suggests to handle cuvettes by touching them near the top by the mark – this mark should line up with the raised ridge on the front of the sample compartment, (Harris, 1993)]. Insert the blank (cuvette with deionized water) into the *sample compartment*. Make sure that the lid is closed.
8. Adjust the transmittance control until the right number reads 100.00. Press the mode button once. The data should read 0.00 (the spectrophotometer should now be set on absorbance). Remove your cuvette and set the mode to transmittance. Set the wavelength to 450 nm. Insert a blank and set transmittance to 100%. Adjust the wavelength to 550 nm. What has happened to the percent transmittance? [According to the manufacturer, Milton Roy, because the instrument is not equally sensitive to all wavelengths, it must be adjusted *every time you change wavelengths* – The transmittance control should be adjusted so that the instrument reads 100.00 transmittance or 0.00 absorbance when a blank is placed in the sample holder, (Harris, 1993)].

**Note:** The structure and procedure of introducing the Spectrophotometer 20D was based upon the description that was provided by the manufacturer's manual (Harris, 1993).

## Activities

1. Repeat steps 1 to 5. Looking at the chalk, determine the range of wavelengths for each color of light and record your data in the following table:

**Table 1**

Color of Light	Wavelength-Lowest Value	Wavelength-Highest Value
Red		
Orange		
Yellow		
Green		
Blue		
Indigo		
Violet		

Using your data, describe the relationship between color and wavelength. Have you seen colors appear in this order before?

2. In addition to the markings, is there anything that has to be more carefully made on a cuvette than on test tubes? Explain.
3. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).
  - Do different solutions absorb light of different colors or wavelengths?
  - Does the wavelength of light that a solution absorbs best depend on the color of the solution?
  - Does the amount of light absorbed depend on concentration?
  - What would be an approximation for the wavelength of the infrared or ultraviolet light? Is it possible to approximate wavelengths smaller than the wavelength of the ultraviolet light or bigger than the wavelength of the infrared light?

- What is the wavelength of maximum absorbance of a colored solution? (Help: Start with 350 nm. Set to 0% transmittance, place the blank into the *sample compartment* and set to zero absorbance. Remove the blank and place the colored solution into the unit. Record the absorbance in a table. Repeat the same process in increments of 25nm. Locate the 50nm region in which the absorbance is highest and repeat the process in increments of 10nm. By plotting absorbance versus wavelength, it will be easier for you to find the maximum absorbance – it's the point where your graph peaks. If you do not have a graphing program, such as, Cricket Graph, use Microsoft Excel.)

## **Part B: Procedure**

As you have seen in the previous part of this experiment, each particular color/dye has a known wavelength where the absorbance of light is at a maximum. By scanning the entire visible spectrum one can determine the wavelength of maximum absorbance for a particular dye. After determining the wavelength of maximum absorbance, the identification of a dye can easily be done by referring to a list of approved dyes and their wavelengths of maximum absorbance.

### **Identifying food dye(s)**

1. Prepare a solution of the food dye. To make a solution of the candy (M&M or Skittle are suggested), place it in a test tube with distilled water, shake gently, and then filter quickly.
2. Turn on the spectrophotometer. The manufacturer suggests allowing 15 minutes before use for warming up. With nothing in the sample compartment, set the transmittance to 0%.
3. Place 3-4 mL of water (if you are using a different solvent, use that one as blank) into a cuvette to serve as a blank.
4. Insert the blank into the spectrophotometer and set the wavelength to 350 nm. Set the absorbance to zero.
5. Replace the blank with the cuvette containing the sample and record the absorbance.
6. Repeat steps 4 and 5 changing the wavelength to 375 nm. Continue recording absorbances for increments of 25 nm until you reach 700 nm. The following table is suggested to be used for recording your data:

**Table 2: Identification of Food Dyes**

Wavelength (nm)	Absorbance
350	
375	
400	
425	
...	
...	
675	
700	

- Use a reference table (Merck Index), which provides the names of the dyes and the wavelengths of maximum absorbances to identify the dyes you were testing at step 6. Record your data in the following table:

**Table 3**

Dye#	Wavelength of maximum absorbance	Name of the dye

For a reference table use the list that is provided by Marmion (1991). The list will provide you with the name of the dye, the solvent system that was used, and the maximum wavelength.

## Activities

- Repeat steps 1 to 7 for food coloring (add one drop of food coloring to 100 mL of distilled water). Use as many colors as possible.
- Explain why was it necessary to scan the blank solution at step 5? If we skip this step how would our experiment be affected?
- Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most

critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions.

- Would this method we are using for identifying dye(s) in food be effective in separating two dyes that are very similar in color?
- Try 2 food colors at once. Do you get 2 peaks that can be used to identify both?
- Does a mixture of different dyes, that corresponds to an already existing dye you have available, have the same wavelength of maximum absorbance with the available dye that matches its color?

## **Final Project/Report**

You have been asked by the Consumer's Union to investigate 20 different brands of candies and report whether they fulfill the federal standards by reporting the dye(s) they are using. You have the freedom to make your own candy selections. Do not forget to summarize all the experiments and the activities you have performed in this project. Have in mind the report must be descriptive and understandable by people that have never done this experiment. Use of tables, diagrams or pictures is strongly suggested. At the end of your report give examples of other possible applications, of this method of identifying dye(s) in food, in our everyday life.

## **References**

Harris, E. and Anderson, M. (1993). *Spectrophotometry Made Simple*. Miltor Roy

Marmion, D. M. (1991). *Handbook of U.S. Colorants – Foods, Drugs, Cosmetics and Medical Devices*. Wiley and Sons.

**Unit 1: Optics – Spectrophotometry**  
**Project 5**

**Save Yourself From a Sunburn: A scientific Way to Determine the  
Most Effective Suntan Lotion**

Do you want to know whether the suntan lotion you are using to protect you from sunburns is the most effective in blocking the sun radiation that causes them? Do you want to know if you are using the right protection factor? Follow this experiment and see how an effective sunscreen (one that completely blocked the radiation in the vicinity of the wavelength that causes sunburns) can be identified spectrophotometrically. Despite the fact that the term might sound complex, for the purpose of this experiment the process you will follow is very simple. Spectrophotometry involves the photometric (the measurement of the interaction of radiant energy with matter) comparison between parts of the spectra (ultraviolet and visible). The experiment is separated in two parts. The first part aims to familiarize you with the operation of the Spectrometer. The second part describes a simple method you can follow in order to identify the suntan lotion that will better protect you from getting sunburned.

**Degree of Difficulty**

Experimental: Moderate

Conceptual: Moderate to Difficult

**Objectives**

Completion of the activities should enable you:

- to understand the function of the spectrometer 20D.
- to set the spectrometer to a specific wavelength.
- to determine the most effective suntan lotion through spectrophotometric analysis.

**Materials for Part A:** Spectrophotometer 20D, 2 cuvettes, Chalk, Water, Kimwipes.

The Educational Absorption Spectra Kit is available from: Milton Roy Company, 820 Linden Avenue, Rochester, NY 14625, Phone: 800-654-9955 (Catalog # 333135, List Price: \$62.00)

**Materials for Part B:** “PABA free” suntan lotions (they can be dissolved with *n*-propanol), cuvettes, spectrophotometer, *n*-propanol, waterproof and not waterproof lotions, deionized water.



## Part A: Procedure

### The digital spectrometer 20D

1. Turn on the Spectrometer (turn the left front knob clockwise).
2. Turn the large knob (wavelength control knob) on the top slowly. What do you observe on the display? What does the number represent?
3. Adjust the wavelength to 450 nm (nm corresponds to *nanometers*, which is wavelength's unit of measurement – 1 nanometer =  $1 \times 10^{-9}$  meters). Place a cuvette with white chalk (or a white piece of plastic/rubber) into the *sample compartment* (if you have difficulty finding it, ask your teacher or use the manual). Look into the sample compartment and describe what you see. What do you observe as you increase the wavelength? How about when you decrease the wavelength?
4. Without removing the chalk, slowly turn the front right knob (Transmittance control knob) both towards right and towards left. What do you observe?
5. Set the wavelength to the lowest possible setting without removing the chalk in your cuvette. Increase the wavelength until you see light appearing on the chalk. This is the *lowest visible wavelength*. What light's wavelength is immediately beneath the lowest visible wave length?
6. Remove the chalk cuvette and close the lid on the sample compartment. Turn the left front knob (Zero control knob) until the data value is 0.0. (The minus sign may flash on and off this is normal.)
7. Pour deionized water in a cuvette with to within 2 cm of the top (the water is called a *blank* because it does not have anything dissolved in it). Make sure that the bottom of the cuvette is clean. Why is it important to keep the cuvette clean? [The manufacturer, Milton Roy, suggests to handle cuvettes by touching them near the top by the mark – this mark should line up with the raised ridge on the front of the sample compartment, (Harris, 1993)]. Insert the blank (cuvette with deionized water) into the *sample compartment*. Make sure that the lid is closed.
8. Adjust the transmittance control until the right number reads 100.00. Press the mode button once. The data should read 0.00 (the spectrophotometer should now be set on absorbance). Remove your cuvette and set the mode to transmittance. Set the wavelength to 450 nm. Insert a blank and set transmittance to 100%. Adjust the wavelength to 550 nm. What has happened to the percent transmittance? [According to the manufacturer, Milton Roy, because the instrument is not equally sensitive to all wavelengths, it must be adjusted *every time you change wavelengths* – The transmittance control should be adjusted so that the instrument reads 100.00 transmittance or 0.00 absorbance when a blank is placed in the sample holder, (Harris, 1993)].

**Note:** The structure and procedure of introducing the Spectrophotometer 20D was based upon the description that was provided by the manufacturer's manual (Harris, 1993).

## Activities

1. Repeat steps 1 to 5. Looking at the chalk, determine the range of wavelengths for each color of light and record your data in the following table:

**Table 1**

Color of Light	Wavelength-Lowest Value	Wavelength-Highest Value
Red		
Orange		
Yellow		
Green		
Blue		
Indigo		
Violet		

Using your data, describe the relationship between color and wavelength. Have you seen colors appear in this order before?

2. In addition to the markings, is there anything that has to be more carefully made on a cuvette than on test tubes? Explain.
3. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).
  - Do different solutions absorb light of different colors or wavelengths?
  - Does the wavelength of light that a solution absorbs best depend on the color of the solution?
  - Does the amount of light absorbed depend on concentration?
  - What would be an approximation for the wavelength of the infrared or ultraviolet light? Is it possible to approximate wavelengths smaller than the

wavelength of the ultraviolet light or bigger than the wavelength of the infrared light?

- What is the wavelength of maximum absorbance of a colored solution? (Help: Start with 350 nm. Set to 0% transmittance, place the blank into the *sample compartment* and set to zero absorbance. Remove the blank and place the colored solution into the unit. Record the absorbance in a table. Repeat the same process in increments of 25nm. Locate the 50nm region in which the absorbance is highest and repeat the process in increments of 10nm. By plotting absorbance versus wavelength, it will be easier for you to find the maximum absorbance – it's the point where your graph peaks. If you do not have a graphing program, such as, Cricket Graph, use Microsoft Excel.)

## Part B: Procedure

Sunburn is caused most severely within the range of 285-315 nm. An effective sunscreen would be one that completely blocked the radiation of these wavelengths (Harris, 1993). In order to be effective, active ingredients of the suntan lotion must exhibit an intense absorption maximum centered around 300 nm while transmitting the longer wavelength tanning radiation (It could also block the tanning wavelengths. There's no requirement that you end up tanned).

Your experience with the spectrophotometer from Part A, can be used to find the absorbance of sun lotions at various wavelengths. Despite the fact that absorption below 320 nm can not be obtained by using spectrophotometer 20D, general trends in absorption can be observed graphically and decisions can be made about the lotions' effectiveness (Harris, 1993).

### Determining effective sunscreen

1. Prepare a solution of the suntan lotion. To make the solution, n-propanol must be used as a solvent. Pour some suntan lotion in a test tube with n-propanol and shake gently (Precision isn't important, but it shouldn't be too dilute for the 20D to detect it, or so concentrated that there's no transmission). After the solution is ready pour 5mL in a cuvette.
2. Turn on the spectrophotometer. The manufacturer suggests allowing 15 minutes before use for warming up. Using n-propanol as a blank, set the wavelength to 420 nm and zero the instrument.
3. Replace the blank with the cuvette containing the sample and record the absorbance.
4. Continue to take readings at 10 nm intervals to the lowest wavelength your spectrophotometer is giving you. Remember to zero the instrument using the blank at every wavelength. Record your data in the following table:

**Table 2**

Wavelength (nm)	Absorbance
420	
410	
400	
390	
380	
370	
360	
...	

5. Use the data you collected in step 4 and graph absorbance versus wavelength.

## Activities

1. “Despite the fact that absorption below 320 nm can not be obtained by using spectrophotometer 20D, general trends in absorption can be observed graphically and decisions can be made about the lotions’ effectiveness.” Explain why this is a good assumption. I.e., why it’s not likely that the sunscreen you buy isn’t a notch filter that blocks only the burning wavelengths?
2. Repeat steps 1 to 5 for different brands of suntan lotion. Use as many as possible.
3. Why did we use n-propanol for blank and not water?
4. Repeat steps 1 to 5 for different brands of non-waterproof suntan lotions. What solvent shall we use in this case?
5. Explain why was it necessary to scan the blank solution at step 2? If we skip this step how would our experiment be affected?
6. Design and perform experiments to answer the following questions. Your experiment’s lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions.
  - Do the suntan lotions of a particular brand, but of different protection factor, have the same sunscreen effectiveness? Check various brands (at least five).
  - Do suntan lotions of different brands, but of the same protection factor, have the same sunscreen effectiveness? Compare all possible combinations (i.e. compare all the brands you have with protection factor 10 and then compare all the brands you have with protection factor 20, 30 etc.)



**YSAP**<sup>®</sup>  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

## **Final Project/Report**

You have been asked by the Consumer's Union to compare 5-10 different brands of suntan lotions and report which of them are the most effective for each particular protection factor and at the most affordable price. You have the freedom to make your own suntan lotion selections. Remember to summarize all the experiments and the activities you have performed in this project. Have in mind the report must be descriptive and understandable by people that have never done this experiment. Use of tables, diagrams or pictures is strongly suggested. The first part of your project report must summarize how suntan lotions are made, which are the active ingredients of the suntan lotion that exhibit an intense absorption maximum centered around 300nm, how do we obtain different protection factors, and why do we need different protection factors.

## **Reference**

Harris, E. and Anderson, M. (1993). *Spectrometry Made Simple*. Miltor Roy

## Unit 1: Optics – Diffraction

### Project 6

## Measuring the Diameter of a Hair with a Laser

Have you ever wondered how much is the width of your hair? Do you want to know how to get a very good estimation of its diameter? Follow this experiment and see how to measure the diameter of a hair and wire using a He-Ne laser. The experiment is separated in two parts. The first part aims to familiarize you with the operation of the laser and to measure the wavelength that the laser is emitting. The second part describes a simple method you can follow in order to measure the diameter of a hair and wire. It is suggested that you study diffraction before performing this experiment, even though it is not necessary for the procedure of the experiment, in order to have a better conceptual understanding of the phenomenon.

### Degree of Difficulty

Experimental: Moderate to Difficult

Conceptual: Difficult

### Objectives

Completion of the activities should enable you:

- to measure the wavelength of light from a He-Ne laser.
- to measure the wavelength of light from a He-Ne laser and measure the diameter of a hair and wire using a He-Ne laser, steel rule, and meter stick.

**Materials:** He-Ne laser, hair, steel ruler (instead of diffraction grating), white screen, meter stick, computer, Microsoft EXCEL (or any other software with graphing capabilities).

### Part A: Procedure

#### The wavelength of light from a He-Ne laser

When lights from two different sources cross they can interfere with each other to produce an alternating pattern of *bright and dark* regions. If the wave peaks are *in phase* there will be *constructive interference* and the *intensity* will be a maximum (bright region on the screen), while if they are exactly out of phase the intensity will be zero (dark region on the screen). The interference can be generated in a number of ways. Every point on a *wave front* could be considered as the source of a new wave spreading out in all directions at the *velocity* of the wave, with the new wave front being the envelope of all the *wavelets* (Rutgers University Laboratory Manual, 2000). If a smooth wave front is

interrupted in some way, the missing wavelets will lead to a new wave front exhibiting a diffraction pattern. A diffraction pattern is easily seen using a *diffraction grating*, a series of regularly spaced scratches or absorbing lines, which serve to remove parts of the wave front in a *periodic fashion* (Rutgers University Laboratory Manual, 2000). The *maxima* in the diffracted spectrum are given by

$$d [\sin(\theta_m) - \sin(\theta_i)] = m\lambda$$

where  $d$  is the spacing between grating lines,  $\lambda$  is the wavelength of the light,  $m$  is the diffracted order,  $\theta_i$  is the angle of incidence, and  $\theta_m$  is the angle of diffraction of the  $m^{\text{th}}$  order maximum, measured from the normal. This is the formula that will be used in the first part of our experiment for determining the laser light's wavelength. Thus, in terms of our data collection we have to measure  $\theta_m$ ,  $\theta_i$ ,  $\lambda$ , and  $m$  ( $d$  is a constant).

1. Set up the laser approximately 1m away from the screen.
2. Take the steel ruler (it must have gratings of mm) and place it between the laser and the screen. The ruler must be at the same height and nearly parallel to the light ( $\theta_m$  and  $\theta_i$  are measured relative to a line perpendicular to the ruler, which means they will be near  $90^\circ$ ).
3. By trial and error change screen's position until you find a place where the diffracted spectrum is clear enough to be studied.
4. After setting up the laser at the appropriate location make the necessary measurements, needed to determine the wavelength, of  $\theta_m$ ,  $\theta_i$ ,  $d$  and  $m$  (use several maxima and average the values to get the final wavelength):
  - $\theta_m$  and  $\theta_i$  will not be calculated directly. Instead you will measure  $D$  (distance from grating lines to screen – this measurement is taken only once because the distance from grating lines to screen does not change) and  $X$  (distance from the level of the steel ruler and the  $m^{\text{th}}$  maxima/bright spot), with a meter stick and then use the trigonometric identity

$$\theta_m = 90^\circ - \tan^{-1}(X/D) \quad (1)$$

- The number of maxima  $m$  was counted directly from the screen (try to get a big number of maxima. Why?).
- Record your data in the following table:

**Table 1**

Ruler-screen distance (D)	Grating width (d)	Angle of incidence (q <sub>i</sub> )	Sin(q <sub>i</sub> )	
<b>m</b>	<b>X<sub>m</sub></b>	<b>Tan(q<sub>m</sub>)</b>	<b>q<sub>m</sub></b>	<b>Sin(q<sub>m</sub>)</b>

5. For getting a value for  $\lambda$  use the formula

$$\sin(q_m) = m\lambda / d + \sin(q_i) \quad (2)$$

Graph  $\sin(q_m)$  versus  $m$  by using Excel. The graph should be a straight line, and from the slope (**slope** =  $-\lambda / d$ ) you can calculate  $\lambda$ . Since only the slope of the line is of our interest for determining  $\lambda$ , the constant factor  $\sin(q_i)$  of the formula can be ignored.

## Activities

1. Briefly summarize the phenomenon you observed in this experiment. Emphasize the reasons why a particular pattern of bright and dark spots appears on the screen.
2. Why can the constant factor  $\sin(q_i)$  of formula (2) be ignored, when graphing  $\sin(q_m)$  versus  $m$ ?
3. How will the graph change if the grating width  $d$  is increased? How will the graph change if the grating width  $d$  is decreased? Is there a limit in terms of how big or how small  $d$  can get, if we want the bright and dark pattern on the screen to be repeated? Explain.
4. Why do we use several maxima and graph the values to get the final wavelength? Explain.

## Part B: Procedure

A single slit will also give a diffraction pattern, with the *minima* (dark spots) in intensity given by

$$B \sin(q_n) = n \lambda \quad (3)$$



where ***B*** is the slit width (in our case it will be the width of the hair), ***n*** the diffraction order, and ***q<sub>n</sub>*** the diffraction angle to the ***n***<sup>th</sup> minimum. The same pattern is formed by an opaque strip of width ***B***. This is the formula that will be used in the second part of our experiment for determining the width of a hair (Halliday, Resnick, & Walker, 1997).

### Determining the width of a hair

1. Position one of your hairs in front of the laser (you can fix it on the laser or use a stand). When fastened make sure that the hair is straight.
2. By trial and error put the screen at a place where the diffracted spectrum is clear enough to be studied.
3. After setting up the laser at the appropriate location make the necessary measurements, needed to determine the wavelength, of  $\theta_n$ , ***B*** and ***n*** (use several minima and average the values to get the final wavelength):
  - $\theta_n$  will not be calculated directly. Instead you will measure ***D*** (distance from the hair to screen – this measurement is taken only once because the distance from the hair to screen does not change) and ***X*** (distance from the central maxima to the ***n***<sup>th</sup> minima), with a meter stick and then use the trigonometric identity

$$q_n = \tan^{-1}(X_n/D) \quad (4)$$

- The number of minima (***n***) was counted directly from the screen (try to get a big number of minima. Why?).
- Record your data in the following table:

**Table 2**

Hair-screen distance ( <b><i>D</i></b> )	Hairwidth ( <b><i>B</i></b> )			
<b><i>n</i></b>	<b><i>X<sub>n</sub></i></b>	<b>Tan(<i>q<sub>n</sub></i>)</b>	<b><i>q<sub>n</sub></i></b>	<b>Sin(<i>q<sub>n</sub></i>)</b>

4. For getting a value for  $\lambda$  use the formula

$$\sin(q_n) = n\lambda / B \quad (3)$$

Graph  $\sin(q_n)$  versus  $n$  by using Excel. The graph will give you a straight line, and from the slope ( $\text{slope} = \lambda / B$ ) you can calculate  $\lambda$ .

## Activities

1. Repeat the same experiment with wires of different diameters. Why do you get the screen pattern you observed in the hair experiment only with opaque strips (thin wires)?
2. Briefly summarize the phenomenon you observed in this experiment. Emphasize the reasons why a particular pattern of bright and dark spots appears on the screen.
3. How will the graph change if the hair width  $B$  is increased? How will the graph change if the hair width  $B$  is decreased? Is there a limit in terms of how big or small  $B$  can get, if we want the bright and dark pattern on the screen to be repeated? Explain.
4. Why do we use several minima and graph the values to get the final wavelength?
5. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).
  - How does the pattern on the screen change as the diameter of the wire changes (use thin wires)? Explain.
  - What will happen if I put two hairs next to each other (vary the distance between the two hairs)? Explain.
  - What will happen if I put three hairs next to each other (vary the distance between the two hairs)? Explain.
  - How does the distance between the minima change? Explain.

- Is the number of minima finite? Explain.
6. When a small source of white light illuminates a compact disk (CD), colored “lanes” are observed. Explain the phenomenon. (Hint: Use concepts from both parts of the experiment and search for applications of *Diffraction Gratings*).

## **Final Project/Report**

Write a report summarizing all the experiments and the activities of both Part A and B and include information on the applications of diffraction in science (i.e. “cleaning” of a satellite picture). Have in mind that your report must be descriptive and understandable by people that have never conducted a similar experiment. Use of diagrams, tables or pictures is strongly suggested.

## **References**

Halliday, D., Resnick, R., & Walker, J. (1997). *Fundamental of Physics*. NY: John Wiley & Sons, Inc.

Rutgers University Laboratory Manual (2000). *Computer Experimentation*. New Brunswick: Rutgers University The State University of New Jersey.

## **Unit 2: Inquiry-Based Physics Projects in Acoustics**

## **Unit 2: Acoustics – Standing Waves on a String**

### **Project 7**

### **Make Your Own Guitar**

Have you ever wondered why a guitar produces sound? Do you want to know how to make one yourself and discover how sound is produced by plucking its strings? Follow this experiment and see how to make a playable guitar.

#### **Degree of Difficulty**

Experimental: Moderate

Conceptual: Moderate to Difficult

#### **Objectives**

Completion of the activities should enable you:

- to construct your own guitar
- to understand the relationships among the length, tightness, and diameter of a string and the note produced when it is plucked.
- to identify the note produced when the string is plucked
- to predict the effect of changes of the length, tightness, and diameter of a string and apply their knowledge to making a playable guitar.

**Materials:** a guitar, a piece of wood (60x50x2.5 cm), guitar strings, 20 weights (0.5kg-10kg, increments of 0.5kgrams), three 4cm lengths of 1cm wooden dowel per string used, hammer, nails, computer, sound sensor (universal lab interface is needed), “Interactive Physics” software or “Macmotion” software.

#### **Procedure**

##### **Constructing the guitar**

1. Put the 60x50x2.5 cm piece of wood on a table.
2. Hammer one nail into the wood every other 5 cm, along one of the smaller sides (8 nails in total). The nail must be about 1 cm away from the periphery of the smaller side. Tie a small loop at the end of the string (any string) and loop it over a nail. Tie a weight (1 kg) to the other end of the string and let it hang over the edge of the table (align the smaller side that is free of nails with one of the tables edges), (see figure 1).

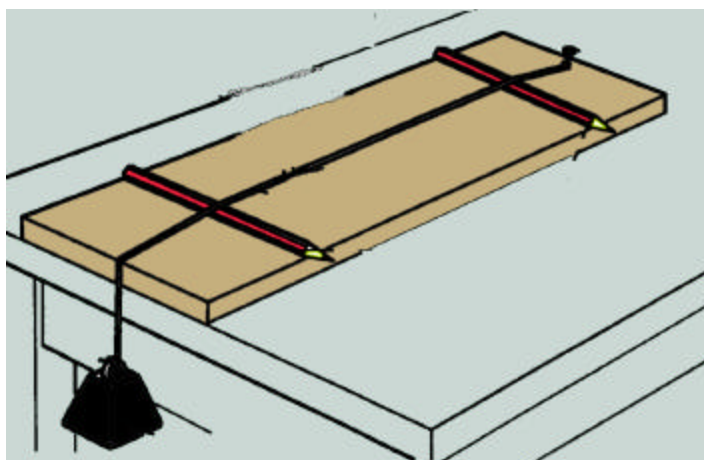


**YSAP**<sup>®</sup>  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

- Put two dowels under the string about 40 cm apart (see figure 1).

**Figure 1**



- Connect the universal lab interface to the computer and the sound sensor to the interface (use the manual or ask your teacher to help you at this stage). Start the computer and open the Interactive Physics Software. Select the real time data collection program and set it to graph frequency versus time (this program collects data and graphs them at the same time) for the first 10 sec.
- Put the sound sensor couple of centimeters away from the center of the string Pluck the string and see the graph that appears on the screen. What does the graph look like? Why does it look like that? What is the frequency of the produced note?
- Repeat step 5 as you move the dowels to different positions. Use the same string, the same weight, and vary the length. Start with the dowels being 5 cm apart and use increments of 5cm. Record your data in the following table:

**Table 1**

Length of the string (distance between the two dowels)	Frequency

Graph frequency versus length (use Microsoft Excel). What does the graph look like? How does the length of the string relate to the frequency? What happens to the note (i.e. loud, soft, high, low etc.)?

## Activities

1. Repeat steps 1 to 6, but use strings of different diameters this time. Use the same length, the same weight, and vary the diameter of the string (use as many strings of different diameter as you have). Compare the resulting graphs of step 6 among each other. Explain your observations. Does the frequency depend upon the diameter of the string? What happens to the note (i.e. loud, soft, high, low etc.)?
2. Repeat steps 1 to 6, but use different weights. Use the same string, the same length, and vary the weight (From 0.5 kg to 10 kg. Use increments of 0.5 kg). Compare the resulting graphs of step 6. Explain your observations. Does the frequency depend upon the tightness of the string? What happens to the note (i.e. loud, soft, high, low etc.)?
3. Describe what happens when a string is plucked. Connect this to how sound is produced and how that sound reaches your ear. In addition, explain how our ears work.
4. How does the volume of the sound change when you place your guitar above an empty box?
5. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).
  - How does the strength of plucking a string relate to the volume of the sound?
  - How does the strength of plucking a string relate to the frequency of the sound?
  - Where do you have to press, along a string, so that you get each note of the scale (doh, ray, me, fah, soh, la, te, doh)? (Hint: Use a real guitar and the Interactive Physics Program, as you did at step 5. Save the graph each note of the real guitar produces and then try to match each note with your one string guitar).
  - Does the whole string vibrate or just the part between the dowels?
  - Suggest how to tune your instrument by using the software you used in this experiment.

- Is it possible to get each note of the scale (doh, ray, me, fah, soh, la, te, doh), with the weights that we have available? The length and diameter must be kept constant.
- Is it possible to get each note of the scale (doh, ray, me, fah, soh, la, te, doh), with the different string diameters that we have available? The length and weight must be kept constant.
- Is it possible to get each note of the scale (doh, ray, me, fah, soh, la, te, doh) by varying only the length of the string?
- What will be the most effective combination of the three variables (length, diameter, tightness) for getting notes of the same frequency as the one-string guitar you made by just varying the length of the string, if we want to have the smallest guitar (in length) possible?

6. How do you think you play higher frequencies on a guitar? Explain.

7. How do you think you tune a string on a guitar? Explain.

8. Explain the importance of the guitar box?

## **Final Project/Report**

You have been asked to construct and write the manual of an eight string guitar (one string for each note), by a major guitar manufacturer (Use the eight nail piece of wood. It is up to you to decide which of the three variables – length, diameter, tightness – will be varied in order to achieve the eight string guitar). You have the freedom to make your own selections concerning the content of the manual. Remember to summarize all the experiments and the activities you have performed in this project. In addition, have in mind the manual must be descriptive and understandable by people that have never used a guitar before. Therefore, sections about how to build the guitar, how to use the guitar, and how/why the guitar works are strongly suggested to be included within your manual. Use of diagrams or pictures is strongly suggested.

## **Reference**

Smith, G. and Holloway, G. (1985). *40 Science Activities*. London: Macmillan Education.



## **Unit 2: Acoustics –Waves in a Vibrating Column of Air**

### **Project 8**

### **Make Your Own Bottle Organ**

Hollow pipes have long been used for making musical sounds. Organ pipes, flutes, and whistles produce sound in the same way. Do you want to know how they work? Follow this experiment and by examining the behavior of air in a bottle that has one of its ends closed off, create your own musical instrument.

### **Degree of Difficulty**

Experimental: Moderate

Conceptual: Moderate to Difficult

### **Objectives**

Completion of the activities should enable you:

- to construct your own bottle organ.
- to understand the relationships between the length of empty space, width, and shape in a bottle and the note produced.
- to identify the note produced.
- to predict the effect of changes of the length of container's empty space, the substance of the container, and shape of the container and apply their knowledge to making an organ.

**Materials:** a flute, 8 plastic or glass bottles, at least one bottle (or bottle shaped container) made of different material than glass or plastic (i.e. glass or plastic, stainless steel, aluminum etc), at least five identical bottles of the same material but of different shapes, permanent marker, water, alcohol, cotton, pipettes (longer than the height of the bottles you are using), liquids of different densities (oil, salt water, corn syrup etc.), computer, sound sensor (universal lab interface is needed), “Interactive Physics” software or “Macmotion” software.

### **Procedure**

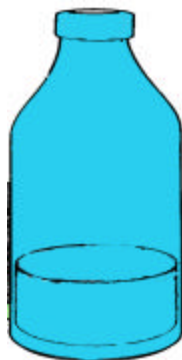
#### **Constructing the bottle organ**

If you blow air across the open end of a bottle, the disturbance due to the moving air at that end propagates along the bottle to the far end. Since one of the bottle's ends is closed, the air is not free to move any further in that direction and the closed end becomes a *node* (point of zero amplitude). However, the sound that is produced by our blow has to

do with the fact that an anti-node (point of maximum amplitude) is being formed at the open end of the bottle.

1. Take an empty bottle (plastic or glass) and fill half of it with water (see figure 1).

**Figure 1**



2. Put some alcohol on a piece of cotton or cloth and clean the bottle top (this is for killing any germs that are on the bottle – do not exchange bottles with other people, if you do so, make sure that you clean it with alcohol before putting it in your mouth).
3. After you make sure that the bottle is clean, press the bottle top to your chin just below your lower lip and blow steadily across the top. If the bottle does not “sing”, add water (as much as needed to raise the level of the water by couple of millimeters) and blow again. If you still do not get a note, repeat the same process (of adding water) until you finally hear a sound.
4. After your bottle “sings” for the first time, keep adding water (as much as needed to raise the level of the water by 4-5mm) and testing it until you get a second note. Repeat the same process and get as many notes as possible.
5. Connect both the universal lab interface to the computer and the sound sensor to the interface (use the manual or ask your teacher to help you at this stage). Start the computer and open the Interactive Physics Software. Select the real time data collection program and set it to graph frequency versus time (this program collects data and graphs them at the same time) for the first 10 sec.
6. Take an empty bottle and repeat steps 2 and 3. Use the sound sensor to graph frequency versus time for each note (since there is not only one point where the bottle “sings”, take the note with the highest amplitude at that particular water level. For this reason you will need to add or remove small quantities of water with a long pipette until you get the note). Put the sound sensor a couple of centimeters away from the top of the bottle and save the graph that appears on the screen for each note. What does the graph look like? Why does it look like that? What is the frequency of the produced note? In addition, use the permanent marker to mark the level of the water

on the bottle for each note. Repeat the same process and get as many notes as possible. Record your data in the following table:

**Table 1**

Length of empty space	Frequency

Graph frequency versus length (use Microsoft Excel). What does the graph look like? How does the length of the empty space relate to the frequency? What happens to the note (i.e. loud, soft, high, low etc.)?

## Activities

1. Repeat steps 1 to 6, but use bottles of different material this time. Use the same height, the same shape, the same liquid (water), and vary the material that the bottle is made of (use as many bottles of different material as you have). Compare the resulting graphs of step 6 among each other. Explain your observations. Does the frequency depend upon the material the bottle is made of? Can you explain why it does or does not depend on the material the bottle is made of? What happens to the note (i.e. loud, soft, high, low etc.)?
2. Repeat steps 1 to 6, but use different shapes. Use the same height, the same material, the same liquid (water), and vary the shape of the bottle (use as many bottles of different shapes as you have). Compare the resulting graphs of step 6 among each other. Explain your observations. Does the frequency depend upon the shape of the bottle? Can you explain why it does or does not depend on the shape of the bottle? What happens to the note (i.e. loud, soft, high, low etc.)?
3. Repeat steps 1 to 6, but use different liquid. Use the same height, the same material, the same shape, and vary the liquid in the bottle (use as many liquids of different densities as you have). Compare the resulting graphs of step 6 among each other. Explain your observations. Does the frequency depend upon the liquid in the bottle? Can you explain why it does or does not depend on the liquid in the bottle? What happens to the note (i.e. loud, soft, high, low etc.)?
4. Describe what happens when air is blown across the top of the bottle. Connect this to how sound is produced and how that sound reaches our ear. In addition, explain how our ears work.
5. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research*

*method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).

- How does the strength of blowing across the top of the bottle relate to the volume of the sound?
- How does the strength of blowing across the top of the bottle relate to the frequency of the note?
- Suppose you take 2 bottles of the same material and shape, but different heights. Add water to the taller bottle so that the length of the air column is the same in both bottles. Will the frequencies be different?
- Choose eight bottles all the same. Make them ‘sing’ a scale (doh, ray, me, fah, soh, la, te, doh) by putting a different amount of water in each (Hint: Remember that you have marked the level of the water on a bottle for each note at step 6). For precision you can use a flute and the Interactive Physics Program, as you did at step 5. Save the graph/data each note of the flute produces and then try to match each note with your one bottle organ.
- Is it possible to get each note of the scale (doh, ray, me, fah, soh, la, te, doh), starting with a bottle already half filled?
- What is the minimum amount of liquid that can be included in the bottle and still to get each note of the scale (doh, ray, me, fah, soh, la, te, doh)? Try to use as many liquids of different density as possible.

6. How do you think you play higher frequencies on a pipe organ? Explain.

7. How do you think you play lower frequencies on a pipe organ? Explain.

## **Final Project/Report**

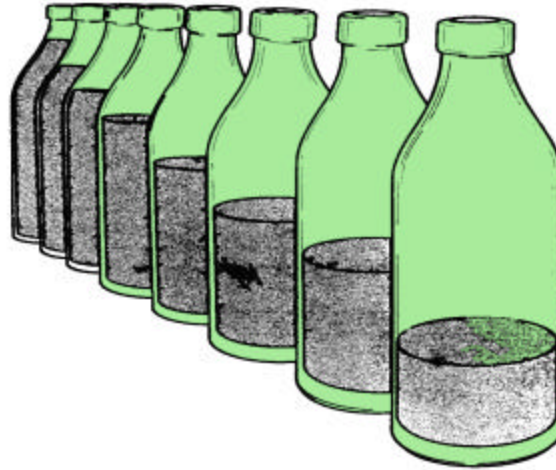
You have been asked to construct and write the manual of an eight-bottle organ (one bottle for each note – see figure 2) by a major music instrument manufacturer (use the eight-bottle organ you already have. It is up to you to decide which of the three variables – length, shape, substance of the container – will be varied in order to achieve the eight bottle organ). You have the freedom to make your own selections concerning the content of the manual. Remember to summarize all the experiments and the activities you have performed in this project. In addition, have in mind that the manual must be descriptive and understandable by people that have never used a bottle organ before. Therefore, sections about how to build the bottle organ, how to use the bottle organ, and how/why the bottle organ works are strongly suggested to be included within your manual. Use of diagrams or pictures is strongly suggested.



**YSAP**<sup>®</sup>  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

**Figure 2**



**(Smith and Holloway, 1985, p. 55)**

## **Reference**

Smith, G. and Holloway, G. (1985). *40 Science Activities*. London: Macmillan Education.

## **Unit 2: Acoustics – Sound Waves**

### **Project 9**

## **What is the Difference Between Noise and Music?**

Music instruments have long been used for entertaining people all over the world. Pianos, guitars, violins, flutes etc. produce musical sounds that we love to hear. But, what makes the sound musical? Do you want to know? Follow this experiment and by using sound and wave generators, produce musical sounds, compare them, and try to understand why they sound musical and not noisy.

### **Degree of Difficulty**

Experimental: Moderate to Difficult  
Conceptual: Difficult

### **Objectives**

Completion of the activities should enable you:

- to produce chords by using a combination of sound and wave generators.
- to understand the relationship between the combination of different frequencies and the chord produced.
- to predict the combination of frequencies needed to give a chord.

**Materials:** 4 sound and wave generators how expensive are they? (Cambridge Physics Outlet, 10 Green St. Bldg. E. Woburn, MA 01801, Toll Free 1-800-932-5227, E-mail: info@cpo.com) computer, sound sensor (universal lab interface is needed), “Interactive Physics” software or “Macmotion” software, 1 plastic tube, 1m long, 5cm diameter.

### **Procedure**

#### **Producing a Chord**

1. Turn the volume of the Sound and Wave Generator (SWG) all the way down. Tune the frequency of the first SWG to 300 Hz, the second SWG to 375 Hz, the third SWG to 450 Hz, and the fourth SWG to 600 Hz.
2. Place the four SWG a couple of centimeters apart, in any order you want, and turn up the volume on all of them. Describe what you hear.

The combination of the above mentioned frequencies gave a particular sound that many music instruments can give (i.e. piano). The sound that was produced is called a chord. For the purposes of these experiments the combination of a 300 Hz, a 375 Hz, a 450 Hz, and a 600 Hz frequencies will be called Chord A.

3. Repeat steps 1 and 2 for the chords B, C and D. However, use the frequencies given in the following table:

**Table 1**

Chords	Frequencies (Hz)			
<b>Chord B</b>	300	360	450	600
<b>Chord C</b>	300	375	500	600
<b>Chord D</b>	300	400	500	600

Do the chords sound the same?

4. For each one of the four chords, fill in a table like the one shown below. In the third column write down the ratio of the frequencies after reducing it to the lowest common denominators (Note that the frequencies are not the same for the four chords, thus make sure that the frequencies column is changed each time).

**Table 2**

Frequencies	Chord A Ratio	Reduced Ratio
375 Hz and 300 Hz		
450 Hz and 300 Hz		
600 Hz and 300 Hz		
450 Hz and 375 Hz		
600 Hz and 375 Hz		
600 Hz and 450 Hz		

5. After the completion of the four tables, compare the tables among each other and try to come up with the pattern that underlies the creation of chords.

**Note:** The structure and procedure of producing a chord by using the sound and waves set of Cambridge Physics Outlet (CPO) was based upon the description that was provided by the manufacturer's online manual (CPO, 2000).

## Activities

1. Use the tables you have created for step 4 and your findings of step 5 and develop a mathematical rule that can predict notes that can be played together and give a chord. Put your mathematical rule in test. First state your predictions (all the combinations that you think that can give a chord) and then test them. Make sure that you record all the successful and unsuccessful combinations in a table similar to the one presented in step 4.
2. Given that the frequency of notes on a scale are as follows,



**Table 3**

Notes	Frequency (Hz)
<b>doh</b>	264
<b>ray</b>	296
<b>me</b>	330
<b>fah</b>	352
<b>soh</b>	396
<b>la</b>	440
<b>te</b>	496
<b>doh</b>	528

Create a new table as you did at step 4 (add two more columns, one for the ratio and the other for the reduced fraction). To calculate the ratio, divide the frequency by the frequency for the lowest note (Doh at 264 Hz).

To western ears, pleasing combinations of notes are those with frequencies that in ratios of small numbers, such as 1:2 and 4:5. Which of the notes, of the table you created, have ratios that reduce to simple fractions? Which of these notes have ratios that don't reduce to simple fractions? Which combination(s), if any, of notes can give the four chords you created at the beginning of the experiment? Explain.

- Describe how sound is produced and how that sound reaches our ear. In addition, explain how our ears work.
- Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).
  - What are the limits of human hearing?
  - What is the speed of sound in air? (Hint: Use one of the SWG's and a one meter plastic tube, about 5cm in diameter. Put the tube close to the generator and try to find the *Fundamental Frequency* (the lowest resonance of a system - Resonance is a natural frequency of vibration of a system). Use the formula  $\text{Speed of sound} = \text{Wavelength} \times \text{Frequency}$ .) Is the speed of sound constant? After you find the speed of sound, find all the frequencies for which the tube *resonates* ('sings') and the corresponding wavelengths.



- Give combination(s) of frequencies that can give chords, which correspond to a combination of the notes introduced in activity 2. Is the number of chords that can be played limited? Explain
- What is the minimum frequency that we can use in order to get a scale that gives the four chords of our experiment and still most of the people are able to hear?

## **Final Project/Report**

Write a report summarizing all the experiments and the activities and include information on the applications of physics of sound (i.e. programming computers to write music). Have in mind that your report must be descriptive and understandable by people that have never conducted a similar experiment. Use of diagrams, tables or pictures is strongly suggested.

## **Reference**

Cambridge Physics Outlet Online Curriculum (2000). Experiments on waves and sound.  
URL: <http://www.cpo.com/>

## **Unit 2: Acoustics – Harmonics / Resonances**

### **Project 10**

### **Waves on a String?**

Music instruments have long been used for entertaining people all over the world. Pianos, guitars, violins etc. produce musical sounds that we love to hear. What do they have in common? Strings! How does a string produce sound? Do you want to know? Follow this experiment and by using a sound and waves set, find out for yourself the answers to these questions or the ones that were troubling you so far!

### **Degree of Difficulty**

Experimental: Moderate  
Conceptual: Moderate

### **Objectives**

Completion of the activities should enable you:

- to produce waves on a string by using a sound and waves set.
- to understand the relationship between frequency and wavelength.
- to measure the frequency and wavelength of several wave patterns.
- to make predictions about the frequency or the wavelength of wave patterns.

**Materials:** 1 sound and waves set (Includes generator, wiggler, fiddlehead, standing wave and resonance experiments, cords, and sturdy case. Requires CPO Timer and Physics Stand - Cambridge Physics Outlet, 10 Green St. Bldg. E. Woburn, MA 01801, Toll Free 1 800 932 5227, E-mail: info@cpo.com) computer, sound sensor (universal lab interface is needed), “Interactive Physics” software or “Macmotion” software.

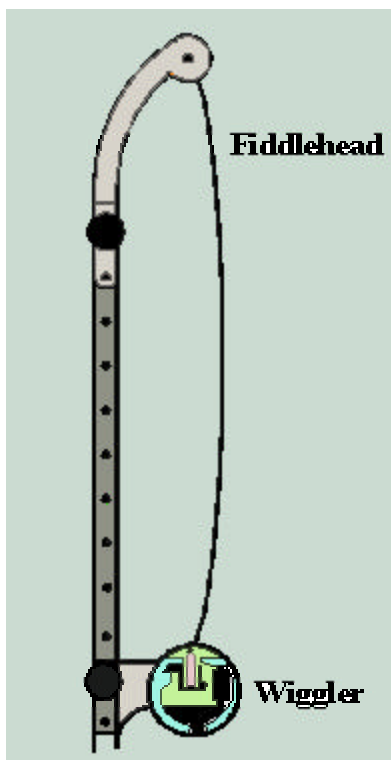
### **Procedure**

#### **Producing waves on a string**

1. Take a stand and place the fiddlehead of your sound and waves set at the top of the stand (see figure 1).
2. Take the wiggler and place it as low as possible on the stand (below the fiddlehead). Make sure that the arm of the wiggler (with the hole at the end) is pointed up (see figure 1).
3. Find the elastic string that is included in your set and tie it through the hole in the wiggler arm. Thread the other end of the string between the washers of the

fiddlehead. Stretch the string until it feels about as tight as a taut rubber band. Tighten the nuts to lock the string in place (see figure 1).

**Figure 1**



(CPO, 2000)

4. Connect the Timer and the Sound and Wave Generator to the wiggler (use the AC adaptor for the timer - not batteries).
5. Turn on the Timer (the switch is on the left side). Press the timer select until the "Frequency" light is lit. Ignore any sounds that the speaker might make or any numeric indications on the screen. Press the Sound and Wave Generator until the "Waves" light is lit. The speaker should be off, and the Sound & Wave Generator now controls the frequency of the wiggler. The numeric indication on the timer's screen now displays the frequency of the wiggler.
6. Reduce the frequency until the wiggler is clicking once every second. What does the timer read? What does this mean? Now put the timer in period mode. What does the timer read? What does this mean?
7. Press the MODE button on the Timer until it is back in frequency mode. Start increasing the frequency and record both the frequency and the corresponding period in the following table:

**Table 1**

Frequency (Hz)	Period (sec)	Frequency x Period

What do you observe from the findings of this table? Are the frequency and the period related? If yes, give the mathematical formula that relates them. What are the units of their product (column 3)?

- Press the MODE button on the Timer until it is back in frequency mode. Raise the frequency until you see the string vibrating in some new, clear pattern. Each time that you see a new pattern, it means that you have found a frequency that the string *resonates* at (resonance is a natural frequency of vibration of a system). Describe what do you observe as the frequency increases. Why does that particular phenomenon happen as the frequency increases? What does this phenomenon have to do with the sound? Explain.

**Note:** The structure and procedure of producing waves on a string by using the sound and waves set of Cambridge Physics Outlet (CPO) was based upon the description that was provided by the manufacturer's online manual (CPO, 2000).

## Activities

- Set the Timer to the frequency mode. Raise the frequency until you excite the *Fundamental Frequency* (the lowest resonance of a system) of the elastic string (probably around 10 to 20 Hertz). The string should vibrate back and forth. Where is the string moving the most? Where is the string moving the least? What is the Fundamental Frequency? What is the period of the vibration?
- Set the Timer to the frequency mode. Raise the frequency until you excite the fundamental frequency. After you reach the fundamental frequency start increasing the frequency (use increments of 5-10 Hz), and explore what happens (look for other resonances of the string, and try to learn rules to help you predict at what frequencies they will occur). Stop after you find two more resonances, beyond the fundamental frequency. Describe and explain your observations for these two new resonances. Is it possible, using the data you have collected so far for the first three wave patterns you observed, to develop a mathematical expression or rule that can help you predict the resonances that will follow? If no, how much more data do you need for a mathematical expression that is able to predict the resonances of a string? After your mathematical rule/expression is developed, make your predictions and test them.

Were your predictions valid? Explain. Continue finding resonances until you cannot see any more on your string. For each new string pattern you observe (the moment that the new pattern occurs) record your data in the following table:

**Table 2**

Description of Pattern	Frequency	Number of points with maximum amplitude	Number of points with minimum amplitude

Graph frequency versus number of points with maximum amplitude. Interpret the resulting graph. What is the relationship between the frequency and the number of points with maximum amplitude?

In this activity, the new wave patterns that you observed, after passing the value of the fundamental frequency, are called *harmonics* (resonances which are more complex, or higher frequency, than the fundamental).

- How many of the points with maximum amplitude fit within one *wavelength*? How many of the points with minimum amplitude fit within one *wavelength*? After you answer this question, fill in the following table based on the results you obtained in activity 2.

**Table 3**

Frequency	Number of points with maximum amplitude	Number of points with minimum amplitude	Wavelength	Frequency x Wavelength
<i>fundamental frequency</i>				
...				

Graph frequency versus wavelength. Interpret the resulting graph. How does the product, *Frequency x Wavelength*, relate to the graph? What is the relationship between the frequency and the wavelength? Give the mathematical expression that relates the two of them.

4. Use the same string you were using for the previous activities, and vary the length (make sure that the tightness of the string stays relatively constant as before – the suggested way to do this is to keep the ratio between the length and the relative diameter of the string constant). Start with the Fiddlehead and the Wiggler being 20 cm apart and use increments of 5cm. Record your data in the following table:

**Table 4**

Length of the string	Fundamental frequency	1 <sup>st</sup> Harmonic	2 <sup>nd</sup> Harmonic	3 <sup>rd</sup> Harmonic	...
20 cm					
25 cm					
30 cm					
35 cm					
40 cm					
45 cm					
...					

Graph fundamental frequency versus length (use Microsoft Excel). What does the graph look like? How does the length of the string relate to the frequency? What do you think happens to the sound that the fundamental frequency is producing as the length is increasing (i.e. loud, soft, high, low etc.)? Explain. What will the graphs of 1<sup>st</sup>, 2<sup>nd</sup>, 3<sup>rd</sup>, etc. harmonic versus length look like? Explain (make your predictions before plotting the graphs). Compare all the resulting graphs among each other and explain your observations.

5. Repeat the steps of activity 4, but use strings of different diameters this time. Use the same length, the same tightness, and vary the diameter of the string (use as many strings of different diameter as you have). Does the frequency depend upon the diameter of the string?
6. Describe what happens when a string is plucked. Connect this to how sound is produced and how that sound reaches your ear. In addition, explain how our ears work.
7. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence

might affect your experiment and therefore, you keep them constant throughout the whole experiment).

- Does the same mathematical expression for predicting the resonances apply to all strings?
- Is there a minimum length of string we need in order to get the first resonance?
- Is there a maximum number of resonances that a given length of string can give?
- How can we increase the amplitude of the maximum points of a given harmonic?
- Does the fundamental frequency of different lengths of string have the same amplitude?
- How does the amplitude of the maximum points change as the frequency increases?
- Do the minimum points of the previous harmonic change position when the new harmonic appears?
- Does the wavelength change as the frequency increases?

8. How do you think you play higher frequencies on a guitar? Explain.

9. How do you think you tune a string on a guitar? Explain.

10. Explain the importance of the guitar box?

## **Final Project/Report**

Write a report summarizing all the experiments and the activities and include information on the applications of physics of sound (i.e. how guitars are made). Have in mind that your report must be descriptive and understandable by people that have never conducted a similar experiment. Use of diagrams, tables or pictures is strongly suggested.

## **Reference**

Cambridge Physics Outlet Online Curriculum (2000). Experiments on waves and sound.  
URL: <http://www.cpo.com/>

## **Unit 2: Acoustics & Optics – Lasers and Loudspeakers**

### **Project 11**

### **Make Your Own Laser Show**

People in the music industry in their effort to make music performances more exciting and, visually, more attractive understood that light had to be synchronized with music. In a way, they were looking for a method that will “make light dance” with their music. The most spectacular of all methods has proven to be the laser shows. But, what does make the light dance? Do you want to know? Follow this experiment and by creating your own “laser show device”, find out for yourself the answers to these questions.

### **Degree of Difficulty**

Experimental: Difficult

Conceptual: Difficult

### **Objectives**

Completion of the activities should enable you:

- to produce Lissajous (the series of plane curves traced by an object executing two mutually perpendicular harmonic oscillations) figures with a laser beam.
- to understand the relationship between the combination of different frequencies and the Lissajous figures produced.
- to predict the combination of frequencies needed to give a Lissajous figure.

**Materials:** laser pen (class 2 with power output 1mw), two 3-10 Volt a.c. power supply, 1 signal generator, 1 frequency multiplier/converter, 1 phase converter, a piece of wood (40x40x2 cm), 2 thin aluminum levers (19.5 cm long each), 2 loudspeakers (13cm), 1 mirror (4x4 cm), 1 rubber rod (4x1x1 cm), a piece of wood (4x4x2 cm), 2 pins, table-tennis ball, araldite, silicon rubber, thin wire (about 2 mm in diameter and about 10cm long), 2 pieces of a thin aluminum lever (3 cm long), two wooden dowels 5.5x2x2 cm (make sure the width of these wooden pieces is slightly bigger than the aluminum lever’s width), drill, hammer, nails (2.5 cm long), ruler.

### **Procedure**

#### **Constructing the laser show device**

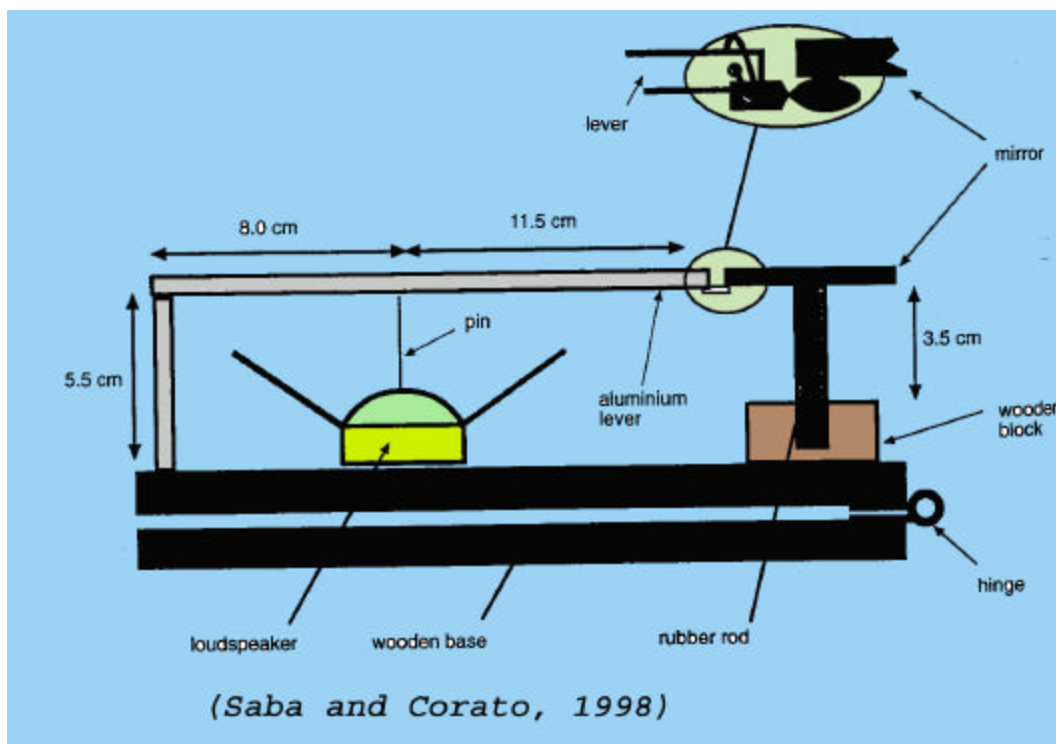
1. Put the 40x40x2 cm piece of wood on a table. Find the mid points of two of the longest and opposite sides. Take your ruler and with a pencil draw a line connecting the two points. Repeat the same steps for the other two opposite sides. In this way you



are separating the wooden board into four equal pieces. Name with a different number each quartile (1,2,3,4).

2. Take the two dowels and cement with araldite the first one in the center of the second quartile and the second one in the center of the fourth quartile (cement one of dowel's smallest surfaces).
3. Cut the table tennis ball into two equal pieces. Cement each piece of the ball (convex side out) with araldite in the center of the loudspeakers. Cement a pin at the top of the ball with araldite.
4. Place the 2 loudspeakers in front of the already fixed dowels (step 2). Make sure that the pin lines up with the center of the ball that is positioned upon the loudspeaker, and that it is 8 cm away from the wooden dowel and 10 cm away from the closest side. At this point check whether the top of the pin and the wooden dowel are at the same height. If not, make the necessary adjustments so both are at the same height.
5. Take a piece of wood (4x4x2 cm) and drill a hole (1x1x0.5 cm) in its center. Insert one of the rubber rod's ends into the wooden block (if the rod is loose use araldite). Cement the center of the back of a mirror to the top of the rubber rod.
6. Drill a hole at one of the ends of each of the 2 thin aluminum levers (about 0.5 cm from the lever's end – the diameter of the hole must be much smaller than the width of the lever, but large enough for the wire to go through).
7. Fasten the end (the one without the hole) of each aluminum rod, with silicon rubber (do not use araldite), at the top of the wooden dowels, that were fixed on the wooden board at step 2, and let the levers lie on the top of the pins (see figure 1).
8. Cement the wooden block of step 5 (in the third quartile) in such a way that the mirror at the top of the rubber rod is 1 cm away from the two aluminum levers and at the same height with them (see figure 1).
9. Twist the 3 cm long aluminum levers 180°. Drill a hole at one of their ends (about 0.5 cm from the lever's end – the diameter of the hole must be much smaller than the width of the lever). Glue to the back of a mirror the end that does not have the hole and connect to each lever by means of a wire that goes through their tiny holes and the holes of the short twisted aluminum pieces (see figure 1).
10. Connect one of the loudspeakers with the 3 V supply and the other one with a signal generator (or functional generator). Make sure that the output impedance of the power supply and the signal generator matches the input impedance of the loudspeakers (in case you have any questions ask your teacher or read the manual of both the loudspeaker and the generator).

**Figure 1**



11. Set your device at a certain angle and shine a laser beam on the mirror. Make sure that you see the reflected beam on a big white screen or on a white painted wall. If the reflected beam scatters, use a long focusing lens to focus the beam on the screen.
12. Start changing the frequency of the signal generator (start from zero) by small increments. Record your data in the following table:

**Table 1**

Shape description (Lissajous figures)	Frequency of the generator

What do you observe? Is there a relationship between the shapes and the frequency? Are the shapes stationary or moving? Why?

13. Start changing the amplitude of the signal generator (start from zero) by small increments (the frequency must be kept constant). Record your data in the following table:

**Table 2**

Shape description	Amplitude of the generator

What do you observe? Explain. Is there a relationship between the shapes and the amplitude?

## Activities

1. Repeat steps 1-10. However, this time connect both loudspeakers to the same power supply (can you think of a reason for doing this?). Connect a frequency multiplier in series with the power supply for only one of the loudspeakers. Start changing the frequency (start from zero) by small increments. Record your data in the following table:

**Table 3**

Shape description	Frequency of the generator

What do you observe? Is there a relationship between the shapes and the frequency? Are the shapes stationary or moving? Why? How is your current setting different from the one you used at step 12? Explain.

2. Repeat steps 1-10. However, this time connect both loudspeakers to the same power supply. Connect a frequency multiplier and a phase converter (if it is difficult to find a phase converter, ask your teacher to make one for you; it is basically a simple RC circuit that delays the phase of the signal) in series with the power supply for only one of the loudspeakers. Start changing the phase by small increments (the frequency

must be kept constant). What do you observe? Is it possible to predict the relationship between the shapes and the phase by just looking at the Lissajous figures? Are the shapes stationary or moving? Why?

3. Explain the reasons of using loudspeakers in your device. Can you think of something else that can replace the loudspeakers and still give Lissajous figures as outcomes?
4. Design and perform experiments to answer the following questions. Your experiment's lab report must include: Title, Purpose, Equipment, Procedure and Data Collection, Data Analysis, Results and Conclusion. The Data Collection is the most critical part of a laboratory experiment. You must follow a *systematic research method* to secure valid answers to your questions. In a systematic research method you define, (a) your independent variable (the variable that is controlled by the researcher and is being changed, increased or decreased, in order to measure any changes regarding the variable we are interested in – It is suggested to change the independent variable by equal increments), (b) your dependent variable (the variable that is under investigation and depends upon the changes of the independent variable), (c) control variables (other variables that you assume that their presence might affect your experiment and therefore, you keep them constant throughout the whole experiment).
  - Can you repeat the results of activity 1 by using a signal generator? Explain.
  - Can you repeat the results of activity 1 by using two signal generators (one for each loudspeaker)? Explain.
  - How can you calibrate a phase converter that is constructed with cooperation with your teacher, by using Lissajous figures?
  - What combinations do you have to make in order to get a rolling ball (hint: use two signal generators)?
  - What combinations do you have to make in order to get a moving figure?
  - What combinations do you have to make in order to get a stationary figure?
  - Which variable(s) relate to the size of the figures?
  - How does the frequency relate with perfect geometrical figures?
  - How does the phase relate with perfect geometrical figures?

## Final Project/Report

Write a report summarizing all the experiments and the activities and include information on the applications of your device in everyday life activities (i.e. musical laser shows). Have in mind that your report must be descriptive and understandable by



**YSAP**<sup>®</sup>  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

people that have never conducted a similar experiment. Use of diagrams, tables or pictures is strongly suggested.

## **References**

Saba, M. and Corato, L. (1998). A giant laser show. *School Science Review*, 80 (291), p. 108-109.

Piper, G. and Lawton, P. (1988). The Laser show. *School Science Review*, 70 (250), p. 87.



*Computer Programming And Inquiry-Based Experimentation In Science Education.*  
*Zacharias C. Zacharia*

## **Unit 3: Computer Programming In Science Education**

## **Unit 3: Computer Programming in Science Education – Spreadsheets**

### **Project 12**

# **Spreadsheets as a Scientific Tool: Using Spreadsheets to Analyze Multiloop Circuits**

Computers have become an essential tool for the acquisition, analysis, presentation, and communication of data in ways which allow you to become a more active participant in research and learning.

A familiarity with even fundamental computer software can help you understand concepts and processes of science. In other words, computer software already existing in your computer (e.g., spreadsheets) can be used as a tool for understanding scientific concepts.

This project aims at familiarizing you with using spreadsheets as a tool for analyzing, calculating and plotting data in science applications/projects. For demonstrating these powerful uses of spreadsheets, electric circuits were chosen. The project involves calculating the current, the resistance, and the voltage of multiloop resistive circuits that usually involve tedious calculations, by using computer-based spreadsheets.

## **Degree of Difficulty**

Experimental: Easy

Conceptual: Moderate

## **Prerequisite Knowledge**

This project is suggested for students that are already familiar with (a) simple (i.e., single loop resistive) and multiloop electric circuits, (b) the laws of conservation of charge and conservation of energy (fundamental laws that apply to all electric circuits), and (c) the concepts of current, resistance, and voltage. In addition, it is required that the students have basic computer skills (e.g., turn on the computer, use the keyboard, use the mouse), have some experience in basic computer functions (e.g., access programs, open programs, save data), and have knowledge of basic computer programs (e.g., word processing programs), especially spreadsheets that are essential for this project. In case the student does not know anything about spreadsheets or needs to refresh her/his memory about what spreadsheets are, and how they work, the following website (<http://www.tutorialfind.com/tutorials/computerbasics/spreadsheets/>) is suggested.

## **Useful Concepts**

- Electric current is the rate at which electric charge flows and is defined by

$$I \equiv \frac{\Delta q}{\Delta t},$$

- where  $\Delta q$  is the amount of positive charge that passes in time  $\Delta t$  through a hypothetical surface that cuts across the conductor.
- A resistive material is said to obey Ohm's law if current is proportional to the voltage across that material:

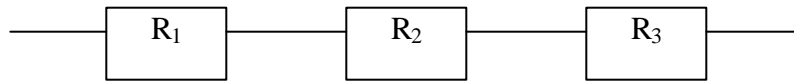
$$V = IR$$

where  $V$  is the voltage,  $I$  is the current, and  $R$  is the resistance.

- The power dissipated in an electric circuit is (assuming the circuit obeys Ohm's law)

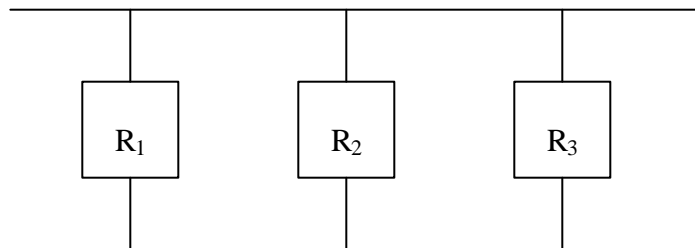
$$P = IV = \frac{V^2}{R} = I^2 R$$

- Kirchhoff's voltage rule states that the algebraic sum of the potential differences (voltages) around a closed conducting loop must be zero.
- Kirchhoff's current rule states that the net current entering (or leaving) any junction must be zero.
- The equivalent resistance  $R_{s \text{ total}}$  of a number of resistors connected in series is



$$R_{s \text{ total}} = R_1 + R_2 + R_3 + \dots$$

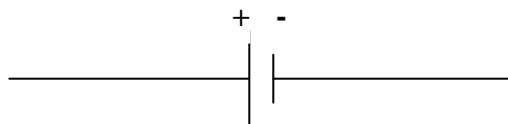
- The equivalent resistance  $R_{p \text{ total}}$  of a number of resistors connected in parallel is



$$\frac{1}{R_p} = \frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \dots$$



- An ideal battery is one that lacks any internal resistance to the internal movement of charge from terminal to terminal. A battery provides a difference in electric potential, or voltage. An "ideal" battery always provides the same voltage, regardless of the current, and a less ideal battery may be treated as an ideal battery in series with an internal resistor. For the purposes of this project we will be considering all batteries to be ideal and we will symbolize them as follows:



The “+” sign is for the positive terminal and the “-” sign is for the negative terminal. For historical reasons, the current is considered positive when it flows from the positive terminal, around the circuit, back to the negative terminal. This is opposite to the flow of electrons, which go from the negative terminal, around the circuit, back to the positive terminal. We will use the historical definition of positive current (For more information use the following reference: Halliday, D., Resnick, R. and Walker, J. (1997). Fundamentals of Physics. NY: John Wiley & Sons, INC.)

## Objectives

Completion of the activities should enable you:

- to learn to use a spreadsheet program as a laboratory tool to analyze data
- to use spreadsheets as a tool to make theoretical calculations
- to use spreadsheets as a tool to plot results

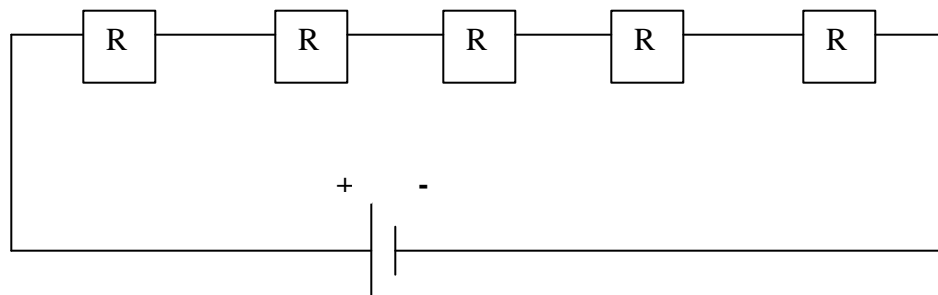
**Materials:** computer, and EXCEL spreadsheets (or any other spreadsheet program - all other common spreadsheet programs do what is needed for this experiment)

## Procedure

### Part A: Introduction to using spreadsheets as a laboratory tool

For getting familiar with how spreadsheets can be used as a laboratory tool to calculate or analyze data, let's consider an example where the current passing through the battery of a single loop circuit, consisting of an ideal battery with voltage  $V = 12$  volts and 5 resistors with resistance  $10\ \Omega$  connected in series (see figure 1), must be calculated. For calculating the current through the battery by using a spreadsheet, the following steps must be followed:

**Figure 1**



1. Enter the voltage and resistor values in a spreadsheet as follows

	<b>A</b>	<b>B</b>	<b>C</b>	<b>D</b>
		Resistance	voltage	Current
<b>1</b>		(ohms)	(volts)	(Amperes)
<b>2</b>	R1	10	12	
<b>3</b>	R2	10		
<b>4</b>	R3	10		
<b>5</b>	R4	10		
<b>6</b>	R5	10		
<b>7</b>	Rtotal			

2. Click in cell B7 and insert the expression “=SUM(B2:B6)”. This expression will give you the sum of the values that are inserted in the cells B2 through B6. Instead of writing the expression “=SUM(B2:B6)”, you can select cell B7 and click on the “**S**” button on the toolbar. Then click on B6, drag the mouse upwards to cell B2, and click on the check next to the text box. What do you observe? How does this result compare with what you got when you used the expression “=SUM(B2:B6)”.
3. Now that you know both the voltage and the total resistance, from Ohm’s law you can calculate the current. Click in cell D2 and insert the expression “=C2/B7”. This expression will give you the ratio between the value of the cell C2, which in this case is the voltage, and the value of cell B7, which is the total resistance that we calculated in step 2. The result that will appear in cell D2 (in this case it will be equal to 0.24) will be the current through the battery.

You must be wondering what was the purpose of spending so much time setting up a spreadsheet for such an easy calculation. The answer to this question is not obvious yet, but will become clear after going through step 4.

4. Assume that one of your resistors, let’s say  $R_1$ , must be replaced with another resistor with resistance  $25\ \Omega$ . Replace the number 10 in cell B2 with the number 25 and press

“Enter”. What do you observe? Is the current value the same as before? How about if you change the voltage from 12 volts to 15 volts? When does the current value change? (only after hitting enter, or clicking on another cell)

5. Create the following table on your spreadsheet:

$R_1$ (Ohms)	$I$ (Amps)
0	
5	
...	
50	

Use the circuit of figure 1. Change  $R_1$  in increments of  $5\ \Omega$ , starting from zero, and keep the resistance values of the other 4 resistors constant ( $=10\ \Omega$ ). Use 10 of your resulting values to plot  $I$  versus  $R_1$ . For making the plot, highlight the numerical values of both variables and then click on the “chart wizard” button on the toolbar (or go under “Insert” and select “chart”). After you click on the “chart wizard” you will see the cursor of your mouse changing into a cross. Click anywhere on your spreadsheet, hold your mouse’s left button, drag it diagonally for 4-5 cm (depending on how big you want your graph to be), and then release the button. At this point you will see a window and the chart wizard dialogue box on the screen. Let the dialogue box guide you through until you get your graph. What does your resulting graph look like? Does the resulting graph intersect the x- and y-axes? Explain. What is the meaning of the slope of the curve?

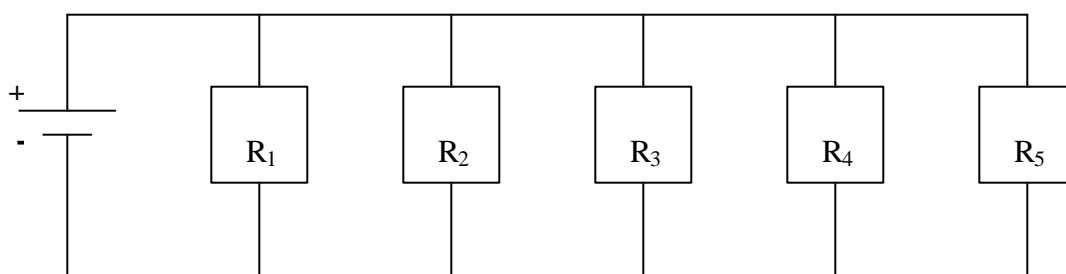
As you have already observed, the spreadsheet we set up for the electric circuit of our example can calculate the current through the battery for as many combinations of different values of resistances and voltages as we want. This is particularly important, not only because we do not have to repeat the same calculations over and over again, but also because we can use this technique for more complicated circuits that involve many resistors and more than one voltage source.

## Activities

1. Is the spreadsheet that you have set up by following steps 1 to 3 capable of calculating the voltage if, instead, the current through the battery is given? Explain.
2. Set up a spreadsheet, where the current through the battery is 1 A and the resistors have the same resistance as in the example given at the beginning, for calculating the voltage.
3. You are given the voltage as 24 volts and the current through the battery as 0.2 A.
  - What must be the resistance of  $R_1$ , if the other resistors are kept the same as in the example? Set up a spreadsheet to make the calculation.
  - What must be the resistance of  $R_1$  and  $R_2$  (assuming that both have the same resistance), if the other three resistors are kept the same as in the example? Set up

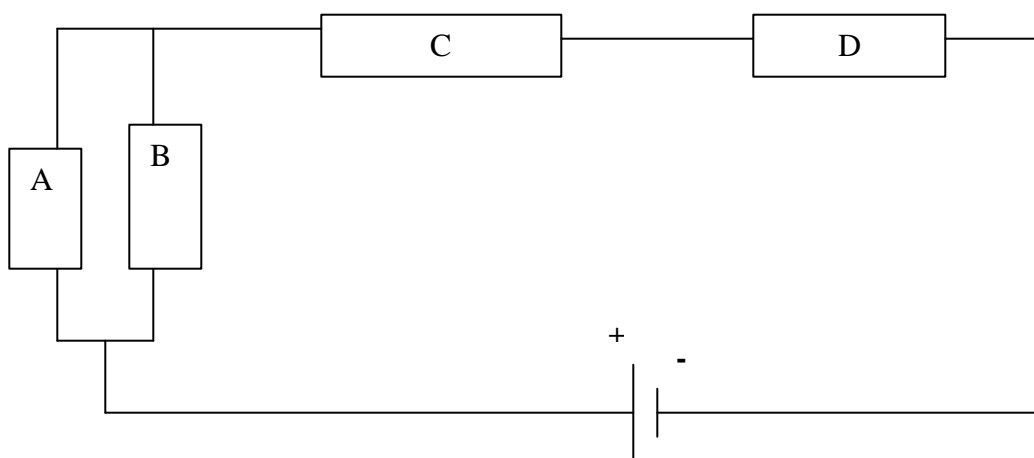
- a spreadsheet to make the calculation.
- What must be the resistance of  $R_1$  and  $R_2$  (assuming that  $R_2$  has half the resistance of  $R_1$ ), if the other three resistors are kept the same as in the example? Set up a spreadsheet to make the calculation.
  - What is the resistance of  $R_1$ ,  $R_2$ ,  $R_3$ ,  $R_4$ , and  $R_5$ , if  $R_2=R_1+2$ ,  $R_3=R_2+2$ ,  $R_4=R_3+2$ ,  $R_5=R_4+2$ ? Set up a spreadsheet to make the calculation.
4. Suppose, now, that the resistors of the example are connected in parallel, instead in series. Follow, again, steps 1 to 5, and do activities 1 to 3. How do the corresponding circuits compare to each other?

**Figure 2**



5. You are given the circuit of figure 3. Set up a spreadsheet to calculate the current through the battery.

**Figure 3**



**Note:**

- Box A contains 10 resistors connected in series. The first five are each  $10\ \Omega$  and the other five are each  $20\ \Omega$ .
- Box B contains 8 resistors. The first four are each  $10\ \Omega$  and are connected in series. The other four are each  $15\ \Omega$  and are connected in parallel with the first four and each other.

- Box C contains 10 resistors connected in parallel. The first five are each  $10\ \Omega$  and the other five are each  $20\ \Omega$ .
- Box D contains 12 resistors. The resistors are connected in parallel in pairs of 2 and then the six pairs are connected in series. The resistance of each resistor is  $10\ \Omega$ .

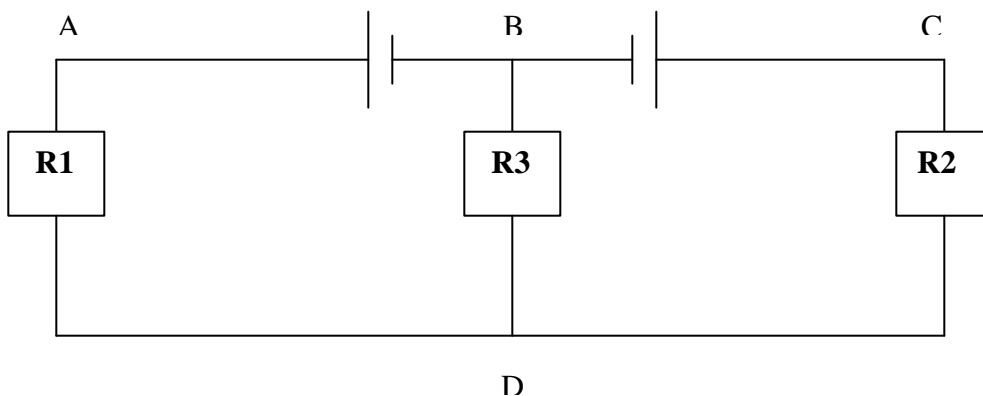
**Hint:** Make a sketch of the circuit diagram and show it to your teacher or mentor. It is important to have a clear diagram! Take each box at a time and calculate its total resistance (follow the same procedures and equations as in previous steps). Then find the total resistance of boxes A and B (note that they are connected in parallel) and as a final step find the total resistance of the circuit by adding the resulting resistance from boxes A and B, with the resistances of boxes C and D (note that A+B, C, and D are connected in series).

## Procedure

### Part B: Using spreadsheets to analyze more complex multiloop circuits

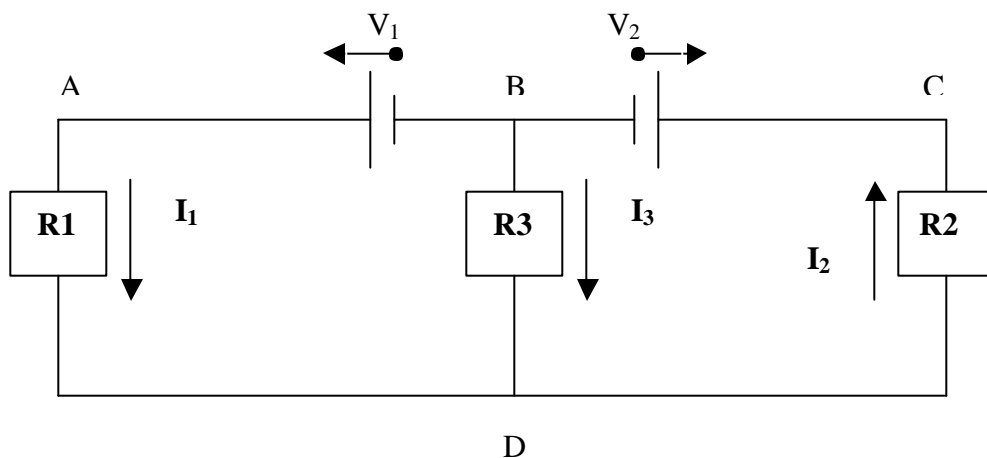
In this part we will be using more complex multiloop circuits, which contain more than one voltage source (see figure 4). For simplicity, we again assume the batteries are ideal. In the circuit of figure 2 there are two junctions (D and B) and three branches connecting the junctions (BAD, BCD, BD). As before, our goal is to set up a spreadsheet that gives us the currents in the three branches.

**Figure 4**

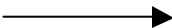
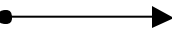


1. Arbitrarily label the currents, using a different symbol for each branch (see figure 5). Current  $i_1$  has the same value everywhere in the branch BAD;  $i_2$  has the same value everywhere in the branch BCD;  $i_3$  has the same value everywhere in the branch BD. The directions of the currents are chosen arbitrarily.

**Figure 5**



**Note:**

-  shows current direction  
 shows current direction through battery (historical definition)

The use of arrows is important because it defines whether the voltage across the battery or the resistor will be considered positive or negative. For the batteries it is considered positive when the current flows from the positive terminal, around the circuit, back to the negative terminal of the battery (historical definition – Voltage Rule: for a current through an ideal battery in the direction of the battery voltage arrow, the change in potential is  $+V$ ; in the opposite direction it is  $-V$ ). For a current through a resistance in the direction of the current, the change in potential is  $-iR$ ; in the opposite direction it is  $+iR$  (Resistance Rule). To understand this better, consider equation (2) in step 4 (where the BADB loop is being traversed in a counterclockwise direction from point B). Kirchhoff's voltage rule gives

$$V_1 - i_1 R_1 + i_3 R_3 = 0$$

- The first term ( $V_1$ ) is positive because the current moves through an ideal battery in the direction of the battery voltage arrow.
- The second term ( $i_1 R_1$ ) is negative because the current moves through a resistance in the direction of the current.
- The third term ( $i_3 R_3$ ) is positive because the current moves through a resistance in the opposite direction of the current.

**Note:** Kirchhoff's voltage rule is important to be understood. In case you have more questions do not hesitate to ask your teacher or mentor.

2. Consider junction D. Charge comes into the junction via incoming currents  $i_1$  and  $i_3$ , and it leaves via outgoing current  $i_2$ ; there is no increase or decrease of charge at the junction. This means that

$$i_2 = i_1 + i_3 \text{ (Kirchhoff's Current Law)} \quad (1)$$

- This equation involves 3 unknowns. To solve the problem you need two more equations involving  $i_2$ ,  $i_1$ , and  $i_3$ . You obtain them by applying the loop rule. You have three loops from which to choose: the BADB loop, the BCDB loop, and the big loop BADCB (see figure 5).
- Choose two of the loops (e.g., the BADB loop and the BCDB loop). If you traverse the BADB loop in a counterclockwise direction from point B, the loop rule or Kirchhoff's voltage rule gives

$$V_1 - i_1 R_1 + i_3 R_3 = 0 \quad (2)$$

If you traverse the BCDB loop in a counterclockwise direction from point B, the loop rule gives us

$$- i_3 R_3 - i_2 R_2 - V_2 = 0 \quad (3)$$

- Solve the equations (1), (2), and (3) to get  $i_2$ ,  $i_1$ , and  $i_3$ . To give you an example, take equation 1 and 3. Substitute  $i_2$  in eq. 3 with  $i_2 = i_1 + i_3$ . What you get is

$$- i_3 R_3 - i_1 R_2 - i_3 R_2 - V_2 = 0 \quad (4)$$

Take, now, equation 1 and rewrite it in the form

$$i_3 = \frac{i_1 R_1 - V_1}{R_3} \quad (5).$$

Substitute  $i_3$  in eq. 4 by using eq. 5, and you get an equation for  $i_1$

$$i_1 = \frac{V_1 R_3 - V_2 R_3 + V_1 R_2}{R_1 R_3 + R_2 R_3 + R_1 R_2} \quad (6)$$

After finding  $i_1$  from eq. 6 you can calculate  $i_3$  from eq. 5 and  $i_2$  from eq. 3

$$i_2 = \frac{-i_3 R_3 - V_2}{R_2} \quad (7)$$

- Assume that  $V_1 = V_2 = 12$  volts, and  $R_1 = R_3 = 10\Omega$ ,  $R_2 = 20\Omega$ . Enter the voltage and resistance values in a spreadsheet as follows

	A	B	C	D	E	F
		Resistance (ohms)		voltage (volts)		Current (Amperes)
1						
2	R1	10	V1	12	i1	
3	R2	20	V2	12	i2	
4	R3	10			i3	

6. Click in cell F2 and insert the expression

$$=((D2*B4)-(D3*B4)+(B3*D2))/((B2*B4)+(B3*B4)+(B3*B2))$$

This expression will give you  $i_1$ . What is the value of  $i_1$ ? Compare this expression with equation 6 and write down the corresponding symbols.

7. Now that you know  $i_1$ , you can calculate  $i_3$ . Click in cell F4 and insert the expression  $=((F2*B2)-(D2))/(B4)$ . What is the value of  $i_3$ ? Compare this expression with equation 5 and write down the corresponding symbols.
8. To calculate  $i_2$ , click in cell F3 and insert the expression  $=(F4*B4)-(D3))/(B3)$ . What is the value of  $i_2$ ? Compare this expression with equation 3 and write down the corresponding symbols.
9. Do the resulting current values make sense? What do the negative signs mean? Is Kirchhoff's law valid in this case? Explain.
10. Assume that one of your resistors, let's say  $R_1$ , must be replaced with another resistor with resistance  $25 \Omega$ . Replace the number 10 in cell B2 with the number 25 and press "Enter". What do you observe? Are the current values the same as before? How about if you change the value of  $V_1$  from 12 volts to 15 volts? Is it possible to change any of the resistors or the voltages, without affecting one of the currents?

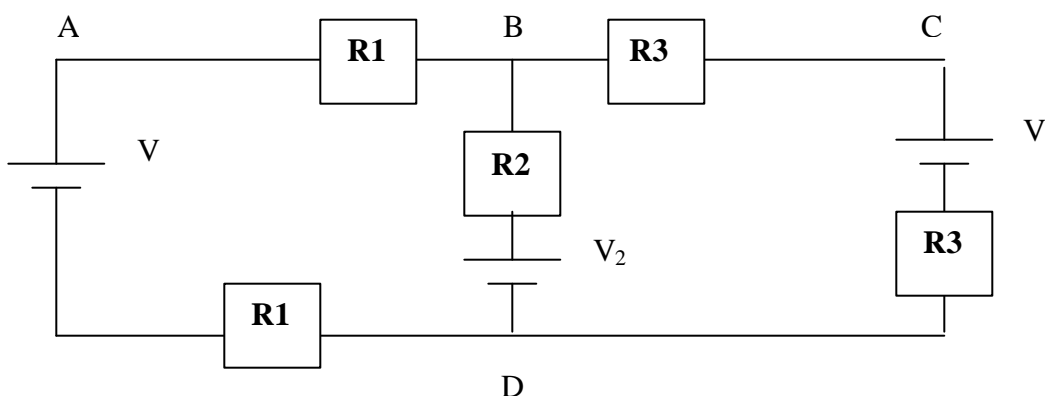
As you have already observed, the spreadsheet we set up for the multiloop electric circuit of our example can calculate the current through each branch for as many combinations of different values of resistances and voltages as we want. This is particularly important, not only because we do not have to repeat the same calculations over and over again, but also because we can use this technique for more complicated multiloop circuits.

## Activities

1. Is the spreadsheet that you have set up by following steps 1 to 7 capable of calculating the voltages if, instead, the current of each branch is given? Explain.



2. Set up a spreadsheet, where  $i_2 = 2A$  and  $i_3 = i_1 = 1A$ , and the resistors have the same resistance as in the example given in this part, to calculate the two voltages ( $V_1$  and  $V_2$ ). Use the same circuit as in figure 3.
3. You are given the voltages to be 15 volts and  $i_2 = 2A$  and  $i_3 = i_1 = 1A$ .
  - What must be the resistance of  $R_1$ , if the other resistors are kept the same as in the example? Set up a spreadsheet to make the calculation.
  - What must be the resistance of  $R_1$  and  $R_2$  (assuming that both have the same resistance), if the other resistor is kept the same as in the example? Set up a spreadsheet to make the calculation.
  - What must be the resistance of  $R_1$  and  $R_2$  (assuming that  $R_2$  has half the resistance of  $R_1$ ), if the other resistor is kept the same as in the example? Set up a spreadsheet to make the calculation.
  - What is the resistance of  $R_1$ ,  $R_2$ ,  $R_3$ , if  $R_2=R_1+2$ ,  $R_3=R_2+2$ ? Set up a spreadsheet to make the calculation.
4. Suppose, now, that each of the resistors in the example is connected in parallel with another resistor ( $R = 15 \Omega$ ). Follow, again, steps 1 to 7, and do activities 1 to 3. How do the corresponding circuits compare to each other?
5. Suppose that you set the directions of the currents to be different this time (e.g., in the opposite direction they are in figure 3). Repeat steps 1-9. How do your results compare? Explain.
6. Set up a spreadsheet, as shown above, for calculating or analyzing data regarding the following circuit:



Give values to all variables and calculate the currents passing through each battery.



**YSAP**<sup>®</sup>  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

## **Final Project/Report**

Choose another science topic for which you can use spreadsheets to calculate, analyze and plot data. Write a report following similar structure as the one used for this project. Your report must be descriptive and understandable by people that have never used spreadsheets before. Therefore, sections about (a) how to use spreadsheets for better understanding the science topic you chose, (b) how to set up spreadsheets for analyzing you project's data, and (c) why spreadsheets are important for the particular functions you set them to perform in your project, are strongly suggested to be included within your report. Use of diagrams or pictures is strongly suggested. At the end of your report, give more examples of possible applications of the spreadsheets as a science tool.

## **Reference**

Halliday,D., Resnick, R. and Walker, J. (1997). *Fundamentals of Physics*. NY: John Willey & Sons, INC.

## **Unit 3: Computer Programming in Science Education – HTML**

### **Project 13**

## **How to Create Your Own WebPages: An Introduction to HTML**

The internet has become an essential classroom tool for the acquisition, analysis, presentation, and communication of data in ways which allow you to become a more active participant in research and learning. A familiarity with how to use the internet can help you research and study topics across all areas of education. But how do people make all these data and information available on the World Wide Web? How do they create their WebPages? Follow us through the process of creating your own WebPages and find out for yourself the answers to these questions or others that have been troubling you!

### **Degree of Difficulty**

Experimental: Moderate

Conceptual: Moderate

### **Prerequisite Knowledge**

This project is suggested for students that are already familiar with basic computer skills (e.g., turn on the computer, use the keyboard, use the mouse), have some experience in basic computer functions (e.g., access programs, open programs, save data), and have knowledge of basic computer programs (e.g., word processing programs).

### **Useful Concepts**

- WWW: World Wide Web.
- SGML: Standard Generalized Markup Language--a standard for describing markup languages.
- DTD: Document Type Definition--this is the formal specification of a markup language, written using SGML.
- HTML: Hypertext Markup Language. HTML is an SGML DTD. In practical terms, HTML is a collection of platform-independent styles (indicated by markup tags) that define the various components of a World Wide Web document. HTML was invented by Tim Berners-Lee while at CERN, the European Laboratory for Particle Physics in Geneva (NCSA, 2000).

- **What an HTML Document Is:** HTML documents are plain-text (also known as ASCII) files that can be created using any text editor (e.g., Emacs or vi on UNIX machines; SimpleText on a Macintosh; Notepad on a Windows machine).
- **Tags:** commands included within the brackets < ... >
- **SimpleText or Notepad:** These are text editors that are available on every computer (SimpleText on a Macintosh; Notepad on a Windows machine). Notepad can be found under “Programs” and then under “Accessories” and SimpleText can be find under “Apple” and then under “Recent Applications”. In case you have problems finding them, use the “Find” option on your computer to find it for you.
- **Getting Your Files on a Server:** If you have access to a Web server at school, contact the individual who maintains the server to see how you can get your files on the Web. For the purposes of this project we will focus on how to create WebPages and how to run them from your hard disk. In case you want to make them public ask your teacher or mentor for help.

## Objectives

Completion of the activities should enable you:

- to learn how to use HTML for creating WebPages
- to use HTML as a tool to create complex tables and to include images in your WebPages
- to create links to other websites

**Materials:** computer, scanner (optional – in case you want to include some personal pictures in your WebPages) and HTML Document software (SimpleText on a Macintosh; Notepad on a Windows machine)

**NOTE:** When following the directions given below do not include in your HTML text the quotation marks used (e.g., if the text is written “<H1>Science is Fun</H1>” – you must use <H1>Science is Fun</H1> - The use of quotation marks is for separating the tags/commands from the rest of the text). However, there are cases where you have to use the quotation marks, but for these case the quotation marks must be included within the brackets < ... > (e.g., <A HREF=" Science Project Page1.htm ">Back to Page 1</A>).

## Procedure

Every HTML document should contain certain standard HTML tags (e.g. <title>). Each document consists of head and body text. The head contains the title, and the body contains the actual text that is made up of paragraphs, lists, and other elements.

For creating your first simple HTML document and web page follow the next steps:

1. Open Notepad/SimpleText and type in the following required elements (the required elements are the <html>, <head>, <title>, and <body> tags):

```
<html>
<head>
<TITLE> My Science Project's Homepage</TITLE>
</head>
<body>
<H1> Welcome To My Science Projects Homepage </H1>
<P> A familiarity with how to use the internet can help you research
and study topics across all areas of education!</P>
<P>The purpose of my project is...</P>
</body>
</html>
```

This is the minimal HTML text you need to use for a simple web page to be created. The title element contains your document title and identifies its content in a global context. The title is typically displayed in the title bar at the top of the browser window, but not inside the window itself. The title is also what is displayed on someone's hotlist or bookmark list, so choose something descriptive, unique, and relatively short. A title is also used to identify your page for search engines. The second and largest part of your HTML document is the body, which contains the content of your document (displayed within the text area of your browser window). The tags explained below are used within the body of your HTML document (NCSA, 2000).

2. Go under "File" and choose "Save As". Choose the place/folder you want to save this file in and then type the name of the file followed by ".htm". Make sure that you include the term **.htm** after your filename, otherwise it will not be readable by a web browser (e.g., Science Project Page1.htm –always use titles that characterize the content of your web page). After typing the name, click on "Save" of the dialogue box.
3. Open your web browser (e.g., Netscape or Internet Explorer) and select "Open Page" under "File". At this point you must have a dialogue box on your screen that asks you to "choose the file" you want to open. Click on "choose file" and select the filename you gave to your saved HTML document (e.g., Science Project Page1.htm – save your files in folders which are easily found). After you select your HTML document click on "Open" of the dialogue box. What do you observe?
4. HTML has six levels of headings, numbered 1 through 6, with 1 being the largest. Headings are typically displayed in larger and/or bolder fonts than normal body text (NCSA, 2000). The first heading in each document should be tagged <H1>. The

syntax of the heading element is: <Hy>Text of heading </Hy>, where y is a number between 1 and 6 specifying the level of the heading. Repeat step 1 for all possible y numbers (1-6) by changing the syntax of the heading element “<H1>Science is Fun</H1>” (e.g. “<H2>Science is Fun</H2>”, “<H3>Science is Fun</H3>” etc.). What do you observe? What happens if you use different numbers (e.g., “<H1>Science is Fun</H6>”)?

5. Unlike documents in most word processors, carriage returns in HTML files aren't significant. In fact, any amount of whitespace - including spaces, linefeeds, and carriage returns - are automatically compressed into a single space when your HTML document is displayed in a browser (NCSA, 2000). Repeat steps 1-3 but this time instead of writing the paragraph content as in step 1 (<P> A familiarity with how to use the internet can help you research and study topics across all areas of education!</P>), write it as:

**<P> A familiarity with how  
to use the internet can help  
you research and study topics  
across all areas of education!</P>**

And then as,

**<P> A familiarity with how to use the internet can  
help you research and study topics across all areas of  
education! </P>**

What do you observe? Compare all cases among them. Do you see the same differences as in the word document? Is it possible to present the paragraph as,

**“A familiarity with how  
to use the internet can help  
you research and study topics  
across all areas of education!”**

on your web page by using what you have learned so far?

6. Repeat steps 1-3 but this time instead of writing the paragraph content as in step 1 (<P> A familiarity with how to use the internet can help you research and study topics across all areas of education!</P>), write it as <P ALIGN=CENTER> A familiarity with how to use the internet can help you research and study topics across all areas of education!</P>. What do you observe? How about if you use the words RIGHT or LEFT, instead of CENTER.
7. Open Notepad/SimpleText and type in the following:

**<html>**

```
<head>
<TITLE> My Science Project's Homepage</TITLE>
</head>
<body>
<H1> Welcome To My Science Project's Homepage </H1>
<P> A familiarity with how to use the internet can help you research
and study topics across all areas of education!</P>
<UL>
<LI> Science is fun
<LI> Science is exciting
<LI> Science is interesting
</UL>
</body>
</html>
```

What do you observe? Explain.

8. Repeat step 7, but instead of using the <UL> and </UL> tags, use <OL> and </OL> respectively. What are the differences between step 7 and 8?
9. Open Notepad/SimpleText and type in the following:

```
<html>
<head>
<TITLE> My Science Project's Homepage</TITLE>
</head>
<body>
<H1> Welcome To My Science Projects Homepage </H1>
<P> A familiarity with how to use the internet can help you research
and study topics across all areas of education!</P>
<DL>
<DT> Science is fun
<DD> Science is exciting.
</DL>
</body>
</html>
```

What do you observe? Explain.

10. Include the following commands in your HTML body text of step 9 (one at a time) and describe their use (write a report where you name the tag and then explain its corresponding function).
- A.       <DL COMPACT>
           <DT> WWW:
           <DD> World Wide Web

- <DT> WWW:**  
**<DD> World Wide Web**  
**</DL>**
- B. **<UL>**  
**<LI> What is Science?**  
**<UL>**  
**<LI> Science is fun**  
**<LI> Science is exciting**  
**<LI> Science is interesting**  
**</UL>**  
**<LI> What is Science?**  
**<UL>**  
**<LI> Science is exciting**  
**<LI> Science is interesting**  
**</UL>**  
**</UL>**
- C. **<P> Toward More Democratic Education for All</P>**  
**<BLOCKQUOTE>**  
**<P>“Most governments have been based on the denial of equal rights; ours began by affirming those rights. They said, some men are too ignorant and vicious, to share in government. Possibly so, said we; and, by your system, you would always keep them ignorant and vicious. We propose to give all a chance; and we expected the weak to grow stronger, the ignorant, wiser; and all better, and happier together.”**  
**</P>**  
**<P>-- Abraham Lincoln, 1858 </P>**  
**</BLOCKQUOTE>**
- D. **Bell Labs Science Grant Program<BR>**  
**Commitment to High School<BR>**  
**Science Education<BR>**
- E. **<HR SIZE=4 WIDTH="50%">**  
**Substitute 4 with numbers 1-6, what to you observe? How about a number bigger than 6? How about a double-digit number?**
- F. **Science is fun**  
**<B>Science is fun</B>**  
**<I> Science is fun </I>**  
**<U> Science is fun </U>**  
**<TT> Science is fun </TT>**



11. The chief power of HTML comes from its ability to link text and/or an image to another document or section of a document. A browser highlights the identified text or image with color and/or underlines to indicate that it is a hypertext link. HTML's single hypertext-related tag is `<A>`, which stands for anchor (NCSA, 2000). To include an anchor in your document:

- start the anchor with `<A` (include a space after the A)
- specify the document you're linking to by entering the parameter `HREF="filename"` followed by a closing right angle bracket (`>`)
- enter the text that will serve as the hypertext link in the current document
- enter the ending anchor tag: `</A>` (no space is needed before the end anchor tag)

An example of a hypertext reference in a file could be “Science Project Page1.htm” or “Science Project Page1.html”:

**`<A HREF=" Science Project Page1.htm ">Back to Page 1</A>`**

To understand this better, repeat steps 1-3 and save your file as “Science Project Page1.htm” and then repeat step 7, but this time before you close your body text (the body text closes when you use the tag `</body>`) type in `<P><A HREF=" Science Project Page1.htm ">Back to Page 1</A></P>`. After you finish step 7, continue with steps 2 and 3, but make sure you will save this file with a different name (e.g. Science Project Page2.htm). As soon as you open your second file (the one you created by repeating step 7) you will see at the end of your web page “Back to Page 1”. Click on “Back to Page 1” and explain what you observe.

Make sure that you save all your files in the same folder. Otherwise you will have to specify in your hypertext reference the relative path from the current document to the linked document. In other words, you have to specify in which folder the specific file, you want to make the link to, is. For example, a link to the file Science Project Page1.htm located in the folder “Project”, which is saved on your desktop, would be: `<A HREF=" Project / Science Project Page1.htm "> Back to Page 1</A>`.

12. Repeat step 11, but this time add the following hypertext reference in your Science Project Page2 text after the hypertext reference you used in step 11: `<P><A HREF="http://www.lucent.com">Lucent</A></P>`. What do you observe when you click on “Lucent”? (Note: Make sure that you are on-line at the time you perform this step. In case you do not have internet access skip this step).
13. You can make it easy for a reader to send electronic mail to you or any other e-mail address you want by including the mailto attribute in a hyperlink. The format is: `<A HREF="mailto:emailinfo@host">Name</a>` (e.g. `<A HREF="mailto:zach@campus.com">Zach</a>`). Repeat step 12 and include the tag given in this step, as well. Do not forget to use your e-mail address and your name in the `<A HREF="mailto:emailinfo@host">Name</a>` tag. Describe what do you

observe (Note: Make sure that you are on-line at the time you perform this step. In case you do not have internet access skip this step) (NCSA, 2000).

14. Most Web browsers can display inline images (that is, images next to text) that are in X Bitmap (XBM), GIF, or JPEG format. Each image takes additional time to download and slows down the initial display of a document. Carefully select your images and the number of images in a document. To include an inline image, enter: `<IMG SRC=ImageName>`, where ImageName is the URL of the image file. The syntax for `<IMG SRC>` URLs is identical to that used in an anchor HREF. If the image file is a GIF file, then the filename part of ImageName must end with .gif. Filenames of X Bitmap images must end with .xbm; JPEG image files must end with .jpg or .jpeg (NCSA, 2000).  
To understand this better, visit a website where copying images is permitted (do not forget that posted material on the Web can be copyrighted and for copying any images for those websites, special permission must be granted by the creators of the particular website) and position your mouse's cursor on the picture you want to copy. Click the right button of your mouse and select "save image as" from the appeared dialogue box. Usually a name is automatically given for each image, but in case you want to change it, you can by erasing the already given name and by writing the new name. After you give it a name, click on "Save" of the dialogue box (make sure you know in which folder you saved your image. The best way is to save them in the same folder you are saving your WebPages). Let's say, you have saved the image in your WebPages' folder and you named it "image1.gif". Repeat step 7, but this time, before you close your body text, type in `<IMG SRC= image1.gif >`. After you finish step 7, continue with steps 2 and 3. Once again, explain what do you observe.
15. You should include two other attributes on `<IMG>` tags to tell your browser the size of the images it is downloading with the text. The HEIGHT and WIDTH attributes let your browser set aside the appropriate space (in pixels) for the images as it downloads the rest of the file (NCSA, 2000). For example, to include "image1.gif" along with the desired dimensions, enter: `< IMG SRC= image1.gif HEIGHT=100 WIDTH=65>`. Repeat step 14 by using this tag, instead of the one you used before in step 14. Compare the two images. Repeat again by using different numbers for HEIGHT and WIDTH. Are all numbers allowed? How many digits can the HEIGHT and WIDTH numbers have? Which are the dimensions that give you the best image quality?
16. Repeat step 14 by using the following tags (one at a time) and describe your observations:
  - `<IMG SRC = " image1.gif " ALT="[ image1]" ALIGN=TOP>`
  - `<IMG SRC = " image1.gif " ALT="[ image1]" ALIGN=CENTER>`
  - `<p ALIGN=CENTER><IMG SRC = " image1.gif " ALT="[image1]"></p>`
  - `<A HREF=" Science Project Page1.htm "><IMG SRC=" image1.gif " ALT="[ image1]"></A>`

- `< A HREF=" Science Project Page1.htm "><IMG SRC=" image1.gif " BORDER=0 ALT="[ image1]"></A>` (Repeat this tag for other BORDER values, besides zero, as well.)
- `<IMG SRC=" image1.gif " BORDER=6 ALT="[ image1]">`
- `<BODY BACKGROUND=" image1.gif ">`
- `<A HREF=" image1.gif ">link anchor</A>`

17. By default, browsers display text in black on a gray background. However, you can change both elements if you want. Some HTML authors select a background color and coordinate it with a change in the color of the text. Always preview changes like this to make sure your pages are readable. You change the color of text, links, visited links, and active links using further attributes of the `<BODY>` tag. For example: `<BODY BGCOLOR="#000000" TEXT="#FFFFFF" LINK="#9690CC">`. This creates a window with a black background (BGCOLOR), white text (TEXT), and silvery hyperlinks (LINK). The six-digit number and letter combinations represent colors by giving their RGB (red, green, blue) value. The six digits are actually three two-digit numbers in sequence, representing the amount of red, green, or blue as a hexadecimal value in the range 00-FF (NCSA, 2000). Repeat steps 1-3, but using further attributes of the `<BODY>` tag, such as `<BODY BGCOLOR="#000000">`, and complete the following table:

Code	Color
000000	
F00000	
FF0000	
FFF000	
000FFF	
0000FF	
00000F	
FFFFFF	
...	

Try to find particular color patterns. Remember to explain your reasoning for each particular pattern you identify. Is there a limit in terms of colors you can create? (To help you track down the combinations that map to specific colors, there is software available for you to do this on your workstation, visit the following website: [www.visibone.com/colorlab/](http://www.visibone.com/colorlab/)).

18. Tables are very useful for presenting tabular information. A table has headings where you explain what the columns include, rows for information, cells for each item. The following tags are of particular importance when creating a table (NCSA, 2000):

- `<TABLE> ... </TABLE>`: defines a table in HTML. If the BORDER attribute is present, your browser displays the table with a border.
- `<CAPTION> ...</CAPTION>`: defines the caption for the title of the table. The default position of the title is centered at the top of the table. The attribute

ALIGN=BOTTOM can be used to position the caption below the table.

NOTE: Any kind of markup tag can be used in the caption.

- **<TR> ... </TR>**: specifies a table row within a table. You may define default attributes for the entire row: ALIGN (LEFT, CENTER, RIGHT) and/or VALIGN (TOP, MIDDLE, BOTTOM). See Table Attributes at the end of this table for more information.
- **<TH> ... </TH>**: defines a table header cell. By default the text in this cell is bold and centered. Table header cells may contain other attributes to determine the characteristics of the cell and/or its contents. See Table Attributes at the end of this table for more information.
- **<TD> ... </TD>**: defines a table data cell. By default the text in this cell is aligned left and centered vertically. Table data cells may contain other attributes to determine the characteristics of the cell and/or its contents. See Table Attributes at the end of this table for more information.

NOTE: Attributes defined within **<TH> ... </TH>** or **<TD> ... </TD>** cells override the default alignment set in a **<TR> ... </TR>**.

- ALIGN (LEFT, CENTER, RIGHT): Horizontal alignment of a cell.
- VALIGN (TOP, MIDDLE, BOTTOM): Vertical alignment of a cell.
- COLSPAN=n: The number (n) of columns a cell spans.
- ROWSPAN=n: The number (n) of rows a cell spans.
- NOWRAP: Turn off word wrapping within a cell.

For example, open Notepad/SimpleText and type in the following required elements:

```
<html>
<head>
<TITLE> My Science Project's Homepage</TITLE>
</head>
<body>
<H1> Welcome To My Science Projects Homepage </H1>
<P> A familiarity with how to use the internet can help you research
and study topics across all areas of education!</P>
<P>The purpose of my project was...</P>

<TABLE >
<!-- start of table definition -->
<CAPTION> Color Coding </CAPTION>
<!-- caption definition -->

<TR>
<!-- start of header row definition -->
<TH> Code </TH>
<TH> Color </TH>
</TR>
```

```
<!-- end of header row definition -->

<TR>
<!-- start of first row definition -->
<TD> 000000 </TD>
<TD> Black </TD>
</TR>
<!-- end of first row definition -->

<TR>
<!-- start of last row definition -->
<TD> FFFFFFFF </TD>
<TD> White </TD>
</TR>
<!-- end of last row definition -->

</TABLE>
<!-- end of table definition -->
</body>
</html>
```

Change the <TABLE> to <TABLE BORDER=2> (Try different numbers). What do you observe? Reproduce the table you created in step 17. Try to fill each cell with the corresponding color, as well. (Note that the <!-- ... --> brackets are used here for only explanatory reasons. You do not have to include them when you will be designing your own tables.)

## Activities

1. Create your own personal WebPages. Include information about yourself, interests, school, extra curricula activities, personal projects, music preferences etc. Use a separate page for each topic, but make sure you link all pages among them (it will be a good idea to visit some other websites to get some ideas). Make your pages colorful and include pictures, tables, and links to other websites. Try to be as creative as possible.
2. Create the following tables in 3 different BORDER sizes. Fill each box with a different color and include both its name and its code within the box, as well.

A.


--	--	--	--

B.


C.


D.

<i>Colors</i>			
<b>Light</b>		<b>Dark</b>	
<i>Name</i>	<i>Code</i>	<i>Name</i>	<i>Code</i>
1. <u>White</u>	5.	9. <u>Black</u>	13.
2. <u>Yellow</u>	6.	10. <u>Brown</u>	14.
3. <u>Sky blue</u>	7.	11. <u>Dark blue</u>	15.
4. ...	8. ...	12. ...	16. ...

3. Create a reference book for someone who wants to learn HTML. Include all tags that were introduced in this project, along with their corresponding explanations. Visit the following websites in case you want to include more tags or if you want to get more in depth:

- <http://www.ology.org/tilt/cgh/>
- <http://www.ucc.ie/info/net/html/doc.html>
- [http://members.aol.com/htmlguru/about\\_html.html](http://members.aol.com/htmlguru/about_html.html)
- [http://www.cc.ukans.edu/~acs/docs/other/HTML\\_quick.shtml](http://www.cc.ukans.edu/~acs/docs/other/HTML_quick.shtml)

4. Create a tutorial on a certain topic (e.g., create a table where you include laws and principles and create for each one of them a link to another web page or website where definitions, equations, and explanations are given).



**YSAP**<sup>®</sup>  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

## **Final Project/Report**

Create a yearlong calendar by using strictly HTML coding. Devote it to something or someone and select pictures that represent each month based on your selection. Include holidays, memorial days, personal events, family and friends birthdays etc. Link some of those dates, particularly holidays and memorial days, to other websites.

## **Reference**

NCSA (2000). *A Beginner's Guide to HTML*. URL: <http://www.ncsa.uiuc.edu/>

## **Unit 3: Computer Programming in Science Education – C**

### **Project 14**

## **How to Create Your Own Computer Programs: An Introduction to C**

This project contains steps that help you to learn about the C programming language. The C programming language was developed in the years 1969 to 1973 and since then is considered one of the basic languages for programming. A familiarity with how to program with C can help you develop and run your own computer programs. But how do people create all these programs we enjoy on our computers? Follow us through the process of creating your own computer program and find out for yourself the answers to these questions.

### **Degree of Difficulty**

Experimental: Moderate to Difficult

Conceptual: Moderate to Difficult

### **Prerequisite Knowledge**

Experience using a Windows operating system is required (in case you are not working under UNIX, but use a C compiler). Experience using DOS is recommended. Strongly suggested for students with experience in other programming languages, such as, Fortran and Pascal.

**NOTE:** This project refers to C under the UNIX operating system or C on Windows if the system has an ANSI C compiler (this does not mean that there is no other way to compile and run your programs). In case you want to work with C under the UNIX operating system access to UNIX will be a prerequisite for this project (without UNIX you cannot run the programs). In case you do not know if you have access to UNIX ask your teacher or Mentor for help.

### **Useful Information/Concepts**

- C is a programming language, where values are stored in variables.
- Every C program consists of one or more **functions**, one of which must be called **main**. The program will always begin by executing the main function. Additional function definitions may precede or follow main.
- According to Gottfried (1990), each Function must contain:
  1. A function **heading**, which consists of the function name, followed by an optional list of **arguments** enclosed in parentheses.



2. A list of argument **declarations**, if arguments are included in the heading.
3. A **compound statement**, which comprises the remainder of the function.

- The **arguments or parameters** are symbols that represent information being passed between the function and other parts of the program.
- Each compound statement is enclosed within a pair of braces, i.e. { ... }. The braces may contain combinations of elementary statements (called **expression statements**) and other compound statements. Thus, compound statements may be nested, one within another. Each expression statement must end with a semicolon (;) (Gottfried, 1990).
- **Comments** (remarks) may appear anywhere within a program, as long as they are placed within the delimiters and / \* ... \* / (e.g., / \* Science is fun \* /). Such comments are helpful in identifying the program's principal features or in explaining the underlying logic of various program features (Gottfried, 1990).
- C programs will look similar under any other system (such as VMS or DOS), some other features will differ from system to system. In particular the method of compiling a program to produce a file of runnable code will be different on each system. The UNIX system is itself written in C. In fact C was invented specifically to implement UNIX (Holmes, 1995).
- In the course of the development of UNIX, hundreds of functions were written to give access to various facets of the system. These functions are available to the programmer in **libraries**, both on UNIX and Windows (Holmes, 1995).

## Objectives

Completion of the activities should enable you:

- Gain a basic understanding in programming theory, including: effective problem solving techniques, software development processes, and designing and analyzing simple algorithms.
- Become familiar with some basic constructs of the C language, including: the language syntax, data types and expressions, input and output, and program control with conditional and looping constructs.
- Learn to how to edit, compile, debug, and run C programs.

**Materials:** computer, a C or C++ compiler, UNIX access.

**NOTE:** When following the directions given bellow do not include in your C program the quotation marks used (the use of quotation marks is for separating the tags/commands from the rest of the text). However, there are cases where you have to use the quotation marks, but for these cases the quotation marks must be included within the parentheses ( ... ) [e.g., printf ("Radius = ? ");].

## Part A

### Introduction

“The art of taking a problem and breaking it down into a set of instructions you can give a computer is the interesting part of programming. Unfortunately it is also the most difficult part of programming as well. If you think that learning to program is simply a matter of learning a programming language you are very wrong. In fact if you think that programming is simply a matter of coming up with a program which solves a problem you are equally wrong!” (Miles, 1995).

There are many things you must consider when writing a program. Let’s say that you are writing your programs for a science class you are taking. Your science teacher assigned you a problem and would like you to write a program to solve it. Initially we are not even going to talk about the programming language, type of computer or anything like that, we are simply going to make sure what the problem is asking for. Consider a science problem, where the calculation of circle areas must be calculated. You are looking for a program which will read the radius of a circle, calculate the area and then write the calculated result. Thus, the first thing you need to do is to specify the problem. When considering how to write the specification of a system there are three important things:

- What information flows into the system.
- What flows out of the system.
- What the system does with the information.

1. Information going in: In the case of your problem we can describe the information as:

- The radius of a circle.

2. Information coming out

The information you want to see is:

- The area of the circle.

3. What the program actually does

The program can derive the value according to the following equation:

circle area = 3.14159 \* radius \* radius

(Note: in this case the (\*) represents multiplication)

Every C program should contain certain standard C commands (e.g. # include <stdio.h>). Each C program will usually consist of LIBRARY FILE ACCESS, TITLE, FUNCTION HEADING, VARIABLE DECLARATIONS, OUTPUT STATEMENT (PROMPT), INPUT STATEMENT, ASSIGNMENT STATEMENT, and an OUTPUT

STATEMENT (Gottfried, 1990). However, the minimum requirements for a C program are:

```
Main()
{

}
```

Every C program must have one and only one `main()` function (this is the start of the program.). The `{}` groups statements together and, as can be seen from above, the `{` is equivalent to begin and the `}` is equivalent to end. Comments can be placed anywhere inside C programs using `/*` and `*/`. For example:

```
/* My science project */
Main()
{
/* Science is fun */
}
```

For creating your first simple C program follow the next steps:

## Procedure

1. Open the C editor that is available to you. You can use any ordinary editor with which you are familiar to create the file. One such editor is `textedit`, which is available on most UNIX systems on Sun OpenWindows. But you can't use it if you telnet to the UNIX system. `Ed` and `vi` are available on all UNIX systems. The filename must, by convention, end with `".c"` (dot, lower case c), e.g., `myscpro.c` or `projectsc.c`. The contents must obey C syntax rules. For example, they might be as in the above example, starting with the line `/* Science is fun */` and ending with the line `/* end of program */`. For example, type in the C program for the calculated circle area we were discussing before:

```
# include <stdio.h>

/* program to calculate the area of a circle */

main ()
{
    float radius, area;
    printf ("Radius = ?");
    scanf ("%f", &radius);
    area = 3. 14 * radius * radius;
    printf ("Area = "%f", area);
}
```

We'll describe this program in detail below. This is the structure of a C program you need to use for a simple program that calculates a circle area. As we mentioned before, each C program, including this one, will usually consist of LIBRARY FILE ACCESS, TITLE, FUNCTION HEADING, VARIABLE DECLARATIONS, OUTPUT STATEMENT (PROMPT), INPUT STATEMENT, ASSIGNMENT STATEMENT, and an OUTPUT STATEMENT (see table 1).

**Table 1**

Code	Explanation
# include <stdio.h>	/* LIBRARY FILE ACCESS */
/* program to calculate the area of a circle */	/* TITLE (COMMENT) */
main ( )	/* FUNCTION HEADING */
float radius, area;	/* VARIABLE DECLARATIONS */
printf ("Radius = ?" );	/* OUTPUT STATEMENT (PROMPT) */
scanf ("%f", &radius);	/* INPUT STATEMENT */
area = 3. 14159 * radius * radius;	/* ASSIGNMENT STATEMENT */
printf ("Area = "%f", area);	/* OUTPUT STATEMENT */

A reasonable question at this point must be what do all this symbols or bits mean?

- **#include**: The pre-processor is the part of the compiler which actually gets your program from the file. This is a pre-processor directive. It is not part of our program, it is an instruction to the compiler to make it do something. It tells the C compiler to include the contents of a file, in this case the system file `stdio.h`. The compiler knows it is a system file, and therefore must be looked for in a special place, by the fact that the name is enclosed in `<>` characters (Miles, 1995).
- **<stdio.h>**: This is the name of the standard library definition file for all STanDard Input Output. Your program will almost certainly want to send stuff to the screen and read things from the keyboard. `stdio.h` is the name of the file in which the functions that we want to use are defined. The `.h` portion is the language extension, which denotes an include (helper) file. The `<>` characters around the name tell C to look in the system area for the file `stdio.h` (Miles, 1995).
- **main( )**: declares the start of the function, while the two curly braces show the start and finish of the function. Curly braces in C are used to group statements together as in a function, or in the body of a loop. Such a grouping is known as a compound statement or a block. C regards the name "main" as a special case and will run this function first. If you forget to have a main function, or mistype the name, the compiler will give you an error message (Miles, 1995).
- **{**: This is an opening curly brace. As the name implies, curly braces come in packs of two, i.e., for every open brace there must be a matching close. Braces allow you to lump pieces of a program together. Such a lump of program is

often called a block. A block can contain the declaration of variables used within it, followed by a sequence of program statements which are executed in order. In this case, the braces enclose the working parts of the function main (Miles, 1995).

- **;**: The semicolon marks the end of the list of variable names, and also the end of that declaration statement. All statements in C programs are ended by the **;** character; this helps to keep the compiler on the right track. The **;** character is actually very important. It tells the compiler where a given statement ends (Miles, 1995).
- **float**: Our program needs to remember certain values as it runs. Notably it will read in values for the radius and then calculate and print values for the circle area. C calls the place where values are put variables. At the beginning of any block you must tell C that you want to reserve some space to hold some data values. Each item can hold a particular kind of value. Essentially, C can handle three types of data, floating point numbers, integer numbers, and characters (i.e., letters, digits, and punctuation). You declare variables of a particular type by giving the type of the data, followed by a list of the names you want the variables to have (Miles, 1995).
- **scanf**: The standard input/output library contains a number of functions for formatted data transfer. The two we are going to use are **scanf** (scan formatted) and **printf** (print formatted). The compiler knows that **scanf** is a function call by the fact that it is followed by a parameter list. (see later). What the compiler does at this point is parcel up the parameters you have given and then call **scanf**, passing the parameter information to it. This function expects a particular sequence and type of parameters (Miles, 1995).
- **"%f"**: This is the set of parameters to the call of **scanf**. **scanf** is short for scan formatted. The function is controlled by the format string, which is the first parameter to the function. The second and successive parameters are addresses into which **scanf** puts the values it has been told to fetch. In C a string is given as a sequence of characters enclosed in **"** characters. You will meet the concept of delimiters regularly in C. A delimiter is a particular character which is to be used to define the limits of something. C uses different delimiters in different places, the **"** character is used to delimit a string of text. When the compiler sees the first **"**, it recognizes that everything up until the next **"** is a sequence of characters which we wish to use. It therefore just assembles the string until it finds another **"**. The **%** character tells **scanf** that we are giving the format of a number which is to be processed. The letter after the **%** character tells **scanf** what kind of value is being fetched, **f** means floating point. This command will cause **scanf** to look for a floating point number on the input (Miles, 1995).
- **area =**: This is an assignment. C uses the **=** character to make assignments happen. Whatever's to the right of the **=** will be placed into the variable which is named on the left of the **=** (Miles, 1995).
- **printf ("Radius = ?")**: prints the words on the screen. The text to be printed is enclosed in double quotes. Sometimes the **\n** symbol is used at the end of

the text, which tells the program to print a newline as part of the output (Miles, 1995).

According to Gottfried (1990), what is important to notice from this first example are the following:

- The program is typed in lowercase. Either upper- or lowercase can be used though it is customary to type ordinary instructions in lowercase. (Uppercase and lowercase characters are not equivalent in C. There are some special situations where certain symbols are characteristically typed in uppercase, but we will not examine them here).
- The first line of the given program contains a reference to a special file (stdio. h) which contains information that must be included in the program when it is compiled. The inclusion of this required information will be handled automatically by the compiler.
- The second line is a comment that identifies the purpose of the program.
- The third line is a heading for the function **main**. The empty parentheses following the name of the function indicate that this function does not include any arguments.
- The remaining five lines of the program are indented and enclosed within a pair of braces. These five lines comprise the compound statement within **main**. The indentation is not significant to the compiler – it's there to make it easier for people to read the program. But beware! If the curly braces don't match the indentation correctly, the program won't work as you expect it to, and it can be very difficult to find the problem.
- The first indented line is a **variable declaration**. It establishes the symbolic names radius and area as **floating-point variables**.
- The remaining four indented lines are expression statements. The second indented line (printf) generates a request for information (namely, a value for the radius). This value is entered into the computer via the third indented line (scanf).
- The fourth indented line is a particular type of expression statement called an assignment statement. This statement causes the area to be calculated from the given value of the radius. Within this statement, the asterisks (\*) indicate multiplication.
- The last indented line (printf) causes the calculated value for the area to be displayed. The numerical value will be preceded by a brief label.
- Notice that each expression statement within the compound statement ends with a semicolon. This is required of all expression statements.
- Finally, notice the liberal use of spacing and indentation, creating **whitespace** within the program. The blank lines separate different parts of the program into logically identifiable components, and the indentation indicates subordinate relationships among the various instructions. These features are not grammatically essential, but their presence is strongly encouraged as a matter of good programming practice (Gottfried, 1990).



2. To execute the program, you have to follow a rather complicated procedure, which involves compilation of the program before running it. There are many C compilers around. `cc` is the default Sun compiler. The GNU C compiler, `gcc`, is popular and available for many platforms. PC users may also be familiar with the Borland `bcc` compiler. Other (less common) C/C++ compilers exist. All the above compilers operate in essentially the same manner and share many common command line options. The best source of information about each compiler is through the online manual pages of your system: e.g., “`man cc`” on UNIX. For the sake of compactness, we will simply refer to the `cc` compiler. To compile your program, simply invoke the command `cc`. The command must be followed by the name of the file containing the C program you wish to compile (Marshall, 1999). Thus, the basic compilation command is:

**`cc program.c`**

where `program.c` is the name of the file. If there are obvious errors in your program (such as mistypings, misspelling one of the key words or omitting a semi-colon), the compiler will detect and report them. When the compiler has successfully digested your program, the compiled version, or executable, is left in a file called **`a.out`** or if the compiler option **`-o`** is used, the file listed after the **`-o`**. It is more convenient to use a **`-o`** and filename in the compilation as in

**`cc -o program program.c`**

which puts the compiled program into the file **`program`** (or any file you name following the “**`-o`**” argument) instead of putting it in the file **`a.out`** (Marshall, 1999). (If you want more detailed information about C compilation and running C, visit [http://www.hull.ac.uk/Hull/CC\\_Web/docs/cnotes/c3.html](http://www.hull.ac.uk/Hull/CC_Web/docs/cnotes/c3.html) or [http://www.strath.ac.uk/CC/Courses/NewCcourse/section3\\_5.html](http://www.strath.ac.uk/CC/Courses/NewCcourse/section3_5.html)).

3. To run an executable program in UNIX, you simply type the name of the file containing it, in this case **`program`** (or **`a.out`**). This executes your program, printing any results to the screen. Running the program for the calculation of the circle area results in an interactive dialog such as that shown below. The user’s response is underlined, for clarity.

Radius = ? 3  
Area = 28. 274309

4. Try different radius values and check the program by using a calculator to see if the program gives you the correct answer. Does the program work for decimal numbers? Does the program work for fractions? Is there an upper or lower limit in the value you can give to the radius?

## Activities

1. Follow the same procedure as above to create a C program that calculates the area of:
  - Squares
  - Rectangles
  - Triangles
  - Hexagons
2. Follow the same procedure as above to create a C program that calculates the volume of:
  - Cube
  - Box
  - Cylinder
  - Pyramid
  - Sphere
3. Create a C program that converts units (i.e., from feet to meters, from Fahrenheit to Celsius, etc.)
4. Create a C program that makes any calculation you want. Consider the capabilities and limitations of the area program through the process of creating your own program. Explain.
5. If you rearrange the lines of the given program, does the program run? What if you discard one of the semicolons?

## Part B

In this part we will attempt to get in more depth by considering a wider range of commands we can use. More specifically, we will focus on **conditional statements (if)** and **looping**, which are two of the main parts of a C program. Pay special attention to the notes listed below before starting the procedure steps of this part.

### A. DATA TYPES AND CONSTANTS

According to Gottfried (1990), the four basic data types are:

- **INTEGER**: These are whole numbers, both positive and negative. Unsigned integers (positive values only) are supported. In addition, there are short (**short int small\_value = 114h;**) and long (**long int big\_number = 245032L;**) integers. The keyword used to define integers is **int**. An example of declaring an integer variable called g is

```
int g;  
g = 10;
```



- **FLOATING POINT:** These are numbers which contain fractional parts, both positive and negative. The keyword used to define float variables is,

**float**

An example of declaring a float variable called gravity constant is,

```
float gravity_constant;  
gravity_constant = 9.8;
```

- **DOUBLE:** These are exponential numbers, both positive and negative. They're the same as **float**, except that they use double the storage space, which means they can retain more digits or precision than **floats**. The keyword used to define double variables is

**double**

An example of declaring a double variable called N is,

```
double N;  
N = 6E+23;
```

- **CHARACTER:** These are single characters. The keyword used to define character variables is,

**char**

An example of declaring a character variable called symbol is

```
char symbol;  
symbol = 'A';
```

Note that the assignment of the character A to the variable letter is done by enclosing the value in single quotes, also known as apostrophes.

*Sample program illustrating each data type*

```
#include < stdio.h >  
  
main()  
  
{  
  
    int g;  
    float gravity_constant;
```

```
char symbol;
double N;
g = 10;          /* assign integer value */
gravity_constant = 9.8; /* assign float value */
symbol = 'A';    /* assign character value */
N = 6E23;        /* assign a double value */

printf("value of g = %d\n", g );
printf("value of gravity constant = %f\n", gravity_constant);
printf("value of symbol = %c\n", symbol);
printf("value of N = %e\n", N);

}
```

- **Initializing Data Variables at Declaration Time:** In C, variables may be initialized with a value when they are declared. Consider the following declaration, which declares an integer variable **count** which is initialized to 8.

```
int count = 8;
```

- **Simple Assignment of Values to Data Variables:** The = operator is used to assign values to data variables. Consider the following statement, which assigns the value 12 an integer variable count, and the letter Z to the character variable letter

```
count = 12;
letter = 'Z';
```

## B. CONDITIONAL EXECUTION – if

### *The if else Statement*

This is used to decide whether to do something at a special point, or to decide between two courses of action. The following test decides whether a book is more expensive than 35 dollars

```
if (book >= 35)
    printf("Expensive\n");
else
    printf("Cheap\n");
```

It is possible to use the **if** part without the else.

```
if (temperature < 40)
    print("Cold\n");
```

Each version consists of a test, (this is the bracketed statement following the **if**). If the test is **true** then the next statement is obeyed. If it is **false** then the statement following the **else** is obeyed if present. After this, the rest of the program continues as normal. If we wish to have more than one statement following the **if** or the **else**, they should be grouped together between curly braces. Such a grouping is called a compound statement or a block (Holmes, 1995).

```
if (book >= 35)
{
    printf("Expensive\n");
    printf("Buy\n");
}
else
{
    printf("Cheap\n");
    printf("Do not buy\n");
}
```

Sometimes we wish to make a multi-way decision based on several conditions. The most general way of doing this is by using the **else if** variant on the **if** statement. This works by cascading several comparisons. As soon as one of these gives a true result, the following statement or block is executed, and no further comparisons are performed (Holmes, 1995). In the following example we are characterizing the weather conditions depending on the temperature.

```
if (temp >= 85)
    printf("Hot\n");
else if (temp >= 70)
    printf("Warm\n");
else if (temp >= 55)
    printf("Cold\n");
else
    printf("Very Cold\n");
```

In this example, all comparisons test a single variable called **temperature**. The same pattern can be used with more or fewer **else ifs**, and the final lone **else** may be left out. It is up to the programmer to devise the correct structure for each programming problem.

### C. RADIX CHANGING

- Data numbers may be expressed in any base by simply altering the modifier, e.g., decimal, octal, or hexadecimal. This is achieved by the letter which follows the **%** sign related to the **printf** argument. (In case you do not know what a radix or an octal or hexadecimal number is, search on the web to find their meanings).

```
#include <stdio.h>
```

```
main() /* Prints the same value in Decimal, Hex and Octal */
```

```
{
    int  number = 100;
    printf("In decimal the number is %d\n", number);
    printf("In hex the number is %x\n", number);
    printf("In octal the number is %o\n", number);
    /* what about %X\n as an argument? */
}
```

#### D. DEFINING VARIABLES IN OCTAL AND HEXADECIMAL

- Integer constants can be defined in octal or hex by using the associated prefix, e.g., to define an integer as an octal constant use 0 as the first character

```
int  sum = 0567; /* zero, not oh */
```

To define an integer as a hex constant use 0x

```
int  sum = 0x7ab4;
int  flag = 0x7AB4; /* note that the hex characters can be either upper or
lower case */
```

*Sample program illustrating each data type*

```
#include <stdio.h>
```

```
main()

{

    int  sum = 100;
    char  letter = 'Z';
    float set1 = 23.567;
    double num2 = 11e+23;

    printf("Integer variable is %d\n", sum);
    printf("Character is %c\n", letter);
    printf("Float variable is %f\n", set1);
    printf("Double variable is %e\n", num2);

}
```

To change the number of decimal places printed out for float or double variables, modify the %f or %e to include a precision value, e.g.,

```
printf("Float variable is %.2f\n", set1 );
```

In this case, the use of %.2f limits the output to two decimal places.

## **E. DIFFERENT TYPES OF INTEGERS**

- A normal integer is limited in range to plus or minus 32767. This value differs from computer to computer. It is possible in C to specify that an integer be stored in four memory locations instead of the normal two. This increases the effective range and allows very large integers to be stored. The way in which this is done is as follows,

```
long big_number = 211030L;
```

To display a long integer, use %l, e.g.,

```
printf("A larger number is %l\n", big_number );
```

Short integers are also available, e.g.,

```
short small_value = 101h;  
printf("The value is %d\n", small_value);
```

## **F. PREPROCESSOR STATEMENTS**

The define statement is used to make programs more readable. Consider the following examples,

```
#define TRUE  1  
#define FALSE 0  
#define NULL  \0  
#define AND   &  
#define OR    |  
#define EQUALS ==  
  
game_over = TRUE;  
while( list_pointer != NULL )  
.....
```

**Note** that preprocessor statements begin with a # symbol, and are NOT terminated by a semi-colon.

## G. ARITHMETIC OPERATORS

The symbols of the arithmetic operators are:-

<b>Multiply</b>	*
<b>Divide</b>	/
<b>Add</b>	+
<b>Subtract</b>	-
<b>Increment</b>	++ /* e.g., ++sum; is almost the same as sum = sum + 1 */
<b>Decrement</b>	-- /* e.g., --sum; is almost the same as sum = sum - 1 */
<b>Modulus</b>	%

**Note:** If you just have a statement **++sum;**, it's the same as **sum++;**. But **x=++sum;** is different than **x=sum++;**. **x=++sum;** is the same as **sum=sum+1; x=sum;**, but **x=sum++;** is the same as **x=sum; sum=sum+1;**. That is, with **++sum;**, **sum** is incremented before its value gets assigned to **x**, while with **x=sum++;**, **sum** is incremented after its value is assigned to **x**, so that **x** gets the initial value, rather than the final value of **sum**.

The following code fragment adds the variables **loop** and **count** together, leaving the result in the variable **sum**

```
sum = loop + count;
```

**Note:** If a % sign must be displayed as part of a text string in a printf, use two, i.e.,%%

```
#include <stdio.h>

main()
{
    int sum = 50;
    float modulus;
    modulus = sum % 10;
    printf("The %% of %d by 10 is %f\n", sum, modulus);
}
```

## H. THE RELATIONAL OPERATORS

These allow the comparison of two or more variables.

<b>==</b>	<b>equal to</b>
<b>!=</b>	<b>not equal</b>
<b>&lt;</b>	<b>less than</b>

**<=**    **less than or equal to**  
**>**     **greater than**  
**>=**   **greater than or equal to**

## **I. GETTING VALUES INTO THE PROGRAM**

- We have used the printf (print-formatted) standard input/output procedure from stdio.h to do the output. What we need is an equivalent procedure to do the input. C has got one of these, it is called scanf (scan-formatted). However, before we can turn it loose we have a little problem to solve; Where does scanf put the value that it fetches? All it was created for is to take something from one place (the keyboard) and put it somewhere else (a location). scanf does not want to know what the variable is called, all it needs to know is where the variable lives.
- When we talk about C variables, we mean a box which the compiler creates for us, with a name nicely painted on it. The compiler makes a box like this and then puts it on a shelf somewhere safe. The compiler then knows that when we say "i" we really mean the box with i written on it. scanf is simply a function that we call to fetch a value and stick it somewhere else. It needs to be told which box to put the result in. C calls referring to a thing by its location rather than its name as pointing.
- When we want scanf to fetch something for us, we have to give it a pointer. You can regard a pointer as a tag on the end of a piece of rope. The rope is tied to one of our boxes. All scanf has to do is follow the piece of rope to a box and then put the value into that box.
- You tell the C compiler to generate a pointer to a particular variable by putting an ampersand "&" in front of the variable name, i.e.,

**x means the value of the variable x**

**&x means a pointer to the box where x is kept, i.e., the address of x.**

scanf looks very like printf, in that it is a format string followed by a list of items, but in this case the items pointers to variables, rather than values, for example:

```
scanf ( "%d %d %d", &i, &j, &k) ;
```

## **Procedure**

1. Repeat the procedure steps 1-3 of Part A, but this time use the following C program:

```
#include <stdio.h>  
main() /* counts to ten */  
{  
    int count;  
    for( count = 1; count <= 10; count = count + 1 )  
    {  
        printf("%d ", count );  
    }  
}
```

```
printf("\n");
}
```

The program declares an integer variable **count**. The first part of the **for** statement, **count = 1**, initializes the value of **count** to 1. The **for** loop continues while the condition **count <= 10** evaluates as **TRUE**. As the variable **count** has just been initialized to 1, this condition is **TRUE** and so the program statement **printf("%d ", count );** is executed, which prints the value of **count** to the screen, followed by a space character. Next, the remaining statement of the **for** is executed **count = count + 1** which adds one to the current value of **count**. Control now passes back to the conditional test, **count <= 10** which evaluates as **TRUE**, so the program statement **printf("%d ", count );** is executed. **count** is incremented again, the condition re-evaluated, etc, until **count** reaches a value of 11. When this occurs, the conditional test **count <= 10** evaluates as **FALSE**, the **for** loop terminates, and program control passes to the statement **printf("\n");** which prints a newline, and then the program terminates, as there are no more statements left to execute.

Compare this program with the previous one (Part A) and write down the explanation for each bit/symbol. What does exactly this program do? What will happen if you change the numbers of **for( count = 1; count <= 10; count = count + 1 )** to

- **for( count = 1; count <= 20; count = count + 1 )**
- **for( count = 2; count <= 20; count = count + 2 )**
- **for( count = 2; count <= 10; count = count + 1 )**
- **for( count = 1; count <= 5; count = count + 1 )**
- **for( count = 5; count <= 100; count = count + 5 )**

2. Repeat the procedure steps 1-3 of Part A, but this time use the following C program:

```
#include <stdio.h>
main() /* a for loop to print out the values 1 to 10 on separate lines*/
{
    for( loop = 1; loop <= 10; loop = loop + 1 )
        printf("%d\n", loop) ;
}
```

What does this program do? How does it differ from the previous program? What will happen if you change the numbers of **for( loop = 1; loop <= 10; loop = loop + 1 )** to

- **for( loop = 1; loop <= 30; loop = loop + 1 )**
- **for( loop = 2; loop <= 10; loop = loop + 2 )**
- **for( loop = 5; loop <= 50; loop = loop + 5 )**
- **for( loop = 2; loop <= 20; loop = loop + 4 )**
- **for( loop = 1; loop <= 15; loop = loop + 1 )**



3. Repeat the procedure steps 1-3 of Part A, but this time use the following C program:

```
#include <stdio.h>
main()
{
    for( loop = 1; loop <= 5; loop = loop + 1 )
    {
        for( count = 1; count <= loop; count = count + 1 )
            printf("%d", count );
        printf("\n");
    }
}
```

What does this program do? How does it differ from the previous programs? What will happen if you change the numbers of **for( loop = 1; loop <= 5; loop = loop + 1 )** to

- **for( loop = 1; loop <= 10; loop = loop + 1 )**
- **for( loop = 2; loop <= 8; loop = loop + 2 )**

4. Repeat the procedure steps 1-3 of Part A, but this time use the following C program:

```
/* Sample program including while */

#include <stdio.h>

main()

{
    int loop = 0;
    while( loop <= 10 )
    {
        printf("%d\n", loop);
        ++loop;
    }
}
```

The above program uses a **while** loop to repeat the statements **printf("%d\n", loop); ++loop;** while the value of the variable **loop** is less than or equal to 10. Note that the variable upon which the **while** is dependent is initialized prior to the **while** statement (in this case the previous line), and also that the value of the variable is altered within the loop, so that eventually the conditional test will succeed and the **while** loop will terminate.

5. The **if** statement allows branching (decision making) depending upon the value or state of variables. This allows statements to be executed or skipped, depending upon decisions. The basic format is (for more details see notes of Part B),

```
if( expression )  
program statement;
```

*Example*

```
if( students < 45 )  
{  
    ++student_count;  
}
```

In the above example, the variable **student\_count** is incremented by one only if the value of the integer variable **students** is less than 45.

Repeat the procedure steps 1-3 of Part A, but this time use the following C program:

```
#include <stdio.h>  
  
main()  
{  
    int number;  
    int valid = 0;  
    while( valid == 0 ) {  
        printf("Enter a number between 1 and 10 -->");  
        scanf("%d", &number);  
        /* assume number is valid */  
        valid = 1;  
        if( number < 1 )  
        {  
            printf("Number is below 1. Please re-enter\n");  
            valid = 0;  
        }  
        if( number > 10 )  
        {  
            printf("Number is above 10. Please re-enter\n");  
            valid = 0;  
        }  
    }  
    printf("The number is %d\n", number );  
}
```

Write down what function this program performs. Explain what happens if you change the number 10 with a smaller and a bigger number.

## Activities

1. Write a C program using a **for loop** to print out characters from A to K. Try to have as many different outputs as possible (i.e. ABCD...K, or A,B,C,D...K etc.)
2. Write a **for loop** to print out on separate lines:
  - the values 1 to 6
  - the values 1 to 13
  - the letters A to N
3. Use a while loop to print the integer values 1 to 10 on the screen: **12345678910, 1 2 3 4 5 6 7 8 9 10, and 1,2,3,4,5,6,7,8,9,10.**
4. Use a **nested while loop** to produce the following output  
  
1  
  
22  
  
333  
  
4444  
  
55555
5. Use an **if** statement to compare the value of an integer called “price” against the value 20, and if it is more, print the text string "Expensive".
6. Use an **if else** statement to create a C program which categorizes products according to their weight. Use as categories the words “Extremely Heavy”, “Very Heavy”, “Heavy”, “Normal”, “Light”, and “Very Light”. You are responsible for defining the corresponding weight ranges. Compare the value of an integer called “weight” against the weight ranges, and print the corresponding characterization (i.e. Heavy, Normal, Light etc.).
7. Create a C program that calculates the total area of houses that have 3 bedrooms, 2 bathrooms, 1 kitchen, 1 sitting room, 1 hallway, and 1 dining room. In addition, create a separate C program that calculates the total volume, as well as the area.
8. Assume that you were hired by a realtor’s office to create a C program, which calculates the total area of houses as in activity 7. This time, though, you have to

categorize houses according to the area they occupy. The program must also have the capability to list the houses that fall within particular ranges (i.e.,  $300 \text{ m}^2 - 500 \text{ m}^2$ ).

9. Write a C program combining parts A and B of this project. For example, write a program that gives you the area of rectangles that are less than  $160 \text{ m}^2$ , more than  $160 \text{ m}^2$ , equal to  $160 \text{ m}^2$ , within the range  $100 \text{ m}^2 - 250 \text{ m}^2$ .
10. Predict what particular function this program performs before running the program. Run the program and explain any discrepancies between what you observe and what you predicted.

```
#include <stdio.h>

main()
{
    int input;
    printf("is the number positive, negative or zero?"\n");
    printf("enter your number here --->");
    scanf("%d", &input );
    (input < 0) ? printf("negative\n") : ((input > 0) ?
    printf("positive\n") : printf("zero\n"));
}
```

Describe what exactly does this program do? By following the same logic, create a program that tells you whether a number is an integer, character, float or double variable.

Note: This program goes beyond what you have been doing so far. You will have to look up the `?`, and find out how to determine what type a variable is. You can find `?` in all the websites that are cited in the references section.

11. Predict what particular function the following program developed by Holmes (1995), performs before running it. Run the program and explain any discrepancies between what you observe and what you predicted.

```
#include <stdio.h>
#define KILOS_PER_POUND .45359
main()
{
    int pounds;
    printf(" US lbs    UK st. lbs    INT Kg\n");
    for(pounds=10; pounds < 250; pounds+=10)
    {
        int stones = pounds / 14;
        int uklbs = pounds % 14;
        float kilos = pounds * KILOS_PER_POUND;
        printf("  %d      %d  %d      %f\n",
```

```
    pounds, stones, uklbs, kilos);
  }
}
```

Describe what exactly this program does. By following the same logic, create a shoe size conversion program. Include in your program the US, UK (United Kingdom), FR (France), and JP (Japan) shoe metric systems. The relationships among the different shoe metric systems are left for you to be found out as part of your research.

12. Predict what particular function the following programs, developed by Holmes (1995), perform before running them. Run the program and explain any discrepancies between what you observe and what you predicted.

**A.**

```
#include <stdio.h>
void print_converted(int pounds)
/* Convert U.S. Weight to Imperial and International Units. Print the
results */
{
    int stones = pounds / 14;
    int uklbs = pounds % 14;
    float kilos_per_pound = 0.45359;
    float kilos = pounds * kilos_per_pound;
    printf("  %3d      %2d %2d      %6.2f\n",
pounds, stones, uklbs, kilos);
}
main()
{
    int us_pounds;
    printf(" US lbs    UK st. lbs    INT Kg\n");
    for(us_pounds=10; us_pounds < 250; us_pounds+=10)
    print_converted(us_pounds);
}
```

**B.**

```
#include <stdio.h>

void print_converted(int pounds)
/* Convert U.S. Weight to Imperial and International
Units. Print the results */
{
    int stones = pounds / 14;
    int uklbs = pounds % 14;
```

```

float kilos_per_pound = 0.45359;
float kilos = pounds * kilos_per_pound;
printf("  %3d      %2d %2d      %6.2f\n",
pounds, stones, uklbs, kilos);
}

main()
{
int us_pounds;
printf("Give an integer weight in Pounds : ");
scanf("%d", &us_pounds);
printf(" US lbs      UK st. lbs      INT Kg\n");
print_converted(us_pounds);
}

```

Compare these programs with the one in activity 11. Explain their differences and similarities. Create two more shoe size conversion programs, as in activity 11, but this time use the logic introduced in this activity.

13. Predict what particular function the following Miles' program performs before running it. Run the program and explain any discrepancies between what you observe and what you predicted.

```

#include <stdio.h>

/* Define our window size range */
#define MAX_HEIGHT 3.0
#define MAX_WIDTH 5.0
#define MIN_HEIGHT 0.75
#define MIN_WIDTH 0.5

/* Define a few costs */
#define COST_TO_MAKE 2.00
#define WOOD_COST 2.00
#define GLASS_COST 3.00
#define MARKUP_FACTOR 2.25

/* Define the maximum number of windows on our house */
#define MAX_WINDOWS 10

/* Program variables : */
/* width - width of current window */
/* height - height of current window */
/* window_cost - cost to make the window */
/* window_sell - amount we sell the window for */
/* house_cost - cost to do the whole house */
/* house_sell - amount we sell the house job for */

```

```

float width, height, window_cost, window_sell, house_cost, house_sell ;

/* no_of_windows - number of windows in the house      */
/* window_count - counter for current window          */
int no_of_windows, window_count ;
void main ()
{
    printf ( "Double Glazing House Calculator\n" ) ;

    do {
        printf ( "Give the number of windows : " ) ;
        scanf ( "%d", &no_of_windows ) ;
    } while ( (no_of_windows <= 0) ||
              (no_of_windows > MAX_WINDOWS) ) ;

    house_cost = 0.0 ;
    house_sell = 0.0 ;

    for (window_count = 1 ; window_count <= no_of_windows ;
         window_count++) {
        printf ( "Window %d details.\n", window_count ) ;
        do {
            printf ( " enter the width : " ) ;
            scanf ( "%f", &width ) ;
        } while ( (width < MIN_WIDTH) ||
                  (width > MAX_WIDTH) ) ;
        do {
            printf ( " enter the height : " ) ;
            scanf ( "%f", &height ) ;
        } while ( (height > MIN_HEIGHT) ||
                  (height < MAX_HEIGHT) ) ;
        window_cost = WOOD_COST * 2 * (width + height) ;
    window_sell = window_cost * MARKUP_FACTOR ;
        printf ( "Window %d costs %.2f, sells for %.2f.\n\n",
        window_count, window_cost, window_sell ) ;
        house_cost += window_cost ;
        house_sell += window_sell ;
    }
    printf ( "\nTotal cost to do all %d windows : %.2f.\n",
            no_of_windows, house_cost ) ;
    printf ( "\nTotal sale price for all %d windows : %.2f.\n",
            no_of_windows, house_sell ) ;
}

```

Describe what exactly this program does.

## Final Project/Report

Create a C program that will be of use for a science project. Include a report explaining the reasoning behind your selection (purpose, benefits from its operation, etc.) and an analytical step by step description of your program. Describe the science project, and how your program is useful for it.

## References

- Holmes, S (1995). *An Introduction to C Programming*. URL:  
[http://www.strath.ac.uk/CC/Courses/NewCcourse/tableofcontents3\\_1.html](http://www.strath.ac.uk/CC/Courses/NewCcourse/tableofcontents3_1.html)
- Gottfried, B. (1990). *Programming with C*. NY: McGraw-Hill, INC.
- Jones, K. (2000). *Introduction to Programming in C/C++ with Vim*.  
URL: [http://www.linuxnewbie.org/nhf/intel/programming/intro\\_c++.html](http://www.linuxnewbie.org/nhf/intel/programming/intro_c++.html)
- Marshall, A. D. (1999). *Programming in C*. URL:  
<http://www.cs.cf.ac.uk/Dave/C/CE.html>
- Miles, R (1995). *Introduction to C Programming*. URL:  
[http://www.hull.ac.uk/Hull/CC\\_Web/docs/cnotes/contents.html](http://www.hull.ac.uk/Hull/CC_Web/docs/cnotes/contents.html)
- Whimster, C. (2000). *An Introduction to C Programming*. URL:  
<http://www.edm2.com/0408/introc1.html>



## **Unit 3: Computer Programming in Science Education – MicroWorlds Pro Project 15**

### **Create Your Own Simulations!**

MicroWorlds Pro is a multimedia programming environment that offers you the possibility to create dynamic, interactive school and Internet projects. It lets you become active web designers, not just passive web viewers. You can use MicroWorlds Pro to enhance your understanding of MicroWorlds and to get a real sense of the depth and breadth of this powerful multimedia programming environment (MicroWorlds Pro, 2000).

Among other things, MicroWorlds Pro allows you to import a picture, insert text, stamp text, create animations, create simulations, and create interactive buttons. This project will mostly focus on creating interactive computer simulations, which have proven an ideal tool for teaching and learning, especially in science education. But enough with the theory, let's find out how people create simulations by using a multimedia environment such as MicroWorlds Pro. Follow us through the process of creating your own animations and find out for yourself the answer to this question or others that have been troubling you!

### **Degree of Difficulty**

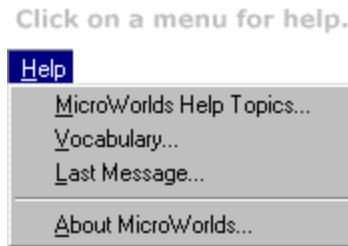
Experimental: Moderate  
Conceptual: Moderate

### **Prerequisite Knowledge**

This project is suggested for students that are already familiar with basic computer skills (e.g., turn on the computer, use the keyboard, use the mouse), have some experience in basic computer functions (e.g., access programs, open programs, save data), and have knowledge of basic computer programs (e.g., word processing programs, spreadsheets etc.).







**Note:** For programming in MicroWorlds Pro, **Logo** knowledge is needed. Logo is a language for computers. It has a very small number of words and grammatical rules. Where a spoken language has sentences, Logo has instructions. Where a spoken language's rules for how to put words into sentences can be complicated with many exceptions, Logo's rules for building instructions are much simpler but more stringent (MicroWorlds Pro, 2000). Thus, it is of particular importance to go through the MicroWorlds Pro topics (click on **Help** and select **MicroWorlds Pro topics** – see figure 1) before starting this project, where, among other things, Logo is introduced.

**Figure 1**







Particularly, read the following topics:






**A. MicroWorlds Pro Fundamentals**

-  An Overview of MicroWorlds Pro
-  Turtles
-  Graphics
-  Text Boxes
-  More About MicroWorlds Objects
-  Programming Environment

**B. Logo Programming**

-  Logo Fundamentals
-  Programming With Turtle Geometry
-  Programming With Texts
-  Programming With Numbers, Words, and Lists

**C. Interesting Concepts and Techniques**

-  Process Management
-  Variables
-  Object Manipulation Under Program Control
-  Programming Techniques
-  Sharing Projects

For the purposes of this project, we will consider all this information to be prior knowledge, including the Logo and MicroWorld Pro vocabulary, which you can find under the Help menu. As mentioned before, the Logo and MicroWorld Pro vocabulary consist of commands that you use to program the functions of each turtle you use.

In case you want to check whether your program's vocabulary or syntax has any mistakes, press the F4 key and MicroWorlds will find common syntax errors. The error message will be displayed, at the same time, in the Status Bar. The errors are highlighted in sequence each time you press F4. If you press F4 when the Procedures Tab area is not displayed, it will be displayed automatically (MicroWorlds Pro, 2000).

## Objectives

Completion of the activities should enable you:

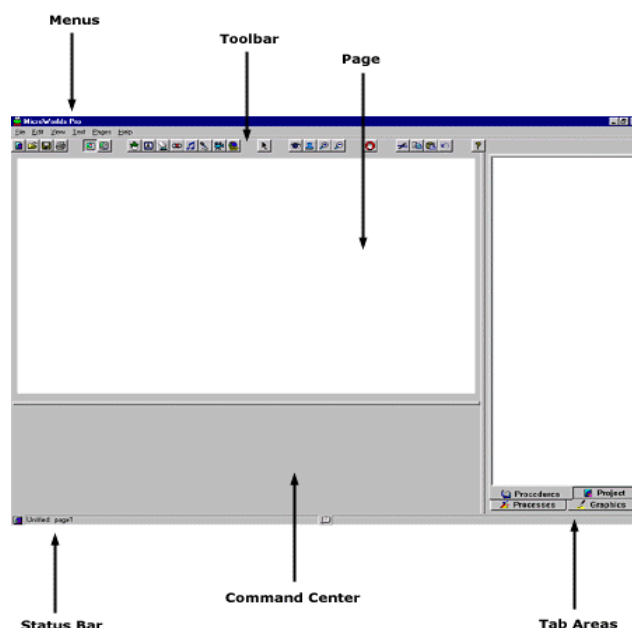
- to learn how to create animations by using MicroWorlds Pro.
- to use MicroWorlds Pro as a tool to create simulations.
- to post your animations or simulations on the World Wide Web.

**Materials:** computer, MicroWorlds Pro (visit <http://www.microworlds.com/solutions/mwpro.html> for purchasing the software), the Learning MicroWorlds Pro and the MicroWorlds Pro Tips and Tricks books that are included in the package of MicroWorlds Pro.

**Useful Information:** The Right Click on your PC's mouse is used to open pop-up menus on objects; the same functionality is obtained on a MAC by Control Click. The PC's Shift key is used to add more than one shape to a turtle and to change the heading of the turtle; the same functionality is obtained on a MAC with the Command key.

Each MicroWorld Pro (MWP) project consists of objects and text presented on separate pages. For every project you start, there is a corresponding single page. A page is accompanied by Menus, Toolbar, Status Bar, Command Center, and Tab Areas (see figure 2).

**Figure 2**




It is also important to know about Logo's punctuation and markers meaning. According to MicroWorlds Pro (2000):




Punctuation and Markers	Meaning
" Quotation mark	Indicates that the next word is to be taken literally. Note that we are only talking about one quote that is placed at the beginning of the word.
[ ] Square brackets	Surround a list.
: Colon	Sometimes referred to as dots. Indicates that the next word is the name of a variable. Refers to contents of the container.
( ) Parentheses	Group things in ways that Logo ordinarily would not or varies the number of inputs for a primitive.
Vertical bars	Surround a long word so you can include spaces.

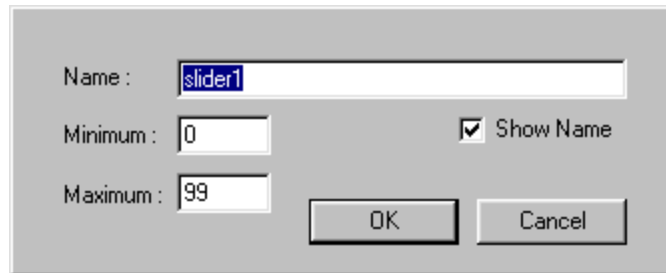
## Part A: Animations involving both movement and motion


### Procedure

Before creating our first simulation, let us introduce animation and, more specifically, explain how movement and motion in an animation can be performed. Let's suppose that you want a car to move across your page. Follow the next steps:

1. Create a turtle by selecting  and click anywhere on the page. Click on the Graphics tab, click on the car shape, and then click on the turtle (or you can type **setshape "car** in the Command Center). What do you observe?
2. Right-click on the car and from the pop-up menu select animate. What do you observe? Now, press the **Shift** button on your keyboard and drag the car towards the left or right (or you can type **seth 270** in the Command Center – 0 corresponds to north, 90 corresponds to East, 180 corresponds to south, 270 corresponds to west). Click on the car and describe what you observe. To stop the car, click on it (or click on the stop button on the toolbar). Try this for different angles (i.e., 30<sup>0</sup>, 45<sup>0</sup>, 56<sup>0</sup>, 23.5<sup>0</sup>) and explain your observations.
3. Right-click on the car and select **Edit**. In the pop-up dialog box, you will see the commands that were inserted automatically due to the animations we have already selected. Change the **forward** input from 5 to 10. What do you observe? Change the **wait** input from 1 to 5. What do you observe?

4. Repeat step 1. Create a button by selecting , click on the page, and type in the dialog box **forward 5 wait 1**. Click on the **OK** button of the dialog box and then click on the button created on your page. Compare it with step 3. How can you create a button without selecting  ?
5. Repeat step 1. Create a slider by selecting , click on the page, and type in the dialog box



Create a turtle, right-click on it and select Edit to open its dialog box. Type **fd slider1** as the turtle's instruction and set it to Many Times. Click on the turtle to start it. Use the slider to change the speed. Compare it with steps 3 and 4. How can you create a slider without selecting  ?

6. Repeat steps 1 to 3 and add two more cars on your page (or you can just right –click on the already existing car, select **copy**, and then click on the page and select **paste**). Type **everyone [clickon]** in the Command Center and press the **enter** button on your keyboard. Then, type **everyone [clickoff]** in the Command Center and press the **enter** button on your keyboard. What do you observe? Now, type **talkto [car1 car2 car3] fd 20** in the Command Center and press the **enter** button on your keyboard. What do you observe? Compare it with the **everyone [clickon]** command.
7. Now, let's consider programming a turtle to look like a dog running. In the Graphics Tab area, locate the two dog shapes. The first one is named **dog 1** and the second **dog 2**. Create a turtle, right-click on it and select edit. In the pop-up dialog box, name the turtle **dog** and in its instructions type **setsh "dog1 wait 5 setsh "dog2 wait 5** (where **setsh** is an abbreviation for **setshape**, either of the two commands works fine). Click the **OK** button of the dialog box and then click on the turtle. Use different numbers and explain your observations.
8. Combine steps 3 and 5 to program the dog to continuously change shapes and move forward.
9. Follow the instructions given in the Learning MicroWorlds Pro book on pages 73-75 (or the instructions given in the MicroWorlds Pro Tips and Tricks book on pages 17-20), and post your program of step 8 on the World Wide Web.

The question that is raised at this point is how can we program a turtle to move in different directions, move towards another object or turtle, follow a certain path etc. The command we are looking for is **towards**, which turns the current turtle towards a specified turtle or object (in case we want to know the distance between the two, we have to use the command **distance**, i.e., **t1 show distance "t2**). The next step includes examples of the **towards** command:

10. Create 3 turtles and spread them out on your page. Type **t1, towards "t2**, in the Command Center. What do you observe? How about, if you type **t1, towards "t3**? Now, name your t1 turtle dog (you can replace the turtle with one of the dog shapes in the Graphics, if you like). Type in the Command Center **everyone [ht] dog, st** (the **everyone [ht]** command hits all turtles on the page and **dog, st** allows dog to reappear on the page). Write the following procedure on the Procedures Tab area:

```
to follow
dog,
towards "t2
fd distance "t2 wait 5
towards "t3
fd distance "t3 wait 5
end
```

Finally, type in the Command Center **dog, follow** and press enter. Explain your observations. Does the program run without the **wait** command? Does the program run if we use **glide** instead of **fd**? What can you do to have a turtle to run around an irregular track?

## Activities

1. Create a table where you give the corresponding explanations of the following Microworlds Pro or Logo programs both before and after you try the suggested program variations (note that the programs that are marked CC must be typed in the Command Center area, and the PT programs must be typed in the Procedures Tab area – do not include the symbols CC or PT in your programs):

Programs	Program Variations
CC: <b>forward 50</b> <b>show heading</b>	Change the forward number Change the orientation of the turtle by using your mouse (see step 1)
CC: <b>show -4 + 10</b>	Change the series of the numbers. Change the signs of the numbers. Change the symbols to (/ or *)
CC: <b>show "science</b> (for this you have to open a text box on your page. Use the <b>A</b> button from your toolbar)	Change the series of the letters. Change the word.



<b>PT:</b> <b>to square :size</b> <b>repeat 4 [forward :size right 90</b> <b>wait 10]</b> <b>end</b> <b>CC:</b> <b>t1, square 100</b>	Vary all numbers, one at a time. Disregard the <b>wait</b> command.  <i>Note: Both CC and PT commands must be entered for the program to run.</i>
<b>CC:</b> <b>repeat 3 [print "science]</b> (you need to open a text box)	Change number and word.
<b>CC:</b> <b>make "message [print " science]</b> <b>repeat 3 :message</b>	Change number and word. Compare it with <b>repeat 3 [print "science]</b>
<b>CC:</b> <b>if ycor &gt; 100 [setc "blue]</b> (after you press enter, position the turtle at the top of your page and press enter again)	Change the number, the symbol from > to <, the name of the color.
<b>CC:</b> <b>forever [forward 1 wait 0.1] forever [ifelse ycor &gt; 0 [setc "green] [setc "blue]]</b>	Change the numbers (one at a time), the symbol from > to <, the name of the colors.
<b>CC:</b> <b>pd forward 50 right 55 forward 50</b> <b>CC:</b> <b>pd repeat 5 [forward 100 right 144]</b> <b>CC:</b> <b>pd repeat 360 [forward 1 right 1]</b>	Change the numbers (one at a time). Change <b>right</b> to <b>left</b> . Change <b>forward</b> to <b>back</b> . Disregard <b>pd</b> .  <i>Note: Run the programs separately. They are included together for comparison reasons.</i>
<b>CC:</b> <b>pd t1, arc 10 18</b> <b>PT:</b> <b>to arc 10 18 repeat 5 [forward 50 right 10] end</b>	Change the numbers (one at a time). Change <b>right</b> to <b>left</b> . Change <b>forward</b> to <b>back</b> . Disregard <b>pd</b> .  <i>Note: Both CC and PT commands must be entered for the program to run.</i>
Create a text box and write six words in it, each word at a separate line. <b>CC:</b> <b>show textcount "text1</b> <b>CC:</b> <b>show textitem 3 "text1</b> <b>CC:</b> <b>show textpick "text1</b>	Change the number from 3 to 1 in the <b>show textitem 3 "text1</b> .  <i>Note: Run the programs separately</i>
Create a text box. <b>CC:</b> <b>printdown "Lucent</b> <b>PT:</b> <b>to printdown : " Lucent</b> <b>if empty? : " Lucent [stop]</b> <b>print first : " Lucent</b> <b>printdown butfirst : " Lucent</b> <b>end</b>	Change the word.  <i>Note: Both CC and PT commands must be entered for the program to run.</i>
<b>CC:</b>	Change the numbers (one at a time).



<pre>to between? :7 :1 :12 output not or (:7 &lt; :1) (:7 &gt; :12) end PT: show between? 7 1 12</pre>	<p><i>Note: Both CC and PT commands must be entered for the program to run.</i></p>
--	---

- Use the **Map**, **Replace**, **Sort**, **Which**, **Intersect**, **Subset**, and **Union** commands in programs that show their function (Hint: See Help Menu's Programming: Useful Tools).
- Use the commands of activity 1 in programs that show their function. Try to combine different commands within one program.
- Use the commands of activity 1 to expand on the program you created in procedure step 8. Try to combine different commands within one program.
- Type the following program in the Command Center: **setshape [bird1 bird2] repeat 20 [fd 2 wait 2] repeat 20 [fd 10 wait 2] glide 100 5.**
  - Before running the program, predict what will happen.
  - Run the program and explain any discrepancies between your observations and your predictions.
  - What other command can you use to substitute **glide**? Can we use different commands and have the same result?
  - Write a program which shows the bird moving back and forth on the horizontal level of your page (Hint: read pages 49-50 from the MicroWorlds Pro Learning by Tom Lough).
- Create an animation that shows a square changing size. Repeat the same for a rectangle, triangle, and hexagon. (For the purposes of this activity the figures must be created by you. To create a new shape, double click on one of the empty shapes in the shapes palette, and then use the paint tools to design the shapes.)
- Repeat activity 2 but this time use a different color each time there is a change in size.
- Repeat step 9, by using the **dolist** command (use the information given in MicroWorlds Pro Tips and Tricks on page 35). Use this type of program to show a bee moving on the perimeter of different path shapes, such as, squares, triangles, hexagons, octagons etc. Set different times of waiting at the tips of the path shapes.
- Use stick figures of at least 5 consecutive frames showing simple animation of some sort (i.e., a woman running). Do the activity by using at first only one figure and then add a second one (the two figures should have some sort of interaction). To create a new shape, double click on one of the empty shapes in the shapes palette, and then use the paint tools to design the shapes (hint: use procedure steps 3 and 5).



10. Create your own movie! Use stick figures of at least 5 consecutive frames showing simple animation of some sort (i.e., a man playing with a ball). First, start a new page and draw your first figure. Second, select Duplicate Page from the Pages menu to create an identical copy of your already existing page. Third, change the graphics on your second page. Fourth, continue this process until you complete the whole set of animation frames you had in mind to create for your movie. In case you want to include sound in your project, you can either create your own melody (select  and follow the directions given in the help menu under the topic **MicroWorlds Objects: Melodies**) or record your own music or sounds (select  and follow the directions given in the help menu under the topic **Recording Sounds**). Finally, in the **Procedures Tab** area, type the following commands and run the program:

```
to flip
dolist [i pagelist] [getpage :i wait 5]
end
```

**Note:** The **dolist** command runs a list of instructions using a range of variable values. In this program, it creates a variable **i** and cycles through the **pagelist**, assigning each page name to **i** in turn. As **: i** assumes that value, **dolist** runs the list of instructions [**getpage :i wait 5**]. This displays each page in sequence (Lough, 2000).

## Part B: Interactive Simulations

### Procedure

After completing the introductory activities of Part A, we are now ready to proceed into creating interactive simulations. But, what is an interactive simulation? It is a virtual representation of a phenomenon, which can be studied through a series of observations as we change the variables that control its behavior. An example of such a simulation would be an incline that has a piece of metal sliding towards the ground and you have the possibility of changing its angle, the mass of the metal, the friction coefficient between the two objects, etc.

This part is of particular importance, because it not only challenges your programming skills in MicroWorld Pro, but also challenges your knowledge in the subject you are simulating. For the purposes of this project, the subject we will emphasize is physics (this does not mean that you can not create simulations for other subjects).

Let's say we want to create a simulation that shows relative velocity in one dimension. First, we have to select the content and concepts we want to introduce through the simulation, and then decide the best way to simulate them. In the case of relative velocities, we need to know the following:

- Velocity is a vector and is defined as the rate of change of displacement. It is not an absolute quantity but is measured relative to other objects. For example, when

you hear a car is traveling 90 km/h, you normally assume that it means a velocity relative to the road.

- To measure any object's velocity we must specify the coordinate system or reference frame in which the measurement is to be made. Usually the origin of the coordinate system is fixed in some other body. In the example with the car, the road was the other body. But, what about driving down the highway when another car passes you with a slightly higher velocity? Although both cars run rapidly down the road, the faster car appears to overtake you very slowly. This is because when two objects are traveling along the same line, the relative velocity is obtained simply by ordinary subtraction (assuming the speeds are small compared to the speed of light):

$$V_{AC} = V_{AB} - V_{BC}$$

Having in mind the aforementioned content and concepts, we are now ready to select the main features that our simulation must include:

- Since velocity is a vector and is defined as the rate of change of displacement, there must be a controlled movement of objects/subjects and an option for deciding which direction to follow (at least for one of the objects/subjects).
- There must be relative movement among the objects/subjects.
- There must be a specified coordinate system or reference frame in which the measurement is to be made.

Finally, we have to select the phenomenon/activity we will simulate to show all the aforementioned. This is again a very serious issue and you have to consider it very carefully, based on the following criteria:

- The phenomenon/activity must clearly show the content and concepts you want to simulate.
- The creation of the phenomenon/activity on your MicroWorlds Pro pages must be possible. It is suggested that you use already existing shapes, icons, pictures etc. (remember that you can import pictures in MicroWorlds Pro from other programs, the internet, etc.). Creating them from scratch is time consuming and sometimes impossible.
- Before starting, make sure you have a clear plan that includes a diagram of what you want to create, the program/vocabulary you will use, and the control equipment you will use (i.e., buttons, sliders, text boxes etc.)

Based on these criteria, some of the phenomena/activities that qualify for the purposes of this project are a person jogging on a running road (gym equipment), a boat moving on running water, and a person walking on a train or plane.

In this project we will create a program that shows a swan moving on running water. Both the velocities of the water and the swan will be controlled through sliders, as well as, the boat's direction (it will be programmed to move both in the same direction the water is moving and in the opposite direction).



**YSAP**<sup>®</sup>  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

Now that we have specified the desired parameters of our simulation, we are ready to proceed with the creation of the simulation:

1. Create the background of your simulation by importing pictures or shapes from the Graphics area (you can also import pictures or shapes from elsewhere, i.e., internet), or by using the available painting tools of Graphics. Include a sky with some clouds (so it can be distinguished from the river), a river (use darker blue than the blue you used for the sky), and a shore (use light brown for ground). An example of such a background could be the following:



2. Create a turtle and turn it to a swan (there is a swan shape in the Graphics area) and put it on the river, as follows



3. At this point we have a problem to solve. The way our background is designed, it is impossible for us to show the flow of the water. We cannot program the blue color of the river to flow, since it is not a turtle. Thus, we have to come up with an idea that will show the flow of the water. This can be done, for example, with a piece of wood (i.e., a log) that floats on the surface of the river and is carried along by the water (we assume that the wood will have the velocity of the stream). The piece of wood that will be added to our simulation must be a turtle, thus create one and turn it into a piece of wood. Our picture, now, changes to



4. Name the swan, debbie, and the piece of wood, wood.
5. Create two sliders. Type in the first slider dialog box, swan, and set the minimum value to zero and the maximum value to 6. Type in the second slider dialog box, river, and set the minimum value to zero and the maximum value to 4.
6. Create two buttons. Name the first one **turn** and the second one **start** (in both cases set the button to Many times).
7. At this point your page should look as follows,



8. Write the following program (only the part that is under the program's column, not the part in explanation's column) in the Procedures Tab area:

Program	Explanation
<pre>to go wood, fd river debbie, if heading = 90 [ down_stream ] if heading = 270 [up_stream] end</pre>	<p>It sets the wood to go forward in the river and defines for debbie what is considered to be downstream (towards left) and what is considered to be upstream (towards right).</p>
<pre>to down_stream setsh "swan1 fd swan + river end</pre>	<p>It sets debbie's shape as it moves downstream and defines the total velocity as the sum of the individual velocities of the swan and the river. "swan1 refers to the shape of the swan that faces towards left. You can find the shape under Graphics.</p>
<pre>to up_stream setsh "swan2 fd swan - river end</pre>	<p>It sets debbie's shape as it moves upstream and defines the total velocity as the difference of the individual velocities of the swan and the river. "swan2 refers to the shape of the swan that faces towards the right. You can find the shape under Graphics.</p>
<pre>to turn debbie, ifelse shape =</pre>	<p>It refers to the function of the <b>turn</b> button. Each time you press it, it turns 180 degrees</p>



<pre> “swan2 [setsh “swan1][setsh “swan2] rt 180 end </pre>	and faces in the opposite direction.
<pre> to startup set "button2 "on? "true end </pre>	It refers to the function of the <b>start</b> button. Each time you press it, the program starts running.

- Run the program, by clicking the **start** button. Vary the velocities of the swan and the river. What do you observe? What is happening when the swan is moving opposite the flow of the river and it has the same velocity (magnitude) as the river? What do you observe when you press the turn button?

## Activities

- Create a series of simulations where you have a car (or any other type of vehicle) accelerating, decelerating, and moving with constant velocity.
- Combine the accelerating, decelerating, and constant velocity vehicle in one simulation.
- Create a simulation to show the phenomenon of buoyancy . Assume that you have 3 objects of different densities (much less than that of water) at the bottom of a glass filled with water. Show how they will behave when you release them. Make the density of each object changeable by using sliders, but always less than that of the water.
- Repeat activity 3, but this time allow the density of the objects to become bigger than that of water, as well. (Hint: make use of the **if** and **if else** statements.)
- Create a similar simulation to the simulation of part B to show the same phenomenon. This time use a different scenario (i.e., a man walking on a ship).
- Expand the simulation of part B by including a third object moving in the river (i.e., a boat). Follow the same procedure we followed in part B.
- Change the ranges of the sliders and explain what will happen if we set their minimum value below zero.
- Create a simulation where you have a car with some initial velocity sliding on the road and stopping due to friction. (Hint: Use a program similar to the one in part B.)
- Create a simulation combining part A and part B. Use the three dolphin shapes given in the Graphics area, which means the dolphin will be changing shape as it moves.



**YSAP**<sup>®</sup>  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

## **Final Project/Report**

Create a series of physics simulations. It does not have to be in kinematics. You can select any physics area you like. Include a report explaining the reasoning behind your selection (see part B) and an analytical step by step description of your programs.

## **References**

Lough, T (2000). *Learning MicroWorlds Pro*. Logo Computer Systems Inc.

Stager, G (2000). *MicroWorlds Pro Tips and Tricks*. Logo Computer Systems Inc.

## **Unit 3: Computer Programming in Science Education – MicroWorlds Pro**

### **Project 16**

## **Create Your Own Computer Games!**

MicroWorlds Pro is a multimedia programming environment that offers you the possibility to create dynamic projects. You can use it to enhance your understanding of MicroWorlds and to get a real sense of the depth and breadth of this powerful multimedia programming environment (MicroWorlds Pro, 2000). Among other things, MicroWorlds Pro allows you to create animations and simulations, which you can easily turn into Games! Games are interactive projects and thus the process of their creation will help you learn more about programming and project development.

### **Degree of Difficulty**

Experimental: Moderate

Conceptual: Moderate

### **Prerequisite Knowledge**

This project is suggested for students that are already familiar with MicroWorlds Pro and Logo programming. In case you are not, please read the introductory part of Project 15 (“Create Your Own Simulations”) and complete all the activities of Part A before starting this project.

### **Objectives**

Completion of the activities should enable you:

- to learn how to create animations and simulations by using MicroWorlds Pro.
- to use MicroWorlds Pro as a tool to create interactive games.
- to post your games on the World Wide Web.

**Materials:** computer, MicroWorlds Pro (visit <http://www.microworlds.com/solutions/mwpro.html> for purchasing the software), the Learning MicroWorlds Pro and the MicroWorlds Pro Tips and Tricks books that are included in the package of MicroWorlds Pro.

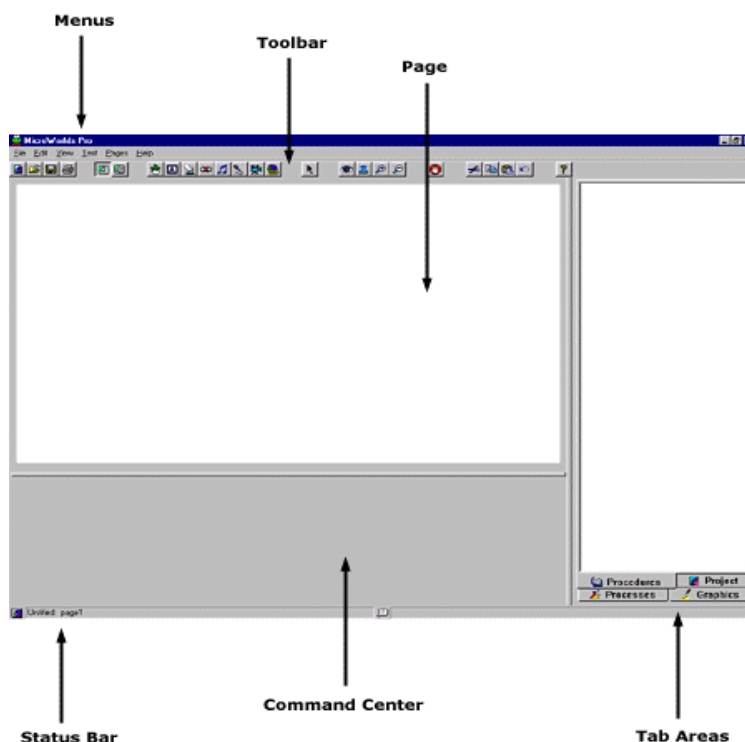
**Useful Information:** The Right Click on your PC’s mouse is used to open pop-up menus on objects; the same functionality is obtained on a Mac by Control Click. The PC’s Shift key is used to add more than one shape to a turtle and to change the heading of the turtle; the same functionality is obtained on a Mac with the Command key.

Do not forget to save your work regularly! A good idea is to save them as a series of project files, each at a different stage of development.



Each MicroWorlds Pro (MWP) project consists of objects and text presented on separate pages. For every project you start, there is a corresponding single page. A page is accompanied by Menus, Toolbar, Status Bar, Command Center, and Tab Areas (see figure 2).

**Figure 2**



It is also important to know about Logo's punctuation and marker meanings. According to MicroWorlds Pro (2000):

Punctuation and Markers	Meaning
" Quotation mark	Indicates that the next word is to be taken literally.
[ ] Square brackets	Surround a list.
: Colon	Sometimes referred to as dots. Indicates that the next word is the name of a variable. Refers to contents of the container.
( ) Parentheses	Group things in ways that Logo ordinarily would not or varies the number of inputs for a primitive.
Vertical bars	Surround a long word so you can include spaces.

## Procedure

After completing the introductory activities of Project 15, we are now ready to proceed into creating interactive games. Developing an interactive project like this one will help you learn about the power and the potential of computer programming. As Lough (2000) mentions, “you will be able to initiate and manage complex actions, and have some creative control at the same time.”

The developmental process of your game is also important to learn, because it not only challenges your programming skills in MicroWorld Pro, but also your knowledge about the game. It is important to know the rules of the game, otherwise you will not be able to make a fully functional game. A suggestion would be to outline the rules of your game, if it is a game that already exists, or define the rules of your game, if it is a game that you are creating, and try to match them with corresponding commands or programs before you start creating your MicroWorlds Pro page. After you finish, begin combining the programs that control the rules of your game into one program.

Let's say you want to create a game called “Help the dog”, in which a dog has to move across highways without being hit by a vehicle. Since it is a new game that you are creating, a set of rules must be defined. For example, the rules of this game might be

- The dog must be able to move in any direction.
- There must be more than two lanes of vehicles passing through.
- The distance among the vehicles will not be the same.
- All vehicles in one will move in the same direction and with the same speed.
- There must be an option for changing the magnitude of the speed so the game can become easier or harder.
- There must be some sort of notifications that the dog has been hit by a vehicle (i.e., a crash sound) or that it has reached its destination.

Having in mind the aforementioned rules, we are now ready to select the main features that our game must include:

- There will be a dog that can move forwards, backwards, to the right, and to the left by using buttons.
- There will be four rows of vehicles (cars, motorcycles, trucks, etc.). Each row's vehicles will be moving at the same direction and speed (the speed will be the same for all vehicles, but it will be changeable - it will be controlled through a slider). Some of the other rows' vehicles will have different direction (opposite) and speed.
- The distance among the vehicles will be fixed but not the same. However, there must be enough space in every row of vehicles that allows the dog to pass through.
- There will be a notification that the dog was hit and a notification that you have completed the task of the game.

Finally, we have to decide what exactly the plot/background of the game will be. This is again a very serious issue and you have to consider it very carefully, based on the following criteria:

- The plot/background must clearly show the figure of your game. Avoid using in your background the colors that the dog shape consists of.
- The creation of the plot/background on your MicroWorlds Pro pages must be feasible. It is suggested that you use already existing shapes, icons, pictures, etc. (remember that you can import pictures in MicroWorlds Pro from other programs, the internet, etc.). Creating them from scratch is usually time consuming and sometimes impossible. It will be helpful if you have a drawing before starting.

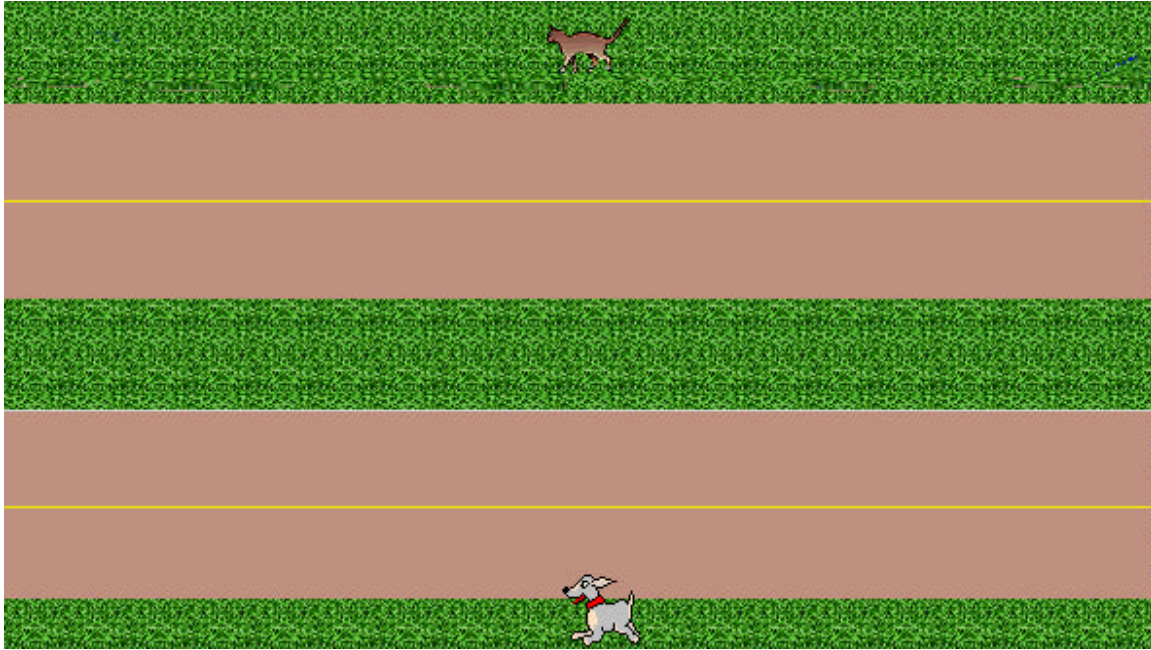
Before starting, always, make sure you have a clear plan that includes a diagram of what you want to create or import, the program/vocabulary you will use, and the control equipment you will use (i.e., buttons, sliders, text boxes, etc.). After you specify all these desired parameters of your game, you can proceed with its creation.


For this project let's consider developing the "Helping the dog" game described above (the idea for this game is based upon an already existing game, named *frogger*, developed by LCSi). The whole developmental procedure of the game is described in the following steps

1. Create the background of your game by importing pictures or shapes from the Graphics area (you can also import pictures or shapes from elsewhere, i.e., internet), or by using the available painting tools of Graphics. Include two highways, and grass lanes. An example of such a background could be the following



2. Create two turtles, turn the first one to a dog and the other one into a cat (you will find the shapes in the Graphics area), and position them, as follows



3. Copy the cat's image onto the background graphics. To freeze the cat at a certain place, select the stamper  and click on a turtle. Stamping a turtle copies its image onto the background graphics.
4. Create two landmarks at the bottom of your page to define the starting point. Use for example the blue ball from the Graphics area. Of course, you can use any of the shapes that are available to you, just make sure that when you will be creating your page's background these shapes do not overlap with your dog's shape or your vehicles' shapes. In case they overlap and you really want to include a particular shape, MicroWorlds pro offers you the option to minimize or maximize shapes. For example, the maximizing button of the Toolbar can have the following effect on a turtle



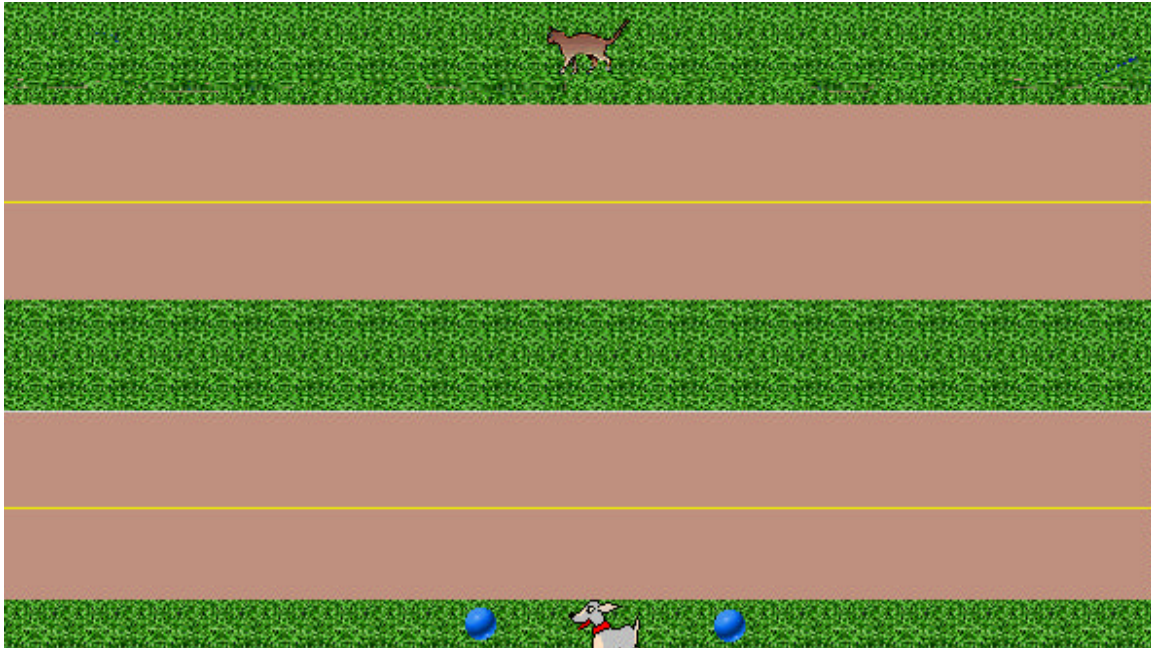
In case you use the blue balls to define your starting point, place them at the bottom of your page as follows (the distance between the two balls is not of particular importance; it depends on your game rules and more specifically how difficult you want to make your game)





YSAP®  
WWW.YSAP.ORG


YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM



5. Create 10 turtles. Place five of them facing towards right and five of them facing towards left. You can orient the turtles towards right by using the command `seth 90` and towards left by using the command `seth 270` (see project 15, Part A, step 2 for more details). If you do not give them an orientation, they will move in the direction of their heading and this will mess up your game. Turn them into vehicles, and place them on the highway as follows



6. Copy the already existing dog shape from the Graphics area, paste it to an empty shape area in the Graphics, and name it `deaddog` (the purpose of this is to have a

shape that shows what happens to the dog in case it gets hit by a car). Double click on your copied shape and from the pop-up dialog box click on the upside down flip option , and then on **OK**. Check your Graphics area to see that your copied shape is transformed to an upside down dog shape. Thus, in the scenario that the dog is hit by a car your page will look like



7. Name all of your turtles. Use the following names (start from the top of your page and move from left to right). In the pop-up dialog boxes, for all 10 vehicles, include in the instructions area the commands **fd carspeed wait 1**, and for the cat include in the instructions area the commands **if touching? "zach" vasso [zach, wait 2 win setpos [5 -150]]**, (see task 7 for more explanatory information). Do not include any commands in the dialog box of the dog.

Shape	Name
cat	vasso
motorcycle	motor
blue car	car1
red car	car2
purple car	car3
yellow car	car4
red jeep	car5
red car	car6
yellow car	car7
orange tractor	car8
red car	car9
dead dog	deaddog
dog	zach







YSAP®  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

8. Create a slider. Type in its dialog box, **carspeed**, and set the minimum value to zero and the maximum value to 8.
9. Create five buttons. Name the first one **up**, the second one **down**, the third one **left.** (make sure to include the dot after left because the button's name must be different from the logo command **left**), fourth one **right.**, and fifth **start** (in all cases set the button to Once).
10. At this point your page should look similar to the following page,



11. Record three small pieces of sound or create three small pieces of melody (about 3-10 seconds each). You will need one introductory piece for starting the game (save it as **startmusic.wav**), a second piece for the vehicles hitting the dog (save it as **deaddog.wav**), and a third piece for winning the game (save it as **yeah.wav**). In case you want to create your own melody, select  and follow the directions given in the help menu under the topic **MicroWorlds Objects: Melodies**. In case you want to record your own music or sounds, select  and follow the directions given in the help menu under the topic **Recording Sounds**. Both procedures are extremely easy, so try to use both (create a melody for one piece and record a sound for the other pieces). Note that after you create your sound files, you have to import them into your program (go under file, choose import, and then select sound). Check in the Project Tab area if the sound files were imported. If not, try again.

12. Now that the background, the turtles, the buttons, the slider, and the music files are set, it is time to start programming the game. The best way to do this is to break down our program in smaller parts and develop programs for each part separately. At the end you can combine all of them and create your game's program. It is suggested to control your turtles' actions with procedures where possible instead of placing commands into dialog boxes (Lough, 2000). During the development of your project, you will typically make several changes. It is usually easier and faster to edit procedures in the Procedures Tab area than to change commands in dialog boxes one at a time. This is especially important for this particular project because it includes a large number of turtles (Lough, 2000). The following tasks are the programming steps you have to follow to create the program of this game (the explanation notes are included for you to understand the reason behind selecting the particular commands):

***Task 1: Program the start button and all the related activities.***

In this introductory part of the program, you have to include the commands that control all the associated activities that relate to starting the game, such as, playing the **startmusic.wav** music file, positioning the dog at its starting point, making all vehicles start moving with controlled speeds based upon parts of a program you will develop for this purpose (in the introductory part you only name/introduce those other parts of the programs that control the behavior of your turtles i.e., **chkspeed** or **forever [check]**).

**to start**  
**startmusic.wav**  
**chkspeed**  
**everyone [clickon]**  
**dog, setpos [5 -150]**  
**forever [check]**  
**end**

Explanatory notes: This part of the program

- sets the game to start with music from the file **startmusic.wav** (as soon as you click on the start button of your game, the music will start playing),
- turns all the turtles on (**everyone [clickon]**)
- indicates that the turtles will have a checked speed (**chkspeed**). Thus, a program must follow specifying how the turtles' speed will be controlled. However, there must be another program explaining when the turtles must be checked.
- sets the dog to start from the point (5, -150). Note that the point (0,0) is the center of your page.
- indicates that all the turtles will be always under control (**forever [check]**) based upon a program named **check** that will be specifying how the turtles will be controlled.



### ***Task 2: Define `chkspeed`.***

It was mentioned in the introductory part (task 1) that the command **`chkspeed`** (or **`checkspeed`**) must be defined in order for the MicroWorlds software to understand what functions to perform when it reads **`chkspeed`**. Thus, you have to include a part in your program that specifies how the turtles' speed will be controlled.

**to `chkspeed`**

**`car5, setspeed carspeed + 2 car8, setspeed carspeed car7, setspeed carspeed car9,`  
**`setspeed carspeed car6, setspeed carspeed + 2 car3, setspeed carspeed + 2 car4,`  
**`setspeed carspeed + 2 car2, setspeed carspeed motor, setspeed carspeed car1,`  
**`setspeed carspeed`  
**end**********

Explanatory notes: This part of the program

- controls the turtles' speed (it is the definition of the program we were referring to, in task 1, with the command **`chkspeed`**)
- sets the cars' speed to be controlled by the slider named **`carspeed`**. For example the commands **`car8, setspeed carspeed`** mean that `car8`'s speed is set and controlled by the slider `carspeed`. Notice that for the vehicles of the second and third row we added a factor of 2 that makes those vehicles move slightly faster than the vehicles of the first and fourth row. The purpose is to make the game a little bit harder and more exciting. You can always exclude the +2 factor or change it (make it smaller or bigger).

### ***Task 3: Program the up, down, right, and left movement buttons***

One of the rules we set at the beginning, when we were designing the game, was to offer the possibility to the dog to move in four directions (up, down, left, right). The easiest way to include this particular feature in the game is to create interactive buttons. Each button must execute a series of commands, which in this particular case must be commands that define the direction and a specific range of distance the dog will move.

**to up**

**`zach, seth 0 fd 47`**

**end**

**to down**

**if `xcor = 5` [stop]**

**`zach, seth 0 bk 47`**

**end**

**to right.**

**`zach, seth 90 fd 25`**

**end**

**to left.**

**zach, seth 90 bk 25  
end**

Explanatory notes: This part of the program

- sets how the dog must behave when the up, down, right, or left button is clicked. For example, **to left. zach, seth 90 bk 25 end** gives directions to the dog to turn 90 degrees and move backwards 25 units.
- forbids the dog to move backwards when it is on the x coordinate 5 (**if xcor = 5 [stop]**). This part was added, once again, to make the game harder. In case you move the dog upwards from the already set starting point (5, -150), you want be able to move it backwards, therefore you will have only three options to avoid the oncoming vehicles (up, left, or right). However, you can always move it first right or left, and then have all the options available. If you want to make the game more fun, include a number of similar restrictions in this part of the program.

#### ***Task 4: Define Check***

It was mentioned in the introductory part (task 1) that, the command **forever [check]** must be defined in order for the MicroWorlds software to understand when the functions included in the part of the program named **check** must be performed. As the word **forever** indicates, the program **check** will always be applied. Now, you have to decide what to include in the **check** part of your program that you want regulated all of the time. For this project, we only need the speed of the cars to be controlled all the time. As for the rest of our turtles, the dog will be controlled by the buttons and the cat will be held fixed, therefore, none of these two have to be included in the **check** program.

**to check**

**car5, setspeed carspeed + 2 car8, setspeed carspeed car7, setspeed carspeed car9,  
setspeed carspeed car6, setspeed carspeed + 2 car3, setspeed carspeed + 2 car4,  
setspeed carspeed + 2 car2, setspeed carspeed motor, setspeed carspeed car1,  
setspeed carspeed**

Explanatory notes: This part of the program

- defines the duration that the turtles' speed will be controlled (it is the definition of the program we were referring to, in task 1, with the command **forever [check]**)
- sets the cars' speed to be forever controlled by the slider named **carspeed**.

### ***Task 5: Program dog's reaction when hit by a car***

One of the game's rules is for it to end when a vehicle hits the dog. Consequently, a program that defines the dog's behavior in the scenario that is being hit by a vehicle must be included.

```

if touching? "zach "car1 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car2 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car3 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car4 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car5 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car6 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car7 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car8 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car9 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car10 [zach, setsh "deaddog wait 2 dead setpos [5 -150]
setsh 1]
end

```

Explanatory notes: This part of the program

- defines a series of conditions that involves touching “zach” the dog. The conditions mention that if zach is touched by a vehicle (**touching? "zach "car1**), he will turn its image to deaddog's upside down image (**zach, setsh "deaddog**) and after 2 seconds a program called dead will be activated (**wait 2 dead**). In addition, zach will go back to its already fixed starting point (5, -150), after the **dead** program has completed its functions, turning at the same time back to its initial upright image (we used number 1 for defining its initial upright image, even though it is named zach, because the program does not allow using zach twice within the brackets [ ]). This implies, though, that the shape of the dog must be placed at the very first spot in the Graphics area, or you can simply substitute 1 with the number of the spot the image is already in. (Click on your image once and you will see its position number in the Status bar).

### ***Task 6: Define dead***


The command **dead** that was included in the conditional statements of the previous part (task 5) must be defined in order for the MicroWorlds software to understand what functions to perform when it reads **dead**.

```
to dead
announce [Game over. Try again.]
deaddog.wav
end
```

Explanatory notes: This part of the program

- defines the functions that will be performed when the dead program starts running (it is the definition of the program we were referring to, in task 5, with the command **dead**)
- programs the game to pop up a dialog box announcing that the game is over and to try again, and play the sound file **deaddog.wav**.

### ***Task 7: Program finishing the game with a win***

In step 7, you were told to include in the cat's instructions area the commands **if touching? "zach "vasso [zach, wait 2 win setpos [5 -150]]**. These commands, basically, initialize the win program (**win**) and then make the dog (zach) return to its starting point (**setpos [5 -150]**). The **win** program includes an announcement that informs you about your win (a pop-up dialog box will show you that **You Won!** – click on **OK** or  to continue with a new game), and a sound file (**yeah.wav**) that plays cheerful sounds or music indicating your win.

```
to win
announce [You Won!]
yeah.wav
end
```

Explanatory notes: This part of the program

- defines the functions that will be performed when the win program starts running (it is the definition of the program we were referring to, in the cat's instructions area, with the command **win**)
- programs the game to pop up a dialog box announcing that “You Won!”, and play, at the same time, the sound file **yeah.wav**.

13. Put all the parts of your program together, as follows

```
to start
startmusic.wav
```

```
chkspeed
everyone [clickon]
dog, setpos [5 -150]
forever [check]
end
```

```
to chkspeed
car5, setspeed carspeed + 2 car8, setspeed carspeed car7, setspeed carspeed car9,
setspeed carspeed car6, setspeed carspeed + 2 car3, setspeed carspeed + 2 car4,
setspeed carspeed + 2 car2, setspeed carspeed motor, setspeed carspeed car1,
setspeed carspeed
end
```

```
to up
zach, seth 0 fd 47
end
to down
if xcor = 5 [stop]
zach, seth 0 bk 47
end
to right.
zach, seth 90 fd 25
end
to left.
zach, seth 90 bk 25
end
```

```
to check
car5, setspeed carspeed + 2 car8, setspeed carspeed car7, setspeed carspeed car9,
setspeed carspeed car6, setspeed carspeed + 2 car3, setspeed carspeed + 2 car4,
setspeed carspeed + 2 car2, setspeed carspeed motor, setspeed carspeed car1,
setspeed carspeed
```


```
if touching? "zach "car1 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car2 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car3 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car4 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car5 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car6 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
```

```
if touching? " zach " car7 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car8 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car9 [zach, setsh "deaddog wait 2 dead setpos [5 -150] setsh
1]
if touching? " zach " car10 [zach, setsh "deaddog wait 2 dead setpos [5 -150]
setsh 1]
end
```

```
to dead
announce [Game over. Try again.]
deaddog.wav
end
```

```
to win
announce [You Won!]
cheers.wav
end
```

Copy the program and paste it into the Procedures Tab area.

14. Run the program, by clicking the **start** button. Vary the speed of the vehicles. What do you observe? In case you want to quit playing, use the  button of the Toolbar.

## Activities

1. Find all the drawbacks regarding the design of the game described above and revise the program to avoid them (hint: if you move the dog left or right and then move it backwards, what happens?)
2. Include more restrictions to make the game more difficult. Make sure, though, that passing the dog across the highways is possible.
3. Include a counter in your game to keep track of your wins and losses (hint: use the information given in the help menu for counters).
4. Create a similar game, but this time, have the obstacles (i.e., stones) moving vertically and the main character (i.e., a dragon) moving sideways.
5. Create a game that presents your main character (i.e., a soldier) jumping over moving obstacles. Use similar rules as in the above game.
6. Create the musical game SIMON. Follow the instructions given in Lough's book, *Learning MicroWorlds Pro* (p. 51).

7. Follow a similar procedure, as in activity 6, and create a memory game that involves science knowledge. Instead of matching colors and music, as SIMON does, match questions with answers.
8. Expand the game described above by having the dog return back to its starting point in order to win.
9. Change the range of the slider and explain what will happen if you set the minimum value below zero. Define the range that you can use for playing the game.
10. Follow the instructions given in the Learning MicroWorlds Pro book (p. 73-75), and post one of your games on the World Wide Web (in case you do not have access to a server, run it on your hard disk).

## **Final Project/Report**

Create a series of games that involve teaching a science topic/concept. Include a report explaining the reasoning behind your selection and an analytical step by step description of your games.

## **References**

- Lough, T (2000). *Learning MicroWorlds Pro*. Logo Computer Systems Inc.
- Stager, G (2000). *MicroWorlds Pro Tips and Tricks*. Logo Computer Systems Inc.

### **Unit 3: Computer Programming in Science Education – Visual Basic Project 17**

## **Create Your Own Windows Processes System Messages: An Introduction to Visual Basic**

This project contains steps that help you to learn about the Visual Basic (VB) programming language and offers you guidance to create a computer-based interactive game. Visual Basic is a unique language because of its different interface, its different style, and its different method of doing things. Unlike other languages, Visual Basic is completely graphically oriented, so you'll be learning a lot about how to control the program to be subject to the whim of the user, and you'll learn how Windows processes system messages so things happen. Visual Basic is also a fun and practical language because coding is minimized with its easy to use graphical interface. Its primary purpose is to create custom databases, but it is fully functional for creating games, applications, diagnostics, modem terminals, ... you name it! Virtually any Windows program can be created with Visual Basic, and you'll find it's the easiest language out there for the power it has.

### **Degree of Difficulty**

Experimental: Moderate to Difficult

Conceptual: Moderate to Difficult

### **Prerequisite Knowledge**

Experience using a Windows operating system is required. A good understanding of procedures and modular programming will be invaluable in learning Visual Basic. A good knowledge of BASIC programming will also help. Experience using DOS is recommended. Strongly suggested for students with experience in other programming languages, such as, C/C<sup>++</sup>, Fortran and Pascal.

**NOTE:** This project refers to Microsoft Visual Basic 6. However, the programs also apply to earlier versions of Visual Basic. If you have a different version of Visual Basic, you may notice some minor differences. If you have any questions, ask your teacher or mentor for help.

### **Useful Information/Concepts**

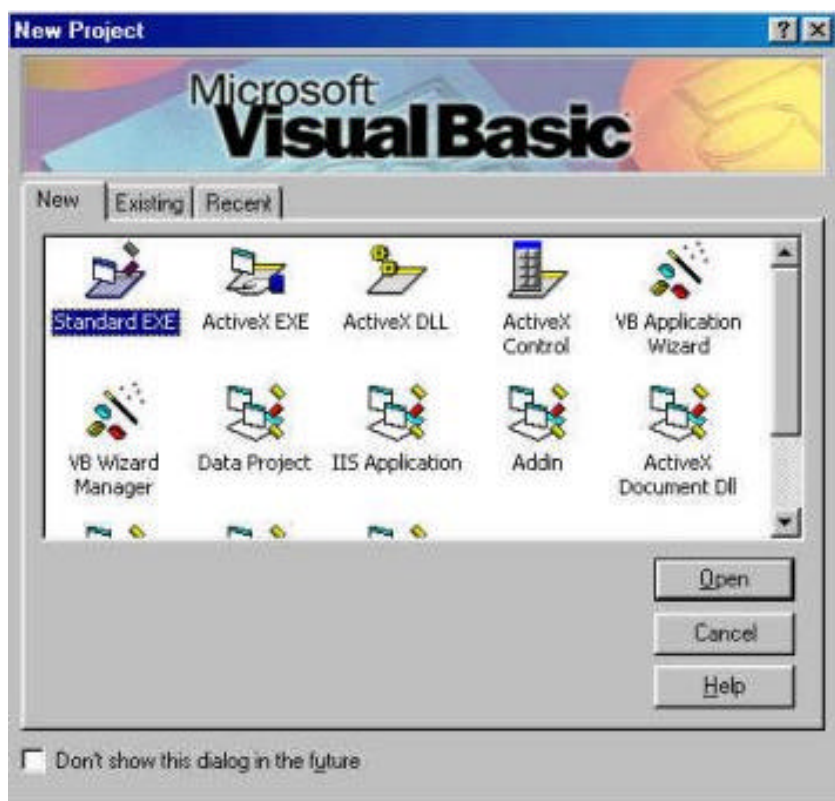
- Created in 1991
- Easy to create visual interface  
set of visual objects – controls  
can drag and drop controls



- Object-oriented language  
Object.Method parameter-list
- Event-driven programming  
blocks of code (event procedures)  
programmer must ensure prerequisites
- Loading Visual Basic (If you do not have the VB software installed on your computer, see the **Materials** section here below for information regarding the purchase of it).

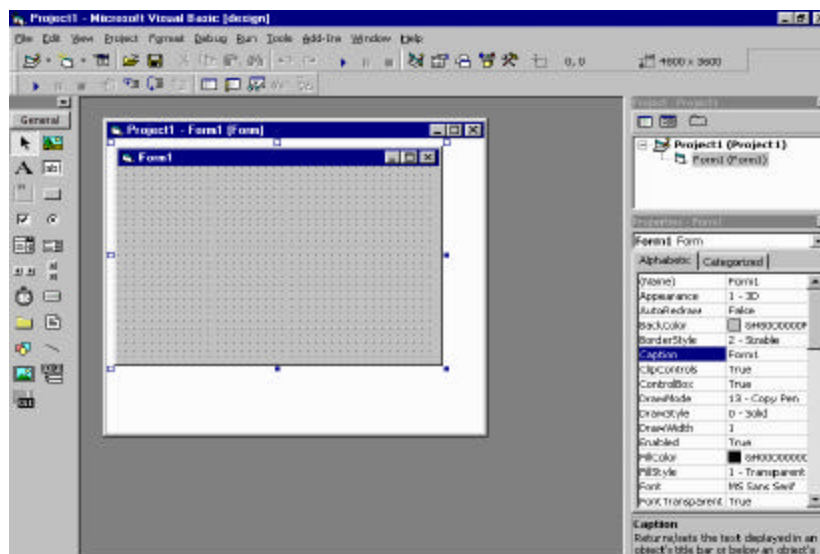
To start Visual Basic: (1) Click the Start button, (2) Point to Programs, (3) Point to Microsoft Visual Studio (or Microsoft Visual Basic if Visual Studio is not visible), (4) Click Visual Basic 6.0. Visual Basic will now be loaded and you will be presented with a window entitled New Project (see figure 1).

**Figure 1**



There are a number of icons in this window. Each icon represents a different type of program (or Project as they are called in Visual Basic) you can create. Highlight "Standard EXE" and click Open. A mini-screen with a title of "Form1" should appear (see figure 2).

**Figure 2**



(Visit <http://www.shep.net/cts/cp20/Intro.htm> for descriptive information regarding the Visual Basic desktop and the project development area. Get to know and understand the parts of the Visual Basic development environment because you will be using it all the time throughout this project.)

However, the most important windows are:

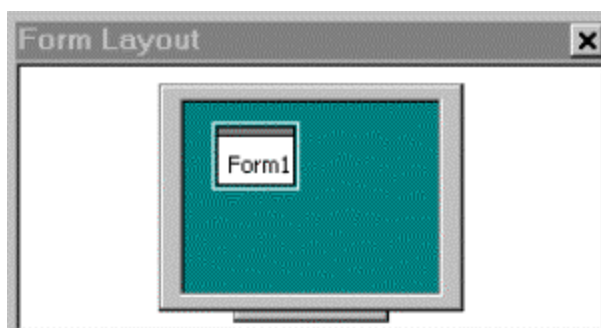
1. **Toolbox:** all your VB controls are contained on the toolbar. You use the buttons on the toolbar to draw controls on the form. When you add a VB, its icon will appear here (visit <http://206.45.2.126/compsci/CPR30S/cs30snotes/Unit1/page1.htm> for more information).

**Figure 3**



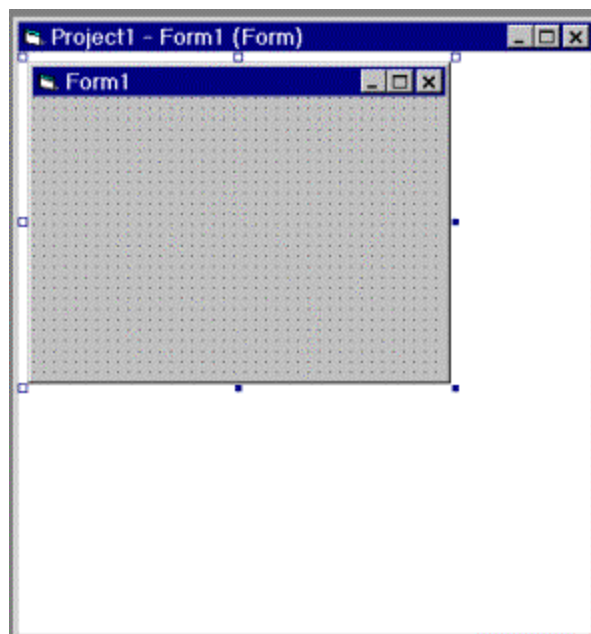
2. **Form:** this is what the user sees. It contains controls, and its actions are defined by the code. Most applications have around 5 - 6 forms, but there are obviously exceptions (visit <http://206.45.2.126/compsci/CPR30S/cs30snotes/Unit1/page1.htm> for more information).

**Figure 4**



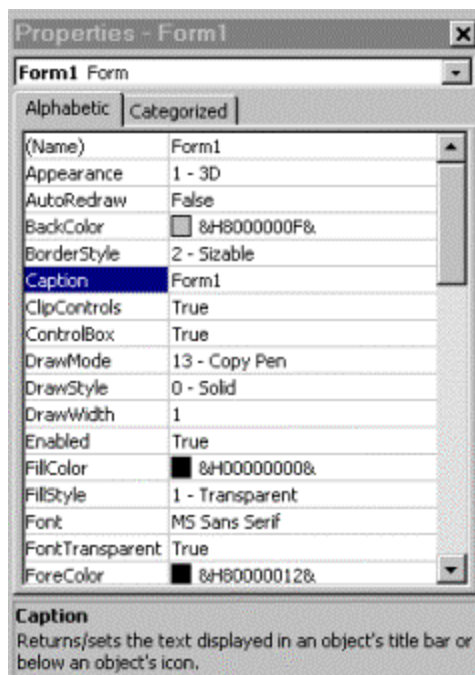
3. **Code Module:** the code for the controls is defined in the code module. Notice the two combo boxes at the top of the code module. The left one shows the object whose code is being defined. The right one shows the Procedure, or event of the control which is being defined (visit <http://206.45.2.126/compsci/CPR30S/cs30snotes/Unit1/page1.htm> for more information).

**Figure 5**



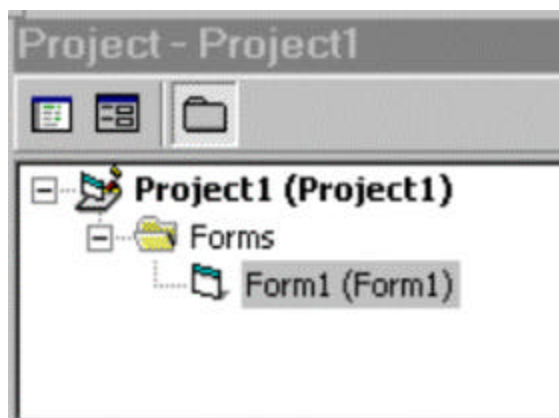
4. **Properties Window:** properties for a control are set in this window, such as the control's color, dimensions, and operations (visit <http://206.45.2.126/compsci/CPR30S/cs30snotes/Unit1/page1.htm> for more information).

**Figure 6**



5. **Project Window:** this is a representation of the .mak file, which defines which files are to be used in the program. Every form, base code module, and vb used with the .mak file is shown in this window (visit <http://206.45.2.126/compsci/CPR30S/cs30snotes/Unit1/page1.htm> for more information).

**Figure 7**



- **Saving Your Project:** Click the green disk on the toolbar. The Save File As box appears. Locate the place on your computer or network where you would like to save your project (create a new folder for each project you create, as projects contain a lot of files). Click the Create New Folder button, and then open that folder.
- **Opening a New Project:** If you want to make a new project when you already have another open, you must close the other one first. To do this, save your current project by clicking the green disk on the toolbar (as above), click File, Remove Project. To start a new Project, click File, New, and double click Standard EXE.
- **Object Orientated Programing:** this means programming using objects. An object could be anything from a button, to a browser engine. The code you write will be based around what happens to the objects in your project, and through using code you can control/get info from objects in a project (VBHQ, 2000).
- **Object Properties:** Each object on a form has its own properties which can be set (Design-time and Run-time) or “got” (Run-time only). Each object has its “top” and “left” properties which tells VB where to place it on the screen in relation to its “container”. A container is simply a bigger object which holds the object selected. For example, a button's container is usually its form. A form's container is usually the screen. If a button did not have a container, whenever you moved the form, the button would not move (VBHQ, 2000).

In VB code, object properties are used to set or get in the following format:

**[ObjectName].[Property]**

All objects in Visual Basic have “names”. This is how objects are distinguished from each other. This also means no two objects on one form can have the same name, otherwise Visual Basic would not know which object you were trying to program with (VBHQ, 2000).

As was mentioned before, properties can either be “set” or “got”. Here's an example of both

```
Label1.caption = "Lucent"  
Variable1 = Label1.caption  
Print Variable1
```

The first line tells VB to change the **caption** property of **label: Label1** to **Lucent**. A label is a piece of text on a form, the caption part being the only thing you ever see. The second line copies the caption property of the same label into a variable called **Variable1**. The final line prints the contents of the variable **Variable1** on a form (VBHQ, 2000).

- Structure of VB Application can contain several different types of files
  1. Project file (.VBP) - information on forms and other resources.
  2. Form modules (.FRM) - information on the controls and code on a form.
  3. Standard (.BAS) and class (.CLS) modules - contain Basic code (no GUI).
  4. Custom controls (.VBX or .OCX) - include specialized controls.
- Design-time is the period when you're designing your Form or writing code – basically any time your program isn't running.
- Runtime, as its name implies, is exactly the opposite of design-time - the time when your program is running.

## Objectives

Completion of the activities should enable you to:

- Gain a basic understanding of programming theory, including: effective problem solving techniques, software development processes
- Develop interfaces using Visual Basic controls
- Understand events in Windows and learn how to write corresponding event handlers
- Create applications containing variables and subroutines
- Learn how to open text data files for input and output
- Use programming constructs to handle the flow of control within an application

**Materials:** computer, Microsoft Visual Basic 6.0 software (standard or professional version - visit <http://msdn.microsoft.com/vbasic/prodinfo/purchase/pricing.asp> for purchasing the software).

## Part A

### Introduction: Your first VB applications

When programming, it is much easier if you follow accepted procedures. A program should have:

- functionality – meets the standards
- user-friendliness
- mobility around user interface
- consistency in appearance & behavior
- user supports (lookups / online help)
- flexibility
- robustness to protect user against mistakes



- consistency in coding style
- code clarity & readability
- modularity in code design
- code maintainability

For this reason, you will follow a step-by-step process as you create your first few programs:

- Analyze and define the problem
- Design the visual interface
- Define user-program interaction (properties of the objects)
- Design the code structure
- Write code
- Test and edit the program

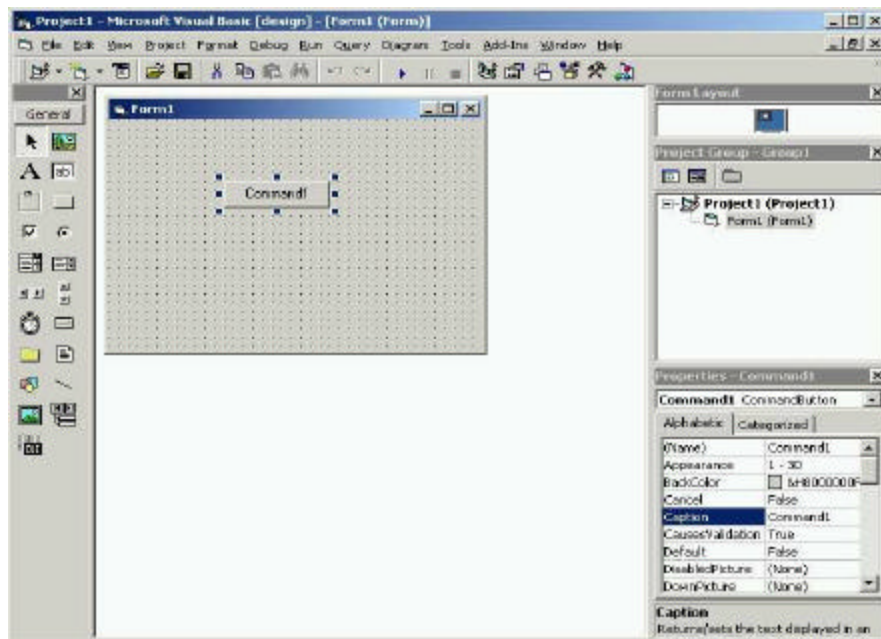
Follow the next steps to create your first application:

## Procedure

1. Create a new project, maximize form window to fill all of VB workspace (as described above, double click Standard EXE and then click on the Open button - this creates an empty project for you to base your program on).
2. To the left of your screen, you should see a lot of little icons under the heading “General” (these are called components or ActiveX controls). Double-click on the component represented by . An item called **Label1** should appear in the center of **Form1**.
3. To the bottom-right of your screen, you should have a box entitled **Properties - Label1**. The items you see here are “properties” of the label. They tell it how to work (i.e., what text to display, what color it should be, etc.). Click on the box to the right of the **Caption** property. Replace the existing **Label1** value with “My First Visual Basic Program.” Press Enter. What do you observe? Now, click on the label and hold your mouse button down. Drag your mouse to the upper left hand corner of **Form1** and release the mouse button. What do you observe?
4. Double click on the button  in the “General” toolbar. This is a Command Button. Now move the mouse to the form, and the cursor should change to a cross (+). Click and hold down the mouse button, and drag it right and down. You should be making a rectangle on the form. Let go of the mouse, and you should have your button. Change its **Caption** property to “Click”. Resize the Command Button using the eight blue squares surrounding it. Your project should look as follows.



**Figure 8**

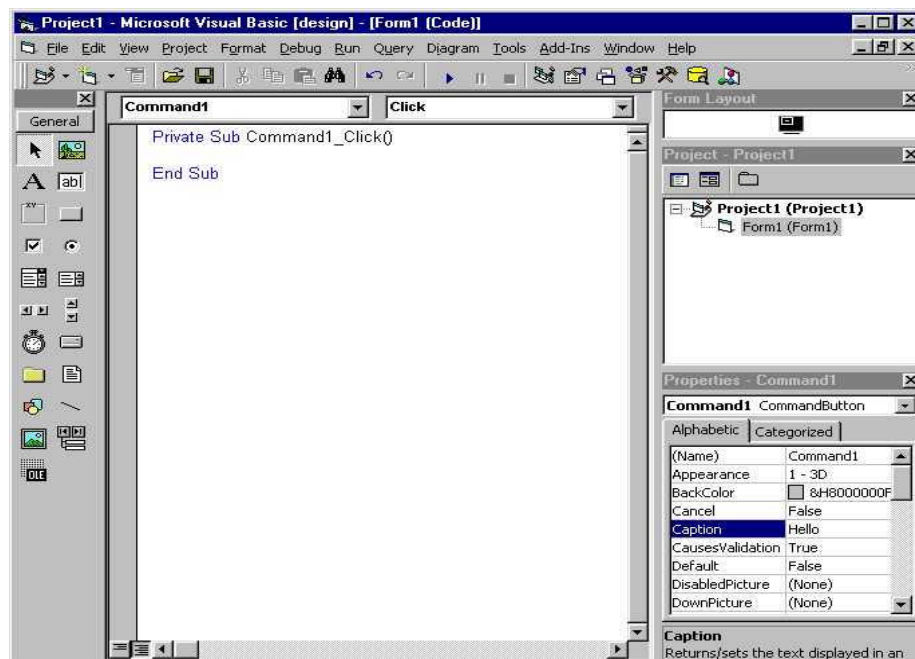


Double click on the Command Button, once again. That should bring up the code screen. Type in

**Private Sub Command1\_Click()**

**End Sub**

**Figure 9**





As soon as you do this, the cursor will start flashing:

```
Private Sub Command1_Click()  
<Cursor flashing here>  
End Sub
```

Type in

```
MsgBox " My First Visual Basic Program."
```

Thus, your program should be

```
Private Sub Command1_Click()  
MsgBox " My First Visual Basic Program."  
End Sub
```

But, what do all these commands mean? The **Private Sub Command1\_Click()** tells Visual Basic to run this line of code when someone clicks on the Command Button called **Command1**. When you do, the line of code you typed in runs - telling Visual Basic to display the designated sentence in a message box (**MsgBox**). The **End Sub** tells VB to stop running the code. In other words, Visual Basic code is just a set of instructions that tell Visual Basic what to do.

5. Click on the (Start) button on the toolbar. Your application should start running. Click on "Click" button. What do you observe?
6. Right click the Command Button you have just created. Click Delete. What do you observe?
7. Insert a command button on the form (as shown above). Make sure the command button is selected (i.e., it has eight small squares surrounding its perimeter). The default caption the command button has is clearly Command1. This is also shown in the Properties window next to Caption. Select the box to the right of Caption and change its value from Command1 to **Click**, then press enter. What do you observe? Each object has its own unique name. This is represented by the (name) property in the Properties window. This property allows us to assign meaningful names to objects. The unique name allows us to distinguish an object from other objects, as we will see later. Using a meaningful name helps us remember what the object does, without looking at its code. Try changing the caption of the form itself. Remember to select it first!
8. Create a new project. Save the project in a new folder (as described earlier). Place a command button on the form. Give the command button the name **My\_VB\_Project** and the caption **Welcome!** Give the form the caption My VB Program. Double click the command button. A new window will open. This window contains all the code for

your project. You will see Visual Basic has created some code automatically to help you:

```
Private Sub My_VB_Project_Click()  
End Sub
```

The first line of code tells Visual Basic that the lines of code between it and End Sub are to be executed when the button with the name **My\_VB\_Project** is clicked. As you can see, there is no code to be executed when the button is clicked. Enter **:Print "Welcome to my VB Page!"**, like this:

```
Private Sub My_VB_Program_Click()  
:Print "Welcome to my VB Project!"  
End Sub
```

Run the program by clicking the start button. What do you observe? How does this compare to step 4? How about using **Print** instead of **:Print**? What is the role of the “:”?

Now, view the interface window again (click View, Object). Double click a blank area on the form. You will see that your code is displayed, along with some code that Visual Basic has generated automatically for you:

```
Private Sub My_First_Button_Click()  
:Print " Welcome to my VB Project!"  
End Sub
```

```
Private Sub Form_Load()
```

```
End Sub
```

As you might have guessed, any code you enter in the Form\_load sub is executed when the program starts up! Try adding **:Print "Welcome to my VB Project!"** between Private Sub Form\_Load() and End Sub, i.e.:


```
Private Sub My_First_Button_Click()  
:Print "Welcome to my VB Project!"  
End Sub
```

```
Private Sub Form_Load()  
:Print "Welcome to my VB Project!"  
End Sub
```

Run the program. What do you observe? How does this compare to step 4? Try clicking the button, what do you observe?


In steps 1-5, we built a program with just two simple controls (a control is just an item you can add to your Form, e.g., a Command Button). But this was just the beginning. There are many controls available to the users, and getting familiar with them will make your programming with VB much easier:


**Picture** - This control is like a form within a form. Its primary use is to display pictures, as the name implies, but it can also be used to contain other objects within a group like a frame, or to display messages. The print method can be used on the picture box control to display text, and other controls can simply be dragged into the box at design time to make them part of the picture box (Aitkenl, 2000).

**Image** -  - The image tool is primarily used for displaying images. You can load a picture into the image control by setting the **.Picture** property of the image control at design time. You have to use the **LoadPicture()** function. The code for loading a picture into an image control called **imgPicture** looks like this:

**imgPicture.Picture = LoadPicture(filename),**


where filename represents any valid .bmp, .rle, or .wmf file (Aitkenl, 2000).


**Label** -  - You often use this control to display information to the user, such as an explanation of what they should do. The user cannot change the contents of a label unless you provide the means necessary to do so. Text is displayed in a label control through its **.caption** property. The **.caption** property can be set at run time or design time. You can also experiment with the other properties like **.borderStyle** to make a border around the label, and **.backStyle** to make the background of the label transparent. Another useful property is the **.WordWrap** property. If the length of the text you want to assign to the **.Caption** property exceeds the width of the label, then the words are placed on the next line (Aitkenl, 2000).


**TextBox** -  - The primary purpose of the text box is to provide a method for the user to type information into the program. A text box can be a one line input box, or it can be several lines long, like a word processor form. The **.text** property of the text box determines its text, which can be set at design time or run time (by the program OR the user). There are four interesting properties that are affiliated with the text box:

- **.MaxLength**: For text input, you can put a limit on the number of characters the user can type into the text box with this property.
- **.Multiline**: If the **.Multiline** property is set to true, then when the user presses enter, the cursor will go to the next line, like a word processor.
- **.PasswordChar**: If you've ever seen a password input box, you'll notice that the password the user types isn't printed in regular characters for the whole world to see; rather, asterisks are displayed for each character the user types. If you are doing a password input box, the **.PasswordChar** property can be useful since it sets what character appears when the user types (Aitkenl, 2000).
- **.ScrollBars**: There are 4 possible values which can be set for this control:


- 0 - None
- 1 - Horizontal
- 2 - Vertical
- 4 - Both


**Command Button** -  - Command buttons are clickable options in windows like OK, CANCEL, RETRY, ABORT, FAIL, etc. When you click the button, the **\_Click** event is generated for the control. The **.caption** property determines what is displayed on the button. This is kind of a cool trick that applies to any control: when you put & in front of a letter in the **.caption** property, that letter is underlined. An underlined character means that the user can press ALT and that letter to click that control (or in most cases, just set the focus to it), (Aitkenl, 2000).


**Drive List Box, Directory List Box, and File List Box** -  - A drive box selects the drive you want to view the files on, a directory list box lists the directories on that hard drive, and a file list box lists the files in the currently selected directory (usually changed in the directory list box).


**Shape** -  - The Shape tool provides a method of making simple geometrical shapes on a form simply by drawing them and setting the **.Shape** property. A shape can be a rectangle, a square, a circle, an oval, a rounded rectangle, or a rounded square, depending on the value of the **.Shape** property. There are a few properties worth noting:

- **.BackColor**: Sets the background of the object
- **.BackStyle**: Sets the background style of the shape
- **.BorderColor**: Sets the color of the border around the shape
- **.BorderStyle**: Sets the style of the border.
- **.BorderWidth**: Sets the width of the border of the shape, in pixels
- **.DrawMode**: Selects how the shape will be drawn
- **.FillColor**: Defines the color the object will be filled with
- **.FillStyle**: The fill of the shape is different from the background of the shape in that the fill is placed over the background

**Check Box** -  - Check boxes are options that can either be checked, unchecked, or grayed. The **.caption** property defines what you are checking.


**Timer** -  - It is very simple and straightforward, and it is extremely useful. When you want code to be executed many times in regular intervals, an option is to use a timer control. All you have to do is draw the control on the form you want to use it on, set the **.interval** property, and add code to the **\_Timer** event. The **.interval** property defines how often the code in the **\_Timer** event will be executed, and is measured in milliseconds (Aitkenl, 2000).

**Combo Box** -  - Here's the first advanced control we're covering. Combo boxes are dropdown list boxes with a text entry box at the top. There are three styles of combo boxes: 0 - Dropdown Combo, 1 - Simple Combo, 2 - Dropdown List. A dropdown combo box displays a list of items when you click the arrow on the right. There is a text box at the top which allows the user to type in stuff, and possibly add an item to the list if that's what the programmer allows (Aitkenl, 2000).


**Horizontal and Vertical Scroll Bars** -  - A scroll bar is a very common windows element. It is the thing on the right and bottom of a Word screen (for instance). It lets you move things into and out of the window. In VB, scroll bars are also used to give input on a scale and at the same time, give the user output on a scale. This scale is manipulated by either clicking an arrow on the scroll bar, clicking the elevator, or by moving the handle. The most common use of these controls in Visual Basic is to input data on a scale. The most necessary value of the scroll bar is the **.Value** property. There is a numerical value for every position the handle has on the scroll bar. The value is changed by defining the **.LargeChange** and **.SmallChange** properties, and bounded by the **.Min** and **.Max** properties.


These are pretty much the most important controls you have to know for the purposes of this project. There are some more that were not included, which you can always look up yourselves. Now it is time to test some of these controls by running some applications:

9. Open Visual Basic and create a new Standard Exe. To the left of your screen should be the control toolbox, displaying a standard set of Visual Basic controls. Double

click on the  button. This is the Check Box control. It should now appear at the center of your Form. Take a glance at the Properties window. These are all the Check Box properties you can change. Don't worry about all the properties. Some are hardly ever used or only required in very specific circumstances. The most important are altering the Caption, Forecolor and Font properties. Go ahead and randomly change a few properties and observe how they affect the control. Make a table describing your observations for each of the properties.

10. Open Visual Basic and create a new Standard Exe. Name your program **Speed**. To the left of your screen should be the control toolbox, displaying a standard set of

Visual Basic controls. Double click on the  button. This is the Vertical Scroll Bar control. It should appear on your Form. Take a glance at the Properties window. These are all the Check Box properties you can change. Set the Minimum property to 0 and the maximum property to 80. To represent the value 40, set the Value property to 40. The thumb will be positioned in the middle of the scroll bar. The position of the thumb and the number stored in the Value property always correspond to each other.

Double-click on the component represented by . An item called **Label1** should appear in the center of **Form1**. The label on the Speed Program's window displays the value represented by the scroll bar. You can display this value in a label by setting the

label's Caption property to the Value property of the scroll bar: **lblSpeed.Caption = vsbSpeed.Value** (remember that the three letter prefix for vertical scroll bars is vsb). In fact, to display the new value stored in the scroll bar's Value property, double click on the Command Button, once again and type in

```
Sub vsbSpeed_Change( )  
lblSpeed.Caption = vsbSpeed.Value  
End Sub
```

What do you observe, when the user drags the thumb back and forth using the mouse? Explain what exactly the commands of the program mean.

11. Try playing with the rest of the control buttons. Make a table describing your observations for each of the properties.

Every single control you add to a Form has a Name property, which appears at the very top of the Properties window. Every time you add a control, an automatic name is assigned – such as Command1. Every time you add a control, rename it with a more meaningful name. For instance, if you have a Text Box that will hold a year, call it "Year".

The Visual Basic community also has its own "naming conventions" where, for instance, Text Box names are preceded by "txt". For instance, if you have a Text Box that will hold a date, the convention is to change its name to "txtYear". Other prefixes include:

<b>Prefix</b>	<b>Used For</b>
btn	CommandButton
cbo	ComboBox
chk	CheckBox
dir	DirectoryList
drv	DriveList
fil	FileList
fra	Frame
frm	Form
grd	Grid
hsb	HorizontalScrollBar
img	Image
lbl	Label
lin	Line
lst	ListBox
mnu	Menu
opt	OptionButton
pic	PictureBox
shp	Shape
tmr	Timer
txt	TextBox

vsb	VerticalScrollBar
-----	-------------------

Note: It's advisable to keep names unique (i.e., txtScience, tmrSpace etc.)

How about changing things in code? You've already seen how you can change the properties of controls while you are in design-time. But what if you want to change such properties while the program is running? For instance, perhaps you want the Caption of your Form to change depending on which button the user clicks.

12. Open Visual Basic and create a Standard Exe. Change the Name of Form1 to **“frmProject”** and the Caption to **“My Project App!”** Add two Command Buttons and change the Caption of the first to **“Hi”** and the second to **“Information.”** Don't forget that you can add any control to your Form, including Command Buttons, by clicking once on the item in the toolbox, and then dragging it out on your Form. You can easily change the properties of such controls by altering values in the properties window. Change the Font property and Name properties of your buttons to something sensible, such as **“cmdHi”** and **“cmdInformation”**.
13. Double click on the Hi button. The Visual Basic code window should appear, looking something like this

```
Private Sub cmdHi_Click()
```

```
End Sub
```

Your cursor should be flashing between the two lines. Type the following:

```
frmProject.Caption = "You clicked Hi – Welcome to my project!"
```

Redisplay the Form by double-clicking on **frmProject** in the Project Explorer window to the top-right of your screen. Double click on the Information button and type in the following code:

```
frmProject.Caption = "You clicked Information – This is an introduction to the purpose of my project..."
```

Now, if you click the “Hi button”, the following code will run when the Click event of the cmdHi control is activated – in other words, whenever you click **cmdHi**, this code will run

```
Private Sub cmdHi_Click()
```

```
frmProject.Caption = "You clicked Hi – Welcome to my project!"
```

```
End Sub
```



You've seen the first and third lines before. The second line simply changes the property of an object during runtime. In this instance, the Caption property of **frmProject** is changed to **"You clicked Hi – Welcome to my project!"**. That basic syntax can be used to change virtually all properties. It goes something like this:

**ObjectName.PropertyName = Value**

If the value is a number or true/false, don't bother enclosing it in quotation marks. For all other values however, use quotes " ".

Try running your application and clicking on the Hi and Information buttons. See what happens? Play around with setting properties in code. Explore the list of available properties that pop up after entering the period following the object name. Try experimenting with the following

**frmProject.BackColor = vbBlue**

**cmdHi.Caption = "Click Hi?"**

**cmdInformation.ToolTipText = "Unavailable at present time. Sorry!"**

Note: Why isn't **vbBlue** surrounded by "quotes"? In Visual Basic, the word **vbBlue** actually represents the 'number' for blue. It's known as a constant and exists so you don't have to remember about five-digit numbers (every color has a corresponding five digit number). Try playing with other colors.

## Activities

1. Create some labels and get their captions to change when you click on the button. Or you could change other properties for the existing objects (Hint: You can see what properties are available to set by looking at the 'Properties' window in design mode. It's to the right hand side of the screen by default, and looks like a list with two columns in it. Each entry in the list is a property, and a property value.).
2. Create a game by using the information provided above. Try to be creative. An example would be to present the players with a story/mystery, give them some hints, and then have them select from a series of buttons you have already created with possible answers to your story (a multiple choice test!). Make sure that you have hidden answers for each one of your buttons (i.e., "Wrong answer. Please Try again.", "You won!").
3. Create a game testing your knowledge in a particular subject area (i.e., science, history, mathematics, etc.). If you want to get fancier, include an option for the player to select the subject area first, and then have him/her proceed with the quiz.
4. Try playing with all the given control buttons. Make a table describing your observations for each of their properties. (Hint: You can see what properties are

available to set by looking at the 'Properties' window in design mode. It's to the right hand side of the screen by default, and looks like a list with two columns in it. Each entry in the list is a property, and a property value.).

5. Repeat step 13, but this time create three buttons. Play around with their setting properties in code. Explore the list of available properties that pop up after entering the period following the object name. Make a table that presents the properties you have experimented with and include corresponding explanations.

## **Part B: User Input/Variable Types**

In any application, there is nearly always going to be some kind of user input. The type of input you are expecting from the user is important, if not your application may not function correctly, because of what you may need to use the data for. Therefore, it is important to introduce at this point various response/feedback types you can to receive from a user, and the methods of getting them (VBHQ, 2000):

Types of Responses	
<b>String</b>	The most commonly used type of input. This can be ANY character on the keyboard, and for any length. For example, to log into an application, the user enters their username (a string) into the system. The application then makes a decision based on the string data - their username - whether to continue the login procedure.
<b>Number*</b>	Can be plus/minus, integer/decimal (Any number of decimal places). For example, an application asks the user how many times they wish to print a report. The user enters 3 (a number), and the application loops the print process by whatever number was entered.
<b>Boolean</b>	Boolean responses have two values, True or False. A good example is a 'Save' dialogue box. The message is "Would you like to Save you Work?", the response is either Yes or No, True or False.
<b>Multiple choice</b> (Single Option)	One option can be selected from a group of many. Example: "Select your Age Range:". The user can only select 1 because they are only one age.
<b>Multiple Choice</b> (Multiple Options)	Many options can be selected from a group of many. Example: "Select your interests for a survey:", a list of possible interests is supplied. Because the user may have more than one interest, there should be the option to reply with many options.
<b>Event Driven</b> (User activates project events)	You can make application decisions based on what the user actually does within the application itself. The best example is a computer game.

\* **NOTE:** A Number is NOT a string. A number has a mathematical value, a string does not. For instance, “123”, “A45”, and “Z8G6” are strings, whereas 123 is a number, as is one hundred twenty three. The symbols that make up a number (0123456789) are called numerals.

Now that you know about the types of responses that could occur within a given application, let’s consider an example where the user has to use the Number type.

1. Start a new project. Add the following components (click the appropriate icons, and draw them on the form) and change the following properties:

Object Type	Property	Value
Form	Caption	Unit converter
Command Button	Name	Command1
	Caption	Convert feet
Label	Name	Label1
	BorderStyle	1
	Caption	0
Label	Name	Label2
	Caption	Length in meters
Label	Name	Label3
	Caption	Length in feet
Label	Name	Label4
	BorderStyle	0
	Caption	1

2. Once you have all the objects listed above on the form with the correct properties, resize the labels and the button to look tidy on the screen (i.e., resize the label length to the size of the text).
3. Now that you have the application outlined, you can write the code to make it work. In design-mode, double-click the command button to bring up the code editor for that button. The default event to write code for is **Click**, so you won't have to change that. Type in the following code:

```
Dim LengthM  
Dim LengthF  
InputTitle = "Feet"
```

```

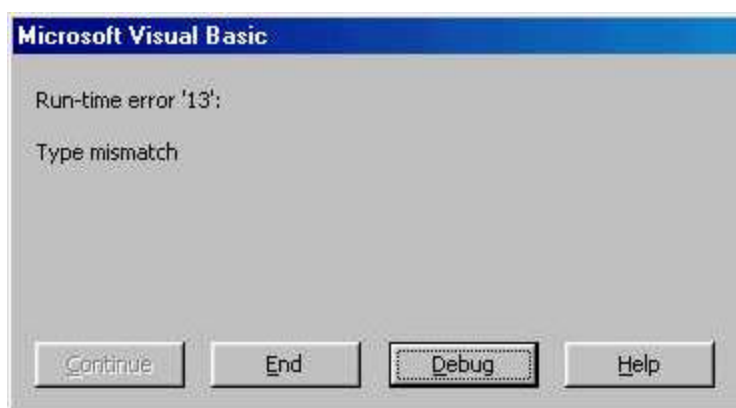
InputText = "Enter Feet Starting Value"
LengthF = InputBox(InputText, InputTitle)
LengthM = LengthF * 3.281
Label4.Caption = LengthF
Label1.Caption = LengthM

```

Code	Explanation
<b>Dim LengthM</b>	Declare the variable <b>LengthM</b> .
<b>Dim LengthF</b>	Declare the variable <b>LengthF</b> .
<b>InputTitle = "Feet"</b>	Puts the text <b>"Feet"</b> into a variable <b>InputTitle</b> .
<b>InputText = "Enter Feet Starting Value"</b>	Puts the text <b>"Enter Feet Starting Value"</b> into a variable <b>InputText</b> .
<b>LengthF = InputBox(InputText, InputTitle)</b>	Brings up an input box with the text as the variable <b>InputText</b> , and the input box title as <b>InputTitle</b> . Whatever the user types into the input box is inserted into the variable <b>LengthF</b> .
<b>LengthM = (LengthF * 3.281)</b>	Assigns the output of the conversion ( <b>LengthF * 3.281</b> ) to the variable <b>LengthM</b> .
<b>Label4.Caption = LengthF</b>	Updates the relevant label caption with the the variable <b>LengthF</b> .
<b>Label1.Caption = LengthM</b>	Updates the relevant label caption with the the variable <b>LengthM</b> .

If you run the application now, type in a valid number, press OK, and it will convert it. However, if you type in an invalid entry, for example anything with a letter in it, you will get the message below, and the program will die (VBHQ, 2000).

**Figure 10**



4. The reason for this pop-up dialog box is that you tried to insert a non-numeric value into an numeric-only variable. (VB is only expecting a number – any number - because of the command **Dim LengthM** you used) To overcome this problem, replace the code for command1 with the following:

```
Dim LengthM
Dim LengthF
InputTitle = "Feet"
On Error GoTo 100
InputText = "Enter Feet Starting Value"
LengthF = InputBox(InputText, InputTitle)
LengthM = LengthF* 3.281
Label4.Caption = LengthF
Label1.Caption = LengthM
Exit Sub
100
Msgbox ("Got Error " & Type Mismatch)
```

The explanations for the extra lines are as follows

Code	Explanation
<b>On Error GoTo 100</b>	If there is an error in the application then go to label <b>100</b>
<b>Exit Sub</b>	Stop processing code in this section
<b>100</b>	Label <b>100</b> in the code
<b>Msgbox ("Got Error " &amp; Err.Description)</b>	Display message box with the text: <b>Got Error &amp; Type Mismatch</b>

Now, if the user enters an invalid entry into the input box, the following message box appears

**Figure 11**



5. In the case that someone wants to keep track of how many times you clicked a particular button or how many times you made a mistake, you have to make use of the program **Dim CountClick as Integer** (or in general **Dim VariableName as DataType**). This program creates in the General Declarations section of your code window a variable called **CountClick** to hold a value of Integer type. Create a Command Button. From the code window, select **cmdVariableCount** from the object

drop-down box and ensure the procedure box reads **Click** (VBHQ, 2000). Visual Basic should create the following code boundaries for this particular item

**Private Sub cmdVariableCount\_Click()**

**End Sub**

6. Type in the following:

**CountClick = CountClick + 1**

**Msgbox "You've clicked me " & CountClick & " times!"**

Code	Explanation
<b>CountClick = CountClick + 1</b>	Sets CountClick to equal whatever it is at that moment, plus one (when we first start the program, it equals zero).
<b>Msgbox "You've clicked me " &amp; CountClick &amp; " times!"</b>	Displays a simple message box saying <b>You've clicked me</b> , then uses the & character to stick in the value of the <b>CountClick</b> variable, and then uses the & character once more to add <b>times!</b> to the message.

Note: The ampersand (&) character just sticks bits of the message together (VBHQ, 2000).

7. Run your program and click on the button a few times. What do you observe?

Now let's consider including a mini-security program. This is useful if you want to allow only particular individuals to use your program.

8. Open Visual Basic and create a new Standard Exe. Change the Form1 Name property to **"frmLogon"** and its caption to **"Enter your Password"**. Create a text box and change its Name property to **"txtPassword"**, its **PasswordChar** property to an asterisk "\*" and remove everything from its Text property (Moore, 2000).
9. Add a command button to the form. Change its Name property to **"cmdLogon"** and its Caption to **"Logon"**.
10. Declare a variable to hold the number of times a user has attempted to enter a password. Open the code window and move to the **Declarations** section. For instance, you could double click on a blank area of the form and then select **General** from the object drop-down box and **Declarations** from the procedure drop-down box (Moore, 2000). Declare as follows:

### Dim LoginCount as Integer

Now in the Click procedure of **cmdLogon**, enter the following code.

```
LoginCount = LoginCount + 1
```

```
If LoginCount > 3 Then
```

```
MsgBox "There is something wrong with your password!"
```

```
Exit Sub
```

```
End If
```

```
If txtPassword.Text = "Zach" Then
```

```
MsgBox "Welcome!"
```

```
Else
```

```
MsgBox "Password incorrect!"
```

```
End If
```

Code	Explanation
<b>LoginCount = LoginCount + 1</b>	Sets <b>LoginCount</b> to equal whatever it is at that moment, plus one (when we first start the program, it equals zero).
<b>If LoginCount &gt; 2 Then</b> <b>MsgBox "There is something wrong with your password!"</b> <b>Exit Sub</b> <b>End If</b>	This is a "conditional logic". If the LoginCount variable is greater than two - in other words, if they've attempted to log on more than two times - then tell them "There is something wrong with your password!" Exit Sub just stops the code running right there, ignoring the rest of your procedure.
<b>If txtPassword.Text = "Zach" Then</b> <b>MsgBox "Welcome!"</b> <b>Else</b> <b>MsgBox "Password incorrect!"</b> <b>End If</b>	This is another example of conditional logic. It checks to see if the textbox text is equal to "Zach". If it is, it congratulates you - otherwise it tells you "Password incorrect!"

11. Run the application. Does it work?



## Activities

1. Follow the information given on <http://161.58.186.98/beginning/vbtutorial4/index2.html> and <http://161.58.186.98/beginning/vbtutorial4/index3.html>, and explain how you can create a "loop" in code and how to get information from the user.
2. Create a form and add a command button ("cmdHobbiesEntry") and list box ("lstHobbies"). In the code behind the command button, prompt the user "How many hobbies do you want to add?" – and then display that number of input boxes to accept hobby names, adding each to the list box.
3. Use an **if** statement to compare the value of an integer called "price" against the value 20, and if it is more, print the text string "Expensive".
4. Create a VB program which categorizes products according to their weight. Use as categories the words "Extremely Heavy", "Very Heavy", "Heavy", "Normal", "Light", and "Very Light". You are responsible for defining the corresponding weight ranges. Compare the value of an integer called "weight" against the weight ranges, and print the corresponding characterization (i.e., Heavy, Normal, Light etc.).
5. Create VB programs that perform the following tasks:
  - Converts temperature from Fahrenheit to Kelvin and Celsius.
  - Converts pounds to kilograms
  - Converts miles to kilometers
  - Calculates the area of various shapes (i.e., squares, rectangles, triangles, hexagons, circles, etc.)
  - Calculates the volume of various shapes (i.e., spheres, pyramids, cylinders, etc.)
  - Calculates mathematical operations (i.e., addition, multiplication, division, etc.)

Try combining programs together on the same form (i.e., calculating the area of various shapes and the volume of various shapes).

6. Assume that you were hired by a realtor's office to create a VB program which calculates the total area of houses. Include a program that categorizes the houses according to the area they occupy.
6. Create a game by using the information provided in Part B. Try to be creative. An example would be to test their knowledge in a particular subject area.
7. Create a questionnaire and perform a research study on a subject that you are interested in (i.e., what are the best studying procedures). Include all types of

responses presented at the beginning of Part B (i.e., string, number, multiple choice etc.).

8. Repeat activity 7, but this time assume that you want only your classmates to serve as your survey's sample. Can you think of a way to do this based on the information given in Part B?
9. Create a multiple choice quiz that tests the knowledge of your sample in a particular subject area, and based on the responses create charts that show the percentages of those selections. (Hint: See step 6 of Part B for how to collect your data. Use spreadsheets to enter your data and to create charts).
10. For students who want to get more in depth, visit the following websites:  
<http://www.cyber-matrix.com/vb.htm>, and <http://www.pgacon.com/visualbasic.htm>.

## **Final Project/Report**

Create a VB program that will be of use for a science project. Include a report explaining the reasoning behind your selection (purpose, benefits from its operation, etc.) and an analytical step by step description of your program. Describe the science project, and how your program is useful for it.

## **References**

Aitkenl, P. (2000). *Visual Basic*. URL: <http://www.pgacon.com/visualbasic.htm>

Moore, K. (2000). *Visual Basic Tutorial*.  
URL: <http://161.58.186.98/beginning/vbtutorial/>

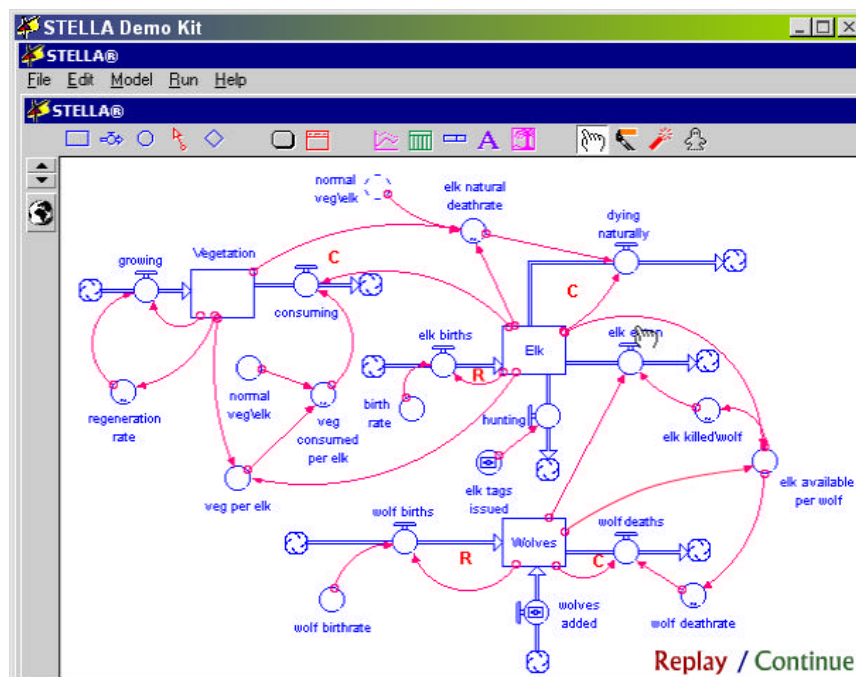
VBHQ (2000). *Visual Basic/Programming Introduction*. URL:  
<http://www.learnvb.f2s.com/tutorials/vbintro.htm>

## Unit 3: Computer Programming in Science Education – STELLA Project 18

### Learn how to create simulations by using a mapping language

Compared to the other programming languages/software introduced so far, STELLA differs in a unique way. It is a mapping language (see figure 1) that involves a more hands-on, “discovery oriented” format. In other words, it is a program tool that supports learner-directed learning. It allows you to unfurl a bit of your model and then to simulate just that bit to add the associated “behavioral” dimension. Drop down a little more structure...then simulate to see what it has added to the dynamics (HPS, 2000). By simulating “just what's showing”, you can build up your understanding of the relationship between structure and dynamic behavior in a systematic fashion - and the pace of the building up is completely under your control (HPS, 2000). Its primary purpose is to simulate real applications, activities or problems in areas such as physics, chemistry, biology and environmental science, but it is fully functional for creating concept maps, games, applications, diagnostics, graphs... you name it!

**Figure 1**



### Degree of Difficulty

Experimental: Moderate to Difficult  
Conceptual: Moderate to Difficult

## Prerequisite Knowledge


Experience using a Windows operating system is required. Students must have some experience in basic computer functions (e.g., access programs, open programs, save data), and have knowledge of basic computer programs (e.g., word processing programs, spreadsheets etc.).

For programming in STELLA, there is no need of knowing any particular programming language. It requires only writing words or equations very much in the same way we do in word processing or spreadsheet programs.

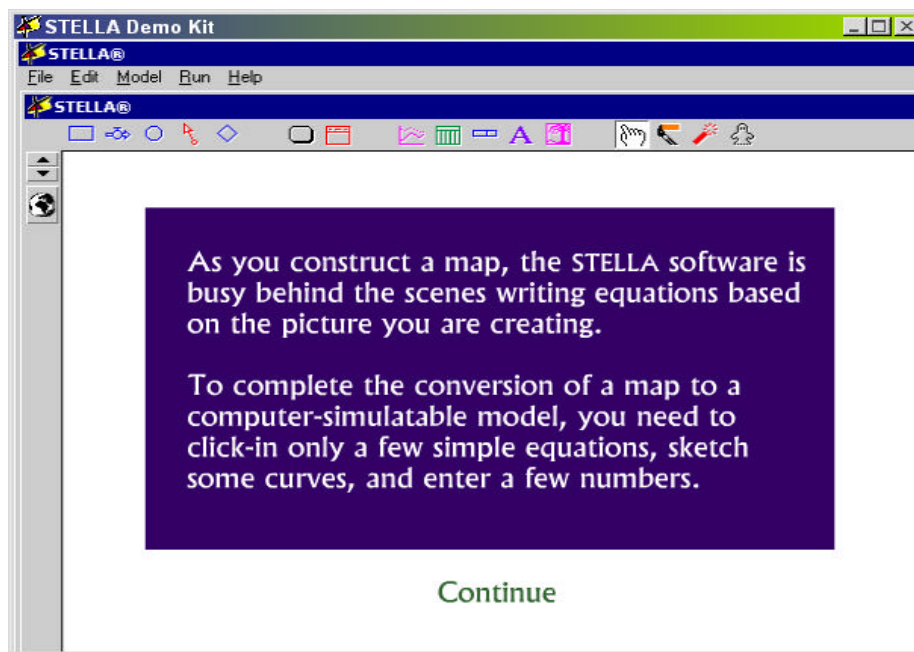
**NOTE:** This project refers to STELLA 7.0. However, the programs also apply to earlier versions of STELLA. If you have a different version of STELLA, you may notice some minor differences. In case you have any questions, ask your teacher or mentor for help.

## Useful Information/Concepts

A series of well-structured and informative tutorials guide you through learning how to use the STELLA software. The tutorials are provided by the supplier of the software at <http://www.hps-inc.com/STELLAdemo.htm> or in case you have already installed the software on your computer, you can find all this information under the Help Menu. Make sure that you cover the following tutorials because for the purposes of this project, we will consider all this introductory information to be prior knowledge:

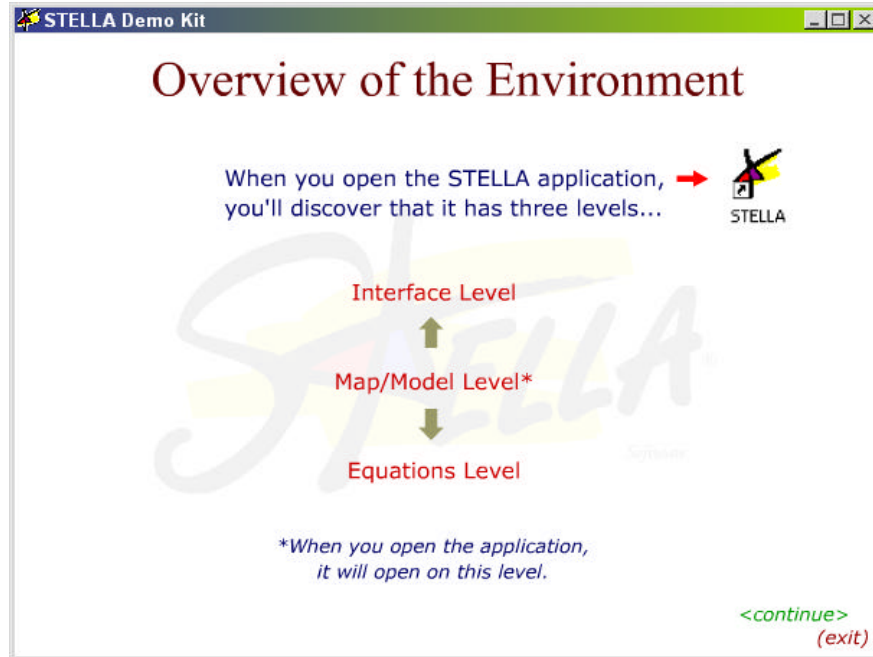
 STELLA: How it works (for example see figure 2).

**Figure 2**



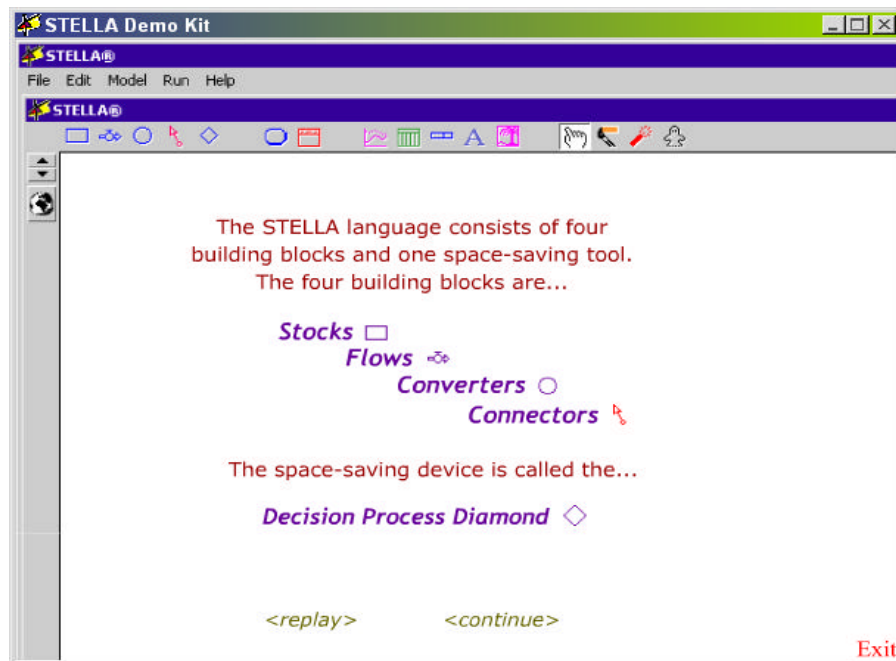
- Overview of the STELLA Environment (for example see figure 3).

**Figure 3**



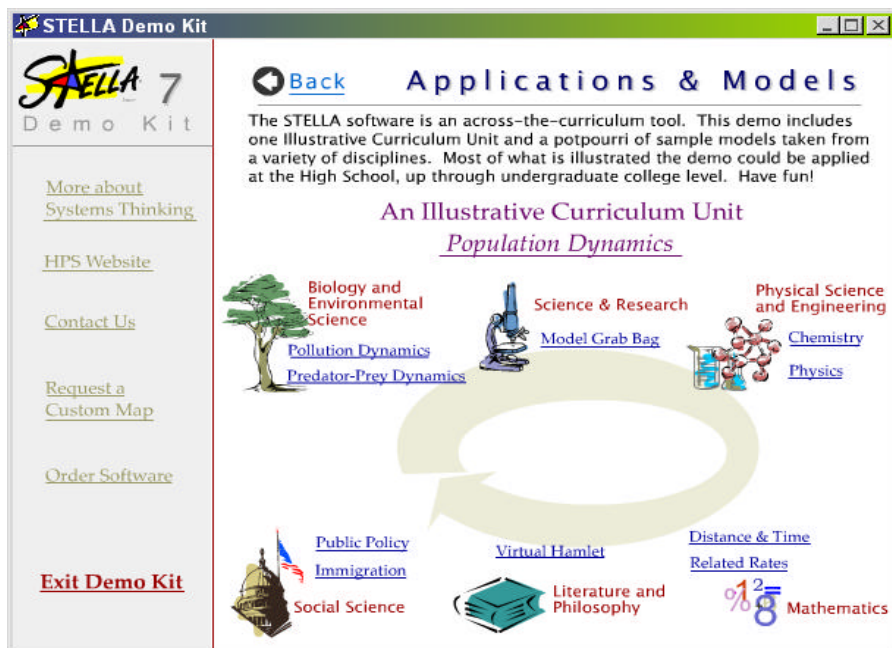
- Overview of the STELLA Language (for example see figure 4).

**Figure 4**



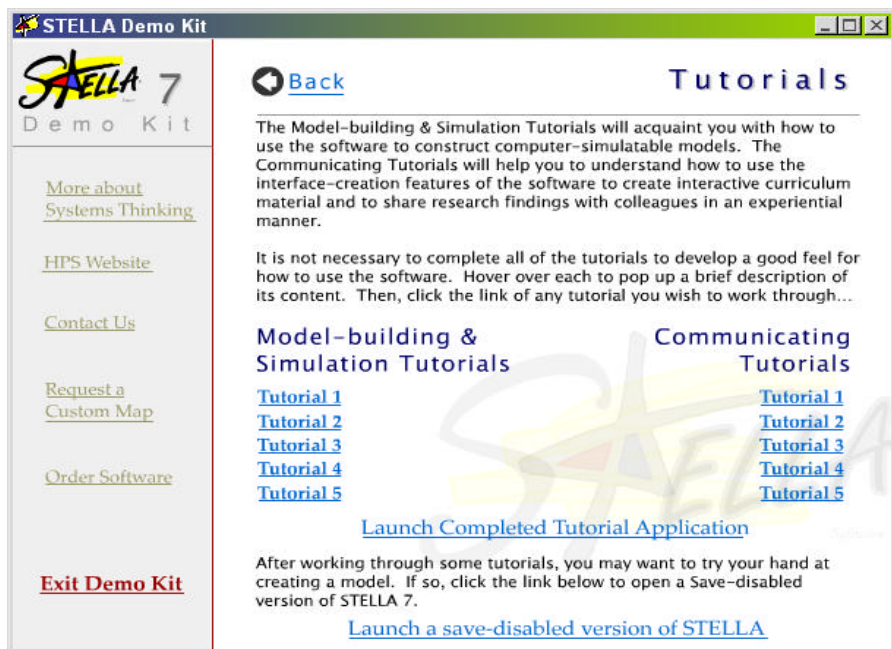
Applications & Models (for example see figure 5).

Figure 5



Tutorials (for example see figure 5).

Figure 5





Note: For running the aforementioned STELLA tutorials, you will need the Multimedia Flash player. In case you do not know what this software exactly is, do not worry. As soon as you select one of the tutorials, the system will inform you whether you have it installed on your computer or not. If you do not have it, there will be an option available to you to download it for free.

## Objectives

Completion of the activities should enable you:

- to learn how to use a mapping language
- to use STELLA as a tool to create simulations
- to graph the results of your simulations

**Materials:** computer, STELLA (visit <http://www.hps-inc.com/ordering/PriceOrderMain.htm> for purchasing the software - for installing the software follow the instructions given by the manufacturer).

## Introduction

For this part of the project, we will focus on developing simulations in physics, specifically, in kinematics. This does not mean that STELLA is designed only for creating physics simulations. It allows you to create all sorts of simulations in any science area you can think of. Kinematics were selected because they are easy to simulate and, therefore, appropriate for an introductory project like this one.

You are probably wondering by now, what an interactive simulation is. It is a virtual representation of a phenomenon, which can be studied through a series of observations as we change the variables that control its behavior. An example of such a simulation would be an incline that has a piece of metal sliding towards the ground and you have the possibility of changing its angle, the mass of the metal, the friction coefficient between the two objects, etc., and being able to observe whether any particular changes have occurred in its behavior.

Developing an interactive project, such as a simulation, will help you learn about the power and the potential of computer programming. STELLA makes it even more interesting because it expresses, in a way, your mind's conceptual maps. You could say that a STELLA project is a mirror reflecting your own unique way of thinking and presenting something. What makes it unique is the flexible mapping language, that allows many and different ways of creating a program, whereas other programming languages are restricting because of the structure of their programming language.

Projects like this one are of particular importance because they not only challenge your programming skills, but also challenge your knowledge of the subject you are simulating.

Let's say you want to create a simulation that shows constant velocity and its graphical representation (Maryland Virtual High School CoreModels Project, 2001). First, analyze and define the problem. In other words, select the content and concepts you



want to introduce through the simulation, and then decide the best way to simulate them. In the case of constant velocity, you need to know the following:

- Velocity is a vector and is defined as the rate of change of displacement:

$$V = \Delta S / \Delta t,$$

where  $\Delta S$  is the change in displacement and  $\Delta t$  the time interval it takes for the change in displacement to happen.

- Velocity is not an absolute quantity but is measured relative to other objects. For example, when you hear a car is traveling 55 km/h, you normally assume that it means a velocity relative to the road.
- For constant velocity the relationship  $V = \Delta S / \Delta t$  becomes linear in  $S$  and  $T$ , rather than in their changes:

$$V = S/t,$$

where  $S$  is the displacement and  $t$  the time.

- The units of velocity are distance/time (e.g., m/s, km/h, ft/s, etc.)

Having in mind the aforementioned content and concepts, you are now ready to select the main features that your simulation must include:

- There must be a way of defining/setting both displacement and time.
- A mathematical relationship (equation) must be used to combine velocity, displacement, and time.

Finally, you have to select the way you will map and simulate all the aforementioned. This is again a very serious issue and you have to consider it very carefully, based on the following criteria:

- The simulation must clearly show the content and concepts you want to simulate.
- The creation of the simulation must be possible by using STELLA.
- Before starting, make sure you have a clear plan that includes a diagram of what you want to create, the equations you will use, and the control equipment you will use (i.e., flows, connectors, converters etc.)
- Define user-program interaction (properties of the objects)
- Design the code structure
- Write code/equations
- Test and edit the program

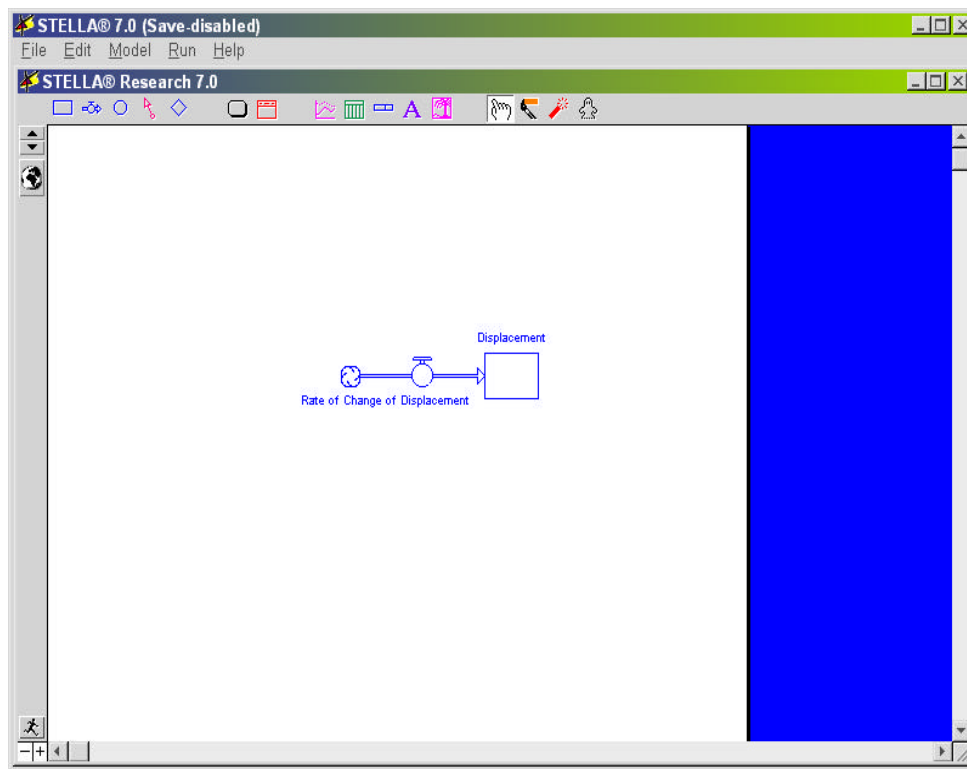
In this project you will simulate constant velocity by using a stock, a flow, and a graph. The stock and the flow will serve as the input variables, and the graph as the output. Now that the desired parameters of the simulation have been specified, you are ready to proceed with the creation of the simulation:



## Procedure

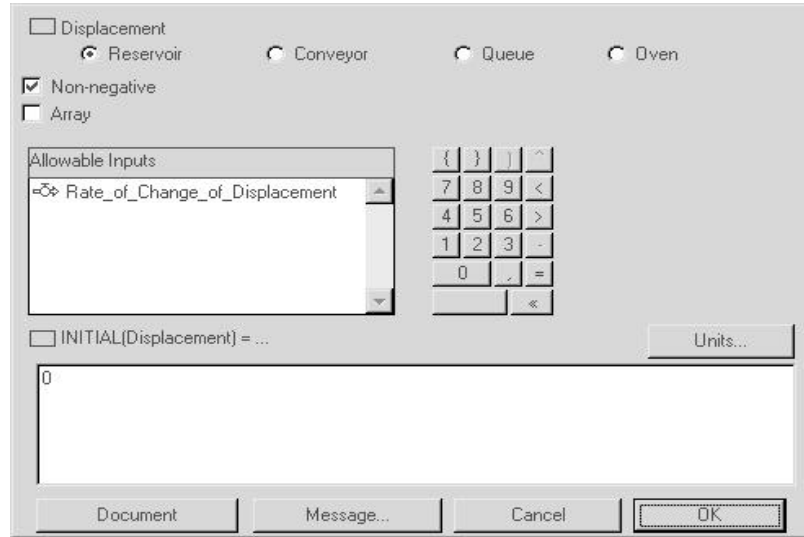
1. Open a new Stella file (double click on the application icon).
2. Click on the stock (reservoir) icon, move the cursor to the center of the process frame and then click again (see figure 4 in case you do not remember which is the stock icon). Name the stock **Displacement**. Click on the flow icon, move the cursor in the left-hand part of the process frame (see figure 4 in case you do not remember which is the flow icon). Press and drag into the stock until the stock becomes shaded. Release the button and name the flow **Rate of Change of Displacement** (see figure 6).

### Figure 6



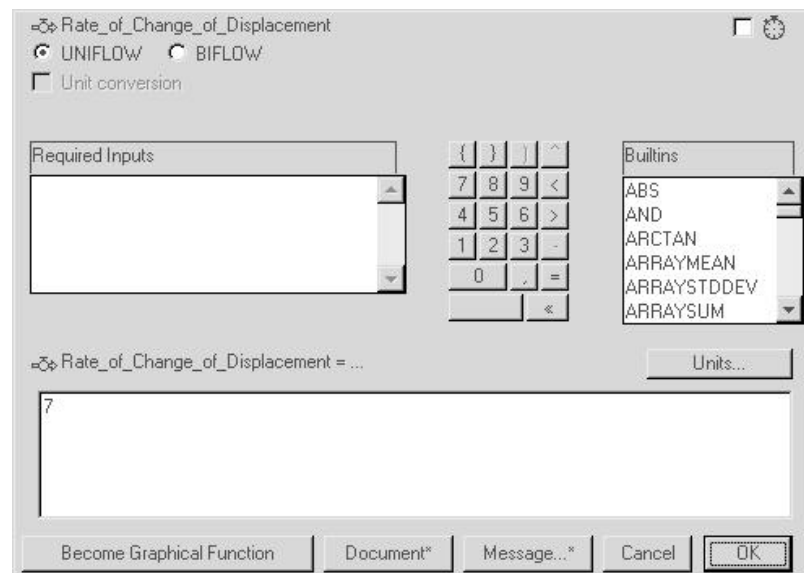
- At this stage, the map of the model has been completed. All the logical connections have been made. What is left to specify is the initial condition required by the stock and the formula used to determine the value of the flow. Click on the planet showing at the top of the left hand margin. The symbol should turn into  $X^2$  and question marks should appear in the flow and stock. Double click on the stock. Enter the initial value of 0 (see figure 7). Select meters from the Units menu. Click on OK when finished.

**Figure 7**



4. Double click on the flow. Set the **Rate\_of\_Change\_of\_Distance = 7** (see figure 8). Select meters per hour from the Units menu. Click on OK when finished. The model is now complete. The next step is to set some parameters, which tell STELLA how to actually run the model.

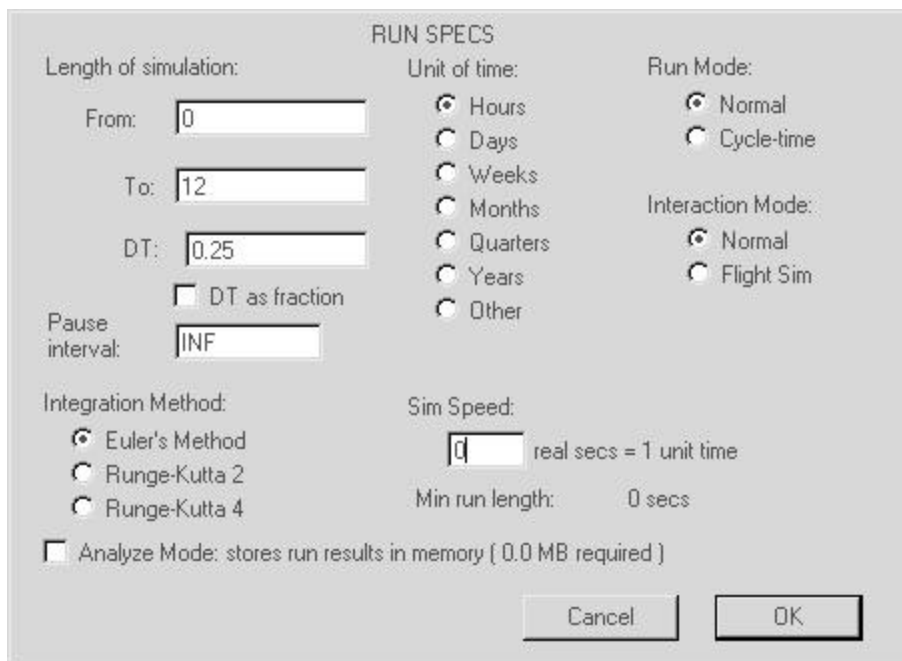
**Figure 8**



5. Select the Run Specs option from the top line RUN menu. When the dialogue box appears, set the DT setting to 0.25, the starting time of the model to 0, the ending time to 12, and the integration method to Euler's [Euler's is the name given to a method of approximating an integral, which is just a fancy name for adding lots of infinitesimal

(very small) things. Runge-Kutta is another approximation]. Lastly, select hours as the time unit (see figure 9). Click on OK when finished.

**Figure 9**



**RUN SPECS**

Length of simulation:

From:

To:

DT:

☐ DT as fraction

Pause interval:

Integration Method:

☒ Euler's Method

☐ Runge-Kutta 2

☐ Runge-Kutta 4

☐ Analyze Mode: stores run results in memory ( 0.0 MB required )

Unit of time:

☒ Hours

☐ Days

☐ Weeks

☐ Months

☐ Quarters

☐ Years

☐ Other

Run Mode:

☒ Normal

☐ Cycle-time

Interaction Mode:

☒ Normal

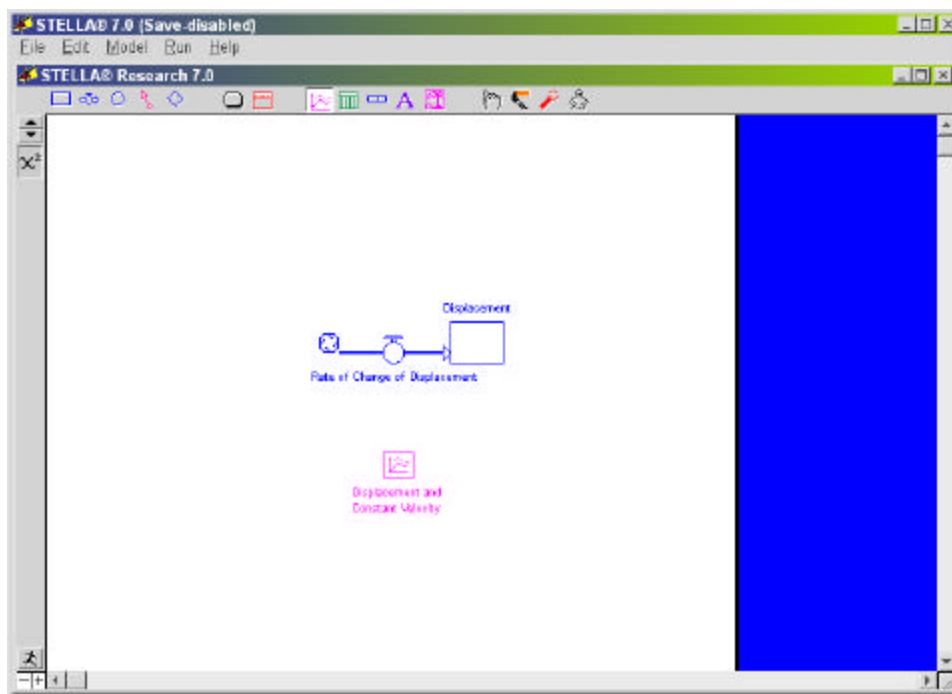
☐ Flight Sim

Sim Speed:  real secs = 1 unit time

Min run length: 0 secs

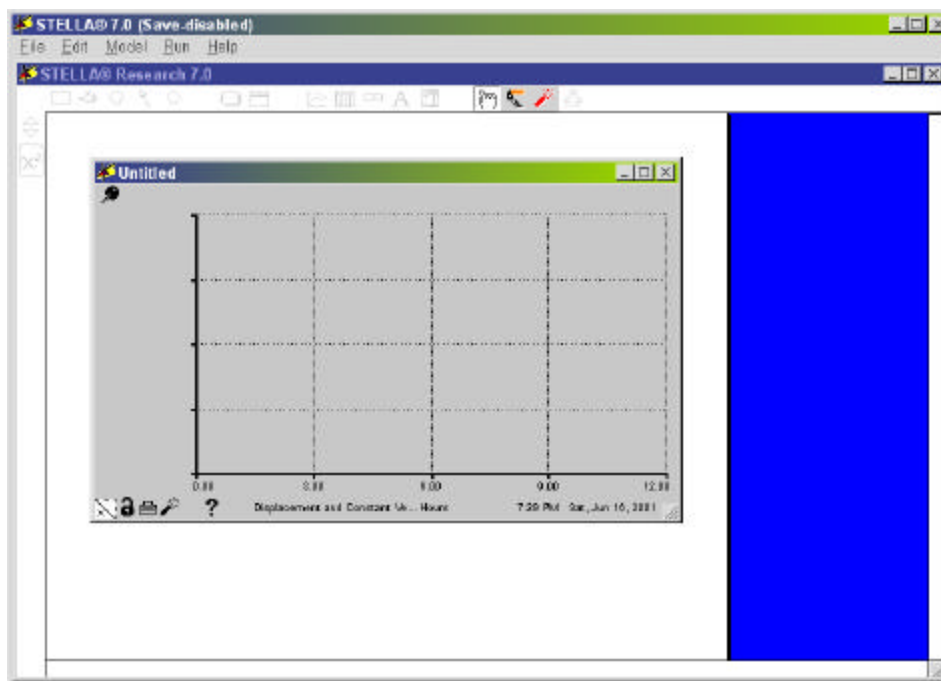
6. To run the simulation, select the RUN option from the RUN menu on the top line of the screen. If all is right, not much should happen (you must not get any notifications from STELLA concerning inconsistencies. If you get a notification, try to resolve the problem based on the feedback/information given to you). You haven't yet arranged for any form of output. Two forms of output are common from STELLA models. One form is graphical output showing how displacement changes over time. A second is tabular output, showing table entries giving precise values of various quantities as the model progresses. The setups for these options are very similar. For this model we will go through the graphical setup. Click on the graph icon, move the cursor below your stock and flow, and then click again (there is no a particular reason for placing the graph below the stock and flow, besides aesthetic reasons. You can place your graph anywhere on your frame). The graph icon must appear on your frame. Name the graph **Displacement and Constant Velocity** (see figure 10).

**Figure 10**



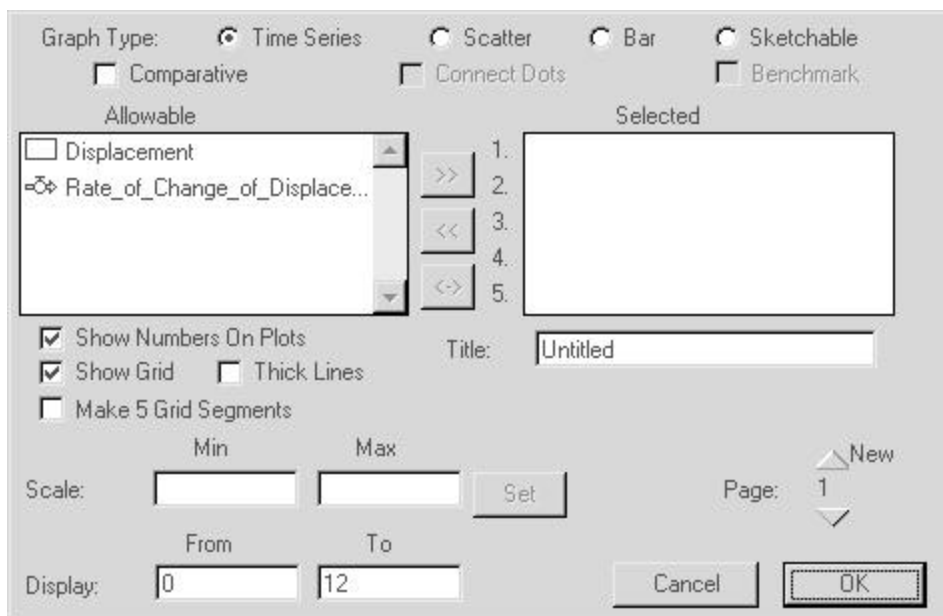
Double click on the graph icon. A graph figure will appear on your frame (see figure 11).

**Figure 11**



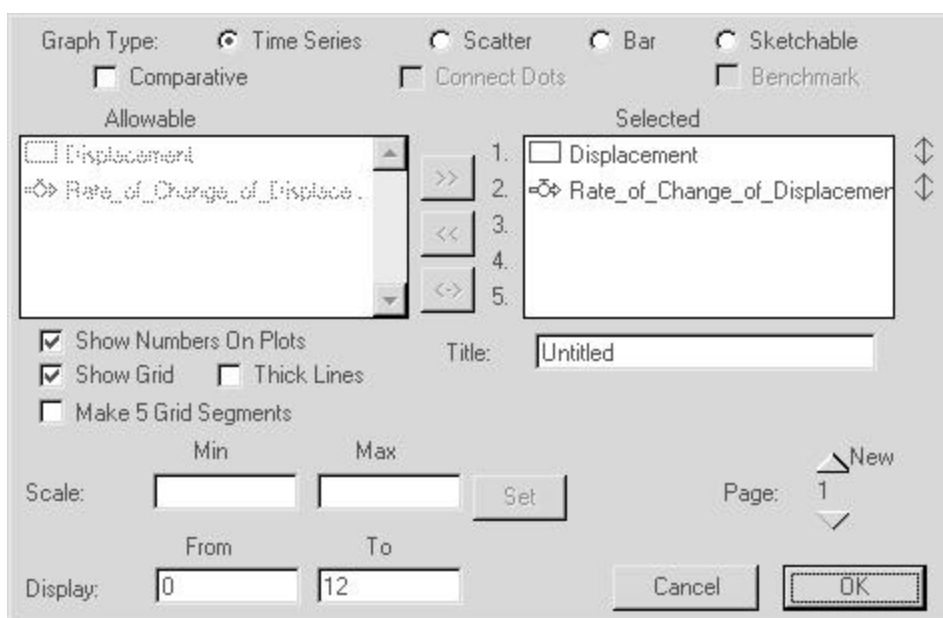
Double click on the graph, and the following dialog box will appear (see figure 12).

**Figure 12**



Notice that both of your variables, Displacement and Rate of Change of Displacement, appear in the Allowable window. First, select the Displacement and transfer it to the Selected window (use the arrow buttons for the transfer. In this case use the right facing arrow button), and second, transfer the Rate of Change of Displacement (see figure 13). Click on OK when finished.

**Figure 13**



7. Run the simulation again. Double click on the graph icon. What do you observe? Which of the two lines represents the **Rate of Change of Displacement Rate** and which the **Displacement**? What is the slope of the two lines? Is it consistent with what you learned in kinematics for constant velocity? Change the input values for both the **Rate of Change of Displacement** and the **Displacement**, and describe how the graph changes.

This is meant to be an introductory model activity, but with extensions can become a much more involved project. Let's consider expanding the constant velocity model into a simulation about constant acceleration and its graphical representation (it is the next logical step to take, since acceleration is the rate of change of velocity) (Maryland Virtual High School CoreModels Project, 2001). Once again, you have to follow the procedures introduced at the beginning of this project, starting with the analysis and the definition of the problem. In the case of constant acceleration, in addition to what we mentioned before for the velocity, you need to know the following:

- Acceleration is a vector and is defined as the rate of change of velocity:

$$a = \Delta V / \Delta t,$$

where  $\Delta V$  is the change in velocity and  $\Delta t$  the time interval it took for the change in velocity to occur.

- For constant velocity the relationship  $a = \Delta V / \Delta t$  becomes linear in  $S$  and  $T$ , rather than in their changes and  $T$ , rather than in their changes:

$$a = V/t,$$

where  $V$  is the velocity and  $t$  the time.

- The units of acceleration are distance/time<sup>2</sup> (i.e., m/s<sup>2</sup>, km/h<sup>2</sup>, ft/s<sup>2</sup>, etc.)

Having in mind the aforementioned content and concepts, you are now ready to select the main features that your simulation must include:

- There must be a way of defining/setting velocity, displacement, and time.
- A set of mathematical relationships (equations) must be used to combine velocity, displacement, and time.

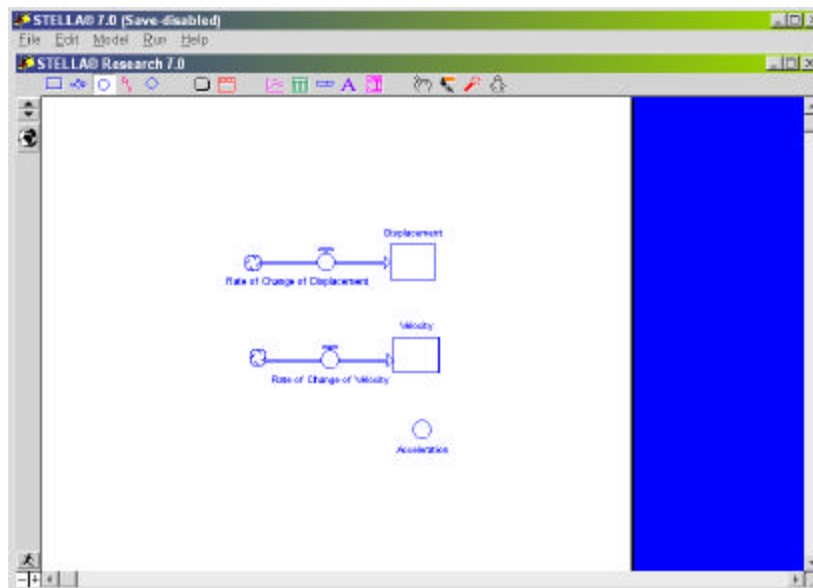
Finally, you have to select the way you will map out and simulate all the aforementioned. For this simulation, a combination of two stocks, two flows, a converter, and a graph will be used. The stocks, the converter, and the flows will serve as the input variables, and the graph as the output. Now that the desired parameters of the simulation have been specified, you are ready to proceed with the creation of the simulation:

8. Repeat steps 1 and 2. Add one more stock icon, right below **Displacement**. Name the stock **Velocity**. Click on the flow icon, move the cursor in the left-hand part of the process frame. Press and drag into the **Velocity** stock until the stock becomes shaded.



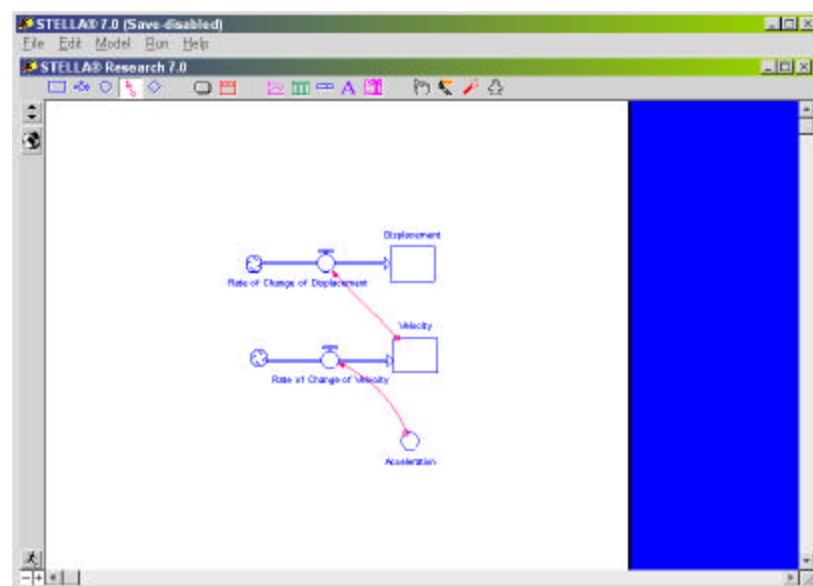
Release the button and name the flow **Rate of Change of Velocity**. Add a converter, right below **Velocity**. Name the converter **Acceleration** (see figure 14).

**Figure 14**



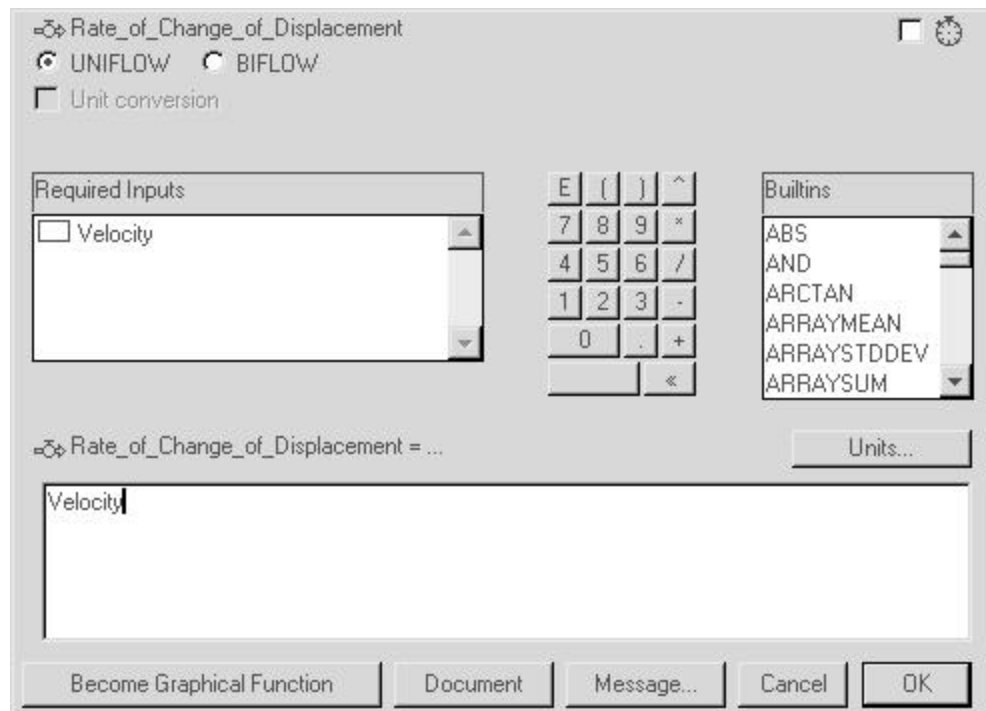
9. Click on the action connector icon, then on the **Velocity** stock, and then on the **Rate of Change of Displacement** flow. This will connect the two variables (this also means that the two variables relate to each other). Repeat the same procedure and connect the **Acceleration** converter to the **Rate of Change of Velocity** (see figure 15).

**Figure 15**



10. At this stage, the map of the model has been completed. All the logical connections have been made. What is left is to specify the initial conditions. Click on the planet showing at the top of the left hand margin. The symbol should turn into  $X^2$  and question marks should appear in the flows, converter, and stocks. Double click on the **Displacement**. Enter the initial value of 0. Select meters from the Units menu. Click on OK when finished.
11. Double click on the **Velocity**. Enter the initial value of 0. Select meters/hr from the Units menu. Click on OK when finished.
12. Double click on the **Acceleration**. Enter the initial value of 3. Select meters/hr<sup>2</sup> from the Units menu. Click on OK when finished.
13. Double click on the **Rate of Change of Displacement** flow. Set the **Rate\_of\_Change\_of\_Distance = Velocity**. Use the list of variables on the upper left-hand side of the dialogue box (see figure 16). Click on OK when finished.

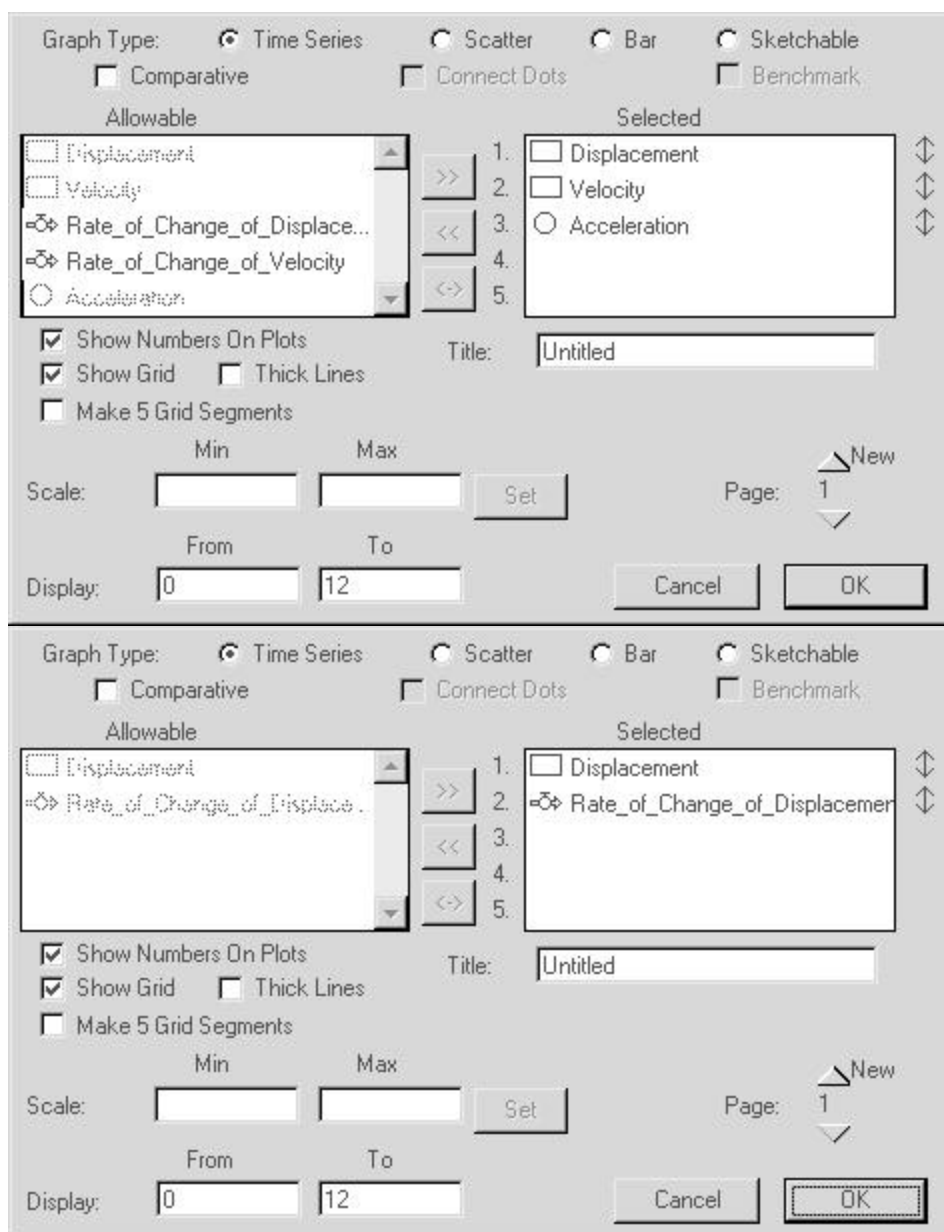
**Figure 16**



14. Double click on the **Rate of Change of Velocity**. Set the **Rate\_of\_Change\_of\_Velocity = Acceleration**. Click on OK when finished.
15. Select the Run Specs option from the top line RUN menu. When the dialogue box appears, set the DT setting to 0.25, the starting time of the model to 0, the ending time to 12, and the integration method to Euler's. Lastly, select hours as the time unit. Click on OK when finished.

16. Click on the graph icon, move the cursor below your model and then click again. The graph icon must appear on your frame. Name the graph **Constant Acceleration**. Double click on the graph icon and then double click on the graph. Select the **Displacement**, **Velocity**, and **Acceleration**, and transfer them to the Selected window (see figure 17). Click on OK when finished.

**Figure 17**



17. Run the simulation again. Double click on the graph icon. What do you observe? Which of the three lines represents the **Displacement**, which the **Velocity**, and which

the **Acceleration**? Describe the slopes of the lines. Are they consistent with what you learned in kinematics? Change the input values and describe how the graph changes.

## Activities

1. Follow the aforementioned procedures to simulate Hooke's Law. As an extension to Hooke's Law, simulate the potential energy of a spring (if you are not familiar with Hooke's Law visit the following websites <http://www.purchon.com/physics/hookeslaw.htm> and <http://www.sciencejoywagon.com/physicszone/lesson/02forces/hookeslaw.htm>).
2. Follow the instructions given on <http://www.stolaf.edu/people/mckelvey/envision.dir/malthus.html> to simulate the Malthusian Growth Model. Answer the two problems given at the end.
3. Create a STELLA program that perform the following tasks (there's no simulation here. The purpose of this activity is to show that STELLA can be used for things other than simulations.):
  - Converts temperature from Fahrenheit to Kelvin and Celsius.
  - Converts pounds to kilograms
  - Converts miles to kilometers
  - Calculates the area of various shapes (i.e., squares, rectangles, triangles, hexagons, circles etc.)
  - Calculates the volume of various shapes (i.e., spheres, pyramids, cylinders etc.)
  - Calculates mathematical operations (i.e., adding, multiplication, deviation etc.)
4. Visit <http://www.ties.k12.mn.us/envision/97/steve.html> to find a series of STELLA simulations. Try to create all the suggested simulations and to answer all the given problems. Based on the experience gained from this activity, create your own series of mathematical models.
5. Create a series of models that simulate the population model of different species (i.e., buffalos, zebras, snakes etc.). Visit <http://indian.nehs.ce.k12.md.us/pages/deerpop.shtml> to see an example. As an extension, create more complicated models that show parts of the food chain (i.e., combine the population model of rats, snakes, and hawks).
6. Visit <http://indian.nehs.ce.k12.md.us/pages/compmod.shtml> to find a series of STELLA simulations. Try to create all the suggested simulations and to answer all the given problems. Based on the experience gained from this activity, create your own series of biological or environmental science simulations.

7. Visit <http://isaac.williamsport.wa.k12.md.us/~ctrout/sciproj/STELLA/> to find a series of STELLA physics simulations. Try to create all the suggested simulations. Based on the experience gained from this activity, create your own series of physics simulations.

## **Final Project/Report**

Create a STELLA program that will be of use for a science project. Include a report explaining the reasoning behind your selection (purpose, benefits from its operation, etc.) and an analytical step by step description of your program. Describe the science project, and how your program is useful for it.

## **Reference**

Maryland Virtual High School CoreModels Project (2001). *STELLA Projects*. URL: <http://isaac.williamsport.wa.k12.md.us/~ctrout/sciproj/STELLA/>.

## Unit 3: Computer Programming in Science Education – Spreadsheets Project 19

### Using Spreadsheet Features as a Mapping Language to Simulate Science Concepts

The use of even fundamental computer software can help you understand concepts and processes of science. Spreadsheets are one of those powerful tools that can offer you the possibility creating dynamic projects. For example, they can be used as a tool for simulating science concepts. But, how can this be done? Follow us through the process of creating your own simulations and find out for yourself.

#### Degree of Difficulty

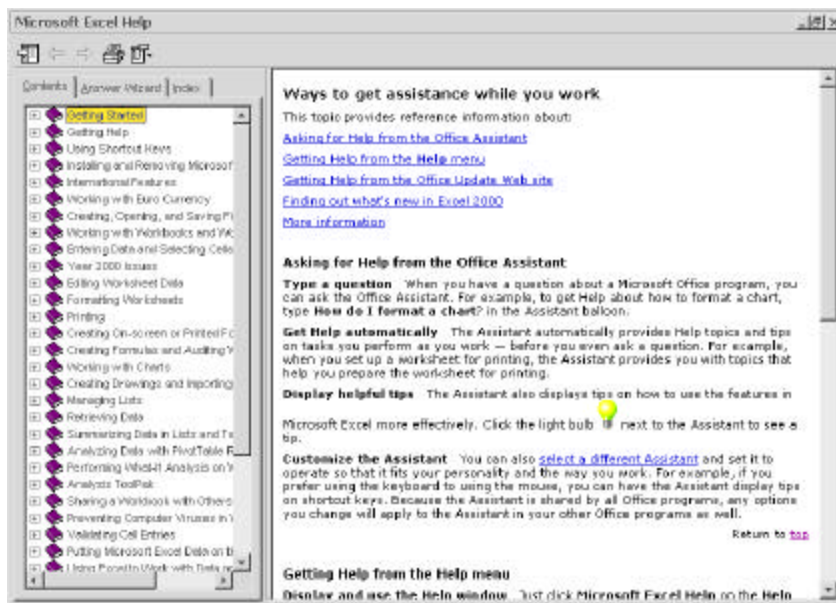
Experimental: Easy

Conceptual: Easy











#### Prerequisite Knowledge

Experience using a Windows operating system is required. Students must have some experience in basic computer functions (e.g., access programs, open programs, save data), and have knowledge of basic computer programs (e.g., word processing programs etc.) including spreadsheets. In case you are not familiar with spreadsheets, use the **Help Menu** (see figure 1 – Note that this is the Help Menu of EXCEL 2000).

**Figure 1**



Despite what version of EXCEL you have you can find under the Help Menu a series of well-structured and informative tutorials/descriptions that guide you through learning how to use the spreadsheets. Make sure that you cover the following tutorials/descriptions because for the purposes of this project, we will consider all this introductory information to be prior knowledge:

-  Getting Started
-  Creating, Opening and Saving Files
-  Working With Workbooks and Worksheets
-  Entering Data and Selecting Cells
-  Editing Worksheet data
-  Formatting Worksheets
-  Creating Formulas and Auditing Workbooks
-  Working With Charts
-  Create Drawings and Import Pictures
-  Analysis ToolPak

However, the features that will be most useful in this project are:

- Enter data and formulas into the spreadsheet cells
- Data and formulas stored in an array of cells - visually appealing
- Ability to control calculations
- Powerful charting capability for visualization

For programming in Spreadsheets, there is no need of knowing any particular programming language. It requires only writing words or equations very much in the same way we do in school mathematics.

**NOTE:** This project refers to EXCEL of Microsoft Office 2000. However, the instructions also apply to other versions of spreadsheets. If you have a different version, you may notice some minor differences. If you have any questions, ask your teacher or mentor for help.

## **Useful Information**

- As mentioned before, this project will focus on developing simulations by using a mapping language. However, spreadsheets were not designed for this particular application, but with some creativeness and imagination you can use it as a mapping and simulating tool. The fact that it was not designed specifically for this kind of application makes your project even more challenging. In cases where you are dealing with software, such as STELLA, that is designed to simulate concepts by using a mapping language, you can make use of the already established software objects, tools, commands etc., whereas the same attempt in spreadsheets



requires starting from the very beginning, which includes setting up the tools and defining the language you will be using. For a better understanding the difference between using spreadsheets for mapping and simulating, and using software with an already established mapping language (e.g., STELLA), it is suggested that you complete this project, then project 18, and then compare the two of them. For this reason, this project's simulations will be derived from kinematics (it will be easier this way to compare the two). But you can create simulations in other science domains.

- An interactive simulation is a virtual representation of a phenomenon, which can be studied through a series of observations as we change the variables that control its behavior. An example of such a simulation would be an incline that has a piece of metal sliding towards the ground and you have the possibility of changing its angle, the mass of the metal, the friction coefficient between the two objects etc., and being able to observe whether any particular changes have occurred in its behavior.
- Developing an interactive project, such as a simulation, will help you learn about the power and the potential of computer programming. Using spreadsheet features as a mapping language makes it even more interesting because it expresses your mind's conceptual maps. What makes it unique is the mapping language, that you make as flexible as possible, and that allows many different ways of creating a program, whereas other programming languages are restricting because of the structure of their programming language.

## Objectives

Completion of the activities should enable you:

- to learn to use a spreadsheet program as a mapping language to simulate science concepts
- to use spreadsheets as a tool to make theoretical calculations
- to graph the results of your simulations.

**Materials:** computer, and EXCEL spreadsheets (or any other spreadsheet program - all other common spreadsheet programs do what is needed for this experiment)

## Introduction

Projects like this one are of particular importance, because they do not only challenge your programming skills, but also challenge your knowledge of the subject you are simulating. Therefore, you have to be very well organized and prepared before starting to create your project. Before starting work on a spreadsheet simulating project, always

- State the problem clearly.
- Describe the input and output information.
- Work the problem by hand or with a calculator for a simple set of data.
- Develop an EXCEL solution (set/define the symbols, tools, objects and language you will use as a mapping language).
- Test the solution using a variety of data sets.

Let's say you want to create a simulation that shows constant velocity and its graphical representation. First, analyze and define the problem. In other words, select the content and concepts you want to introduce through the simulation, and then decide the best way to simulate them. In the case of constant velocity, you need to know the following:

- Velocity is a vector and is defined as the rate of change of displacement:

$$V = \Delta S / \Delta t,$$

where  $\Delta S$  is the change in displacement and  $\Delta t$  the time interval it takes for the change in displacement to happen.

- Velocity is not an absolute quantity, but is measured relative to other objects. For example, when you hear a car is traveling 55 km/h, you normally assume that it means a velocity relative to the road.
- For constant velocity the relationship  $V = \Delta S / \Delta t$  becomes linear in  $S$  and  $T$ , rather than in their changes:

$$V = S/t,$$

where  $S$  is the displacement and  $t$  the time.

- The units of velocity are distance/time (e.g., m/s, km/h, ft/s, etc.)

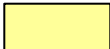
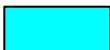

Having in mind the aforementioned content and concepts, you are now ready to select the main features that your simulation must include:

- There must be a way of defining/setting both displacement and time.
- A mathematical relationship (equation) must be used to combine velocity, displacement, and time.

Finally, you have to select the way you will map and simulate all the aforementioned. This is again a very serious issue, and you have to consider it very carefully, based on the following criteria:

- The simulation must clearly show the content and concepts you want to simulate.
- The creation of the simulation must be possible by using spreadsheets.

- Before starting, make sure you have a clear plan that includes a diagram of what you want to create, the equations you will use, and the control equipment you will use. For example, in this project the following objects will be used for mapping:

Symbol/Object	Name	Use
	Rate of change	Dependent Variable. The cell includes the equation that relates participating variables.
	Variable	Independent Variable. The cell includes the value of the Independent Variable.
	Connector	Shows relationships between variables. The tip of the arrow shows the dependent variable and the other end of the arrow the independent variable.

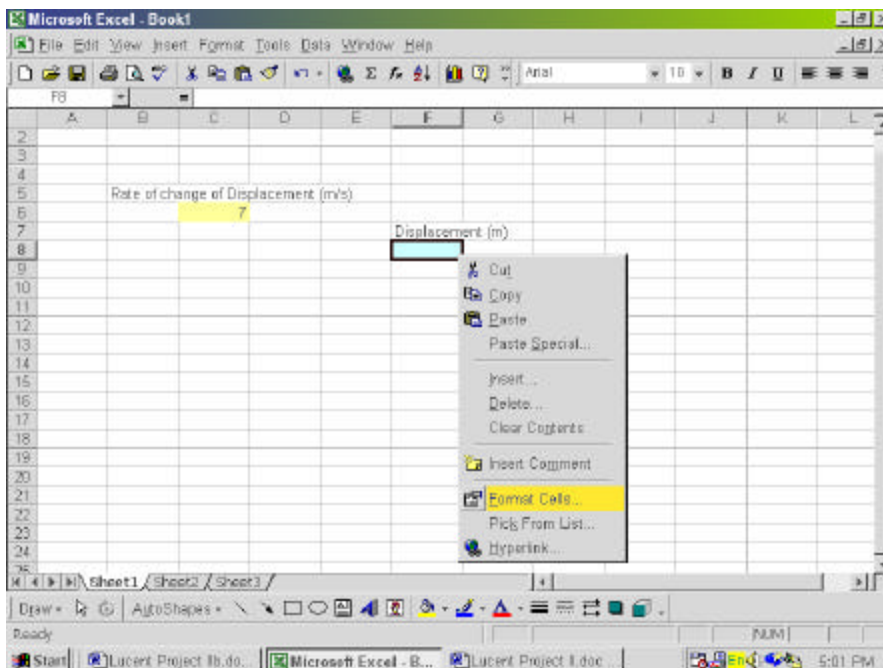
- Define user-program interaction (properties of the objects)
- Test and edit the program


Now that the desired parameters of the simulation have been specified, you are ready to proceed with the creation of the simulation:

## Procedure

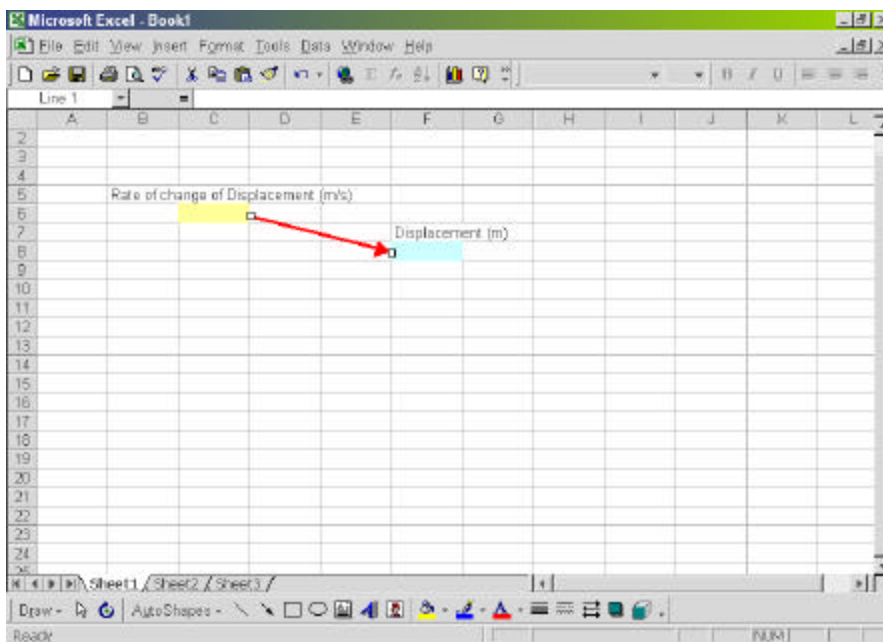
1. Open a new worksheet.
2. Enter the names **Rate of change of Displacement** and **Displacement** in cells B5 and F7, respectively. Click on cell B6 (the cell right below the **Rate of change of Displacement** cell), then right click and change the filling color of the cell to light yellow. For changing the color click on Format Cells... (see figure 2), select Pattern, and then select the color.
3. Click on cell F8 (the cell right below the **Displacement** cell), then right click and change the filling color of the cell to light blue (make sure that the color you selected does not obscure entered data).
4. Click on cells B5 and F7, and include the appropriate units in parentheses next to **Rate of change of Displacement** and **Displacement**. At this point your worksheet should look similar to figure 2.


**Figure 2**



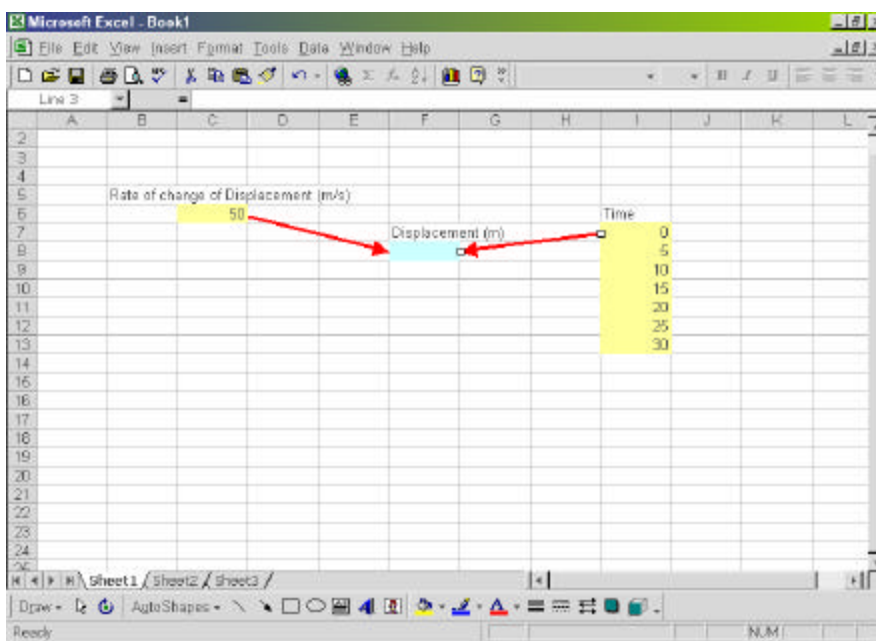
5. Select the cursor image  from the Drawing Toolbar (in case it is not part of the tools that appear on your screen, go under View, select Toolbars, and then Drawing). Click on cell B5, move the cursor to the center of the left side of cell F7, and then click again. Select the created arrow, right click, select Format Auto Shape..., and change its color to red (see figure 3).

**Figure 3**



6. Click on the yellow cell and enter the initial value of 50 (this means that the initial velocity will be 50 m/s).
7. Click on cell I6 and name it Time. Click on cell I7 (the cell right below the **Time** cell), then right click and change the filling color of the cell to light yellow. Click again on cell I7 and enter the initial value of 0 (this means that the time starts at 0 s). Select the cursor image  from the Drawing Toolbar. Click on cell I7, move the cursor to the center of the right side of cell F7, and then click again. Select the created arrow and change its color to red (see figure 4).
8. Click on cell I8 and enter the initial value of 5. Highlight the cells I7 and I8, and move your cursor towards the lower right corner of the I8 cell until it becomes a cross. As soon as it becomes a cross, click and drag your mouse downwards for five cells. Release your mouse. You should see those six cells to be filled with certain values (multiples of five), (see figure 4). This way, you define the time range you are interested in graphing afterwards.

**Figure 4**

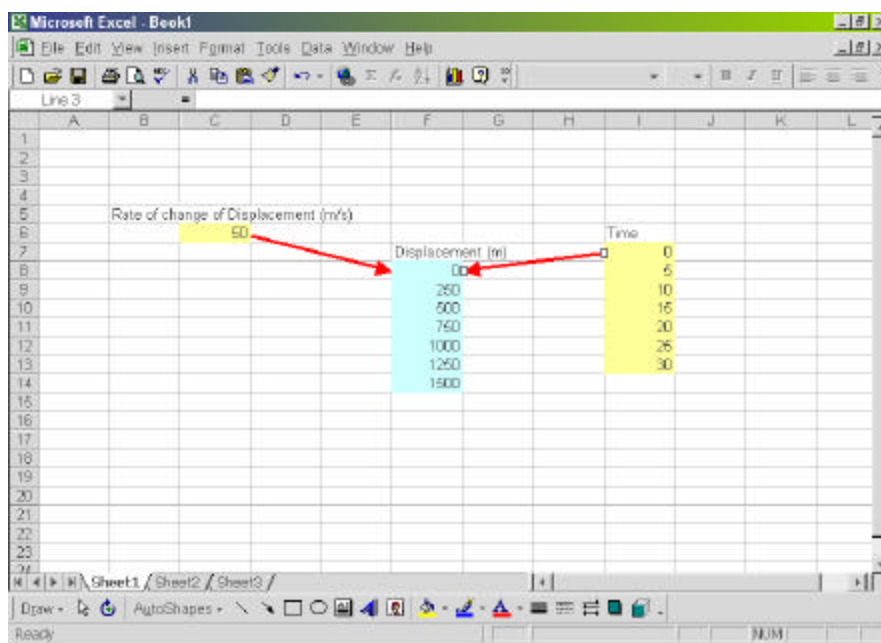


At this stage, the map, of the model has been completed. All the logical connections have been made. What is left is to define how the displacement depends upon the rate of change of displacement (velocity). This is the point where our theoretical review comes in. As we mentioned at the beginning of this project, it is known from kinematics that when the rate of change of displacement is constant (constant velocity), the relationship between the displacement and the constant velocity is linear. In fact, the displacement equals the product of constant velocity and time:

$$S = V \times t$$

9. Click on the light blue cell and insert the expression  $=\$C\$6*17$  (the \$ signs are included to declare that the value included in cell C6 is a constant). This expression will give you the product of constant velocity (50 m/s) and time (0 s). However, we are interested in a wider time interval, not only what the displacement is at 0 sec. To get the other displacement values, click on the blue cell and move your cursor towards the lower right corner of the cell until it becomes a cross. As soon as it becomes a cross, click and drag your mouse downwards for six cells. Release your mouse. You must see those seven cells to be filled with certain values (multiples of fifty), (see figure 5). The model is now complete.

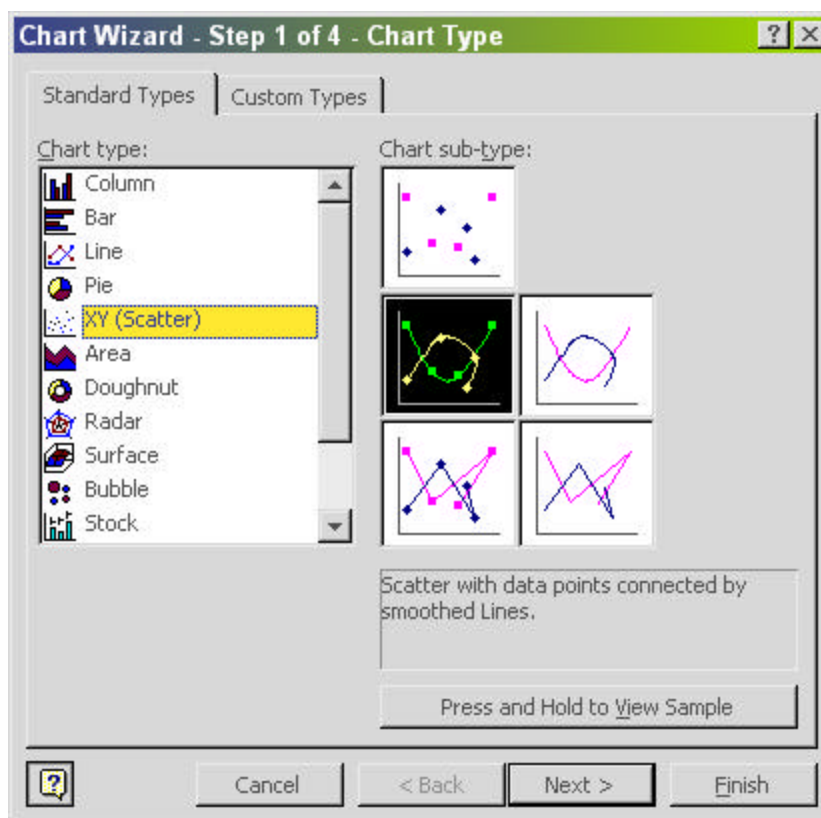
**Figure 5**



10. Even though our model is complete, what we see on the screen is not that informative. Therefore, a graphical output showing how displacement changes over time is really necessary. Highlight all the time values, press and hold the control key on your keyboard, and then highlight all the Displacement values. Release the control key after you finish highlighting.
11. Select the **Chart Wizard** from your Excel Toolbar (or go under “Insert” and select “chart”). After you click on the **Chart Wizard**, you will see the cursor of your mouse change to a cross. Click on a free space of your spreadsheet, hold your mouse’s left button, drag it diagonally for 4-5 cm (depending on how big you want your graph to be), and then release the button. At this point you will see a window and the chart wizard dialogue box on the screen. Select the Scatter Plot option (see figure 6).

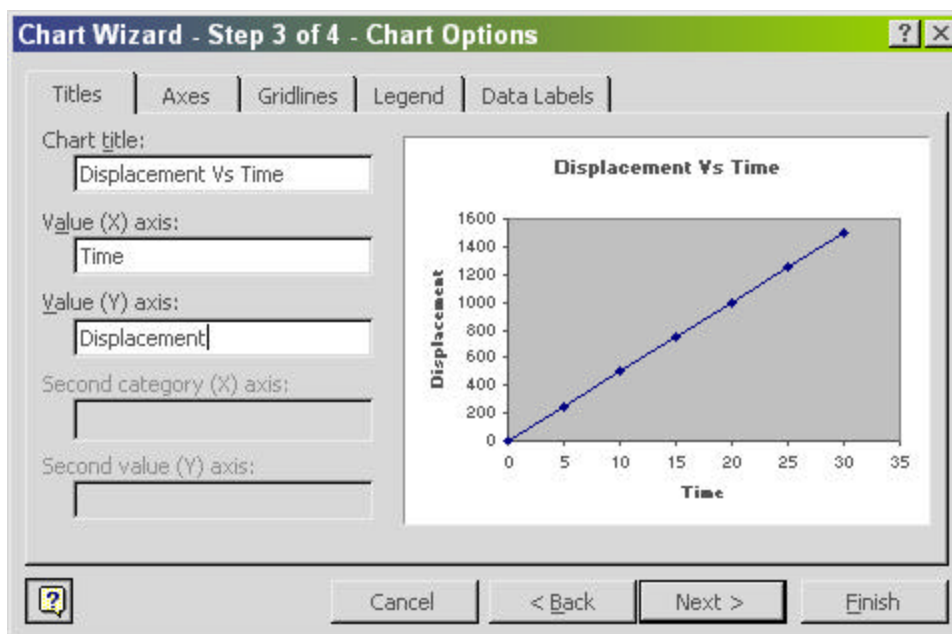


**Figure 6**



12. Let the dialogue box guide you through until you get your graph (see figure 7).

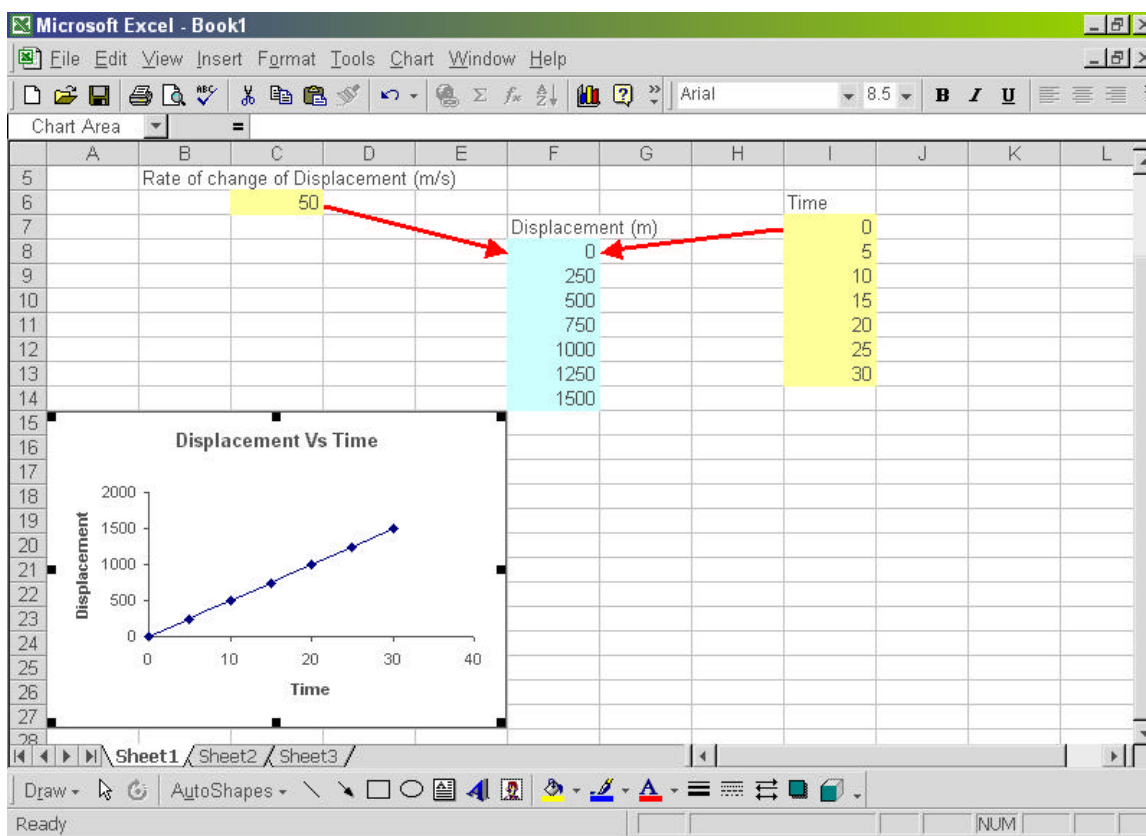
**Figure 7**





13. By the end, your worksheet should look like figure 8.

**Figure 8**



14. Your simulation is now complete. Change the input value of the **Rate of Change of Displacement (Constant Velocity)**, and run your simulation again. What do you observe? How does the slope of the line change? Is it consistent with what you learned in kinematics for constant velocity? Continue changing the input values for both the **Rate of Change of Displacement** and the **Time**, and describe how the graph changes.

This was meant to be an introductory model activity, but with extensions can become a much more involved project. Let's consider expanding the constant velocity model into a simulation about constant acceleration and its graphical representation (it is the next logical step to take, since acceleration is the rate of change of velocity). Once again, you have to follow the procedures introduced at the beginning of this project, starting with the analysis and the definition of the problem. In the case of constant acceleration, in addition to what we mentioned before for the velocity, you need to know the following:

- Acceleration is a vector and is defined as the rate of change of velocity:

$$a = \Delta V / \Delta t,$$

where  $\Delta V$  is the change in velocity and  $\Delta t$  the time interval it took for the change in velocity to occur.

- For constant velocity the relationship  $a = \Delta V / \Delta t$  becomes linear in  $V$  and  $T$ , rather than in their changes:

$$a = V/t,$$

where  $V$  is the velocity and  $t$  the time, or

$$a = S/t^2,$$



since  $V = S/t$ .

- The units of acceleration are distance/time<sup>2</sup> (i.e., m/s<sup>2</sup>, km/h<sup>2</sup>, ft/s<sup>2</sup>, etc.)

Having in mind the aforementioned content and concepts, you are now ready to select the main features that your simulation must include:

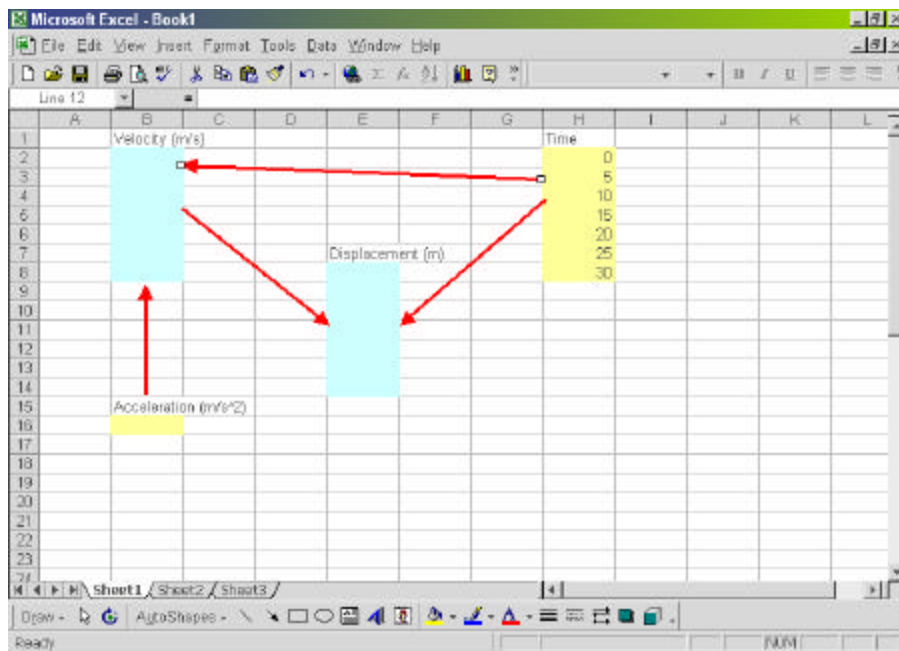
- There must be a way of defining/setting velocity, displacement, and time.
- A set of mathematical relationships (equations) must be used to combine velocity, displacement, and time.

Finally, you have to select the way you will map and simulate all the aforementioned. For this simulation, an additional variable (acceleration) will be used (in addition to the model we constructed for constant velocity). Now that the desired parameters of the simulation have been specified, you are ready to proceed with the creation of the simulation:

15. Repeat steps 1 and 8 (note that in this part, the cell numbers for Displacement, Time, etc. will not be the same as in the previous part – see figure 9 and make the necessary adjustments). Rename the **Rate of Change of Displacement** variable to **Velocity**, and change the color of its cells to light blue (see figure 9). Why do we have to change the color of the velocity cells?
16. Add one more variable, in cell B15, and name it **Acceleration**. Click on cell B16 (the cell right below the **Acceleration** cell), then right click and change the filling color of the cell into light yellow (see figure 9).
17. Select the cursor image  from the Drawing Toolbar. Click on the **Acceleration** cell (B15), move the cursor towards the Velocity cells (i.e., B8), and then click again. Select the created arrow, and change its color to red (see figure 9).
18. Select the cursor image  from the Drawing Toolbar. Click on the time box, move the cursor towards the Velocity cells (e.g., B2), and then click again. Select the

created arrow, and change its color to red. At this point your worksheet should look similar to figure 9.

**Figure 9**



At this stage, the map of the model has been completed. All the logical connections have been made. What is left is to define how the displacement depends upon the velocity, and how the velocity depends upon the acceleration. As we mentioned at the beginning of this project, it is known from kinematics that when the velocity is constant, the relationship between the displacement and the constant velocity is linear. In fact, the displacement equals the product of velocity and time

$$S = Vt$$

In the case where the acceleration is constant, the relationship between the acceleration and the constant velocity is linear. The velocity equals the product of constant acceleration and time

$$V = at$$

However, the relationship between the time and the displacement is not linear:

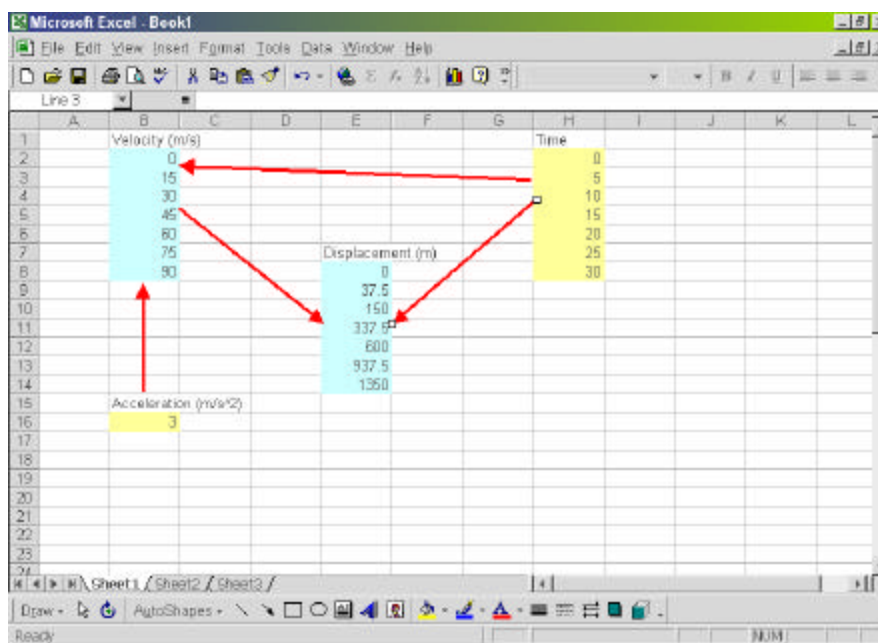
$$S = \frac{1}{2} a t^2 = \frac{1}{2} V t$$

19. Click on the yellow cell under the **Acceleration** and enter the value of 3.
20. Click on the first light blue cell under the **Velocity** and insert the expression **=B\$16\*H2**. This expression will give you the product of constant acceleration (3

m/s<sup>2</sup>) and time (0 s). However, we are interested in a wider time interval and not only what the velocity is at 0 sec. To get the other velocity values, click on the light blue cell and move your cursor towards the lower right corner of the cell until it becomes a cross. As soon as it becomes a cross, click and drag your mouse downwards for six cells. Release your mouse.

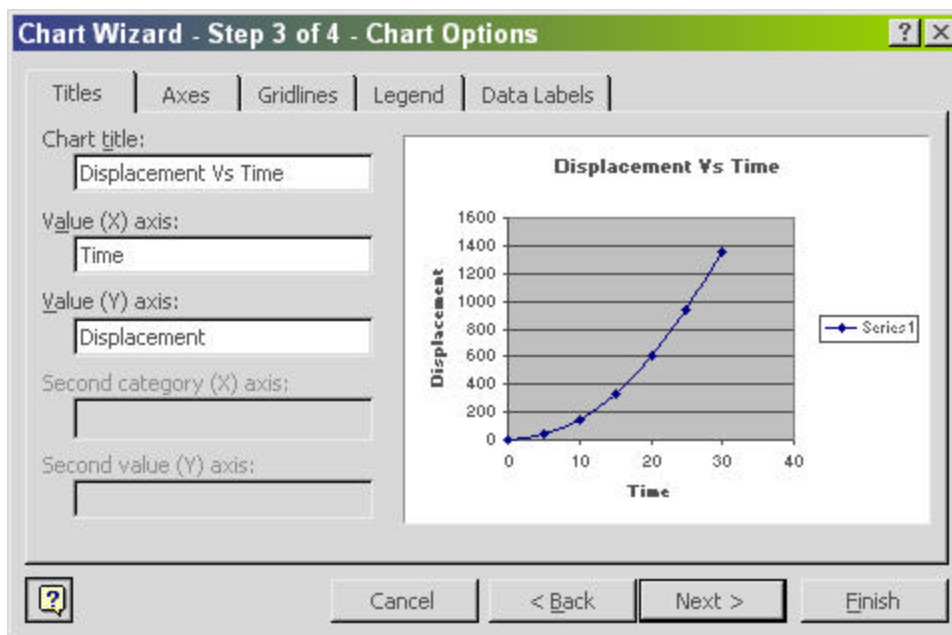
21. Click on the blue cell right below the **Displacement**, and insert the expression  $=1/2*(B2*H2)$ . This expression will give you the product of velocity (0 m/s<sup>2</sup>) and time (0 s), times 1/2. To get the other displacement values, click on the first light blue cell and move your cursor towards the lower right corner of the cell until it becomes a cross. As soon as it becomes a cross, click and drag your mouse downwards for six cells. Release your mouse.

**Figure 10**



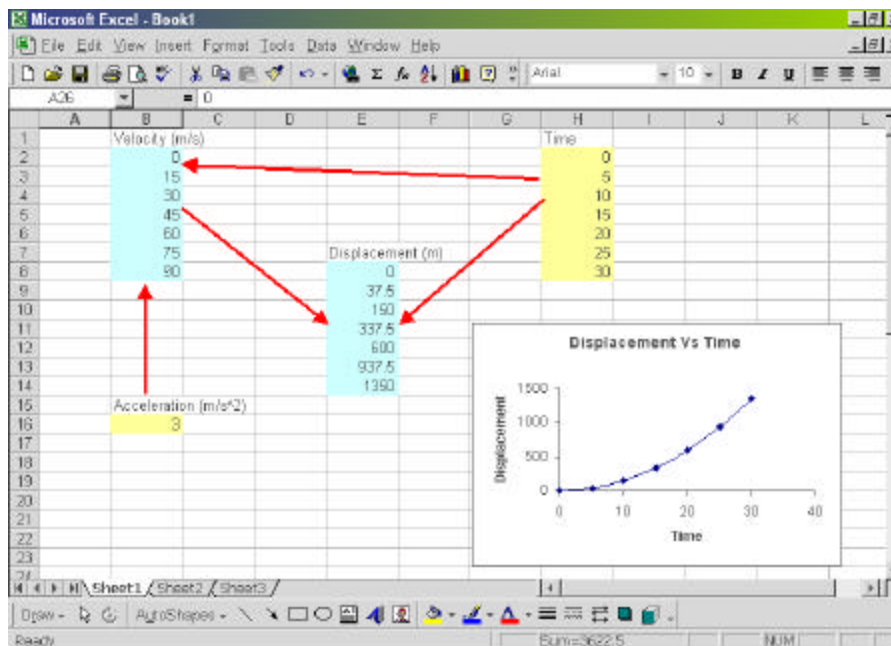
22. The model is now complete. However, what we see on the screen is not that informative. Therefore, a graphical output showing how displacement changes over time is really necessary. Highlight all the time values, press and hold the control key on your keyboard, and then highlight all the Displacement values. Release the control key after you finish highlighting.
23. Select the **Chart Wizard** from your Excel Toolbar. Click on a free space of your spreadsheet, hold your mouse's left button, drag it diagonally for 4-5 cm (depending on how big you want your graph to be), and then release the button. You should see a window and the chart wizard dialogue box on the screen. Select the Scatter Plot option.
24. Let the dialogue box guide you through until you get your graph (see figure 11).

Figure 11



25. By the end, your worksheet should look like figure 12.

Figure 12



26. Your simulation is now complete. Change the input value of the **Constant Acceleration**, and run your simulation again. What do you observe? How does the slope of the plot change? Is it consistent with what you learned in kinematics for

constant acceleration? Continue changing all the input values and describe how the graph changes.

## Activities

1. Describe the pros and cons of using spreadsheet features as a mapping language.
2. If you were supposed to create software that uses a mapping language for simulating, what features would you include that do not appear in spreadsheets?
3. If you had to plot velocity versus time in the last simulation of this project, what would the graph look like? Make the necessary arrangements to include this graph on the worksheet of figure 12.
4. Follow the aforementioned procedures to simulate the power of an electric circuit.
5. Follow the aforementioned procedures to simulate Hooke's Law. As an extension to Hooke's Law, simulate the potential energy of a spring (if you are not familiar with Hooke's Law visit the following websites  
<http://www.purchon.com/physics/hookeslaw.htm> and  
<http://www.sciencejoywagon.com/physicszone/lesson/02forces/hookeslaw.htm>).
6. Read the information given on  
<http://www.stolaf.edu/people/mckelvey/envision.dir/malthus.html> about Malthusian Growth Model and try to simulate the concepts involved by using spreadsheets (note that the instructions given for simulating the Malthusian Growth Model on this website refer to STELLA, a program that uses a mapping language for simulating. Thus, do not follow the instructions given on how to create the simulation. Get information regarding the theoretical background of the concepts you are interested in and follow the process introduced in this project).
7. Create a spreadsheet that is used as a units converter (e.g., convert temperature from Fahrenheit to Kelvin and Celsius, convert pounds to kilograms, convert miles to kilometers, etc). Try to use as many variables as possible (e.g., weight, length, temperature etc.).
8. Study the mathematical simulations given on  
<http://www.ties.k12.mn.us/envision/97/steve.html> and try to simulate the concepts involved by using spreadsheets (note that the instructions, diagrams, graphs, and equations given on this website refer to STELLA, a program that uses a mapping language for simulating. Thus, do not follow the instructions given on how to create the simulation. Get information regarding the theoretical background of the concepts you are interested in and follow the process introduced in this project). Based on the experience gained from this activity, create your own series of mathematical models.



9. Study the biology and environmental science simulations given on <http://indian.nehs.ce.k12.md.us/pages/deerpop.shtml> and try to simulate the concepts involved by using spreadsheets (note that the instructions, diagrams, graphs, and equations given on this website refer to STELLA, a program that uses a mapping language for simulating. Thus, do not follow the instructions given on how to create the simulation. Get information regarding the theoretical background of the concepts you are interested in and follow the process introduced in this project). For example, create a series of models that simulate the population model of different species (i.e., buffalos, zebras, snakes etc.). As an extension, create more complicated models that show parts of the food chain (e.g., combine the population model of rats, snakes, and hawks).
  
10. Study the simulations given on <http://indian.nehs.ce.k12.md.us/pages/compmod.shtml> and try to simulate the concepts involved by using spreadsheets (note that the instructions, diagrams, graphs, and equations given on this website refer to STELLA, a program that uses a mapping language for simulating. Thus, do not follow the instructions given on how to create the simulation. Get information regarding the theoretical background of the concepts you are interested in and follow the process introduced in this project). Based on the experience gained from this activity, create your own series of biological or environmental science simulations.
  
11. Study the physics simulations given on <http://isaac.williamsport.wa.k12.md.us/~ctrout/sciproj/STELLA/> and try to simulate the concepts involved by using spreadsheets (note that the instructions, diagrams, graphs, and equations given on this website refer to STELLA, a program that uses a mapping language for simulating. Thus, do not follow the instructions given on how to create the simulation. Get information regarding the theoretical background of the concepts you are interested in and follow the process introduced in this project). Based on the experience gained from this activity, create your own series of physics simulations.

## **Final Project/Report**

Create a spreadsheet simulating program that will be of use for a science project. Include a report explaining the reasoning behind your selection (purpose, benefits from its operation, etc.) and an analytical step-by-step description of your program. Describe the science project, and how your program is useful for it.



### **Unit 3: Computer Programming in Science Education – Pascal**

#### **Project 20**

## **Create Your Own Computer Programs: An Introduction to Pascal**

This project is intended as an introduction to the Pascal programming language. On the basis that the best way to learn a programming language is to practice writing plenty of programs in that language, we introduce Pascal structures, methods and disciplines and illustrate their use through sample Pascal programs. The disciplines learnt in these simple programs can then be applied in more elaborate programming circumstances, including creating programs useful to your science classes.

### **Degree of Difficulty**

Experimental: Moderate to Difficult

Conceptual: Moderate to Difficult

### **Prerequisite Knowledge**

Experience using a Windows operating system is required. Experience using DOS is recommended. Strongly suggested for students with experience in other programming languages, such as, Fortran and C/C<sup>++</sup>.

**NOTE:** This project refers to Pascal Version 1.0.4, which you can download for free from <http://www.brain.uni-freiburg.de/~klaus/fpc/fpc.html> (this does not mean that there is no other way to compile and run your programs). In case you want to work with other Pascal compilers visit <http://www.mit.edu/~taoyue/tutorials/pascal/compilers.html> (for Windows users Delphi is strongly suggested – For Free Pascal Compilers, Free Delphi Compilers, visit <http://www.thefreecountry.com/developercity/pascal.shtml>). There is useful information about installing them and running them. Make sure that you understand how to run Pascal programs on the compiler you choose, because it will be taken for granted for the purposes of this project. The coding/programs included in this project are compatible with all compilers. If you are confused about selecting a Pascal compiler, ask your teacher or Mentor for help.

### **Useful Information/Concepts**

- PASCAL is a programming language named after the 17th century mathematician Blaise Pascal.
- Pascal provides a teaching language that highlights concepts common to all computer languages, and standardizes the language in such a way that it makes programs easy to write.

- Strict programming rules.
- Words in Pascal: Pascal includes the use of "words", that is, strings of characters that identify elements within the program. There are many different kinds of words identifying variables, programs, and subprograms (known as procedures and functions). The programmer gets to give meaning to many of these, but some are predefined.
- Reserved Words - There are a number of words which have a fixed meaning, and cannot be redefined by the programmer (see table 1). Generally, these words identify program sections or program flow control structures (which we will cover later): **inherited, inline, interface, label, library, mod, nil, not, object, of, or, packed, and, asm, array, begin, case, const, constructor, destructor, div, procedure, program, record, repeat, set, shl, shr, string, then, to, type, unit, until, uses, var, while, with, xor, do, downto, else, end, exports, file, for, function, goto, if, implementation, and in.**
- Standard identifiers - Standard identifiers have a predefined meaning, but a programmer can override that definition with one of her or his own. While this is possible, it is not advisable, as this reduces "common understanding" of what a programmer was attempting at a given point in a program, making your program less maintainable (visit <http://www.mit.edu/~taoyue/tutorials/pascal/pas1b.html> for more details).
- Syntax diagrams: These are used throughout the text, and can be quite helpful to the visual learner. There are three symbols in these, connected by flow arrows. Once you are used to them, they are quite self-evident and are a good shorthand for remembering certain structures.
  - Items enclosed in an ellipse are: Reserved or Fully Defined
  - Items enclosed in a rectangle are: Defined Elsewhere
  - Separators are enclosed in a circle
- Identifiers: When a Turbo Pascal programmer is coming up with new words (or identifiers) she or he must follow only a couple simple rules. The identifier must be properly formed, and should have a meaningful name (visit <http://www.mit.edu/~taoyue/tutorials/pascal/pas1b.html> for more details on Identifiers and <http://www.mit.edu/~taoyue/tutorials/pascal/compilers.html> for more details on Turbo Pascal).
  - Rules for forming - an identifier must start with a letter or an underscore, can contain only letters, underscores, or numbers, and can have a maximum of 64 characters.
  - Meaningful names - Multiple words can be combined to form a more meaningful name: e.g, fname for a variable to hold a first name is

somewhat meaningful but `first_name` or `FirstName` is much more meaningful.

- Basic program components - Every Pascal program has the following three components:
  1. Program heading: `program identifier(input,output);` Note that this is probably the only identifier which you only type once, so you can use more of the 64 characters available (visit <http://www.mit.edu/~taoyue/tutorials/pascal/pas1b.html> for more details).
  2. Declaration section: `var const.` Things that will be used later on in your program are declared here.
  3. Executable section: `begin...end.` Note that the end is followed by a period. All the statements in between the **begin** and **end** are things that you want your program to do: receive information, change it somehow, or output it to the screen or printer.
- Writing code in Pascal:
  - Executable statement: An executable statement consists of valid identifiers, standard identifiers, reserved words, numbers and/or characters together with appropriate punctuation. In a sense, an executable statement is much like a phrase within a sentence.
  - Semicolon as a separator. Every executable statement needs to be terminated with a semicolon. There is only one exception to this rule, and that is where the statement is followed by the word **END**.
- Pascal is still extremely useful compared to other programming languages. For example, C and C<sup>++</sup> are very symbolic languages. Where Pascal chooses words (e.g. `begin-end`), C/C<sup>++</sup> chooses symbols (`{-}`). Also, C and C<sup>++</sup> are not strongly-typed languages. In Pascal, mixing types often led to an error. In C/C<sup>++</sup>, nothing would happen. To understand better the differences between Pascal and C/C<sup>++</sup> complete both this project and project 14.

## Objectives

Completion of the activities should enable you:

- to gain a basic understanding of programming theory, including: effective problem solving techniques, software development processes, and designing and analyzing simple algorithms.
- to become familiar with some basic constructs of the Pascal language, including: the language syntax, data types and expressions, input and output, and program control with conditional and looping constructs.
- to learn to how to edit, compile, debug, and run Pascal programs.

**Materials:** computer, a Pascal compiler.

## Part A

### Introduction

“The art of taking a problem and breaking it down into a set of instructions you can give a computer is the interesting part of programming. Unfortunately it is also the most difficult part of programming as well. If you think that learning to program is simply a matter of learning a programming language you are very wrong. In fact if you think that programming is simply a matter of coming up with a program which solves a problem you are equally wrong!” (Miles, 1995).

There are many things you must consider when writing a program. First, you have to make sure what the program is for. When considering how to write the specification of a system there are three important things :

- What data flows into the system.
- What flows out of the system.
- What the system does with the data.

Second, after defining and analyzing your program, you have to be aware of how to structure and create your Pascal program. According to Yue (2001), the basic structure of a Pascal program looks as follows:

**PROGRAM** ProgramName (FileList);

**CONST**

(\* Constant declarations \*)

**TYPE**

(\* Type declarations \*)

**VAR**

(\* Variable declarations \*)

(\* Subprogram definitions \*)

**BEGIN**

(\* Executable statements \*)

**END.**

The elements of a program must be in the correct order, though some may be omitted if not needed. Here's a program that does nothing, but has all the required elements:

**program Science;  
begin  
end.**

As you have probably noticed, there are three important issues yielding from the basic Pascal program structure that have to be addressed: nesting, comments, and spaces and end-of-lines.

Pascal comments start with a (\*) and end with a (\*). The compiler always matches the first (\*) with the first (\*), ignoring everything in between. If you use a nest looking like this (\* (\* \*) \*), you will get an error. The second \*) is left without its matching (\*) (Yue, 2001). To overcome this problem, make use of braces {}, which supersede parenthesis-stars (\* \*). You will not get an error, for example, if you do this: { (\* *Comment* \*) }, (Yue, 2001).

Commenting has two purposes: first, it makes your code easier to understand. If you write your code without comments, you may come back to it a year later and have a lot of difficulty figuring out what you've done or why you did it that way. Another use of commenting is to figure out errors in your program. When you don't know what is causing an error in your code, you can comment out any suspect code segments (Yue, 2001).

All spaces and end-of-lines are ignored by the Pascal compiler unless they are inside a string. However, to make your program readable by human beings, you should indent your statements and put separate statements on separate lines.

For creating your first simple Pascal program follow the next steps:

1. Create a file and type in the following commands:

```
(* My First Pascal Program*)

program Myfirst;

begin

writeln ('My First Pascal Program');

end.

(* End of Program *)
```

2. Open the Pascal compiler that is available to you and then compile the file you have created (Visit <http://www.mit.edu/~taoyue/tutorials/pascal/compilers.html> in case you do not know how to compile a file or ask your Teacher or Mentor for help.)
3. Run this program from the command prompt on your system. What do you observe?

This program simply says '**My First Pascal Program**' and terminates. Let's go through it line by line (see table 1):

**Table 1**

Code	Explanation
(* My First Pascal Program*)	Any text that occurs between a "(" and ")" is a comment and is ignored by the Pascal compiler. It can contain freeform text and is usually used to make the program more comprehensible to the reader.
program Myfirst;	This line, which identifies the name of the program as <b>first</b> , is purely decorative in modern versions of Pascal. It is used only for historical reasons.
begin	The <b>begin</b> statement indicates to the compiler that the processing which the program is to execute is to follow. Usually, before a <b>begin</b> statement will be found variable and constant declarations, subroutines and comments. In this case, the <b>begin</b> statement indicates the start of the main body of the program.
writeln ('My First Pascal Program');	The <b>writeln</b> command writes a line of text to the output device, usually the computer screen. The text to be written is then passed as a parameter to the command, in quotes and delimited by brackets. As with all Pascal commands, the <b>writeln</b> command is terminated with a semicolon ";". After writing the text passed to it as a parameter to the screen, <b>writeln</b> writes a newline character to the screen, causing the next line of text to be written on a new line. If you do not wish to "throw" to a new line, use the <b>write</b> command instead.
end.	The <b>end</b> command indicates the end of the main body of processing which commenced with the <b>begin</b> command. Where <b>end</b> is the last line of the program, indicating the completion of the block of code initiated by the <b>begin</b> command, it is always followed by a period ".". Where the <b>end</b> command indicates the end of a block of code which is not the end of the main sequence of commands, e.g., when it is completing a block of code relating to an <b>if</b> or <b>for</b> statement, it is not followed by a period.
(* End of Program *)	Once again, any material between the "(" and ")" is treated as a comment and is ignored by the compiler.

That's the end of your first Pascal program. Easy, right? Now let's consider getting into more detail and more complex programs. But first, it is important to review some of the theory behind the Pascal Variables.

According to Brown and Henry (1997), variables store values and information. They allow programs to perform calculations and store data for later retrieval. Variables store numbers, names, text messages, etc. Pascal supports FOUR standard variable types, which are:

**Table 2**

Variable type	Explanation	Examples
integer	Integer variables store whole numbers. In other words, numbers with no decimal points.	2, 878, -34667, 0, 12878
char	Character variables hold any valid character which is typed from the keyboard. In other words, letters, punctuation, digits, special symbols etc.	AAA, 90SS, My_First, ., GO;AWAY, [ ], { }, =, +, \,  , % , ( ), *, \$
Boolean	Boolean variables, also called logical variables, can only have one of two possible states, true or false.	True = 1, False = 0
real	Real variables are positive or negative numbers which include decimal places and/or exponents.	3.26, -3567.590, 456.997E+09

Note: char and Boolean are called ORDINAL types (This is because they have a limited, specified range of values), (Brown and Henry, 1997).

In Pascal, all variables must be explicitly declared. A variable is declared using the following format:

**var variable\_name : variable type;**

An example of declaring several variables is:

```
var
  age, year, grade : integer; (*Declares integer variable names age, year,
  grade*)
  volume : real; (*Declares a variable called volume which is a real number
  *)
  Letter : char; (*Declares a variable of type char*)
  DidYouPass : Boolean; (*Declares a variable called DidYouPass which is
  a Boolean value (true or false) *)
```

In a Pascal program, the aforementioned declaration of variables occurs after the program heading, and before the keyword **begin**:

**program MySecond (output);**



```

var
  number1: integer;
  number2: real;
  number3: real;
begin
  number1 := 3; { this makes number1 equal to 3 }
  number2 := 46.78; { this makes number2 equal to 46.78 }
  number3 := number1 + number2;
  writeln( number1, ' + ', number2, ' = ', number3 )
end.

```

This program declares one integer (number1), and two real numbers (number2 and number3). The assignment operator **:=**, after the **begin** statement, assigns values/expressions to the variables you have declared using the assignment operator. Unlike many other programming languages, Pascal does not use the equal sign, but a colon equal **:=** for assignment. The equal sign alone is used for comparison only.

The value/expression can either be a single value: **number1 := 3**; or it can be an arithmetic sequence: **number3 := number1 + number2**; The arithmetic operators in Pascal are:

**Table 3**

<i>Operator</i>	<i>Operation</i>	<i>Operands</i>	<i>Result</i>	<i>Example</i>
+	Addition or unary positive	real or integer	real or integer	<b>Program Add (output);</b> <b>var number1, number2, result</b> <b>: integer;</b> <b>begin</b> <b>number1 := 15;</b> <b>number2 := 29;</b> <b>result := number1 +</b> <b>number2;</b> <b>writeln(number1, " plus</b> <b>", number2, " is ", result )</b> <b>end.</b>
-	Subtraction or unary negative	real or integer	real or integer	<b>program Subtract (output);</b> <b>var number1, number2, result</b> <b>: integer;</b> <b>begin</b> <b>number1 := 78;</b> <b>number2 := 2;</b> <b>result := number1 -</b> <b>number2;</b> <b>writeln(number1, " minus</b> <b>", number2, " is ", result )</b> <b>end.</b>
	Multiplication	real or	real or	<b>program Multiply (output);</b>



**YSAP**<sup>®</sup>  
WWW.YSAP.ORG

YOUNG  
SCIENCE  
ACHIEVERS  
PROGRAM

<b>*</b>		integer	integer	<b>var number1, number2, result : integer;</b> <b>begin</b> <b>number1 := 18;</b> <b>number2 := 27;</b> <b>result := number1 * number2;</b> <b>writeln(number1, " multiplied by ", number2, " is ", result )</b> <b>end.</b>
<b>/</b>	Real division	real or integer	real	<b>program Divide (output);</b> <b>var number1, number2, result : integer;</b> <b>begin</b> <b>number1 := 87.9;</b> <b>number2 := 9.3;</b> <b>result := number1 / number2;</b> <b>writeln(number1, " divided by ", number2, " is ", result )</b> <b>end.</b>
<b>div</b>	Integer division	integer	integer	<b>program Divide (output);</b> <b>var number1, number2, result : integer;</b> <b>begin</b> <b>number1 := 27;</b> <b>number2 := 9;</b> <b>result := number1 div number2;</b> <b>writeln(number1, " divided by ", number2, " is ", result )</b> <b>end.</b>
<b>mod</b>	Modulus (remainder division)	integer	integer	<b>program MODULUS (output);</b> <b>var number1, number2, number3 : integer;</b> <b>begin</b> <b>number1 := 3;</b> <b>number2 := 10;</b> <b>number3 := number2 MOD number1;</b> <b>writeln( number2:2,' modulus',number1:2,'</b>



				<b>is',number3:2) end.</b>
--	--	--	--	--------------------------------

Note: In case you want to get more information regarding Standard Mathematical Functions (i.e., absolute value, arctan in radians, cosine of a radian measure, etc.) visit <http://www.mit.edu/~taoyue/tutorials/pascal/pas1f.html>.

Now let's attempt to create another Pascal program that combines the aforementioned:

4. Open the Pascal compiler that is available to you and type in the following commands:

```
program MySecond (output);  
  var   age: integer;  
        haircolor: char;  
        weight : real;  
  begin  
    age := 15;  
    haircolor := 'Brown';  
    weight := 70.5;  
    writeln('age = ', age );  
    writeln('haircolor = ', haircolor );  
    writeln(' weight  = ', weight )  
  end.
```

5. Run this program from the command prompt on your system. What do you observe?

## Activities

1. Run the Pascal programs given in table 3. In addition, create you own programs for each one of the operators.
2. Create a Pascal program that shows the difference between “/” and **div**.
3. Create a Pascal program that calculates the area of:
  - Squares
  - Rectangles
  - Triangles
  - Hexagons
4. Create a Pascal program that calculates the volume of:
  - Cube
  - Box

- Cylinder
- Pyramid
- Sphere

5. Create a Pascal program that converts units (i.e., from feet to meters, from Fahrenheit to Celsius, etc.)
6. Create a Pascal program that makes any calculation you want.
7. If you rearrange the lines of the given programs, will the programs run? What if you discard one of the semicolons?
8. Are the following valid variable declarations?

```
var day, month : integer;  
time : real;
```

```
var time : real;  
day : integer;  
month : integer;
```

9. Write a program which calculates and prints on the screen, the time required to travel 300 miles at a speed of 60 mph.
10. Write a program to calculate the gross pay for a worker named Zach given that Zach worked 60 hours at \$10.50 per hour.
11. What is displayed by the following program?

```
program Problem1 (output);  
var a, b : integer;  
c : real;  
begin  
a := 11; b := 9; c := 4.27;  
writeln('A = ', a + 3 );  
writeln('B = ', b - 2 );  
writeln('C = ', c / 2 )  
end.
```

12. Create a Pascal program that sums up and averages the following five integers: 49, 34, 28, 37, 25.
13. Predict what particular function the following program, developed by Yue (2001), perform before running them. Run the program and explain any discrepancies between what you observe and what you predicted.

```

program SumAverage;

const
    NumberOfIntegers = 5;

var
    A, B, C, D, E : integer;
    Sum : integer;
    Average : real;

begin  (* Main *)
    A := 45;
    B := 7;
    C := 68;
    D := 2;
    E := 34;
    Sum := A + B + C + D + E;
    Average := Sum / 5;
    writeln ('Number of integers = ', NumberOfIntegers);
    writeln ('Number1 = ', A);
    writeln ('Number2 = ', B);
    writeln ('Number3 = ', C);
    writeln ('Number4 = ', D);
    writeln ('Number5 = ', E);
    writeln ('Sum = ', Sum);
    writeln ('Average = ', Average)
end.

```

## Part B

In this part we will attempt to get in more depth by considering a wider range of commands we can use. More specifically, we will focus on **conditional statements (if)**, which are one of the main parts of a Pascal program. Pay special attention to the notes listed below before proceeding with the directions included in each step.

## Procedure

### Getting Information from the Keyboard into a Program

It is convenient to accept data while a program is running. The **read** and **readln** statements allow you to read values and characters from the keyboard, placing them directly into specified variables (Brown and Henry, 1997). To better understand this, run the following program:

- 1 Create a file and type in the following commands:

```

program READDEMO (input, output);
var   number1, number2 : integer;
begin
        writeln('Please enter two numbers separated by a space');
        read( number1 );
        read( number2 );
        writeln;
        writeln('Number1 is ', number1 , ' Number2 is ', number2 )
    end.

```

2. Open the Pascal compiler that is available to you and then compile the file you have created.
3. Run this program from the command prompt on your system. What do you observe?
4. When run, the program will display the message “**Please enter two numbers separated by a space.**” Type two numbers (any numbers), and then press the return key. What do you observe?
5. Replace the **readln** statement with **read** and run the program.
6. When run, the program will display the message “**Please enter two numbers separated by a space.**” Type two numbers (any numbers), and then press the return key. What do you observe? Is it different from the result of step 3?
7. Run the program of step 4 once again. Type a number (any number), and then press the return key. Proceed with typing another number (any number), and then press the return key. What do you observe? Is it different from the result of steps 3 and 5? What can you say about the differences between the statement **readln** and **read**?
8. For more details regarding the statements **readln** and **read** visit <http://www.mit.edu/~taoyue/tutorials/pascal/pas2a.html>.

### **Specifying the Display format for the Output Variables**

According to Brown and Henry (1997), when variables are displayed, Pascal assigns a specified number of character spaces (called a field width) to display them. The field widths for the various data types are,

```

INTEGER - Number of digits + 1 { or +2 if negative }
CHAR    - 1 for each character
REAL    - 12
BOOLEAN - 4 if true, 5 if false

```

Often, the allotted field size is too big for the majority of display output. Pascal provides a way in which the programmer can specify the field size for each output. The syntax to display scientific notation in a specified field width is:

**'text string':fieldsize, variable:fieldsize**

The output is right-justified in a field of the specified integer width. If the width is not long enough for the data, the width specification will be ignored and the data will be displayed in its entirety (Yue, 2001). For example, if you include in a Pascal program **writeln('Hi!':10,'Hi!!':10,'Hi!!!');**, the display output will be as follows,

Hi!.....Hi!!.....Hi!!!

where ... indicates a space.

Note that to specify the field width of text or a particular variable, use a colon (:) followed by the field size.

## **Making Decisions**

Most programs are designed to make decisions, including Pascal. There are several statements available in the Pascal language for this, and the **IF**, **IF THEN**, and **IF THEN ELSE** statements are three of them. Along with these three statements, there are relational operators, some of which are listed in table 4 (visit <http://www.mit.edu/~taoyue/tutorials/pascal/pas3b.html> for more information), that allow the programmer to test various variables against other variables or values.

**Table 4**

<b>Operators</b>	<b>Explanation</b>
<	less than
>	greater than
=	equal to
<=	less than or equal to
>=	greater than or equal to
<>	not equal to

The format for the IF THEN Pascal statement is,

**if condition\_is\_true then**  
**execute\_this\_program\_statement;**

The function of this statement involves evaluating a certain condition to see if it's true. When the condition is true, the program statement will be executed. If the condition is not true, then the program statement following the keyword **then** will be ignored. See the next step for an example.



9. Create a file and type in the following commands:

```
program IFTHEN (input, output);
  var   number, age : integer;
  begin
    number := 15;
    writeln('Guess my age! Select a number between 10 and 20');
    readln( age );
    if number = age then writeln('Well Done!');
    if number <> age then writeln('Wrong guess.')
  end.
```

10. Open the Pascal compiler that is available to you and then compile the file you have created.
11. Run this program from the command prompt on your system. As soon as you run the program you will be asked to enter a number between 10 and 20. Enter the number 11 and press the return key. What do you observe? Now, enter the number 15 and press the return key. What do you observe?

### **Executing more than one statement as part of an IF**

To execute more than one program statement when an **if** statement is true, the program statements are grouped using the **begin** and **end** keywords (Brown and Henry, 1997). Whether a semi-colon follows the **end** keyword depends upon what comes after it. When followed by another **end** or **end.** there is no semi-colon. See the next step for an example.

12. Create a file and type in the following commands:

```
program IFTHEN (input, output);
  var   number, age : integer;
  begin
    number := 15;
    writeln('Guess my age! Select a number between 10 and 20');
    readln( age );
    if number = age then
      begin
        writeln('Well Done!');
        writeln('How did you know?')
      end;
    if number <> age then writeln('Wrong guess.')
  end.

program IFTHEN2 (input, output);
```

```

var    number, age : integer;
begin
    number := 15;
    writeln('Guess my age! Select a number between 10 and 20');
    readln( age );
    if number = age then
        begin
            writeln('Well Done!');
            writeln('How did you know?')
        end
    end.

```

13. Open the Pascal compiler that is available to you and then compile the file you have created.
14. Run the program. What do you observe when you enter number 15? What do you observe when you enter any number, but 15? How is this program different from the program of step 9?

The IF statement can also include an ELSE statement, which specifies the statement (or block or group of statements) to be executed when the condition associated with the IF statement is false (Brown and Henry, 1997). For comparing it with the previous two conditional statements, let's consider an example.

15. Rewrite the program of step 12 using an **IF THEN ELSE** statement, as follows

```

program IF_THEN_ELSE_DEMO (input, output);
var    number, age : integer;
begin
    number := 15;
    writeln('Guess my age! Select a number between 10 and 20');
    readln( age );
    if number = age then
        writeln('Well Done!')
    else
        writeln('Wrong guess.')
    end.

```

```

program IFTHEN (input, output);
var    number, age : integer;
begin
    number := 15;
    writeln('Guess my age! Select a number between 10 and 20');
    readln( age );
    if number = age then writeln('Well Done!');
    if number <> age then writeln('Wrong guess.')

```

**end.**

16. Run the program. What do you observe when you enter number 15? What do you observe when you enter any number, but 15? Is this program different from the program of step 9 and 12?

In case you want to execute more than one statement when a condition is true (or false for that matter). Consider the following portion of code,

```
if number = age then
begin
    writeln('Well Done!');
    writeln('How did you know?')
end
else
    begin
        writeln('Wrong guess. ');
        writeln('Better luck next time')
    end; {semi-colon depends on next keyword}
```

### The AND, OR, and NOT Statements.

The AND, OR and NOT keywords are used where you want to execute a block of code (or statement) when more than one condition is necessary (Brown and Henry, 1997).

**Table 5**

Operator	Purpose	Example
AND	The statement is executed only if BOTH conditions are true.	if (A = 0) AND (B = 1) then writeln('Good Job!');
NOT	Converts TRUE to FALSE, and the opposite	if NOT ((A = 0) AND (B = 1)) then writeln('Try again!');
OR	The statement is executed if EITHER argument is true.	if (A = 0) OR (B = 1) then writeln('You are Close!');

Note: For more information regarding conditional statements, visit <http://www.mit.edu/~taoyue/tutorials/pascal/pas3ca.html>, and <http://www.mit.edu/~taoyue/tutorials/pascal/pas3b.html>.

Another way of using the AND, OR, and NOT statements is in combination with Boolean variables (see table 6).

**Table 6**

Expression	Returns
$x = ((0 \leq \text{testvar}) \text{ AND } (\text{testvar} \leq 9))$	$x = \text{TRUE}$ if testvar is in the range 0 to 9
$y = ((0 > \text{testvar}) \text{ OR } (\text{testvar} > 9))$	$y = \text{TRUE}$ if testvar is less than 0 or greater than 9
$z = (\text{NOT}(\text{testvar} = 7))$	$z = \text{FALSE}$ if testvar = 7

In the above examples, it is assumed that x, y, and z have all been declared to be Boolean variables. Note the importance of the parentheses in deciding the sequence of evaluation.

When checking for equality in a test, it is important not to inadvertently use the assignment ( $:=$ ) operator instead checking for equality ( $=$ ). The compiler will show an error if this mistake is made, e.g.,

```

program EQUALITY;
var a: integer;
begin
    a := 3;
    if (a = 99) then writeln ('This writeln command will never be executed');
    if (a = 3) then writeln ('a has the value 3');
    { if (a := 3) then writeln ('a has the value 3'); This command would
      throw up a compiler error }
end.

```

For completeness, where the variables A and B both evaluate to TRUE or FALSE, the result of the AND, OR and NOT operators on the variables is shown below:

**Table 7**

A	B	A AND B	A OR B	NOT(A)	NOT(B)
FALSE	FALSE	FALSE	FALSE	TRUE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE	FALSE
TRUE	FALSE	FALSE	TRUE	FALSE	TRUE
TRUE	TRUE	TRUE	TRUE	FALSE	FALSE

The most common use of binary operators is in combining comparisons. Usually the result of the comparison will result in a program taking one or another course of action depending on the result of the comparison, thereby regulating the flow of control of the program. An example of the use of binary operators is shown in the next step.

17. Open the Pascal compiler that is available to you and type in the following statements:

```

program precedence3;
var a, b : Boolean;
begin
    a := TRUE;
    b := FALSE;
    writeln ('a is ',a,' b is ',b);
    writeln ('a AND b gives ', a AND b);
    writeln ('a OR b gives ', a OR b);
    writeln ('NOT a gives ', NOT(a),' NOT b gives ', NOT(b));
    writeln ('NOT (a OR b) gives ', NOT (a OR b));
    writeln ('(NOT a) OR b gives ', (NOT a) OR b);
    writeln ('NOT (a AND b) gives ', NOT (a AND b));
    writeln ('(NOT a) AND b gives ', (NOT a) AND b);
end.

```

18. Run the program and explain its function.

## Activities

1. Write a Pascal program that

- sums the two integer variables **A** and **B** into the variable **Speed**
- sums the two integer variables **A** and **B** into the variable **Speed**, and then subtracts the value 45.8 from the variable **Speed**
- displays the value of the integer variable **acceleration**
- reads in a character value into the variable **grammar**

Note: All programs must write out the values, so it can be checked.

2. Write a Pascal statement which

- compares the integer variable **sum** to the constant value 25, and if it is the same, prints the string "You are correct!"
- compares the character variable **A** to the character variable **B**, and if it is not the same, prints the value of **A**
- compares the character variable **weight** to the constant **W**, and if less, prints the text string "Too light", otherwise print the text string "Too heavy"
- displays the value of the real variable **degrees** using a fieldwidth of 5 with three decimal places

3. Ohm's law states that the voltage (V) in a circuit is equal to the current (I) flowing in the circuit multiplied by the resistance in the circuit (R). Write a program to enter the values of resistance and current, displaying the voltage which would exist.

4. Write a Pascal program that displays the AND, NOT, OR combinations of the Boolean variables 1 and 0 shown in the following table.

A	B	A AND B	A OR B	NOT(A)
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	0

Hint: See step 14.

5. Write a Pascal program that gives you the average speed of a plane for a given distance and time traveled (average speed = distance traveled / time traveled). The program must allow the user to enter in the desired values of distance and time, and display the resulting average speed. Expand this program by including more equations from kinematics (i.e., velocity = acceleration x time).  
Note: The program should also protect itself against a value of zero for the time.
6. Write a program which gets two resistance values, and then displays their equivalent value when placed in series and then in parallel [the total series resistance is  $R1 + R2$ , and the parallel resistance is  $(R1 * R2) / (R1 + R2)$ ].
7. Write a Pascal statement that displays the value of the real variable **Average\_Speed** using a field width of five places, nine places, and 13 places.
8. Write a Pascal statement that displays the characters from A to K using a field width of three places.
9. What is displayed when the following program created by Brown and Henry (1997) is executed? Examine the program to predict the result before running the program.

```

program IF_THEN_ELSE_TEST (output);
var   a, b, c, d : integer;
begin
    a := 5; b := 3; c := 99; d := 5;
    if a > 6 then writeln('A');
    if a > b then writeln('B');
    if b = c then
        begin
            writeln('C');
            writeln('D')
        end;
    if b <> c then writeln('E') else writeln('F');
    if a >= c then writeln('G') else writeln('H');
    if a <= d then
        begin

```

```
writeln('I');
writeln('J')
end
end.
```

10. Create a program that shows the amount of money in your account, calculates your monthly expenses, subtracts your expenses from your account, and notifies you when half of your income is spent, one fourth of it is spent, and when there is no more money left in your account.
11. Use an **if** statement to compare the value of an integer called “price” against the value 20, and if it is more, print "Expensive".
12. Use an **if else** statement to create a Pascal program which categorizes products according to their weight. Use as categories the words “Extremely Heavy”, “Very Heavy”, “Heavy”, “Normal”, “Light”, and “Very Light”. You are responsible for defining the corresponding weight ranges. Compare the value of an integer called “weight” against the weight ranges, and print the corresponding characterization (i.e., Heavy, Normal, Light etc.).
13. Write a program which gets two values, call them LENGTH and WIDTH. Print the value of the largest variable.
14. Modify the program you wrote for activity 6, to accept three values, LENGTH, WIDTH, and HEIGHT. Print the value of the largest variable.
15. What is displayed after the following program created by Brown and Henry (1997) is executed? Examine the program to predict the result before running the program.

```
program AND_OR_NOT_DEMO (output);
var   a, b, c : integer;
begin
    a := 5; b := 3; c := 99;
    if (a = 5) or (b > 2) then writeln('A');
    if (a < 5) and (b > 2) then writeln('B');
    if (a = 5) and (b = 2) then writeln('C');
    if (c <> 6) or (b > 10) then writeln('D') else writeln('E');
    if (b = 3) and (c = 99) then writeln('F');
    if (a = 1) or (b = 2) then writeln('G');
    if not( (a < 5) and (b > 2)) then writeln('H')
end.
```



16. Create a Pascal program that calculates the total area of houses that have 3 bedrooms, 2 bathrooms, 1 kitchen, 1 sitting room, 1 hallway, and 1 dining room. In addition, create a separate C program that calculates the total volume, as well as the area. Compare the results from the two programs.
17. Assume that you were hired by a realtor's office to create a Pascal program which calculates the total area of houses as in activity 7. This time, though, you have to categorize houses according to the area they occupy. The program must also have the capability to list the houses that fall within particular ranges (i.e., 300 m<sup>2</sup> – 500 m<sup>2</sup>).
18. Predict what particular function this program performs before running the program. Run the program and explain any discrepancies between what you observe and what you predicted.

```

program temperature_conversion;
var celsius, fahrenheit : real;
var option : char;
begin
    writeln ('Enter Option : ');
    writeln ('C to convert Celsius to Farenheit ');
    writeln ('F to convert Farenheit to Celsius ');
    readln (option);
    if ((option = 'c') OR (option = 'C')) then
        begin
            writeln ('Enter temperature in Celsius ');
            readln (celsius);
            fahrenheit := 32 + (celsius * 9/5);
            writeln (celsius:4:2, ' Celsius is ',fahrenheit:4:2,' Farenheit equivalent');
            end
            else if ((option = 'f') OR (option = 'F')) then
                begin
                    writeln ('Enter temperature in Farenheit ');
                    readln (fahrenheit);
                    celsius := (fahrenheit - 32) * 5 / 9;
                    writeln (fahrenheit:4:2, ' Farenheit is ',celsius:4:2,' Celsius equivalent');
                    end
        end.

```

19. Based on activity 16, create a Pascal program that converts weight units (i.e., kilos to pounds and vice versa). Expand your program by including conversions of other physical units.
20. In case you want to get in more depth regarding programming with Pascal, visit the websites included in the references.



## **Final Project/Report**

Create a Pascal program that will be of use for a science project. Include a report explaining the reasoning behind your selection (purpose, benefits from its operation, etc.) and an analytical step by step description of your program. Describe the science project, and how your program is useful for it.

## **References**

Brown, B. and Henry, P (1997). *Pascal Programming Online Notes*. URL:  
<http://msnhomepages.talkcity.com/LibraryLawn/brownbr/pascal/pstart.htm>

Yue, T. (2001). *Learn Pascal*.  
URL: <http://www.mit.edu/~taoyue/tutorials/pascal/contents.html>