

SCHOOL OF COMPUTING

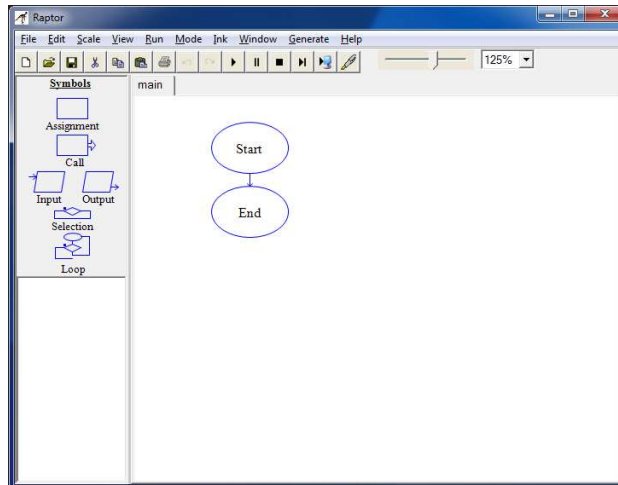
ST0502 Fundamentals of Programming

Practical 1: Problem Solving Concepts

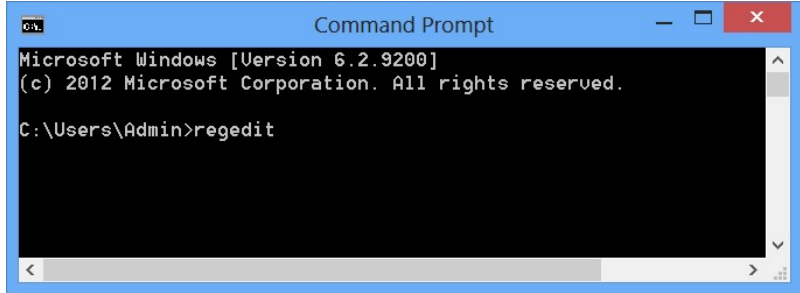
What you will learn / do in this lab

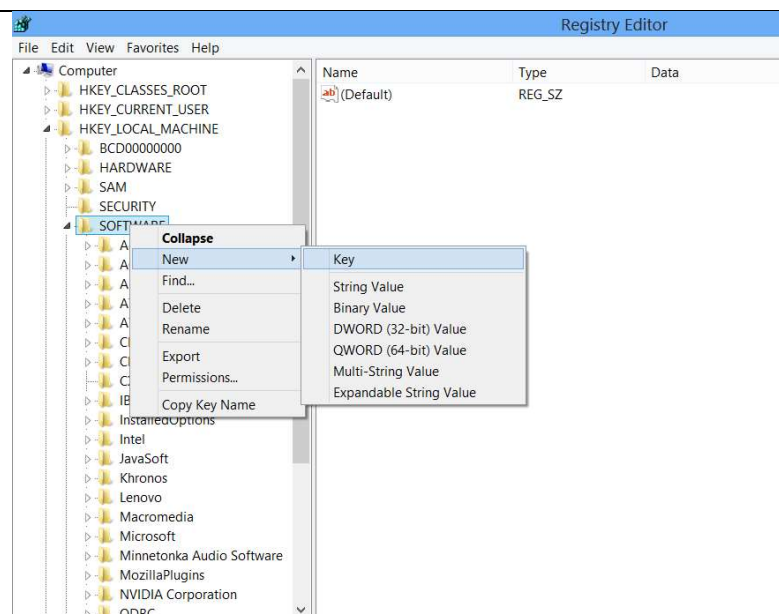
- Install Raptor - a flowchart tool
- Use Raptor and pseudo-code to develop solution to a problem

1. Download and install Raptor
 - a. Raptor is a flowchart tool we will be using for drawing flowcharts. Download Raptor from <http://raptor.martincarlisle.com> by clicking on **Download Latest Version** button.
 - b. Install the file downloaded in the previous step.
 - c. Test Raptor by running the program. You should see this screen below and you may close the program if successful.



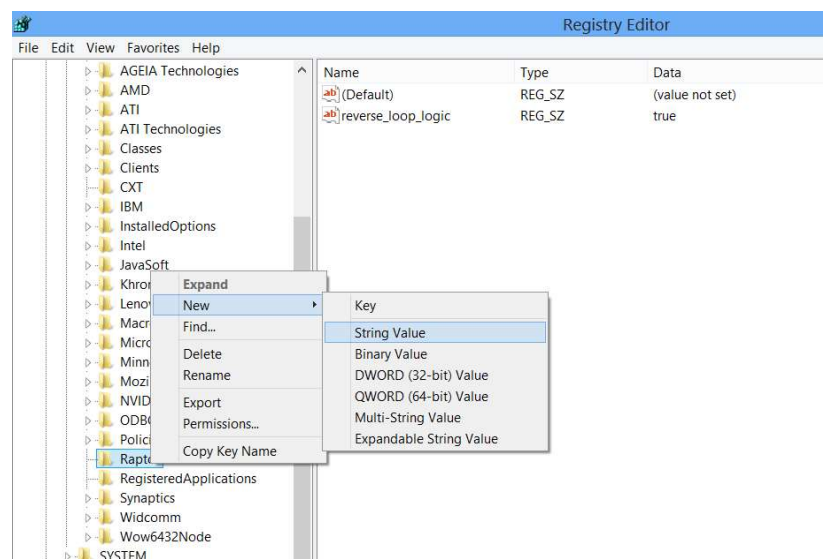
- d. Close Raptor and do the setting of Raptor by following the steps below:

Step 1:	<p>Open the Command Prompt, in the Command Prompt, type in regedit to edit the registry entry for Raptor.</p> 
Step 2:	<p>If you do not see the Raptor key under HKEY_LOCAL_MACHINE\Software, create one by clicking New → Key, and enter Raptor.</p>

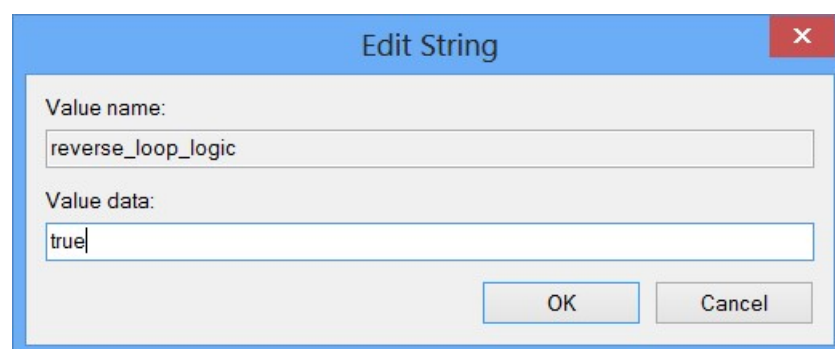


Step
3:

In the **Raptor** key, right-click and create a **new String** value



and enter **reverse_loop_logic** as Value name and set the value data to be **true**.

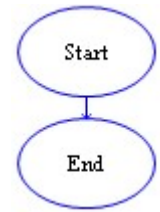


Click OK and close the Registry Editor.

2. Let's take some time to understand Raptor statements/symbols and how it works:

RAPTOR Program Structure

A RAPTOR program is a set of connected symbols that represent actions to be performed. The arrows that connect the symbols determine the order in which the actions are performed. When executing a RAPTOR program, you begin at the **Start** symbol and follow the arrows to execute the program. A RAPTOR program stops executing when the **End** symbol is reached. The smallest RAPTOR program (which does nothing) is depicted at the right. By placing additional RAPTOR statements between the **Start** and **End** symbols you can create meaningful RAPTOR programs.



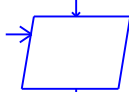

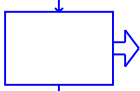
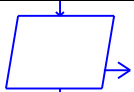
Introduction to RAPTOR Statements/Symbols

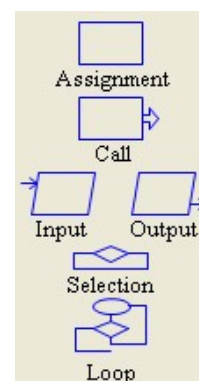
RAPTOR has six (6) basic symbols, where each symbol represents a unique type of instruction. The basic symbols are shown at the right.

The typical computer program has three basic components:

- INPUT – get the data values that are needed to accomplish the task.
- PROCESS – manipulate the data values to accomplish the task.
- OUTPUT – display the values which provide a solution to the task.

These three components have a direct correlation to RAPTOR instructions as shown in the following table.

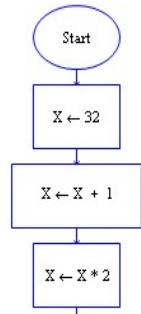
Purpose	Symbol	Name	Description
INPUT		input statement	Allow the user to enter data. Each data value is stored in a variable .
PROCESSING		assignment statement	Change the value of a variable using some type of mathematical calculation.
PROCESSING		procedure call	Execute a group of instructions defined in the named procedure.
OUTPUT		output statement	Display the output, it can display the value of variables .



They all do something to variables! You must understand the concept of a variable.

RAPTOR Variables

Variables are computer memory locations that hold a data value. At any given time a variable can only hold a single value. However, the value of a variable can vary (change) as a program executes. That's why we call them "variables"! As an example, study the following table that traces the value of a variable called X.

Description	Value of X	Program
<ul style="list-style-type: none"> When the program begins, no variables exist. In RAPTOR, variables are automatically created when they are first used in a statement. 	Undefined	
<ul style="list-style-type: none"> The first assignment statement, $X \leftarrow 32$, assigns the data value 32 to the variable X. 	32	
<ul style="list-style-type: none"> The next assignment statement, $X \leftarrow X + 1$, retrieves the current value of X, 32, adds 1 to it, and puts the result, 33, in the variable X. 	33	
<ul style="list-style-type: none"> The next assignment statement, $X \leftarrow X * 2$, retrieves the current value of X, 33, multiplies it by 2, and puts the result, 66, in the variable X. 	66	

During the execution of the previous example program, the variable X stored three distinct values. **Please note that the order of statements in a program is very important.** If you re-ordered these three assignment statements, the values stored into X would be different.

All variables should be given meaningful and descriptive names by the programmer. Variable names should relate to the purpose the variable serves in your program. A variable name must start with a letter and can contain only letters, numerical digits, and underscores (but no spaces or other special characters).

IMPORTANT: If you give each value in a program a meaningful, descriptive variable name, it will help you think more clearly about the problem you are solving and it will help you find errors in your program.

When a RAPTOR program begins execution, no variables exist. The first time RAPTOR encounters a new variable name, it automatically creates a new memory location and associates this variable name with the new memory. The variable will exist from that point in the program execution until the program terminates. When a new variable is created, its initial value determines whether the variable will store numerical data or textual data. **This is called the variable's data type. A variable's data type cannot change during the execution of a program.** In summary, variables are automatically created by RAPTOR and can hold either:

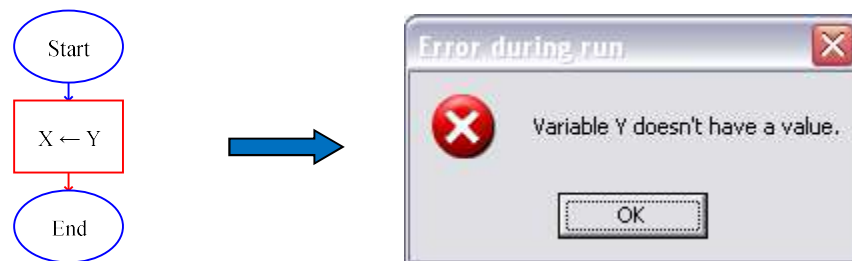
- Numbers e.g., 12, 567, -4, 3.1415, 0.000371, or
- Strings e.g., "Hello, how are you?", "James Bond", "The value of x is "

Common errors when using variables:

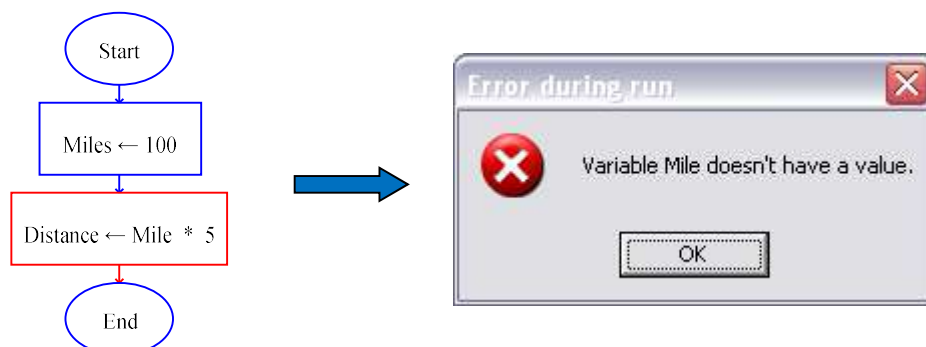
Error 1: "Variable does not have a value"

There are two common reasons for this error:

- 1) The variable has not been given a value.



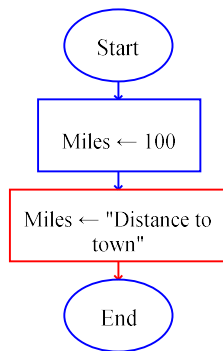
- 2) The variable name was misspelled.



Error 2: "Can't assign string to numeric variable"

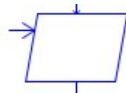
"Can't assign numeric to string variable"

This error will occur if your statements attempt to change the data type of a variable.



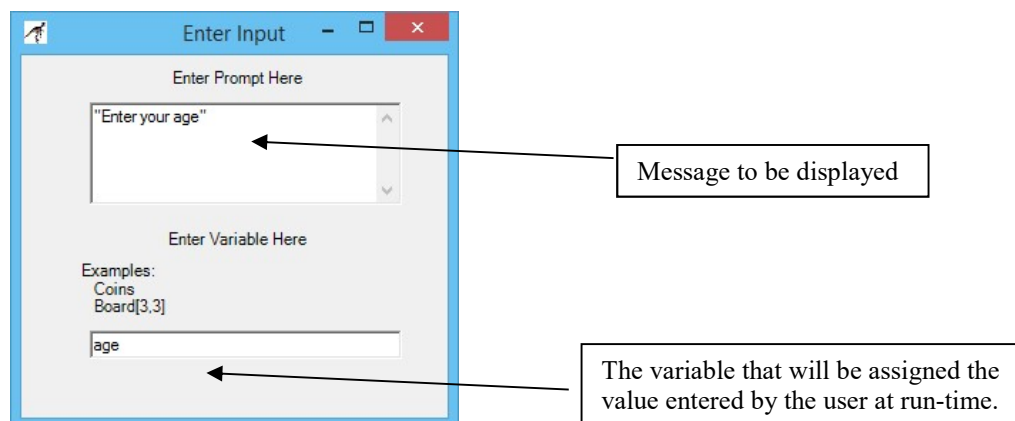
RAPTOR Statements/Symbols

Input Statement/Symbol

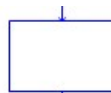


An input statement/symbol allows the user of a program to enter a value into a program variable during program execution. **It is important that a user know exactly what type of value is expected for input.** Therefore, when you define an input statement you specify a string of text that will be the *prompt* that describes the required input.

When you define an input statement, you must specify two things: **a prompt and the variable that will be assigned the value enter by the user at run-time.**



Assignment



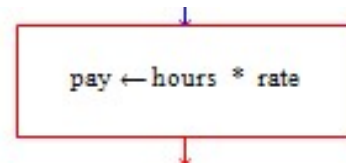
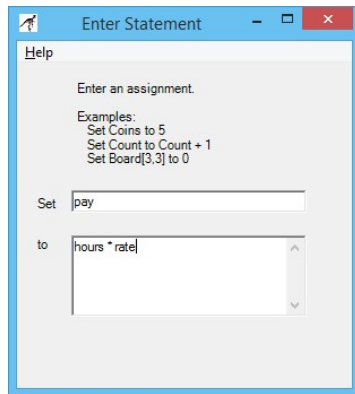
Statement/Symbol

The assignment symbol is used to perform a computation and then store the results in a variable. The variable to be assigned a value is entering into the **"Set"** field, and the computation to perform is enter into the **"to"** field.

An assignment statement is displayed inside its RAPTOR symbol using the syntax:

Variable ← Expression

For example, the statement created by the dialog box below is displayed as:



One assignment statement can only change the value of a single variable, that is, the variable on the left hand side of the arrow. If this variable did not exist prior to the statement, a new variable is created. If this variable did exist prior to the statement, then its previous value is lost and its new value is based on the computation that is performed. No variables on the right hand side of the arrow (i.e., the expression) are ever changed by the assignment statement.

Expressions

The expression (or computation) of an assignment statement can be any simple or complex equation that computes a single value. An expression is a combination of values (either constants or variables) and operators.

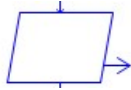
The following table briefly shows the frequently used operators.

Operation	Description	Example
+	addition	3+4 is 7
-	subtraction	3-4 is -1
-	negation	-3 is a negative 3
*	multiplication	3*4 is 12
/	division	3/4 is 0.75

The result of evaluating of an expression in an assignment statement must be either a single number or a single string of text. Most of your expressions will compute numbers, but you can also perform simple text manipulation by using a plus sign (+) to join two or more strings of text into a single string. You can also join numerical values with strings to create a single string. The following example assignment statements demonstrate string manipulation.

```
Full_name ← "Joe " + "Alexander " + "Smith"
Answer ← "The average is " + (Total / Number)
```

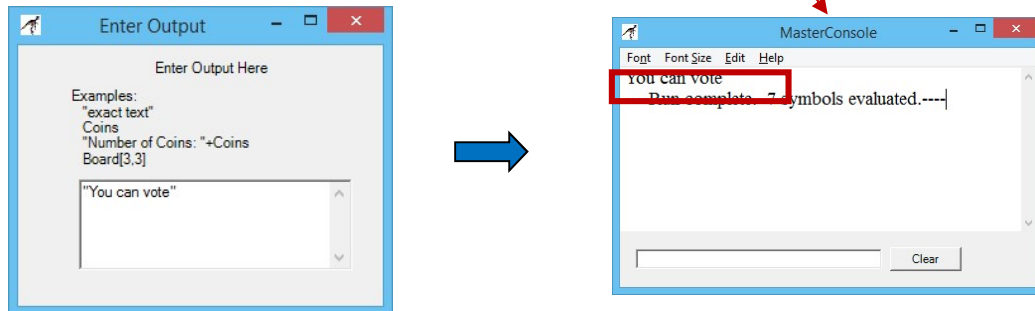

Output Statement/Symbol



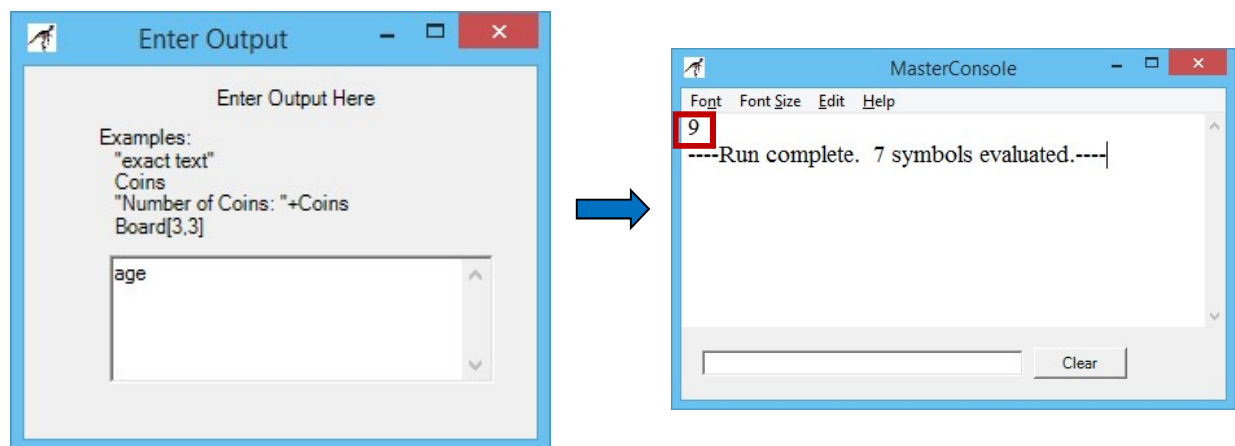
In RAPTOR, an output statement displays a value to the MasterConsole window when it is executed.

Here are two examples:

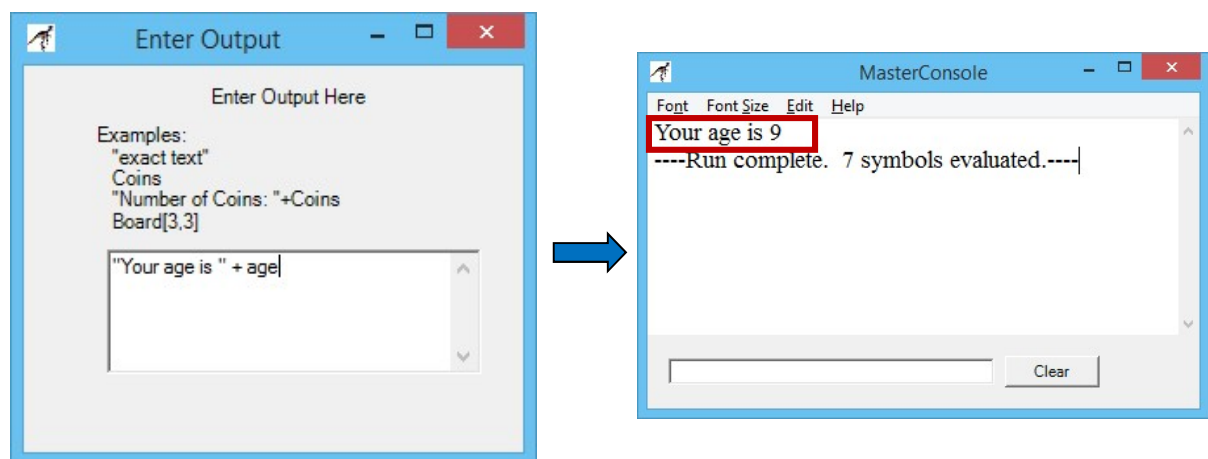
Example 1 – display texts



Example 2 – display the value of a variable *age* (age is a variable that has value, such as 9)



Example 3 – display texts + the value of a variable *age*



Take note that you can display multiple values with a single output statement by building a string of text using the string plus (+) operator. When you build a single string from two or more values, you must distinguish the text from the values to be calculated by enclosing any text in quote marks ("). In such cases, **the quote marks are not displayed in the output window**. For example, the expression,

"Your age is " + age

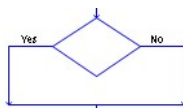
will display as :

Your age is 9

Notice that the quote marks are not displayed on the output device. The quote marks are used to surround any text that is not part of an expression to be evaluated.

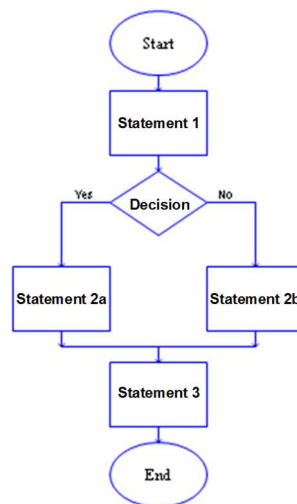
RAPTOR Control Commands

Selection Control



It is common that you will need to make a decision about some condition of your program's data to determine whether certain statements should be executed.

A *selection-control statement* allows you to make "decisions" in your code about the current state of your program's data and then to take one of **two** alternative paths to a "next" statement. The RAPTOR code below illustrates a selection-control statement, which is always drawn as a diamond. All decisions are stated as "yes/no" questions. When a program is executed, if the answer to a decision is "yes" (or true), then the left branch of control is taken. If the answer is "no" (or false), then the right branch of control is taken.



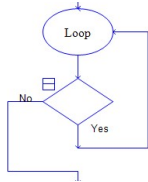
In this example, either statement 2a or statement 2b will be executed, **but never both**. Note that there are two possible executions of this example program:

Possibility 1	Possibility 2
Statement 1	Statement 1
Statement 2a	Statement 2b
Statement 3	Statement 3

Also note that either of the two paths of a selection-control statement could be empty or could contain several statements. It would be inappropriate for both paths to be empty or for both paths to have the exact

same statements, because then your decision, Yes or No, would have no effect during program execution (since nothing different would happen based on the decision).

Loop (Iteration) Control



A Loop (iteration) control statement allows you to repeat one or more statements until some condition becomes false. One ellipse and one diamond symbol is used to represent a loop in RAPTOR. The number of times that the loop is executed is controlled by the decision expression that is entered into the diamond symbol. During execution, when the diamond symbol is executed, if the decision expression evaluates to "yes," then the "yes" branch is taken, which always leads back to the Loop statement and repetition. The statements to be repeated can be placed above or below the decision diamond. Infinite loops are bad and special care should be used to make sure your loops always terminate.

Now you should be ready to use Raptor to design flowchart and solve the problems below.

- Design a flowchart which asks the user to enter his salary. It then computes and displays the increment of the salary. The increment is 5% of the salary. E.g. if the salary is \$2000, then he will get \$100 increment.

Input(s)	salary
Output	increment
Process	Salary * (5/100)

4. Design a flowchart which asks the user to input the size of a square. The user will be asked to choose between perimeter and area. The result will be calculated and then displayed.

Input(s)	
Output	
Process	

5. Draw a flowchart to compute and display the factorial Number (**Number!**),

Where **Number! = 1 x 2 x 3 x Number.**

*Note that the user input is **Number**, you may assume the number entered is always greater than 1.*

Input(s)	
Output	
Process	

6. (Optional question) Design a flowchart which asks the user to input a number. The multiplication table of that number will be displayed (as shown below).

e.g. when 5 is entered:

5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
5 x 10 = 50
5 x 11 = 55
5 x 12 = 60

- END -