BASIC PROGRAMMING CONCEPT

LESSON 1
PROGRAM AND PROGRAMMING

PROGRAM

A computer program is a series of organised instructions that directs a computer to perform tasks. Without programs, computers are useless.

A program is like a recipe. It contains a list of variables (called ingredients) and a list of statements (called directions) that tell the computer what to do with the variables.

Like a recipe, a program can be written in different programming languages which may express the steps differently according to the programming language syntax, but deliver the same end result.

PROGRAMMING

Programming is a creation of a set of commands or instructions which directs a computer in carrying out a task.

Later these commands or instructions will be compiled and/or interpreted and then transformed to executable instructions that a computer or electronic device can execute or run.
PROGRAMMING LANGUAGE

A programming language is a set of words, symbols and codes that enables humans to communicate with computers.

It is a language used for writing computer programs, that direct a computer to perform computation and to organise the flow of control between mechanical devices.

EXAMPLES PROGRAMMING LANGUAGE

Hundreds of programming languages exist today. Each language has its own standard or rules for writing the commands and/or instructions.

Examples of programming languages are:

- BASIC (Beginner’s All Purpose Symbolic Instruction Code)
- Pascal
- C
- Smalltalk.

EXT : CAREER IN PROGRAMMING

A programmer is someone who writes computer programs. One who adopts and practices a formal approach to programming is sometimes also referred to as a programmer analyst, computer scientist, software engineer and software analyst.

A programmer analyst designs computer programs besides writing them.

A computer scientist is a generic or broad term for a professional with expertise in computer software.

These different job titles are quite subjective as different companies may define them differently. Professional programmers may work in corporate IT departments, software houses and service companies.

Sometimes professional programmers work for consulting companies and their work often takes them to their client’s workplace.
LESSON 2
GENERATIONS OF PROGRAMMING LANGUAGE

A low-level programming language is a programming language that provides little or no abstraction from computer’s microprocessor.

A high-level programming language is a programming language that is more abstract, easier to use, and more portable across platforms.
FIRST GENERATION OF PROGRAMMING LANGUAGE

The first generation of programming language, or 1GL, is machine language. Machine language is a set of instructions and data that a computer's central processing unit can execute directly.

Machine language statements are written in binary code, and each statement corresponds to one machine action.

SECOND GENERATION PROGRAMMING LANGUAGE

The second generation programming language, or 2GL, is assembly language. Assembly language is the human-readable notation for the machine language used to control specific computer operations.

An assembly language programmer writes instructions using symbolic instruction codes that are meaningful abbreviations or mnemonics.

An assembler is a program that translates assembly language into machine language.

Since assembly language consist of human-readable abbreviations, the assembler must first convert assembly language into machine-readable language before the computer can readily understand its instructions.
THIRD GENERATION PROGRAMMING LANGUAGE

The third generation of programming language, 3GL, or procedural language uses a series of English-like words, that are closer to human language, to write instructions.

High-level programming languages make complex programming simpler and easier to read, write and maintain. Programs written in a high-level programming language must be translated into machine language by a compiler or interpreter.

PASCAL, FORTRAN, BASIC, COBOL, C and C++ are examples of third generation programming languages.

FOURTH GENERATION PROGRAMMING LANGUAGE

The fourth generation programming language or non-procedural language, often abbreviated as 4GL, enables users to access data in a database.

A very high-level programming language is often referred to as goal-oriented programming language because it is usually limited to a very specific application and it might use syntax that is never used in other programming languages.
SQL, NOMAD and FOCUS are examples of fourth generation programming languages.

**FIFTH GENERATION PROGRAMMING LANGUAGE**

The fifth generation programming language or visual programming language, is also known as natural language.

Provides a visual or graphical interface, called a visual programming environment, for creating source codes.

Fifth generation programming allows people to interact with computers without needing any specialised knowledge.

People can talk to computers and the voice recognition systems can convert spoken sounds into written words, but these systems do not understand what they are writing; they simply take dictation.

Prolog and Mercury are the best known fifth-generation languages.
EXT: OPEN PROGRAMMING LANGUAGE

The Open Programming Language (OPL) is an embedded programming language found in portable devices that run the Symbian Operating System.

For example mobile telephones and PDAs.

OPL is an interpreted language that is analogous to BASIC.

In the early years, before the computer was invented, there are several inventions of counting machines.

LESSON 3
PROGRAMMING APPROACHES

STRUCTURED PROGRAMMING EDUCATION

Structured programming often uses a top-down design model where developers map out the overall program structure into separate subsections from top to bottom.

In the top-down design model, programs are drawn as rectangles. A top-down design means that the whole program is broken down into smaller sections that are known as modules. A program may have a module or several modules.
Structured programming is beneficial for organising and coding computer programs which employ a hierarchy of modules. This means that control is passed downwards only through the hierarchy.

Examples of structured programming languages include Ada, Pascal and Fortran.

OBJECT-ORIENTED PROGRAMMING

The object-oriented approach refers to a special type of programming approach that combines data with functions to create objects.

In an object-oriented program, the objects have relationships with one another.

One of the earliest OOP languages is Smalltalk. Java, Visual Basic and C++ are examples of popular OOP languages.
DIFFERENCE BETWEEN STRUCTURED AND OBJECT ORIENTED PROGRAMMING

- Structured programming often uses a top-down design model.
- The object-oriented programming approach uses objects.

LESSON 4
TRANSLATOR

Sometimes two people cannot understand each other because they don’t speak the same language. So they need the help of a third person who understands both languages. This third person is known as a translator.

All software packages or programs are written in high-level languages, for example, C++, Visual Basic and Java.

However, in order for the computer to be able to carry out the instructions, the high-level languages must be translated into machine language before the computer can understand and execute the instructions in the program.

The translation of high level languages to machine language is performed by a translator.

PROGRAM

Have you ever wondered how your computer runs your favourite software? Your favourite software is a program that consists of several instructions that perform its operation.
A programmer will write a source code which consists of the instructions needed to run a program. Then the compiler or interpreter with assembler will translates the source code into machine language which is made of a sequence of bits (eg. 01100011).

The computer will load the machine code and run the program.

**ASSEMBLER**

An assembler is a computer program for translating assembly language — essentially, a mnemonic representation of machine language — into machine language.

For example in intel 80836, the assembly language for the ‘no operation’ command is NOP and its machine code representation is 10010000.

Example of assemblers are MACRO-80 Assembler and Microsoft MASM.

**INTERPRETER**

Interpreter is used to interpret and execute program directly from its source without compiling it first. The source code of an interpreted language is interpreted and executed in real time when the user execute it.

The interpreter will read each codes converts it to machine code and executes it line by line until the end of the program.

Examples of interpreter-based language are BASIC, Logo and Smalltalk.
COMPILER

The source code (in text format) will be converted into machine code which is a file consisting of binary machine code that can be executed on a computer. If the compiler encounters any errors, it records them in the program-listing file.

When a user wants to run the program, the object program is loaded into the memory of the computer and the program instructions begin executing.

A compiled code generally runs faster than programs based on interpreted language. Several programming languages like C++, Pascal and COBOL used compilers as their translators.

LESSON 5
HOW TO INSTALL VISUAL BASIC 6.0

VISUAL BASIC 6.0 INSTALLATION

Microsoft Visual Basic 6.0 was designed to be easy to learn and use. The language not only allows programmers to easily create simple GUI applications, but also has the flexibility to develop fairly complex applications as well.

Programming in Microsoft Visual Basic 6.0 is a combination of visually arranging components or controls on a form. Hence a simple program can be created without the programmer having to write many lines of code.

VISUAL BASIC 6.0 FEATURES

Microsoft Visual Basic is fast and easy with intuitive tools that enable you to rapidly build your own Windows applications. You can use Microsoft Visual Basic to develop programs such as games, calculator, phonebook database and lots more.

The word “Visual” refers to the technique used to build the Graphical User Interface (GUI). Visual Basic features an easy “drag and drop” method to produce an interactive Graphical User Interface (GUI) for your program.
The “Basic” part refers to the BASIC (Beginners All-Purpose Symbolic Instruction Code) language, a high level programming language develop by Microsoft.

VISUAL BASIC 6.0 FEATURES

Among the main features of Visual Basic are drag and drop user interface, data access features, ActiveX technologies and internet capabilities. Drag and drop user interface allows you to instantly create an interactive user interface for your software without dozens of codes.

Data access features allow your application to access information from a database such as a telephone book program.

ActiveX technologies allow you to make use of the functionality provided by other applications.

Internet capabilities allow your program to interactively utilise the Internet.
LESSON 6
BASIC ELEMENT IN PROGRAMMING

5 Basic elements in programming.

1. Constant.
2. Variable.
3. Data Type.
4. Operators.
5. Control Structures.

CONSTANTS AND VARIABLES

Constants
Constant is a data container that stores information. The value will never change (remains constant) at any time during the course of a program.

Declare is the official term used in programming to announce to the program the condition of statement in programming.

Variables
Variable is a data container that stores information. The value inside may change at any time during the course of a program.
DATA TYPES, OPERATOR AND CONTROL STRUCTURES

![Data Types Diagram]

**Data Types**

- **String**
  - Example: "This is String."

- **Integer**
  - Example: 1, 20, 1050

- **Floating point**
  - Example: 1.25, 3.141

Operator is a symbol that tells what action to perform.

<table>
<thead>
<tr>
<th>Operators symbol</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>+ (plus)</td>
<td>marks = 20 + 50</td>
</tr>
<tr>
<td>- (minus)</td>
<td>marks = 31 - 12</td>
</tr>
<tr>
<td>/ (divided)</td>
<td>marks = 100 / 2</td>
</tr>
<tr>
<td>* (multiplied)</td>
<td>marks = 50 * 2</td>
</tr>
<tr>
<td>&gt; (greater than)</td>
<td>marks &gt; 1</td>
</tr>
<tr>
<td>&lt; (less than)</td>
<td>marks &lt; 100</td>
</tr>
<tr>
<td>= (equal to)</td>
<td>marks == 100</td>
</tr>
<tr>
<td>&lt;&gt; (not equal to)</td>
<td>marks &lt;&gt; 100</td>
</tr>
</tbody>
</table>

Control structures allow the programmer to control the flow of a program.

```plaintext
if marks>50 then
  {status=pass}
else
  {status=fail}
```

For example, Marks = 80

```
BEGIN
  marks = 80
  if marks>50 then
    status = pass
  else
    status = fail

  if status = pass then
    END
  else
    Fail

END
```
LESSON 7
THE DIFFERENCES BETWEEN ETHICS AND LAW

CONSTANTS AND VARIABLES
Constant is a virtual data container that stores information. The value will never change (remains constant) at any time during the course of a program.

Variables is a virtual data container that stores information. The value inside may change at any time during the course of a program.

Differences between Constants and Variables

<table>
<thead>
<tr>
<th>Constants</th>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>MARGIN = 10</td>
<td>MARGIN = 10</td>
</tr>
<tr>
<td>Value can be initialized</td>
<td>can be stored</td>
</tr>
</tbody>
</table>

Differences between Constants and Variables

- **Constants**: Value is not changeable during the course of the program.
- **Variables**: Value can be changed anytime during the course of the program.
LESSON 8
RIGHTS DATA TYPES

DATA EXAMPLES FOR DIFFERENT DATA TYPES

Data type determines the type of data a variable can store, for example a number or a character. Examples of data types are integer, double, string and boolean.

<table>
<thead>
<tr>
<th>Data Types</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>18, 79, 21</td>
</tr>
<tr>
<td>Double</td>
<td>41.5, 31.4</td>
</tr>
<tr>
<td>String</td>
<td>Kok Keong, Amira, Eden, Alex</td>
</tr>
<tr>
<td>Boolean</td>
<td>TRUE, FALSE</td>
</tr>
</tbody>
</table>
DATA EXAMPLES FOR DIFFERENT DATA TYPES

**Integer**  
Integer data type contains any whole number value that does not have any fractional part.

```
Const Year_Birth = 1990
```

This is how we declare an integer type constant in Visual Basic statement.

```
Dim Age As Integer  
Age = -17
```

This is how we declare an integer type variable in Visual Basic statement.

**Double**  
Any number value that may and could contain a fractional part.

```
Const PI = 3.142
```

This is how we declare a double type constant in Visual Basic statement.

```
Dim Marks As Double  
Marks = 60.5
```

This is how we declare a double type variable in Visual Basic statement.

**String**  
Any value that contains a sequence of characters.

```
Const Name = "AHMAD"
```

This is how we declare string type constant in Visual Basic statement.

```
Dim Address As String  
Address = "Kuala Lumpur"
```

This is how we declare string type variable in Visual Basic statement.
**Boolean**
Boolean type consists either a True or False value. Programmers usually use it to store status.

```
Const Input_Status = true
```
This is how we declare a boolean type constant in Visual Basic statement.

```
Dim Input_Status As Boolean
Input_Status = true
```
This is how we declare a boolean type variable in Visual Basic statement.

**OTHER DATA TYPES**

<table>
<thead>
<tr>
<th>Data type</th>
<th>Size</th>
<th>Sample usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer</td>
<td>2 bytes</td>
<td>Dim Discount As Integer</td>
</tr>
<tr>
<td>Long Integer</td>
<td>4 bytes</td>
<td>Dim Loan As Long</td>
</tr>
<tr>
<td>Single-precision floating point</td>
<td>4 bytes</td>
<td>Dim Price As Single</td>
</tr>
<tr>
<td>Double-precision floating point</td>
<td>8 bytes</td>
<td>Dim PI As Double</td>
</tr>
<tr>
<td>Currency</td>
<td>8 bytes</td>
<td>Dim Debt As Currency</td>
</tr>
<tr>
<td>String</td>
<td>1 byte per character</td>
<td>Dim Input As String</td>
</tr>
<tr>
<td>Boolean</td>
<td>2 bytes</td>
<td>Dim Flag As Boolean</td>
</tr>
</tbody>
</table>
LESSON 9
MATHEMATICAL AND LOGICAL OPERATORS

MATHEMATICAL, RELATIONAL AND LOGICAL OPERATORS
Operator is a symbol or notation that tells a computer to perform certain actions or operations.

An example: the plus (+) notation will tell the computer to perform the "add" operation.

Let's look at some examples of Mathematical Operators.
Let's look at some examples of Relational Operators.

**Multiply Operator**

\[ A = 30, \quad B = 15 \]

\[ \text{Answer} = A \times B \]

**MultiplyOperator**

\[ A \times B = 450 \]

**Divide Operator**

\[ A = 30, \quad B = 15 \]

\[ \text{Answer} = A \div B \]

**DivideOperator**

\[ A \div B = 2 \]

Private Sub Form_Load()
    Dim A As Integer
    Dim B As Integer
    Dim Answer As Integer
    A = 30
    B = 15
    Answer = A * B
    MsgBox ("A \times B = " & Answer)
End Sub

Private Sub Form_Load()
    Dim A As Integer
    Dim B As Integer
    Dim Answer As Integer
    A = 30
    B = 15
    Answer = A / B
    MsgBox ("A \div B = " & Answer)
End Sub

Equal to
\[ A = B \]

Greater than
\[ A > B \]

Less than
\[ A < B \]

Greater than or equal to
\[ A \geq B \]

Less than or equal to
\[ A \leq B \]
FUNCTION OF MATHEMATICAL OPERATORS

Mathematical operators are notations that tell the computer to perform mathematical operations.

FUNCTIONS OF RELATIONAL OPERATORS

Relational operators perform comparison between two elements.

They return an element of logical 1 (True) where the relation is true, and element of logical 0 (False) where the relation is false.

The diagram shows some common relational operators and their expression.
Let's understand a relational operation with the following examples.

In this expression, A >= B is used to test if the value of the left expression (A) is greater than or equal to that of the right expression (B).

If the conditions are met, then the program will return a logical 1 and proceed to print "A is greater than or equal to B" else it will return a logical 0 and proceed to print "A is less than B".

FUNCTIONS OF LOGICAL OPERATORS

Logical operators are notations that tell the computer to perform logical operations.

Examples of Logical operation are: AND, OR, and NOT.

Logical operator compares 2 conditions and returns a TRUE or FALSE value.
**AND operator**

The diagram shows a truth table of AND operator.

Notice that truth value of X AND Y is True (1) if only both X and Y are True (1).

Else it is false (0)

```
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X AND Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

1 represent "True"
0 represent "False"

**OR operator**

The diagram shows a truth table of OR operator.

Notice that truth value of X OR Y is only True (1) if either X or Y are True (1) or both X and Y are true (1).

Else it is false (0)

```
<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>X OR Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
```

1 represent "True"
0 represent "False"

**NOT operator**

The diagram shows the truth table of NOT operator NOT X is the negation of X, it is essentially the 1’s complement operation.

Notice that truth value of NOT X is True (1) when X is False and vice versa.

```
<table>
<thead>
<tr>
<th>X</th>
<th>NOT X</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
```
DIFFERENCES IN MATHEMATICAL AND LOGICAL OPERATORS

As we have learned, there are some differences between mathematical, relational and logical operators.

Function:

Mathematical operators perform mathematical operations such as plus or subtract.

- $20 + 15 = 35$
- $20 - 15 = 5$
- $20 \times 15 = 300$
- $20 \div 15 = 1.33$

Relational operators perform element-by-element comparisons between two arrays.

- $A = B$
- $C > D$
- $E < F$
- $G \geq H$
- $I \leq J$

Logical operators perform logical operations such as checking the condition of two Boolean values.

- $A \text{ AND } B$
- $C \text{ OR } D$
- $\text{ NOT } E$
Symbols:

These operators have their own symbols based on the programming language.

![Operator's Symbol](image)

**EQUALITY OPERATORS**

The equality determine an operand equal to or not equal to another operand.

Remember we must use "=" and not "<>" when testing if two primitive values are equal in Microsoft Visual Basic 6.0.

**LESSON 10**

**PSEUDO CODES**

Pseudo code is text only sentences that describe the logic and program flow of a computer program. Pseudo code resembles plain English.

It usually does not have any specific programming language syntax and grammar.

Pseudo code is directly linked to the computer codes because each pseudo code statement can often be converted into the programming language virtually line by line.
There are no set rules for writing pseudo code.

A programmer can have his or her personalised pseudo code.

He or she must use consistent language and syntax in the pseudo code, so that he or she can understand it at a later stage.

**AN EXAMPLE PROGRAM BASED ON A PSEUDO CODE**

This program will basically calculate the Volume of a sphere based on the given value \( r \).

BEGIN
Declare \( \pi \) as a constant
Declare "\( r \)" and "volume"
Variable "\( r \)" equal to 10
Execute sphere's volume formula
Print result in message box (MsgBox)
END

**EXT : STANDARDS OF GOOD PSEUDO CODE**

Pseudo-Code is simply a numbered list of instructions to perform some task. In this course we will enforce three standards for good pseudo code

Number each instruction. This is to enforce the notion of an ordered sequence of the operations.

Furthermore we introduce a dot notation (e.g. 3.1 come after 3 but before 4) to number subordinate operations for conditional and iterative operations

Each instruction should be unambiguous and effectively computable.

Completeness. Nothing is left out.

Pseudo-code is best understood by looking at examples.
Each example below demonstrates one of the control structures used in algorithms: sequential operations, conditional operations, and iterative operations.

We also list all variables used at the end of the pseudo-code.

Example:

Computing discount:

Pseudo-code:
Computing the final price of an item after figuring in discount.

Note the three types of instructions:

input (request),
process/calculate (=) and
output (print)

LESSON 11
FLOW CHART

Let’s identify some of the main elements in the flow chart. We have five main elements in a flow chart.

**Terminator** shows the beginning or end of a program.

**Flowline and arrowhead** use to connect symbols and indicate the sequences of operation.

**Input or output** shows either an input operation (e.g. an INPUT from the user) or an output operation (e.g. PRINT some messages).

**Process** shows a process to be carried out (e.g. calculation).

**Decision** shows a decision (or choice) to be made. The program should continue along one of two routes (e.g. if...else).

A flow chart is a diagram using symbols to show the step-by-step sequence of procedures in a program. A flow chart describes the logic and program flow of a computer program graphically.

**Example:**

A student requests his marks from the program.

The program will then check whether his marks is more/equal to 50 or not.

If the marks are over or equal to 50, the program will print a congratulatory message together with the marks.

If the marks are lower than 50 then the program will print a motivational message together with the result.
LESSON 12
CONTROL STRUCTURES

Control structure is a structure of statements in programming that allows the programmer to control the flow of a program.

Control structure can be divided into sequence, selection and repetition control structures.

SEQUENCE CONTROL

Sequence control refers to the linear execution of codes within a program. In sequence control, the statements are executed one by one in consecutive order.

In sequence control, the statements are executed one by one in consecutive order.
Let’s see an example of pseudo code that has sequence control structure.

```
Pseudo Code of Sequence Control

00 BEGIN
01 Request date of birth
02 Request today’s date
03 Calculate age
    (today’s date minus date of birth)
04 Print age
05 END
```

This program will request the user’s date of birth and then request today’s date, calculate the age and finally will print the user’s age.

For example, today’s date is 1-JAN-2006, then the results will be shown as follows:

<table>
<thead>
<tr>
<th>Student's name</th>
<th>Student's date of birth</th>
<th>Age</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hassan b. Zamori</td>
<td>1-3-1988</td>
<td>17</td>
</tr>
<tr>
<td>Rosita bt. Kamarudin</td>
<td>22-6-1986</td>
<td>19</td>
</tr>
<tr>
<td>Sirajuddin b. Najib</td>
<td>14-4-1987</td>
<td>18</td>
</tr>
</tbody>
</table>

Let’s see another example of pseudo code that has sequence control structure.

```
BEGIN
Request employee ID, employee salary, employee allowance
Calculate totals salary
    (salary plus allowance)
Print salary
END
```

The following is the result of the example.

<table>
<thead>
<tr>
<th>Employee ID</th>
<th>Salary (RM)</th>
<th>Allowance (RM)</th>
<th>Total (RM)</th>
</tr>
</thead>
<tbody>
<tr>
<td>32156</td>
<td>2500</td>
<td>100</td>
<td>2600</td>
</tr>
<tr>
<td>41324</td>
<td>3200</td>
<td>150</td>
<td>3350</td>
</tr>
<tr>
<td>32667</td>
<td>2500</td>
<td>160</td>
<td>2660</td>
</tr>
</tbody>
</table>

Let’s see the flow chart for a general sequence control structure.

As we can see, the flow chart for sequence control is very simple.

It will execute statement 1 followed by statement 2 and any following statements.

**EXAMPLE**
The flow chart represents a program that will request the user’s date of birth and then request today’s date, calculate the age and finally will display the user’s age.

**SELECTION CONTROL**

There are times when you want your program to make a decision based on the situation given.

For example, a program that stores student’s marks may respond differently to different marks.

Or maybe a simple mathematical program will display its result as odd or even, based on the result.

Selection control enables the programmer to assign different events for different situations.

An example of selection control is “If...Then...Else” statement. The basic pseudo code for “If...Then...Else” statement is as follows.

Let’s see a pseudo code example for the “If...Then...Else” statement for a program that will print “You are too heavy to ride the toy car” if the student’s weight is more than or equal to 50.

Otherwise it will print “You can ride the toy car”

**Let’s see the flow chart for a similar selection control example.**
Let’s see a program that will respond differently for different service hours in a school library.

A student requests her service hours in a library from the program, the program will then check whether her service hours are more than/equal to 30 or not.

If the service hours are over or equal to 30, the program will print a message, “Thank you for your service”.

If the service hours are lower than 30 then the program will print a message, “Please continue to serve in the library”.

**Let’s write an example of a program that implements sequence control structure.**

This program will declare a constant pi equal to 3.142. Then the program declares two variables to be used in the program (r and volume).

The program will then assign some value to the r variable.

The program will then calculate the volume using the formula volume = (4/3)πr

The program will then display the volume calculated with a message box.
Now write an example program that implements selection control structure.

Write a program that will retrieve the current date from the system.

If today’s date is more than 15 then the program will display a message box with the message “We are towards the end of the month”.

Else the program will print “We are at the beginning of the month”.

DIFFERENTIATE BETWEEN SELECTION CONTROL AND SEQUENCE CONTROL
EXT : REPETITION CONTROL STRUCTURES

Let’s learn about the last control structure which is repetition control structure. A repetition control structure allows the programmer to specify an action to be repeated while some condition remains true.

**Pseudo Code:**

```
BEGIN
While there are more items on my shopping list
   Purchase next item and cross it off my list
END
```

This is a pseudo code of "While" structure.

```
Dim product As Integer
product = 3
While product <= 3000
   product = product*3
END While
```

The value of product will increase in this order:
3, 9, 27, 81, 243, 729, 2187, 6561

This program segment will continue its looping until the condition of variable named product is greater than 3000.

This is a flow chart of a repetition control structure.