**INTRODUCTION TO PROGRAMMING**

**(COM 113)**

**LECTURER – NDUESO WALTER OKORIE**

**PROGRAMME – NATIONAL DIPLOMA**

**DEPARTMENT - COMPUTER SCIENCE**

**1.0 WHAT IS PROGRAMMING?**

Programming is the act of instructing computers to carry out tasks.” It is often referred to as **coding**. Programming is also defined as a way to instruct the computer to perform various tasks.

“Instruct the computer”: this basically means that you provide the computer a set of instructions that are written in a language that the computer can understand. The instructions could be of various types. For example:

* Adding 2 numbers,
* Rounding off a number, etc.

Just like we humans can understand a few languages (English, Spanish, Mandarin, French, etc.), so is the case with computers. Computers understand instructions that are written in a specific syntactical form called a programming language.

“Perform various tasks”: the tasks could be simple ones like we discussed above (adding 2 numbers, rounding off a number) or complex ones which may involve a sequence of multiple instructions. For example:

* Calculating simple interest, given principal, rate and time.
* Calculating the average return on a stock over the last 5 years.

The above 2 tasks require complex calculations. They cannot usually be expressed in simple instructions like adding 2 numbers, etc.

Hence, in summary, Programming is a way to tell computers to do a specific task.

So then, what is a **computer program**? A computer program is a sequence of instructions that the computer executes. Computer programs are also referred to as **code or Instructions.**

Every time we use smart devices, some code is running in the background. Moving a mouse pointer from one part of your computer screen to the other may seem like a simple task, but in reality, so many lines of code just ran. An act as simple as typing letters into Google Docs leads to lines of code being executed in the background. It’s all code everywhere.

Machines have their natural language like humans do. Computers do not understand the human language. The natural language of computers is the binary code — 1 and 0. These represent two states: **on (1)** and **off (0)**.

That is the natural language of electronic equipment. It would be hectic for us as humans to communicate with the computer in binary.

To communicate with machines who speak binary, we do so in a language that’s closer to our own natural language. Such as English, French, Swahili or Arabic. Programming languages are close to our natural languages. But they are more structured and must be thoroughly learned.

They could be high level or low level languages. High level programming languages are farther away from the machine language than low level languages. This “farther away” is usually called an **abstraction**, but we will not go into that in this series. Let’s not get distracted :)

The computer needs a way to understand our human language. To do this, we’ll need a translator.

**1.1 FEATURES OF A GOOD PROGRAM**

A good computer program should have following characteristics:

### **Portability**

Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes. A good program should be supported by many different computers. The program should compile and run smoothly on different platforms. So, portability is measured by how a software application can be transferred from one computer environment to another without failure. A program is said to be more portable, if it is easily adopted in different computer systems.

**Readability**: The program should be written in such a way that it makes other programmers or users to follow the logic of the program without much effort. If a program is written structurally, it helps the programmers to understand their own program in a better way.

### **Maintainability**

It is the process of fixing program errors and improving the program. If a program is easy to read and understand, then its maintenance will be easier. It should also have quality to easily meet new requirements.

### **Efficiency**

Every program requires certain processing time and memory to process the instructions and data. As the processing power and memory are the most precious resources of a computer, a program should be laid out in such a manner that it utilizes the least amount of memory and processing time.

**Structural**:

To develop a program, the task must be broken down into a number of subtasks. These subtasks are developed independently, and each subtask is able to perform the assigned job without the help of any other subtask. If a program is developed structurally, it becomes more readable, and the testing and documentation process also gets easier.

**Flexibility**: Program should be written in such a manner that it allows to add new features without changing the existing module. It should be always ready to meet new requirements. A high flexible software is always ready for a new world of possibilities. Most of the programs are developed for a certain period and they require modifications from time to time.

For example, in case of payroll management system, as the time progresses, some employees may leave the company while some others may join. Hence, the payroll application should be flexible enough to incorporate all the changes without having to reconstruct the entire application.

### **Reliable**

The user's actual needs will change from time-to-time, so program is said to be reliable if it works smoothly in every version. It is measured as highly reliable if it gives same performance in all simple to complex conditions.

### **Cost Effectiveness**

Cost Effectiveness is the key to measure the program quality. Cost must be measured over the life of the program and must include both cost and human cost of producing these programs.

**Generality:**

Apart from flexibility, the program should also be general. Generality means that if a program is developed for a particular task, then it should also be used for all similar tasks of the same domain. For example, if a program is developed for a particular organization, then it should suit all the other similar organizations.

**Accuracy or correctness:**

A good program must be able to solve a task it was supposed to do by generating and accurate result.

**Interoperability**:

A quality software must be able to interface with other software system. E.g you can use your facebook credentials to login to some website. Also, you can integrate an SMS API from another website into a web application or website. API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

**Security**:

This is the most important characteristics of a good program or software. The software must be able to protect and secure user’s data and information.

**User friendly:**

A good program must have a simple GUI which will be easy to use by all types of users without much assistance.

**GUI** Stands for "Graphical User Interface" is a system of interactive visual components for computer [software](https://www.computerhope.com/jargon/s/software.htm).

**1.2 STEPS IN DEVELOPING A GOOD PROGRAM**

**Problem definition**

The first step in the process of program development is the thorough understanding and identification of the problem for which is the program or software is to be developed. In this step the problem has to be defined formally.

In this stage, a question is asked - “What are the [current](https://stackify.com/sdlc-phases-identify-problems/) problems?” This stage of the program development means getting input from all stakeholders, including customers, salespeople, industry experts, and programmers. Learn the strengths and weaknesses of the current system with improvement as the goal.

**Analyzing the program**

In this stage or step or phase, the end-user’s requirements should be determined and documented, what their expectations are for the system and how it will perform.

**Program design**

The next stage is the program design. it is the major aspect of program development. The software developer makes use of tools like algorithms and flowcharts to develop the design of the program.

### **Coding or Writing The Program**

The next step after designing the [algorithm](https://sciencerack.com/scheduling-algorithm/) is to write the program in a high-level language. This process is known as coding.

### **Test Execution**

The process of executing the program to find out errors or bugs is called test execution. It helps a programmer to check the logic of the program. It also ensures that the program is error-free and workable.

### **Debugging**

Debugging is a process of detecting, locating and correcting the bugs in a program. It is performed by running the program again and again.

### **Final Documentation**

### Documentation is a written detailed description of the programming cycle and specific facts about the program. Typical program documentation materials include the origin and nature of the problem, a brief narrative description of the program, logic tools such as flowcharts and pseudocode, data-record descriptions, program listings, and testing results. Comments in the program itself are also considered an essential part of documentation. since turnover is high in the computer industry, written documentation is needed so that those who come after you can make any necessary modifications in the program or track down any errors that you missed.

### **Maintenance:**

Once when the customers starts using the developed system then the actual problems comes up and needs to be solved from time to time. This process where the care is taken for the developed product is known as maintenance.

**2.0 ALGORITHM AND FLOWCHART**

**ALGORITHM**: An **algorithm** is a sequence of steps to solve a particular problem or algorithm is an ordered set of unambiguous steps that produces a result and terminates in a finite time.

## **2.1 CHARACTERISTICS OF AN ALGORITHM**

Not all procedures can be called an algorithm. An algorithm should have the below mentioned characteristics −

* **Unambiguous −** Algorithm should be clear and unambiguous. Each of its steps (or phases), and their input/outputs should be clear and must lead to only one meaning.
* **Input −** An algorithm can have 0 or more well defined inputs.
* **Output −** An algorithm should have 1 or more well defined outputs, and should match the desired output.
* **Finiteness −** Algorithms must terminate after a finite number of steps.
* **Feasibility −** should be feasible with the available resources.
* **Independent −** An algorithm should have step-by-step directions which should be independent of any programming code.

**2.1 ADVANTAGES OF ALGORITHM**

1. It is a step-wise representation of a solution to a given problem, which makes it easy to understand.
2. An algorithm uses a definite procedure.
3. It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
4. Every step in an algorithm has its own logical sequence so it is easy to debug.

**2.3 HOW TO WRITE ALGORITHMS**

**Step 1-**Define your algorithms input: Many algorithms take in data to be processed, e.g. to calculate the area of rectangle input may be the rectangle height and rectangle width.

**Step 2-** Define the variables: Algorithm's variables allow you to use it for more than one place. We can define two variables for rectangle height and rectangle width as HEIGHT and WIDTH (or H & W).We should use meaningful variable name e.g. instead of using H & W use HEIGHT and WIDT Has variable name.

**Step 3-**Outline the algorithm's operations: Use input variable for computation purpose, e.g. to find area of rectangle multiply the HEIGHT and WIDTH variable and store the value in new variable (say) AREA. An algorithm's operations can take the form of multiple steps and even branch, depending on the value of the input variables.

**Step 4-**Output the results of your algorithm's operations: In case of area of rectangle output will be the value stored in variable AREA. if the input variables described a rectangle with a HEIGHT of 2 and a WIDTH of 3, the algorithm would output the value of 6.

**2.4 EXAMPLES**

1. **Algorithm to find the sum of two numbers**

Algorithm

Step-1 Start

Step-2 Input first numbers say A

Step-3 Input second number say B

Step-4 SUM = A + B

Step-5 Display SUM

Step-6 Stop

1. **Algorithm to convert temperature from Celsius to Fahrenheit**

C : temperature in Celsius

F : temperature Fahrenheit

Algorithm

Step-1 Start

Step-2 Input temperature in Celsius say C

Step-3 F = (9.0/5.0 xC) + 32

Step-4 Display Temperature in Fahrenheit F

Step-5 Stop

1. **Algorithm to convert temperature from Fahrenheit to Celsius**

C: temperature in Celsius

F : temperature Fahrenheit

Algorithm

Step-1 Start

Step-2 Input temperature in Fahrenheit say F

Step-3 C = 5.0/9.0 (F -32 )

Step-4 Display Temperature in Celsius C

Step-5 Stop

1. Algorithm to find Area and Perimeter of Square

L : Side Length of Square

AREA: Area of Square

PERIMETER: Perimeter of Square

Algorithm

Step-1 Start

Step-2 Input Side Length of Square say L

Step-3 Area = L x L

Step-4 PERIMETER = 4 x L

Step-5 Display AREA, PERIMETER

Step-6 Stop

**2.5 FLOWCHART**

A flowchart is a picture of the separate steps of a process in sequential order.

The first design of flowchart goes back to 1945 which was designed by John Von Neumann.

**2.6 Advantages of flowchart**

•Flowchart is an excellent way of communicating the logic of a program.

•Easy and efficient to analyze problem using flowchart.

•During program development cycle, the flowchart plays the role of a blueprint, which makes program development process easier.

•After successful development of a program, it needs continuous timely maintenance during the course of its operation. The flowchart makes program or system maintenance easier.

•It is easy to convert the flowchart into any programming language code

**2.7 FLOWCHART SYMBOLS AND THEIR MEANING**

### **Start and end symbol**

https://creately.com/sites/default/files/objectdetail/thumbnails/start_end.png

This symbol is also referred to as the terminator symbol as it represents starting and end points, as well as potential outcomes of a process path. The start and end symbols will be an elongated oval shape.

### **Process symbol**

https://creately.com/sites/default/files/objectdetail/thumbnails/process_0.png

This common symbol is shaped as a rectangle, and it can also be called the action symbol. It represents an action, function or process and can be considered one of the most-used flowchart symbols.

### **Document symbol**

https://creately.com/sites/default/files/objectdetail/thumbnails/document.png

This symbol is shaped like a rectangle with its bottom side in a wave, and it is used to represent the input or output of a document. For instance, this symbol might be used to outline a document input, such as receiving an email or report. Similarly, it can be used to represent a document output like producing a presentation or project.

### **Connector symbol**

These symbols are little circles used to connect separate elements of a flowchart across a whole page. Connectors are typically used in flowcharts with more complex processes, such as software or application development.

### **5. Decision symbol**

https://creately.com/sites/default/files/objectdetail/thumbnails/decision.png

Shaped as a rhombus, this symbol is used to indicate a question that results in a "yes" or "no" answer, as well as a possible "true" or "false" situation. Depending on the answer to the proposed question, the flowchart can then split into various branches to complete the outline of the workflow.

### **6. Off-page connector or link symbol**

This symbol looks like an upside-down pentagon and is oftentimes used in more complex flowcharts to connect the separate elements of multiple pages. There can be a page number within each shape, allowing for easier reference.

### **7. Input and output symbol**

https://creately.com/sites/default/files/objectdetail/thumbnails/data.png

This symbol is used to represent any data that can be available for input and output. Also referred to as the data symbol, this shape can also be used to represent the resources used or produced. Shaped as a parallelogram, it may sometimes be substituted with the paper tape symbol.

### **8. Multiple documents symbol**

Conversely to the document symbol, this symbol represents more than one document in the flowchart. It has the same meaning as the document symbol, with the addition of more documents or reports that may be needed for the workflow.

### **9. Database symbol**

This cylinder-shaped flowchart symbol represents the data that is stored on a storage service that can allow for user searches and filters. For instance, the data need to run a real estate app can be housed within a storage service for users to search and filter results by price, location and other search criteria.

### **10. Stored data symbol**

Also referred to as the data storage symbol, this flowchart symbol represents where the data is stored within a workflow process. For instance, a software developer producing a new program for accounting might use this symbol to represent the specific location of various financial data within different calculation processes.

### **11. Merge symbol**

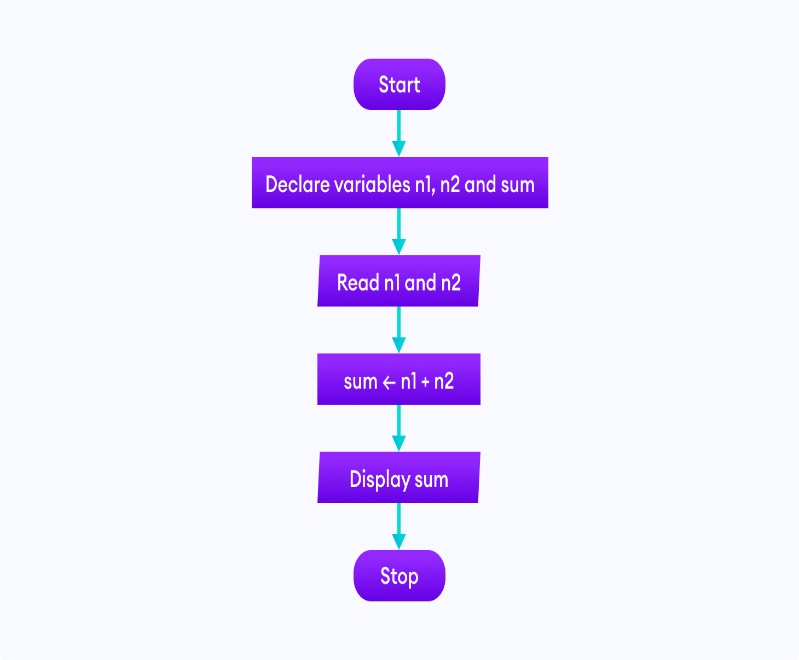
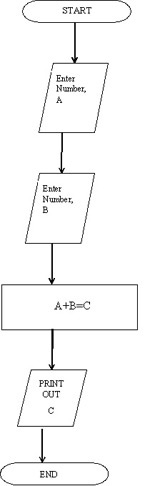
This triangle shape is used to symbolize the merging of multiple paths into one single process path.

### **12. Display symbol**

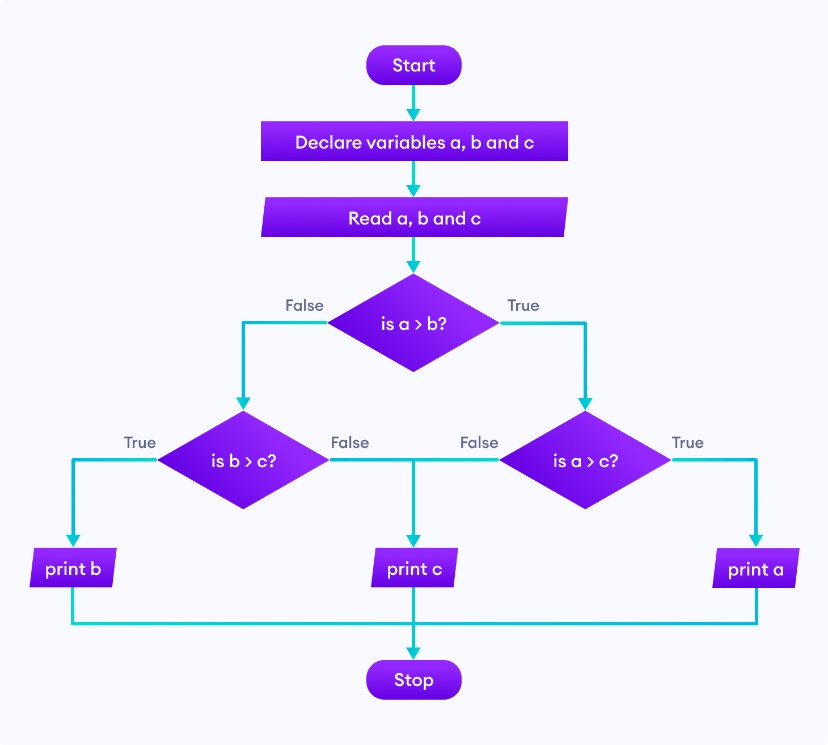
This symbol is used in a flowchart to show where data or information will be displayed within the process.

### **2.8 EXAMPLES OF FLOWCHARTS IN PROGRAMMING**

**1. Add two numbers entered by the user.**

 OR 

**2. Find the largest among three different numbers entered by the user.**



**2.9 DATA FLOW DIAGRAM**

Data flow diagrams are used to graphically represent the flow of data in a business information system. DFD describes the processes that are involved in a system to transfer data from the input to the file storage and reports generation. .A data flow diagram (DFD) maps out the flow of information for any process or system. It uses defined symbols like rectangles, circles and arrows, plus short text labels, to show data inputs, outputs, storage points and the routes between each destination.

**2.10 PSEUDOCODE**

Pseudocode is a simple way of writing programming code in English. Pseudocode is not actual programming language. It uses short phrases to write code for programs before you actually create it in a specific language. Once you know what the program is about and how it will function, then you can use pseudocode to create statements to achieve the required results for your program.

Pseudocode makes creating programs easier. Programs can be complex and long; preparation is the key. For years, flowcharts were used to map out programs before writing one line of code in a language. However, they were difficult to modify and with the advancement of programming languages, it was difficult to display all parts of a program with a flowchart. It is challenging to find a mistake without understanding the complete flow of a program. That is where pseudocode becomes more appealing.

To use pseudocode, all you do is write what you want your program to say in English. Pseudocode allows you to translate your statements into any language because there are no special commands and it is not standardized. Writing out programs before you code can enable you to better organize and see where you may have left out needed parts in your programs. All you have to do is write it out in your own words in short statements. Let's look at some examples.

## **2.12 EXAMPLES OF PSEUDOCODE**

1) A Pseudocode to create a program to add 2 numbers together and then display the result.

Start Program  
Enter two numbers, A, B  
Add the numbers together  
Print Sum  
End Program

2) A pseudocode to compute the area of a rectangle:

Get the length, l, and width, w  
Compute the area = l\*w  
Display the area

3) A pseudocode to compute the perimeter of a rectangle:

Enter length, l  
Enter width, w  
Compute Perimeter = 2\*l + 2\*w  
Display Perimeter of a rectangle

**3.0 SELECTION CONTROL STRUCTURE**

A control structure is like a block of programming that analyses variables and chooses a direction in which to go based on given parameters. The term flow control details the direction the program takes (which way program control "flows"). Hence it is the basic decision-making process in computing.

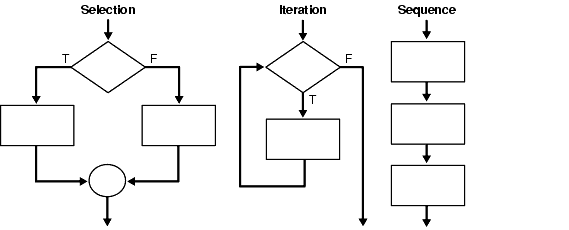
**3.1 Types of control structures**

Flow of control through any given function is implemented with three basic types of control structures:

* **Sequential**: default mode.

Sequence is the default control structure; instructions are executed one after another. They might, for example, carry out a series of arithmetic operations, assigning results to variables, to find the roots of a quadratic equation ax2 + bx + c = 0. The sequence control structure is the simplest of the three fundamental control structures that you learned about here.

* **Selection**: used for decisions, branching -- choosing between 2 or more alternative paths. Selection determines which path a program takes when it is running...
* **Repetition**: used for looping, i.e. repeating a piece of code multiple times in a row.



**3.2 CONTROL STRUCTURE EXAMPLES**

1. Simple Algorithms That Use The Sequence Control Structure

Example 3.1 Add Three numbers  
A defining diagram  
B Solution algorithm  
Example 3.2 Find average temperature  
A Defining diagram  
B Solution algorithm  
Example 3.3 Calculate mowing time  
A Defining diagram  
B Solution algorithm

1. Flowcharts and the Selection Control Structure

Simple IF statement  
Null ELSE statement  
Combined IF statement  
Nested IF statement

1. Simple Alogrithms That Use the Selection Control Structure

Example 4.1 Read three characters  
A Defining diagram  
B Solution algorithm  
Example 4.2 Process customer record  
A Defining diagram  
B Solution algorithm  
Example 4.3 Calculate employee's pay  
A Defining diagram  
B Solution algorithm

The CASE Structure Expressed as a Flowchart

Example 4.4 Process customer record  
A Defining diagram  
B Solution Algorithm

1. Simple Algorithms That Use the Repetition Control Structure

Example 5.1 Fahrenheit-Celsius conversion  
A Defining diagram  
B Solution algorithm  
Example 5.2 Print examination scores  
A Defining diagram  
B Solution algorithm  
Example 5.3 Process student enrolments  
A Defining diagram  
B Solution algorithm  
Example 5.4 Process inventory items  
A Defining diagram  
B Solution algorithm

* 1. **MODULAR PROGRAM DESIGN PRINCIPLES**

Modular programming is the process of subdividing a computer program into separate sub-programs(Module).

**A module** is a separate software component. It can often be used in a variety of applications and functions with other components of the system. Similar functions are grouped in the same unit of programming code and separate functions are developed as separate units of code so that the code can be reused by other applications.

**The benefits of modular programming are**

* Less code has to be written.
* A single procedure can be developed for reuse, eliminating the need to retype the code many times.
* Programs can be designed more easily because a small team deals with only a small part of the entire code.
* Modular programming allows many programmers to collaborate on the same application.
* The code is stored across multiple files.
* Code is short, simple and easy to understand.
* Errors can easily be identified, as they are localized to a subroutine or function.
* The same code can be used in many applications.
* The scoping of variables can easily be controlled.

# **4.1 TOP-DOWN DESIGN**

Top-down is a programming style, in which design begins by specifying complex pieces and then dividing them into successively smaller pieces. In the top-down approach, a complex algorithm is broken down into smaller fragments, better known as ‘modules.’ These modules are then further broken down into more smaller fragments until they can no longer be fragmented. This process is called ‘modularization.’ However, during the modularization process, you must always maintain the integrity and originality of the algorithm.

By breaking a bigger problem into smaller fragments, the top-down approach minimizes the complications usually incurred while designing algorithms. Furthermore, in this approach, each function in a code is unique and works independently of other functions. The top-down approach is heavily used in the C programming language.

### **4.2 BENEFITS OF TOP-DOWN DESIGN**

* Breaking a problem down into smaller parts/tasks makes it far easier to understand, solve and manage
* Top down design allows several programmers or teams to work on the same project, without getting in each other’s way
* Each module of code to be tested separately

Note: Once all the modules are linked, the whole program is tested again to make sure there are no clashes.

# **4.3 EXAMPLE OF A TOP DOWN PROGRAM DESIGN FOR A PAYROLL MANAGEMENT SYSTEM**

**OBTAIN INPUT**

**PAYROLL**

**CALCULATE PAY**

**PRODUCE OUTPUT**

**TIME CARD**

**EMPLOYEESRECORDS**

**COMPUTE PAY**

**COMPUTE DEDUCTION**

**PAYROLL REPORT**

**PAYSLIP**

This is an example of a Top-Down program design for a Payroll program. Each of the boxes shown is a module (correspond to program modules). Under the rules of Top-Down design, each module should have a single function. The program must pass in sequence from one module to the next until all modules have being processed by the computer. Three boxes – obtain input, calculate pay and produce output – correspond to three principal computer system operations. They are input, process and output.

**7.0 DEBUGGING AND MAINTAINING PROGRAM**

**Definition:** Debugging is the process of detecting and removing of existing and potential errors (also called as ‘bugs’) in a software code that can cause it to behave unexpectedly or crash. To prevent incorrect operation of a software or system, debugging is used to find and resolve bugs or defects. When various subsystems or modules are tightly coupled, debugging becomes harder as any change in one module may cause more bugs to appear in another. Sometimes it takes more time to debug a program than to code it.

7.1 **DEBUGGING PROCESS:**   
  
1. Reproduce the problem.   
  
2. Describe the bug. Try to get as much input from the user to get the exact reason.   
  
3. Capture the program snapshot when the bug appears. Try to get all the variable values and states of the program at that time.   
  
4. Analyse the snapshot based on the state and action. Based on that try to find the cause of the bug.   
  
5. Fix the existing bug, but also check that any new bug does not occur.

**7.2 SOURCES OF BUGS IN A PROGRAM**

With coding errors, the source of the problem lies with the person who implements the code. Examples of coding errors include:

* Typographical error.
* Calling the wrong function (“moveUp”, instead of “moveDown”)
* Using the wrong variable names in the wrong places (“moveTo(y, x)” instead of “moveTo(x, y)”)
* Failing to initialize a variable (“y = x + 1″, where x has not been set)
* Skipping a check for an error return

**7.3 TYPES OF ERROR**

## **1. Logical Errors**

A logic error (or logical error) is a mistake in a program's [source code](https://techterms.com/definition/sourcecode) that results in incorrect or unexpected behavior. It is a type of [runtime error](https://techterms.com/definition/runtime_error) that may simply produce the wrong [output](https://techterms.com/definition/output) or may cause a program to crash while running.

Logical errors are the hardest of all error types to detect. They do not cause the program to crash or simply not work at all, they cause it to “misbehave” in some way, rendering wrong output of some kind. One [example](http://www.hedgethink.com/hedge-fund-performance-returns-to-positive-ground/" \t "_blank) of a logic error is null reference.

Incorrect: if ($i=1) { ... }

Correct: if ($i==1) { ... }

## **2. Syntax Errors**

## A syntax error is an error in the source code of a program. Since computer programs must follow strict syntax to compile correctly, any aspects of the code that do not conform to the syntax of the programming language will produce a syntax error.

## MATLAB itself will flag syntax errors and give an error message.

## Semantic errors are improper

## **3. Semantic Errors**

Uses of “program statements.” Though different definitions of semantic error exist, we are saying here that logic errors produce wrong data while semantic errors produce nothing meaningful at all. Semantic errors have to do with meaning/context. It’s like using the wrong word in the wrong place in a human language sentence. A [computer language](http://www.hedgethink.com/fintech-revolution-ai-iot-driven-blockchain-tsunami/" \t "_blank) example would be confusing a metric with an imperial input value.

1. **Run-time error**

A runtime error is an application error that occurs during program execution. When a user executes your program and the code doesn’t work as anticipated, a runtime error occurs. Code might work correctly on your machine, but the web server or the end user might have a different configuration or it might interact with other software in a way that could cause a runtime error.

**7.4 DEBUGGING TECHNIQUES**

To perform the debugging process easily and efficiently, it is necessary to follow some techniques. The most commonly used debugging strategies are,

* Debugging by brute force
* Induction strategy
* Deduction strategy
* Backtracking strategy and
* Debugging by testing.

**Debugging by brute force** is the most commonly used technique. This is done by taking memory dumps of the program which contains a large amount of information with intermediate values and analyzing them, but analyzing the information and finding the bugs leads to a waste of time and effort.

**Induction strategy** includes the Location of relevant data, the Organization of data, the Devising hypothesis (provides possible causes of errors), and the Proving hypothesis.

**Deduction strategy** includes Identification of possible causes of bugs or hypothesis Elimination of possible causes using the information Refining of the hypothesis( analyzing one-by-one)

**The backtracking strategy** is used to locate errors in small programs. When an error occurs, the program is traced one step backward during the evaluation of values to find the cause of bug or error.

**Debugging by testing** is the conjunction with debugging by induction and debugging by deduction technique. The test cases used in debugging are different from the test cases used in the testing process.

### **7.5 DEBUGGING TOOLS**

A software tool or program used to test and debug the other programs is called a debugger or a debugging tool. These tools analyze the test run and find the lines of codes that are not executed. Simulators in other debugging tools allow the user to know about the display and behavior of the operating system or any other computing device. Most of the open-source tools and scripting languages don’t run an IDE and they require the manual process.

Mostly used **Debugging Tools** are GDB, DDD, and Eclipse.

* **GDB  Tool:** This type of tool is used in Unix programming. GDB is pre-installed in all Linux systems if not, it is necessary to download the GCC compiler package.
* **DDD  Tool:** DDD means Data Display Debugger, which is used to run a Graphic User Interface (GUI) in Unix systems.
* **Eclipse:** An IDE  tool is the integration of an editor, build tool, debugger and other development tools. IDE is the most popular Eclipse  tool. It works more efficiently when compared to the DDD, GDB and other tools.

**Other debugging tools is listed below.**

* DBG is a PHP Debugger and Profiler
* Distributed Debugging Tool (Allinea DDT)
* DDTLite — Allinea DDTLite for Visual Studio 2008
* Debugger OpenGL, OpenGL ES, and OpenCL Debugger and Profiler. For Windows, Linux, Mac OS X, and iPhone
* GNU Debugger (GDB), GNU Binutils
* Java Platform Debugger Architecture source Java debugger
* VB Watch Debugger — debugger for Visual Basic 6.0
* Microsoft Visual Studio Debugger
* Xdebug — PHP debugger and profiler

**7.6 DIFFERENCES BETWEEN DEBUGGING AND MAINTENANCE**

|  |  |
| --- | --- |
| **DEBUGGING** | **MAINTENANCE** |
| This is the process of identifying and removing errors from a software. | Maintenance is the process of making a software continue to work correctly, in the face of other modifications. |
| Debugging happens while developing the program or software. | Maintenance is what happens during the life of the code after it has been released to production. |

**8.0 GOOD PROGRAMMING PRACTICES**

**STRUCTURED PROGRAMMING**

Structured Programming Approach can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other. It doesn’t support the possibility of jumping from one instruction to some other with the help of any statement like GOTO, etc. Therefore, the instructions in this approach will be executed in a serial and structured manner.

The languages that support Structured programming approach are:

* C
* C++
* Java
* C#

The structured program mainly consists of three types of elements:

* Selection Statements
* Sequence Statements
* Iteration Statements

The structured program consists of well-structured and separated modules. But the entry and exit in a structured program is a single-time event. It means that the program uses single-entry and single-exit elements. Therefore a structured program is well maintained, neat and clean program. This is the reason why the Structured Programming Approach is well accepted in the programming world.

**Advantages of Structured Programming Approach:**

1. Easier to read and understand
2. User Friendly
3. Easier to Maintain
4. Mainly problem based instead of being machine based
5. Development is easier as it requires less effort and time
6. Easier to Debug
7. Machine-Independent, mostly.

**Disadvantages of Structured Programming Approach:**

1. Since it is Machine-Independent, So it takes time to convert into machine code.
2. The converted machine code is not the same as for assembly language.
3. The program depends upon changeable factors like data-types. Therefore it needs to be updated with the need on the go.
4. Usually the development in this approach takes longer time as it is language-dependent. Whereas in the case of assembly language, the development takes lesser time as it is fixed for the machine.

# **8.1 Software Documentation**

Software documentation is written material, images, or video instructions that come with computer software. As a rule, software documentation explains how to use a program or a service.

However, there may be different types of software documentation, depending on the audience it is created for. Here are some examples of the software documentation types:

* Requirements documentation. Typically created in the beginning of a software development project. Has the goal to clearly and precisely specify the expectations in regards to the software being created. May include functional requirements, limitations, hardware or software requirements, compatibility requirements, and so on.
* Architecture documentation. Defines the high-level architecture of the software system being created. May describe the main components of the system, their roles and functions, as well as the data and control flow among those components.
* Technical documentation - Documentation of the software code, algorithms, APIs. Written for the technical audience like software developers.
* End user documentation - Refer to [User Guide](https://clickhelp.com/software-documentation-glossary/user-guide/).

## **8.2 PROGRAM DOCUMENTATION TECHNIQUES**

There’s a certain process developed over the years for creating quality program documentation. Some steps can alter, but the skeleton remains the same. The most common documentation authoring process includes five steps:

1. First, there’s the user analysis. At this stage the basic characteristics of the users are researched. This is highly important to know for whom the documentation is written order to tailor it out for them.
2. On this stage, the planning is done and the documents are actually written.
3. The documentation draft review is performed. Documentation writers gather feedback on the draft created at the previous step.
4. Usability testing. The documentation usability is tested. The QA team can be included into this step for higher [efficiency of help authoring](http://testing-companies.com/qa-engineers-in-the-software-documentation-workflow/" \t "_blank).
5. The final step is editing. All the information that was collected on steps 3 and 4 is analysed in order to produce the final draft.

**8.3 GRAPHICAL USER INTERFACE**

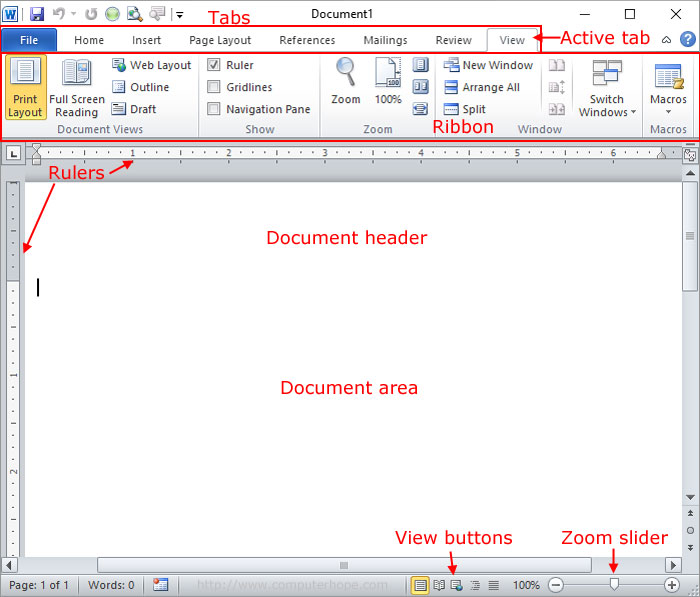
**Graphical user interface (GUI)**, a [computer program](https://www.britannica.com/technology/computer-program) that enables a person to communicate with a computer through the use of symbols, visual [metaphors](https://www.merriam-webster.com/dictionary/metaphors), and pointing devices. Best known for its implementation in [Apple Inc.](https://www.britannica.com/topic/Apple-Inc)’s Macintosh and [Microsoft Corporation](https://www.britannica.com/topic/Microsoft-Corporation)’s Windows [operating system](https://www.britannica.com/technology/operating-system), the GUI has replaced the [arcane](https://www.merriam-webster.com/dictionary/arcane) and difficult textual interfaces of earlier computing with a relatively intuitive system that has made computer operation not only easier to learn but more pleasant and natural. The GUI is now the standard computer interface, and its components have themselves become unmistakable cultural [artifacts](https://www.merriam-webster.com/dictionary/artifacts).

GUI objects include [icons](https://www.computerhope.com/jargon/i/icon.htm), [cursors](https://www.computerhope.com/jargon/c/cursor.htm), and [buttons](https://www.computerhope.com/jargon/b/button.htm). These graphical elements are sometimes enhanced with sounds, or visual effects like [transparency](https://www.computerhope.com/jargon/t/transpar.htm) and [drop shadows](https://www.computerhope.com/jargon/d/dropshad.htm). A GUI is considered to be more [user-friendly](https://www.computerhope.com/jargon/u/userfrie.htm) than a text-based [command-line interface](https://www.computerhope.com/jargon/c/commandi.htm), such as [MS-DOS](https://www.computerhope.com/jargon/m/msdos.htm), or the [shell](https://www.computerhope.com/jargon/s/shell.htm) of [Unix-like](https://www.computerhope.com/jargon/u/unix-like.htm) operating systems.

The GUI was first developed at [Xerox PARC](https://www.computerhope.com/jargon/x/xparc.htm) by [Alan Kay](https://www.computerhope.com/people/alan_kay.htm), [Douglas Engelbart](https://www.computerhope.com/people/douglas_engelbart.htm), and a group of other researchers in [1981](https://www.computerhope.com/history/1981.htm). Later, [Apple](https://www.computerhope.com/comp/apple.htm) introduced the [Lisa computer](https://www.computerhope.com/jargon/l/lisa-computer.htm) with a GUI on January 19, [1983](https://www.computerhope.com/history/1983.htm).



*Windows 7 [desktop](https://www.computerhope.com/jargon/d/desktop.htm) and an example of a GUI*



*Microsoft Word 2010 document GUI*

**8.4 INTERACTIVE PROCESSING**

Interactive processing is a Computer processing in which the user can modify the operation appropriately while observing results at critical steps. Interactive processing means that the person needs to provide the computer with instructions whilst it is doing the processing.

Interactive Processing accepts and responds to input from user , direct user interaction whilst a program is running “question and answer” between user and computer gives the user an immediate response.

Interactive Processing allows customers to check whether the goods they want are in stock. when they enter the item number they wish to purchase, the system will immediately tell them whether the item is available.

**Examples**: Booking concert tickets Ordering books online Handling bank accounts Booking a holiday -fill in a web form, -submit it -site informs you of the room you have booked.

**Another example**, imagine that a computer is running a program that takes a set of files from one directory and does some work on each one. As each file is processed the computer sends a screen message to the operator "Where do you want this file to be stored". i.e the user 'interacts' with the computer to complete the processing.

**9.0 OBJECT ORIENTED PROGRAMMING**

Object-oriented programming (OOP) is a computer programming model that organizes software design around data, or [objects](https://searchapparchitecture.techtarget.com/definition/object), rather than functions and logic. An object can be defined as a data field that has unique attributes and behavior.

Object-oriented programming – As the name suggests uses [objects](https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/" \l "objects) in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

OOP focuses on the objects that developers want to manipulate rather than the logic required to manipulate them. This approach to programming is well-suited for [programs](https://searchsoftwarequality.techtarget.com/definition/program) that are large, complex and actively updated or maintained.

### **9.1 FEATURES OF OOP**

**[Abstraction](https://www.geeksforgeeks.org/abstraction-in-c/" \t "_blank)**: Abstraction means displaying only essential information and hiding the details. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation.

Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car but he does not know about how on pressing accelerator the speed is actually increasing, he does not know about the inner mechanism of the car or the implementation of accelerator, brakes etc in the car. This is what abstraction is.

**Encapsulation:** This means that a group of related properties, methods, and other members are treated as a single unit or object.

**Inheritance:** Describes the ability to create new classes based on an existing class.

**Polymorphism:** means that you can have multiple classes that can be used interchangeably, even though each class implements the same properties or methods in different ways. A person at the same time can have different characteristic. Like a man at the same time is a father, a husband, an employee. So the same person posses different behaviour in different situations. This is called polymorphism

### **Object-oriented programming languages**

While [Simula](https://whatis.techtarget.com/definition/Simula-simulation-language) is credited as the first object-oriented programming language, the most popular OOP languages are:

* [Java](https://www.theserverside.com/definition/Java)
* [JavaScript](https://www.theserverside.com/definition/JavaScript)
* [Python](https://whatis.techtarget.com/definition/Python)
* [C++](https://searchsqlserver.techtarget.com/definition/C)
* [Visual Basic .NET](https://searchwindevelopment.techtarget.com/definition/Visual-Basic-NET)
* [Ruby](https://whatis.techtarget.com/definition/Ruby)
* [Scala](https://searchbusinessanalytics.techtarget.com/definition/Scala-Scalable-Language)
* [PHP](https://whatis.techtarget.com/definition/Personal-Home-Page-PHP?_ga=2.262457871.282285998.1608077432-315944932.1607499123)

**9.2 ADVANTAGES OF OBJECT ORIENTED PROGRAMMING**

* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the C++ code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
* OOP makes it possible to create full reusable applications with less code and shorter development time

# **9.3 PROPERTIES, OBJECTS, EVENTS, METHODS AND CLASSES**

## **Objects**

An object represents an element of Word, such as a document, a paragraph, a bookmark, or a single character. A collection is an object that contains several other objects, usually of the same type; for example, all the bookmark objects in a document are contained in a single collection object. By using properties and methods, you can modify a single object or a whole collection of objects.

## **Property**

A property is an attribute of an object or an aspect of its behavior. For example, properties of a document include its name, its content, and its save status, and whether change tracking is turned on. To change the characteristics of an object, you change the values of its properties. A **property of an object** can be explained as a variable that is attached to the **object**. The **properties of an object** define the characteristics of the **object**.

To set the value of a property, follow the reference to an object with a period, the property name, an equal sign, and the new property value. The following example turns on change tracking in the document named "MyDoc.doc".

**VB**

Sub TrackChanges()

Documents("Sales.doc").TrackRevisions = True

End Sub

In this example, Documents refers to the collection of open documents, and the name "Sales.doc" identifies a single document in the collection. The **[TrackRevisions](https://docs.microsoft.com/en-us/office/vba/api/word.document.trackrevisions)** property is set for that single document.

Some properties cannot be set. The Help topic for a property indicates whether that property can be set (read/write) or can only be read (read-only).

You can return information about an object by returning the value of one of its properties. The following example returns the name of the active document.

VB

Sub GetDocumentName()

Dim strDocName As String

strDocName = ActiveDocument.Name

MsgBox strDocName

End Sub

In this example, ActiveDocument refers to the document in the active window in Word. The name of that document is assigned to the variable refers to the document in the active window in Word. The name of that document is assigned to the variable strDocName.

**Method**

A **method** is a procedure or function associated with a class. As part of a class, a method defines a particular behavior of a class instance. A class can have more than one method. The idea of methods appears in all object-oriented programming languages. Methods are similar to functions or procedures in other programming languages such as C, SQL and Delphi.

Methods are the actions that perform operations on a variable. A method accepts parameters as arguments, manipulates these, and then produces an output when the method is called on an **object**.

## **Visual Basic Methods Example**

Following is the example of using the methods in a visual basic programming language.

 Module Module1

    Sub Main()

        Dim result As String = GetUserDetails("Suresh Dasari", 31)

        Console.WriteLine(result)

        GetDetails()

        Console.ReadLine()

    End Sub

    Public Sub GetDetails()

        Console.WriteLine("Press Enter Key to Exit..")

    End Sub

    Public Function GetUserDetails(ByVal name As String, ByVal age As Integer) As String

        Dim info As String = String.Format("Name: {0}, Age: {1}", name, age)

        Return info

    End Function

End Module

**Event**

In oop, an event is an action that occurs as a result of the user or another source, such as a mouse click. An event handler is a routine that deals with the event, allowing a programmer to write code that will be executed when the event occurs.

In Vb.net, Events are basically a user action like key press, clicks, mouse movements, etc., or some occurrence like system generated notifications. Applications need to respond to events when they occur.

Public Class Form1

Private Sub Form1\_Load(sender As Object, e As EventArgs) Handles MyBase.Load

' Set the caption bar text of the form.

Me.Text = "tutorialspont.com"

End Sub

Private Sub txtID\_MouseEnter(sender As Object, e As EventArgs)\_

Handles txtID.MouseEnter

'code for handling mouse enter on ID textbox

txtID.BackColor = Color.CornflowerBlue

txtID.ForeColor = Color.White

End Sub

End Class

**Class**

A **class** describes the contents of the **objects** that belong to it: it describes an aggregate of data fields (called instance variables), and defines the operations (called methods). **object**: an **object** is an element (or instance) of a **class**; **objects** have the behaviors of their **class**. It can also be defined as a **class** is a blueprint that defines the variables and the methods common to all objects of a certain kind. The **class** for our bicycle example would **declare** the instance variables necessary to contain the current gear, the current cadence, and so on, for each bicycle object.

### Java Example

Create an object called "myObj" and print the value of x:

public class Main {

int x = 5;

public static void main(String[] args) {

Main **myObj** = new Main();

System.out.println(myObj.x);

}

}

**9.4 HOW OPP IS IMPLEMENTED IN VISUAL BASIC**

Visual Basic supports object-oriented programming.

*Object-oriented programming (OOP)* is a programming paradigm that uses objects and their interactions to design applications and computer programs.

There are some basic programming concepts in OOP:

* Abstraction
* Polymorphism
* Encapsulation
* Inheritance

The *abstraction* is simplifying complex reality by modeling classes appropriate to the problem. The *polymorphism* is the process of using an operator or function in different ways for different data input. The *encapsulation* hides the implementation details of a class from other objects. The *inheritance* is a way to form new classes using classes that have already been defined.

## **Objects**

Objects are basic building blocks of a Visual Basic OOP program. An object is a combination of data and methods. In a OOP program, we create objects. These objects communicate together through methods. Each object can receive messages, send messages and process data.

There are two steps in creating an object. First, we create a class. A class is a template for an object. It is a blueprint, which describes the state and behavior that the objects of the class all share. A class can be used to create many objects. Objects created at runtime from a class are called instances of that particular class.

Option Strict On

Module Example

Class Being

End Class

Sub Main()

Dim b as New Being

Console.WriteLine(b.ToString())

End Sub

End Module

In our first example, we create a simple object.

Class Being

End Class

This is a simple class definition. The body of the template is empty. It does not have any data or methods.

Dim b as New Being

We create a new instance of the Being class. For this we have the New keyword. The b variable is the handle to the created object.

Console.WriteLine(b.ToString())

The ToString() method of the object gives some basic description of the object.

$ ./object.exe

Example+Being

We don't get much info, since the class definition was empty. We get the object class name and the module name, where the instance of this object was created.

## Object attributes

Object attributes is the data bundled in an instance of a class. The object attributes are called instance variables or member fields. An instance variable is a variable defined in a class, for which each object in the class has a separate copy.

Option Strict On

Module Example

Class Person

Public Name As String

End Class

Sub Main()

Dim p1 as New Person

p1.Name = "Jane"

Dim p2 as New Person

p2.Name = "Beky"

Console.WriteLine(p1.Name)

Console.WriteLine(p2.Name)

End Sub

End Module

In the above Visual Basic code, we have a Person class with one member field.

Class Person

Public Name As String

End Class

We declare a Name member field. The Public keyword specifies that the member field will be accessible outside the Class End Class block.

Dim p1 as New Person

p1.Name = "Jane"

We create an instance of the Person class. And set the Name variable to "Jane". We use the dot operator to access the attributes of objects.

Dim p2 as New Person

p2.Name = "Beky"

We create another instance of the Person class. Here we set the variable to "Beky".

Console.WriteLine(p1.Name)

Console.WriteLine(p2.Name)

We print the contents of the variables to the console.

$ ./person.exe

Jane

Beky

We see the output of the program. Each instance of the Person class has a separate copy of the Name member field.

## **Methods**

Methods are functions/procedures defined inside the body of a class. They are used to perform operations with the attributes of our objects. Methods are essential in encapsulation concept of the OOP paradigm. For example, we might have a Connect method in our AccessDatabase class. We need not to be informed how exactly Connect connects to the database. We only know that it is used to connect to a database. This is essential in dividing responsibilities in programming, especially in large applications.

Option Strict On

Module Example

Class Circle

Public Radius As Integer

Public Sub SetRadius(ByVal Radius As Integer)

Me.Radius = Radius

End Sub

Public Function Area() As Double

Return Me.Radius \* Me.Radius \* Math.PI

End Function

End Class

Sub Main()

Dim c As New Circle

c.SetRadius(5)

Console.WriteLine(c.Area())

End Sub

End Module

In the code example, we have a Circle class. We define two methods.

Public Radius As Integer

We have one member field. It is the Radius of the circle. The Public keyword is an access specifier. It tells that the variable is fully accessible from the outside world.

Public Sub SetRadius(ByVal Radius As Integer)

Me.Radius = Radius

End Sub

This is the SetRadius() method. It is a normal Visual Basic procedure. The Me variable is a special variable, which we use to access the member fields from methods.

Public Function Area() As Double

Return Me.Radius \* Me.Radius \* Math.PI

End Function

The Area() method returns the area of a circle. The Math.PI is a built-in constant.

$ ./circle.exe

78.5398163397448

Running the example.

## Access modifiers

Access modifiers set the visibility of methods and member fields. Visual Basic has five access modifiers: Public, Protected, Private, Friend, and ProtectedFriend. Public members can be accessed from anywhere. Protected members can be accessed only within the class itself and by inherited and parent classes. Friend members may be accessed from within the same assembly (exe or DLL). ProtectedFriend is a union of protected and friend modifiers.

Access modifiers protect data against accidental modifications. They make the programs more robust.

Option Strict On

Module Example

Class Person

Public Name As String

Private Age As Byte

Public Function GetAge() As Byte

Return Me.Age

End Function

Public Sub SetAge(ByVal Age As Byte)

Me.Age = Age

End Sub

End Class

Sub Main()

Dim p as New Person

p.Name = "Jane"

p.setAge(17)

Console.WriteLine("{0} is {1} years old", \_

p.Name, p.GetAge)

End Sub

End Module

In the above program, we have two member fields. One is declared Public, the other Private.

Public Function GetAge() As Byte

Return Me.Age

End Function

If a member field is Private, the only way to access it is via methods. If we want to modify an attribute outside the class, the method must be declared Public. This is an important aspect of data protection.

Public Sub SetAge(ByVal Age As Byte)

Me.Age = Age

End Sub

The SetAge() method enables us to change the private Age variable from outside of the class definition.

Dim p as New Person

p.Name = "Jane"

We create a new instance of the Person class. Because the Name attribute is Public, we can access it directly. However, this is not recommended.

p.setAge(17)

The SetAge() method modifies the Age member field. It cannot be accessed or modified directly, because it is declared Private.

Console.WriteLine("{0} is {1} years old", \_

p.Name, p.GetAge)

Finally, we access both members to build a string.

$ ./modifiers.exe

Jane is 17 years old

Running the example.

Option Strict On

Module Example

Class Base

Public Name As String = "Base"

Protected Id As Integer = 5323

Private IsDefined As Boolean = True

End Class

Class Derived

Inherits Base

Public Sub Info()

Console.WriteLine("This is Derived Class")

Console.WriteLine("Members inherited:")

Console.WriteLine(Me.Name)

Console.WriteLine(Me.Id)

'Console.WriteLine(Me.IsDefined)

End Sub

End Class

Sub Main()

Dim drv As Derived = New Derived

drv.Info()

End Sub

End Module

In the preceding program, we have a Derived class, which inherits from the Base class. The Base class has three member fields. All with different access modifiers. The IsDefined member is not inherited. The Private modifier prevents this.

Class Derived

Inherits Base

The class Derived inherits from the Base class.

Console.WriteLine(Me.Name)

Console.WriteLine(Me.Id)

'Console.WriteLine(Me.IsDefined)

The Public and the Protected members are inherited by the Derived class. They can be accessed. The Private member is not inherited. The line accessing the member field is commented. If we uncommented the line, it would not compile.

$ ./protected.exe

This is Derived Class

Members inherited:

Base

5323

Running the program, we receive this output. The Public and Protected members are inherited, the Private member is not.

## **Inheritance**

The inheritance is a way to form new classes using classes that have already been defined. The newly formed classes are called derived classes, the classes that we derive from are called base classes. Important benefits of inheritance are code reuse and reduction of complexity of a program. The derived classes (descendants) override or extend the functionality of base classes (ancestors).

Option Strict On

Module Example

Class Being

Sub New()

Console.WriteLine("Being is created")

End Sub

End Class

Class Human

Inherits Being

Sub New()

Console.WriteLine("Human is created")

End Sub

End Class

Sub Main()

Dim h As New Human

End Sub

End Module

In this program, we have two classes. A base Being class and a derived Human class. The derived class inherits from the base class.

Class Human

Inherits Being

In Visual Basic, we use the Inherits keyword to create inheritance relations.

Dim h As New Human

We instantiate the derived Human class.

$ ./inheritance.exe

Being is created

Human is created

We can see that both constructors were called. First, the constructor of the base class is called, then the constructor of the derived class.

A more complex example follows.

Option Strict On

Module Example

Class Being

Dim Shared Count As Integer = 0

Sub New()

Count = Count + 1

Console.WriteLine("Being is created")

End Sub

Sub GetCount()

Console.WriteLine("There are {0} Beings", Count)

End Sub

End Class

Class Human

Inherits Being

Sub New()

Console.WriteLine("Human is created")

End Sub

End Class

Class Animal

Inherits Being

Sub New

Console.WriteLine("Animal is created")

End Sub

End Class

Class Dog

Inherits Animal

Sub New()

Console.WriteLine("Dog is created")

End Sub

End Class

Sub Main()

Dim h As New Human

Dim d As New Dog

d.GetCount()

End Sub

End Module

We have four classes. The inheritance hierarchy is more complicated. The Human and the Animal classes inherit from the Being class. And the Dog class inherits directly from the Animal class and indirectly from the Being class. We also introduce a concept of a Shared variable.

Dim Shared Count As Integer = 0

We define a Shared variable. Shared members are members that are shared by all instances of a class. In other programming languages, they are called static members.

Sub New()

Count = Count + 1

Console.WriteLine("Being is created")

End Sub

Each time the Being class is instantiated, we increase the Count variable by one. This way we keep track of the number of instances created.

Class Animal

Inherits Being

...

Class Dog

Inherits Animal

...

The Animal inherits from the Being and the Dog inherits from the Animal. Indirectly, the Dog inherits from the Being as well.

Dim h As New Human

Dim d As New Dog

d.GetCount

We create instances from the Human and from the Dog classes. We call the GetCount() method of the Dog object.

$ ./inheritance2.exe

Being is created

Human is created

Being is created

Animal is created

Dog is created

There are 2 Beings

The Human object calls two constructors: the Dog object calls three constructors. There are two beings instantiated.

## **Abstract classes and methods**

Abstract classes cannot be instantiated. If a class contains at least one abstract method, it must be declared abstract too. Abstract methods cannot be implemented, they merely declare the methods' signatures. When we inherit from an abstract class, all abstract methods must be implemented by the derived class. Furthermore, these methods must be declared with the same of less restricted visibility.

Unlike Interfaces, abstract classes may have methods with full implementation and may also have defined member fields. So abstract classes may provide a partial implementation. Programmers often put some common functionality into abstract classes. And these abstract classes are later subclassed to provide more specific implementation. For example, the Qt graphics library has a QAbstractButton, which is the abstract base class of button widgets, providing functionality common to buttons. Buttons Q3Button, QCheckBox, QPushButton, QRadioButton, and QToolButton inherit from this base abstract class.

Formally put, abstract classes are used to enforce a protocol. A protocol is a set of operations, which all implementing objects must support.

Option Strict On

Module Example

MustInherit Class Drawing

Protected x As Integer = 0

Protected y As Integer = 0

Public MustOverride Function Area() As Double

Public Function GetCoordinates() As String

Return String.Format("x: {0}, y: {1}", \_

Me.x, Me.y)

End Function

End Class

Class Circle

Inherits Drawing

Private Radius As Integer

Sub New(ByVal x As Integer, ByVal y As Integer, \_

ByVal r As Integer)

Me.x = x

Me.y = y

Me.Radius = r

End Sub

Public Overrides Function Area() As Double

Return Me.Radius \* Me.Radius \* Math.PI

End Function

Public Overrides Function ToString() As String

Return String.Format("Circle, at x: {0}, y: {1}, radius: {2}", \_

Me.x, Me.y, Me.Radius)

End Function

End Class

Sub Main()

Dim c as New Circle(12, 45, 22)

Console.WriteLine(c)

Console.WriteLine("Area of circle: {0}", c.Area())

Console.WriteLine(c.GetCoordinates())

End Sub

End Module