**1. Introduction- Elementary Programming Principles**

**Elementary Programming Principles**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Introduction**  Programming is the art of designing computer programs. When designing these programs, there are a number of procedural steps that need to be observed and laid down. Previously, systems used to operate by plugging and unplugging wires on the switch board which was more of a physical adventure than automated. However, with programming devices, systems got automated due to software advancements and creations and that is how digital technology came into being courtesy of programming.  **Computer Programming**  Computers work by running a set of instructions called Programs (software or codes).  Programming involves creation of computer programs to solve a particular task. When creating computer programs, one is likely to use programming languages which create a platform to create programs. A programming language is a special set of symbols that can be translated into machine readable form by the computer when arranged in a particular sequence or order. Each language has a special sequence or order of writing characters usually referred to as syntax.  **Brief History of Programming Languages**  Evolution of programming languages  - Years 50: Creation of high-level languages ​​(closer to humans). - Years 60: Expansion of specialized languages​​. Forth. Simula I. Lisp, Cobol. Trying unsuccessfully to impose general languages​​: Algol, PL / 1. - Years 70: Duel between structured programming with Pascal and efficiency of C language. Basic generalized on personal computers from 1977, until the late 80s. - Years 80: Experimentating other ways including objects. ML. Smalltalk. On computers, we now use C, Pascal, Basic compiled. - Years 90: Generalization of object-oriented programming with the performance of microcomputers. Java, Perl, Python languages ​​in addition to microphones. - 2000s: Internet Programming (and future innovations, see end of text). - Years 2010: Concurrency and asynchronicity. JavaScript and Go languages among others help to create online fluid applications.    **First language**  Ada Lovelace and Babbage and its nephew were writing programs for the project of "difference engine", and then the "analytical engine". In 1945, the German K. Zuse, inventor of the Z3 computer would have defined an evolved language for this engine (with arrays and records). Few documents of the epoch about this language exist.  Follow this web address to learn more about evolution of programming languages: <http://www.scriptol.com/programming/history.php>  **Description of terms used in Programming**  **Source Program**  This is the program code that the programmer enters in the program editor window that is not yet translated into machine readable form. The source program is usually created using a particular programming language like Pascal, C, C++, Visual Basic, Java e.t.c.  **Object Code**  This is program code that is in machine readable form. A source code that is not in machine readable form must be translated into object code.  **Translators**  This refers to language processors such as assemblers, interpreters and compilers that convert the source program into object code.  **Assemblers**  An assembler translates assembly language into machine language that the computer can understand and execute.  **Interpreter**  An interpreter translates the source program line-by-line, allowing the CPU to execute one line before translating the next. The translated line is not stored in the computer memory. It means that every time, the program is needed for execution, it has to be translated. This method of translating programs was very common in early computers that did not have enough memory to store the object code as a file that can be executed later.  **Compiler**  A compiler translates the entire source program into object code. The object code file can be made into full executable program by carrying out another process known as **linking** which joins object code to all the other files that are needed for the execution of the program. After  the linking process, an executable file (application file) is generated. This file is stored on a storage media such as a disk with a name that has a unique extension (.EXE).  Differences between interpreters and compilers   |  |  | | --- | --- | | **Interpreters** | **Compilers** | | 1.      Translates the source program one statement at a time | Translates the entire source code at once before execution | | 2.      Translates the program each time it is run hence slower than compiling | Compiled program (object code) can be saved on a storage media and run as required, hence executes faster than interpreted programs | | 3.      Interpreted object code takes less memory compared to compiled program | Compiled programs require more memory as the object file are larger. | |

### 2. Levels of Programming Languages- Low Level Languages

|  |  |
| --- | --- |
|  | Levels of Programming languages Programming languages are categorized into two levels namely:  1.       Low- Level Languages  2.       High-Level Languages Low- Level Languages A **low-level programming language** is a programming language that provides little or no abstraction from a computer's instruction set architecture. Generally this refers to either machine code or assembly language. The word "low" refers to the small or nonexistent amount of abstraction between the language and machine language; because of this, low-level languages are sometimes described as being "close to the hardware".  Low-level languages can be converted to machine code without using a compiler or interpreter, and the resulting code runs directly on the processor. A program written in a low-level language can be made to run very quickly, and with a very small memory footprint; an equivalent program in a high-level language will be more heavyweight. Low-level languages are simple, but are considered difficult to use, due to the numerous technical details which must be remembered.  There are two languages used in low level languages namely:  a)      Machine language (First generation language)  b)      Assembly Language (Second generation language) Machine language (First generation language) Sometimes referred to as **machine code** or **object code**, **machine language** is a collection of binary digits or bits that the computer reads and interprets. Machine language is the only language a computer is capable of understanding.    Instructions are written using binary logic    They require many lines of logic to accomplish a task    This very code is hard for humans to understand unless he/she is equipped with special knowledge in machine level programming    Different CPUs have different machine codes this is in reference to the coding schemes we learned earlier like **ASCII** and **EBCDIC**    These programs are easy to execute by the CPU but hard to understand by humans Advantage Machine Language: The only advantage is that program of machine language run very fast because no translation program is required for the CPU. Disadvantages Machine Language: 1. It is very difficult to program in machine language. The programmer has to know details of hardware to write program. 2. The programmer has to remember a lot of codes to write a program which results in program errors. 3. It is difficult to debug the program.  Example:  **Machine Instruction       Machine Operation**  **00000000             Stop Program**  **00000001             Turn bulb fully on**  **00000010             Turn bulb fully off**  **00000100             Dim bulb by 10%**  **00001000             Brighten bulb by 10%**  **00010000             If bulb is fully on, skip over next instruction**  **00100000             If bulb is fully off, skip over next instruction**  **01000000             Go to start of program (address 0)** Assembly Language Sometimes referred to as **assembly** or **ASL**, **assembly language** is a low-level programming language used to interface with computer hardware.    They were developed to overcome the difficulties of understanding and using machine languages    It attempted to make computer languages readable    This language allowed programmers to write programs as a set of symbolic operation codes called Mnemonics- this is a shortened two or three letter words.    Programs written in assembly language require an assembler in order convert them into machine language that the computer can understand.    Assembly language is machine dependent- this means, a program written for one computer cannot be used on another. The following disadvantages are observed with the assembly languages.  1. It is time consuming for an assembler to write and then test the program. 2. Assembly language programs are not portable. 3. It is necessary to remember the registers of CPU and mnemonic instructions by the programmer. 4. Several mnemonic instructions are needed to write in assembly language than a single line in high-level language. Thus, assembly language programs are longer than the high language programs.     Example:  **MOV r0, #0C ;load base address of string into r0 LOAD: MOV r1,(r0) ;load contents into r1 CALL PRINT ; call a print routine to print the character in r1 INC r0 ;point to next character JMP LOAD ;load next character** |

|  |  |
| --- | --- |
|  | High level languages  * These languages are close to human languages such that humans can read and understand easily than low level languages * These languages are machine independent this means that they are not subject to hardware models and configurations * These languages are classified into five groups  namely:   I.                    Third generation languages (3GLs)  II.                  Fourth Generation Languages  (4GLS)  III.                Fifth generation Languages (5GLs)  IV.                Object Oriented Languages (OOPs)  V.                  Web scripting Languages Third generation languages (3GLs)  * These languages are also called procedural or structured languages; this means that these are languages that make it possible to break a program into components called **modules** each performing a particular task. This is referred to as structured programming * In structured programming   i.  Large programs can be broken down into smaller sub programs each performing a single task  ii.  Use of a few simple control structures in problem solving. These control structures include sequence, selection and iteration   * Structured programming languages are flexible, easier to read and modify   **Examples:**  **Pascal:** was initially developed for teaching structured languages  **Fortran:** (FORmula TRANslator) this language was developed for mathematicians, scientists and engineers. It involves writing programs with mathematical expressions.  **COBOL:** (common business oriented language); this language was developed to solve business problems e.g. developing data processing applications such as computer – based inventory control systems.  **Basic**: (Beginners All- Purpose Symbolic Instructional Code); this language was developed to initially enable students learn programming. This is a simple general purpose language used for developing business and educational applications. This was the first high level language that was available for microcomputer users.  **C**: This is a programming language mainly used for developing system software such as the operating system. It is one of the most popular and powerful high – level languages in the business world because of its ability to provide the programmer with powerful features of low level languages and at the same time easily understandable as a high level language.  **ADA:** this is a language named after the first lady programmer. Ada Lovelace. Ada is suitable for developing military, industrial and real-time systems Fourth Generation Languages (4GLs)  Fourth generation languages make programming an even easier task than the third generation language because they present the programmer with more programming tools which include: Command buttons, forms, textboxes, Combo boxes etc.  Here, selects graphical objects on the screen called controls then uses them to create designs on a base form. The programmer may also use an application generator works behind the scenes to generate the necessary code, hence the programmer is freed from tedious work of writing the code.  **Examples of fourth generation languages**  Visual Basic, Delphi Pascal and Visual COBOL Fifth Generation Languages (5 GLs) These languages are designed to depict human-like intelligence.  With these languages, the programmer only worries about what problem needs to be solved and what conditions need to be met without worrying about how to implement an algorithm to solve them.  **Examples:**  PROLOG, Mercury, Lisp and OCCAM Object-Oriented Programming Languages (OOPs) The concept behind OOPs is to look at a program as having various objects interacting to make up a whole. Each object has specific data values that are unique to it (called state) and a set of the things it can accomplish called (functions or behavior). This process of having data and functions that operate on the data within an object is called **encapsulation**. Several objects can then be linked to form a complete program.  **Examples:**  C++, Java and SmallTalk Web Scripting Languages The World Wide Web has thrown up a number of new scripting languages specifically designed for web page design. It has also seen an upsurge in popularity of some older languages originally designed for other purposes but which happen to be well-suited for web applications. The following identifies some of the major scripting languages associated with Web page design:  **Examples:**   * **HTML** (HyperText Markup Language). All web pages are written using HTML (although some of the scripting languages listed below may also be used to provide more versatility). HTML is not really a programing language. It is essentially a set of tags which can be embedded within Ascii text to control how the text should be displayed when viewed in a web browser. * **DHTML** (Dynmaic HTML). DHTML is not a language at all. Rather it is a name applied to various technologies (e.g. Cascading Style Sheets (CSS), Document Object Model (DOM) and scripting languages such as JavaScript) used to supplement HTML to provide web pages with dynamic content. DHTML does not depend upon plug-ins and should in theory run on any HTML 4 compatible browser. Processing is client-side. * **JavaScript**. Although it shares some similarities with Java, JavaScript is not a subset of Java. Java and JavaScript were developed independently. JavaScript (originally called LiveScript) is a scripting language developed by Netscape for enhancing web pages. It was probably renamed JavaScript to cash in on the popularity of Java. This has unfortunately resulted in a lot of confusion as Java may also be used to enhance webpages. JavaScript is primarily used for client-side processing. * **JScript** is a Microsoft version of JavaScript. JScript is generally used for client-side processing, but it may also be used for server-side processing. Active Server Pages, for example, may be set-up to run JScript rather than VBScript. * **VBScript** is a subset of Visual Basic for Applications (VBA). VBScript is an interpreter language, whereas VB can be compiled. VBScript is often used for server-side scripting and is the default language for Active Server Pages (ASP). It may also be used fot client-side scripting (although it is currently only supported by Internet Explorer). * **Perl** (Practical Extraction and Report Language) is a powerful text processing language developed by Larry Wall. It originated on Unix platforms, but interpreters are now available for Windows and Macintosh. Scripts can be easily ported between platforms. Although CGI scripts can be written in other languages (e.g. C or Visual Basic), Perl is often the language of choice for CGI (Common Gateway Interface) scripts. Although useful for web applications, Perl is an extremely versatile language and may also be used for a multitude of non-web related purposes. * **PerlScript** is a subset of Perl. It may be used for ActiveX and Active Server Pages. * **Tcl** (Tool Command Language) - sometimes pronounced *tickle* - was developed by John Ousterhout. Like Perl it may be used for CGI scripting or for writing applications. It may be also used to write client-side scripts which run on the user's browser. * **PHP** was created in 1994 by Rasmus Lerdorf to add dynamic content to an HTML page. PHP initially stood for 'Personal Home Page', but now it is generally translated as 'PHP Hypertext Preprocessor'. The PHP code is embedded within the HTML code between special tags. When the page is accessed the server processes the PHP code and then sends the output from the script as HTML code to the client. * **CFML** (ColdFusion Markup Language) is a tag-based scripting language developed by Allaire (now merged with Macromedia) as part of the ColdFusion system for dynamic web page creation and database access. The CFML commands are embedded in HTML files. The results of database queries are converted into HTML by the server. * **Ajax** is an abbreviation for Asynchronous JavaScript and XML. It is not a language as such, but a group of interrelated client-side web development techniques used to create interactive web applications.  Advantages and disadvantages of low-level and high level languagesAdvantages of low level languages  * The CPU understands machine language directly without translation * The processor executes them faster because complex instructions already broken down into smaller simpler ones * Low level languages are stable and hardly crash or break down once written  Disadvantages of low level languages  * They are difficult and cumbersome to use and learn * They require highly trained experts both to develop and maintain programs * Removing errors (debugging) In a low level language programs is difficult * Low level programs are machine dependent i.e. they are not transferable from one hardware or software platform to another. Thus, they are not portable  Advantages of high level languages  * High level languages are portable i.e. they are transferable from one computer to another * They are user friendly and easy to use and learn * Are more flexible. Thus, enhancing creativity and increasing productivity * Are easier to debug (correct errors)  Disadvantages of high level languages  * Due to large volume of code involved when generating, these programs are slower in processing * They have to be interpreted or compiled to machine readable form before the computer can execute them. * They require large computer memory to run it  Topical questions 1.       Define the term computer program  2.       What is programming?  3.       State three advantages of high level languages over low level languages  4.       List four examples of high level languages and for each, state its most appropriate application area  5.       Why is an executable file unique when compared to any other file?  6.       Differentiate between a compiler and an interpreter. Why did early computers work well with interpreters?  7.       List the various examples of programming languages per generation  8.       Write the following in full:  A)     HTML  B)      OOP  9.       Distinguish between source program and object code in programming  10.   State one advantage of machinery language over the other languages  11.   Define the term encapsulation as used in object oriented programming languages  12.   Distinguish between:  a)      Low level and high level languages  b)      Assembly and machine languages  c)       Third generation and forth generation languages  d)      Object Oriented Languages and Web Scripting Languages  13.   Explore necessary advantages and disadvantages of low level and high level languages stating two examples of each. |

### Program Development

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | PROGRAM DEVELOPMENTIntroduction Development of programs is the art of preparing and designing necessary structures to come up with computer instructions that can be used to accomplish a task in the computer. There are six procedural steps involved in program development namely:  1>     Problem recognition  2>     Problem definition  3>     Program design  4>     Program coding  5>     Program testing and debugging  6>     Implementation and maintenance  There should be some kind of uniformity and relevance in execution from one stage to another and in every stage, documentation is necessary. The diagram below elaborates this.  stages of program development Problem Recognition  * This is the understanding and interpretation of a particular problem. To understand these problems, you need to highlight key words like: what needs to be ***computed, saved, compared, sorted, input etc.*** * To understand these problems, the programmer should:   A>    Talk to users in order to assess their situations  B>    Understand problems that need solutions  C>    Siege an opportunity to develop the new system  D>    Study the new directive given by the management requiring a change in the   status quo  E>     Make a formal presentation of the system being designed Problem Definition  * This is also referred to as **problem analysis**. * Here the programmer tries to determine or define the likely input, processing activities and the expected output using the keyword outlined at the problem recognition stage. * Finally, the boundaries of the expected program will have been established. In case several methods are identified that can be used to solve the same problem, then the best alternative should be chosen.  Program design  * This is the actual processing of the program or problem solving logic called the algorithm. * An algorithm refers to a definite number of logical steps that a program follows in order to solve a problem. It is the programmer who will usually come up with the algorithm after carefully analyzing the requirements specifications. * Most programs are not holistic, rather monolithic which means they are made up of several units called modules which work together to form the whole. * In modular programming, each module performs a specific task. This approach makes a program flexible, easier to read and carry out error correction. * This stage enables the programmer to come up with models of the expected program. The models show the flow of events and data throughout the entire program from the time data is input to the time the program gives out expected information.  Program Coding  * This is the art of converting a program design into an equivalent program. * This is possible by coding the program with the use of a specific programming language like: Java, Pascal, C++, Visual basic etc. * To code these programs, one needs  a translator which converts source program into object code like assemblers, interpreters and compilers  Program testing and debugging  * This stage involves removing errors from the program (debugging) and execution to test its integrity. * There are two possible errors to debug:   **1>     Syntax error**  These are errors that originate from improper use of language syntax (grammar, spelling, spacing, punctuation and skipping syntax) **syntax**- is the set of rules that defines the combinations of symbols that are considered to be a correctly structured document or fragment in that language.  For example: in most programming languages, declaration lines end with a semi-colon, failure to insert a semi-colon can result into a syntax error  **Consider this PHP code:**  <?php echo "My first PHP script!"**;** ?>  If words like “echo” is misspelled like “icho” or “?” is missing at the start line or end line or quotations are missing in the statement “My first PHP script!" can lead to syntax error  **2>     Logical errors**   * + They are not detectable by the translator. The program runs but gives wrong output or halts during execution   Example: consider the PHP code above  The user may mistakenly alter the statement i.e. instead of typing echo "My first PHP script!" he/she types   echo "My second PHP script!"  The code will execute but the output will be wrong. Therefore, logical errors are done due to entry of logical parameters Methods of error detection -          There are several methods of testing the program for errors. These include:  **1>     Desk checking (dry-run)**  This involves going through the program while still on paper before entering it in the program editor. This helps the programmer to detect the most obvious syntax and logical errors  **2>     Using debugging utilities**  This involves entering the program in the program editor and trying to run debugging utilities to correct syntax errors  **3>     Using test data**  The programmer enters various data variations and extremes including data with errors to test whether the system will grind to halt Implementation and maintenanceImplementation -          This is the actual delivery and installation of the new program to be ready for use. Review and maintenance  Proper training and post implementation support of users will always reduce the chances of having those entering invalid data that can crash the program. Program documentation  * This is writing of formal support materials explaining how the program was developed * Documentation can be used by users, installed by operators and modified by programmers * All stages of the program development should be documented in order to help during future modification of the program. * Documentation is either internal or external * **Internal documentation**: is written as program comments that cannot be executed by the program editor. They help programmers to understand the source code * **External documentation**:  refers to reference materials such as user manuals printed as booklets. User manuals are common examples of external documentation. * **There are three target groups of external documentation namely:**  1. **User oriented documentation**: these type enables the user to learn how to use the program as quickly as possible and with little help from the program developer 2. **Operator oriented documentation**: it is meant for computer operators such as the technical staff. It helps them to install and maintain the program 3. **Programmer oriented documentation**: it is a detailed documentation written for skilled programmers. This documentation provides necessary technical information to help in future modification of the program. In this type of documentation, all stages of the program development should be documented because:  * There may come a need to revise or modify the program * Other programmers may think of ways of improving your program.  Summary (modern): **Six Steps of PDLC**   |  |  |  | | --- | --- | --- | | **STEP** | **PROCEDURE** | **DESCRIPTION** | | 1 | Analyze the problem | Precisely define the problem to be solved, and write program specifications – descriptions of the program’s inputs, processing, outputs, and user interface. | | 2 | Design the program | Develop a detailed logic plan using a tool such as pseudo code, flowcharts, object structure diagrams, or event diagrams to group the program’s activities into modules; devise a method of solution or algorithm for each module; and test the solution algorithms. | | 3 | Code the program | Translate the design into an application using a programming language or application development tool by creating the user interface and writing code; include internal documentation – comments and remarks within the code that explain the purpose of code statements. | | 4 | Test and debug the program | Test the program, finding and correcting errors (debugging) until it is error free and contains enough safeguards to ensure the desired results. | | 5 | Formalize the solution | Review and, if necessary, revise internal documentation; formalize and complete end-user (external) documentation | | 6 | Maintain the program | Provide education and support to end users; correct any unanticipated errors that emerge and identify user-requested modifications (enhancements). Once errors or enhancements are identified, the program development life cycle begins again at Step 1. |    Trial Questions 1.       Give one advantage of compiling a program rather than interpreting it  2.       Outline at least six stages of program development in their respective order  3.       Highlight two disadvantages of monolithic programs  4.       State two advantages of modular programming  5.       In what stage of the development does program documentation justify your answer  6.       Differentiate between a flowchart and pseudo code  7.       What is a program bug?  8.       Explain why it is important to test a program before implementing it. |

### 5. Development of Algorithm- pseudocodes

## Development of Algorithm

|  |  |
| --- | --- |
|  | Development of algorithms Development of algorithms can be defined as logical steps that a program follows in order to solve a problem.  Various tools are used to depict algorithms. They include:  -          Decision trees  -          Decision tables  -          Flowcharts and  -          Pseudo codes  Pseudo codes A pseudo code is a set of statements written in a human readable language but expressing the processing logic of a program.  -          Pseudo codes are not executable by the computer Guidelines for designing a good pseudo code To design a good pseudocode, proceed as follows   1. The statement must be short, clear and readable 2. The statements must not have more than one meaning i.e. should be unambiguous 3. The pseudo code lines should be clearly outlined and indented clearly 4. A pseudo code should show clearly the start and stop of executable statements and the control structures (to be discussed later in the section) 5. The input, output and processing statements should be clearly stated, using keywords such as   Several keywords are often used to indicate common input, output, and processing operations.  Input:          **READ, OBTAIN, GET** Output:         **PRINT, DISPLAY, SHOW**  Compute:        **COMPUTE, CALCULATE, DETERMINE**  Initialize:     **SET, INIT**  Add one:        **INCREMENT, BUMP** Example: Write a pseudo code that can be used to prompt the user to enter the diameter of a circle which will be used to calculate the perimeter and area of the circle Answers **Start**  PRINT “enter diameter parameter”         READ D  PERIMETER = 22/7\*D  AREA = 22/7\*D/2\*D/2  PRINT PERIMETER  PRINT AREA  **Stop** Trial Questions 1.       Write a pseudo code that can be used to prompt the user to enter two numbers, calculate the sum and average of the two numbers and then display the output on the screen  2.       Write a structured algorithm that would prompt the user to enter the length and width of a rectangle, calculate the area and perimeter then display the result  3.       Write a pseudo code for a program that can be used to classify people according to the age limit. If a person is more than 20 years; output “adult” else output “young person”  4.       What factors do you consider when designing a good pseudo code  5.       What is the importance of writing algorithms |

#### [5.1-FlowChart](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/5-development-of-aligorithm/51-flowchart)

# Flow Charts

Flow charts are used to represent any well defined sequence of activities. they are used to convey in diagrammatic form, the logic, processing operations and flow of control required of a computer program

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Uses Of Flowcharts Flowcharts are used by programmers to:   1. To plan the structure of a program before it is written 2. To describe the structure of a program after it has been written  Flowchart Symbols Flowchart symbols are combined with short text clues which are a form of shorthand understood by programmers. Below is a list of symbols:   |  |  |  | | --- | --- | --- | |  | **Symbol** | **Meaning** | | **1- Start/Stop** |  | Ellipse: denotes the beginning and end of the program algorithm | | **2- Input/Output** |  | Parallelogram: used to denote an input or output operation. For example, READ A, B, PRINT SUM | | **3- Process** |  | Rectangle: Indicates that a processing or data transformation is taking place. For example SUM = A+B | | **4- Decision** |  | Rhombus: Used to specify a condition. A condition must evaluate to a Boolean value (true or false) for the program to execute the next instructions | | **5- Connector** | [connector](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/5-development-of-aligorithm/51-flowchart/connector.png?attredirects=0) | Circle: Used as a connecting point or interface for arrows coming from different directions | | **6- Arrow** | [arrow](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/5-development-of-aligorithm/51-flowchart/arrow.png?attredirects=0) | Arrow: Used to indicate the direction of flow of the program logic |  Guidelines for Drawing a Flowchart  1. there should be only ***one entry/starting point*** and ***one exit point*** of the program algorithm - ***except for decisions and connectors*** 2. Use the ***correct symbol*** at each stage in the flowchart 3. The ***logical flow should be clearly shown*** using arrows. 4. Use connectors to reduce the number of flow-lines 5. Add notes if necessary, but don't clutter the diagram 6. Lines should ideally be vertical or horizontal  6. Program Control StructuresProgram Control StructuresIntroduction: program control structures are blocks of statements that determine how statements are to be executed, in structured programming languages, there are three control structures namely; *sequence, selection and iteration (looping)* Subpage Listing  * [1. Sequence](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/1sequence) * [2. Selection](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/2selection)   + [Selection Examples](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/2selection/selection-examples)   + [Selection Questions](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/2selection/selection-questions) * [Iteration (Looping)](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/iterationlooping)   + [The FOR Loop Examples](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/iterationlooping/the-for-loop-examples)   + [The FOR Loop Questions](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/iterationlooping/the-for-loop-questions) * [Review Questions](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/reviewquestions)  1. Sequence posted Jan 6, 2015, 11:59 AM by Maurice Nyamoti   [ updated Jan 6, 2015, 12:19 PM ] Sequence In this structure, the computer reads instructions from a program file starting from the first top line and proceeding downwards one -by -one to the end. This is called sequential programming execution i.e.  [sequence](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/1sequence/sequence.png?attredirects=0)  In sequential program execution, the computer perform tasks that are arranged sequentially one after another.  Just like in a school system, a std5 student must first pass ECD, std 1 - 4 before std 5 in that sequence. 2. Selection posted Jan 6, 2015, 12:36 PM by Maurice Nyamoti   [ updated May 6, 2015, 11:07 AM ] Selection In selection control, execution of statements depends on a condition which is either a **true** or false   |  |  | | --- | --- | |  | There are four types of selection controls. Namely:   1. IF.....THEN 2. IF.....THEN.....ELSE 3. NESTED IF 4. CASE SECTION  IF....THEN The IF....THEN selection is used if only one option is available. All other options are ignored  Format:  IF <condition> THEN      Statements;  ENDIF  Flowchart Sample IF....THEN....ELSE This type of selection is suitable where there are two available options  General Format IF *<condition>* THEN     *Statements;* ELSE     *Statements;* EndIF  [if....then....else](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/2selection/ifthenelse.png?attredirects=0)  Flowchart Sample Nested IF Selection This type of selection is used where two or more options have to be considered to make a selection.  **General Format** IF <condition> THEN             statements     ELSE     IF <condition> THEN             statements     ELSE         IF<condition> THEN             Statements     ELSE             Statements         ENDIF     ENDIF ENDIF  Flowchart Sample CASE Construct This is an alternative to the Nested IF especially where there are several options to choose from. Its preferred because it reduces the many lines of code. However, the boolean expression for the case selection can only be expressed using integers and alphabetic characters only. The General format should be  CASE *integer* OF or CASE *Char* OF  **Example:** Case x of  label 1: statement 1 label 2: statement 2 label 3: statement 3 . . . label n: statement n-1  Else  statement  Endcase |  Iteration (Looping)Iteration (Looping)  |  |  | | --- | --- | |  | Introduction Iteration is also referred to as looping or repetition.  Its designed to execute the same block of code again and again until a certain condition is fulfilled.  The three main looping controls are:-   1. The WHILE Loop 2. The REPEAT....UNTIL Loop 3. The FOR Loop  The WHILE Loop The "WHILE" loop is used if a condition has to be met before the statements within the loop are executed.  **General Format**  WHILE *condition* DO  *statement*  ENDWHILE  [whileloop](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/iterationlooping/whileloop.png?attredirects=0) The REPEAT....UNTIL loop The REPEAT....UNTIL loop allows the statements within it to be executed at least once since the condition is tested at the end of the loop.  **General Format**  REPEAT  *statements*  UNTIL *<condition>*  [repeatuntilloop](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/iterationlooping/repeatuntilloop.png?attredirects=0) The FOR Loop The FOR loop is used in circumstances where the execution of the chosen statements has to be repeated a predetermined number of times.  General Format  //pseudocode for 'FOR" loop that counts from the lower limit  FOR loop variable = lower limit to upper limit DO  statements  ENDFOR  [forloop1](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/iterationlooping/forloop1.png?attredirects=0)  //pseudocode for a 'FOR' loop that counts from the upper limit down to lower limit  FOR loop variable = upper limit DOWN TO lower limit DO  statements;  ENDFOR  [forloop2](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/elementary-programming-principles/6-programcontrolstructures/iterationlooping/forloop2.png?attredirects=0) | |

### SYSTEM DEVELOPMENT

### 1. What is a system?

posted May 2, 2015, 8:35 AM by Maurice Nyamoti   [ updated May 2, 2015, 8:38 AM ]

|  |
| --- |
| What is a system? A system is a set of organized components which interact in a given environment and within a specified boundary to achieve collective goals and objectives that are emergent Example of a system is a computer system. i.e. [computer system](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/system-development/system-development-introduction/1whatisasystem/1.png?attredirects=0) |

**2. Description of a System**

posted May 2, 2015, 8:56 AM by Maurice Nyamoti   [ updated May 2, 2015, 8:56 AM ]

|  |  |  |  |
| --- | --- | --- | --- |
| A system can either be described as either:   1. Soft System or 2. Hard System   **Soft System**  A system is described as soft because of the following characteristics:   1. Boundaries keep on changing 2. Goals and Objectives usually conflict due to human factors like attitudes and preferences 3. Exact measures of performance cannot be clearly captured   **Hard Systems**  A system is said to be hard because of the following characteristics:   1. Goals and objectives can clearly be defined 2. Outcomes of the systems processes are predictable and can be modeled accurately  3. Characteristics of Systems posted May 2, 2015, 9:44 AM by Maurice Nyamoti   [ updated May 2, 2015, 9:55 AM ]   |  | | --- | | All systems have some similar characteristics as illustrated below: 1. Holistic thinking A system is considered as a whole because many entities will work together to make a complex whole example: a human body is a complex whole made up of small entities like cells, chromosomes, enzymes, water, bones etc. 2. Subsystems A system is made up of different components (subsystems) example: a human body system is made up of other sub-systems like; circulatory system, digestive system, neural systems etc. 3. Boundary and Environment Each system has a space (boundary) within which the components operate. A system has two types of entities namely: internal and external entities. An internal entity operates from within the system boundary like the liver of a human body however, external entities operate from outside the boundary but are part of the system i.e. Food and Air in a human body 4. Purpose The purpose of each system is to perform a particular task or achieve a goal. Example: The purpose of a human body system is to support life 5. Process A system has the obligation to transform or process data from one state to another Example: In a human digestive system, food as transformed into nutrients like proteins and vitamins while the remaining is discharged as waste 6. System Entropy The word entropy means decay. Systems decay over time. example: A human body eventually dies after a period of time when subsystems cannot support it anymore 7. Inputs and Outputs A system communicates with environment by receiving inputs and giving outputs example: In a human body system, it takes food as an input and gives output through excretory process 8: Open VS Closed Systems A system can be described as either open or closed. An open system receives input and gives output while a closed system does not. 9. Control A system adapts to changes in the environment in order to give the expected output or to perform to the expected level. This is possible through the feedback which can be used to influence inputs example: If a human body is ill, the brain system sends a feedback (pain) this feedback will influence the person to seek medical care. |  4. Information System4. Information System  |  |  | | --- | --- | |  | * An Information System is a man-made system that facilitates an organization’s operational functions and supports management decision making by providing information that managers can use to plan and control the activities of the firm. * is the collection of technical and human resources that provide the storage, computing, distribution, and communication for the information required by all or some part of an enterprise. * Management Information System (MIS) is a planned system of collecting, storing, and disseminating data in the form of information needed to carry out the functions of management.  Purpose of information Systems  1. Supporting information processing ~ by enhancing tasks such as data collection, processing and communication 2. Helping in decision making ~ by collecting operational data, analyzing it and generating reports that can be used to support the decision making processing 3. Enable sharing of information  Why develop new information systems?  1. New opportunities ~ a chance to improve quality of internal processes and services delivery in the organisation 2. Problems ~ Need to solve problems at hand 3. Directives ~ Change in policy or rules buy the company or Government  Role of a System analyst A **systems analyst** is an IT professional who specializes in analyzing, designing and implementing information **systems**. **System analysts** assess the suitability of information **systems** in terms of their intended outcomes and liaise with end users, software vendors and programmers in order to achieve these outcomes.   1. Examine existing IT systems and business models; 2. Analyses systems requirements; 3. Implement, configure and test feasible solutions. 4. Liaise and report to internal and external clients and stakeholders, including colleagues and developers. 5. Is involved throughout the development process. The analyst acts as liaison between the client and the developers. 6. Conduct a cost analysis and agree the time frame to implement the proposed solution. They specify and shape the system requirements and operations, the user interface and output and present the proposal to the client. 7. Is the overall project manager of the information system being implemented | |

**5. Theories of Development**

|  |
| --- |
| **Subpage Listing**   * [1. Introduction on theories of system development](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/system-development/system-development-introduction/5-theories-of-development/1introductionontheoriesofsystemdevelopment) * [2. Stages of system Development](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/system-development/system-development-introduction/5-theories-of-development/2stagesofsystemdevelopment) |

**1. Introduction on theories of system development**

|  |
| --- |
| The aim of all these theories and methods is to identify business requirements and to develop information systems that effectively meet them  Some of these theories include:   1. Traditional approach 2. Rapid Application Development (RAD) 3. The structured approach   **1. Traditional Approach**  It relies mostly on the skills and experience of individual staff members carrying out the project. There are no formal procedures followed and therefore the success of the system is dependent on individual effort. In this approach, the manual system is replaced with a computerised one without change in overall structure of the former system, hence the weaknesses of the former system are not addressed and are carried forward to the new system.  **2. Rapid Application Development (RAD)**  RAD is also known as **prototyping**. A **prototype** is a sample used to test a concept or a process this is a concept that products can be developed faster and of higher quality through:   * Gathering requirements using workshops or focus groups * Prototyping and early, reiterative user testing of designs * The re-use of software components * A rigidly paced schedule that defers design improvements to the next product version * Less formality in reviews and other team communication   **Advantages of the RAD model:**   1. Reduced development time. 2. Increases re-usability of components 3. Quick initial reviews occur 4. Encourages customer feedback 5. Integration from very beginning solves a lot of integration issues.   **Disadvantages of RAD model:**   1. Depends on strong team and individual performances for identifying business requirements. 2. Only system that can be modularized can be built using RAD 3. Requires highly skilled developers/designers. 4. High dependency on modeling skills 5. Inapplicable to cheaper projects as cost of modeling and automated code generation is very high.   **3. The Structured Approach**  Structured approach involves a set of stages that should be followed when developing a system. Each stage is documented in specifying the activities that should be carried out. |

**2. Stages of system Development**

|  |
| --- |
| There are seven stages in system development lifecycle namely:  [sdlc](http://webartika.manyamfranchise.com/notes/computernotes/computerform3/system-development/system-development-introduction/5-theories-of-development/2stagesofsystemdevelopment/14.png?attredirects=0)  **Problem recognition and definition**  In this phase, the system analyst seeks answers to these two questions   1. Is the proposed system worthy pursuing? 2. Is the proposed project worthy looking at?  * **Problem definition** is performed by the senior members of the team with inputs from the customer, the sales department, market surveys and domain experts in the industry. This information is then used to plan the basic project approach and to conduct product feasibility study in the economical, operational, and technical areas. The scope of the project, constraints, budget and schedule is defined here * Problem definition is also called **problem analysis**; its the process of identifying the problem and finding out any constraints that may limit the solution * **A feasibility study (preliminary investigation)** is conducted at this stage; **Feasibility study** is an evaluation and analysis of the potential of a proposed project. It is based on extensive investigation and research to support the process of decision making.   + **Operational feasibility**; establishes the extent to which the users are comfortable or happy with the proposed or new system   + **Schedule feasibility**; establishes whether the development of the proposed system will be accomplished within the available time   + **Technical feasibility**; establishes whether the technology available is sufficient or can be upgraded for the new system. It also seeks to find out whether the staff have relevant technical skills to develop and use the new system   + **Economic feasibility**; establishes whether developing the new system is cost effective by analyzing all the costs and benefits of the proposed system. * After feasibility study has been conducted, a documentation is prepared to cover all the activities that took place.   **Information Gathering** |