**JOSHUA ETUK CSC 206 19/SCI01/095**

1. **A) Programming** is the process of taking an algorithm and encoding it into a notation.

**B)** A **computer program** is a collection of instructions that can be executed by a **computer** to perform a specific task.

**C)** A **programming language** is a vocabulary and set of grammatical rules for instructing a computer or computing device to perform specific tasks. The term **programming language** usually refers to high-level **languages**, such as BASIC, C, C++, COBOL, Java, FORTRAN, Ada, and Pascal.

1. MACHINE LANGUAGE

Most computers work by executing stored programs in a fetch-execute cycle. Machine code generally features:

* **Registers** to store values and intermediate results
* Very low-level machine instructions (add, sub, div, sqrt) which operate on these registers and/or memory
* Labels and conditional jumps to express control flow
* A lack of memory management support — programmers do that themselves

The machine instructions are carried out in the hardware of the machine, so machine code is by definition machine-dependent. Different machines have different instruction sets. The instructions and their operands are all just bits.

ASSEMBLY LANGUAGES

An assembly language is an encoding of machine code into something more readable. It assigns human-readable **labels** (or names) to storage locations, jump targets, and subroutine starting addresses, but doesn’t really go too far beyond that. It’s really isomorphic to its machine language

HIGH LEVEL LANGUAGES

A high-level language gets away from all the constraints of a particular machine. HLLs may have features such as:

* Names for almost everything: variables, types, subroutines, constants, modules
* Complex expressions (e.g. 2 \* (y^5) >= 88 && sqrt(4.8) / 2 % 3 == 9)
* Control structures (conditionals, switches, loops)
* Composite types (arrays, structs)
* Type declarations
* Type checking
* Easy, often implicit, ways to manage global, local and heap storage
* Subroutines with their own private scope
* Abstract data types, modules, packages, classes
* Exceptions
1.

**Syntax And Structure**. Programming languages for commands can overlap just like when using words in spoken languages. To produce text to screen in Ruby or [Python](https://www.python.org/about/gettingstarted/), you are to use ‘print command,’ similar to using imprimer and imprimir when we want to print in French and Spanish.

**Functionality Of Languages.**All these languages can make the same functionality, similar to how all spoken languages can reflect the same phrases, objects, and emotions.

**Natural Lifespan.**Programming languages are created when a talented programmer attempts to create a fresh way or an easier method of expressing a computational idea. He presents this idea to his fellow programmers for approval. If the other programmers come to an agreement, they implement the language and use it for their programs. Hence, the programming language spreads and becomes existent.

1. Procedural Programming

Problem is broken down into procedures, or blocks of code that perform one task each. All procedures taken together form the whole program. It is suitable only for small programs that have low level of complexity.

**Example** − For a calculator program that does addition, subtraction, multiplication, division, square root and comparison, each of these operations can be developed as separate procedures. In the main program each procedure would be invoked on the basis of user’s choice.

Object-oriented Programming

Here the solution revolves around entities or objects that are part of problem. The solution deals with how to store data related to the entities, how the entities behave and how they interact with each other to give a cohesive solution.

**Example** − If we have to develop a payroll management system, we will have entities like employees, salary structure, leave rules, etc. around which the solution must be built.

Functional Programming

Here the problem, or the desired solution, is broken down into functional units. Each unit performs its own task and is self-sufficient. These units are then stitched together to form the complete solution.

**Example** − A payroll processing can have functional units like employee data maintenance, basic salary calculation, gross salary calculation, leave processing, loan repayment processing, etc.

Logical Programming

Here the problem is broken down into logical units rather than functional units. **Example:** In a school management system, users have very defined roles like class teacher, subject teacher, lab assistant, coordinator, academic in-charge, etc. So the software can be divided into units depending on user roles. Each user can have different interface, permissions, etc.

Software developers may choose one or a combination of more than one of these methodologies to develop a software. Note that in each of the methodologies discussed, problem has to be broken down into smaller units. To do this, developers use any of the following two approaches −

* Top-down approach
* Bottom-up approach

Top-down or Modular Approach

The problem is broken down into smaller units, which may be further broken down into even smaller units. Each unit is called a **module**. Each module is a self-sufficient unit that has everything necessary to perform its task.

The following illustration shows an example of how you can follow modular approach to create different modules while developing a payroll processing program.



Bottom-up Approach

In bottom-up approach, system design starts with the lowest level of components, which are then interconnected to get higher level components. This process continues till a hierarchy of all system components is generated. However, in real-life scenario it is very difficult to know all lowest level components at the outset. So bottoms up approach is used only for very simple problems.

Let us look at the components of a calculator program.

1. **Problem analysis**: The phase of the program development life cycle in which the problem is carefully considered and the program specifications are developed.
2. **Programmer**: A person whose job it is to write, test, and maintain computer programs.
3. **Program design**: The phase of the program development life cycle in which the program specifications are expanded into a complete design of the new program.
4. **Flowchart**: A program design tool that graphically shows step-by-step the actions a computer program will take.
5. **Pseudocode**: A program design tool that uses English-like statements to outline the logic of a program.
6. **Unified Modeling Language (UML)**: A set of standard notations for creating business models; widely used for modeling object-oriented programs.
7. **Control structure**: A pattern for controlling the flow of logic in a computer program, module, or method.
8. **Sequence control structure**: A series of statements that follow one another.
9. **Selection control structure**: A series of statements in which the results of a decision determine the direction the program takes.
10. **Repetition control structure**: A series of statements in a loop that are repeated until a particular condition is met.
11. **Program coding**: The phase of the program development life cycle in which the program code is written using a programming language.
12. **Coding**: The process of writing the programming language statements to create a computer program.
13. **Source code**: A computer program before it is compiled.
14. **Program debugging and testing**: The phase of the program development life cycle that ensures a program is correct and works as intended.
15. 
16. **Identify the problem:** What problem does your program solve? If you can’t clearly state what your program does, you won’t know how to design it.
17. **Identify the user:** Who’s going to use your program?
18. **Determine the target computer:** Which computer do people need to run your program? Is it a Windows computer, a Macintosh, a mainframe, a computer running Linux, a handheld Palm or Pocket PC, or a supercomputer?
19. **Determine your programming skill:** Are you going to write the entire thing yourself or get help from others? If you’re going to get others to help you, which parts of the program are they going to write?

7

**Structured programming** is a **programming** paradigm aimed at improving the clarity, quality, and development time of a computer **program** by making extensive use of the **structured** control flow constructs of selection (if/then/else) and repetition (while and for), block structures, and subroutines.

 8

* Application programs are easier to read and understand.
* Application programs are less likely to contain logic errors.
* Errors are more easily found.
* Higher productivity during application program development.
* Improved application program design.
* Application programs are more easily maintained.

9

After the processing requirements are known, the actual logic of the solution can be determined. In order to do this, it is necessary to know the basic logic patterns that the computer is able to execute. The power of the computer comes in large part through the programmer's ability to specify the sequence in which statements in a program are to be executed. However, the computer can execute only four basic logic patterns: the simple sequence, the selection pattern, the loop, and the branch. Pro-gramming languages may have more complicated statements, but they all are based on various combinations of these four patterns.

**SIMPLE SEQUENCE**

In a **simple sequence** the computer executes one statement after another in the order in which they are listed in the program. It is the easiest pattern to understand. The Figure below demonstrates the simple sequence pattern as it relates to the payroll example.

**SELECTION**

The **selection** pattern requires that the computer make a choice. The choice it makes, however, is based not on personal preference but on pure logic. Each selection is made on the basis of the results of a comparison. The computer can determine if a given value is greater than, equal to, or less than another value; these are the only comparisons the computer is capable of making. Complex com-parisons are made by combining two or more simple comparisons. This process of requiring the computer to make a selection or choice is often referred to as conditional programming logic. The Figure below illustrates the selection pattern by demonstrating how the logic of the payroll example would consider overtime pay.

10

The timetable will be in such a way that one lecturer has at most 2 classes in a day and the students will have at least one class of a particular course in a week. In a day. there will be at most 4 classes exclusive of night classes…No lecturer can have two classes concurrently in a day

11 Ssrjdrcc start

 Input 23 (radius)

 Calculate

 Volume =

 (4/3)\*3.14\*(23)^3

 Display 49,884=are

 End

12

BASE= pi \* (r)^2

Volume = 1/3 \* b \* h

Input volume = 200cm3

Imput height = 18

b = (3\* volume)/ h

= (3\*200)/18 = 33.3

(r)^2 = (base)/pi = 33.3/3.14 = 10.6

R= mathsqrt(10.6)= 3.25

Print “The radius is 3.25”

13

Structured Programming is designed which focuses on **process**/ logical structure and then data required for that process.

Object Oriented Programming is designed which focuses on **data**.

Structured programming follows**top-down approach**.

Object oriented programming follows **bottom-up approach**.

Structured Programming is also known as **Modular Programming** and a subset of **procedural programming language**.

Object Oriented Programming supports **inheritance, encapsulation, abstraction**, **polymorphism**, etc.

In Structured Programming, Programs are divided into small self contained **functions**.

In Object Oriented Programming, Programs are divided into small entities called **objects**.

Structured Programming is **less** secure as there is no way of **data hiding**.

Object Oriented Programming is more secure as having data hiding feature.

Structured Programming can solve **moderately** complex programs.

Object Oriented Programming can solve any **complex**programs.

Structured Programming provides **less** **reusability**, more function dependency.

Object Oriented Programming provides more reusability, less function **dependency**.

Less abstraction and less flexibility.

More abstraction and more **flexibility**.

14

#include <stdio.h>

This is a preprocessor command that includes standard input output header file(stdio.h) from the C library before compiling a C program

int main()

This is the main function from where execution of any C program begins.

const float rad = 7.5; float cir; cir = 2 \* pi \* rad;

means the radius is 7.5 which is a constant while the circumference is 2 \*pi\* rad

printf(“the value of circumference of a circle is %d”, ci

to print the value to the circumference

return 0; }

to end the program

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |