CS1CRT02 (B.Sc. Computer Science)
CA1CRT02 (BCA)
CA1CRT02 (BSc CA Triple Main)
IT1CRT03 (BSc.IT)

Methodology of Programming and C Language

Semester I

MG University, Kottayam

Syllabus

Module 1

Introduction to programming, Classification of computer languages, Language translators (Assembler, Compiler, Interpreter), Linker, Characteristics of a good programming language, Factors for selecting a language, Subprogram, Purpose of program planning, Algorithm, Flowchart, Pseudo code, Control structures (sequence, selection, Iteration), Testing and debugging

Module 2

C Character Set, Delimiters, Types of Tokens, C Keywords, Identifiers, Constants, Variables, Rules for defining variables, Data types, C data types, Declaring and initialization of variables, Type modifiers, Type conversion, Operators and Expressions- Properties of operators, Priority of operators, Comma and conditional operator, Arithmetic operators, Relational operators, Assignment operators and expressions, Logical Operators, Bitwise operators

Module 3

Input and Output in C – Formatted functions, unformatted functions, commonly used library functions, Decision Statements If, if-else, nested if-else, if-else-if ladder, break, continue, go-to, switch, nested switch, switch case and nested if. Loop control- for loops, nested for loops, while loops, do while loop.

Module 4

Array, initialization, array terminology, characteristics of an array, one dimensional array and operations, two dimensional arrays and operations. Strings and standard functions, Pointers, Features of Pointer, Pointer and address, Pointer declaration, void wild constant pointers, Arithmetic operations with pointers, pointer and arrays, pointers and two dimensional arrays.

Module 5

Basics of a function, function definition, return statement, Types of functions, call by value and reference. Recursion -Types of recursion, Rules for recursive function, direct and indirect recursion, recursion vs iterations, Advantages and disadvantages of recursion. Storage class, Structure and union, Features of structures, Declaration and initialization of structures, array of structures, Pointer to structure, structure and functions, typedef, bit fields , enumerated data types, Union, Dynamic memory allocation, memory models, memory allocation functions.

Book of Study:

1. Ashok Kamthane - Programming in C, Third Edition, Pearson Education
2. P K Sinha & Priti Sinha - Computer Fundamentals, Fourth Edition, BPB Publications.

Reference:

1. E. Balaguruswamy -Programming in ANSI C ,Seventh Edition, McGraw Hill Education
2. Byron Gotfried - Programming with C, Second Edition, Schaums Outline series. McGraw Hill

Module 1

INTRODUCTION TO PROGRAMMING

Computer is an electronic device that accepts data, process those data, store the data and results the output. Data are accepted by keyboard or any other input devices. Data are processed by means of central processing unit or simply CPU. Some kind of memories are used to store the data and results. And these results as delivered to the output or visualized by monitors.

Computers can perform its operation by the means of programs. Programs are set or group of instructions that are loaded into memory. Instructions are set of commands that what a computer to do or perform. Art of writing or creating programs are known as programming.

Computer can communicate with the users by means of languages. Computers can only understand its own language known as machine language. Machine language consist of 1s and 0s. Or simply known as binary. Machine language is the lowest level language because computer knows only the patterns or combinations of 1s and 0s.

In order to write or create programs in binary is complicated and confused. Because change in 0 as 1 or 1 as 0 will create serious errors on the result. So the experts designed languages to communicate with the computers known as high level languages. This will avoid complication and confusion since high level languages are written in English like language. That’s why C is called high level language since it has a structure like English. English words are used in C Programming. It is very easy to learn C programming.

CLASSIFICATION OF COMPUTER LANGUAGES

Computer Languages are classified into two. They are

1. Low Level Language
2. High Level Language.

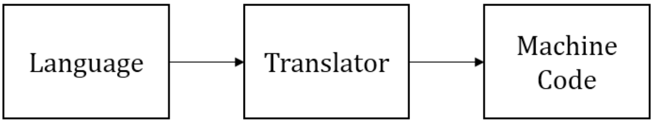
Low Level Language

Programs or set of instructions written in the form of binary (1s and 0s) are known as Low Level Language or machine language. This is the only language that can be understood or interpreted by a computer.

High Level Language

Programs or set of instructions that are written in structured languages such as English are known as High Level Languages. High level languages are not directly interpreted or understood by a computer. Examples: C, FORTRAN, COBOL, BASIC etc.

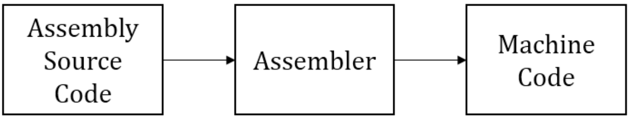
LANGUAGE TRANSLATORS



There are some special programs are required to make the computer understand what a computer to do. Such language converting programs are known as Language Translator. The word translator means decoder or interpreter. The process is known as Language translation. Some special programs designed and create to do the translation job for a computer are as follows.

1. Assembler
2. Compiler
3. Interpreter

Assembler



Assembler is a special program to convert assembly language into machine language. Assembly language means that language written in mnemonics. It is also seemed to be written in English-like words, but in the form of abbreviations.

Example, a program written in mnemonics or assembly code is shown.

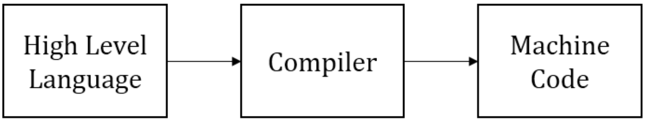
```
MVI A, 6AH
MOV B, A
ADD B
STA 5000H
```

The program shown above is an assembly program for Intel 8085 Microprocessor systems. MVI means “Move Immediately”. That means 6A$_H$ is a hexadecimal number that should copy or move immediately to register A in the microprocessor. MOV means “to move” a data from source register to destination register. Here the content 6A$_H$ of the register A will be copied to Register B. ADD B means content of Register B will be “added” to the Register A. STA means “store the content of the register A” to the memory location 5000$_H$. Register is a memory location inside the microprocessor for temporary storage of data.

So the assembly language consist of mnemonics or assembly codes such as MVI, MOV, ADD and STA are English-like words but abbreviations in assembly programming. So an assembly programmer can easily identify the logic and nature of the program to get the output.

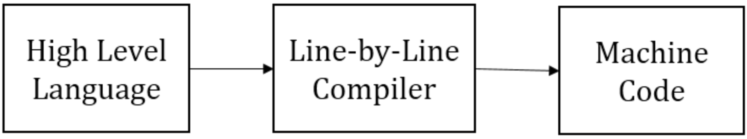
Those programs are interpreted or understood by an assembler. It will convert assembly language to its own machine language.

Compiler



Another special program are designed and developed to convert high level language to machine languages are known as compiler. The word compile means to “convert”. High level languages such as C, FORTRAN, COBOL, BASIC etc. are to be compiled to machine language.

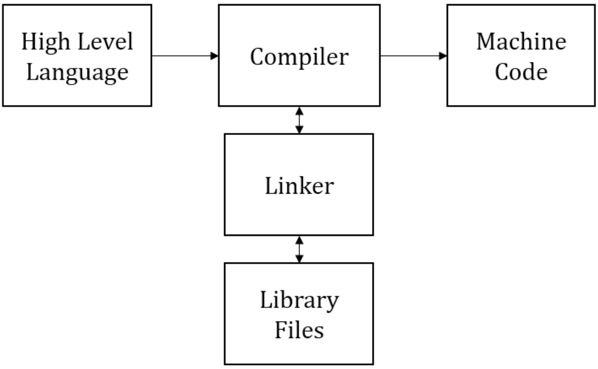
Interpreter



Interpreter is also a program that converts high level language to machine language line by line. On other hand, it could be said that the interpreter is “compiler online”. Because as soon as a line of statement is completed in the interpreter platform, it will check the syntax and semantics of the statement at the end of the line and compile it. If there exist any mistake in the statement it will report error to the programmer.

For Example, Microsoft Visual Basic IDE, while coding, report error, when the syntax is wrong. So the programmer could be able to correct the syntax errors by the end of the line of the statement itself.

Linker



We know that compilers are capable for converting high level languages to machine languages. Moreover additional programs are also required to compile more complex user programs. In order to compile complex programs created by the user it may need to link some library files in the computer to get the desired output. Some additional programs are required to link library files. Such programs are known as linkers. Linkers are nothing but the programs run along with the compiler for linking library files so as to get the final machine code.

CHARACTERISTICS OF A GOOD PROGRAMMING LANGUAGE

There are some popular and powerful high-level programming languages. There are several characteristics believed to be important for making it good.

Naturalness

A good language should be natural for the application area for which it is designed. That is, it should provide appropriate operators, data structures, control structures and a natural syntax to facilitate programmers to code their problems easily and efficiently. FORTRAN and COBOL are good examples of languages possessing high degree of naturalness in scientific and business application areas, respectively.

Abstraction

Abstraction means ability to define and then use complicated structures or operations in ways that allow many of the details to be ignored. The degree of abstraction allowed by a language directly affects its ease of programming.

Efficiency

Programs written in a good language are translated into machine code efficiently, are executed and require relatively less space in memory. That is, a good programming language is supported with a good language translator (a compiler or an interpreter) that gives due consideration to space and time efficiency.

Structured Programming Support

A good language should have necessary features to allow programmers to write their programs based on the concepts of structured programming .This property greatly affects the ease with which a program may be written, tested and maintained. Moreover, it forces a programmer to look at a problem in a logical way so that fewer errors are created while writing a program for the problem.

Compactness

In a good language, programmers should be able to express the intended operations concisely without losing readability. Programmers generally do not like a verbose language because they need to write too much.

Locality

A good language should be such that while writing a program, a programmer need not jump around the visually as the text of a program is prepared. This allows the programmer to concentrate almost solely on the part of the program around the statement currently being worked with. COBOL and to some extent C and Pascal lack locality because data definitions are separated from processing statements, perhaps by many pages of code, or have to appear before any processing statement in the function/procedure.

Extensibility

A good language should also allow extensions through a simply, natural and neat mechanism. Almost all languages provide subprogram definition mechanisms for the purpose, but some languages are weak in this aspect.

Suitability to its Environment

Depending upon the type of application for which a programming language has been designed, the language must also be made suitable to its environment. For Example, a language designed for a real-time applications must be interactive in nature. On the other hand, languages used for data-processing jobs like payroll, stores accounting etc. may be designed to operative in batch mode.

FACTORS FOR SELECTING A LANGUAGE

The factors we select or choose a language is described as follows.

Popularity

This is a very important one. You are more likely to find people to collaborate with if you use a popular language. You are also more likely to find reference material and other help.

Language-domain match.

Select a language that matches your problem domain. You can do this by looking at what other people in your field are using. Look at the code that solves problems that you are working on.

Availability of libraries

If there's a library that solves your problem well, then it will be helpful to choose the language for your need.

Efficiency

Languages compilers should be efficient. Look at the efficiency of compilers or interpreters for your language. Be aware that interpreted code will run properly.

Expressiveness

The number of lines of code you create per hour is not a strong function of language, so favor languages that are expressive or powerful.

Tool Support

Popularity usually buys tool support. Some languages are easier to write tools. If you are a tool-oriented user, choose a language with good tool support.

SUBPROGRAM

A subprogram is a program called by another program to perform a particular task or function for the program. When a task needs to be performed multiple times, you can make it into a separate section. The complete program is thus made up of multiple smaller, independent subprograms that work together with the main program.

There are two different general types of subprograms available:

- **External** — they exist as stand-alone programs that are listed in the program menu, and can be executed just like a regular program.
- **Internal** — they are contained inside the program itself, so that they can be called by the program whenever needed.

PURPOSE OF PROGRAM PLANNING

Before writing a program, one should know what is going to be programmed. Problems are solved using computer programs. A Programmer should analyze the problem very well from the beginning to end. He should have an idea and design the problem before it implement. Also imagination of the result should be predicted. Else he would get unpredicted and undesirable output.

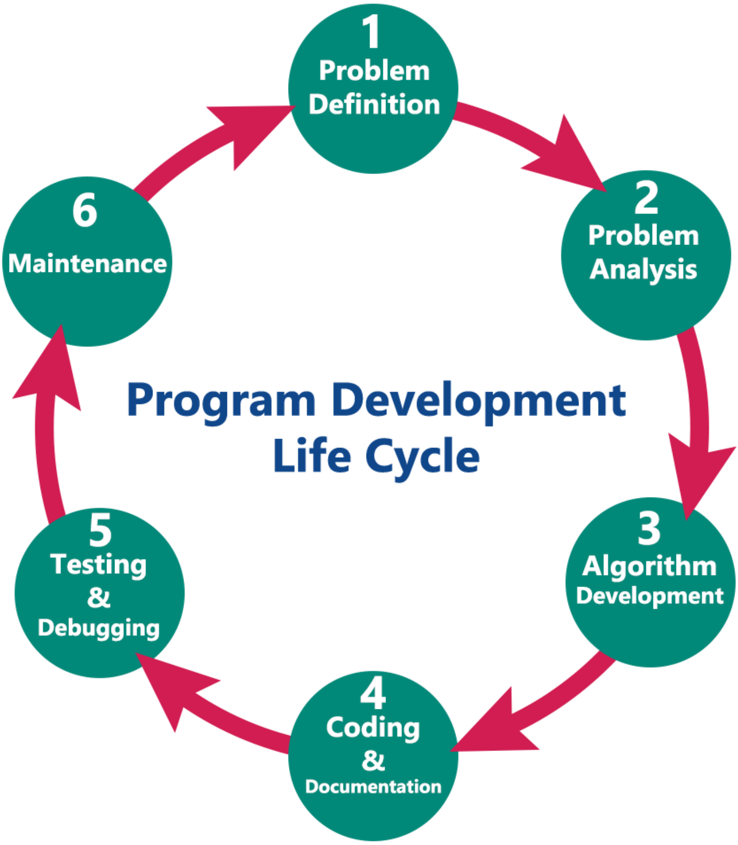
Program Life Cycle

When we want to develop a program using any programming language, we follow a sequence of steps. These steps are called phases in program development. The program development life cycle is a set of steps or phases that are used to develop a program in any programming language.

Generally, program development life cycle contains 6 phases, they are as follows:

- Problem Definition
- Problem Analysis
- Algorithm Development
- Coding & Documentation

- Testing & Debugging
- Maintenance



Problem Definition

In this phase, we define the problem statement and we decide the boundaries of the problem. In this phase we need to understand the problem statement, what is our requirement, what should be the output of the problem solution. These are defined in this first phase of the program development life cycle.

Problem Analysis

In phase 2, we determine the requirements like variables, functions, etc. to solve the problem. That means we gather the required resources to solve the problem defined in the problem definition phase. We also determine the bounds of the solution.

Algorithm Development

During this phase, we develop a step by step procedure to solve the problem using the specification given in the previous phase. This phase is very important for program development. That means we write the solution in step by step statements.

Coding & Documentation

This phase uses a programming language to write or implement actual programming instructions for the steps defined in the previous phase. In this phase, we construct actual program. That means we write the program to solve the given problem using programming languages like C, C++, Java etc.

Testing & Debugging

During this phase, we check whether the code written in previous step is solving the specified problem or not. That means we test the program whether it is solving the problem for various input data values or not. We also test that whether it is providing the desired output or not.

Maintenance

During this phase, the program is actively used by the users. If any enhancements found in this phase, all the phases are to be repeated again to make the enhancements. That means in this phase, the solution (program) is used by the end user. If the user encounters any problem or wants any enhancement, then we need to repeat all the phases from the starting, so that the encountered problem is solved or enhancement is added.

There are some basic tools for programming before the programmer start to implement or coding the programs. The basic tools are as follows.

Algorithm

Any problem can be ever followed by sequence of steps or procedures. Those sequence of steps or procedures are termed as algorithm.

Example: Recipe for making a cup of Tea.

Here the problem is to make a cup of tea to drink. Analyze the problem very well and check for the availability of requirements. For making tea; water, milk, tea powder, sugar sauce pan, filter etc. are required.

Algorithm for making a cup of Tea

1. START stove
2. Take ½ cup WATER
3. Take ½ cup MILK
4. Pour both into SAUCE PAN.
5. BOIL it
6. Put ½ tea-spoon of TEA POWDER into boiled material.
7. Mix 1 tea-spoon of SUGAR on it.
8. STIR it well
9. FILTER tea to a cup
10. STOP stove

The example shows above is, just to illustrate and to understand what the algorithm is.

For anything we want to make, we should follow some steps. These steps are known as algorithm.

Similarly, problems are solved by writing sequence of statements or steps in chronological order. The manner in which writing sequence of statements or steps in chronological order to solve a problem is known as an algorithm.

Example: Sum of two numbers

1. START Program
2. READ First Number
3. READ Second Number
4. ADD First Number with Second Number
5. ASSIGN to SUM
6. DISPLAY SUM
7. END Program

The problem above is to monitor or display the sum of two numbers are illustrated in the form of an algorithm. We can use comfortable languages or its abbreviations to implement an algorithm as follows.

```
1. START
2. READ NUM1
3. READ NUM2
4. SUM←NUM1+NUM2
5. PRINT SUM
6. END
```

Algorithm should be

1. Simple
2. Short
3. Understandable
4. Keep proper sequence
5. maintain ethical logic
6. Result oriented.

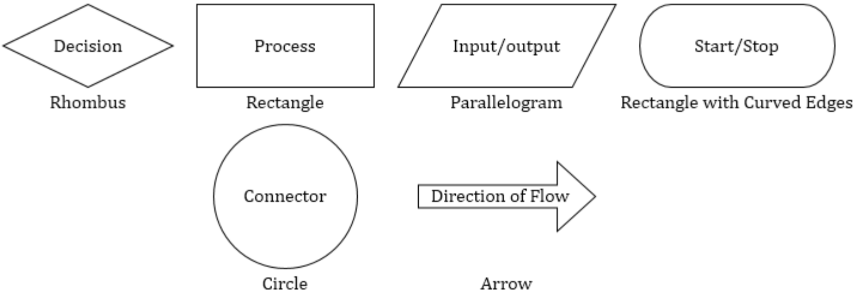
Another Example: Largest of Two Numbers.

```
1. Start
2. Read Num1
3. Read Num2
4. Is (Num1>Num2)?
5. Yes: Print Num1 is Large, GOTO Step 7
6. Else: Print Num2 is Large
7. End
```

It is possible to switch from a step to another without violating the logic of the program. Algorithms are written in English like language or can be written in a language that we feel like.

Flowchart

The pictorial or graphical representation of flow of a program is known as flowchart. If the algorithms or programs are displayed in the form of a picture then it will be more noticeable and recognizable. Only thing what we need to know some specific shapes to represent each processes or actions. The fundamental shapes and descriptions using in flowcharts are as follows.



Rhombus: For decision making and branching the flow of execution.

Rectangle: For processing and assigning variables.

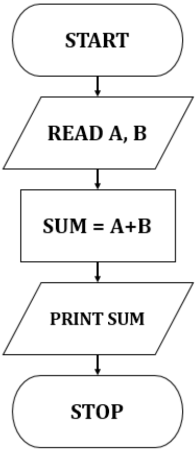
Parallelogram: Accessing inputs and printing outputs.

Rectangle with Curved Edges: Start/Begin or Stop/End of the program execution.

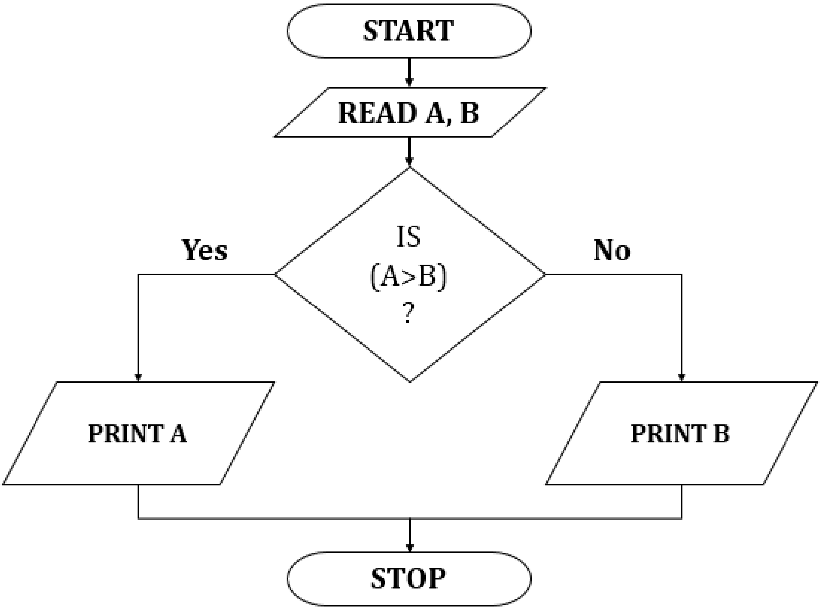
Circle: Connectors to continue the flow of execution.

Arrow: Represents the direction of Flow of execution

Example: Flowchart for Sum of two numbers



Example: Flowchart to find Largest of Two Numbers.



Pseudo code

The term pseudo means false. The false codes used to represent the flow of execution of a program is known as pseudo codes. Usually programmers use pseudo codes for rough work on their problem solving. Most probably, they use pen and papers for their rough programming work. They should not bother syntax or semantics of any high level programming languages.

Pseudo code is an informal high-level description of the operating principle of a computer program or any other algorithm. It uses the structural conventions of a normal programming language, but is intended for human reading rather than machine reading.

Example: Pseudo Code to find Largest of Two Numbers.

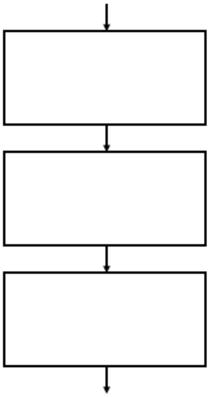
```
Rd N1, N2
N1>N2?
Y: Prn N1
N: Prn N2
```

CONTROL STRUCTURES

A control structure is a block of programming that analyzes variables and chooses a direction in which to go based on given parameters. Any computer program can be written using the basic control structures. They can be combined in any way necessary to deal with a given problem. The control structures are:

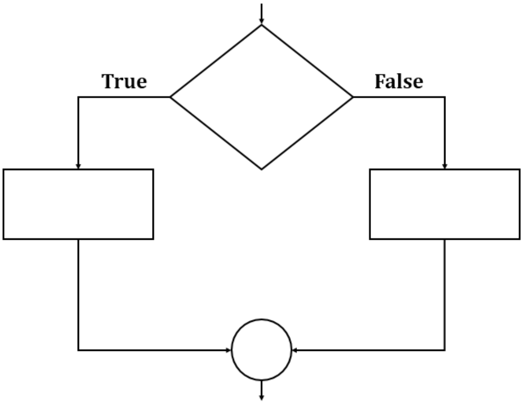
1. Sequence
2. Selection
3. Iteration

Sequence



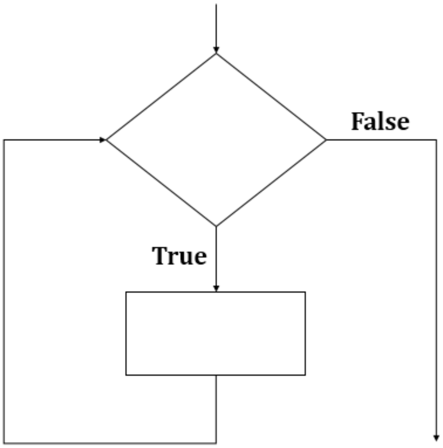
The sequence structure simply executes a sequence of statements in the order in which they occur.

Selection



The selection structure tests a condition, then executes one sequence of statements instead of another, depending on whether the condition is true or false. A condition is any variable or expression that returns a Boolean value (TRUE or FALSE).

Iteration



The iteration structure executes a sequence of statements repeatedly as long as a condition holds true.

TESTING AND DEBUGGING

Testing is meant to find defects or errors in the code, or from a different angle, to prove to a suitable level that the program does what it is supposed to do. It can be manual or automated. It has many different types, like unit testing, integration testing, system testing user acceptance testing, stress testing, load testing, and soak testing etc.

Bug means error. Debugging is the process of finding and removing a specific bugs or errors from the program. It is always a manual, on-off process, as all bugs are different.