Programming in C

I semester, BCA

UNIT I : Introduction to Programming Concept

Compiled by :
Ankit Bhattarai

Syllabus

Unit	Contents	Hours	Remarks
1.	Introduction of Programming Concept	4	
2.	Introduction to C	6	
3.	Control Structure	7	
4.	Function	6	
5.	Array, Pointer and String	7	
6.	Structure and Union	7	
7.	Input Output and File handling	5	
8.	Introduction to Graphics	3	

Practical Works

Credit hours : 3

Practical Works

- Designing algorithm and draw flow chart for sequence, decision making and repetition
- concept of general programming: Creating different shapes using graphics function, Use modes and initialization techniques
- Developing basic structure of C programmed.
- Declaring and assigning variables and constants.
- Applying input and output build in function
- Using arithmetic operators
- Giving demo of type conversion
- Use of if, if-else and switch statement
- Use while, do-while, for and nested loop concept.
- Use of Jumping statement
- Prototype, call and define function
- Pass the different parameter methods
- Use call by reference methods using function.
- Create a recursion function
- Array declares, define, initialize.
- Creating a single or multi-dimensional array.
- Using pointer and demo for arithmetic function.
- Using different string function in program.
- Example of passing Array and String in function
- Creating structure data types with application of loop.
- Creating union data types.
- Example of Structure and Function
- Creating file handling application for open, read, write and appends.
- Handling the random access files.
- Applying the text formatting function.

Resources

Resources

1. Some chapters in pdf form.
2. Question Collection & Assignments (50-80 Questions)

Books

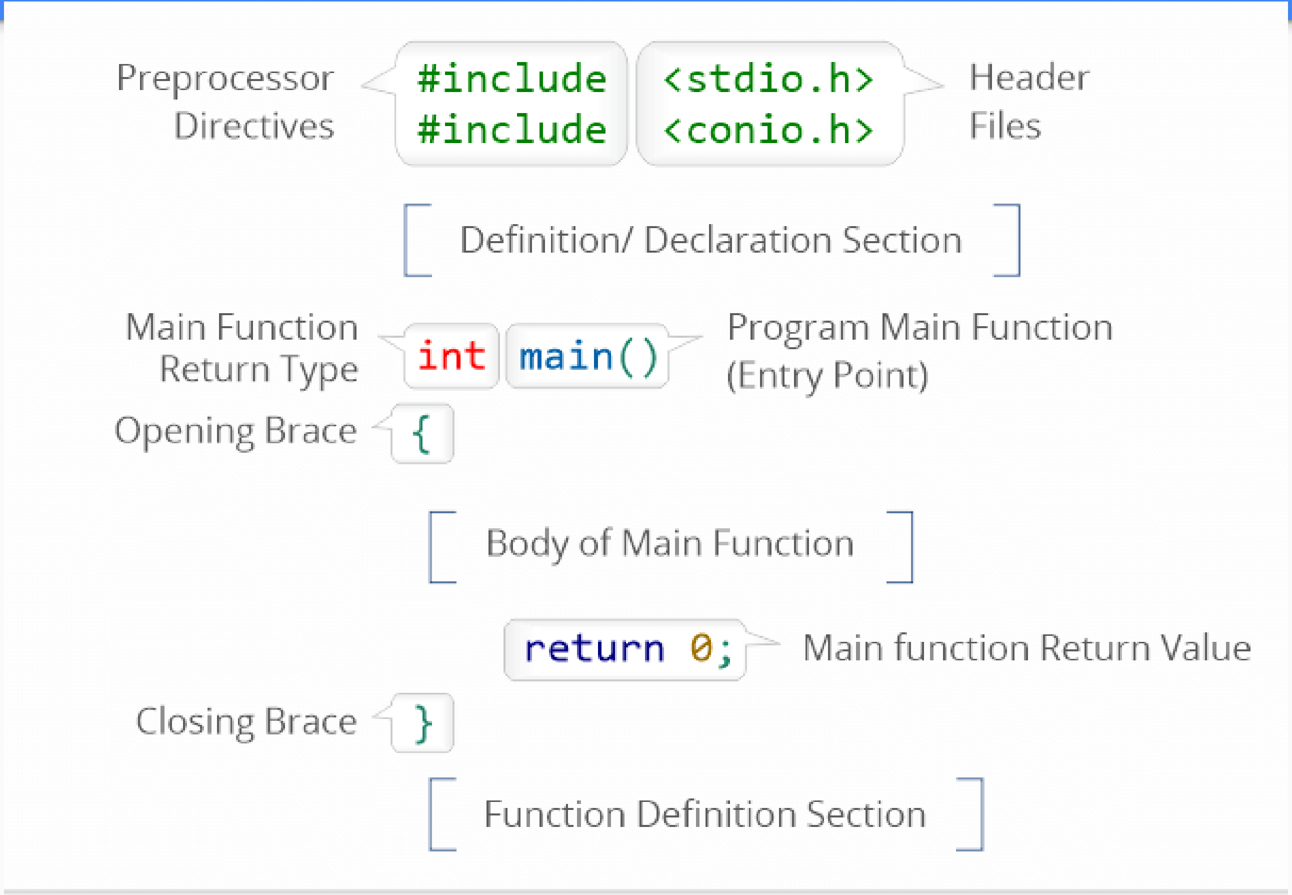
Text Books:

- Kanetkar, Y. P. (2008). Let us C8th Ed, New Delhi, BPB Publication (Unit 1 -8)
- Balagurusamy, E. (2007). Programming in ANSI C. New Delhi, India: Tata McGraw-Hill.

References materials:

- Raman, R. (1984). Computer programming in C, New Delhi, PHI. India
- Carlo Ghezi, Mehdi Jazayeri, “Programming Language Concepts”, John Wiley and Sons
- B.S. Gottfried(2001), Schaum’s Outline Series for Programming with , Second Edition, Tata McGraw Hill Publishing Company, New Delhi.

Format for C program


Preprocessor Directives
#include
#include
<stdio.h>
<conio.h>
Header Files
Definition/ Declaration Section
Main Function Return Type
int
main()
Program Main Function (Entry Point)
Opening Brace
{
Body of Main Function
return 0;
Main function Return Value
Closing Brace
}
Function Definition Section

Unit 1

(4 hrs.)

Introduction of Programming Concept

- Introduction of Programming Language: Language Translator, Assembler, Compiler, Interpreter, Syntax and Semantics
- Programming Design Tools: Algorithm, Flow chart, Pseudo codes
- Features of good program
- Different Programming Techniques: Procedural Programming, Modular Programming, Object Oriented Programming

Introduction to Programming Language

- A programming language is a standardized communication technique for describing instructions for a computer.
- Each programming language has a set of syntactic and semantic rules used to define computer programs.
- A programming language enables a programmer to precisely specify what data computer is act upon, how these data are to be stored/transmitted and what actions are to be taken under circumstances.

Introduction to Programming Language

Programming language are classified mainly in two categories:

- Low level programming language
- High level programming language

i) Low level programming language

Low level language are specific to hardware. Before creating a program, it is required to have through knowledge of that hardware. low level programming language are divided into two types.

- Machine language
- Assembly language

Introduction to Programming Language : Low Level Language

Machine language: Machine language are lowest- level programming language. A computer understands program written only in the machine language. It is directly executable by computer without the need for translation by a compiler or an assembler.

- Machine code consist entirely of the 0’s and 1’s of the binary system. Early computers were programmed using machine language. Programs written in machine language are faster and efficient.
- Writing program in machine language is very tedious, time consuming, difficult to find bugs in longer programs.

Introduction to Programming Language: Low Level Language

Assembly language

- In assembly language, instead of using numeric opcodes (i.e. pattern of 0 &1), mnemonics are used e.g. ADD,SUB etc.
- Program written in assembly language must be converted into machine language which could be done by assembler.
- Assembly language program written for one type of CPU won’t run on another. So that assembly language is time consuming and machine dependent.

Introduction to Programming Language

Advantages of low level languages

- Programs developed using low level languages are fast and memory efficient.
- There is no need of any compiler or interpreters to translate the source to machine code. Thus, cuts the compilation and interpretation time.
- It can directly communicate with hardware devices.

Disadvantages of low level languages

- Programs developed using low level languages are machine dependent and are not portable.
- It is difficult to develop, debug and maintain.
- Programmer must have additional knowledge of the computer architecture of particular machine, for programming in low level language

Introduction to Programming Language

ii) High Level programming language

- The syntax of high level is closer to human language. High level language were developed to make programming easier. Most of the high level language are English like languages. They use familiar English words, Special symbols (! & etc.) in their syntax. Therefore high level language are easier to read, write, understand and maintain.
- Each high level language has their own set of grammar and rules to represent set of instructions. E.g. C ,C++,Java, FORTAN etc.
- Program written in high level language also need to translated into machine language. This can be done either by compiler or interpreter.

Introduction to Programming Language

Advantages of High level language

- High level languages are programmer friendly.
- They are easy to write, debug and maintain.
- They are portable, which means same code can run on different hardware.
- It is machine independent language and easy to learn.

Disadvantages of High level language

- It takes additional translation times to translate the source to machine code.
- Execution of high level programs are comparatively slower than low level programs.
- Compared to low level programs, they are generally less memory efficient.
- Cannot communicate directly with the hardware.

Differences between high level language and low level language.

Introduction to Programming Language : High Level Vs Low Level Language

High Level Language	Low Level Language
It is programmer friendly language	It is machine friendly language
Compiler or interpreter is required for translation.	Assembler is required for translation.
They execute slower.	They execute faster.
For writing program hardware knowledge is not required.	For writing program hardware knowledge is required
They are easier to learn and understand by human.	It is difficult to learn and understand by human.
It can run on any platform	It is machine dependent
It is simple to debug and maintain.	It is difficult to debug and maintain as compared to high level language.
Example: C, C++, Java etc.	Example: Assembly language

Language Translator

- A programmer write a program in high level language that is to be translated into machine language equivalent code. This task is achieved by using language translator.

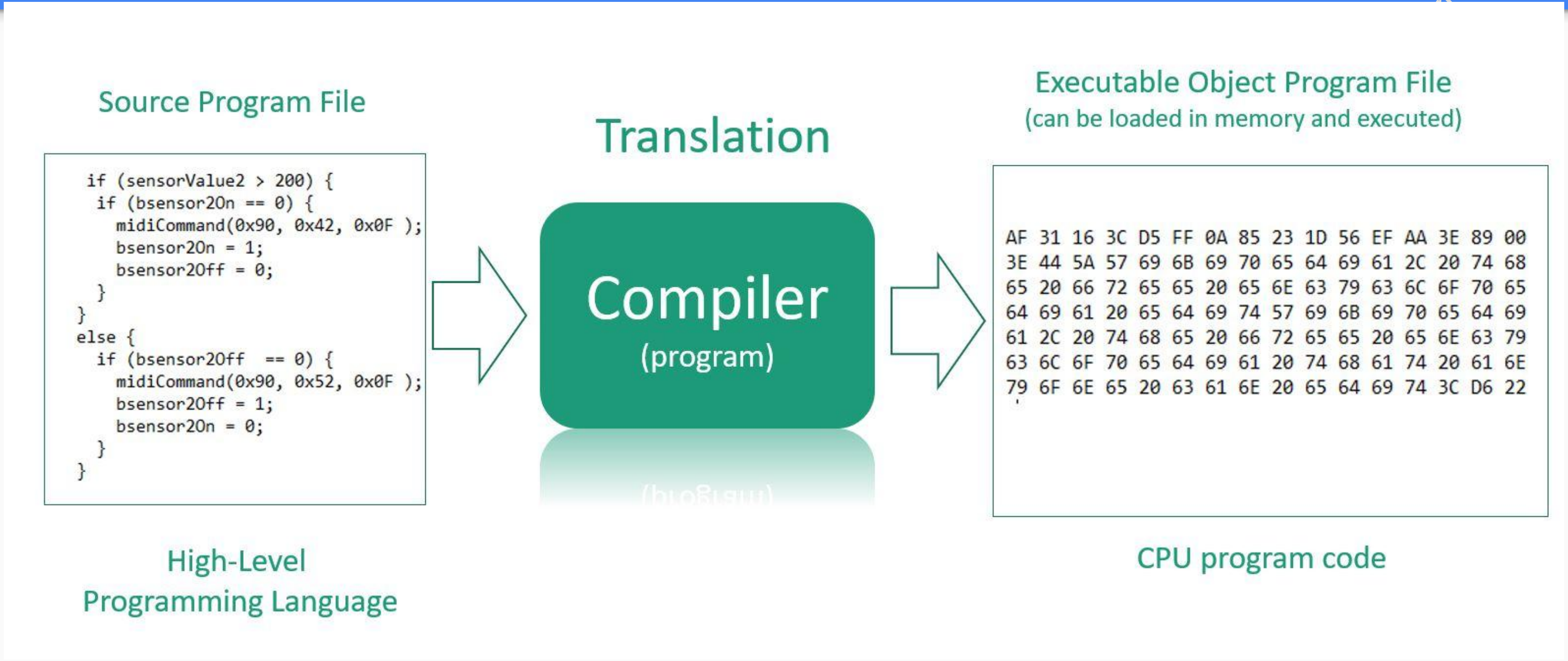
The common language translator are:

- Compiler
- Interpreter
- Assembler

Language Translator: Compiler Vs Interpreter

Compiler	Interpreter
A compiler translates the entire source code to object code and then only object code is executed.	An interpreter translates one statement at a time, executes it and continues for another statement
Compiler is faster than interpreter	Interpreter is slower than compiler
It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.	It continuously translates the program until the first error is met, in which case it stops. Hence debugging is easy.
A compiler is a complex program	Interpreter is less complex program than compiler.
As compared to an interpreter developing a compiler is difficult.	As compared to a compiler developing interpreter is easier.
Programming language like C,C++,FORTAN use compiler	Python use interpreter.

Language Translator: Compiler



```
 if (sensorValue2 > 200) {
  if (bsensor2On == 0) {
    midiCommand(0x90, 0x42, 0x0F );
    bsensor2On = 1;
    bsensor2Off = 0;
  }
 }
else {
  if (bsensor2Off  == 0) {
    midiCommand(0x90, 0x52, 0x0F );
    bsensor2Off = 1;
    bsensor2On = 0;
  }
}
```

```
AF 31 16 3C D5 FF 0A 85 23 1D 56 EF AA 3E 89 00
3E 44 5A 57 69 6B 69 70 65 64 69 61 2C 20 74 68
65 20 66 72 65 65 20 65 6E 63 79 63 6C 6F 70 65
64 69 61 20 65 64 69 74 57 69 6B 69 70 65 64 69
61 2C 20 74 68 65 20 66 72 65 65 20 65 6E 63 79
63 6C 6F 70 65 64 69 61 20 74 68 61 74 20 61 6E
79 6F 6E 65 20 63 61 6E 20 65 64 69 74 3C D6 22
```

Language Translator: Assembler

- An assembler is a program (software) which translates the program written in assembly language to machine language.
- It takes the basic commands and operations from assembly code and converts them into binary code that can be recognized by a specific type of processor.
- Assemblers are similar to compilers in that they produce executable code. However, assemblers are more simplistic since they only convert low-level code (assembly language) to machine code.
- Since each assembly language is designed for a specific processor, assembling a program is performed using a simple one-to-one mapping from assembly code to machine code in that they produce executable code.

Syntax and Semantics

Syntax

Definition: Syntax refers to the rules that define the structure or format of a programming language. It dictates how symbols, keywords, and expressions should be written to form valid statements.

Analogy: Syntax is like grammar in a natural language. Just as sentences must follow grammatical rules to be understood, code must adhere to a language's syntax to run.

Key Features:

Keywords: Reserved words that have special meanings (e.g., if, else, for in Python).

Operators: Symbols used for operations (e.g., +, -, *).

Punctuation: Characters like ;, {}, or : that organize code.

Structure: How statements and blocks are defined, like indentation in Python or braces {} in C.

Syntax and Semantics

Semantics

Definition: Semantics defines the meaning or behavior of the code. It's about what the program does when executed, rather than how it is written.

Analogy: Semantics is like the meaning of a sentence. Even if a sentence follows grammar rules (syntax), it must make sense (semantics).

Key Aspects:

Variable meaning: Assigning and interpreting values correctly.

Control flow: How loops, conditions, and functions execute.

Error handling: Understanding runtime errors when semantic rules are violated.

Syntax and Semantics

Syntax is correct but semantics is wrong:

```
x = "10" + 5  # Syntax is fine, but it raises a TypeError as you can't add a string and an integer.
```

Semantically correct code:

```
x = 10 + 5  # This is both syntactically and semantically correct.
```

Programming Design Tools: Algorithm, Flow chart, Pseudo codes

Programming Design Tools: Algorithm, Flow chart, Pseudo codes

An **Algorithm** is defined as a sequence of steps or procedures necessary to solve problem. To be an algorithm set of step must be unambiguous. Each step tells what task to be performed.

An Excellent Algorithm should have the following properties.

i. **Finiteness:** Each algorithm must have finite number of steps to solve a problem.

ii. **Definiteness:** The action of each step should be defined clearly without any ambiguity.

iii. **Inputs:** Inputs of the algorithm should be define precisely, which can be given initially or while the algorithm runs.

iv. **Output:** An algorithm can give one or more result. After an algorithm terminates, algorithm must give desired result.

v. **Effectiveness:** An algorithm should be more effective among different ways of solving the problem.

Programming Design Tools: Algorithm, Flow chart, Pseudo codes

Algorithm to find the sum of any two numbers

```
Step 1: Start
Step 2: Declare variables a ,b and sum
Step 3: Read value of a and b
Step 4: calculate sum= a + b
Step 5: Display sum
Step 6: Stop
```

Algorithm for calculating the simple interest using the formula i=p*t*r/100

```
Step 1: Start
Step 2: Declare variables p, t, r and i
Step 3: Read value of p, t, r
Step 4: i=p*t*r/100
Step 5: Display i
Step 6: Stop
```

Programming Design Tools: Algorithm, Flow chart, Pseudo codes

An algorithm to convert Fahrenheit when temp in Centigrade is given.

```
Step 1: Start
Step 2: Declare variables f and c
Step 3: Read value of c
Step 4: calculate f=1.8*c+32
Step 5: Display f
Step 6: Stop
```

An algorithm to calculate the area of rectangle

```
Step 1: Start
Step 2: Declare variables l, b and area
Step 3: Read value of l and b
Step 4: calculate area=l*b
Step 5: Display
```

Programming Design Tools: Algorithm, Flow chart, Pseudo codes

Advantages of Algorithms:

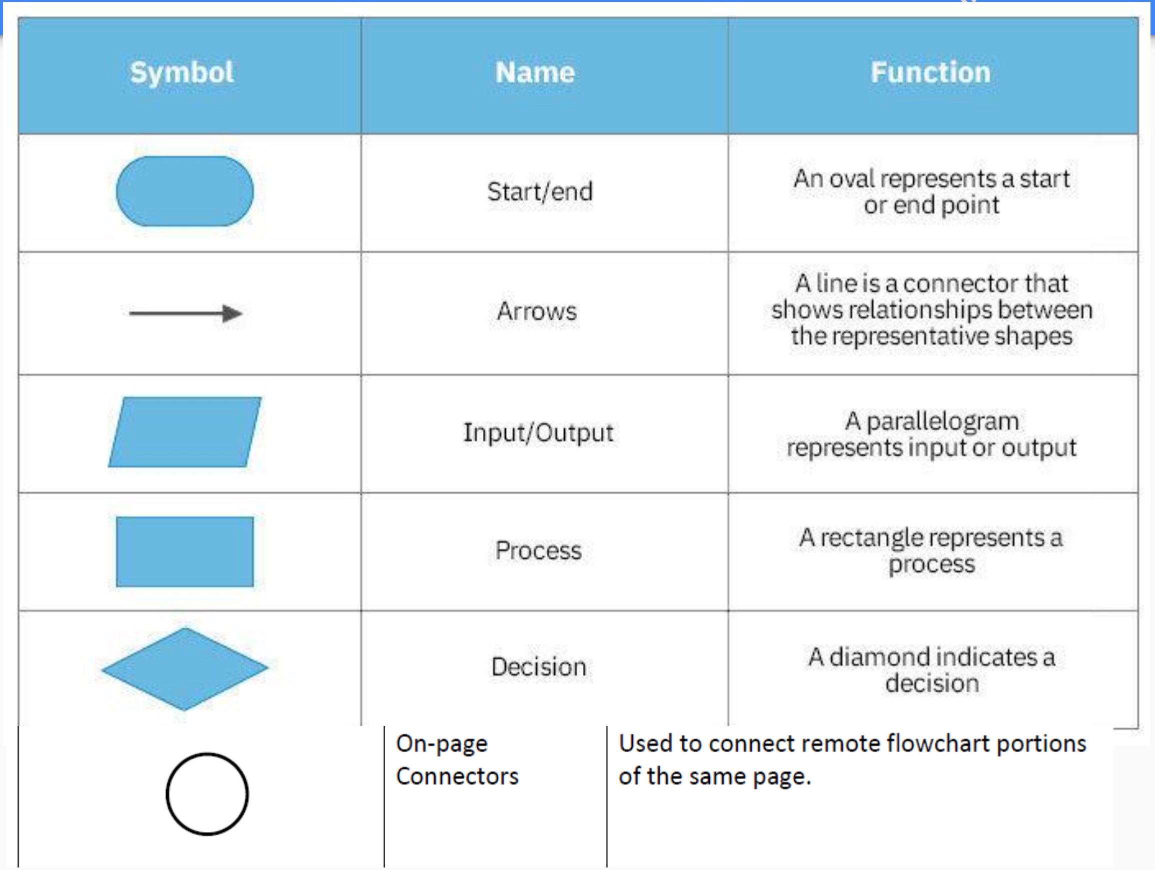
- It is a step-wise representation of a solution to a given problem, which makes it easy to understand.
- An algorithm uses a definite procedure.
- It is not dependent on any programming language, so it is easy to understand for anyone even without programming knowledge.
- Every step in an algorithm has its own logical sequence so it is easy to debug.
- By using algorithm, the problem is broken down into smaller pieces or steps hence, it is easier for programmer to convert it into an actual program.

Disadvantages of Algorithms:

- Algorithm is Time consuming.
- Difficult to show Branching and Looping in Algorithms.
- Big tasks are difficult to put in Algorithms.

Flow chart

- A flowchart is an pictorial representation of an algorithm. It uses boxes of different shapes to denote different types of instructions.
- The actual instructions are written within these boxes use clear and concise statements. These boxes are connected by solid lines having arrow marks to indicate the flow of operation, that is exact sequence in which instructions are to be executed.



Symbol	Name	Function
⬭	Start/end	An oval represents a start or end point
→	Arrows	A line is a connector that shows relationships between the representative shapes
▱	Input/Output	A parallelogram represents input or output
▭	Process	A rectangle represents a process
◆	Decision	A diamond indicates a decision
○	On-page Connectors	Used to connect remote flowchart portions of the same page.

Flow chart

Algorithm to find the sum of any two numbers

Step 1: Start

Step 2: Declare variables a ,b and sum

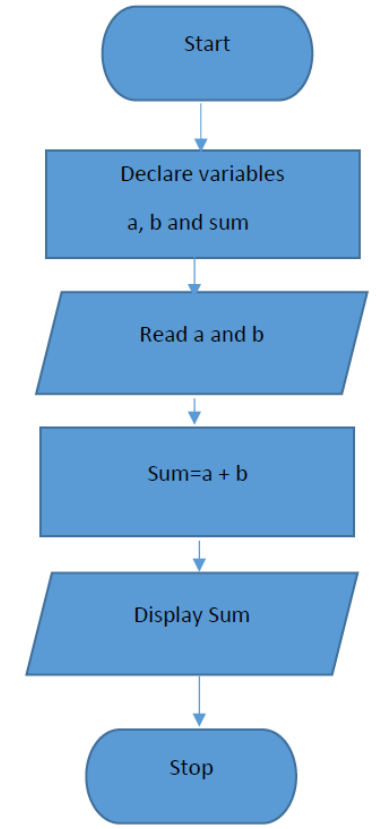
Step 3: Read value of a and b

Step 4: calculate sum= a + b

Step 5: Display sum

Step 6: Stop

Flowchart to add two numbers

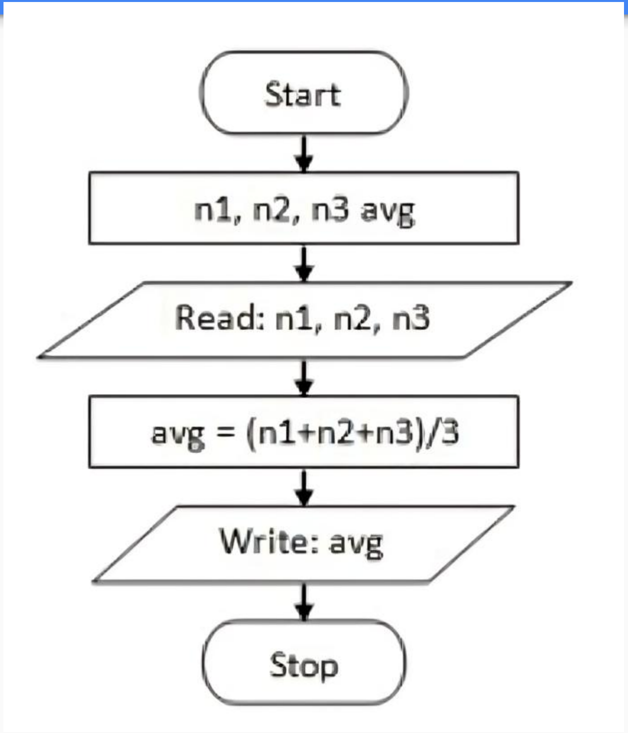


Flow chart

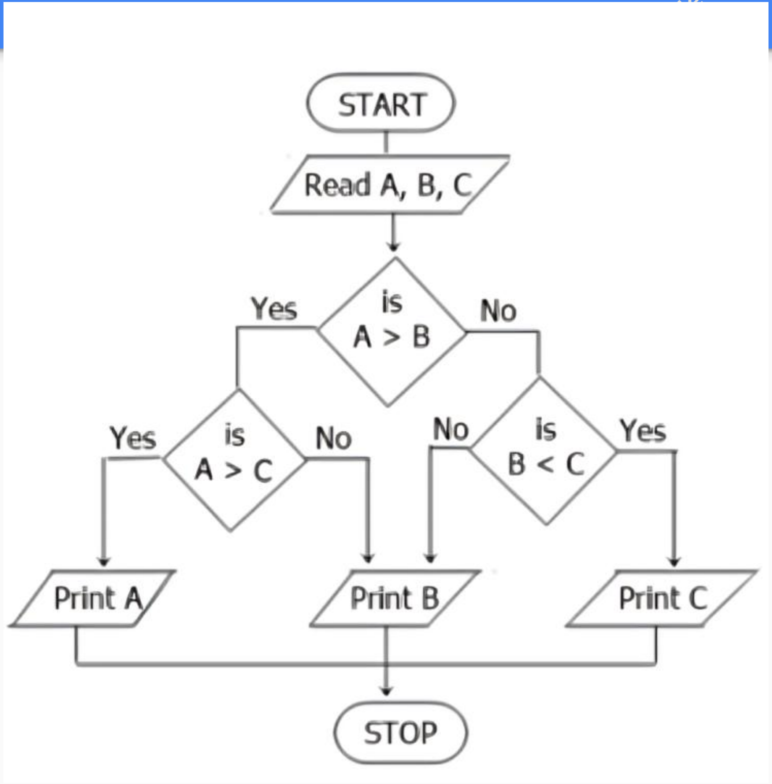
Advantages of using flowcharts

1. **Communication**: Flowcharts are better way of communicating the logic of the system to all concerned.
2. **Effective analysis**: With the help of flowchart, problem can be analyzed in more efficient way.
3. **Proper Documentation**: Program flowcharts serve as a good program documentation, which is needed for various purposes
4. **Efficient coding** : The flowcharts act as a guide or blueprint during the system analysis and program development phase
5. **Proper Debugging**: The flowchart helps in debugging process.
6. **Efficient program maintenance**: Maintenance of operating program becomes easy with the help of flowchart.

Flow chart



Flowchart of average of three numbers



Flowchart of largest of three numbers

Pseudocodes

- Pseudocode is a way of writing or describing programming code or algorithms in a natural language such as English.
- Pseudocode cannot be compiled nor executed as it is only meant to be read by humans. It is also known as 'false code' or 'representation of code.

```
// Pseudocode for adding two numbers

Start

1. Declare variables:
   - int num1, num2, sum;

2. Prompt the user to input the first number:
   - Print "Enter the first number:"
   - Read num1

3. Prompt the user to input the second number:
   - Print "Enter the second number:"
   - Read num2

4. Add the two numbers and store the result in sum:
   - sum = num1 + num2

5. Display the result:
   - Print "The sum is:" and the value of sum

End
```

Pseudocodes

Pros of pseudocodes:

1. Easy to write
2. Easy to convert to code
3. Easy to communicate
4. Errors doesn’t matter
5. Simple to change or update.

Question: Write a pseudocode to find the smallest between two numbers.

Program

- A program is a set of instructions that a computer can execute to perform a specific task. These instructions are written in a programming language, which is a formal language designed for expressing computations. Programs can range from simple scripts to complex applications.

Features of a Good Program: A good program is not just about getting the job done; it's about doing it well. Here are some key features that distinguish a good program:

❑ **Correctness:** The program should produce the expected output for all valid inputs. It should handle errors gracefully and provide informative messages.

❑ **Efficiency:** The program should use resources (CPU time, memory) efficiently. It should avoid unnecessary computations or data storage.

❑ **Readability:** The code should be easy to understand and maintain. Use meaningful variable names, comments, and proper indentation.

❑ **Maintainability:** The program should be easy to modify or update. Well-structured code with modular design makes it easier to change.

Program

- **Usability**: The program should be easy to use for the intended audience. A good user interface can significantly improve the user experience.
- **Reliability**: The program should be robust and handle unexpected situations without crashing. Regular testing and debugging are essential.
- **Reusability**: The program or parts of it should be reusable in other projects. This can save time and effort in future development.
- **Portability:** The program should be able to run on different platforms (operating systems, hardware) with minimal changes.
- **Security:** The program should be secure against vulnerabilities and attacks. This is especially important for programs handling sensitive data.
- **Documentation:** Good documentation is essential for both developers and users. It should include instructions, explanations, and examples.

Different Programming Techniques: Procedural Programming, Modular Programming, Object Oriented Programming

Procedural Programming

A programming paradigm based on structured procedures or routines (functions) that operate on data.

Key Characteristics:

- Sequential execution of instructions.
- Uses procedures or functions to break the program into smaller parts.
- Data and functions are separate; data is passed to functions.

Advantages:

- Simplicity: Easy to learn and implement for small-scale projects.
- Efficient: Direct control over hardware and resources.
- Good for linear and straightforward tasks.

Disadvantages:

- Poor scalability: Difficult to manage for large programs.
- Lack of reusability: Code often needs duplication.
- Difficult to maintain: Changes in data structures can require significant changes in functions.

Examples: Languages: C, Pascal, FORTRAN.

Modular Programming

A technique that divides a program into separate, self-contained modules that perform specific functions.

Key Characteristics:

- Emphasizes code modularity and reusability.
- Each module performs a distinct task and interacts with other modules.
- Facilitates team collaboration.

Advantages:

- Reusability: Modules can be used in different programs.
- Maintainability: Easier to debug and update individual modules.
- Scalability: Suitable for large and complex systems.

Disadvantages:

- Overhead: Slightly more complex than procedural programming.
- Dependency: Modules may need to be tightly coupled if not designed well.

Examples: Languages: Python, Java, Ada.

Object Oriented Programming

A programming paradigm based on the concept of "objects," which contain data (attributes) and code (methods).

Key Characteristics:

- Encapsulation: Bundles data and methods that operate on the data.
- Inheritance: Enables a class to derive properties and behavior from another class.
- Polymorphism: Allows methods to perform different tasks based on the object that invokes them.
- Abstraction: Hides implementation details from the user.

Advantages:

- Code reusability: Classes can be reused across programs.
- Scalability: Facilitates large-scale application development.
- Easier maintenance: Modular structure makes it easier to manage changes.

Disadvantages:

- Complexity: Can be harder to learn and implement for beginners.
- Overhead: May introduce performance and resource consumption issues.

Examples: Languages: Python, Java, C++, C#.

Differences between Procedural and Object Oriented Programming

Procedural Programming	Object Oriented Programming
In Procedural programming, a program is divided into small programs that are referred to as functions.	In OOP, a program is divided into small parts that are referred to as objects.
It is less secure than OOPs.	Data hiding is possible in object-oriented programming due to abstraction. So, it is more secure than procedural programming.
It follows a top-down approach.	It follows a bottom-up approach.
There are no access modifiers in procedural programming.	The access modifiers in OOP are named as private, public, and protected.
Procedural programming does not have the concept of inheritance.	There is a feature of inheritance in object-oriented programming.
It is not appropriate for complex problems.	It is appropriate for complex problems.
Examples of Procedural programming include C, Fortran, Pascal, and VB.	The examples of object-oriented programming are - .NET, C#, Python, Java, VB.NET, and C++.

Also refer to Class Notes too.

THANK YOU

Any Queries ?