BASICS OF PROGRAMMING

1. INTRODUCTION

Intelligence is one of the key characteristics which differentiate a human being from other living creatures on the earth. Basic intelligence covers day to day problem solving and making strategies to handle different situations which keep arising in day to day life. One person goes Bank to withdraw money. After knowing the balance in his account, he/she decides to with draw the entire amount from his account but he/she has to leave minimum balance in his account. Here deciding about how much amount he/she may with draw from the account is one of the examples of the basic intelligence. During the process of solving any problem, one tries to find the necessary steps to be taken in a sequence. In this Unit you will develop your understanding about problem solving and approaches.

2. PROBLEM SOLVING

Can you think of a day in your life which goes without problem solving? Answer to this question is of course, No. In our life we are bound to solve problems. In our day to day activity such as purchasing something from a general store and making payments, depositing fee in school, or withdrawing money from bank account. All these activities involve some kind of problem solving. It can be said that whatever activity a human being or machine do for achieving a specified objective comes under problem solving. To make it clearer, let us see some other examples.

Example1: If you are watching a news channel on your TV and you want to change it to a sports channel, you need to do something i.e. move to that channel by pressing that channel number on your remote. This is a kind of problem solving.

Example 2: One Monday morning, a student is ready to go to school but yet he/she has not picked up those books and copies which are required as per timetable. So here picking up books and copies as per timetable is a kind of problem solving.

Example 3: If someone asks to you, what is time now? So seeing time in your watch and telling him is also a kind of problem solving.

Example 4: Some students in a class plan to go on picnic and decide to share the expenses among them. So calculating total expenses and the amount an individual have to give for picnic is also a kind of problem solving.

Now, broadly we can say that problem is a kind of barrier to achieve something and problem solving is a process to get that barrier removed by performing some sequence of activities

Here it is necessary to mention that all the problems in the world can not be solved. There are some problems which have no solution and these problems are called *Open Problems*.

If you can solve a given problem then you can also write an algorithm for it. In next section we will learn what is an *algorithm.*

2.1 ALGORITHM

Algorithm can be defined as: “A sequence of activities to be processed for getting desired output from a given input.”

Webopedia defines an algorithm as: “A formula or set of steps for solving a particular problem. To be an algorithm, a set of rules must be unambiguous and have a clear stopping point”. There may be more than one way to solve a problem, so there may be more than one algorithm for a problem.

Now, if we take definition of algorithm as: “A sequence of activities to be processed for getting desired output from a given input.” Then we can say that:
1. Getting specified output is essential after algorithm is executed.
2. One will get output only if algorithm stops after finite time.
3. Activities in an algorithm to be clearly defined in other words for it to be unambiguous.
Before writing an algorithm for a problem, one should find out what is/are the inputs to the algorithm and what is/are expected output after running the algorithm.

Now let us take some exercises to develop an algorithm for some simple problems: While writing algorithms we will use following symbol for different operations:
‘+’ for Addition
‘-’ for Subtraction
‘*’ for Multiplication
‘/’ for Division and
‘ ’ for assignment. For example A X*3 means A will have a value of X*3.

Example of Algorithm

Problem 1: Find the area of a Circle of radius r.
Inputs to the algorithm:
Radius r of the Circle.
Expected output:
Area of the Circle

Algorithm:
```
Step1: Read\input the Radius r of the Circle
Step2: Area PI*r*r // calculation of area
Step3: Print Area
```

Problem2: Write an algorithm to read two numbers and find their sum.
Inputs to the algorithm:
First num1.
Second num2.
Expected output:
Sum of the two numbers.

Algorithm:
```
Step1: Start
Step2: Read\input the first num1.
Step3: Read\input the second num2.
Step4: Sum num1+num2 // calculation of sum
Step5: Print Sum
Step6: End
```

Problem 3: Convert temperature Fahrenheit to Celsius
Inputs to the algorithm:
Temperature in Fahrenheit
Expected output:
Temperature in Celsius

Algorithm:
```
Step1: Start
Step 2: Read Temperature in Fahrenheit F
Step 3: C 5/9*(F32)
```

Step 4: Print Temperature in Celsius: C
Step5: End

Type of Algorithms

The algorithm and flowchart, classification to the three types of *control structures*. They are:
1. Sequence
2. Branching (Selection)
3. Loop (Repetition)

These three control structures are sufficient for all purposes. The sequence is exemplified by sequence of statements place one after the other – the one above or before another gets executed first. In flowcharts, sequence of statements is usually contained in the rectangular process box.

The *branch* refers to a binary decision based on some condition. If the condition is true, one of the two branches is explored; if the condition is false, the other alternative is taken. This is usually represented by the 'if-then' construct in pseudo-codes and programs. In flowcharts, this is represented by the diamond-shaped decision box. This structure is also known as the *selection* structure.

Problem1: write algorithm to find the greater number between two numbers
```
Step1: Start
Step2: Read/input A and B
Step3: If A greater than B then C=A
Step4: if B greater than A then C=B
Step5: Print C
Step6: End
```

Problem2: write algorithm to find the result of equation: ( ) {
```
Step1: Start
Step2: Read/input x
Step3: If X Less than zero then F=-X
Step4: if X greater than or equal zero then F=X
Step5: Print F
Step6: End
```

Problem3: A algorithm to find the largest value of any three numbers.
```
Step1: Start
```

```
Step2: Read/input A,B and C
Step3: If (A>=B) and (A>=C) then Max=A
Step4: If (B>=A) and (B>=C) then Max=B
Step5:If (C>=A) and (C>=B) then Max=C
Step6: Print Max
Step7: End
```

The *loop* allows a statement or a sequence of statements to be repeatedly executed based on some loop condition. It is represented by the ‘while’ and ‘for’ constructs in most programming languages, for unbounded loops and bounded loops respectively. (Unbounded loops refer to those whose number of iterations depends on the eventuality that the termination condition is satisfied; bounded loops refer to those whose number of iterations is known before-hand.) In the flowcharts, a back arrow hints the presence of a loop. A trip around the loop is known as iteration. You must ensure that the condition for the termination of the looping must be satisfied after some finite number of iterations, otherwise it ends up as an infinite loop, a common mistake made by inexperienced programmers. The loop is also known as the *repetition* structure.

Examples:

Problem1: An algorithm to calculate even numbers between 0 and 99

```
1. Start
2. I ← 0
3. Write I in standard output
4. I ← I+2
5. If (I <=98) then go to line 3
6. End
```

Problem3: Design an algorithm which generates even numbers between 1000 and 2000 and then prints them in the standard output. It should also print total sum:

```
1. Start
2. I ← 1000 and S ← 0
3. Write I
4. S ← S + I
5. I ← I + 2
6. If (I <= 2000) then go to line 3
else go to line 7
7. Write S
8. End
```

Properties of algorithm

Donald Ervin Knuth has given a list of five properties for a,algorithm, these properties are:

1) Finiteness: An algorithm must always terminate after a finite number of steps. It means after every step one reach closer to solution of the problem and after a finite number of steps algorithm reaches to an end point.

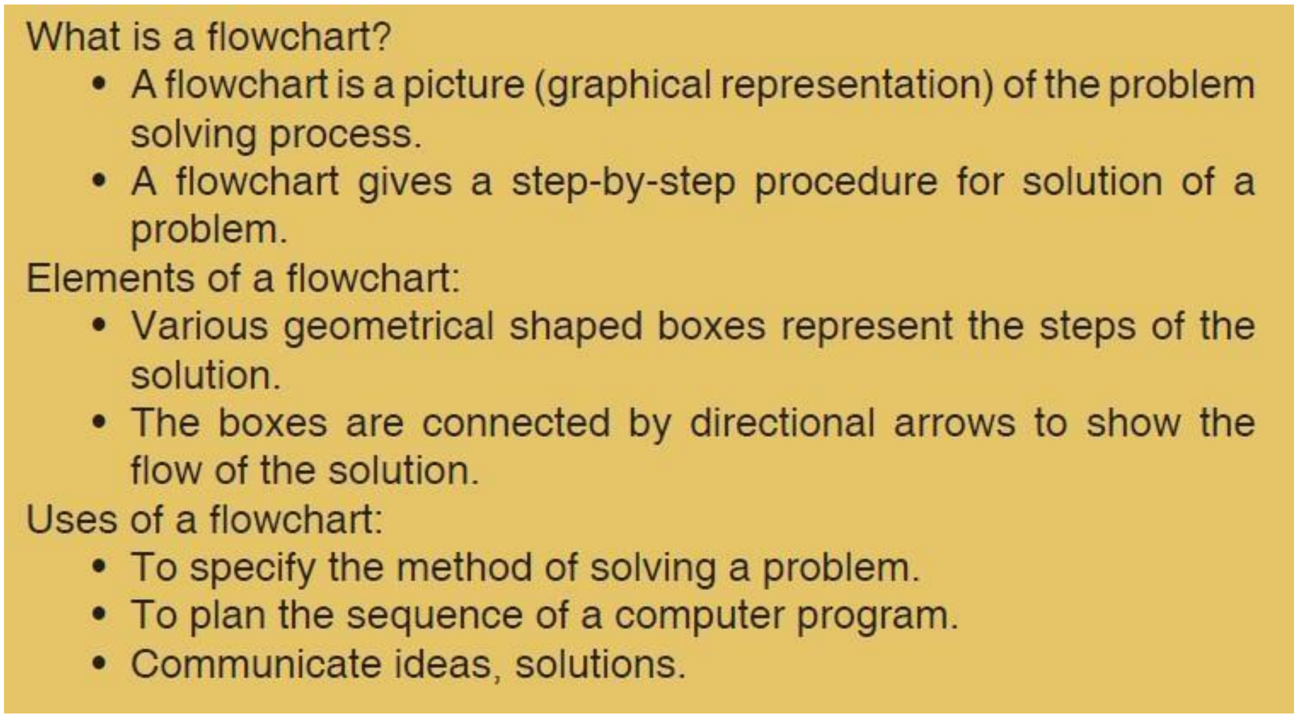
2) Definiteness: Each step of an algorithm must be precisely defined. It is done by well thought actions to be performed at each step of the algorithm. Also the actions are defined unambiguously for each activity in the algorithm.

3) Input: Any operation you perform need some beginning value/quantities associated with different activities in the operation. So the value/quantities are given to the algorithm before it begins.

4) Output: One always expects output/result (expected value/quantities) in terms of output from an algorithm. The result may be obtained at different stages of the algorithm. If some result is from the intermediate stage of the operation then it is known as intermediate result and result obtained at the end of algorithm is known as end result. The output is expected value/quantities always have a specified relation to the inputs

5) Effectiveness: Algorithms to be developed/written using basic operations. Actually operations should be basic, so that even they can in principle be done exactly and in a finite amount of time by a person, by using paper and pencil only.

2.2 FLOWCHART



What is a flowchart?

- A flowchart is a picture (graphical representation) of the problem solving process.
- A flowchart gives a step-by-step procedure for solution of a problem.

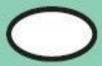
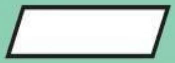
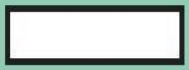
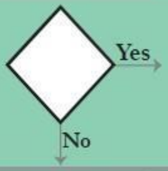
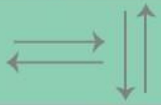
Elements of a flowchart:

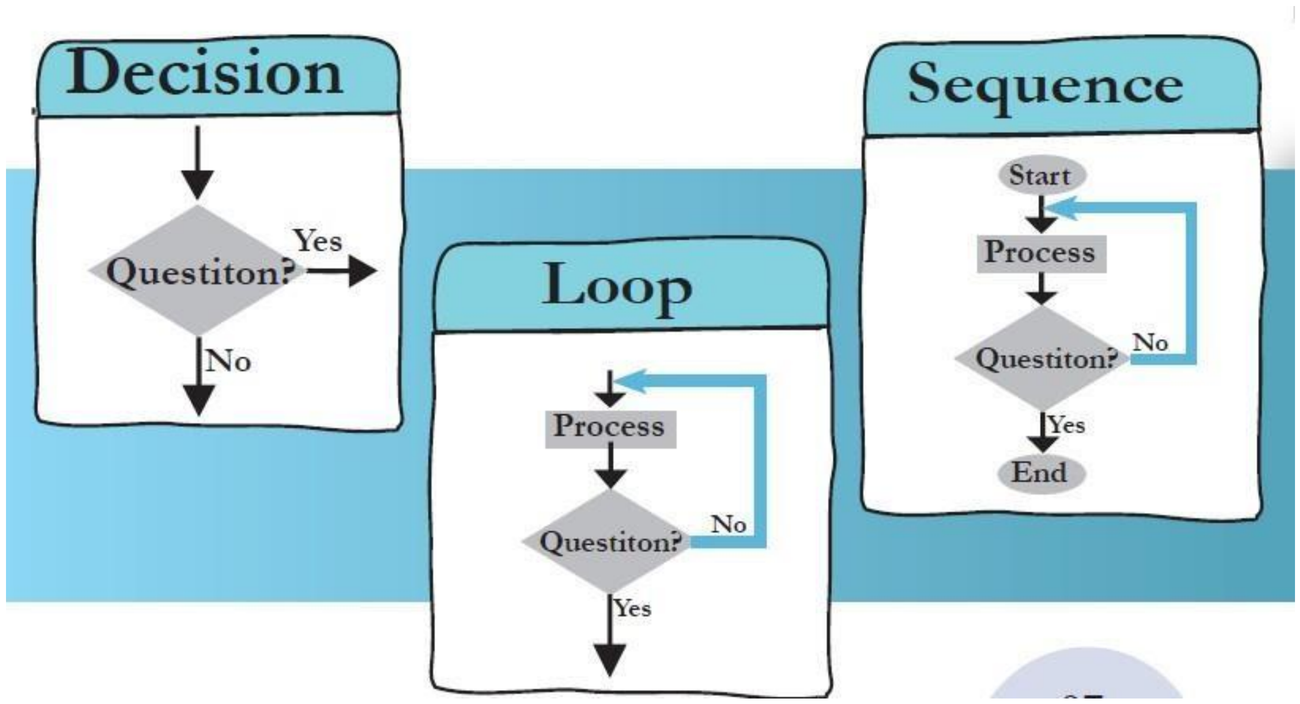
- Various geometrical shaped boxes represent the steps of the solution.
- The boxes are connected by directional arrows to show the flow of the solution.

Uses of a flowchart:

- To specify the method of solving a problem.
- To plan the sequence of a computer program.
- Communicate ideas, solutions.

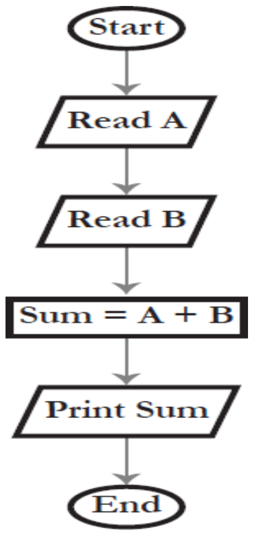
Flowchart symbols and their purpose

Flowchart symbols	Geometric shape	Purpose
Ellipse		Ellipse is used to indicate the start and end of a flowchart. Start written in the ellipse indicates the beginning of a flowchart. End or Stop or Exit written in the ellipse indicates the end of the flowchart.
Parallelogram		A parallelogram is used to read data (input) or to print data (output).
Rectangle		A rectangle is used to show the processing that takes place in the flowchart.
Diamond		A diamond with two branches is used to show the decision making step in a flowchart. A question is specified in the diamond. The next step in the sequence is based on the answer to the question which is “Yes” or “No”.
Arrows		Arrows are used to connect the steps in a flowchart, to show the flow or sequence of the problem solving process



Examples of Flowchart :

1. Find Sum of Two Numbers.



Computer Language

Just as humans use language to communicate, and different regions have different languages, computers also have their own languages that are specific to them.

Different kinds of languages have been developed to perform different types of work on the computer. Basically, languages can be divided into two categories according to how the computer understands them.

Two Basic Types of Computer Language

- **Low-Level Languages:** A language that corresponds directly to a specific machine
- **High-Level Languages:** Any language that is independent of the machine

Low-Level Languages

Low-level computer languages are either machine codes or are very close them. A computer cannot understand instructions given to it in high-level languages or in English. It can only understand and execute instructions given in the form of machine language i.e. binary. There are two types of low-level languages:

- **Machine Language:** a language that is directly interpreted into the hardware
- **Assembly Language:** a slightly more user-friendly language that directly corresponds to machine language

Machine Language

Machine language is the lowest and most elementary level of programming language and was the first type of programming language to be developed. Machine language is basically the only language that a computer can understand and it is usually written in hex.

In fact, a manufacturer designs a computer to obey just one language, its machine code, which is represented inside the computer by a string of binary digits (bits) 0 and 1. The symbol 0 stands for the absence of an electric pulse and the 1 stands for the presence of an electric pulse. Since a computer is capable of recognizing electric signals, it understands machine language.

Advantages	Disadvantages
Machine language makes fast and efficient use of the computer.	All operation codes have to be remembered
It requires no translator to translate the code. It is directly understood by the computer.	All memory addresses have to be remembered.
	It is hard to amend or find errors in a program written in the machine language.

Assembly Language

Assembly language was developed to overcome some of the many inconveniences of machine language. This is another low-level but very important language in which operation codes and operands are given in the form of alphanumeric symbols instead of 0’s and l’s.

These alphanumeric symbols are known as mnemonic codes and can combine in a maximum of five-letter combinations e.g. ADD for addition, SUB for subtraction, START, LABEL etc. Because of this feature, assembly language is also known as ‘Symbolic Programming Language.'

This language is also very difficult and needs a lot of practice to master it because there is only a little English support in this language. Mostly assembly language is used to help in compiler orientations. The instructions of the assembly language are converted to machine codes by a language translator and then they are executed by the computer.

Advantages	Disadvantages
Assembly language is easier to understand and use as compared to machine language.	Like machine language, it is also machine dependent/specific.
It is easy to locate and correct errors.	Since it is machine dependent, the programmer also needs to understand the hardware.
It is easily modified.	

High-Level Languages

High-level computer languages use formats that are similar to English. The purpose of developing high-level languages was to enable people to write programs easily, in their own native language environment (English).

High-level languages are basically symbolic languages that use English words and/or mathematical symbols rather than mnemonic codes. Each instruction in the high-level language is translated into many machine language instructions that the computer can understand.

Advantages	Disadvantages
High-level languages are user-friendly	A high-level language has to be translated into the machine language by a translator, which takes up time
They are similar to English and use English vocabulary and well-known symbols	The object code generated by a translator might be inefficient compared to an equivalent assembly language program
They are easier to learn	
They are easier to maintain	
They are problem-oriented rather than 'machine'-based	

Types of High-Level Languages

Many languages have been developed for achieving a variety of different tasks. Some are fairly specialized, and others are quite general.

These languages, categorized according to their use, are:

1) Algebraic Formula-Type Processing

These languages are oriented towards the computational procedures for solving mathematical and statistical problems.

Examples include:

- BASIC (Beginners All Purpose Symbolic Instruction Code)
- FORTRAN (Formula Translation)

- PL/I (Programming Language, Version 1)
- ALGOL (Algorithmic Language)
- APL (A Programming Language)

2. Business Data Processing

These languages are best able to maintain data processing procedures and problems involved in handling files. Some examples include:

- COBOL (Common Business Oriented Language)
- RPG (Report Program Generator)

3. String and List Processing

These are used for string manipulation, including search patterns and inserting and deleting characters. Examples are:

- LISP (List Processing)
- Prolog (Program in Logic)

4. Object-Oriented Programming Language

In OOP, the computer program is divided into objects. Examples are:

- C++
- Java

5. Visual Programming Language

These programming languages are designed for building Windows-based applications.Examples are:

- Visual Basic
- Visual Java
- Visual C

.Compiler :

compiler is a computer program that translates a program in a *source language* into an equivalent program in a *target language*.

Compilers: Translate a source (human-writable) program to an executable (machine-readable) program.

- Translate the entire program.
- Convert the entire program to machine code, when the syntax errors are removed then converted into the object code
- Requires more main memory
- Neither source nor the compiler are required for execution.
- Slow for debugging and testing.
- Execution time is less.

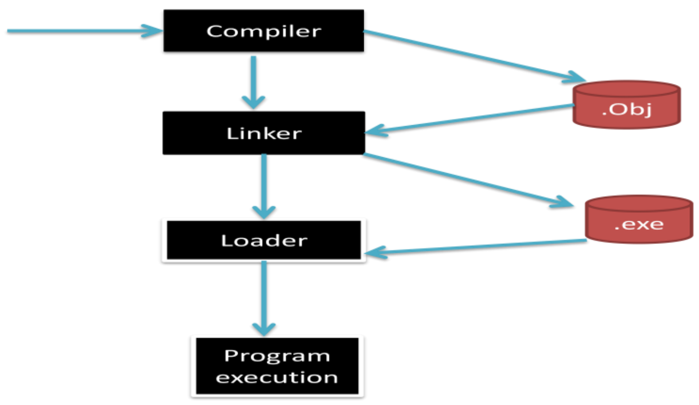
Interpreter :

Interpreter is a programthat executes instructions written in a high-level language.

Interpreters: Convert a source program and execute it at the same time.

- Translate the program line by line.
- each time the program is executed ,every line is checked for syntax error & then converted to equivalent machine code directly.
- Requires less main memory
- Source program and the interpreter are required for execution.
- Good for fast debugging and testing.
- Execution time is more.

Execution of C Program :



Linker :

Tool that merges the object files produced by *separate compilation* or assembly and creates an executable file.

A program that takes as input the object files of one or more separately compiled program modules, and links them together into a complete executable program, resolving reference from one module to another.

• Three tasks:-

- ❑ Searches the program to find library routines used by program, e.g. printf(),sqrt(),strcat() and various other.
- ❑ Determines the memory locations that code from each module will occupy and relocates its instructions by adjusting absolute references.

Relocation, which modifies the object program so that it can be loaded at an address different from the location originally specified.

It combines two or more separate object programs and supplies the information needed to allow references between them .

Loader :

Loading means physically placing the machine instructions and data into main memory.

Loader is a program that places a program’s instructions and data into primary strorage.

Object(.obj) File and Executable(.exe) File :

A single object file might contain machine code for only one procedure or a set of procedures.

An executable file must contain all the machine code needed for a particular program;

Basic Structure of C Program :

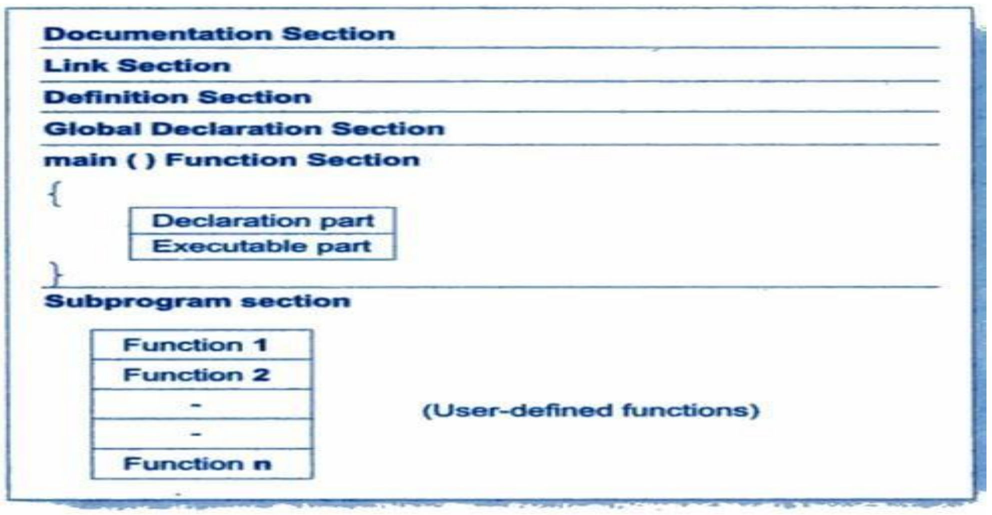
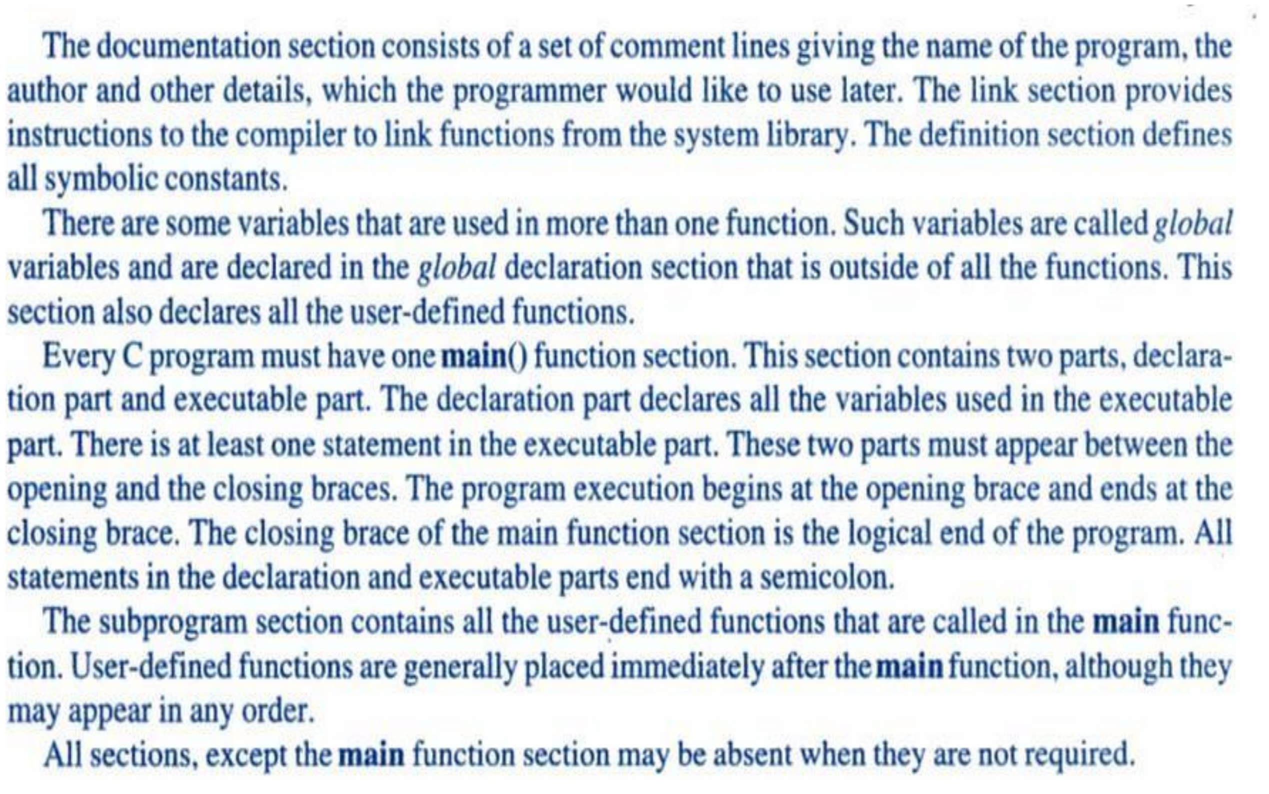


Fig. 1.9 *An overview of a C program*



The documentation section consists of a set of comment lines giving the name of the program, the author and other details, which the programmer would like to use later. The link section provides instructions to the compiler to link functions from the system library. The definition section defines all symbolic constants.

There are some variables that are used in more than one function. Such variables are called *global* variables and are declared in the *global* declaration section that is outside of all the functions. This section also declares all the user-defined functions.

Every C program must have one **main**() function section. This section contains two parts, declaration part and executable part. The declaration part declares all the variables used in the executable part. There is at least one statement in the executable part. These two parts must appear between the opening and the closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function section is the logical end of the program. All statements in the declaration and executable parts end with a semicolon.

The subprogram section contains all the user-defined functions that are called in the **main** function. User-defined functions are generally placed immediately after the **main** function, although they may appear in any order.

All sections, except the **main** function section may be absent when they are not required.