

Programming Language:

COMPUTER LANGUAGES

To write a program (tells what to do) for a computer, we must use a computer language.

Over the years computer languages have evolved from machine languages to natural languages.

The following is the summary of computer languages

1940's	--	Machine Languages
1950's	--	Symbolic Languages
1960's	--	High Level Languages

Language Translators

These are the programs which are used for converting the programs in one language into machine language instructions, so that they can be executed by the computer.

- 1) **Compiler:** It is a program which is used to convert the high level language programs into machine language
- 2) **Assembler:** It is a program which is used to convert the assembly level language programs into machine language
- 3) **Interpreter:** It is a program, it takes one statement of a high level language program, translates it into machine language instruction and then immediately executes the resulting machine language instruction and so on.

Comparison between a Compiler and Interpreter

COMPILER	INTERPRETER
A Compiler is used to compile an entire program and an executable program is generated through the object program	An interpreter is used to translate each line of the program code immediately as it is entered
The executable program is stored in a disk for future use or to run it in another computer	The executable program is generated in RAM and the interpreter is required for each run of the program
The compiled programs run faster	The Interpreted programs run slower
Most of the Languages use compiler	A very few languages use interpreters.

ALGORITHM

Algorithm is a finite sequence of instructions, each of which has a clear meaning and can be performed with a finite amount of effort in a finite length of time. No matter what the input values may be, an algorithm terminates after executing a finite number of instructions.

We represent an algorithm using a pseudo language that is a combination of the constructs of a programming language together with informal English statements.

The ordered set of instructions required to solve a problem is known as an *algorithm*.

The characteristics of a good algorithm are:

- **Precision** – the steps are precisely stated (defined).
- **Uniqueness** – results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
- **Finiteness** – the algorithm stops after a finite number of instructions are executed.
- **Input** – the algorithm receives input.
- **Output** – the algorithm produces output.
- **Generality** – the algorithm applies to a set of inputs.

Example

Q. Write an algorithm to find out number is odd or even?

Ans.









```
step 1 : start
step 2 : input number
step 3 : rem=number mod 2
step 4 : if rem=0 then
print "number even"
else
    print "number odd"
endif
step 5 : stop
```

FLOWCHART

Flowchart is a diagrammatic representation of an algorithm. Flowchart is very helpful in writing program and explaining program to others.

Symbols Used In Flowchart

Different symbols are used for different states in flowchart, For example: Input/Output and decision making has different symbols. The table below describes all the symbols that are used in making flowchart

Symbol	Purpose	Description
	Flow line	Used to indicate the flow of logic by connecting symbols.
	Terminal(Stop/Start)	Used to represent start and end of flowchart.
	Input/Output	Used for input and output operation.
	Processing	Used for airthmetic operations and data-manipulations.
	Desicion	Used to represent the operation in which there are two alternatives, true and false.
	On-page Connector	Used to join different flowline
	Off-page Connector	Used to connect flowchart portion on different page.
	Predefined Process/Function	Used to represent a group of statements performing one processing task.

INTRODUCTION TO CPP LANGUAGE

INTRO:-C++ is an object oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.

What is C++?

- C++ is a cross-platform language that can be used to create high-performance applications.
 - C++ was developed in **1979** at Bell Labs by **Bjarne Stroustrup** on variety of platforms, such as windows, Mac OS & various version of UNIX OS, as an extension to the C language.
 - C++ gives programmers a high level of control over system resources and memory.
 - The language was updated 4 major times in 2011, 2014, 2017, and 2020 to C++11, C++14, C++17, C++20.
-

Why Use C++

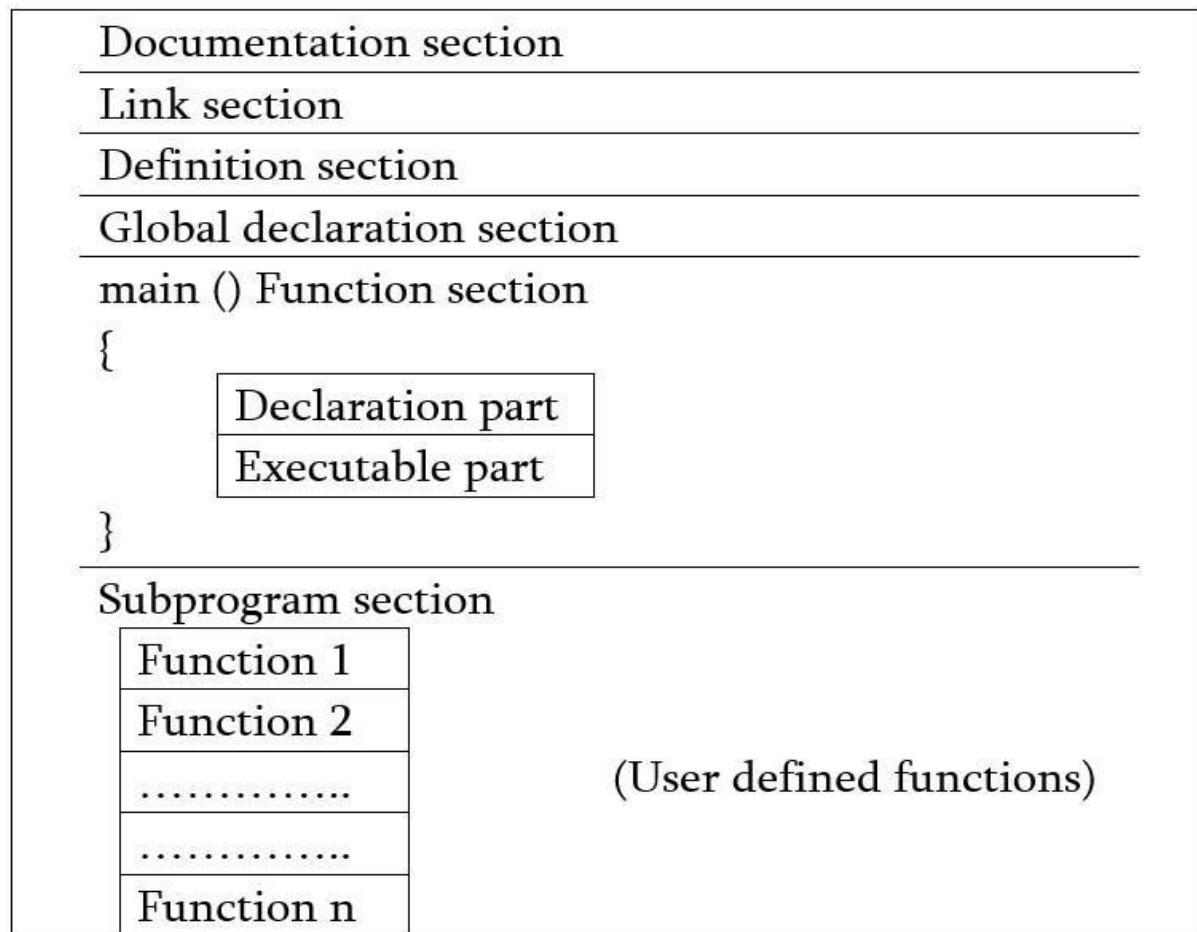
- C++ is one of the world's most popular programming languages.
 - C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.
 - C++ is portable and can be used to develop applications that can be adapted to multiple platforms.
 - C++ is fun and easy to learn!
 - As C++ is close to C, C# and Java, it makes it easy for programmers to switch to C++ or vice versa.
-

Difference between C and C++

C++ was developed as an extension of C, and both languages have almost the same syntax.

- The main difference between C and C++ is that C++ support classes and objects, while C does not.
- C++ is bottom to top approach while C is top to bottom approach.

BASIC STRUCTURE OF C/CPP PROGRAMMING



1. **Documentation section:** 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.

2. **Link section:** The link section provides instructions to the compiler to link functions from the system library such as using the #include directive.
3. **Definition section:** The definition section defines all symbolic constants such using the #define directive.
4. **Global declaration section:** 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.
5. **main () function section:** Every C program must have one main function section. This section contains two parts; declaration part and executable part
 1. **Declaration part:** The declaration part declares all the variables used in the executable part.
 2. **Executable part:** There is at least one statement in the executable part. These two parts must appear between the opening and closing braces. The program execution begins at the opening brace and ends at the closing brace. The closing brace of the main function is the logical end of the program. All statements in the declaration and executable part end with a semicolon.
6. **Subprogram section:** If the program is a multi-function program then 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.

PROCESS OF COMPILING AND RUNNING CPP PROGRAM

We will briefly highlight key features of the Compilation model here.

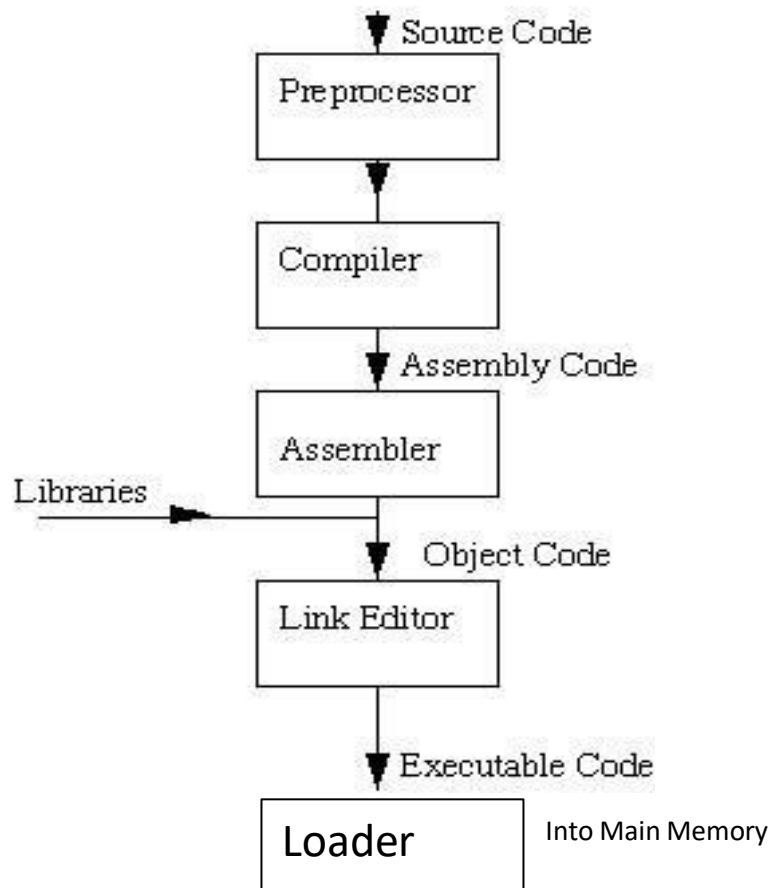


Fig:C++ Compilation Model

The Preprocessor: Convert into Interactive code(.i)

The Preprocessor accepts source code as input and is responsible for

- removing comments
- Interpreting special *preprocessor directives* denoted by #. For example
- #include -- includes contents of a named file. Files usually called *header* files.
e.g ○ #include <math.h> -- standard library maths file.
○ #include <iostream.h> -- standard library I/O file
- #define -- defines a symbolic name or constant. Macro substitution. ○
#define MAX_ARRAY_SIZE 100

CPP Compiler : Covert into Assembly code(.s)

The CPP compiler translates source to assembly code. The source code is received from the preprocessor.

Assembler: Convert into Object code (.obj)

The assembler creates object code.

Linker: Convert into executable code (.exe)

If a source file references library functions or functions defined in other source files the *link editor* combines these functions (with main()) to create an executable file.

Loader: Load into main Memory

CPP TOKENS

CPP tokens are the basic buildings blocks in CPP language which are constructed together to write a CPP program.

Each and every smallest individual unit in a CPP program is known as tokens.

CPP tokens are of six types. They are

Keywords	(eg: int, while),
Identifiers	(eg: main, total),
Constants	(eg: 10, 20),
Strings	(eg: —total , —hello),
Special symbols	(eg: (), {}),
Operators	(eg: +, /,-,*)

CPP KEYWORDS

CPP keywords are the words that convey a special meaning to the CPP compiler. The keywords cannot be used as variable names.

auto	break	case	char	const
continue	default	do	double	else
enum	extern	float	for	goto
if	int	long	register	return
short	signed	sizeof	static	struct
switch	typedef	union	unsigned	void
volatile	while			

CPP IDENTIFIERS

Identifiers are used as the general terminology for the names of variables, functions and arrays. These are user defined names consisting of arbitrarily long sequence of letters and digits with either a letter or the underscore(_) as a first character.

There are certain rules that should be followed while naming c identifiers:

- They must begin with a letter or underscore (_).
- They must consist of only letters, digits, or underscore. No other special character is allowed.
- It should not be a keyword.
- It must not contain white space.
- It should be up to 31 characters long as only first 31 characters are significant.

CPP CONSTANTS

A CPP constant refers to the data items that do not change their value during the program execution.

Several types of CPP constants that are allowed in CPP are:

NOTE: An escape sequence consumes only one byte of space as it represents a single character.

Escape Sequence	Description
a	Audible alert(bell)
b	Backspace
f	Form feed
n	New line
r	Carriage return
t	Horizontal tab
v	Vertical tab
\	Backslash
“	Double quotation mark
‘	Single quotation mark
?	Question mark
	Null

VARIABLES

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in C has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

The name of a variable can be composed of letters, digits, and the underscore character. It must begin with either a letter or an underscore. Upper and lowercase letters are distinct because CPP is case-sensitive. Based on the basic types explained in the previous chapter, there will be the following basic variable types –

Type	Description
char	Typically a single octet(one byte). This is an integer type.
int	The most natural size of integer for the machine.
float	A single-precision floating point value.
double	A double-precision floating point value.
void	Represents the absence of type.

Variable Definition in C++

A variable definition tells the compiler where and how much storage to create for the variable. A variable definition specifies a data type and contains a list of one or more variables of that type as follows –

```
type variable_list;
```

Types of Data Types in CPP

Here are the five major categories into which data types are divided in CPP language:

Data Type	Example of Data Type
Basic Data Type	Floating-point, integer, double, character.
Derived Data Type	structure, array, etc.
Enumerated Data Type	Enums
Void Data Type	Empty Value
Bool Type	True or False

The basic data types are also known as the primary data types in CPP programming.

Primary Data Types in CPP

Here are the five primitive or primary data types that one can find in CPP programming language:

- 1. Integer** – We use these for storing various whole numbers, such as 5, 8, 67, 2390, etc.
- 2. Character** – It refers to all ASCII character sets as well as the single alphabets, such as 'x', 'Y', etc.
- 3. Double** – These include all large types of numeric values that do not come under either floating-point data type or integer data type.
- 4. Floating-point** – These refer to all the real number values or decimal points, such as 40.1, 820.673, 5.9, etc.
- 5. Void** – This term refers to no values at all. We mostly use this data type when defining the functions in a program.

Integer Types

Type	Storage size	Value range
Char	1 byte	-128 to 127 or 0 to 255
unsigned char	1 byte	0 to 255
signed char	1 byte	-128 to 127
Int	2 or 4 bytes	-32,768 to 32,767 or -2,147,483,648 to 2,147,483,647
unsigned int	2 or 4 bytes	0 to 65,535 or 0 to 4,294,967,295
Short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
Long	8 bytes or (4bytes for 32 bit OS)	-9223372036854775808 to 9223372036854775807
unsigned long	8 bytes	0 to 18446744073709551615

To get the exact size of a type or a variable on a particular platform, you can use the **sizeof** operator.

Floating-Point Types

The following table provide the details of standard floating-point types with storage sizes and value ranges and their precision –

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

The void Type: no value is specified

OPERATORS AND EXPRESSIONS :

CPP language offers many types of operators. They are,

- Arithmetic operators (+, -, *, /, %)
 - Assignment operators (=)
 - Relational operators (<, <=, >, >=)
 - Logical operators (and, or, !)
 - Bit wise operators (&, |, ~, <<, >>)
 - Conditional operators (ternary operators)
(condition?true:false)
 - Increment/decrement operators (post & pre)
 - Special operators (&, *)
-

C++ Syntax :

Example:

```
#include <iostream.h>
#include<conio.h>

int main() {
    clrscr();
    cout << "Welcome to all!";
    getch();
    return 0;
}
```

Example explained

Line 1: `#include <iostream.h>` is a **header file library** that lets us work with input and output objects, such as `cout` (used in line 6). Header files add functionality to C++ programs.

Line 2: `#include <conio.h>` is a **header file library** that lets us work with such as `clrscr()` & `getch()` (used in line 5 & line 8). Header files add functionality to C++ programs.

Line 3: A blank line. C++ ignores white space. But we use it to make the code more readable.

Line 4: Another thing that always appear in a C++ program, is `int main()`. This is called a **function**. Any code inside its curly brackets `{ }` will be executed.

Line 5: Another thing that always appear in a C++ program, is `clrscr()`. This is called a **clear screen function**. It erases the previous output.

Line 6: `cout` (pronounced "see-out") is an **object** used together with the *insertion operator* (`<<`) to output/print text. In our example it will output "Welcome to all".

Note: Every C++ statement ends with a semicolon `;`.

Note: The body of `int main()` could also been written as:

```
void main () {clrscr(); cout << "Welcome to all! ";getch(); return 0; }
```

Remember: The compiler ignores white spaces. However, multiple lines makes the code more readable.

Line 6: Another thing that always appear in a C++ program, is `getch()`. This is called a **get char function**.It holds the console window for output.

Line 7: `return 0` ends the main function.

Line 8: Do not forget to add the closing curly bracket `}` to actually end the main function.

C++ User Output

`cout` is used to output (print) values

`cout` (pronounced "see-out") is an **object** used together with the *insertion operator* (`<<`) to output/print text

C++ User Input

You have already learned that `cout` is used to output (print) values. Now we will use `cin` to get user input.

`cin` is a predefined variable that reads data from the keyboard with the extraction operator (`>>`).

C++ Syntax in latest version(onwards C11 version) :

Example:

```
#include <iostream>
using namespace std;

int main()
{
    cout << "Welcome to all!";
    return 0;
}
```
