0301102 LOGIC DEVELOPMENT & PROGRAMMING

UNIT – 2

Principles of Programming Language

Introduction to Programming

- A computer is a device that can accept human instruction, processes it and responds to it or a computer is a computational device which is used to process the data under the control of a computer program.
- Program is a sequence of instruction along with data.
- A program is a set of instructions given to a computer to perform a specific operation or computer is a computational device which is used to process the data under the control of a computer program.
- While executing the program, raw data is processed into a desired output format.
- These computer programs are written in a programming language which are high level languages.

Introduction to Programming

- Like we have different languages to communicate with each other, likewise, we have different languages like C, C++, C#, Java, python, etc to communicate with the computers.
- The computer only understands binary language (the language of 0's and 1's) also called machine-understandable language or low-level language but the programs we are going to write are in a high-level language which is almost similar to human language.
- The main() is a standard function that you will always include in any program that you are going to create from now onwards.
- Note that the execution of the program starts from the main() function.

Introduction to Programming

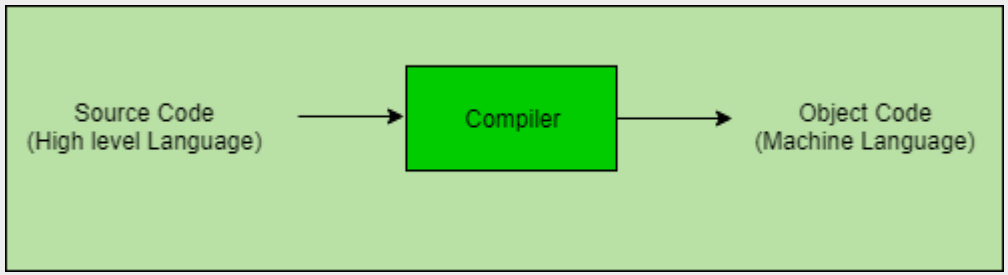
- **There are 3 types categories of language:**
- **Machine Language**
- **Assembly Language**
- **Higher level Language**
- **Machine Language:** It is a computer's natural language which can be directly understood by the system. It is directly executed by the CPU. A program written in 1's and 0's is called Machine language(binary code).
- It is directly understood by the processor so has faster execution time since the programs written in this language need not to be tanslated.
- It is very difficult to program since all the instructions are to be represented by 0s and 1s.
- It is time consuming and difficult to find error and to debug.

Introduction to Programming

- **Assembly Language:** Assembly language is a little easier than machine language. It uses more convenient numbers, symbols, and abbreviations to describe the huge strings of 1s and 0s, to make it both easier and more memorable to type in instructions.
- The programs are written in alphanumeric symbols, instead of 0’s and 1’s. This numeric symbols are called mnemonics like ADD, SUB, PRINTF, etc.
- The program is converted into machine code by assembler.
- It is not portable because every processor has its own assembly langauge.
- **Higher level Language:** This language uses English-like statements and symbols, and are independent of the type of computer you are using. This language is very easy to understand and speedy also. The programs can be written in English words. It is portable.
- The high level language is converted into machine language by one of the two different languages translator programs; **interpreter or compiler.**

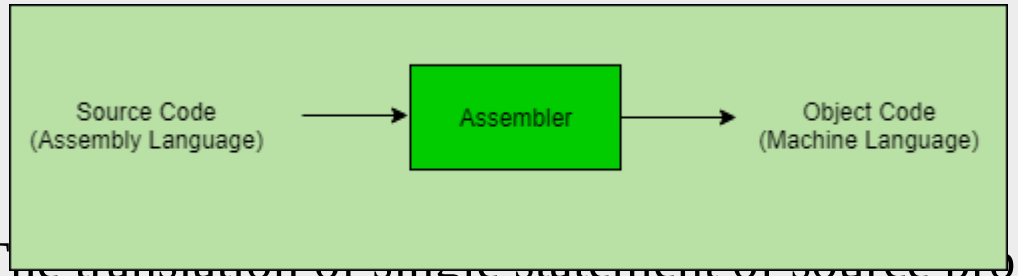
Introduction to Programming

- **Higher level Language:** The programs are written in high level languages like C, C++, Java, Python etc. and are called **source code.**
- These source code cannot be executed directly by the computer and must be converted into machine language to be executed.
- Hence, a special translator system software is used to translate the program written in high-level language into machine code is called Language Processor and the program after translated into machine code.
- **Compiler:** The language processor that reads the complete source program written in high level language as a whole in one go and translates it into an equivalent program in machine language is called as a **Compiler**.
- Example: C, C++, C#, Java
- The compiler specifies the errors at the end of compilation with line numbers when there are any errors in the source code.



Introduction to Programming

- **Assembler:** The Assembler is used to translate the program written in Assembly language into machine code. The source program is a input of assembler that contains assembly language instructions. The output generated by assembler is the object code or machine code understandable by the computer.



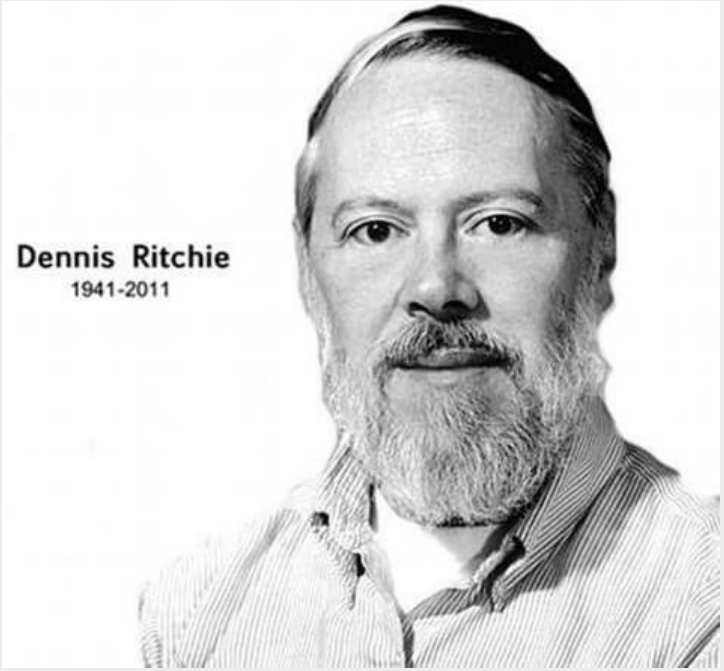
- **Interpreter:** The translation of single statement of source program into machine code is done by language processor and executes it immediately before moving on to the next line is called an interpreter. If there is an error in the statement, the interpreter terminates its translating process at that statement and displays an error message. The interpreter moves on to the next line for execution only after removal of the error.
- Example: Perl, Python.
-

Introduction to Programming

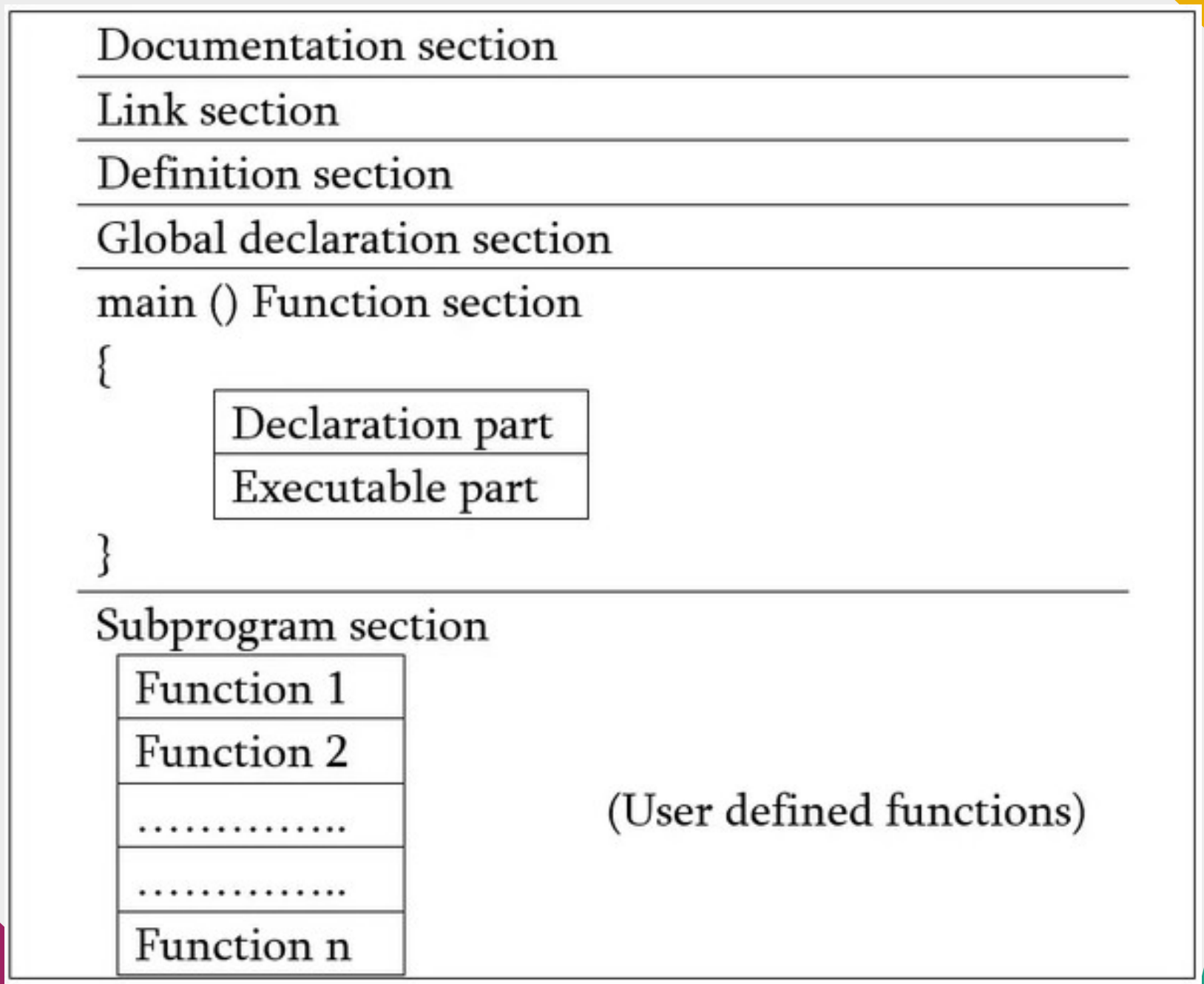
COMPILER	INTERPRETER
A compiler is a program which coverts the entire source code of a programming language into executable machine code for a CPU.	interpreter takes a source program and runs it line by line, translating each line as it comes to it.
Compiler takes large amount of time to analyze the entire source code but the overall execution time of the program is comparatively faster.	Interpreter takes less amount of time to analyze the source code but the overall execution time of the program is slower.
Compiler generates the error message only after scanning the whole program, so debugging is comparatively hard as the error can be present any where in the program.	Its Debugging is easier as it continues translating the program until the error is met

Introduction to C

- C is a general-purpose programming language that is extremely popular, simple and flexible.
- C is a procedural programming language.
- It was initially developed by Dennis Ritchie in the year 1972.
- It was mainly developed as a system programming language to write an operating system.
- C is a base for the programming.



Structure of Program



Structure of Program

- **Documentation section:** The Documentation section consists of a set of comment lines. Like giving the name of the program, the author and other details, which the programmer would like to use later.
- **Link section:** The link section provides instruction to the compiler to link the header files or functions from the system library such as using the #include<stdio.h>.
- **Definition section:** The definition section defines all symbolic constants such by using the #define directive.
- **Global declaration section:** There are some variables that are used in more than one function, such variables are called global variables.
- In C there are two types of variable declaration,
  - **Local variable declaration:** Variables that are declared inside the main function.
  - **Global variable declaration:** Variables that are declared outside the main function.

Structure of Program

- **Main function section:** Every C-program should have one main() function.
- **This section contains two parts:**
  - **Declaration part:** The declaration part declares all the variables used in the executable part.
  - **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.
- **Sub-program section:** If the program is a multi-function program, then the subprogram section contains all user-defined functions that are called in the main() function.

Structure of Program

```
//comments (if any)
#include <header files here>
int main()
{
   …
        logic
   …
}
```

Even though the comments are optional, it is **definitely recommend to have at least a single-line comment describing what the program does.**

The main() function is the first thing that runs. Everything that needs to be executed has to be somehow inside the main().

Structure of Program

```
#include <stdio.h>

int main() {
   /* my first program in C */
   printf("Hello, World! \n");

   return 0;
}
```

Structure of Program

- The first line of the program #include <stdio.h> is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
- The next line int main() is the main function where the program execution begins.
- The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
- The next line printf(...) is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
- The next line return 0; terminates the main() function and returns the value 0.

Structure of Program

- The first line of the program #include <stdio.h> is a preprocessor command, which tells a C compiler to include stdio.h file before going to actual compilation.
- The next line int main() is the main function where the program execution begins.
- The next line /*...*/ will be ignored by the compiler and it has been put to add additional comments in the program. So such lines are called comments in the program.
- The next line printf(...) is another function available in C which causes the message "Hello, World!" to be displayed on the screen.
- The next line return 0; terminates the main() function and returns the value 0.

Structure of Program

BASIC STRUCTURE OF A 'C' PROGRAM:

Documentation section **[Used for Comments]**
Link section
Definition section
Global declaration section **[Variable used in more than one function]**
main() **{** **Declaration part** **Executable part** **}**
Subprogram section **[User-defined Function]** **Function1** **Function 2** **:** **:** **Function n**

Example:

```
//Sample Prog Created by:Bsource

#include<stdio.h>
#include<conio.h>

void fun();

int a=10;

void main()
{
clrscr();
printf("a value inside main(): %d",a);
fun();
}

void fun()
{
printf("\na value inside fun(): %d",a);
}
```

Variables

- Variables – used in computer programming to store specific values within a program. In other words, it is used to store some form of data.
- A variable is a name of the memory location.
- Different types of variables require different amounts of memory.
- **Syntax to declare a variable:**
  - **type variable_list;**
- **The example of declaring the variable:**
  - int a;
  - float b;
  - char c;
- Here, a, b, c are variables. The int, float, char are the data types.

Variables

- We can also provide values while declaring the variables as given below:
  - int a=10,b=20;//declaring 2 variable of integer type
  - float f=20.8;
  - char c='A';
- **Rules for defining variables:**
  - A variable can have alphabets, digits, and underscore.
  - A variable name can start with the alphabet, and underscore only. It can't start with a digit.
  - Variables are case sensitive.
  - No special symbols are allowed other than underscore.
  - No whitespace is allowed within the variable name.
  - A variable name must not be any reserved word or keyword, e.g. int, float, etc.
- **Valid variable names:** int a; **OR** int _ab; **OR** int a30;
- **Invalid variable names:** int 2; **OR** int hello world; **OR** int long;

Variables

- **Types of variables in c:**
  - local variable
  - global variable
  - static variable
  - automatic variable
  - external variable
- **Local Variable:** A variable that is declared inside the function or block is called a local variable. It must be declared at the start of the block.
- **Example:**

```
void function1()
{
    int x=10;//local variable
}
```

Variables

- **Global Variable:** A variable that is declared outside the function or block is called a global variable. Any function can change the value of the global variable. It is available to all the functions.
- It must be declared at the start of the block.
- **Example:**

```
int value=20;//global variable

void function1()

{

   int x=10;//local variable

}
```

Variables

- **Static Variable:** A variable that is declared with the static keyword is called static variable.
- It retains its value between multiple function calls.
- **Example:**

```
void function1()
{
    int x=10;//local variable
    static int y=10;//static variable
    x=x+1;
    y=y+1;
    printf("%d,%d",x,y);
}
```

If you call this function many times, the local variable will print the same value for each function call, e.g, 11,11,11 and so on. But the static variable will print the incremented value in each function call, e.g. 11, 12, 13 and so on.

Variables

- **External Variable:**
- We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use extern keyword.
- **myfile.h**

```
extern int x=10;//external variable (also global)
```

- **program1.c**

```
#include "myfile.h"
#include<stdio.h>
void printValue()
{
      printf("Global variable: %d", x);
}
```

Variables

- **Automatic Variable:**
- All variables in C that are declared inside the block, are automatic variables by default. We can explicitly declare an automatic variable using auto keyword.
- **Example:**

```
void main()
{
    int x=10;//local variable (also automatic)
    auto int y=20;//automatic variable
}
```

Variables

- **External Variable:**
- We can share a variable in multiple C source files by using an external variable. To declare an external variable, you need to use extern keyword.
- **myfile.h**

```
extern int x=10;//external variable (also global)
```

- **program1.c**

```
#include "myfile.h"
#include<stdio.h>
void printValue()
{
      printf("Global variable: %d", x);
}
```

Variables

- **Note: Global Variables and extern**
- A **global variable** is a variable that is defined outside all functions and available to all functions.
- These variables are unaffected by scopes and are always available, which means that a global variable exists until the program ends.
- It is possible to create a global variable in one file and access it from another file. In order to do this, the variable must be declared in both files, but the keyword **extern** must precede the "second" declaration.