Chapter -1
Introduction to Structured Programming

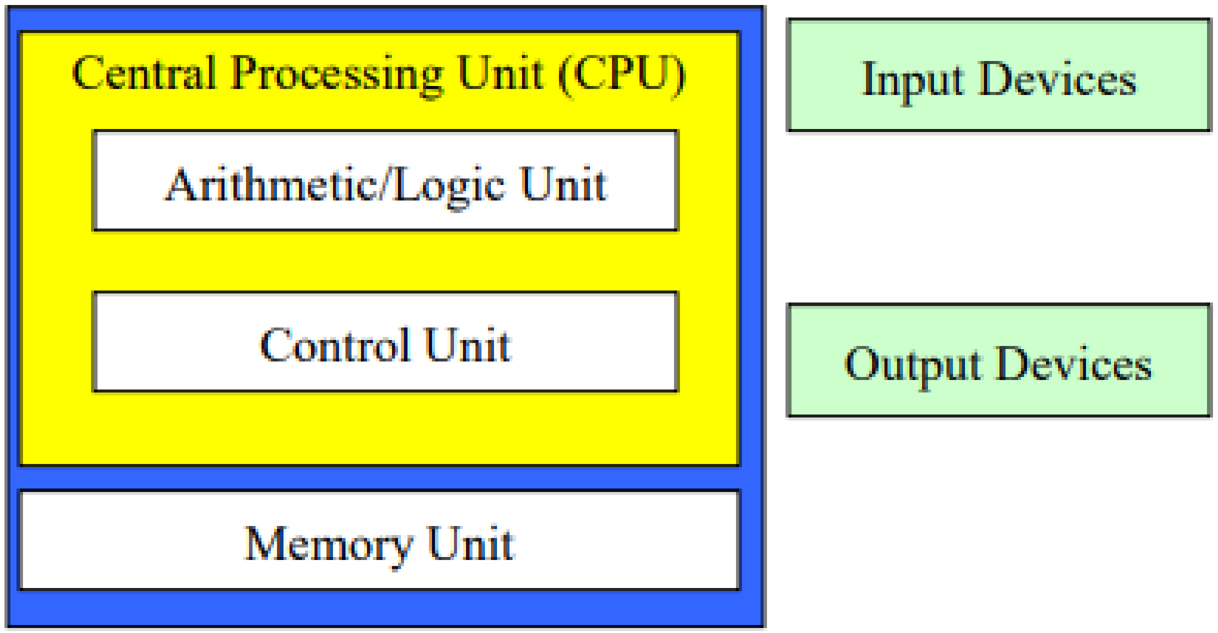
Course Code: CIS 115 & 115 L
Course Title: Structured Programming
Course Teacher: Md. Faruk Hosen

Prepared by: ABK Bhuiyan (Jehad)

Overview

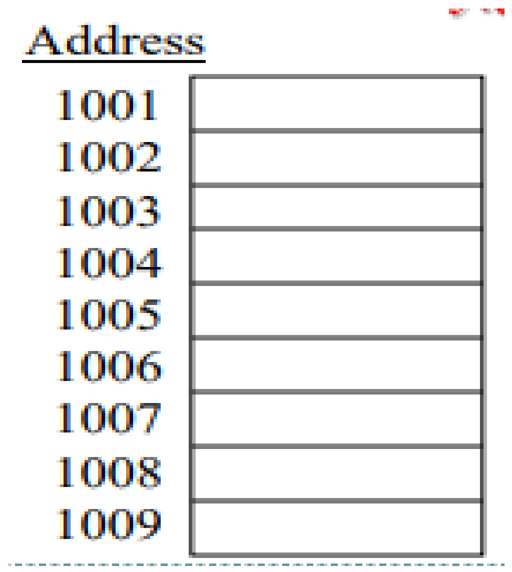
- Course Info
- Introduction to computer hardware
- Introduction to computer programming
- Introduction to c programming
- The code life cycle

Basic Computer Components



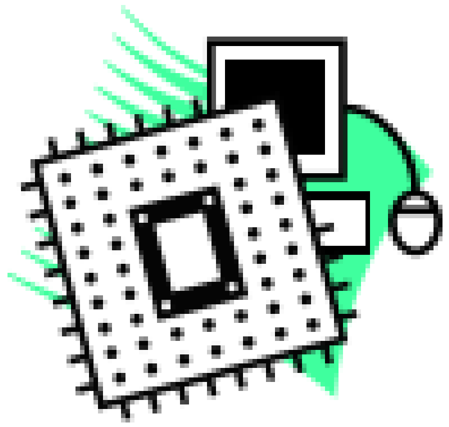
Memory Unit

- Ordered sequence of cells or locations
- Stores instructions and data in binary
- Types of memory
  - Read-Only Memory (ROM)
  - Random Access Memory (RAM)



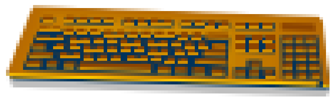
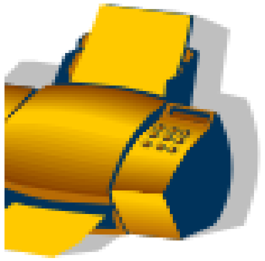
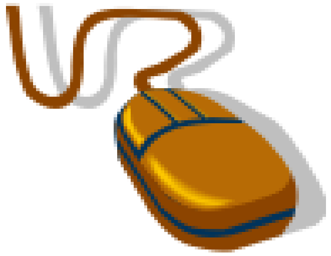
Central Processing Unit

- Executes stored instructions
- Arithmetic/Logic Unit (ALU)
  - Performs arithmetic and logical operations
- Control Unit
  - Controls the other components
  - Guarantees instructions are executed in sequence



Input & Output Devices

- Interaction with humans
- Gathers data (Input)
- Displays results (Output)

What is Programming?

- Planning or scheduling the performance of a task.
- Consciously thinking about each step
- Example: Accelerating in a car
  - 1. Move right foot to gas pedal
  - 2. Apply pressure to gas pedal with right foot
  - 3. If speed is too high, apply less pressure.
  - 4. If speed is too low, apply more pressure.

What is Computer Programming?

- Planning or scheduling a sequence of steps for a computer to follow to perform a task.
- Basically, telling a computer what to do and how to do it.
- A program:
  - A sequence of steps to be performed by a computer.
  - Expressed in a computer language.

Computer Languages

- A set of
  - Symbols (punctuation),
  - Special words or keywords (vocabulary),
  - And rules (grammar)

Used to construct a computer program.

Different Types of Languages

- Machine Language
- Assembly Language
- High-Level Language

Address	Contents
2034	10010110
2035	11101010
2036	00010010
2037	10101010
2038	10010110
2039	11101010
2040	11111111
2041	01010101
2042	10101101

Mnemonic	Instruction
ADD	10010011

Sample Program

```
MUL X,10
ADD X,Y
STO Z,20
SUB X,Z
```

Unstructured Programming approach

In this approach programmer put all their code in one place or say in one program. There is no harm as such but when line of code increases, situation starts getting unmanageable. A programmer can face so many problems like:

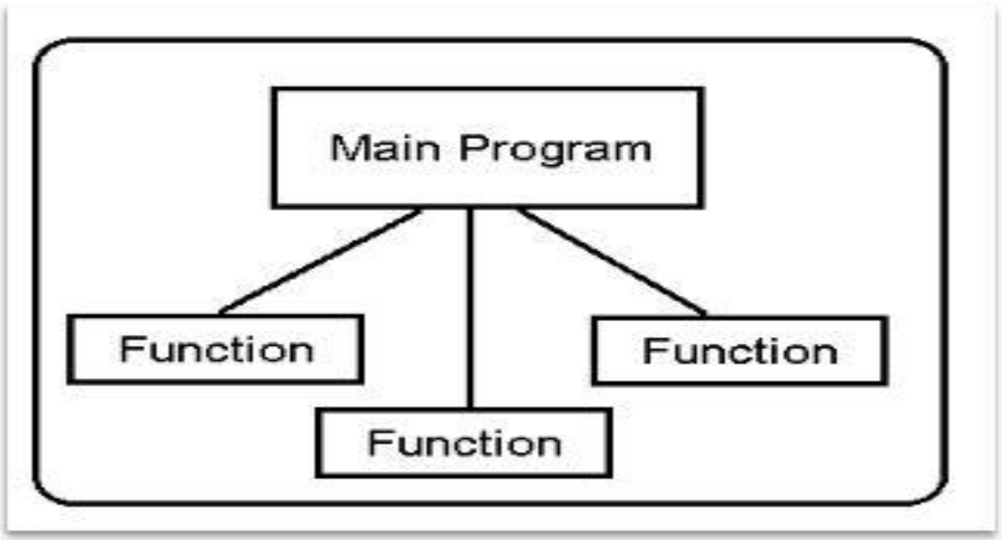
- Duplicity and redundancy of code
- Code maintenance problem
- Readability and many more

This approach is totally useless if working on a large project.

Structured Programming approach-1

- To overcome the previous situation it was needed to come up with a structured approach to make some code separation keeping reusability in mind.
- So the next approach was **Structured Programming or Procedure Oriented Programming (POP) approach**.
- In this approach a new idea came up and a set of execution code was keep in a place and it was called function.

Structured Programming approach-2

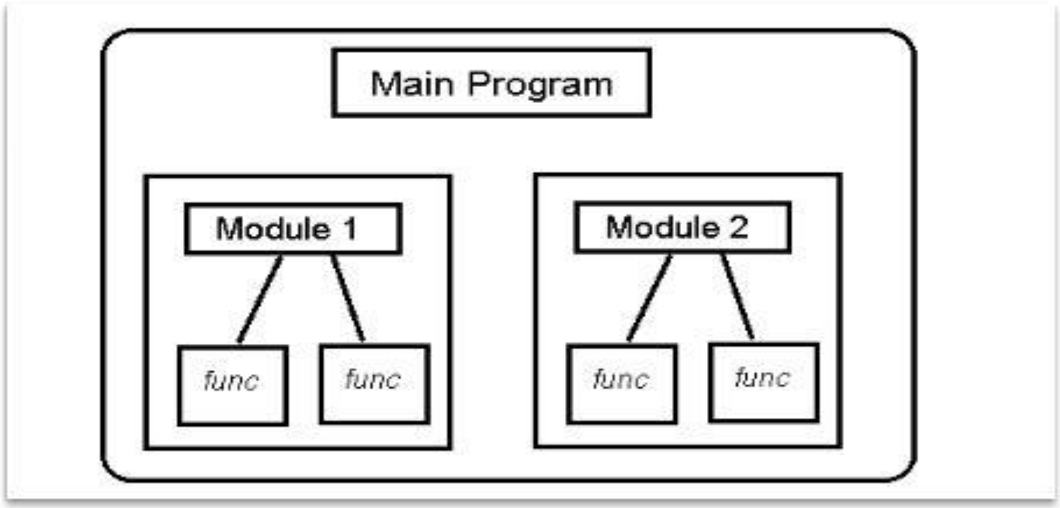


Program was divided into separate functions which was easier to manage and re-use as compared to unstructured approach. This was very helpful from reusability point of view but still all code was in same place so still there was a problem to manage them.

Modular Programming approach-1

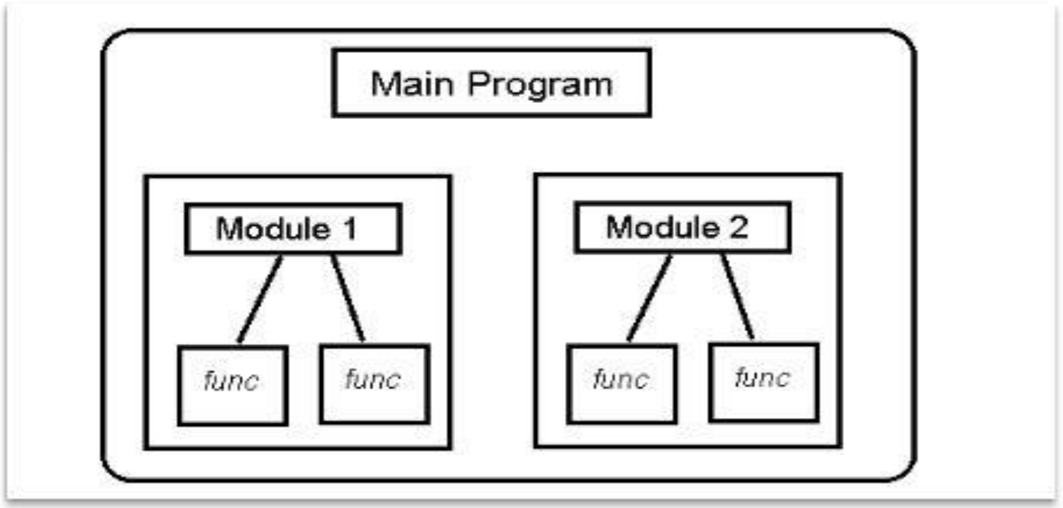
- A new idea came into light and common functionalities was grouped into modules.
- This approach helped in dividing programs into small parts called modules. This approach was really helpful. All was good except one.
- All modules can have their own data and also they were capable to maintain their internal state. But this came as a problem because there can be only one state per module so each module exists at most once in a program.

Modular Programming approach-2



All the above approach was good enough in their own scope but still there was a need for something else which may be really helpful in making programmer's life easier.

Modular Programming approach-2



All the above approach was good enough in their own scope but still there was a need for something else which may be really helpful in making programmer's life easier.

Structure Programming Language

Some of the languages initially used for structured programming include: BASIC, ALGOL, Pascal, PL/I and Ada.

The most popular Structure Programming Language in the world is ‘C’.

A Brief History of C

- C is a general-purpose language which has been closely associated with the UNIX operating system for which it was developed - since the system and most of the programs that run it are written in C.

- Many of the important ideas of C stem from the language BCPL, developed by Martin Richards. The influence of BCPL on C proceeded indirectly through the language **B**, which was written by Ken Thompson in 1970 at Bell Labs, for the first UNIX system on a DEC PDP-7. **BCPL** and **B** are "type less" languages whereas C provides a variety of data types.

A Brief History of C

- In 1972 Dennis Ritchie at Bell Labs writes C and in 1978 the publication of The C Programming Language by Kernighan & Ritchie caused a revolution in the computing world.

- In 1983, the American National Standards Institute (ANSI) established a committee to provide a modern, comprehensive definition of C. The resulting definition, the ANSI standard, or "ANSI C", was completed late 1988.

Characteristics of ‘C’

The Unix operating system and virtually all Unix applications are written in the C language. C has now become a widely used professional language for various reasons such as:

- Easy to learn
- Structured language
- It produces efficient programs.
- It can handle low-level activities.
- It can be compiled on a variety of computers.

Facts about ‘C’

- C was invented to write an operating system called UNIX.
- C is a successor of B language which was introduced around 1970
- The language was formalized in 1988 by the American National Standard Institute (ANSI).
- By 1973 UNIX OS almost totally written in C.
- Today C is the most widely used **System Programming Language**.
- Most of the state of the art software have been implemented using C

System Programming Language in 'C'

- A system programming language usually refers to a programming language used for system programming; such languages are designed for writing system software, which usually requires different development approaches when compared with application software.
- System software is computer software designed to operate and control the computer hardware, and to provide a platform for running application software. System software includes software categories such as operating systems, utility software, device drivers, compilers, and linkers.

Why use ‘C’?

C has been used successfully for every type of programming problem imaginable from operating systems to spreadsheets to expert systems - and efficient **compilers** are available for machines ranging in power from the **Apple** Macintosh to the **Cray** supercomputers.

The largest measure of C's success seems to be based on purely practical considerations:

- the portability of the compiler;
- the standard library concept;
- a powerful and varied repertoire of operators;
- an elegant syntax;
- ready access to the hardware when needed;
- and the ease with which applications can be optimized by hand-coding isolated procedures

Examples of the use of ‘C’

C was initially used for system development work, in particular the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as code written in assembly language. Some examples of the use of C might be:

- Operating Systems
- Language Compilers
- Assemblers
- Text Editors
- Print Spoolers
- Network Drivers
- Modern Programs
- Data Bases
- Language Interpreters
- Utilities

High Level Language

- A high-level language is one which is understandable by us humans.
- It contains words and phrases from the English (or other) language.
- A program written in high-level language is called a **source code**.

Machine Level Language

- But a computer does not understand high-level language.

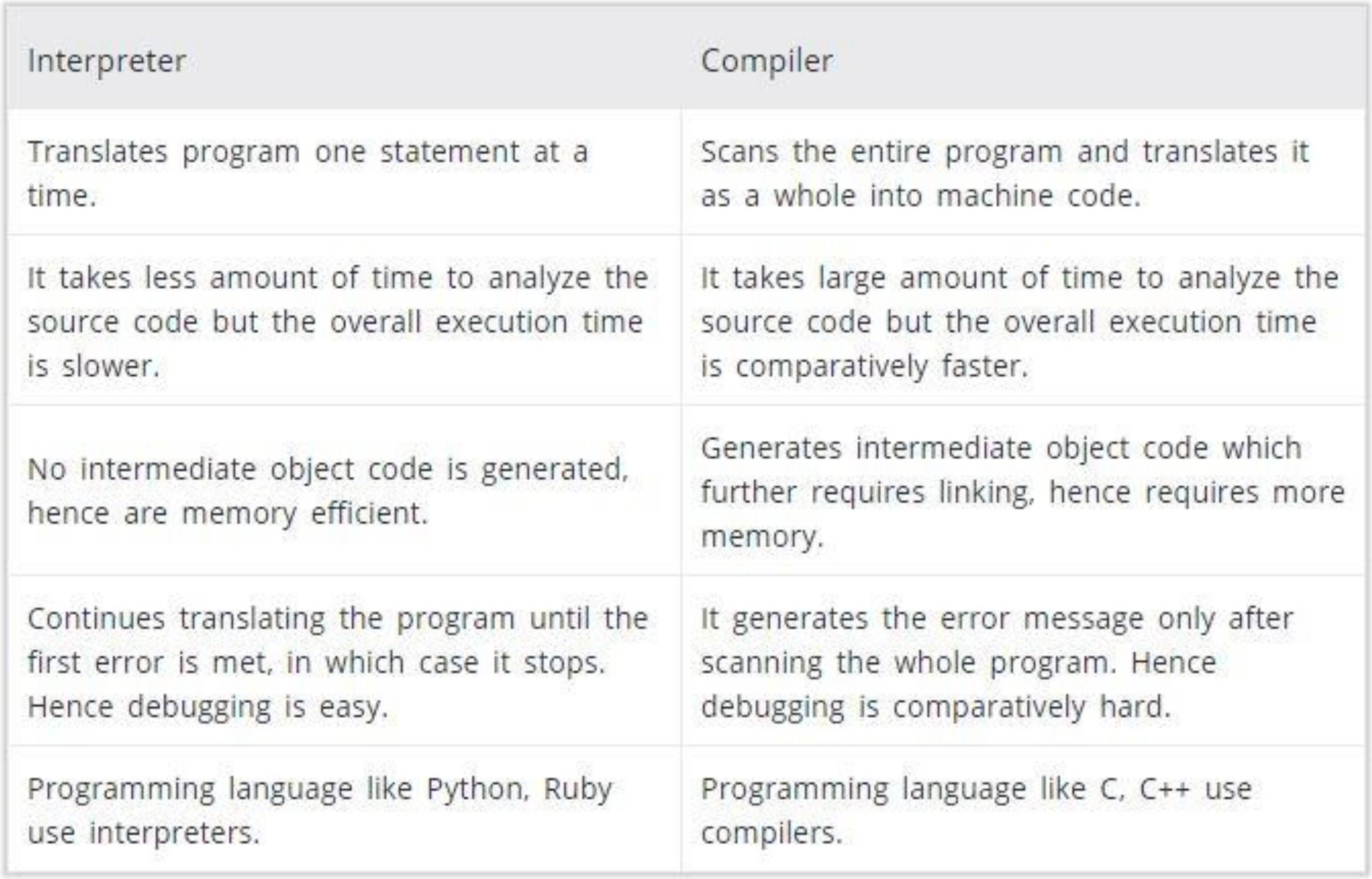
- It only understands program written in **0's and 1's** in binary, called the **machine code**.

Source Code to Machine Code

- We need to **convert the source code into machine code** and this is accomplished by **compilers and interpreters**.

- Hence, a **compiler or an interpreter** is a program that converts program written in **high-level language into machine code** understood by the computer.

Interpreter versus Compiler



Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
It takes less amount of time to analyze the source code but the overall execution time is slower.	It takes large amount of time to analyze the source code but the overall execution time is comparatively faster.
No intermediate object code is generated, hence are memory efficient.	Generates intermediate object code which further requires linking, hence requires more memory.
Continues translating the program until the first error is met, in which case it stops. Hence debugging is easy.	It generates the error message only after scanning the whole program. Hence debugging is comparatively hard.
Programming language like Python, Ruby use interpreters.	Programming language like C, C++ use compilers.

Working of compiler and interpreter

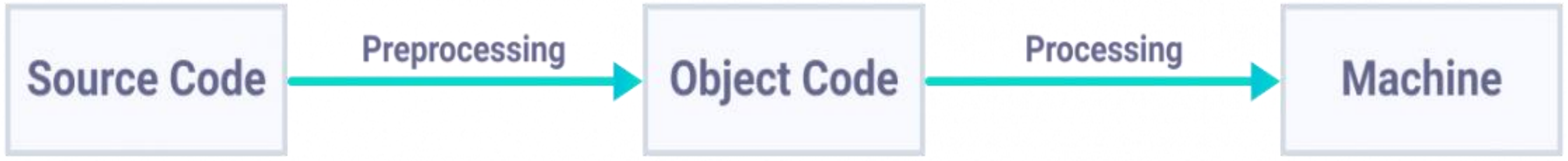


Figure: Compiler

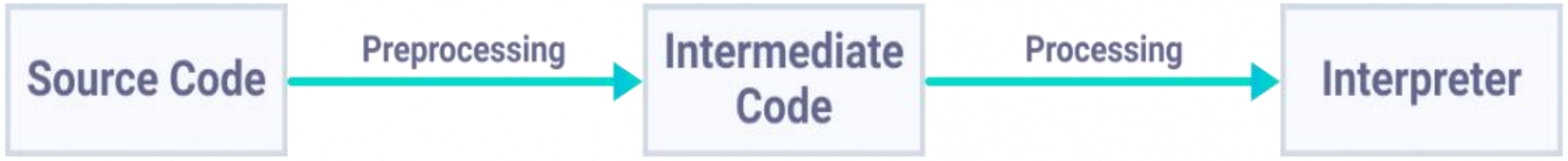
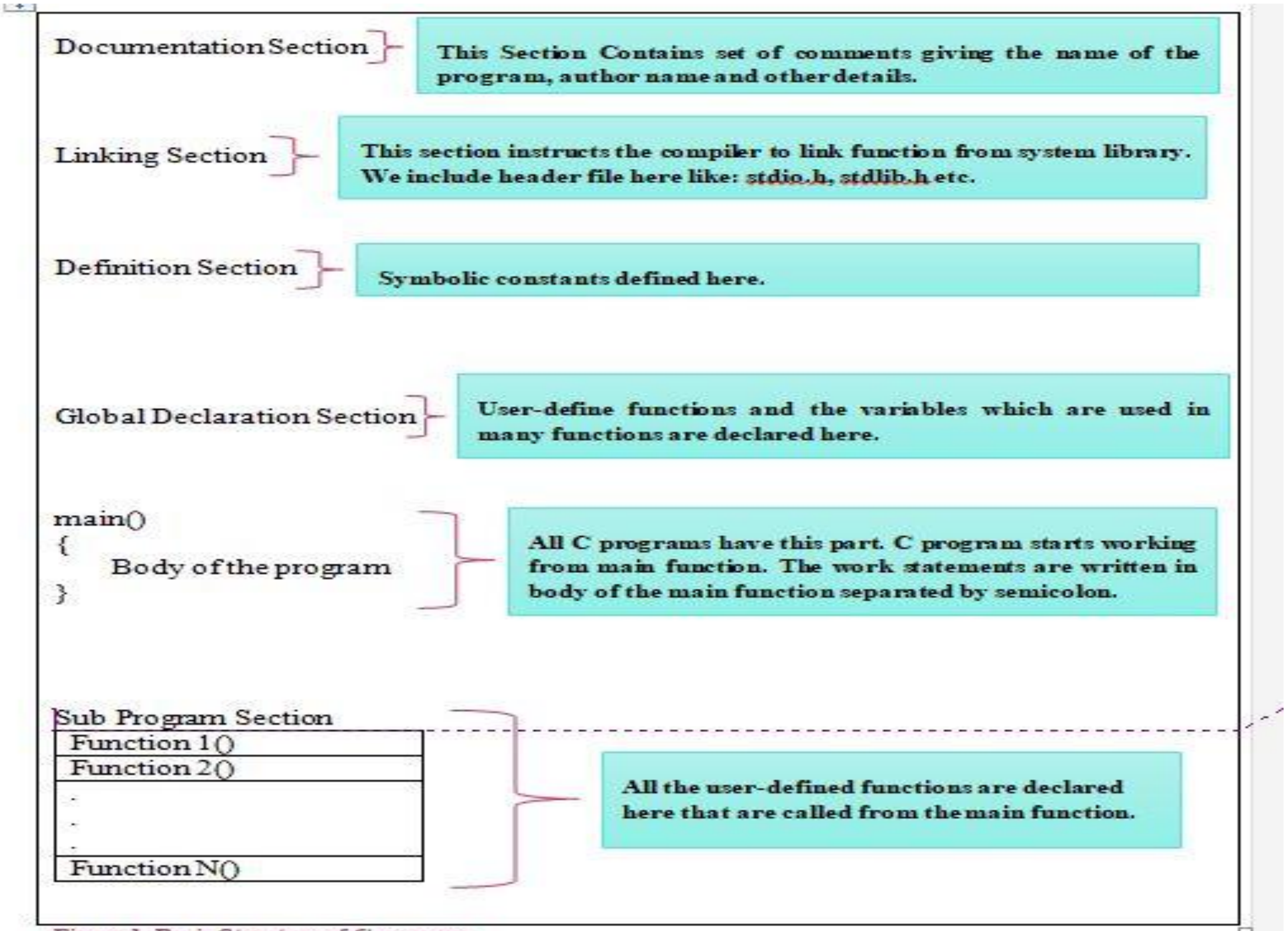


Figure: Interpreter

Basic Structure of a C Program



Documentation Section
This Section Contains set of comments giving the name of the program, author name and other details.
Linking Section
This section instructs the compiler to link function from system library. We include header file here like: stdio.h, stdlib.h etc.
Definition Section
Symbolic constants defined here.
Global Declaration Section
User-define functions and the variables which are used in many functions are declared here.
main()
{
Body of the program
}
All C programs have this part. C program starts working from main function. The work statements are written in body of the main function separated by semicolon.
Sub Program Section
Function 1()
Function 2()
Function N()
All the user-defined functions are declared here that are called from the main function.

Figure 1: Basic Structure of C program

Simple Program in C

If we look at the following example the structure will be more understandable:

```
/* A program to print "Hello World!"
Author: Nayeema Rahman
email: nayeema@daffodil.ac */

#include <stdio.h>

main() {
        printf("Hello World!");
         }
```

This is the Documentation section, contains program information and author information

Linking section links stdio.h (Standard Input output) header file with this program

Main function contains the statement for printing message Hello world!

Basic Terminology

As we learned earlier C is a structured program and it modularizes the works in different functions. We write these functions for our purpose and many others are stored in the library.

Library functions are grouped according to their category and stored in different files known as header file.

PREPROCESSOR DIRECTIVE

For accessing these functions we have to link that file with our program and tell the compiler about the files to be accessed.

For instructing the compiler we need to use the pre-processor directive **#include**. We write it in linking section. As we see the example in previous section stdio.h was included. Format of writing:

- **#include<filename>**.

Here file name is the header file name.

SL	Directive & Description
1	#define Substitutes a preprocessor macro.
2	#include Inserts a particular header from another file.
3	#undef Undefines a preprocessor macro.
4	#ifdef Returns true if this macro is defined.
5	#ifndef Returns true if this macro is not defined.
6	#if Tests if a compile time condition is true.
7	#else The alternative for #if.
8	#elif #else and #if in one statement.
9	#endif Ends preprocessor conditional.
10	#error Prints error message on stderr.
11	#pragma Issues special commands to the compiler, using a standardized method.

THE MAIN FUNCTION

The main function is common part of every C program. Execution of a C program starts from the main function. The main is a user-defined function as the body of this function is defined by the user. Some common forms of main function:

- void main()
- main()
- int main(void)
- void main(void)

Explanation of Common Forms

- **void main():** Here empty parenthesis also mean the same as previous that is no arguments passed to it and the return type is specify by a keyword void which means no information is returned from this function.
- **main():** Empty parenthesis indicates that there is no arguments passed to the main function. And there is no return type before main, so it will not return any value.
- **int main(void):** The void key word in parenthesis says that there will be no arguments passed to it. Before main the keyword int specify that the function will return an integer type value.
- **void main(void):** This means same as the void main().

Anyone can use any format for declaring main function. But there is some compiler that does not support all the formats but all of these are standard.

Execution of a C program

For executing a program in C, need to follow a series of steps.

These steps are:

- **Create the Program**
- **Compile The Program**
- **Linking the program with C library**
- **Executing Program**

CREATING PROGRAM

- Program must be written into a file. The file name consists of alphabet, letter or digit. **The extension of the file must be c, example: test.c.**

- The file can be created with the help of **a text editor**. There are also many C program editor can be found as like **Turbo C, Code Blocks and Visual C** etc. They are featured editor, help the programmer to write code, compile code as they have integrated C compiler and also debugging facility to find and solve logical errors.

COMPILE AND LINKING

- As we know computer does not understand other than machine codes. So we need to generate the machine code of program we have written.
- **Compiling** is the process converting the program to machine code and **Compiler** is a computer program that takes a program as input and generates machine code. Example: **GNU GCC is a c compiler.**
- As the most of C editors have compiler integrated with them so we need not worry about to configure and install compiler other than the editor.
- Linking is the process of putting together other program files and functions that are required by the program. For example: the printf() function requires stdio.h (Standard Input Output) library of the system and linked with the program. The modern editors do this work for us. If we want to do it by ourselves, then the command is:

```
cc filename -lm
```

EXECUTING PROGRAM

- After linking and compiling we find and object file in the program directory. We can execute the program through this file. From command line we can run as:
- Step 1: Open the terminal
- Step 2: Execute the using the command:

  gcc fileName.c -o fileName

  Example: gcc test.c -o test
- Step 3: Aster executing we will find filaName.exe file. We can run it using the command

  "./filename".

  Example:./test.

If we use any of the modern editors as mentioned earlier than we can do all of that by clicking the button build and run.

INSTALLING COMPILER

- As we already know that if we install any editor **Code blocks or Turbo C** we need not install compiler as they have compiler integrated with them.
- For installing **Code block** go to the code blocks official site and download the editor from the link (**http://www.codeblocks.org/downloads**).

Install as like normal other software. It does not need to configure after install. We can write code and run.

- In case of **Turbo C** just download

(**http://turbo-c.apponic.com/**)

Unzip in **c drive** and **click trubo-c icon in tc/bin directory in c drive.**