

# Module Code: CSE401

## Session 1: C Programming

### Session Speaker:

**Prema Monish**

[premamonish.tlll@msruas.ac.in](mailto:premamonish.tlll@msruas.ac.in)



# Session Objectives

- To Learn the elements of C program
- To Learn steps in C programming
- To Learn basic structure of C program
- To Learn about Standard input/output Libraries



# Session Topics

- What is C programming?
- How C programming works?
- Basic Structure of C programming
- Getting started with C programming
- Standard I/O libraries



# Introduction

- **What is Computer Programming?**

Giving instructions /commanding to the computer & to interact with it we need language to communicate.

Different kinds of language –Middle level ,low level & high level language

E.g.: C, C++, Java, Python etc

- **Why Programming required?**

consider example of arranging 10 numbers in ascending order it is easy to do , but if it is 10,000 numbers it is difficult , long time required and errors may occur.

So to simplify we develop a program which does this task accurately and quickly .



# Introduction to C

- C is a general purpose computing programming language.
- C was invented and was first implemented by **Dennis Ritchie** with the Unix Operating System in 1972.
- C is often called a **Middle Level language** as it combines the elements of high-level languages with the functionalism of assembly language
- C is a **Structured Language**.



Dennis Ritchie



# Characteristics of C

- C will allow you to create any task you may have in mind
- *Robust Programming*
- *Portability*
- *Structured programming language*
- *Control the memory efficiently*
- *Various application*



# Usage of C

- C's primary use is for **System Programming**, including implementing **Operating Systems** and **Embedded System applications**. E.g. Windows, Unix ,Linux ,Rtlinux
- One consequence of C's wide acceptance and efficiency is that the **Compilers, Libraries, And Interpreters** of other higher-level languages are often implemented in C.
- Mobiles devices, Microwave oven, digital cameras etc are getting smarter due to combination of **Microprocessor, OS & C Program** embedded with it .



# Levels of Programming Languages

- Low Level Language-Machine language & Assembly language
- High Level Language
- Middle Level language





# Machine Languages

- **Machine language** program is a series of coded instructions i.e 0's and 1's
- They are normally specified using both binary codes
- Direct memory accessing
- Not human readable

E.g. 0100

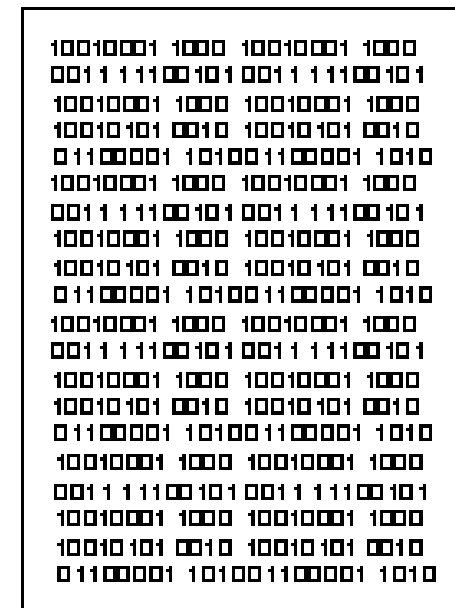
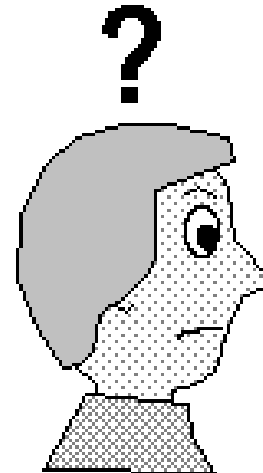
001101

00101      10001   10000

01110

111001

...



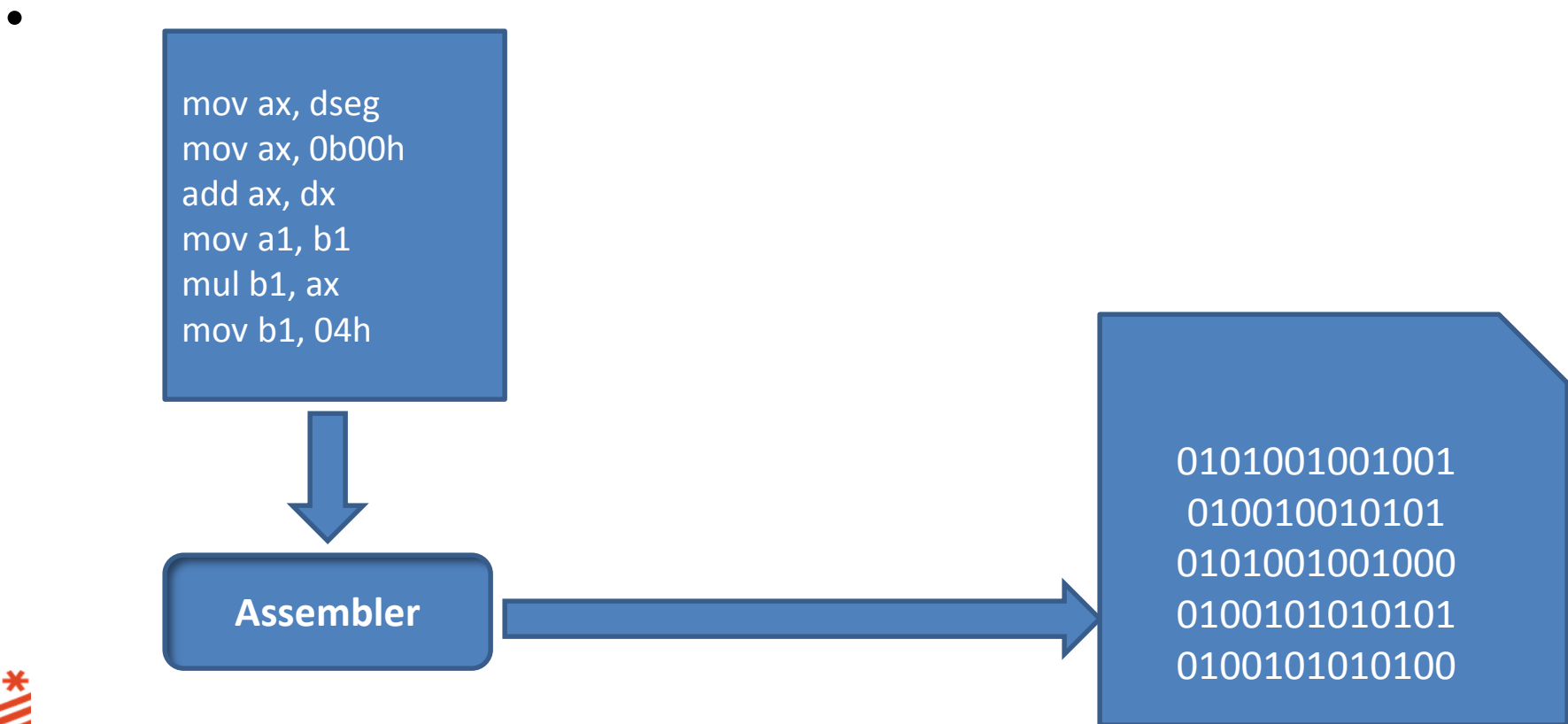
# Assembly Languages

- It implements a *symbolic representation* of the binary machine codes and other constants needed to program a particular CPU architecture
- A special program known as an *assembler* , translates the assembly language program from its symbolic format into the specific machine instructions of the computer system.
- Because a one to one correspondence still exists between each assembly language statement and a specific machine instruction, it is regarded as low level languages.



# Assembly Languages

- Instruction set is different for each computer processor, so it is not portable

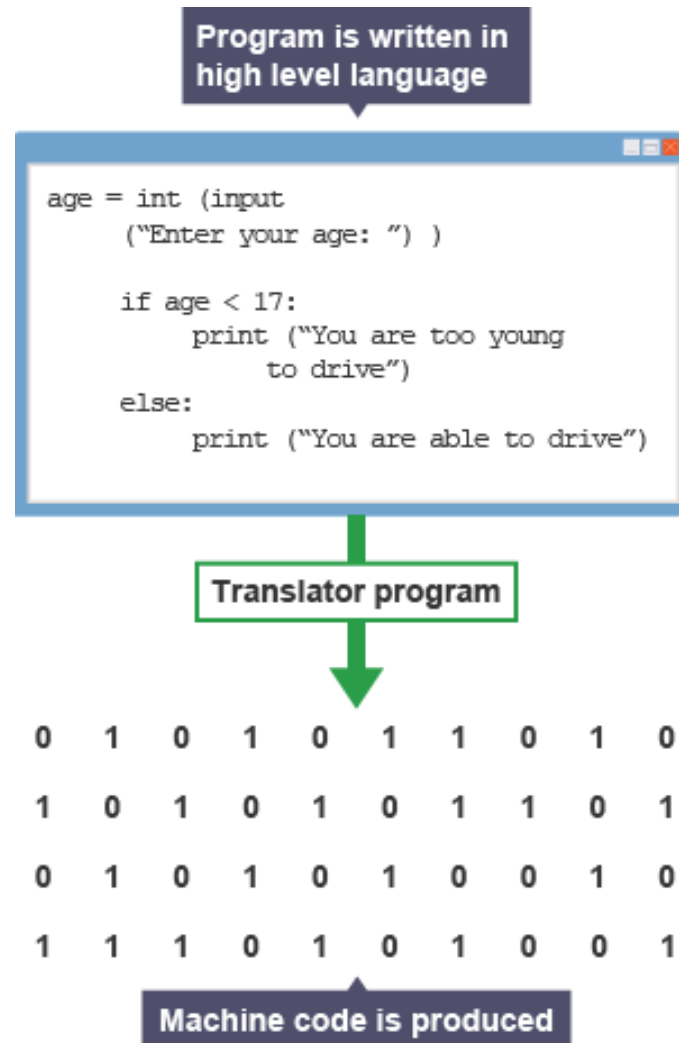


# High-Level Languages

- Higher Level Languages
  - Use traditional programming logic where the programming instructions tell the computer what to do and how to perform the required operations.
  - E.g. FORTRAN. Formula translation language , java, python, C#
  - One Fortran instruction or statement resulted in many different machine instructions being executed unlike assembly language



# High-Level Languages



# Middle Level Language

- It is combination of *High Level Language* and *Low Level Language*.
- E.g. C—Low level Language- using pointers memory accessed directly  
C-High Level Language-It is human understandable language



# How C Program Works ?

- There are four fundamental stages or processes in creation of any C program
  1. Editor
  2. Compiler /Interpreter
  3. Linker
  4. Execute



# Editor

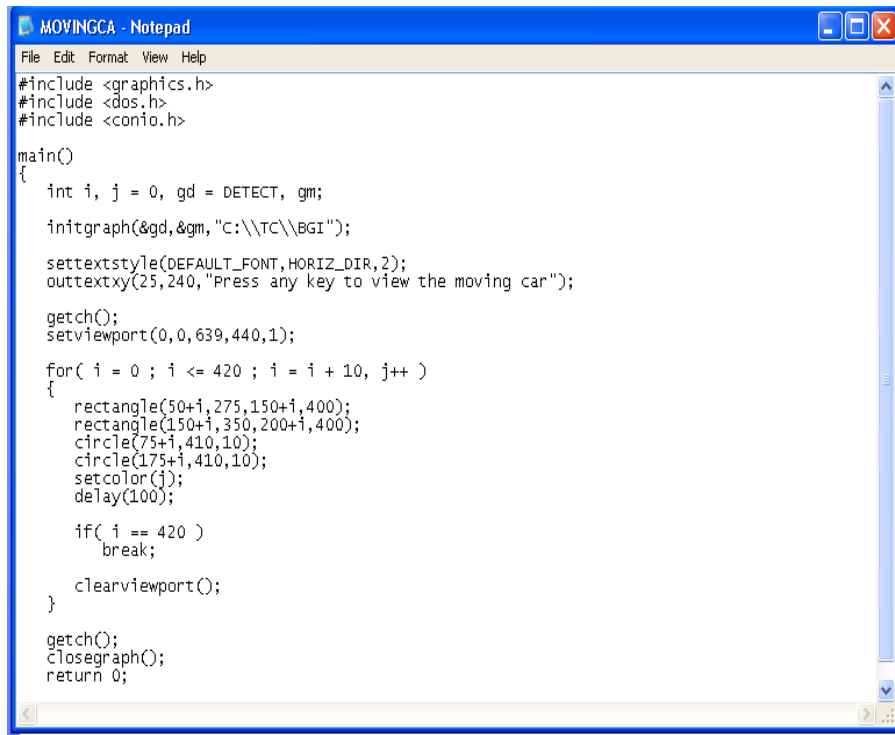
- Once program is written then you need to type it & instruct the machine to execute it.
- To type C program you need another program called **Editor**
- **General-purpose** text editor to create your source files, but the editor must store the code as plain text without any extra formatting data embedded in it.eg. Notepad
- **Integrated development environment (IDE)** – is compiler system with an editor included, it will provide a lot of features that make it easier to write and organize your source programs. E.g.Turbo C,Netbeans
- **Editor** output is source file with extension “.c”





# Editor

## General Purpose Text Editor



```
MOVINGCA - Notepad
File Edit Format View Help

#include <graphics.h>
#include <dos.h>
#include <conio.h>

main()
{
    int i, j = 0, gd = DETECT, gm;

    initgraph(&gd,&gm,"C:\\TC\\BGI");

    settextstyle(DEFAULT_FONT,HORIZ_DIR,2);
    outtextxy(25,240,"Press any key to view the moving car");

    getch();
    setviewport(0,0,639,440,1);

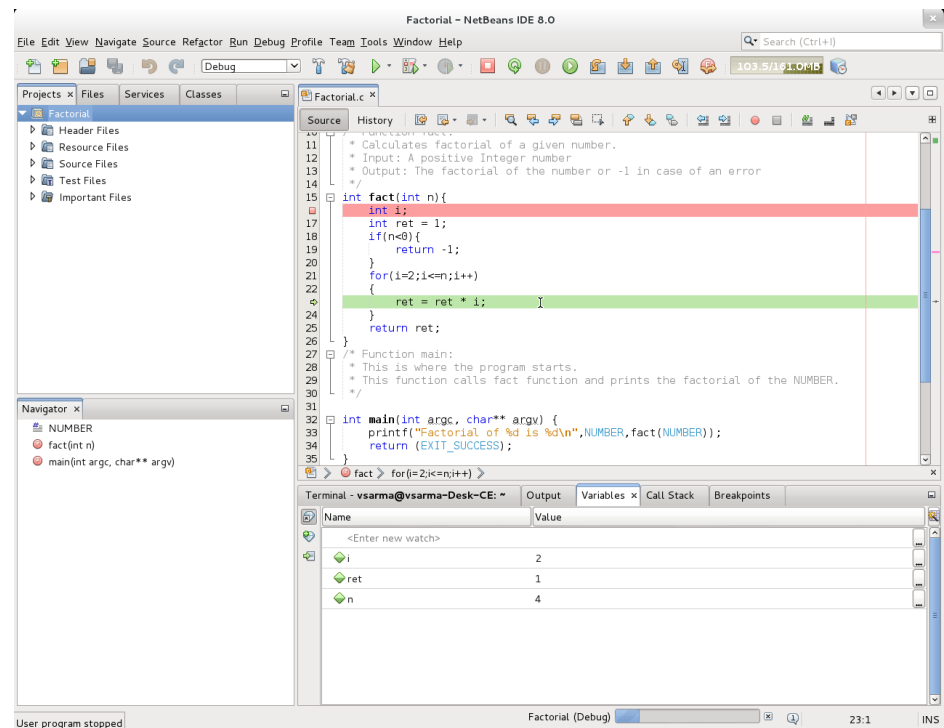
    for( i = 0 ; i <= 420 ; i = i + 10, j++ )
    {
        rectangle(50+i,275,150+i,400);
        rectangle(150+i,350,200+i,400);
        circle(75+i,410,10);
        circle(175+i,410,10);
        setcolor(j);
        delay(100);

        if( i == 420 )
            break;

        clearviewport();
    }

    getch();
    closegraph();
    return 0;
}
```

## IDE – eg.NetBeans



Linux/Unix –Vi editor, Vim editor, GNU Emacs editor(Gedit)

Microsoft Windows-freeware and shareware programming editors



# Compiler

- The Compiler searches all the errors of a program and lists them. If the program is error free then it converts the code of program into machine code at one go and then the program can be executed by separate commands.
- The **input** to this stage is the file you produce during your editing, which is usually referred to as a source file.
- The **output** from the compiler is known as object file and it is stored in files called *object files*, which usually have names with the extension .obj in the Microsoft Windows environment, or .o in the Linux/UNIX environment
- The compiler can detect several different kinds of errors during the translation process, and most of these will prevent the object file from being created.



# Compiler

- Integrated Development Environment (IDE)- is compiler system with an editor included in it.
- E.g. Turbo C, Turbo C++ & Microsoft C –Ms-Dos  
Visual C++ & Borland C++ - Windows  
GCC Compiler –Linux
- Compilation is a **two-stage process**.
- **Preprocessing phase**, during which your code may be modified or added to. E.g. Macros - Your source file can include preprocessing *macros*, which you use to add to or modify the C program statements.
- **Actual compilation** that generates the object code.



# Interpreter

- An interpreter takes *each line* of code and translates it into machine code before running it. It then goes to the next line, translates it, runs it, and so on until it has completed every line of code in your program.
- Advantage
  - Program will run even if it's not finished.
  - Easy to find errors.
- Disadvantage
  - Code has to be translated each time the program is run
  - Slows program down as it has to be translated



# Compiler Vs Interpreter

Compiler	Interpreter
Compiler Takes <b>Entire</b> program as input	Takes <b>Single</b> instruction as input
Intermediate Object Code is <b>Generated</b>	<b>No</b> Intermediate Object Code is <b>Generated</b>
Conditional Control Statements are Executes <b>faster</b>	Conditional Control Statements are Executes <b>slower</b>
<b>Memory Requirement : More</b> (Since Object Code is Generated)	<b>Memory Requirement is Less</b>
<b>Errors</b> are displayed after <b>entire program</b> is checked	<b>Errors</b> are displayed for <b>every instruction</b> interpreted (if any)
Program need not be <b>compiled</b> every time	Every time higher level program is converted into lower level program



# Linker

- The linker combines the object modules generated by the compiler from source code files, adds required code modules from the standard library supplied as part of C, and welds everything into an executable whole
- It detects & reports errors e.g. Missing or non Existed Library in program .
- If program consists of libraries which contain such as performing I/p & o/p, square root , date & time etc are included into program
- **Failure** during link phase means need to edit source code
- **Success** during link phase produce executable file
- E.g. “.exe” –Microsoft windows, Unix/Linux- no such extension
- Many IDEs have a *build in option*, which will compile and link your program in a single operation.

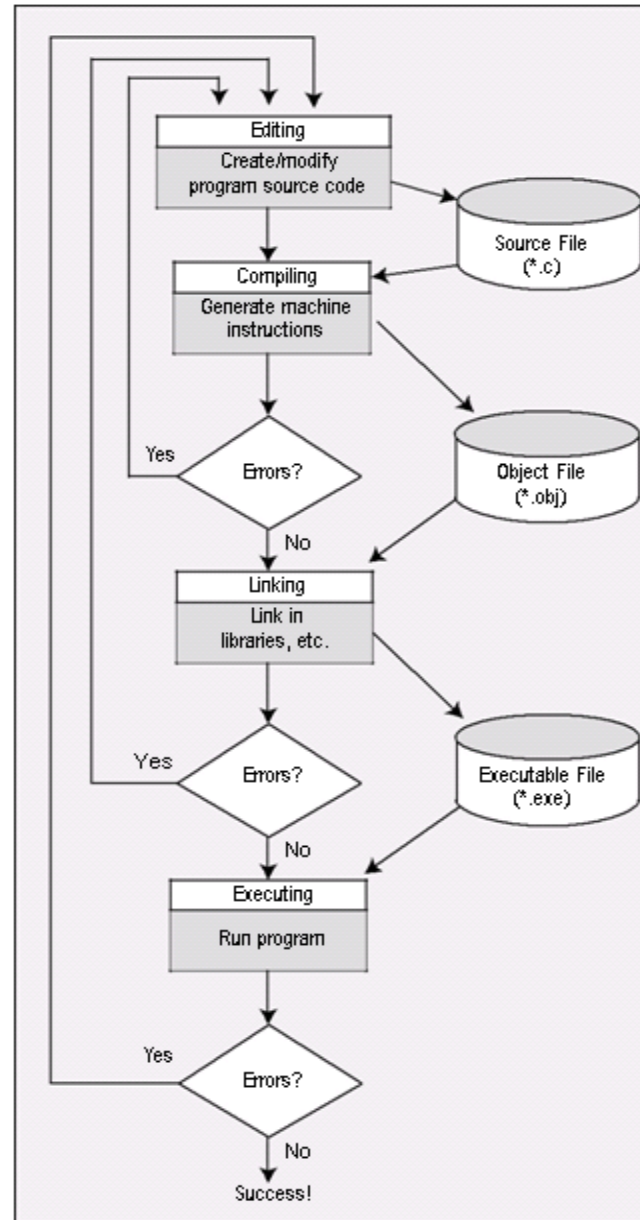


# Execute

- **Loader** -Before Execution phase program is loaded into memory
- The **Execution** stage is where you run your program, having completed all the previous processes successfully
- This stage can also generate a wide variety of error conditions that can include producing the wrong output
- In **Unix /Linux** you can just enter the name of the file that has been compiled and linked to execute the program.
- In most **IDEs**, you'll find an appropriate menu command that allows you to run or execute your compiled program
- In **Windows**, you can run the .exe file for your program as you would any other executable



# Creating And Executing A Program





# Structure of C Program

This indicates the beginning of a comment

This indicates the end of a comment

Everything following // on the line is a comment and therefore ignored by the compiler

The include directive includes the contents of an external file into your source file

Int indicates that main () returns an integer value

This brace indicates the start of the body of the main() function

The body of a function is all the code enclosed between the outermost braces

This brace indicates the end of the body of the main() function

```
/* Structure of C program */  
#include<stdio.h>  
int main()  
{  
    printf("Welcome to MSRUEAS\n");  
    .....  
    return 0;  
}  
// End main execution
```

Stdio.h is a standard header file required to use input/output functions such as printf()

This statement ends the execution of main() and returns control to the operating system



# General Structure of a C Program

- **Opening comment block**
  - Should always contain author's name, name of the file, and purpose of the program
- **Preprocessor directives**
  - include all header files for needed libraries
  - define any useful constants
- **Global variables**
- **Function prototypes**
  - Must be declared before the function is called
- **Main function**
  - Program execution begins at the start of main() and ends when it reaches the end of main()
- **Other function definitions**



# General Programming Rules

- C statements are always terminated with a semicolon “;”.
- Blank lines are ignored
- White space (blanks, tabs and new lines) must separate keywords from other things
- Comments –
  - All text enclosed within “/\* ----- \*/”
  - Text on the same line following “//”
  - Examples:
    - // This is a comment
    - /\* So is  
this. \*/



# Comment

- This isn't actually part of the program code, in that it isn't telling the computer to do anything. It's simply a *comment*, and it's there to remind you, or someone else reading your code, what the program does.
- **Multiline comment** -Anything between `/*` and `*/` is treated as a comment.

E.g. `/* Structure of C program*/`

- **Single line comment** -You can add a comment at the end of a line of code using a different notation, like this:

E.g. `printf("Beware the Ides of March!"); // print the line`

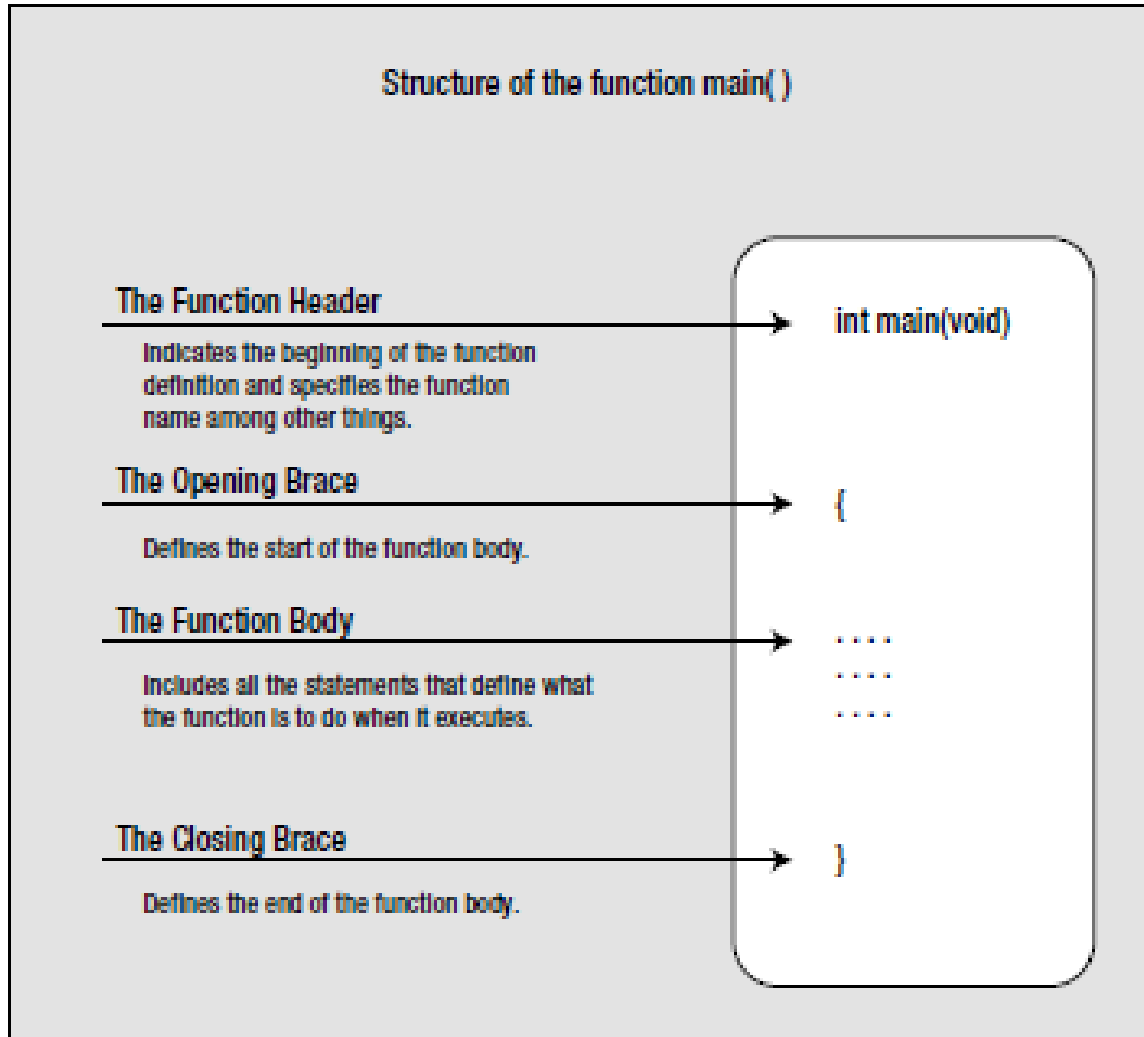


# Preprocessing Directives

- `#include <stdio.h> //` This is a preprocessor directive
- the program won't work without it.
- The symbol `#` indicates this is a *preprocessing directive*, which is an instruction to your compiler to do something before compiling the source code.
- The compiler handles these directives during an *initial preprocessing* phase before the compilation process starts.
- In this case, the compiler is instructed to “include” in your program the contents of the file with the name “stdio.h.” This file is called a *header file*, because it's usually included at the head of a program source file.
- In this case, because you're using the `printf()` function from the standard library, you have to include the `stdio.h` header file.
- All header files in C have file names with the extension *.h*



# Main function



# Operating systems

- An **operating system** is a program that controls the entire operation of a computer system.
- All input and output operations that are performed on a computer system are channeled through the operating system.
- It must also manage the computer system's resources and must handle the execution of programs. E.g. Linux, Microsoft Windows XP.



# Terms and Terminologies

- **Source code:** The text of a program that a user can read, commonly thought of as the program.
- **Object code:** Translation of the source code of a program into machine code , while the computer can read and execute directly.
- **Linker:** A program that links separately compiled modules into one program. The output of the linker is an executable program.





# Terms and Terminologies

- **Library :** The file containing the standard functions that your program can use. These functions include all I/O operations as well as other useful routines.
- **Compile time:** The time during which the program is being compiled
- **Run time :** The time during which the program is executing



# C Input/output Functions

- ANSI standard has defined many library functions for input and output in C language. Functions `printf()` and `scanf()` are the most commonly used to display out and take input respectively. Let us consider an example:
- **`printf()`** is a library function to display output which only works if `#include<stdio.h>` is included at the beginning
- Returns number of ***characters printed***

```
#include <stdio.h>    //This is needed to run printf() function.
int main()
{
    printf("Welcome to MSRUAS"); //displays the content inside quotation
    return 0;
}
```



# C Input/output Functions

- The **scanf()** function is used to take input from user. In this program, the user is asked a input and value is stored in variable a. Note the **'&'** sign before a. **&a** denotes the address of a and value is stored in that address.
- Returns, number of **characters read**.

```
#include <stdio.h>
int main()
{
    float a;
    printf("Enter value: ");
    scanf("%f", &a);
    printf("Value=%f",a);    //%f is used for floats instead of %d
    return 0;
}
```



# Format Specifiers

## Integer Data Type

%d : decimal integer

%ld : long integer

%u : unsigned integer

%x : hexadecimal

%o : octal

## Floating Point Data Type

%f : floating-point number

%lf : double

## Character Data Type

%c : single character

%s : string



# Format Specifier

```
#include<stdio.h>
main()
{
    int a=15,b;
    float c=2.35467;
    b = a / 2;
    printf("%d\n",b);
    printf("%3d\n",b);
    printf("%03d\n",b);
    printf(":%s:\n", "Hello, world!");
    printf(":%15s:\n", "Hello, world!");
    printf("%3.2f\n",c);
}
```

## Output:

```
7
 7
007
:Hello, world!:
:  Hello, world!:
2.35
```



# Format Specifier

- In the second printf statement we print the same decimal, but we use a width (%3d) to say that we want three digits (positions) reserved for the output.
- The result is that two “space characters” are placed before printing the character.
- In the third printf statement we say almost the same as the previous one. Print the output with a width of three digits, but fill the space with 0.
- %3.2f (print as a floating point at least 3 wide and a precision of 2 after decimal point)



# Escape Sequence

- Escape character – backslash \
- When encountering a backslash in a string, the compiler looks ahead at the next character and combines it with \ to form **escape sequence**



# Common Escape Sequences

<code>\b</code>	Backspace
<code>\f</code>	Form feed
<code>\n</code>	New line
<code>\r</code>	Carriage return
<code>\t</code>	Horizontal tab
<code>\"</code>	Double quote
<code>\'</code>	Single quote
<code>\\</code>	Backslash
<code>\v</code>	Vertical tab
<code>\a</code>	Alert(very small sound)
<code>\?</code>	Question mark





# The C Standard Library

- **C programs consist of pieces/modules called functions**
    - A programmer can create his own functions
      - Advantage: the programmer knows exactly how it works
      - Disadvantage: time consuming
    - Programmers will often use the C library functions
      - Use these as building blocks
    - Avoid re-inventing the wheel
      - If a pre-made function exists, generally best to use it rather than write your own
      - Library functions carefully written, efficient, and portable
- E.g-Stdio.h,



# The C Standard Library

- **<ctype.h>**- includes numerous standard library functions to handle characters E.g. tolower(c) ,toupper(c)
- **<math.h>**-C compiler support basic math functions like cos, sin, sqrt,..
- **<stdio.h>**-various functions for performing input and output like gets,scanf,printf,puts,fopen,...
- **<stdlib.h>**- malloc, calloc, atoi, atol
- **<string.h>**-String related functions E.g. Strcpy, strcat, strcmp
- **<time.h>**-functions to get and manipulate date and time information E.g.clock,time,difftime



# Summary

- C is a general purpose computing programming language which was invented and was first implemented by **Dennis Ritchie**
- The main characteristics of C language are portability, structured programming, memory efficiency etc.,
- Basic Structure of C programming
- Both C input/output statements are supported by C
- Formatting Specifier
- Standard Library Functions



# ANY QUERIES

