UNIT-II

Introduction- C Structure- syntax and constructs of ANSI C - Variable Names, Data Type and Sizes, Constants, Declarations - Arithmetic Operators, Relational Operators, Logical Operators, Type Conversion, Increment and Decrement Operators, Bitwise Operators, Assignment Operators and Expressions, Precedence and Order of Evaluation

Overview of C

TOPICS

- INTRODUCTION- C LANGUAGE
- FEATURES OF C LANGUAGE
- STRUCTURE OF A C PROGRAM
- HEADER FILES
- COMMENTS
- CODE INDENDATION
- COMPILATION AND EXECUTION OF A C PROGRAM
- INPUT/OUTPUT FUNCTIONS

INTRODUCTION- C LANGUAGE

- C is a programming language developed at AT & T's Bell Laboratories of USA in 1972.
- It was designed and written by Dennis Ritchie.
- C was limited to use within Bell Laboratories until 1978, when Brian Kernighan and Ritchie published a definitive description of the language.
- C is a general purpose programming language
- C is a Procedural Language.

HISTORY OF C LANGUAGE

FEATURES OF C LANGUAGE

- Simple and Efficient
- Portability
- Modularity
- Speed
- Case Sensitive
- Dynamism
- Extensibility
- Flexibility and Adaptability

WHY C IS POPULAR?

- C is a popular language since it is simple, reliable and easy to use. It is a language which has survived for more than 3 decades even when new languages, tools and technologies have evolved.
- Major parts of popular operating systems like Windows, UNIX, Linux are written in C.
- Common consumer devices like microwave oven, washing machines and digital cameras are getting smarter day by day. This smartness comes from a microprocessor, an operating system and a program embedded in this devices. Such programs are written in Embedded C.
- Many popular gaming frameworks have been built using C language.
- To closely interact with hardware devices C language provides features and also make these interactions feasible without compromising performance.

STRUCTURE OF A C PROGRAM

Documentation section

Link section

Definition section

Global declaration section

main () Function section

{

Declaration part
Executable part

}

Subprogram section

Function 1
Function 2
……..
……..
Function n

(User defined functions)

STRUCTURE OF A C PROGRAM

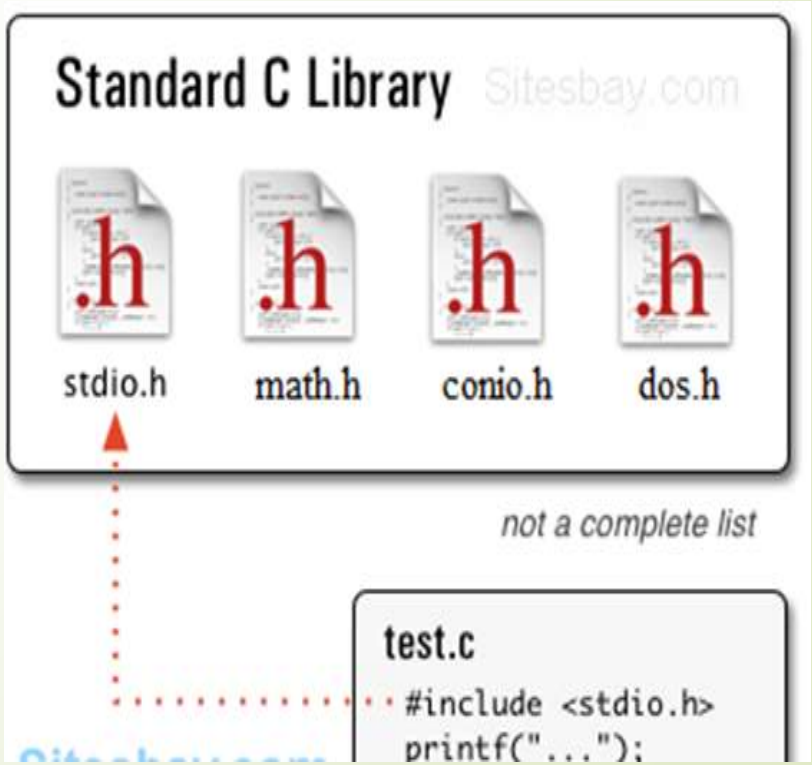
STRUCTURE	SAMPLE C PROGRAM
Documentation Section	//Sample Program
Link Section	#include<stdio.h> #include<conio.h>
Function Declaration Section	void func1();
Global Declaration Section	int a=10;

Main Section	void main ()
	{
	clrscr();
	printf (“the value of a is inside main%d”,a);
	fun();
	}

Subroutine Section	void fun ()
	{
	printf (“the value of a is inside func1%d”, a);
	}

HEADER FILES

- A header file is a file with extension .h which contains C function declarations and macro definitions to be shared between several source files.
- There are two types of header files:
  - Files that comes with the compiler(predefined)
  - Files that are written by the Programmer



COMMENTS

- In C, comments can be placed anywhere in the program which are not executed as a part of the program.
- Comments are used for the better understandability of the program by others and also helps in better debugging of the program.
- Adding Comments to a program is a highly recommended practice.

SYNTAX OF COMMENT

Single line Commenting can be done in 2 ways,

```
/* Comments here */
/* Sample program-find area of circle*/
//Comment goes here
//Sample program-find area of circle
```

Multiple Line Commenting is done as follows,

```
/*Comment in line 1
Comment in line 2
Comment in line n */
/* Sample program-find area of circle
Programming Language-C */
```

CODE INDENDATION

- Indentation is one of the most important aspects in any programming domain. Indentation is a way to organize and document your source code.
- Proper code indentation will make it:
  - Easier to read
  - Easier to understand
  - Easier to modify
  - Easier to maintain
  - Easier to enhance

CODE INDENDATION

- How to Indent a C program?
- Always indent the body (bodies) of a statement with a uniform amount of space(2,4 or 8 spaces) from the first character of the statement. Statements that have bodies include:
  - Loops
  - If statements
  - Subprograms

CODE INDENDATION

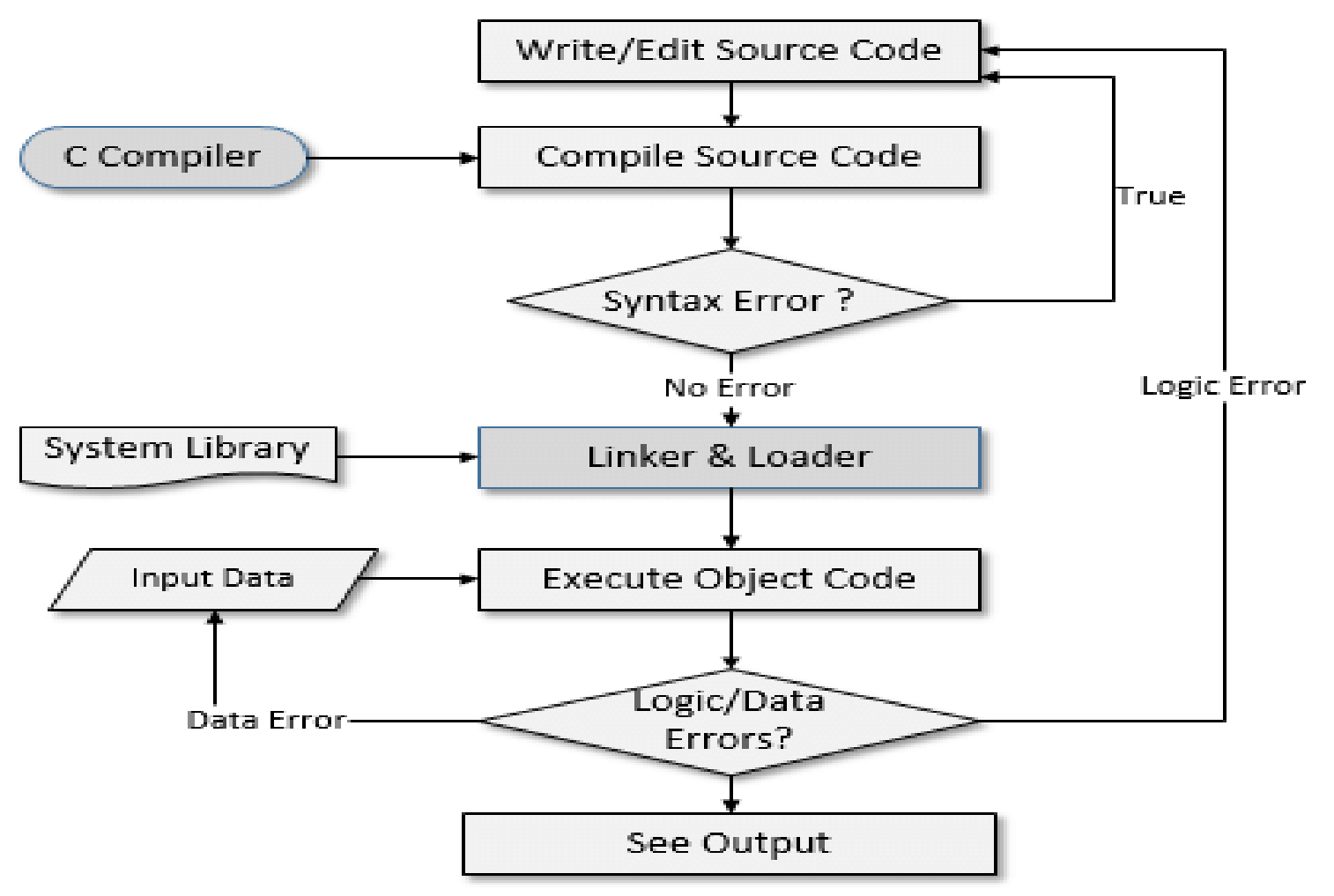
Example 1:

```
int main (int arg c, char *argv[])
{
        int   a,
        float  b;
        printf("Give me an 'a' : ");
        scanf("%d",&a);
        printf(" Give me a 'b' : ");
        scanf("%f",&b);
        . . .
}
```

Example 2:

```
if (victor(human))
{       human_wins++;
        printf("humble servant.\n");
}
else
{       computer_wins++;
        printf("Your destiny \n");
}
```

COMPILATION AND EXECUTION



COMPILER VS INTEPRETER

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.

COMPILATION AND EXECUTION

Compiling and Executing a C program in gcc(Linux/Windows):

- Create a C program using an editor (vi editor/Notepad/Notepad++) and save the file as sample.c.

  $vi sample.c **>open using notepad/notepad++**
- Compile it using below command

  $gcc sample.c **>gcc sample.c**
- After compilation execute it using below command.

  $ ./a.out **>a**

NOTEPAD++

```c
#include<stdio.h>
int main()
{
int n1,n2,n3;
int sum;
scanf("%d %d %d",&n1,&n2,&n3);
sum=n1+n2+n3;
printf("The sum is:%d",sum);
return 0;
}
```

Command prompt

```
Microsoft Windows [Version 10.0.19042.928]
(c) Microsoft Corporation. All rights reserved.

C:\Users\dsorn>cd..

C:\Users>cd..

C:\>cd td*

C:\TDM-GCC-64>cd bin

C:\TDM-GCC-64\bin>gcc add.c

C:\TDM-GCC-64\bin>a
12 12 12
The sum is:36
C:\TDM-GCC-64\bin>
```

INPUT/OUTPUT FUNCTIONS

- In C Language a library of functions (predefined functions) is provided to perform I/O operations. The I/O library functions are listed the “header” file <stdio.h>.
- All input and output is performed with streams. A "stream" is a sequence of characters organized into lines. Each line consists of zero or more characters and ends with the "newline" character.
- Standard input stream is called "stdin" and is normally connected to the keyboard.
- Standard output stream is called "stdout" and is normally connected to the display screen.
- Standard error stream is called "stderr" and is also normally connected to the screen.

INPUT/OUTPUT FUNCTIONS

Basic format specifiers:

- d -- displays a decimal (base 10) integer
- f -- displays a floating point value
- c -- displays a single character
- s -- displays a string of characters

- l -- used with other specifiers to indicate a "long"
- e -- displays a floating point value in exponential notation
- g -- displays a number in either "e" or "f" format

INPUT/OUTPUT FUNCTIONS

FORMATTED INPUT FUNCTION: scanf ()

This function provides for formatted input from the keyboard.

Syntax:

scanf (“format” , &var1, &var2, …) ;

The “format” is a listing of the data types of the variables to be input and the & in front of each variable name tells the system WHERE(address) to store the value that is given as input. It provides the address for the variable**.**

Example:

```
float a;  int b;
scanf (“%f%d”, &a, &b);
```

INPUT/OUTPUT FUNCTIONS

FORMATTED OUTPUT FUNCTION: printf ()

This function provides for formatted output.

Syntax:

printf (“format” , var1, var2, …) ;

The “format” is a listing of the data types of the variables(Var1,var2,…) to be output in the monitor screen.

Example:

int b=10;

printf(“%d”,b); -----> OUTPUT : 10

printf(“Welcome to REC”); ---→OUTPUT: Welcome to REC

MCQ

A C Compiler:

A. Translates a C program into Assemble Code for the underlying CPU

B. Translates any High Level program into Assemble code for the underlying CPU

C. Translates a C program into Machine Language Code

D. Translates C library functions to Machine level code

ANSWER: C