Introduction to C

Chapter 2

C Programming

- C is a general purpose high level programming language.
- It was originally developed by Dennis Ritchie for the development of Unix Operating System.
- General Purpose: C Programming is designed for developing software that applies in wide range of applications.
- Procedural: C Program are the sets of one or many functions. Each function performs a specific task in C program. These functions are called in sequence to make the program work as designed.
-

Assignment 2

- Write short notes on a brief history of C language.
- Why is C called a middle level language?

Features of C programming

1. Simple Language
2. Structured language
3. Portable
4. Middle-level Language
5. Fast Execution
6. Rich Library Set
7. Memory Management
8. Modular

Advantages

1. **Easy to Learn**

   -easy to learn middle-level language

2. **Low-level Language Support**

   -very close to the machine language because it supports features like pointer, but and bit-manipulation

   -allows programmers to write directly to the memory-popular to program the hardware devices, Operating System, drivers, kernels, etc

3. **Structured Programming Language**

   -functions and subroutines are its main structural components

   -makes a program easier to understand and modify

4. **Produces Efficient Programs**

   -C is a compiled programming language. This creates fast and efficient executable files.

   -provides a set of library functions for common utilities

   -the inbuilt functions help to make the development faster

5. **Produces Portable Programs**

   -programs in C language can run easily on any other compilers with little or no modifications at all

   -provides universality and portability across various computer architectures

6. **Powerful Programming Language**

-provides wide-range of inbuilt data types and ability to create customized data types using structures as well

-as C standard library, a large set of commonly used Input/ Output, Mathematical, String and other related functions are provided

-has a rich set of control statements, arithmetic operator, loops, etc, which provides a powerful tool for a programmer to implement his logic as a C program

7. **Memory Management**

-provides support for dynamic memory allocation

-by calling library functions such as *malloc*, *calloc* and *free*, we can allocate any free space and free the allocated memory space at anytime

Limitations

1. Lacks the concept of **Object-Oriented Programming** (**OOP**)
2. Lacks **runtime checking**
3. No strict type checking

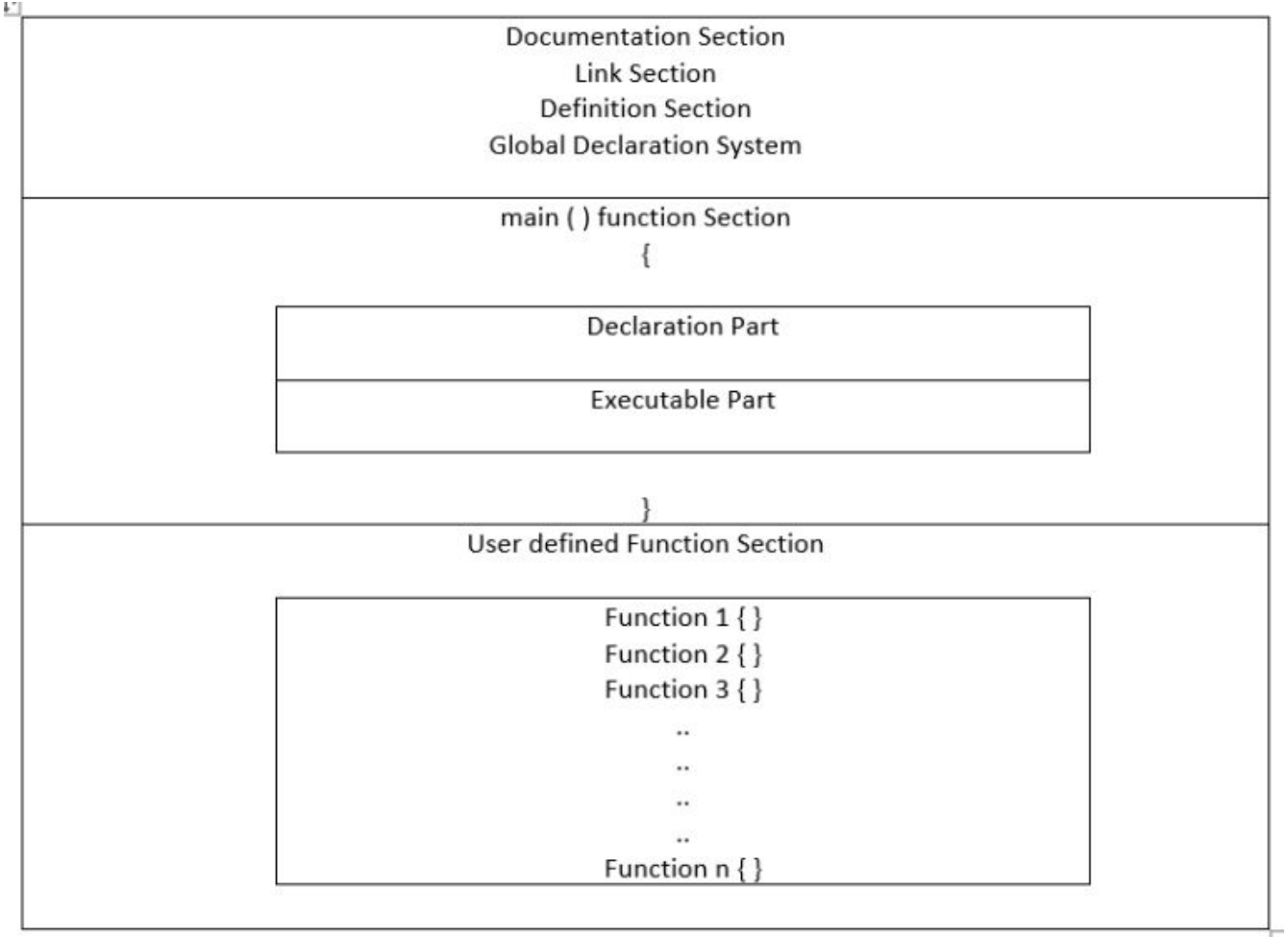
   example: *integer type data can be passed for the declaration of float type data*
4. Lacks the concept of **namespace**

   *(A **namespace** is a set of symbols that are used to organize objects of various kinds, so that these objects may be referred to by name.)*
5. Lacks the concept of **constructor** or **destructor**

   *(In class-based object-oriented programming, a **constructor** is a special type of subroutine called to create an object.*

   *And, a **destructor** is a method which is automatically invoked when the object is destroyed.)*

Structure of C program



DOCUMENTATION SECTION

This section consists of comments, some description of the program, name of the programmer and any other useful points that can be referenced later.

LINK SECTION

This section provides instruction to the compiler to link function from the library functions.

DEFINITION SECTION

This section defines all the symbolic constants.

GLOBAL DECLARATION SECTION

This section consists of all the function declaration and global variables.

MAIN ( ) FUNCTION SECTION

Every C program must have a main() function which is the starting point of the program execution.

SUBPROGRAM

This section contains all the user-defined functions.

First C program “Hello World”

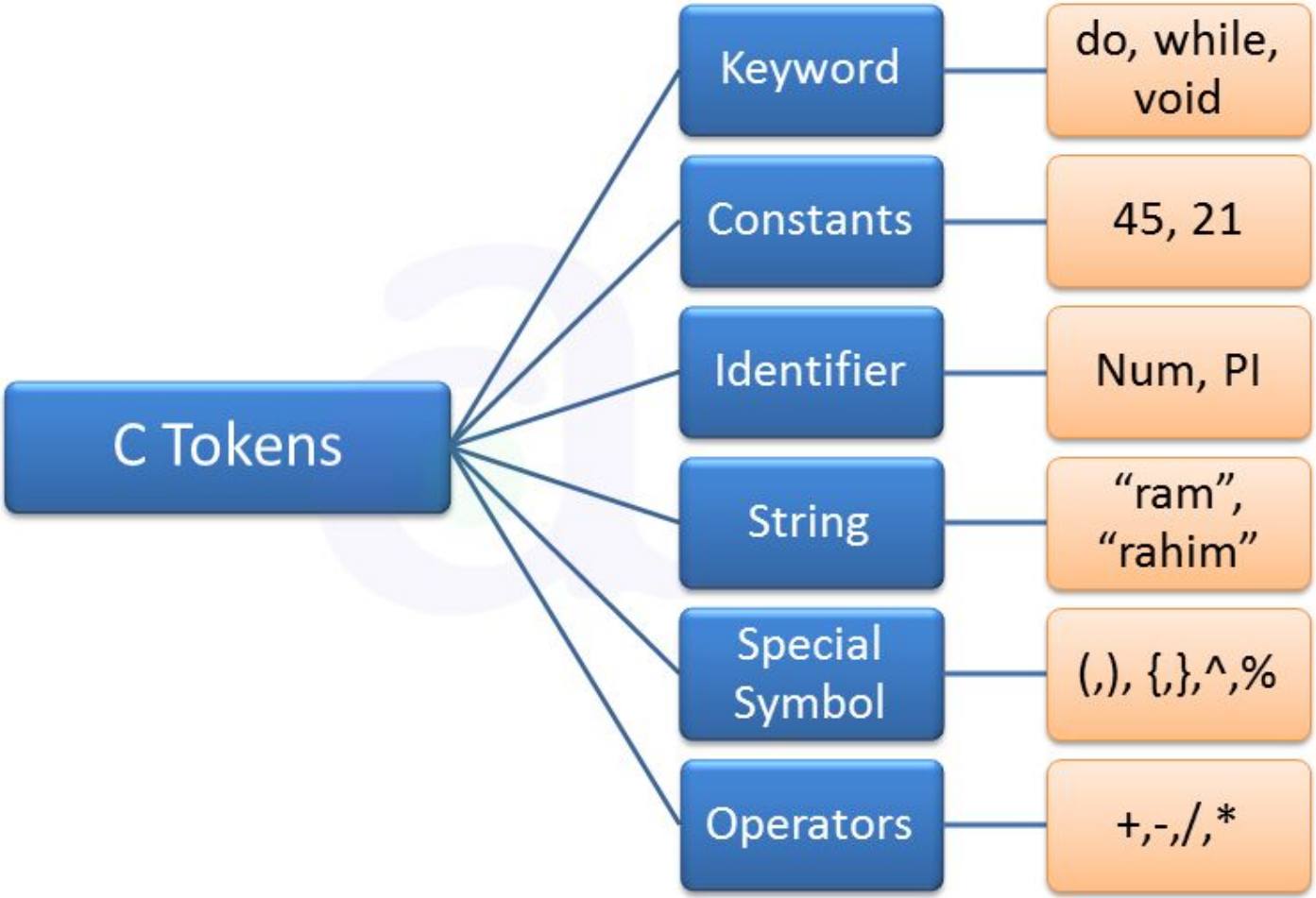
```
#include <stdio.h>                    include information about standard library

main()                                define a function named main
                                      that receives no argument values
{                                     statements of main are enclosed in braces

      printf("hello, world\n");        main calls library function printf
                                       to print this sequence of characters;
}                                      \n represents the newline character
```

The first C program.

Note: For Turbo C/C++ IDE, you should include “conio.h”, console input/output
And in the main function “getch()” at the bottom of the file, to hold the output screen

C Tokens

- C Tokens are the smallest building block or smallest unit of a C program. The compiler breaks a program into the smallest possible units and proceeds to the various stages of the compilation, which is called token.
- C tokens are:
  - Keywords
  - Identifiers
  - Constants
  - Strings
  - Operator etc.


C Tokens
Keyword
do, while, void
Constants
45, 21
Identifier
Num, PI
String
“ram”, “rahim”
Special Symbol
(,), {,},^,%
Operators
+,-,/,*

Keywords are predefined, reserved words used in programming that have special meanings to the compiler. Keywords are part of the syntax and they cannot be used as an identifier.

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

- Identifier refers to name given to entities such as variables, functions, structures etc.
- Identifiers must be unique. They are created to give a unique name to an entity to identify it during the execution of the program

For eg: int money

char sky

double accountBalance

Int , char, double are keywords.

And money, sky, accountBalance are identifiers.

Rules for naming identifiers

1. A valid identifier can have letters (both uppercase and lowercase letters), digits and underscores.
2. The first letter of an identifier should be either a letter or an underscore.
3. You cannot use keywords like `int`, `while` etc. as identifiers.
4. There is no rule on how long an identifier can be. However, you may run into problems in some compilers if the identifier is longer than 31 characters.

You can choose any name as an identifier if you follow the above rule, however, give meaningful names to identifiers that make sense.

Special Characters in C Programming				
,	<	>	.	_
(	)	;	$	:
%	[	]	#	?
'	&	{	}	"
^	!	*	/	|
-	\	~	+	

Variables

- In programming, a variable is a container (storage area) to hold data.
- To indicate the storage area, each variable should be given a unique name (identifier). Variable names are just the symbolic representation of a memory location. For example:

  int playerScore = 35;

- Here, `playerScore` is a variable of `int` type. Here, the variable is assigned an integer value `35`.
- The value of a variable can be changed, hence the name variable.

Rules for naming a variable

1. A variable name can only have letters (both uppercase and lowercase letters), digits and underscore.
2. The first letter of a variable should be either a letter or an underscore.
3. There is no rule on how long a variable name (identifier) can be. However, you may run into problems in some compilers if the variable name is longer than 31 characters.

Escape Sequences	
Escape Sequences	Character
`\b`	Backspace
`\f`	Form feed
`\n`	Newline
`\r`	Return
`\t`	Horizontal tab
`\v`	Vertical tab
`\\`	Backslash
`\'`	Single quotation mark
`\"`	Double quotation mark
`\?`	Question mark
`\0`	Null character

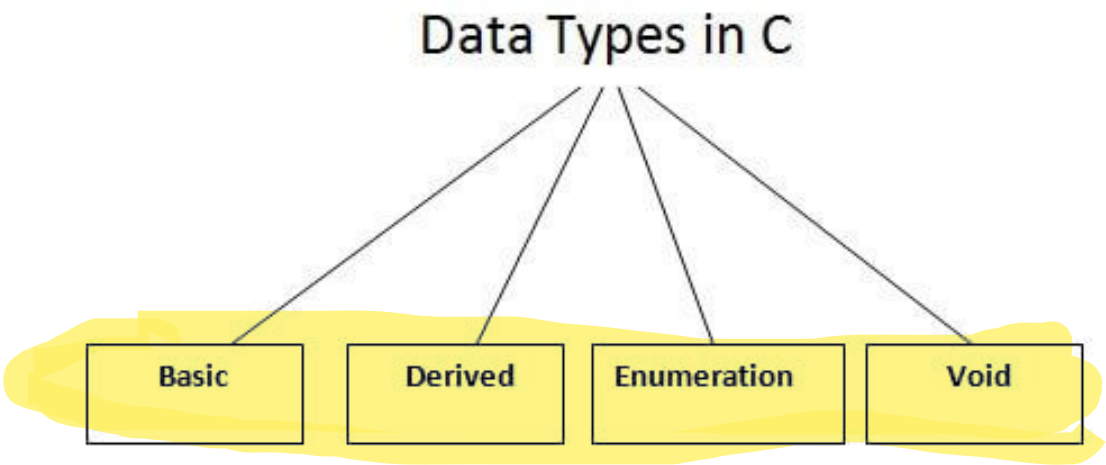
Constants

If you want to define a variable whose value cannot be changed, you can use the `const` keyword. This will create a constant. For example,

```
const double PI = 3.14;
```

If we try to change PI,

```
PI=2.5; // error
```



Types	Data Types
Basic Data Type	int, char, float, double
Derived Data Type	array, pointer, structure, union
Enumeration Data Type	enum
Void Data Type	void

Basic Data types

Type	Size (bytes)	Format Specifier
`int`	at least 2, usually 4	`%d`, `%i`
`char`	1	`%c`
`float`	4	`%f`
`double`	8	`%lf`
`short int`	2 usually	`%hd`
`unsigned int`	at least 2, usually 4	`%u`
`long int`	at least 4, usually 8	`%ld`, `%li`
`long long int`	at least 8	`%lld`, `%lli`
`unsigned long int`	at least 4	`%lu`
`unsigned long long int`	at least 8	`%llu`
`signed char`	1	`%c`
`unsigned char`	1	`%c`
`long double`	at least 10, usually 12 or 16	`%Lf`

Data Types	Memory Size	Range
char	1 byte	−128 to 127
signed char	1 byte	−128 to 127
unsigned char	1 byte	0 to 255
short	2 byte	−32,768 to 32,767
signed short	2 byte	−32,768 to 32,767
unsigned short	2 byte	0 to 65,535
int	2 byte	−32,768 to 32,767
signed int	2 byte	−32,768 to 32,767
unsigned int	2 byte	0 to 65,535
short int	2 byte	−32,768 to 32,767
signed short int	2 byte	−32,768 to 32,767
unsigned short int	2 byte	0 to 65,535
long int	4 byte	-2,147,483,648 to 2,147,483,647
signed long int	4 byte	-2,147,483,648 to 2,147,483,647
unsigned long int	4 byte	0 to 4,294,967,295
float	4 byte	
double	8 byte	
long double	10 byte	

C input/output

In C programming, `printf()` is one of the main output function. The function sends formatted output to the screen. For example, C programming language provides many of the built-in functions to read given input and write data on screen, printer or in any file.

scanf() and printf() functions

```
# include <stdio.h>

# include<conio.h>

void main ( )

{

i n t i ;

p r i n t f ( ” Enter a v al u e ” ) ;

s c a n f (”%d”,& i ) ;

p r i n t f ( ”\nYou e n t e r e d : %d ” , i ) ;

g e t c h ( ) ;

}
```

Format Specifiers

Data Type	Format Specifier
`int`	`%d`
`char`	`%c`
`float`	`%f`
`double`	`%lf`
`short int`	`%hd`
`unsigned int`	`%u`
`long int`	`%li`
`long long int`	`%lli`
`unsigned long int`	`%lu`
`unsigned long long int`	`%llu`
`signed char`	`%c`
`unsigned char`	`%c`
`long double`	`%Lf`

Operators in C

An operator is a symbol that tells the compiler to perform specific mathematical or logical functions. C language is rich in built-in operators and provides the following types of operators.

- Arithmetic Operator
- Relational Operator
- Logical Operator
- Bitwise Operator
- Assignment Operator
- Misc Operator

Arithmetic Operator

The basic operators for performing arithmetic are the same in many computer languages:

1. + Addition

2. - Subtraction

3. * Multiplication

4. / Division

5. % Modulus (Remainder)

For exponentiations we use the library function pow. The order of precedence of these operators is % / * + - . it can be overruled by parenthesis.

Division of an integer quantity by another is referred to integer division. This operation results in truncation. i.e.When applied to integers, the division operator / discards any remainder, so 1 / 2 is 0 and 7 / 4 is 1. But when either operand is a floating-point quantity (type float or double), the division operator yields a floating-point result, with a potentially nonzero fractional part. So 1 / 2.0 is 0.5, and 7.0 / 4.0 is 1.75. A operator acts up on a single operand to produce a new value is called a unary operator. The decrement and increment operators - ++ and – are unary operators. They increase and decrease the value by 1.

sizeof() is another unary operator. The output given by the sizeof() operator de-pends on the computer architecture. The values stated by sizeof() operator down below are based on a 16 bit compiler.

1 int x , y ;

2 y=s i z e o f (x) ;

The value of y is 2. The sizeof() of an integer type data is 2 that of float is 4, that of double is 8, that of char is 1.

```
// Working of arithmetic operators
#include <stdio.h>
int main() {
    int a = 9,b = 4, c;
    c = a+b;
    printf("a+b = %d \n",c);
    c = a-b;
    printf("a-b = %d \n",c);
    c = a*b;
    printf("a*b = %d \n",c);
    c = a/b;
    printf("a/b = %d \n",c);
    c = a%b;
    printf("Remainder when a divided by b = %d \n",c);
    return 0;
```

Relational Operator

< (less than), <= (less than or equal to), > (greater than), >= (greater than or equal to), == (equal to) and ! = (not equal to) are relational operators. A logical expression is expression connected with a relational operator.

For example ‘b*b– 4*a*c<0 is a logical expression. Its value is either true or false.

1 int i , j , k ;

2 i =1;

3 j =2;

4 K=i+j ;

The expression k>5 evaluates to false and the expression k<5 evaluates to true.

Logical Operator

The relational operators work with arbitrary numbers and generate true/false values.You can also combine true/false values by using the Boolean operators, which take true/false values as operands and compute new true/false values. The three Boolean operators are:

1. && AND

2. || OR

3. ! NOT

The && (‘‘and’’) operator takes two true/false values and produces a true (1) result if both operands are true (that is, if the left- hand side is true and the right-hand side is true). The || (‘‘or’’) operator takes two true/false values and produces a true (1) result if either operand is true. The ! (‘‘not’’) operator takes a single true/false value and negates it, turning false to true and true to false (0 to 1 and nonzero to 0).

&& (and) and || (or) are logical operators which are used to connect logical expressions. Where as ! (not) is unary operator, acts on a single logical expression.

1 (a<5) && (a>2)

2 (a<=3) | | (a>2)

In the example if a= 3 or a=6 the logical expression returns true.

Assignment Operator

These operators are used for assigning a value of expression to another identifier.

=, + =, - = , * =, /= and % = are assignment operators.

a = b+c results in storing the value of b+c in ‘a’.

a += 5 results in increasing the value of a by 5.

a /= 3 results in storing the value a/3 in a and it is equivalent a=a/3

C increment and decrement operators

C programming has two operators increment `++` and decrement `--` to change the value of an operand (constant or variable) by 1.

Increment `++` increases the value by 1 whereas decrement `--` decreases the value by 1. These two operators are unary operators, meaning they only operate on a single operand.

```
// Working of increment and decrement operators
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;

    printf("++a = %d \n", ++a);
    printf("--b = %d \n", --b);
    printf("++c = %f \n", ++c);
    printf("--d = %f \n", --d);

    return 0;
}
```

C assignment operator

An assignment operator is used for assigning a value to a variable. The most common assignment operator is =

Operator	Example	Same as
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
/=	a /= b	a = a/b
%=	a %= b	a = a%b

Conditional Operator

The operator ?: is the conditional operator. It is used as variable 1 = expression 1 ? expression 2 : expression 3. Here expression 1 is a logical expression and expression 2 and expression 3 are expressions having numerical values. If expression 1 is true, value of expression 2 is assigned to variable 1 and otherwise expression3 is assigned.

```
1 int a , b , c , d , e ;
2 a=3; b=5; c =8;
3 d=(a<b ) ? a : b ;
4 e=(b>c ) ? b : c ;
```

The evaluation of the expression results the value of d = 3 and e = 8.

Bitwise Operator

C has a distinction of supporting special operator known as bitwise operator for manipulation of data at bit level. These operators are used for testing bits or shifting them right or left. Bitwise operator may not be applied to float or double. Following are the bitwise operators.

- & bitwise AND
- | bitwise OR
- ˆ bitwise exclusive OR
- << shift left
- >> shift right

Operator Precedence in C

Operator precedence determines the grouping of terms in an expression and decides how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has a higher precedence than the addition operator.

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has a higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.An arithmetic expression without parenthesis will be evaluated from left to right using the rules of precedence. There are two distinct priority levels of arithmetic operators in C.

• Higher Priority * /

• Lower Priority + -

The expression x=a-b/3+c*2-1 where a=9, b=12 and c=3 will be evaluated as

step1: x=9-12/3+3*2-1 Higher priority operators left to right division first

step2: x=9-4+3*2-1 Higher priority operators left to right multiply

step3: x=9-4+6-1 Lower priority operators left to right subtraction

step4: x=5+6-1 Lower priority operators left to right addition

step5: x=11-1 Lower priority operators left to right subtraction

step6: x=10 Final Result

Comments

- In the C Programming Language, you can place comments in your source code that are not executed as part of the program.
- Comments provide clarity to the C source code allowing others to better understand what the code was intended to accomplish and greatly helping in debugging the code. Comments are especially important in large projects containing hundreds or thousands of lines of source code or in projects in which many contributors are working on the source code.

Single line comment: You can use “//” for single line comment.

// {Comment Section}

Multiple Line comment: A comment starts with a slash asterisk /* and ends with a asterisk slash */ and can be anywhere in your program.

```
/*
    {Comment Section}
*/
```

Exercises

- WAP to find the average of two numbers.
- WAP to find remainder and quotient.
- WAP to find the area and circumference of a circle
- WAP to find the area of equilateral triangle
- WAP to swap two numbers using temporary variable and without using temporary variable
- WAP to find ASCII value of character entered by user.

Function	Description	Example
sqrt(x)	square root of x	sqrt(4.0) is 2.0 sqrt(10.0) is 3.162278
exp(x)	exponential (e^x)	exp(1.0) is 2.718282 exp(4.0) is 54.598150
log(x)	natural logarithm of x (base e)	log(2.0) is 0.693147 log(4.0) is 1.386294
log10(x)	logarithm of x (base 10)	log10(10.0) is 1.0 log10(100.0) is 2.0
fabs(x)	absolute value of x	fabs(2.0) is 2.0 fabs(-2.0) is 2.0
ceil(x)	rounds x to smallest integer not less than x	ceil(9.2) is 10.0 ceil(-9.2) is -9.0
floor(x)	rounds x to largest integer not greater than x	floor(9.2) is 9.0 floor(-9.2) is -10.0
pow(x,y)	x raised to power y (x^y)	pow(2,2) is 4.0
fmod(x)	remainder of x/y as floating-point number	fmod(13.657, 2.333) is 1.992
sin(x)	sine of x (x in radian)	sin(0.0) is 0.0
cos(x)	cosine of x (x in radian)	cos(0.0) is 1.0
tan(x)	tangent of x (x in radian)	tan(0.0) is 0.0

```c
#include <stdio.h>
#include <math.h>
int main()
{
    printf("%f\n",sqrt(10.0));
    printf("%f\n",exp(4.0));
    printf("%f\n",log(4.0));
    printf("%f\n",log10(100.0));
    printf("%f\n",fabs(-5.2));
    printf("%f\n",ceil(4.5));
    printf("%f\n",floor(-4.5));
    printf("%f\n",pow(4.0,.5));
    printf("%f\n",fmod(4.5,2.0));
    printf("%f\n",sin(0.0));
    printf("%f\n",cos(0.0));
    printf("%f\n",tan(0.0));
    return 0;
}
```

ASCII

ASCII value represents the English characters as numbers, each letter is assigned a number from 0 to 127. The character sets used in modern computers, in HTML, and on the Internet, are all based on ASCII.

```
#include <stdio.h>

int main()

{

    char ch;

    printf("Enter any character:");

    scanf("%c", &ch);

    /* Using the format specifiers we can get the ASCII code

     * of a character. When we use %d format specifier for a

     * char variable then it displays the ASCII value of a char

     */

    printf("ASCII value of character %c is: %d", ch, ch);

    return 0;   }
```