UNIT I BASICS OF C PROGRAMMING

Introduction to programming paradigms - Structure of C program - C programming: Data Types – Storage classes - Constants – Enumeration Constants - Keywords – Operators: Precedence and Associativity - Expressions - Input/Output statements, Assignment statements – Decision making statements - Switch statement - Looping statements – Pre-processor directives - Compilation process

---

Part A

2 marks

1 List the rules for defining variables in C
2 What are various types of C operators?
3 What is scope of a variable? List the two types of scopes .
4 Write any four escape sequences in C
5 What are the uses of the following keywords:
auto, double, int, long
6 Illustrate any bitwise operator with example
7 What is ternary operator?
8 Differentiate between relational and logical expressions
9 List any four short hand assignment operators
10 Differentiate between while and do.. while statements
11 What is operator precedence and associativity. Illustrate with example.
12 Differentiate between break and continue statements.
13 What are the various I/O functions in C
14 What is modulo operator?
15 Give the use of preprocessor
16 Define implicit type conversion
17 Define register storage class
18 What will be the output of the following code

```c
#include "stdio.h"
int main(){
int a, b= 50, c;
a = 5 <= 8 && 6 != 5;
c =  b++ + ++b;
printf("a= %d, b= %d, c=%d \n", a, b, c);
}
```

Part B

13 marks

1 Describe the statements for decision making, branching and looping

2 i) Explain different data types in C with examples? (7)
ii) Write the C program to find roots of a quadratic equation (6)

3 i) What are the different operators in C? Explain with examples (7)
ii) Write the C program to find whether given number is prime or not (6)

4 i) What is storage class? List and explain with example (7)
ii) Write the C program to find the sum of digits of an integer (6)

5 Explain briefly the formatted and unformatted I/O functions in C

Part C

15 marks

1. Write short notes on
   i) register storage class (4)
   ii) #include statement (4)
   iii) #ifdef.... #endif (3)
   iv) signed and unsigned integers (4)

Structure of C program

```c
                        // comments
/*  ************************************************
    Program: add.c
    Author: Ashok.R
    Verion: 1.0.1
    This program adds two numbers and print the result
    ************************************************
*/
                        //  pre-processor directives
#include <stdio.h>
                        // Global declarations
int result = 0;
void add(int, int);
                        // main() function
int main()
{
                        // local declarations
    int a = 10;
    int b = 20;
                        // statements
    add(a,b);
    printf("%d", result);
    return 0;
}

                        // other functions
void add(int x, int y)
{
    result = x + y;
}
```

Comments

Comments are included to improve the readability of the program

1. Single line comments
   starts with //
   Example:
   // This is single line comment
2. Multi line comments
   starts with /* and ends with */
   Example:
   As shown in the example program

Pre-processor directives

- Starts with #
- It is executed before the compiler compiles the source code

- Example:
  #include directive includes the specific header file to the program before compilation.

Global declarations

Variables can be declared globally and can be accessed anywhere within the program

Functions

A C program contains one or more functions. The statements inside a function are written in a sequence to perform a specific task. The main() function must be included in every program. The execution of a C program begins at main() function.

Every function has two parts:

1. Header:
   Format:
   return type function name (arguments)
   Example:
   void add(int x, int y)

2. Body:
   A sequence of statements enclosed within curly braces {}

C Tokens

The smallest individual units of a C program.
Types of tokens: keywords, identifiers, constants, string, special symbols, operators

Keywords

Reserved words to imply special meaning to the compiler. There are 32 keywords in C.

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

All the keywords must be written in lower case.

Identifiers

Names to identify the program elements such as variables, arrays and functions.

Identifier (Variable) naming rules

- It must begin with a letter
- Only letters, digits and underscore(_) are permitted to be used in an identifier name
- No other special characters are allowed
- Case sensitive
- Keywords can not be used as identifiers

Escape sequences

Non printable characters

\a	alert
\b	back space
\t	horizontal tab
\v	vertical tab
\n	new line
\r	carriage return

Constants

The value is known at the compile time and does not change during the execution of the program.

- Numeric Constants
    - Integer Constants

    - Real Constants
- Character Constants
    - Single Character Constants
    - String Constants

Integer Constants

An integer constant is a sequence of digits. There are 3 types of integers, namely decimal integer, octal integers and hexadecimal integer.

Decimal Integers consists of a set of digits 0 to 9 preceded by an optional + or - sign. Spaces, commas and non digit characters are not permitted between digits. Examples for valid decimal integer constants are 123, -31, + 78

Octal Integers constant consists of any combination of digits from 0 through 7 with an 0 at the beginning. Some examples of octal integers are 026, 00347, 0676

Hexadecimal integer constant is preceded by 0X or 0x, they may contain alphabets from A to F or a to f. The alphabets A to F refer to 10 to 15 in decimal digits. Examples of valid hexadecimal integers are 0X2, 0X8C

Real Constants

Real Constants consists of a fractional part in their representation. Integer constants are inadequate to represent quantities that vary continuously. These quantities are represented by numbers containing fractional parts like 26.082. Examples of real constants are

0.0026, -0.97, 435.29, +487.0

Real Numbers can also be represented by exponential notation. The general form for exponential notation is

mantissa **e** exponent

Example: 13e-4 ➔ 13×10^{-4} ➔ 0.0013

Single Character Constants

A Single Character constant represent a single character which is enclosed in a pair of single quotation symbols. Example for character constants are '5', 'x'

All character constants have equivalent integer values which are called ASCII Values.
The ASCII value for ‘a’ is 97 and for ‘A’, it is 65.

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[ENG OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

String Constants

A string constant is a set of characters enclosed in double quotation marks. The characters in a string constant sequence may be an alphabet, number, special character and blank space. Example of string constants are "Hello World"

Enumeration Constants

"Enumerated" data type is the user defined data type which can take on only finite set of values.

Example:

enum size(S=30, M=32, L=34, XL=36);
enum size mydress = M;

An enumeration is a list of constant integer values.

Variables

A variable is an entity whose value can vary during the execution of a program.
Example:

int a; int b=5;

a = 10;
a = b+a;

Data types

Primitive data types

char: The most basic data type in C. It stores a single character and occupies 1 byte of memory.

int: An int variable is used to store an integer and occupies 4 bytes (32 bits) in a 32 bit machine.

float: It is used to store decimal numbers (numbers with floating point value) with single precision.

double: It is used to store decimal numbers (numbers with floating point value) with double precision.

Examples:

```
char grade = 'A';
int a = 10;
float height = 154.8;
```

Integer types

Type	Storage size	Value range
char	1 byte (8 bits)	-2^7 to 2^7-1 (or) -128 to 127
unsigned char	1 byte	0 to 2^8-1 (or) 0 to 255
signed char	1 byte	-128 to 127
int	4 bytes (32 bits)	-2^{31} to $2^{31}-1$ (or) -2,147,483,648 to 2,147,483,647
unsigned int	4 bytes	0 to $2^{32}-1$ (or) 0 to 4,294,967,295
short	2 bytes	-32,768 to 32,767
unsigned short	2 bytes	0 to 65,535
long	8 bytes	-2^{63} to $2^{63}-1$
unsigned long	8 bytes	0 to $2^{64}-1$

Floating point types

Type	Storage size	Value range	Precision
float	4 byte	1.2E-38 to 3.4E+38	6 decimal places
double	8 byte	2.3E-308 to 1.7E+308	15 decimal places
long double	10 byte	3.4E-4932 to 1.1E+4932	19 decimal places

Derived data types
Arrays, functions and pointers are examples for derived data types.

User defined data types
Structures and unions are examples for user defined data types.

Const Qualifier

The qualifier const can be applied to the declaration of any variable to specify that its value will not be changed.

Example:

const float PI = 3.14;

The value of the variable PI cannot be changed in the program.

Storage classes

Storage Classes are used to describe about the features of a variable/function. These features basically include the scope, visibility and life-time which help us to trace the existence of a particular variable during the runtime of a program.

C language uses 4 storage classes.

auto: This is the default storage class for all the variables declared inside a function or a block. They are assigned a garbage value by default whenever they are declared.

extern: Extern storage class simply tells us that the variable is defined elsewhere and not within the same block where it is used. The main purpose of using extern variables is that they can be accessed between two different files which are part of a large program.

static: Static variables preserve the value of their last use in their scope. They are initialized only once and exist till the termination of the program. The value of static variables persists between function calls. By default, they are assigned the value 0 by the compiler.

register: The compiler tries to store these variables in the register of the microprocessor if a free register is available. The access of register variables are much faster than that of the variables stored in the memory during the runtime of the program. If a free register is not available, these are then stored in the memory only. We cannot obtain the address of a register variable using pointers.

Examples:

```
auto char grade = 'A';
int a = 10;  // All variables are auto by default
register int b = 200;
static int c = 10;
extern int d;
```

Storage class	Storage	initial value	life time
auto	RAM	garbage	automatic
register	CPU registers	garbage	automatic
static	RAM	zero	static
extern	RAM	zero	static

Operators

Arithmetic Operators: These are the operators used to perform arithmetic/mathematical operations on operands.

Arithmetic operator are of two types:

1. **Unary Operators**: Operators that operates or works with a single operand are unary operators. (++, --)
   Example: a++; b--;
2. **Binary Operators**: Operators that operates or works with two operands are binary operators. (+, - , *, /, %)
   Example: a + b;

Example program:

```
#include<stdio.h >
void main()
{
	int numb1, num2, sum, sub, mul, div, mod;
	scanf ("%d %d", &num1, &num2);

	sum = num1+num2;
	printf("\n The sum is = %d", sum);

	sub = num1-num2;
	printf("\n The difference is = %d", sub);

	mul = num1*num2;
	printf("\n The product is = %d", mul);

	div = num1/num2;
	printf("\n The division is = %d", div);

	mod = num1%num2;
	printf("\n The modulus is = %d", mod);
}
```

Relational Operators: Relational operators are used for comparison of the values of two operands.

Operator	Meaning
<	is less than
<=	is less than or equal to
>	is greater than
>=	is greater than or equal to

==	is equal to
!=	is not equal to

Example:
6.5 <= 25 ➔ TRUE
-65 > 0 ➔ FALSE
10 < 7 + 5 ➔ TRUE

Logical Operators: Logical Operators are used to combine two or more conditions. The result of the operation of a logical operator is a boolean value either true or false.

Operator	**Meaning**
&&	Logical AND
||	Logical OR
!	Logical NOT

Example:
6.5 <= 25 && -65 > 0 ➔ FALSE
6.5 <= 25 || -65 > 0 ➔ TRUE

Assignment Operators: Assignment operators are used to assign value to a variable. It evaluates the expression in RHS(right hand side) and assigns the result to the variable in the LHS (left hand side).

For example:

```
a = 10;
b = 20;
ch = 'y';
```

Shorthand assignment operators

Statement with simple assignment operator	**Statement with shorthand operator**
a = a + 1	a += 1
a = a – 1	a -= 1
a = a * (n+1)	a *= (n+1)

a = a / (n+1)	a /= (n+1)
a = a % b	a %= b

Bitwise Operators: The Bitwise operators is used to perform bit-level operations on the operands.

Operator	**Meaning**
&	Bitwise AND
|	Bitwise OR
^	Bitwise Exclusive
<<	Shift left
>>	Shift right

Example:
a = 4 << 2;

Operation	decimal value	binary representation
	4	0000 0000 0000 0100
after right shift 4 by 2	16	0000 0000 0001 0000

The ‘a’ will be assigned with ‘16’.

Other operators

sizeof operator
comma operator
Array subscript operator [ ]
direct member access operator (dot)
Indirect member access (arrow - >)
Address-of operator (&)
Indirection operator (*)

The sizeof Operator
The operator size of gives the size of the data type or variable in terms of bytes occupied in the memory. The operand may be a variable, a constant or a data type qualifier.

Example

`int a = 32;` `printf("%d", sizeof(a));`	4
`float f = 34.56;` `printf("%d", sizeof(f));`	8
`int a[4] = {1,2,3,4};` `printf("%d", sizeof(a));`	16

Ternary operator (?:)

The ternary operator is an operator that takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison.

Example:

c = a < b ? a : b ;

The above statement assigns the minimum of ‘a’ and ‘b’ to ‘c’.

Precedence and associativity

When an expression has more than one operator, they will be evaluated in an order defined by their precedence and associativity.

Precedence: denotes the priority (which operator gets evaluated first)

Associativity: defines the order for the operators with the same precedence or same operators (Left to right or right to left).

Order	Category	Operator	Operation	Associativity
1	Highest precedence	( ) [ ] → : : .	Function call	L → R Left to Right
2	Unary	! ~ + - ++ - - & * Size of	Logical negation (NOT) Bitwise 1's complement Unary plus Unary minus Pre or post increment Pre or post decrement Address Indirection Size of operand in bytes	R → L Right -> Left
3	Member Access	.* →*	Dereference Dereference	L → R
4	Multiplication	* / %	Multiply Divide Modulus	L → R
5	Additive	+ -	Binary Plus Binary Minus	L → R
6	Shift	<< >>	Shift Left Shift Right	L → R
7	Relational	< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	L → R
8	Equality	== !=	Equal to Not Equal to	L → R
9	Bitwise AAND	&	Bitwise AND	L → R
10	Bitwise XOR	^	Bitwise XOR	L → R
11	Bitwise OR	|	Bitwise OR	L → R
12	Logical AND	&&	Logical AND	L → R

14	Conditional	? :	Ternary Operator	R → L
15	Assignment	= *= %= /= += -= &= ^= |= <<= >>=	Assignment Assign product Assign reminder Assign quotient Assign sum Assign difference Assign bitwise AND Assign bitwise XOR Assign bitwise OR Assign left shift Assign right shift	

Example:

5 * 8 / 2 + 3

The expression is evaluated based on the precedence of the operators and associativity.

⇨ 40 / 2 + 3
⇨ 20 + 3
⇨ 23

The result after evaluating the given expression is 23.

Type conversions in expressions

Type conversion or typecasting refers to changing an object of one datatype into another.

Implicit type conversion

A mixed mode expression consists of operands of different data types.

Example:

```c
#include "stdio.h"

int main(void) {
  int i = 4;
  float f = 1.5;
  float z = i / f;
  printf("%f", z);
  return 0;
}
```

Output

```
2.666667
```

Before evaluation of mixed mode expression, an object of lower type is converted to higher type. In the above case, the object i is converted to float from int before evaluation. Such automatic type conversion is implicit type conversion.

Explicit type conversion

```c
#include "stdio.h"

int main(void) {
  int i = 4;
  float f = 1.5;
  float z = i / (int) f;
  printf("%f", z);
  return 0;
}
```

Output

```
4.000000
```

Input & Output

Formatted Input - printf()

This function is used to print text as well as value of the variables on the standard output device (monitor), printf is very basic library function in c language that is declared in stdio.h header file.

Syntax:

```
printf("message");

printf("message + format-specifier",variable-list);
```

First printf() style printf the simple text on the monitor, while second printf() prints the message with values of the variable list.

```c
#include <stdio.h>

int main()
{
        printf("Message-1");
        printf("Message-2");
        printf("Message-3");
        return 0;
}
```

Output

```
Message-1Message-2Message-3
```

How to print value of the variables?

To print values of the variables, you need to understand about **format specifiers** are the special characters followed by % sign, which are used to print values of the variable s from variable list.

Format specifiers

Here are the list some of the format specifiers, use them in printf() & scanf() to format & print values of the variables:

Character	(char)	%c
Integer	(int)	%d
Insigned integer	(unsigned int)	%ld
Long	(long)	%ld
Unsigned long	(unsigned long)	%lu

Float	(float)	%f
Double	(double)	%lf
Octal Value	(octal value)	%o
Hexadecimal Value	(hex value)	%x
String	(char[])	%s

NOTE Use ‘u’ for unsigned type modifier, ‘l’ for long.

Escape Sequences

To print special extra line/spaces etc, we use escape sequences, these characters are followed by ‘\’ (slash).

\\	\
\”	“
\’	‘
\?	?
\a	Alert
\b	Back space
\n	New Line
\t	Horizontal tab
\v	Vertical tab
\r	Carriage return

Consider the following examples:

```c
#include <stdio.h>

int main()
{
        int     num=100;
        float   val=1.23f;
        char    sex='M';

        //print values using different printf
        printf("Output1:");
        printf("%d",num);
        printf("%f",val);
        printf("%c",sex);

        //print values using single printf
        printf("\nOutput2:"); // \n: for new line in c
        printf("%d,%f,%c",num,val,sex);
        return 0;
```

```
}
```

Output

```
Output1:1001.230000M
Output2:100,1.230000,M
```

Output of Integer Numbers

The general format for printing a integer number is **% w d**

Here percent sign (%) denotes that a specifier for conversion follows and **w** is an integer number which specifies the width of the field of the number that is being printed. The data type character **d** indicates that the value to be printed is integer.

Example:

```
int main(void) {
  printf("%4d\n%4d\n%4d\n", 1, 100, 1000);
  return 0;
}
```

Output:

```
   1
 100
1000
```

			1
	1	0	0
1	0	0	0

Output of Real Numbers:

The general format of specifying a real number output is :

%w.p f

w – the minimum number of positions that are to be used for the display of the value.

p – the number of digits to be displayed after the decimal point.

Program:

```
int main(void) {
  printf("%10.2f\n", 12.4567);
  printf("%10.2f\n", 1234567.4567);
  return 0;
}
```

Output:

```
     12.46
1234567.46
```

Printing of Strings

The format specification for outputting strings is:

%w.ps

w - specifies the field width for display

p - specifies only first p characters of the string are to be displayed.

Example:

```
int main(void) {
  printf("%10.2s\n", "CProgramming");
  printf("%10.8s\n", "CProgramming");
  return 0;
}
```

Output:

```
        CP
  CProgram
```

Formatted Input - scanf()

This function is used to get (input) value from the keyboard. We pass format specifiers, in which format we want to take input.

Syntax:

> scanf(“format-specifier”, &var-name);
>
> scanf(“fromat-specifier-list”, &var-name-list);

First type of scanf() takes the single value for the variable and second type of scanf() will take the multiple values for the variable list.

Consider the following examples:

```c
#include <stdio.h>

int main()
{
        int     a;
        float   b;
        char    c;

        printf("Enter an integer number (value of a)?:");
        scanf("%d",&a);

        printf("Enter a float number (value of b)?:");
        scanf("%f",&b);

        printf("Enter a character  (value of c)?:");
        fflush(stdin); // to flush (clear) input buffer
        scanf("%c",&c);

        printf("\na=%d,b=%f,c=%c",a,b,c);
        return 0;
}
```

Output

```
Enter an integer number (value of a)?:1234
Enter a float number (value of b)?:1.2345
Enter a character  (value of c)?:G

a=1234,b=1.234500,c=G
```

Consider the following examples to read multiple value in single scanf statement:

```c
#include <stdio.h>

int main()
{

        int     a;
        float   b;
        char    c;

        printf("\nEnter value of a,b,c (an integer, a float, a character):");
        scanf("%d%f%c",&a,&b,&c);

        printf("\na=%d,b=%f,c=%c",a,b,c);
        return 0;
}
```

Output

```
Enter value of a,b,c (an integer, a float, a character):1234 1.2345 G
a=1234,b=1.234500,c=
```

Here, G will not store into c variable, because we are not flushing input buffer here. So either you will have to take input of c first or you will have to read value of c separately.

% x d

Here percent sign (%) denotes that a specifier for conversion follows and **x** is an integer number which specifies the width of the field of the number that is being read. The data type character **d** indicates that the number should be read in integer mode.

Example :

```c
#include "stdio.h"

int main(void) {
  int i, j;
  scanf("%3d %4d", &i, &j);
  printf("%d %d", i, j);
  return 0;
}
```

Output

```
123456789
123 4567
```

Unformatted I/O

C provides many low level functions to read and write unformatted data. These functions are explained next.

getchar()

getchar() function will read a single character from the standard input. The return value of getchar() is the first character in the standard input.The input is read until the Enter key is pressed, but only the first character in the input will be returned.

putchar()

putchar() function will print a single character on standard output. The character to be printed is passed to putchar() function as an argument. The return value of putchar() is the character which was written to the output.
getchar() and putchar() functions are part of the standard C library header stdio.h

Example

```
#include <stdio.h>
int main()
{
      char ch;
      printf("Input some character and finish by pressing the Enter key.\n");
      ch = getchar();
      printf("The input character is ");
      putchar(ch);
      return 0;
}
```

Output

```
Input some character and finish by pressing the Enter key
apple
The input character is a
```

Please note that when you entered the word apple those characters were echoed on the screen. You also terminated the input by pressing the Enter key. However getchar() function took only the first character, which is "a" from the input and putchar() function printed it out.

String input and output

gets (str) where str is a string variable.

puts (str) where str is a string variable:

Example program

```
# include < stdio.h >
void main ( )
{
    char s [80];
```

```
    printf ("Type a string less than 80 characters:");
    gets (s);
    printf ("The string types is:");
    puts(s);
}
```

Sequence, Selection and looping

Sequence

The statements are executed one after another in a sequence of steps

Example :
A=10
B=20
C=a+b
After the above sequence of statements are executed the value of C is 30.

Selection or Decision making

The program flow is controlled by the selection or decision made depending on a condition.

IF statement:

```
# include <stdio.h>
void main( )
{
     int numbers;
     printf ("Type a number:");
     scanf ("%d", &number);
     if (number < 0)
        number = -number;
     printf ("The absolute value is %d \n", number);
}
```

The above program checks the value of the input number to see if it is less than zero. If it is then the following program statement which negates the value of the number is executed. If the value of the number is not less than zero, we do not want to negate it then this statement is automatically skipped. The absolute number is then displayed by the program, and program execution ends.

IF-ELSE statement :

```
#include "stdio.h"

int main(void) {
  int a, b, c;
  scanf("%d %d", &a, &b);

  if(a>b)
       c=a;
  else
       c=b;

  printf("The greatest: %d\n", c);
```

```
}
```

Output:

```
12 34
The greatest: 34
```

In the above program, two inputs a and b are given. If ‘a’ is greater than’b’, ‘c’ is assigned with ‘a’. Else, ‘c’ is assigned with ‘b’.Thus, the greatest number is assigned to c. Selection statements help to make alternative actions based on a condition.

Nested if-else statement

If-else statements that contain inner selection statements are known as nested statements.

```
#include "stdio.h"

int main(void) {
  int a, b, c, d;
  scanf("%d %d %d", &a, &b, &c);

  if(a > b)
  {
    if(a > c)
      d = a;
    else
      d = c;
  }
  else
    {
    if(b > c)
      d = b;
    else
      d = c;
  }

  printf("The greatest: %d\n", d);

}
```

Output

```
12 56 34
The greatest: 56
```

The Else-if ladder

When a series of many conditions have to be checked we may use the ladder else if statement which takes the following general form:

```
if (condition1)
  statement – 1;
else if (condition2)
  statement2;
```

```
else if (condition3)
  statement3;
else if (condition)
  statement n;
else
  default statement;
statement-x;
```

Example program

Using if else ladder to find the grade for the input mark.

mark	grade
91-100	O
81-90	A+
71-80	A
61-70	B+
50-60	B
< 50	RA

```c
#include "stdio.h"

int main(void) {
  int mark;
  scanf("%d", &mark);
  if (mark > 90) printf("O");
  else if (mark > 80) printf("A+");
  else if (mark > 70) printf("A");
  else if (mark > 60) printf("B+");
  else if (mark >= 50) printf("B");
  else printf("RA");
  return 0;
}
```

Output

```
56
B
```

<u>The Switch Statement:</u>

The switch statement allows a program to select one statement for execution out of a set of alternatives.
The general format of the Switch Statement is:

```
Switch (expression)
{

case label1:
```

```
        Block1;
        break;
case label2:
        Block2;
        break;
……….
……….
default:
        Default-block;
        break;
}
```

When the switch statement is executed the control expression is evaluated first and the value is compared with the case label values in the given order. If the label matches with the value of the expression then the control is transferred directly to the group of statements which follow the label. If none of the statements matches then the statement against the default is executed. The default statement is optional in switch statement in case if any default statement is not given and if none of the condition matches then no action takes place in this case the control transfers to the next statement of the if else statement.

Example: A program to perform arithmetic calculations

```
#include<stdio.h>
#include<math.h>
int main()
{
      int choice;
      int x, y;
      float z;
      printf("1 Addition\n");
      printf("2 Subtraction\n");
      printf("3 Multiplication\n");
      printf("4 Division\n");
      printf("5 Square root\n");
      printf("Enter choice:");
      scanf("%d", &choice);
      printf("Enter the input values\n");
      scanf("%d", &x);
      if(choice != 5)
            scanf("%d",&y);
      switch(choice)
      {
            case 1: z = x + y; break;
            case 2: z = x - y; break;
            case 3: z = x * y; break;
            case 4: z = (float) x / y; break;
            case 5: z = sqrt(x) ; break;
```

```
    }
    printf("%f", z);
}
```

INPUT

```
1       Addition
2       Subtraction
3       Multiplication
4       Division
5       Square root
Enter choice: 5
Enter the input values
 5
```

OUTPUT

```
2.236068
```

Repetition or Iteration or Looping

The loop statement repeats a particular block of statements for specific number of times or until a condition is met.

The following statements make iteration
1.for loop
2.while loop
3.do..while loop

for loop

The for loop provides a more concise loop control structure. The general form of the for loop is:

for (initialization; test condition; increment)
{
body of the loop
}

When the control enters for loop the variables used in for loop is initialized with the starting value such as I=0,count=0. The value which was initialized is then checked with the given test condition. The test condition is a relational expression, such as I < 5 that checks whether the given condition is satisfied or not if the given condition is satisfied the control enters the body of the loop or else it will exit the loop. The body of the loop is entered only if the test condition is satisfied and after the completion of the execution of the loop the control is transferred back to the increment part of the loop. The control variable is incremented using an assignment statement such as I=I+1 or simply I++ and the new value of the control variable is again tested to check whether it satisfies the loop condition. If the value of the control variable satisfies then the

body of the loop is again executed. The process goes on till the control variable fails to satisfy the condition.

Example

```
for(i=1; i<=10; i++)
 {
        Printf("%d x 5 =%d\n",i,i*5);
}
```

The above code snippet prints the multiplication table of 5

```
1 x 5 = 5
2 x 5 = 10
.
.
.
10 x 5 = 50
```

Nesting of for loops

Nested for loops consist of an **outer for loop** with one or more **inner for loops**.

For example:

```
for (i=1;i<=100;i++)
{
        for(j=1;j<=50;j++)
        {
                …
        }
}
```

The above loop will run for 100*50 iterations.

<u>while loop</u>

The general format of the while statement is:

```
while (test condition)
{

body of the loop

}
```

Here the given test condition is evaluated and if the condition is true then the body of the loop is executed. After the execution of the body, the test condition is once again evaluated and if it is true, the body is executed once again. This process of repeated execution of the body continues until the test condition finally becomes false and the control is transferred out of the loop. On exit, the program continues with the statements immediately after the body of the loop.

```
// Program to find whether given number is Armstrong or not.

#include<stdio.h>
void main(){
    int A;
    int sum = 0;
    int T;
    scanf("%d", &A);
    T = A;
    while (T > 0)
    {
        d = T % 10;
        sum += pow(d,3);
        T /= 10;
    }
    if ( sum == A)
        printf("Armstrong number");
    else
        printf("Not an armstrong number");
}
```

Output

371
Armstrong number

The do while statement:

The syntax of the do while loop is:

do
{

statement;

}
while(expression);

In contrast to while loop, the do-while loop tests at the bottom of the loop after executing the body. Since the body of the loop is executed first and then the loop condition is checked we can be assured that the body of the loop is executed at least once.

The Break Statement:

Sometimes while executing a loop it becomes desirable to skip a part of the loop or quit the loop as soon as certain condition occurs, for example consider searching a particular number in a set of 100 numbers as soon as the search number is found it is desirable to terminate the loop. C language permits a jump from one statement to another within a loop as well as to jump out of the loop. The break statement allows us to accomplish this task. A break statement provides an early exit from for, while, do and switch constructs. A break causes the innermost enclosing loop or switch to be exited immediately.

Example program to illustrate the use of break statement.

```
/* A program to find the average of the marks*/
#include < stdio.h >
void main()
{
    int I, num=0;
    float sum=0,average;
    printf("Input the marks, -1 to end\n");
    while(1)
    {
        scanf("%d",&I);
        if(I==-1)
        break;
        sum+=I;
        num++;
    }
}
```

Continue statement:

During loop operations it may be necessary to skip a part of the body of the loop under certain conditions. Like the break statement C supports similar statement called continue statement. The continue statement causes the loop to be continued with the next iteration after skipping any statement in between. The continue with the next iteration the format of the continue statement is simply:

continue;

Consider the following program that finds the sum of five positive integers. If a negative number is entered, the sum is not performed since the remaining part of the loop is skipped using continue statement.

```
#include < stdio.h >
void main()
{
      int I=1, num, sum=0;
      for (I = 0; I < 5; I++)
      {
```

```
            printf("Enter the integer");
            scanf("%I", &num);
            if(num < 0)
            {
                  printf("You have entered a negative number");
                  continue;
            }
            sum+=num;
      }
      printf("The sum of positive numbers entered = %d",sum);
}
```

exit function

We can jump out of the program by using the exit function. If we want to break out of the program for a reason and return to the operating system this exit() is used.

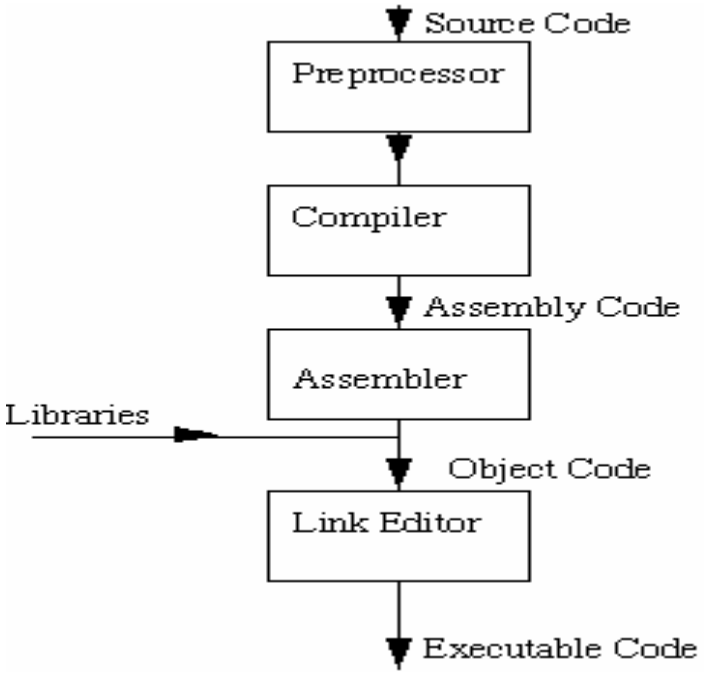
For Example:

if(condition) exit(0);

This function takes an integer value as its argument. Zero value is used to indicate normal termination, non zero value is used to indicate abnormal termination.

Compilation process

A compiler is used to translate a source code (.c) into a machine executable object code (.obj).
The linker links the runtime libraries and creates the executable file(.exe).
The executable file can be directly run on the computer system.



The four stages of compilation process is shown above.

Pre-processor
-includes content of header files
#include<stdio.h>
Includes standard library for I/O.
-Expands macros
#define SIZE 100
Defines symbolic name or constant and constitutes its occurrence.

Compiler
Translates source to assembly code(.s extension).

Assembler
The assembler creates the machine object code (.o extension)
[.OBJ in MSDOS]

Linker
If a source file references the library functions, the linker combines these functions to create an executable file.

Preprocessor directives

#include directive

Includes a copy of a specified file in place of the directive.
#include<stdio.h>
Includes standard library for I/O.

#define directive

Creates symbolic constants
#define PI 3.14159
Replaces all subsequent occurrences of symbolic constant PI with the numeric constant 3.14159

Macros

A macro is created with #define directive. The macro identifier is replaced in the program with the replacement text before the program is compiled.
#define AREA(X) (PI*X*X)
Wherever the macro AREA(X) appears it is replaced with the text PI*X*X
For example
a=AREA(10)
is expanded as
a=PI*10*10;
before compilation. Further the symbolic constants PI is also replaced with 3.14159 before compilation.

#undef directive

Undefines symbolic constant or a nacro(if defined elsewhere)
Once undefined a name can be redefined with# define.
Conditional pre processor
Conditional compilation is commonly used for debugging

```
#ifdef DEBUG
        printf("X=%d\n",X)
#endif
```

If the symbolic constant DEBUG has been defined before the #ifdef directive, the printf statement will be executed.

Example Programs

1.Find the sum of individual digits of a number.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int n,sum=0;
    printf("enter a +ve integer"); // enter a integer value
    scanf("%d",&n);
    while(n>0)  // checks the condition
    {
        sum=sum+n%10;   // sum + remainder value
        n=n/10;

    }
    printf("sum of individual digits of a positive integer is
%d",sum);  // prints the sum of individual digits
    getch();
}
```

Output:

```
Enter any number

1234

Sum of individual digits of a given number is 10
```

2.Check whether a number is prime or not

```
#include <stdio.h>
int main()
{
    int n, i, flag = 0;

    printf("Enter a positive integer: ");
    scanf("%d",&n);

    for(i=2; i<=n/2; ++i)
    {
        // condition for nonprime number
        if(n%i==0)
        {
            flag=1;
            break;
        }
    }

    if (flag==0)
        printf("%d is a prime number.",n);
```

```
    else
        printf("%d is not a prime number.",n);

    return 0;
}
```

Output

Enter a positive integer: 29

29 is a prime number.

3.Find the factorial of a number

```
#include <stdio.h>
int main()
{
    int n, i;
    unsigned long long factorial = 1;

    printf("Enter an integer: ");
    scanf("%d",&n);

    // show error if the user enters a negative integer
    if (n < 0)
        printf("Error! Factorial of a negative number doesn't
exist.");

    else
    {
        for(i=1; i<=n; ++i)
        {
            factorial *= i;              // factorial = factorial*i;
        }
        printf("Factorial of %d = %llu", n, factorial);
    }

    return 0;
}
```

Output

Enter an integer: 10

```
Factorial of 10 = 3628800
```

4.Generate Fibonacci series

```c
#include <stdio.h>
int main()
{
    int i, n, t1 = 0, t2 = 1, nextTerm;

    printf("Enter the number of terms: ");
    scanf("%d", &n);

    printf("Fibonacci Series: ");

    for (i = 1; i <= n; ++i)
    {
        printf("%d, ", t1);
        nextTerm = t1 + t2;
        t1 = t2;
        t2 = nextTerm;
    }
    return 0;
}
```

Output

```
Enter the number of terms: 10

Fibonacci Series: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34,
```

5.Checking leap year

```c
#include <stdio.h>

int main()
{
    int year;

    printf("Enter a year: ");
    scanf("%d",&year);

    if(year%4 == 0)
```

```
    {
        if( year%100 == 0)
        {
            // year is divisible by 400, hence the year is a leap year
            if ( year%400 == 0)
                printf("%d is a leap year.", year);
            else
                printf("%d is not a leap year.", year);
        }
        else
            printf("%d is a leap year.", year );
    }
    else
        printf("%d is not a leap year.", year);

    return 0;
}
```

Output 1

```
Enter a year: 1900

1900 is not a leap year.
```

Output 2

```
Enter a year: 2012

2012 is a leap year.
```

6. To solve quadratic equation

```
#include <stdio.h>
#include <math.h>

int main()
{
    double a, b, c, determinant, root1,root2, realPart, imaginaryPart;

    printf("Enter coefficients a, b and c: ");
    scanf("%lf %lf %lf",&a, &b, &c);

    determinant = b*b-4*a*c;

    // condition for real and different roots
    if (determinant > 0)
    {
    // sqrt() function returns square root
        root1 = (-b+sqrt(determinant))/(2*a);
        root2 = (-b-sqrt(determinant))/(2*a);

        printf("root1 = %.2lf and root2 = %.2lf",root1 , root2);
    }

    //condition for real and equal roots
    else if (determinant == 0)
    {
        root1 = root2 = -b/(2*a);

        printf("root1 = root2 = %.2lf;", root1);
    }

    // if roots are not real
    else
    {
        realPart = -b/(2*a);
        imaginaryPart = sqrt(-determinant)/(2*a);
        printf("root1 = %.2lf+%.2lfi and root2 = %.2f-%.2fi",
realPart, imaginaryPart, realPart, imaginaryPart);
    }

    return 0;
}
```

Output

```
Enter coefficients a, b and c: 2.3
```

4

5.6

Roots are: -0.87+1.30i and -0.87-1.30i

7. To check Armstrong number

```
#include <stdio.h>
int main()
{
    int number, originalNumber, remainder, result = 0;

    printf("Enter a three digit integer: ");
    scanf("%d", &number);

    originalNumber = number;

    while (originalNumber != 0)
    {
        remainder = originalNumber%10;
        result += remainder*remainder*remainder;
        originalNumber /= 10;
    }

    if(result == number)
        printf("%d is an Armstrong number.",number);
    else
        printf("%d is not an Armstrong number.",number);

    return 0;
}
```

Output

```
Enter a three digit integer: 371

371 is an Armstrong number.
```

Part A

2 marks

1 List the rules for defining variables in C

- It must begin with a letter
- Only letters, digits and underscore(_) are permitted to be used in an variable name
- No other special characters are allowed
- Case sensitive
- Keywords can not be used as variables

2 What are various types of C operators?

Primitive data types: char, int, float, double
Derived data types: arrays, pointers
User defined data types: structures, union

3 What is scope of a variable? List the two types of scopes .

The scope of a variable is the region of a program, only in which it is visible.
Two types of scope:
Local variable : Visible only in the block in which it is defined
Global variable: Visible across the entire program

4 Write any four escape sequences in C

\n	New Line
\t	Horizontal tab
\v	Vertical tab
\r	Carriage return

5 What are the uses of the following keywords:
auto, double, int, long

auto: This is the default storage class for all the variables declared inside a function or a block. The auto variable persist only within its scope.
double: It is used to store decimal numbers (numbers with floating point value) with double precision.
int: An int variable is used to store an integer and occupies 4 bytes (32 bits) in a 32 bit machine
long: A long variable stores an integer and occupies 8 bytes

6 Illustrate any bitwise operator with example

Example:
a = 4 << 2;

Operation	decimal value	binary representation
	4	0000 0000 0000 0100
after right shift 4 by 2	16	0000 0000 0001 0000

The ‘a’ will be assigned with ‘16’.

7 What is ternary operator?

The ternary operator is an operator that takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison.

Example:

```
c = a < b ?  a : b ;
```

The above statement assigns the minimum of ‘a’ and ‘b’ to ‘c’.

8 Differentiate between relational and logical expressions

Relational operators are used for comparison of the values of two operands.
Example:
6.5 <= 25 ➔ TRUE
-65 > 0 ➔ FALSE
10 < 7 + 5 ➔ TRUE
Operators are used to combine two or more conditions. The result of the operation of a logical operator is a boolean value either true or false.
Example:
6.5 <= 25 && -65 > 0 ➔ FALSE
6.5 <= 25 || -65 > 0 ➔ TRUE

9 List any four short hand assignment operators

Statement with simple assignment operator	**Statement with shorthand operator**
a = a + 1	a += 1
a = a – 1	a -= 1
a = a * (n+1)	a *= (n+1)

a = a / (n+1)	a /= (n+1)
a = a % b	a %= b

10 Differentiate between while and do.. while statements

In contrast to while loop, the do-while loop tests at the bottom of the loop after executing the body. Since the body of the loop is executed first and then the loop condition is checked we can be assured that the body of the loop is executed at least once.

`int i = 10;`
`while(i > 20)`
`{`
`  printf("not printed");`
`}`
	`int i = 10;`
	`do`
	`{`
	`  printf("printed once");`
	`}while(i > 20);`

No output	Output: printed once

11 What is operator precedence and associativity. Illustrate with example.

When an expression has more than one operator, they will be evaluated in an order defined by their precedence and associativity.

Precedence: denotes the priority (which operator gets evaluated first)

Associativity: defines the order for the operators with the same precedence or same operators (Left to right or right to left).

Example:

5 * 8 / 2 + 3

The expression is evaluated based on the precedence of the operators and associativity.

⇨ 40 / 2 + 3
⇨ 20 + 3
⇨ 23
The result after evaluating the given expression is 23.

12 Differentiate between break and continue statements.

A break statement provides an early exit from for, while, do and switch constructs.

The continue statement causes the loop to be continued with the next iteration after skipping any statement in between.

13 What are the various I/O functions in C

printf(), scanf(), getchar(), putchar(), gets(), puts()

14 What is modulo operator?

It returns the reminder after the dividing op1 by op2

example: 5 % 2 returns the reminder 1

15 Give the use of preprocessor

#include directive includes the specific header file to the program before compilation.
Expands macros
#define SIZE 100
Defines symbolic name or constant and constitutes its occurrence.

16 Define implicit type conversion

An **implicit conversion** of a **data** type occurs when the expression evaluator automatically converts the **data** from one **data** type to another. For example, if a small int is compared to an int, the small int is **implicitly converted** to int before the comparison is performed.

17 Define register storage class

The compiler tries to store these variables in the register of the microprocessor if a free register is available. The access of register variables are much faster than that of the variables stored in the memory during the runtime of the program.

18 What will be the output of the following code

```
#include "stdio.h"
int main(){
int a, b= 50, c;
a = 5 <= 8 && 6 != 5;
c =  b++ + ++b;
printf("a= %d, b= %d, c=%d \n", a, b, c);
}
```

Answer:

`5 <= 8`	1 (True)
`6 != 5`	1 (True)
`a = 5 <= 8 && 6 != 5`	a = 1 (True and True is True)
`c = b++ + ++b` ➔ `51 + 51`	b = 52 c = 102

Output:
a = 1, b = 52, c = 102