20CS1101 –PROGRAMMING FOR PROBLEM SOLVING

(Common to all branches)

UNIT–II

OPERATORS AND EXPRESSIONS: Introduction, Operator Precedence and Associativity, Operator Types. **INPUT AND OUTPUT IN C:** Formatted and Unformatted functions, Commonly used library functions.

→Operators and expressions:

Operator: Operator is a symbol, which operates one or more operands with a specific meaning. C has very rich in-built in operators. An operator tells the computer to perform some specific operation.

Example: sum=a+b where a and b are operands and + is an operator

Expression: The combination of operators and operand is called a expression.

Operators in C language: They are different types of operators in C language. These are

a)Arithmetic operators b) Assignment operators c) Conditional operators d) Relational operators

e) Logical operators f) Bitwise operators g) Increment and decrement operators(Unary operators)

h) Special operators.

a) Arithmetic operators: The arithmetic operators in C language are

operator	**Name**	**Purpose**
+	plus	Addition
-	Minus	Subtraction
*	asterisk	Multiplication
/	slash	Division(returns quotient)
%	modules	Division(return remainder)

Note:

Operator	**Precedence**
( )	First precedence
*	Second precedence
/ or %	Second precedence
+	Low precedence
-	Low precedence

/* Write a c program based on arithmetic operator */

```
#include<stdio.h>
#include<condio.h>
void main( )
  {
```

```
 printf("%d", 10+2);
  printf("%d", 10-2);
  printf("%d", 10*2);
  printf("%d", 10/2);
  printf("%d", 10%2);
getch();
}
```

Output: 12 8 20 5 0

/* write a c program to show precedence of the arithmetic operators */

```
#include<stdio.h>
#include<conio.h>
void main( )
{
 printf("%d", 2+3*2);
printf("%d", 6/6*3);
printf("%d",3+3/6);
printf(%d",(3+3)/6;
getch();
}
```

Output: 8 3 3 1

b) Assignment operators: Assignment operators are used to assign the result of an expression to a variable.

Example:

```
a) int     k;
       k=0;
b) int a, b, c;
   a=10;
   c=a=b;
```

Statement with simple assignment operator	**Statement with short hand 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
Table: short hand assignment operators	

Advantages of shorthand assignment operator:

1.What appears on the left-hand side need not be repeated and therefore it becomes easier to write.

2.The statement is more concise and easier to read.

3.The statement is more efficient.

/* Write a C program based on the assignment operator */

```
#include<stdio.h>
#include<conio.h>
void main( )
{
  int a,b,c;
a=10;
b=c=a;
a+=b+c;
printf("a=%d   b=%d    c=%d", a, b, c);
getch();
}
```

Output: a=30 b=10 c=10

/* Write a C program to prints a sequence of squares of numbers using shorthand operator */

```
#include<stdio.h>
#include<conio.h>
#define N  100
#define A 2
void main( )
 {
    int a;
   a=A;
  while(a<N)
   {
      printf("%d\n", a);
     a*=a;
    }
getch();
}
```

Output: 2 4 16

c) **Conditional operator [or] ternary operator:** The conditional operator? and: are sometimes called ternary operator. It operates on three operands, and it is a conditioned form of an if-else C statement.

Syntax: Exp1? Exp2 : Exp3

If Exp1 is true then Exp2 will be executed. If Exp1 is false then Exp3 will be executed.

Example: Write a c program to use the conditional operator with two statements

```
#include<stdio.h>

#include<conio.h>
void main( )
  {
    clrscr();
    3>2 ? printf("True") : printf("False");
    getch();
  }
```

Write a c program to check whether given num is zero or non zero using conditional operator or ternary operator.

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
     int num;
     clrscr();
     printf("enter the number");
    scanf("%d",&num);
   num? printf("non zero") : printf("zero");
    gethc();
}
```

Write a c program to find the large value among three given values using ternary operator

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
   int a, b, c, max;
   clrscr();
   printf("enter the a, b, c values");
   scanf("%d%d%d", &a,&b,&c);
  max= (a>b && a>c) ? a: ((b>c) ? b: c);
```

```
  printf("max=%d", max);
 getch();
}
```

d) **Relational operator:** The operators are used to compare arithmetic , logical and character expressions.

Operator	**Purpose**
= =	is equal to
! =	not equal
<	less than
>	grater than
< =	less than or equal
>=	greater than or equal

Write a c program show the activity of relational operators

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
    clrscr();
                printf("%d" , 2<3);
                printf("%d" , 2>3);
                printf("%d" , 2==3);
                printf("%d" , 2!=3);
                printf("%d" , 2<=3);
                printf("%d" , 2>=3);
                getch();
               }
```

E) Logical operator: These operators are used to compare or evaluate logical and relational expressions. The result of the logical expression will be either **true(1) or false (0).**

Operator	**Activity**
&&	AND
||	OR
!	Not

AND (&&): If both inputs are true then the output is true otherwise false.

A	B	A&&B
0	0	0
0	1	0
1	0	0
1	1	1

OR(| |): If both inputs are false then the output is false otherwise true.

A	B	A&&B
0	0	0
0	1	1
1	0	1
1	1	1

Not (!) : If the input is true then output is false. If the input is false then output is true

A	~A
0	1
1	0

Write a c program to display truth table of AND using && operator

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
    clrscr();
                printf(" 0&&0=%d\n" , 0&&0);
               printf(" 0&&1=%d\n" , 0&&1);
               printf(" 1&&0=%d\n" , 1&&0);
               printf(" 1&&1=%d\n" , 1&&1);
               getch( );
         }
```

Write a c program to display truth table of | | (OR)

```
 #include<stdio.h>
#include<conio.h>
void main( )
  {
   clrscr();
               printf(" 0||0=%d\n" , 0||0);
```

```
        printf(" 0||1=%d\n" , 0||1);
        printf(" 1||0=%d\n" , 1||0);
        printf(" 1||1=%d\n" , 1||1);
        getch( );
    }
```

Write a c program to display truth table of (!) not operator

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
    clrscr();
        printf(" !0=%d\n" , !0);
        printf(" !1=%d\n" , !1);
        getch( );
    }
```

F) Bitwise operator: These operators are used to operate with bits of data for complementing, testing, setting or shifting the bits of data in a byte or word.

Operator	Operator name	Purpose
~	complement	for 1's complement
>>	right shift	to move bits to right
<<	left shift	to move bits to left
&	bitwise AND	to reset (0) / compare bits
!	bitwise OR	to reser(1)/ compare bits
^	bitwise exclusive OR (XOR)	to clear the bits

Note: Bitwise operators work only with char and int data types and not useful for float, double, void or complex data types.

Write a c program for ~ (complement operator)

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
   char num=10;
   clrscr();
        printf("%d\n" , ~num);
        getch( );
    }
```

Write a c program for bitwise AND operator (&)

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
    char a=10;
   clrscr();
              printf(" %d\n" , a&2);
             getch( );
         }
```

Output: 2

Hint: Binary form of 10 = 0 0 0 1 0 1 0

Binary form of 2 = 0 0 0 0 0 0 1 0

AND operaration =0 0 0 0 0 0 1 0

Write a c program for bitwise OR operator (|)

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
    char a=10;
   clrscr();
              printf(" %d\n" , a|2);
             getch( );
         }
```

Output: 10

Hint: Binary form of 10 = 0 0 0 0 1 0 1 0

Binary form of 2 = 0 0 0 0 0 0 1 0

OR operaration = 0 0 0 0 1 0 1 0

Write a c program for bitwise XOR operator (^)

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
    char  a=10;
```

```
    clrscr();
                printf(" %d\n" , a^2);
               getch( );
          }
```

Write a c program to shift input data by two bits right (>>)

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
                int x, y;
               clrscr();
               printf("enter the number ");
               scanf("%d",&x);
                y=x>>2;
              printf(" the right shifted data is %d", y);
              getch( );
            }
```

Formula:- y=n/2s where n is number and s is number of position to be shifted.

Write a c program to shift input data by two bits right (<<)

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
                int x, y;
               clrscr();
               printf("enter the number ");
               scanf("%d",&x);
                y=x<<2;
              printf(" the right shifted data is %d", y);
              getch( );
            }
```

Formula:- y=n*2s where n is number and s is number of position to be shifted.

G) Increment and decrement operator: C has tow useful operators

1) ++ (increment operator)

2) – (decrement operator)

The ++ operator adds 1 to its operand. The – operator subtracts 1 from its operand.

These operators may be either pre increment/or pre decrement or post increment /post decrement.

Write a c program to show the effect of increment operator as a suffix.

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
                int a, z, x=10, y=20;
               clrscr();
               z= x*y++;
               a=x*y;
               printf(" %d %d", z,a);
               getch();
            }
```

Output: z=200 a=210

Note: z=x*y++ means first y value is assigned, later y will be incremented so z=200 and a=210

Write a c program to show the effect of increment operator as a prefix.

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
                int a, z, x=10, y=20;
               clrscr();
               z= x* ++y;
               a=x*y;
               printf(" %d %d", z,a);
               getch();
            }
```

Output: z=210 a=210

Write a c program to perform increment and decrement operator

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
                int , x=6, y=4,z;
               clrscr();
               z= ++x + y-- + x++ - x-- + y++ + --y +  ++x;
              printf("y= %d    z= %d     x= %d" y, z, x);
```

```
        getch();
    }
```

H) Special operators: The special operators are

1) sizeof ( ) operator

2) Comma operator (,)

3) Address of operator (&)

4) Value at address operator (*)

1) sizeof ( ) operator: the sizeof operator is used to obtain the size of a variable which occupies in system's memory. The sizeof() return value (in bytes) is the size of a variable or type with in the parenthesis.

Syntax: sizeof (object);

Where object may be any data type, variable or expression.

Example:

```
int  n1;
n1=  sizeof(int);
```

Write a c program on sizeof( ) operator

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
        int x;
        float y;
        clrscr();
printf("\n sizeof(x) =%d bytes", size(x));
printf("\n sizeof(y) =%d bytes", size(y));
printf("\n address of x =% u and y=%u", &x,&y);
getch();
}
```

2) Comma operator: The comma operator separates the expressions

Syntax: exp1, exp2,…..... expn;

Example:

```
#include<stdio.h>
#include<conio.h>
void main( )
  {
        printf("addition=%d\n sub=%d", 2+3, 5-4);
        getch( );
    }
```

3) Address of operator(&): This operator is used to find the address of a variable of any data type.

Example: m=&n; here address of n is assigned to m. This m is not a ordinary variable, it is a variable which holds the address of the other variable.

4) Value at address (*): This operator is used to find the value at a particular memory location. This operator is also called as indirection operator or dereferencing operator.

→Precedence of operators

All arithmetic expressions may contain two or more operators, then we may have some problem about how to get it exactly executed. To answer those questions one has to understand the precedence of operators. The order of priority in which the operations are performed in an expression is called **precedence.** The precedence of all operators is shown below.

Description	**Operator**	**Rank**	**Associativity**
Function expression	( )	1	Left to right
Array expression	[ ]	1	Left to right
Unary plus	+	2	Right to left
Unary minus	-	2	Right to left
Increment/decrement	++/--	2	Right to left
Logical negation	!	2	Right to left
One's complement	~	2	Right to left
Pointer reference	*	2	Right to left
Address of	&	2	Right to left
Sizeof an object	Sizeof	2	Right to left
Type cast(conversion)	(type)	2	Right to left
Multiplication	*	3	Left to right
Division	/	3	Left to right
Modulus	%	3	Left to right
Addition	+	4	Left to right

Subtraction	-	4	Left to right
Left shift	<<	5	Left to right
Right shift	>>	5	Left to right
Less than	<	6	Left to right
Less than or equal to	<=	6	Left to right
Greater than	>	6	Left to right
Greater than or equal to	>=	6	Left to right
Equality	= =	7	Left to right
Not equal to	!=	7	Left to right
Bitwise AND	&	8	Left to right
Bitwise XOR	^	9	Left to right
Bitwise OR	|	10	Left to right
Logical AND	&&	11	Left to right
Logical OR	||	12	Left to right
Conditional	?:	13	Right to left
Assignment	=	14	Right to left
Comma operator	,	15	Left to right

→→Formatted and Unformatted functions

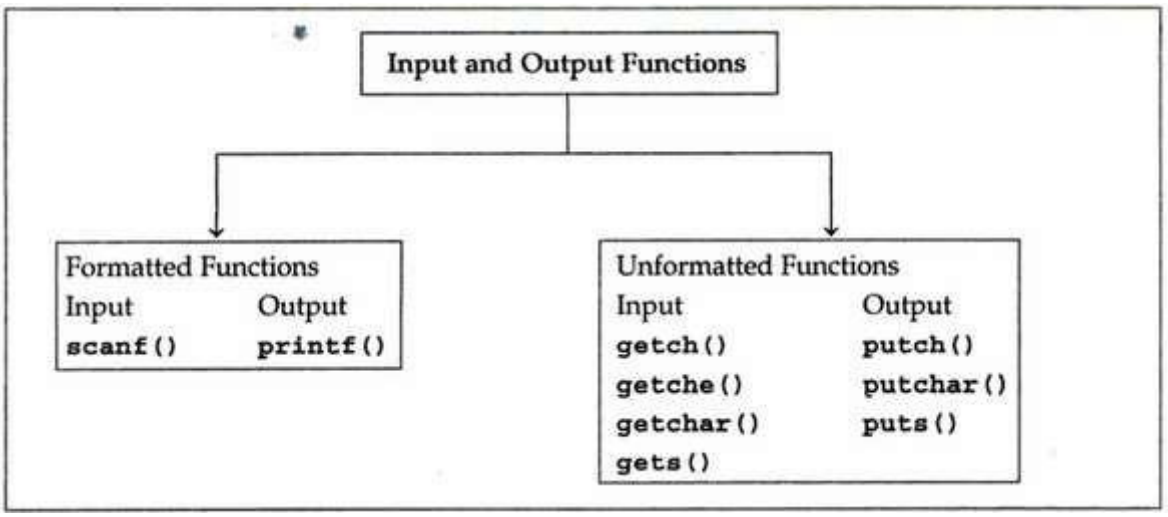
Introduction

Reading the data from the input device and displaying the results on the screen, are the two main tasks of any program. To perform these tasks user friendly C has a number of input and output functions. When a program needs data, it takes the data through the input functions and sends results obtained through the output functions.

There are number of I/O functions in C based on the data types. The input/output functions are classified into two types

1) Formatted functions: The formatted input/output functions read and write all types of data values. They require conversion symbol to identify the data type.

2) Unformatted functions: The unformatted input/out functions only work with the character data type. They do not require conversion symbol for identification of data types because they work only with character data type.



Formatted functions

a) The printf ( ) statement

The printf ( ) function prints all types of data values to the console. It requires conversion symbol and variable names to print the data. The conversion symbol and variable names should be same in number. The example of print ( ) statement is given below.

Example

```
main()
{
  int x=2;
  float y=2.2;
  char z='C';
  printf("%d  %f  %c",x,y,z);
}
```

OUTPUT:

2 2.2000 C

b) The scanf ( ) statement

The scanf ( ) statement reads all types of data values. It is used for runtime assignment of variables. The scanf ( ) statement also requires conversion symbol to identify the data to be read during the execution of a program.

Syntax

Scanf ("%d %f %c", &a, &b, &c);

4.1 Write a program to show the effect of mismatch of data types.

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

void main()
{
      int  a;
      clrscr();
      printf ("Enter value of 'A' : ");
      scanf ("%c",&a);
      printf ("A=%c",a);
}
```

OUTPUT:

```
Enter value of 'A' : 8
A=8
```

Unformatted functions

C has three types of I/O functions.

A) Character I/O

B) String I/O

A) Character I/O

The getchar() and putchar() Functions

The **int getchar(void)** This function reads character type data from the standard input. It reads one character at a time till the user presses the enter key.

The **int putchar(int c)** function puts the passed character on the screen and returns the same character. This function puts only single character at a time.

```
#include <stdio.h>
int main( )
{
   int c;
   printf( "Enter a value :");
   c = getchar( );
   printf( "\nYou entered: ");
   putchar( c );
   return 0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads only a single character and displays it as follows −

```
$./a.out
```

Enter a value : this is test

You entered: t

The getch ( ) and getche( )

These functions read any alphanumeric characters from the standard input device. The character entered is not displayed by getch ( ) function.

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

void main()
{
clrscr();
printf("Enter any two alphabetics ");
getche();
getch();
}
```

OUTPUT:

```
Enter any two alphabetics  A
```

The putch( )

This function prints any alphanumeric character taken by the standard input device.

```
main()
{
char ch;
clrscr();
printf("Press any key to continue ");
ch=getch();
printf ("\n You Pressed : ");
putch(ch);
}
```

OUTPUT:

```
Press any key to continue
You Pressed : 9
```

B) String I/O

3.The gets() and puts() Functions

The gets( )

This function is used for accepting any string through stdin(keyboard) until enter key is pressed. The header file stdio.h is needed for implementing the above function.

The puts( )

This function prints the string or character array.

```
#include <stdio.h>
int main( ) {

  char str[100];
  printf( "Enter a value :");
  gets( str );
  printf( "\nYou entered: ");
```

```
   puts( str );
   return 0;
}
```

When the above code is compiled and executed, it waits for you to input some text. When you enter a text and press enter, then the program proceeds and reads the complete line till end, and displays it as follows −

$./a.out

Enter a value : this is test

You entered: this is test

→Commonly Used Library Functions

1.clrscr ( ) This function is used to clear the screen. It clears the previous output from the screen and displays the output of the current program from the first line of the screen. It is defined in **conio.h** header file. The syntax is as follows.

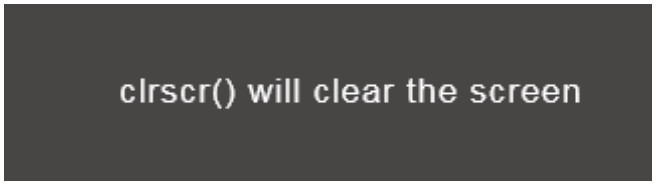
Syntax

clrscr( );

clrscr() Function Program

```
#include <stdio.h> //header file section
#include <conio.h>
int main()
{
printf("Before clrscr");
clrscr();
printf("clrscr() will clear the screen");
return 0;
}
```

Out put



2 exit ( ) This function terminated the program. It is defined in **process** header file. The syntax is as follows

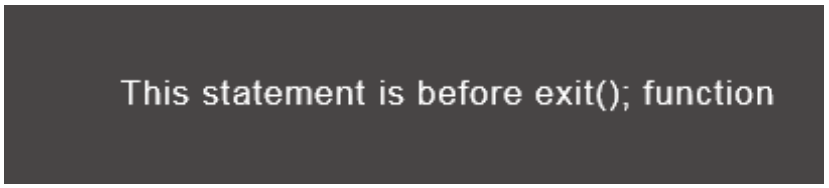
Syntax

exit ( );

exit() **Function Program**

```c
#include <stdio.h> //header file section
#include <stdlib.h>
int main()
{
printf("This statement is before exit(); function ");
exit(0);
printf("It will not display ");
return 0;
}
```

Output



3. sleep ( ) This function paused the execution of the program for a given number of seconds. The number of seconds is to be enclosed between the parenthesis. It is defined in **dos.h** header file. The syntax is as follows.

Syntax

Sleep(1);

sleep() Function Program

```c
#include <stdio.h>
#include <unistd.h>
int main()
{
printf("Countdown... ");
printf("\n 3");
sleep(1);
printf("\n 2");
sleep(1);
printf("\n 1");
sleep(1);
printf("\n Celebration Time ");
return 0;
}
```

Output

4.system ( ) This function is helpful in executing the different **dos commands.** It returns 0 on success and -1 on failure. The syntax is as follow.

Syntax

System ("dir");

The command should be enclosed within the double quotation marks.