Department of Electronic Engineering
NED University of Engineering & Technology

LABORATORY WORKBOOK
For the Course

PROGRAMMING LANGUAGES

(TC-105)

Instructor Name:	
Student Name:	
Roll Number:	**Batch:**
Semester:	**Year:**
Department:	

LABORATORY WORK BOOK

For The Course

TC-103 PROGRAMMING WITH C-LANGUAGE

Prepared By:

Danish Mahmood Khan (Sr. Laboratory Engineer)

Revised By:

Engr. Shafaq Mustafa (Lecturer)

Reviewed By

Dr. Aamir Zeb (Assistant Professor)

Approved By:

Board of Studies of Department of Electronic Engineering

INTRODUCTION

C is an imperative (procedural) systems implementation language. It was designed to be compiled using a relatively straightforward compiler to provide low-level access to memory; language constructs that map efficiently to machine instructions, and to require minimal run- time support. C was therefore useful for many applications that had formerly been coded in assembly language.

Despite its low-level capabilities, the language was designed to encourage cross-platform programming. A standards-compliant and portably written C program can be compiled for a very wide variety of computer platforms and operating systems with few changes to its source code. The language has become available on a very wide range of platforms, from embedded microcontrollers to supercomputers.

The Practical Workbook for “Programming with C-Language” introduces the basic as well as advanced concepts of programming using C language. C has been selected for this purpose because it encompasses the characteristics of both the high level languages (that give better programming efficiency and faster program development) and the low level languages (which have a better machine efficiency).

Each practical in this workbook contains syntax of statements/commands. Also, in order to facilitate the students, some programs have been provided explaining the use of these commands. For a wider scope of usage of the commands exercises are also given so that the students can understand how to use these commands in actual programming.

Note:
Various Contents of this work book have been taken from Internet as well as from the “Programming Language” manuals of Department of Computer and Information Systems, Department Of Electrical Engineering and Department of Electronic Engineering at NED University of Engineering and Technology.

CONTENTS

LAB SESSION 01

To explore Turbo C IDE and Programming Environment

Student Name: ______________________________

Roll Number: ____________________ **Batch:** ______________

Semester: ______________________ **Year:** ______________

Total Marks	
Marks Obtained	

Remarks (If Any): ______________________________

Instructor Name: ______________________________

Instructor Signature: ____________________ **Date:** __________

LAB SESSION 01

OBJECTIVE:
Introduction of Turbo C IDE and Programming Environment

THEORY:

The Development Environment - Integrated Development Environment (IDE)

The Turbo C compiler has its own built-in text editor. The files you create with text editor are called source files, and for C++ they typically are named with the extension .CPP, .CP, or .C.

The C Developing Environment, also called as Programmer's Platform, is a screen display with windows and pull-down menus. The program listing, error messages and other information are displayed in separate windows. The menus may be used to invoke all the operations necessary to develop the program, including editing, compiling, linking, and debugging and program execution.

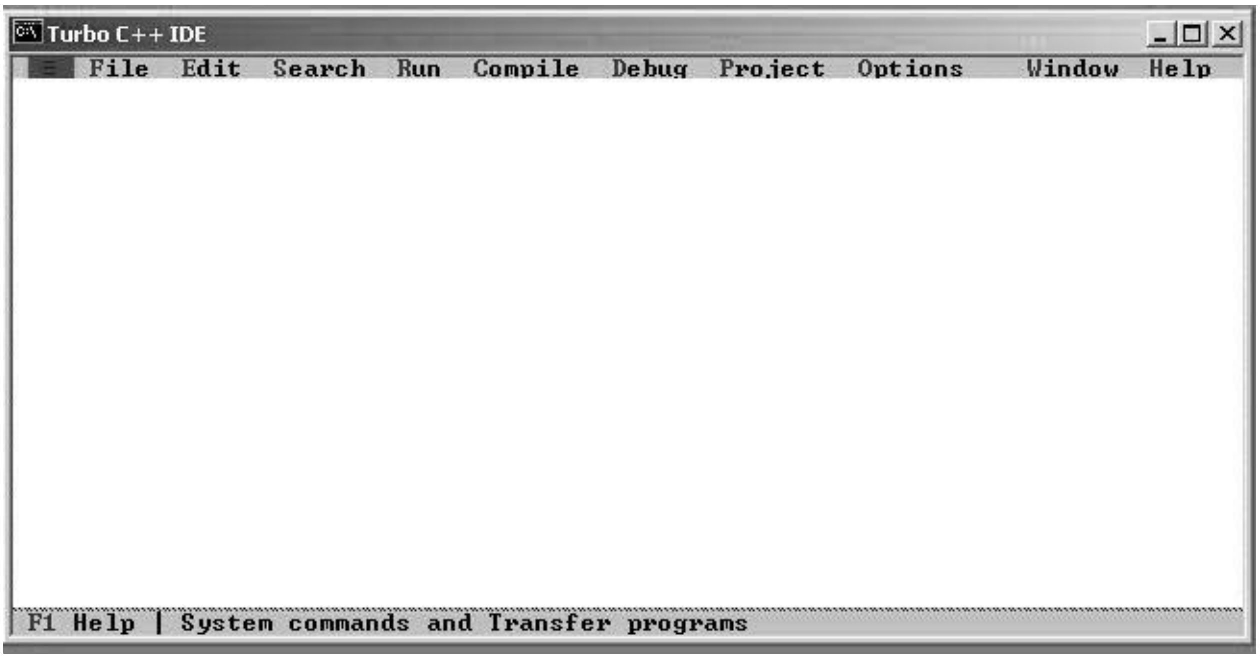

Turbo C++ IDE
≡ File Edit Search Run Compile Debug Project Options Window Help
F1 Help | System commands and Transfer programs

Figure 1.1: TURBO C IDE Environment

Invoking the IDE

To invoke the IDE from the windows you need to double click the TC icon in the directory c:\tc\bin. The alternate approach is that we can make a shortcut of tc.exe on the desktop. This makes you enter the IDE interface, which initially displays only a menu bar at the top of the screen and a status line below will appear. The menu bar displays the menu names and the status line tells what various function keys will do.

Default Directory

The default directory of Turbo C compiler is c:\tc\bin.

Using Menus

If the menu bar is inactive, it may be invoked by pressing the [F10] function key. To select different menu, move the highlight left or right with cursor (arrow) keys. You can also revoke the selection by pressing the key combination for the specific menu.

Opening New Window

To type a program, you need to open an Edit Window. For this, open file menu and click "new". A window will appear on the screen where the program may be typed.

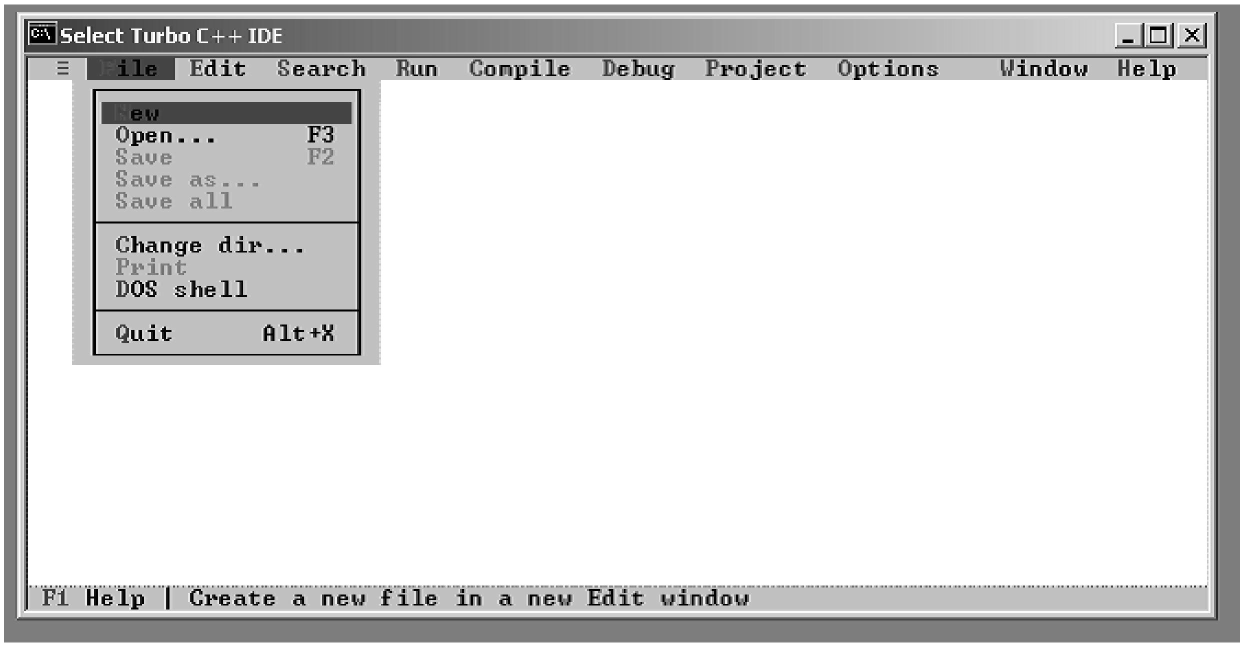

Select Turbo C++ IDE
≡ File Edit Search Run Compile Debug Project Options Window Help
New
Open... F3
Save F2
Save as...
Save all
Change dir...
Print
DOS shell
Quit Alt+X
F1 Help | Create a new file in a new Edit window

Figure 1.2: Opening new window

Implementing a Simple C Program

```
                    /* Basic C Program */
                        /*Hello.CPP*/
#include<conio.h>
#include<stdio.h>
void main (void)    /* Defining main function*/
{
clrscr(); /* Clears previous contents of screen*/
printf( "\n\n Hello World……"); /*Prints the string
argument on the screen*/
getch();  /* Hold the output screen until a key is hit*/
}
```

Saving a Program

To save the program, select **save** command from the **file** menu. This function can also be performed by pressing the [F2] button. A dialog box will appear asking for the path and name of the file. Provide an appropriate and unique file name. You can save the program after compiling too but saving it before compilation is more appropriate.

Making an Executable File

The source file is required to be turned into an executable file. This is called "Making" of the .exe file. The steps required to create an executable file are:

1. Create a source file, with a .c extension.
2. Compile the source code into a file with the .obj extension.
3. Link your .obj file with any needed libraries to produce an executable program.

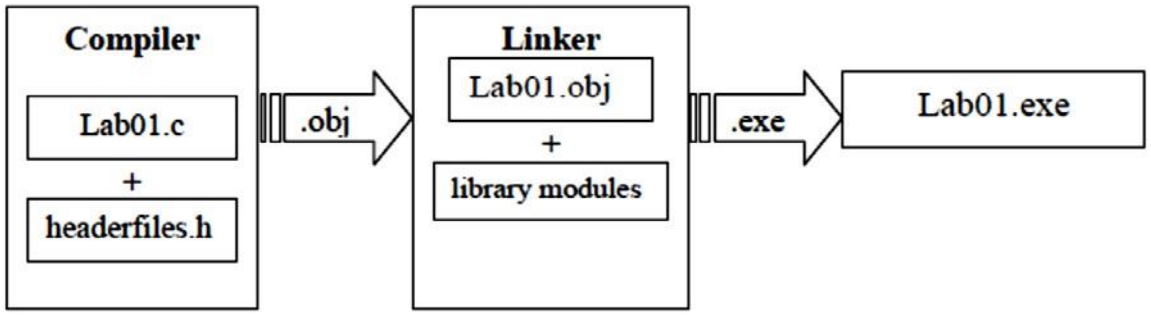


Figure 1.3: Making an executable file

All the above steps can be done by using Run option from the menu bar or using key combination Ctrl+F9 (By this linking & compiling is done in one step).

Compiling the Source Code

Although the source code in your file is somewhat cryptic, and anyone who doesn't know C will struggle to understand what it is for, it is still in what we call human-readable form. But, for the computer to understand this source c ode, it must be converted into machine-readable form. This is done by using a compiler. Hence, compiling is the process in which source code is translated into machine understandable language.

It can be done by selecting Compile option from menu bar or using key combination Alt+F9.

Creating an Executable File with the Linker

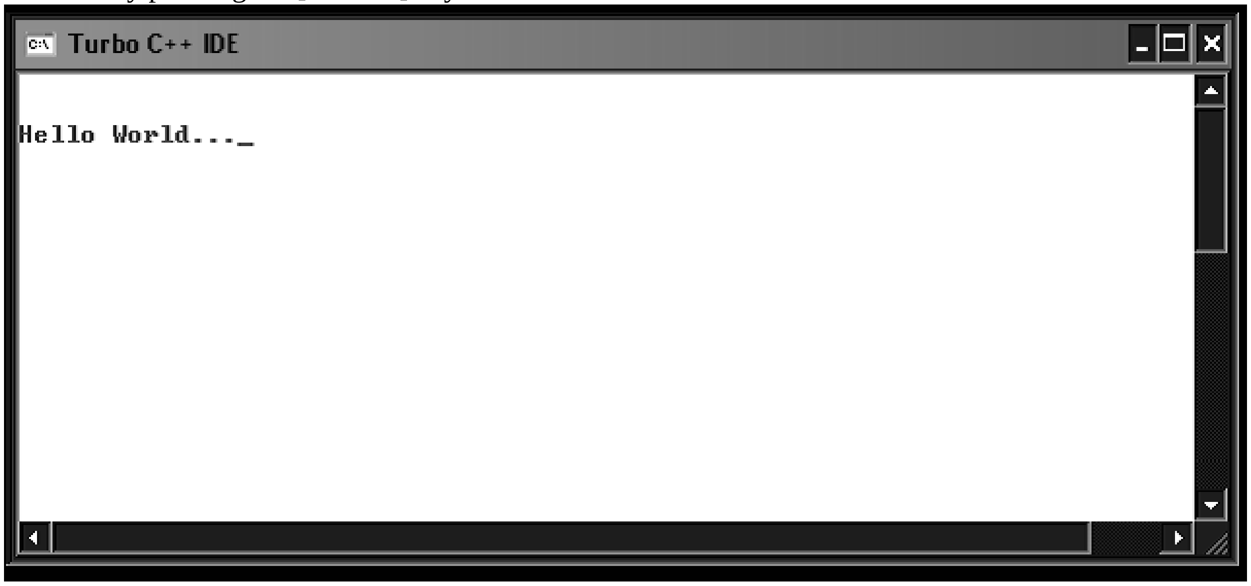
After your source code is compiled, an object file is produced. This file is often named with the extension .OBJ. This is still not an executable program, however. To turn this into an executable program, you must run your linker. C programs are typically created by linking together one or more OBJ files with one or more libraries. A library is a collection of linkable files that were supplied with your compiler.

Compiling and linking in the IDE

In the Turbo C IDE, compiling and linking can be performed together in one step. There are two ways to do this: you can select Make EXE from the compile menu or you can press the [F9] key.

Executing a Program

If the program is compiled and linked without errors, the program is executed by selecting Run from the Run Menu or by pressing the [Ctrl+F9] key combination.



```
Hello World..._
```

Figure 1.4: Executing a program

The Development Cycle

If every program worked the first time you tried it that would be the complete development cycle: Write the program, compile the source code, link the program, and run it.
Unfortunately, almost every program, no matter how trivial, can and will have errors, or bugs, in the program. Some bugs will cause the compile to fail, some will cause the link to fail, and some will only show up when you run the program.

Whatever type of bug you find, you must fix it, and that involves editing your source code, recompiling and re-linking, and then re-running the program.

Correcting Errors

If the compiler recognizes some error, it will let you know through the Compiler window. You'll see that the number of errors is not listed as 0, and the word “Error” appears instead of the word “Success” at the bottom of the window. The errors are to be removed by returning to the edit window. Usually these errors are a result of a typing mistake. The compiler will not only tell you what you did wrong; they'll point you to the exact place in your code where you made the mistake.

Exiting IDE

An Edit window may be closed in a number of different ways. You can click on the small square
in the upper left corner, you can select **close** from the window menu, or you can press the [Alt][F3] combination. To exit from the IDE, select **Exit** from the **File** Menu or press [Alt][X] Combination.

LABORATORY TASKS:

1. Type the following program in C Editor and execute it. Mention the Error.

```
void main(void)
{
printf("  This is my first program in C   ");
}
```

2. Add the following line at the beginning of the above program. Recompile the program. What is the output?

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

3. Make the following changes to the program. What Errors are observed?

i. Write Void instead of void.

ii. Write void main (void);

iii. Remove the semi colon ';'

iv. Erase any one of brace '{' or '}'

RESULT:

LAB SESSION 02

To study basic building blocks of C-language such as data types and input-output functions

Student Name: ____________________________________

Roll Number: ____________________ **Batch:** _______________

Semester: ______________________ **Year:** _______________

Total Marks	
Marks Obtained	

Remarks (If Any): ________________________________

Instructor Name: ____________________________________

Instructor Signature: __________________ **Date:** ___________

LAB SESSION 02

OBJETIVE:

To study basic building blocks of C-language such as data types and input-output functions

THEORY:

This Lab is concerned with the basic elements used to construct C elements. These elements includes the C character set, identifiers, keywords, data types, constants, variables, expressions statements and escape sequences.

Comments:

Comments statements will not to be compiled. Comments are simply the statements to improve program readability and to document program properly. Comments begins with /* and end with */, text is placed in between them.

/* Lab Session 2 */

printf() Function:

This function is used to output combination of numerical values, single character and strings.

Syntax:-

printf(“fomat specifier” , variable or constant); printf(“text “);

Example:-

printf(“Area of circle is %f sqmm” ,3.756);

scanf() Function:

The purpose of scanf() function is to except data from keyboard, and place that data to a memory location specified in its argument.

Syntax:-

scanf(“fomat specifiers” , address of variable);

Examples:-

scanf(“ %d” , &r);

Escape Sequences:

These are non printing characters. They are special character set, each with specific meaning.
An escape sequence always begins with a back slash and is followed by one or more special characters.

Escape Sequence	Meaning
\n	New Line
\t	Horizontal Tab
\a	Alert(Bell)
\\	Back Slash
\”	Quotation Mark
\f	Form feed
\r	Carriage Return
\0	Null

Table 2.1: Escape sequences

Variables:

A variable name is a location in memory where a value can be stored for use by a program. All variables must be defined with a name and a data type in the code before they can be used in a program.

A variable name in C should be a valid identifier. An identifier is a series of characters consisting of letters, digits and underscore and does not begin with a digit. C is case sensitive i.e. area and Area can’t be treated as same.

There are certain reserved words called **Keywords** that have standard, predefined meanings in C. These keywords can be used only for their intended purpose; they can’t be used as programmer defined identifier.

Data Types:

C supports several different types of data, each of which may be represented differently within the computer’s memory. The basic data types are listed below in Table 2.2.

Data Type	Meaning	Bytes
char	Character	1
int	Integer	2
short	Short Integer	2
long	Long Integer	4
Unsigned	Unsigned Integer	2
Float	Floating Number(Decimal)	4
Double	Double Precision Floating Point Number	8

Table 2.2: Data type and storage allocation

Format Specifiers:

Format specifier specifies that which type of data has to be print or read into. Following is a list of different format specifiers in table 2.3.

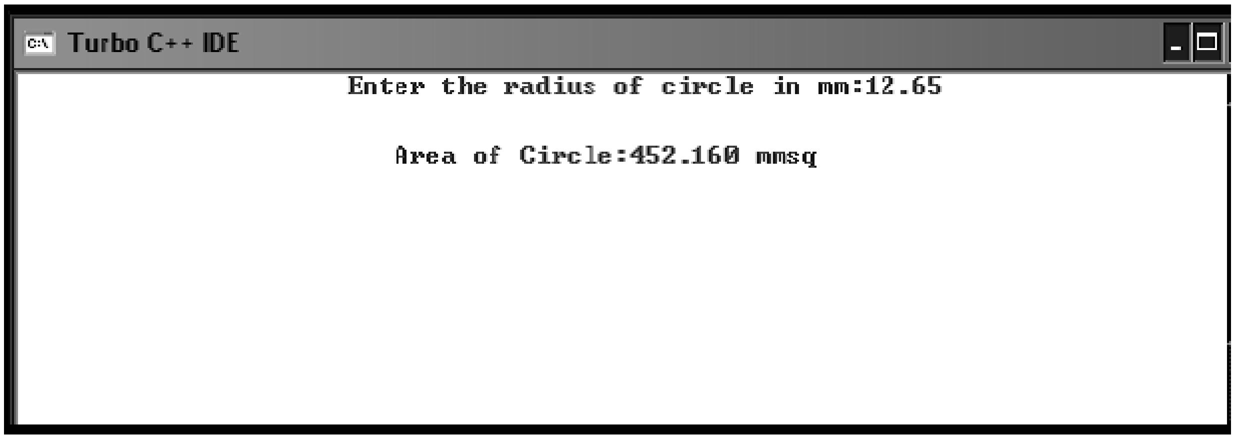
Specifiers	Meaning
%c	Character
%d	Integer
%f	Float value
%e	Float value in exponential form
%u	Unsigned Integer
%x	Hexadecimal integer (unsigned)
%o	Octal value
%s	String

Table 2.3: Format specifiers

Example:

```
#include<conio.h>
#include<stdio.h>
void main (void)             /* Defining main function*/
{
clrscr();          /* Clears previous contents of screen*/
int r;            /* Declares a variable of type integer */
float area;       /* Declares a variable of type float */
float Pi=3.14;     /*Initializing a variable of type float */
printf("\t\t\tEnter the radius of circle:");       /*Output the
string on the screen*/
scanf("%d", &r); /*Stores the value on the address of variable
r*/
area=Pi*r*r;      /*Evaluating area and assigning the value to
variable area*/
printf("\n\n\t\t\tArea of Circle is :%0.3f ",area);
getch();
}
```

Output:



```
                    Enter the radius of circle in mm:12.65

                       Area of Circle:452.160 mmsq
```

Figure 2.1: Output

LABORATORY TASKS:

1) Identify and correct the errors in the following statements.

a) scanf(“d “,value);

b) printf(“ The answer of %d+%d is “\n,x,y);

c) scanf(“ %d%d” ,&number1,number2);

d) printf(“The number is %d /n” ,&number1);

2) Write a single C statement to accomplish the following tasks.

a) Prompt the user to enter an integer in inverted commas.
Like this “Enter an integer: “

b) Read an integer from the keyboard and store the value entered in the variable **a**

c) Read two integers from the keyboard and store the value entered in the variable a & b .

3) What do these codes print?

a) printf(“ \n*\n**\n***\n****\n*****”);

b) printf(“This is\base”);

c) printf(“\n\t\t\t1\n\t\t2\n\t3\n4\n\t5\n\t\t6\n\t\t\t7”);

RESULT:

LAB SESSION 03

To study the different types of arithmetic and logical operators

Student Name: ______________________________

Roll Number: ____________________ **Batch:** ______________

Semester: ______________________ **Year:** ______________

Total Marks	
Marks Obtained	

Remarks (If Any): ______________________________

Instructor Name: ______________________________

Instructor Signature: __________________ **Date:** __________

LAB SESSION 03

OBJECTIVE:

To study the different types of arithmetic and logical operators

THEORY:

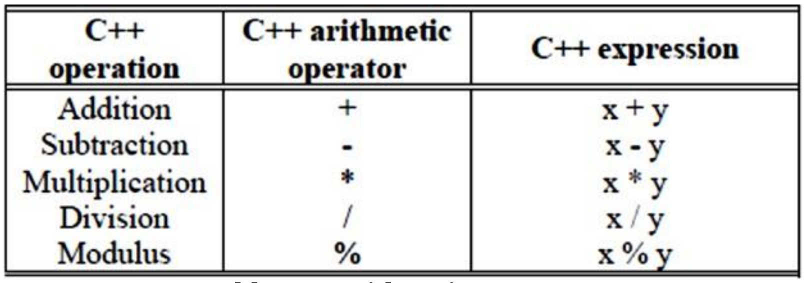
In C, there are various operators, used to form expressions. The data items on which the operators act upon are called operands. Some operators require two operands while other act upon only one operand.

They are classified as:

1. Arithmetic Operators (binary type)
2. Unary Operators
3. Relational and Logical Operators
4. Assignment Operator

Arithmetic Operators:

In C, most programs perform arithmetic calculations. Arithmetic calculations can be performed by using the following arithmetic operators. Table 3.1 summarizes the C arithmetic operators. Note the use of various special symbols not used in algebra. The asterisk (*) indicates multiplication and the percent sign (%) is the modulus or remainder operator. The arithmetic operators in the Table are all binary operators, i.e., operators that take two operands.

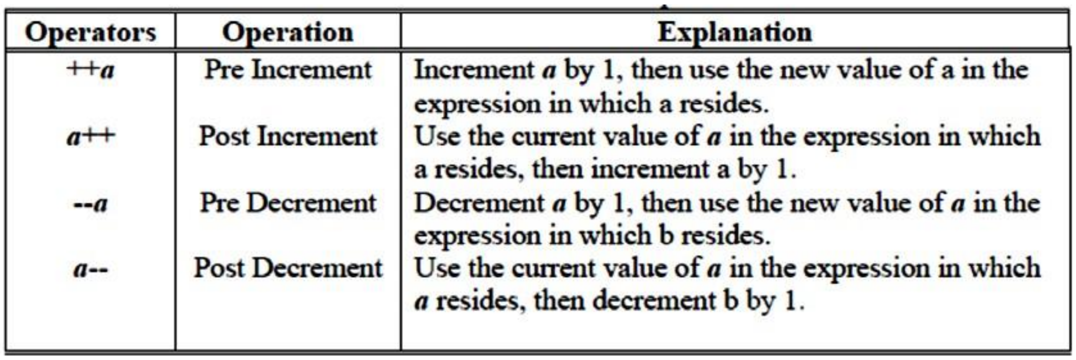


C++ operation	C++ arithmetic operator	C++ expression
Addition	+	x + y
Subtraction	-	x - y
Multiplication	*	x * y
Division	/	x / y
Modulus	%	x % y

Table 3.1: Arithmetic operators

Unary Operators:

In addition to the arithmetic assignment operators, C++ also provides two unary operators that act upon on a single operand to produce a new value, for adding 1 to or subtracting 1 from the value of a numeric variable. These are the unary increment operator, ++, and the unary decrement operator, --, which are summarized in the Table 3.2.



Operators	Operation	Explanation
++*a*	Pre Increment	Increment *a* by 1, then use the new value of a in the expression in which a resides.
a++	Post Increment	Use the current value of *a* in the expression in which a resides, then increment a by 1.
--*a*	Pre Decrement	Decrement *a* by 1, then use the new value of *a* in the expression in which b resides.
a--	Post Decrement	Use the current value of *a* in the expression in which *a* resides, then decrement b by 1.

Table 3.2: Increment and decrement operators

Assignment Operators:

C++ provides several assignment operators for abbreviating assignment expressions. For example, the statement: c = c + 3;
can be abbreviated with the addition assignment operator += as

c += 3;

The += operator adds the value of the expression on the right of the operator to the value of the variable on the left of the operator and stores the result in the variable on the left of the operator. Thus the assignment c += 3 adds 3 to c. Table 3.3 shows the arithmetic assignment operators, sample expressions using these operators and explanations.

Assignment operator	Sample expression	Explanation	Assigns
Assume: int c = 3, d = 5, e = 4, f = 6, g = 12;			
Subtraction -=	d -= 4	d = d - 4	1 to d
Multiplication *=	e *= 5	e = e * 5	20 to e
Division /=	f /= 3	f = f / 3	2 to f
Remainder %=	g %= 9	g = g % 9	3 to g

Table 3.3: Arithmetic assignment operators

Summary of Operator Precedence and Associativity:

Table 3.4 adds the logical operators to the operator precedence and associativity chart. The operators are shown from top to bottom, in decreasing order of precedence.

Operators						Associativity	Type
()						left to right	parentheses
++	--					left to right	unary (postfix)
++	--	+	-	!		right to left	unary (prefix)
*	/	%				left to right	multiplicative
+	-					left to right	additive
<<	>>					left to right	insertion/extraction
<	<=	>	>=			left to right	relational
==	!=					left to right	equality
&&						left to right	logical AND
||						left to right	logical OR
?:						right to left	conditional
=	+=	-=	*=	/=	%=	right to left	assignment

Table 3.4: Operator precedence and associativity

LABORATORY TASKS:

1) Identify and correct the errors in the following statements.

a) if (c<7);
printf(C is less than 7\n);

____

____

b) if (c =>7);
printf(C is equal to or less than 7\n);

____

____

c)printf(Remainder of %d divided by %d is \n , x , y , x % y);

____

____

d) num1+num2=ans;

____

____

2) a. Evaluate the following.
1) 9.0/6.0 +5/2 =________________________
2) 9*3/4 =______________
3) 14%7 +3%4 =_________________________

b. Determine the value assigned to the relevant variable. int a;
loat b;

1) b = 5/4 ;	b = ________
2) a = 5/4 ;	a = ________
3) b = 5/2 +3.0;	b = ________

c. Determine the value of int x after each statement. Initially x =5.

I. printf(%d\n , x); Ans: x = ______ printf(%d\n , ++x); Ans: x = ___
printf(%d\n , x++); Ans: x = ______ printf(%d\n , x); Ans: x = ______

II. printf(%d\n , x); Ans: x = _________
printf(%d\n , --x); Ans: x = _________
printf(%d\n , x--); Ans: x = _________
printf(%d\n , x); Ans: x = _________

3) State the order of evaluation of the operators in each of the following C statements and show the value of x after each statement is performed.

a) x = 7 + 3 * 6 / 2 1;
b) x = 2 % 2 + 2 * 2 -2 / 2;
c) x = (3 * 9 * (3 + (9 * 3 / (3)))) ;

Answer:

a)_________
b)_________
c)_________

4) Write a program that asks the user to enter two numbers, obtain the two numbers from the user and print the sum, difference, quotient and remainder of the two.

RESULT:

LAB SESSION 04

To apply decision making

Student Name: ______________________________

Roll Number: ________________ **Batch:** ____________

Semester: ________________ **Year:** ____________

Total Marks	
Marks Obtained	

Remarks (If Any): ______________________________

Instructor Name: ______________________________

Instructor Signature: ________________ **Date:** ____________

LAB SESSION 04

OBJECTIVE:
To apply decision making

THEORY:

Normally, your program flows along line by line in the order in which it appears in your source code. But, it is sometimes required to execute a particular portion of code only if certain condition is true; or false i.e. you have to make decision in your program. There are three major decision making structures. Four decision making structures:

1. If statement
2. If-else statement
3. Switch case
4. Conditional Operator (Rarely used)

The if statement:
The if statement enables you to test for a condition
(such as whether two variables are equal) and branch to different parts of your code, depending on the result.
The simplest form of an if statement is:

```
if (expression)
statement;
```

The expression may consist of logical or relational operators like (> >= < <= && ||)

Example:

```
void main(void)
{
int var;
printf("Enter any number;"); scanf("%d",&var);
if(var==10)
printf("The user entered number is Ten");
                                }
```

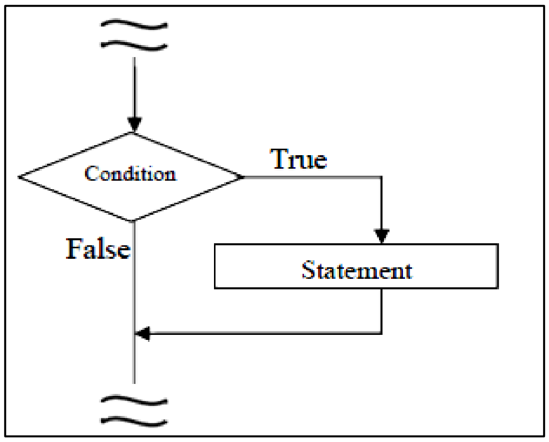


Figure 4.1: Flow Chart (If-statement)

The if-else statement:

Often your program will want to take one branch if your condition is true, another if it is false. The keyword else can be used to perform this functionality:

```
if (expression)
statement;
    else
    statement;
```

Note: To execute multiple statements when a condition is true or false, parentheses are used.

Consider the following example that checks whether the input character is an upper case or lower case:

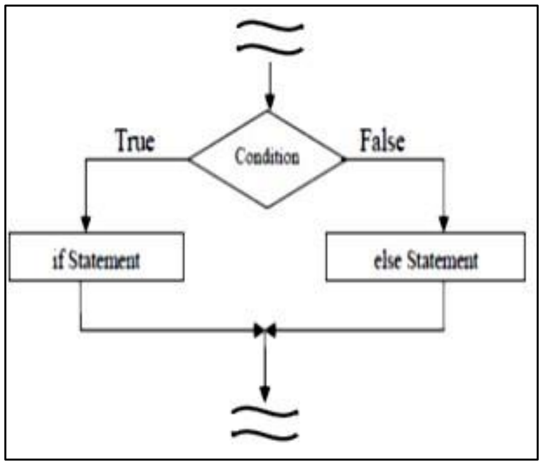


Figure 4.2: Flowchart (If-else statement)

Example:

```
void main(void)
{
char ch;
printf("Enter any character");
ch=getche();
if(ch>='A'&&ch<='Z')
     printf("%c is an upper case character",ch);
     else
 printf("%c is a lower case character",ch); getch();
}
```

The switch Statement:

Unlike if, which evaluates one value, switch statements allow you to branch on any of a number of different values. The general form of the switch statement is:

```
switch (expression)
{
case valueOne:
statement;
break;
case valueTwo:
statement;
break;
....
case valueN:
statement;
break;
default:
statement;
}
```

Example:

```
 void main(void)
 {
 clrscr(); char grade;
  printf(“\n Enter your Grade: ”); grade=getche();
 switch(grade)
 {
 case ‘A’:
 case ‘a’:
 printf(“\n Your percentage is 80 or above 80 ”); break;

 case ‘B’:
 case ‘b’:
printf(“\n Your percentage is in 70-80 ”); break;

 default:
 printf(“\n Your percentage is below 70 ”);

}
getch();
}
```

Conditional (Ternary) Operator:

The conditional operator (?:) is C’s only ternary operator; that is, it is the only operator to take three terms.

The conditional operator takes three expressions and returns a value: (expression1) ? (expression2) : (expression3);

This line is read as "If expression1 is true, return the value of expression2; otherwise, return the value of expression3." Typically, this value would be assigned to a variable.

Example:

```
        void main(void)
{       clrscr(); float per;
        printf(“\n Enter your percentage;”); scanf(“%f”,&per);
        printf(“\n you are”);
        printf(“%s”, per >= 60 ?“Passed”: ”Failed”); getch();      }
```

Typecasting:

Typecasting allow a variable of one type to act like another for a single operation. In C typecasting is performed by placing, in front of the value, the type name in parentheses.

LABORATORY TASKS:

1. Write a program that takes a number as input from user and checks whether the number is even or odd.

a) Using if-else

b) Using conditional operator

2. Mention the output for the following program:

```
  #include<stdio.h> void main()
{
int a=100; if(a>10)
printf("Shahid Afridi"); else if(a>20) printf("Shoaib Akhtar"); else if(a>30) printf("Kamran Akmal");
}
```

3. Write a program that declares and initializes two numbers with your_roll_no and your_friend_roll_no and displays the greater of the two. Use ternary operator.

RESULT:

LAB SESSION 05

To apply looping constructs

Student Name: ______________________________

Roll Number: ____________________ **Batch:** ______________

Semester: ______________________ **Year:** ______________

Total Marks	
Marks Obtained	

Remarks (If Any): ______________________________

Instructor Name: ______________________________

Instructor Signature: ____________________ **Date:** __________

LAB SESSION 05

OBJECTIVE:
To apply looping constructs

THEORY:
The concept of looping provides us a way to execute a set of instructions more than once until a particular condition is true. In C, loop is constructed by three ways.

Types of loops:

1) for Loop
   i. simple for loop   ii. Nested for loop
2) while Loop
   i. simple while loop   ii. Nested while loop
3) do - while Loop
   i. simple do while loop   ii. Nested do while loop

The for Statement
The for loop is appropriate when you know in advanced how many times the loop will be executed. Here you have a counter variable whose limits are define. The general form of the for statement is

for (initialization of counter; loop continuation condition; increment counter)
{
statements;
}
The initialization expression initializes the loop's control variable or counter (it is normally set to 0); loop continuation condition determines whether the loop counter is within its limits or not and finally the increment statement increments the counter variable after executing the inner loop statements. The flow chart of the for loop can be shown as

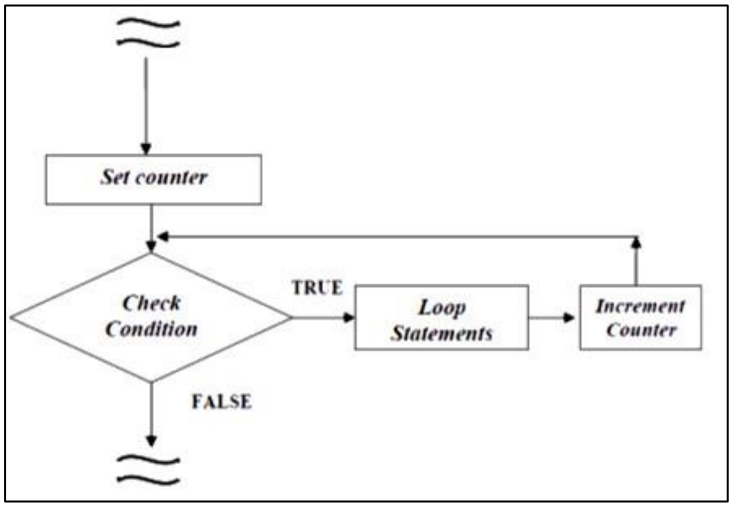
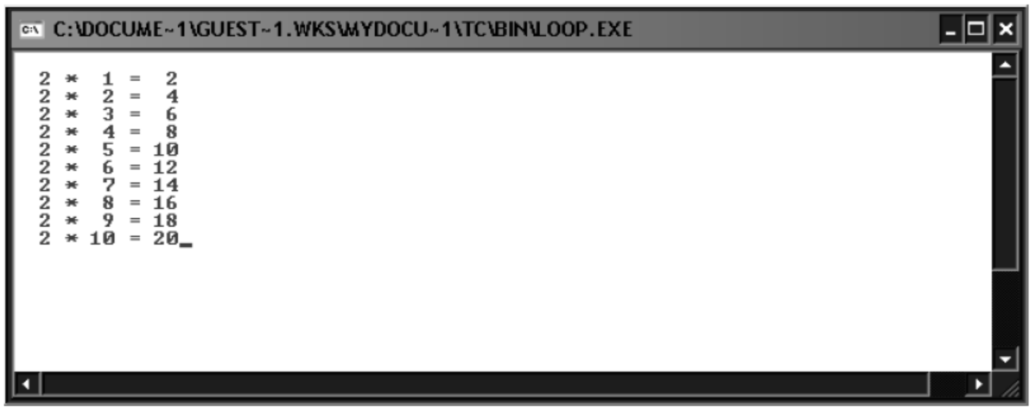


Figure 5.1: Flow Chart (for loop)

Example:

```
void main(void)
{
clrscr();
int counter;          /*Declaring a counter variable*/
int table=2;        /*To print the table of */
for(counter =1;counter <=10;counter++)
   printf("\n%3d *%3d =%3d",table,counter,table*counter);
getch();
}
```

Output:



```
2 *  1 =  2
2 *  2 =  4
2 *  3 =  6
2 *  4 =  8
2 *  5 = 10
2 *  6 = 12
2 *  7 = 14
2 *  8 = 16
2 *  9 = 18
2 * 10 = 20_
```

Figure 5.2: Output

The while Statement:

The while loop is used to carry out looping operations, in which a group of statements is executed repeatedly, if condition following while is true otherwise control is transferred to the end of the loop. Here we do not know how many times the loop will be executed.

The general form of the while statement is
while (condition)
{
statement1;
:
statement2;
}

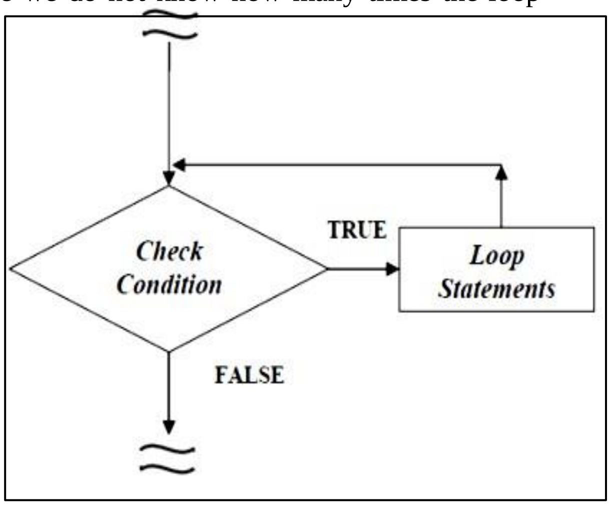
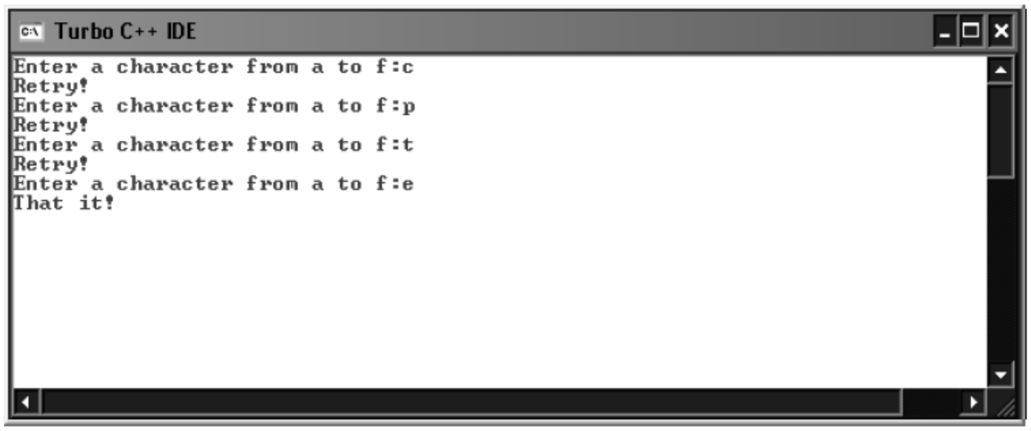


Figure 5.3: Flow chart(While Loop)

Example:

```
void main(void)
{
clrscr();
char guess;          /*Declaring a counter variable*/
printf("Enter a character from a to f:");
guess=getche();
while(guess!='e')
     {
     printf("\nRetry!");
     printf("\nEnter a character from a to f:");
     guess=getche();
     }
printf("\nThats it!");
getch();
}
```

Output:



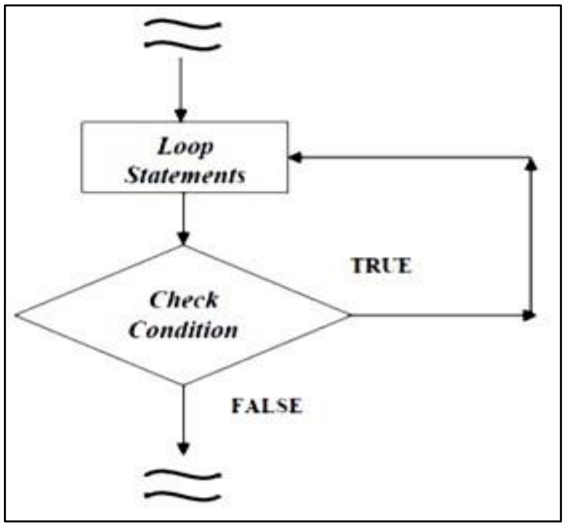
```
Enter a character from a to f:c
Retry!
Enter a character from a to f:p
Retry!
Enter a character from a to f:t
Retry!
Enter a character from a to f:e
That it!
```

Figure 5.4: Output

The do while Statement

The do while repetition statement is similar to
the while statement. In the while statement, the loop-continuation condition test occurs at the beginning of the loop before the body of the loop executes. The do while statement tests the loop-continuation condition after the loop body executes; therefore, the loop body always executes at least once.

do
{ **Statement;**
}
while (condition);

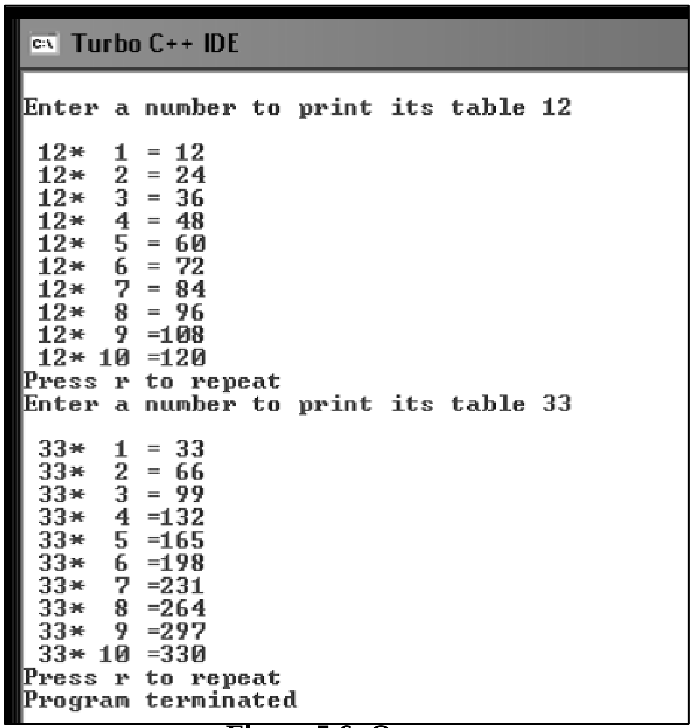


This loop must be executed at least once because the condition is checked at the end. If the condition is following while is true the control is transferred to the beginning of the loop statement otherwise control is transferred to the statement following while statement.

Example:

```
void main(void)
{
clrcsr();
char guess;           /*Declaring a counter variable*/
int cnt,num;
do
     {
            printf("\nEnter a number to print its table");
            scanf("%d",&num);
     for(cnt=1;cnt<=10;cnt++)
          printf("\n%3d*%3d =%3d",num,cnt,num*cnt);
     printf("Press r to repeat");
     guess=getch();
     }
while(guess =='r');
printf("\n Program terminated");
getch();
}
```

Output:



```
Enter a number to print its table 12

 12*  1 = 12
 12*  2 = 24
 12*  3 = 36
 12*  4 = 48
 12*  5 = 60
 12*  6 = 72
 12*  7 = 84
 12*  8 = 96
 12*  9 =108
 12* 10 =120
Press r to repeat
Enter a number to print its table 33

 33*  1 = 33
 33*  2 = 66
 33*  3 = 99
 33*  4 =132
 33*  5 =165
 33*  6 =198
 33*  7 =231
 33*  8 =264
 33*  9 =297
 33* 10 =330
Press r to repeat
Program terminated
```

Figure 5.6: Output

LABORATORY TASKS:

1. Write down the output of the following program statements

i. for (i=1; i<=10;i++) printf(“%d \n”,i);

ii.   int a = 10, b = 10; for(inti=1;i<=a;i++)

{ a++; b--;

printf(“a = %d,b=%d\t”,a,b);

}

2 Write a program to generate a series of first 50 even numbers

3.Write a program to generate tables from 2 to 20 with first 10 terms

4. Write two programs, which may be used to input a sentence.
    Terminate when Enter key is pressed. (Use for and while loop)

5.Write a program to enter the numbers till the user wants and at the end it should display the count of positive, negative and zeros entered.

6.Write a program to find the range of a set of numbers. Range is the difference between the smallest and biggest number in the list.

7.Write programs to display the following patterns

```
*
**
***
****
*****
******
```

```
    1
   121
  12321
 1234321
123454321
```

RESULT:

LAB SESSION 06

To explore functions

Student Name: ______________________________

Roll Number: ________________ **Batch:** ____________

Semester: ____________________ **Year:** ____________

Total Marks	
Marks Obtained	

Remarks (If Any): ____________________________

Instructor Name: ____________________________

Instructor Signature: ______________ **Date:** __________

LAB SESSION 06

OBJECTIVE:
To explore functions

THEORY:
The general structure of a function declaration is as follows:

return_type function_name(arguments);

Before defining a function, it is required to declare the function i.e. to specify the function prototype. A function declaration is followed by a semicolon ';'. Unlike the function definition only data type are to be mentioned for arguments in the function declaration. The function call is made as follows:

return_type = function_name(arguments);

There are four types of functions depending on the return type and arguments:

- Functions that take nothing as argument and return nothing.
- Functions that take arguments but return nothing.
- Functions that do not take arguments but return something.
- Functions that take arguments and return something.

Example 1:
Consider a simple example of function declaration, definition and call. void function1(void);

```
void function2(void)
{
printf("Writing in Function2\n");
}
void main(void)
{
                printf("Writing in main\n"); function1( );

       }        void function1(void) printf("Writing in Function1\n");
                function2( );
       {


       }
```

Example 2:
Consider another example that adds two numbers using a function sum() . void sum(void);

```
void main(void)
```

```
{
printf("\nProgram to print sum of two numbers\n"); sum(void);
}
void sum(void)
{
int num1,num2,sum; printf("Enter 1st number:"); scanf("%d",&num1); printf("Enter 2nd number:");
scanf("%d",&num2); sum=num1+num2;
printf("Sum of %d+%d=%d",num1,num2,sum);
}
```

Recursion

Recursion is an ability of a function to call itself.

Example:

An example: A program that calculates the following series using recursion.
n + (n-1) + (n-2) + ……….. + 3 +2 + 1

```
int add(int); void main(void)
{
int num,ans;
printf("Enter any number:"); scanf("%d",&num); ans=add(num); printf("Answer=%d",ans); getch();
}
int add(int n)
{
int result; if(n==1) return 1;
result=add(n-1) + n; return result;
}
```

Built-in Functions

There are various header files which contain built-in functions. The programmer can include those header files in any program and then use the built-in function by just calling them.

LABORATORY TASKS:

1. Using function, write a complete program that prints your name 10 times. The function can take no arguments and should not return any value.

2. Write function definition that takes two complex numbers as argument and prints their sum.

3. Using a function, swap the values of two variables. The function takes two values of Variables as arguments and returns the swapped values

4. Identify the errors (if any) in the following code:

a)

```
        func(int a,int b)
{               int a;
a=20;
return a;                       }
```

b)

```
        #include<stdio.h> int main()
{       int myfunc(int); int b; b=myfunc(20); printf("%d",b); return 0;     }

int myfunc(int a)
{               a > 20? return(10): return(20);
}
```

5. Using recursion, write a program that takes a number as input and print its binary equivalent.

____

____

____

____

____

____

____

____

____

____

6. main( ) is a function. Write a function which calls main( ). What is the output of this program?

____

____

____

____

____

RESULT:

LAB SESSION 07

To study the preprocessor directives

Student Name: ______________________________

Roll Number: __________________ **Batch:** ______________

Semester: ____________________ **Year:** ______________

Total Marks	
Marks Obtained	

Remarks (If Any): ______________________________

Instructor Name: ______________________________

Instructor Signature: __________________ **Date:** __________

LAB SESSION 07

OBJECTIVE:

To study the preprocessor directives

THEORY:

Preprocessor directives are actually the instructions to the compiler itself. They are not translated but are operated directly by the compiler. The most common preprocessor directives are

i. include directive
ii. define directive

i. **include directive:** The include directive is used to include files like as we include header files in the beginning of the program using #include directive like

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

ii. **define directive:** It is used to assign names to different constants or statements which are to be used repeatedly in a program. These defined values or statement can be used by main or in the user defined functions as well. They are used for

a) Defining a constant b) Defining a statement c) Defining a mathematical expression

Example

```
#define pi 3.142
#define p printf("enter a new number");
#define for(a) (4/3.0)*pi*(a*a*a);
```

They are also termed as macros.

LABORATORY TASKS:

1. Write a program which calculates and returns the area and volume of a sphere using define directive.

2. Write a program which takes four integers a, b, c, d as input and prints the largest one using define directive.

3. Which of the following are correctly formed #define statements:

#define INCH PER FEET 12

#define SQR (X) (X * X)

#define SQR(X) X * X

#define SQR(X) (X * X)

RESULT:

LAB SESSION 08

To apply array concept

Student Name: ____________________________

Roll Number: ________________ **Batch:** ____________

Semester: ________________ **Year:** ____________

Total Marks	
Marks Obtained	

Remarks (If Any): ____________________________

Instructor Name: ____________________________

Instructor Signature: ________________ **Date:** __________

LAB SESSION 08

OBJECTIVE:

To apply array concept

THEORY:

In C we define an array (also termed as subscripted variable) as a collection of variables of certain data type, placed contiguously in memory. Let s examine this definition more closely. Like any other variable in C, an array must be defined int TC[15];

This statement declares an array variable, named TC, capable of holding 15 integer type data elements. The brackets [] tell the compiler that we are dealing with an array.

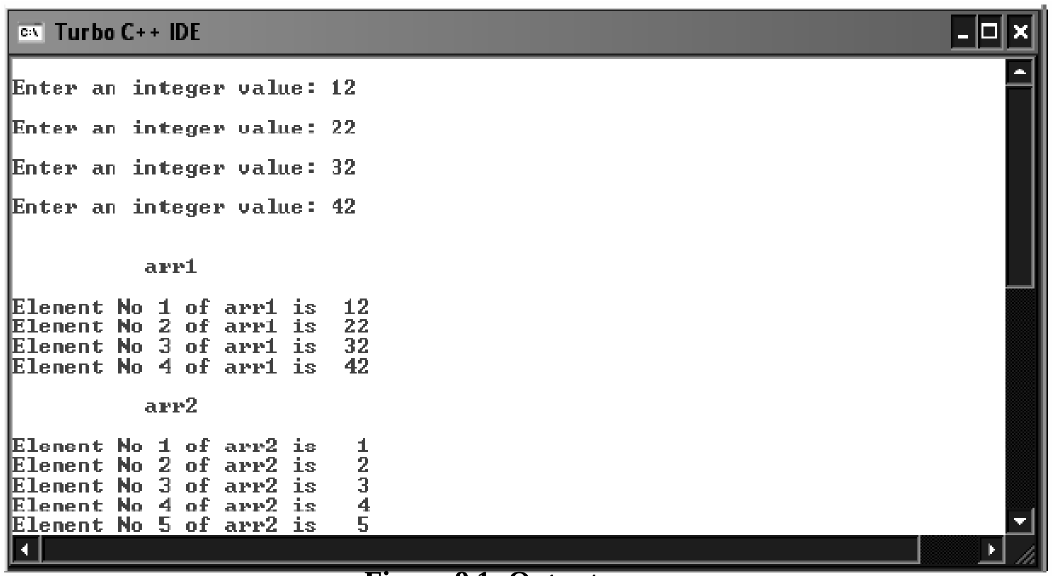
Example:

The following example presents how to define arrays, how to initialize arrays and how to perform many common array manipulations.

```
void main(void)
{
int arr1[4];        /*Declaring   4-element   array   of
integer type*/
int arr2[4]={1,2,3,4,5};
for(int i=0;i<4;i++)          /*Accessing arr1*/
     {
     printf("\nEnter an integer value: ");
     scanf("%d",&arr[i]);
     }
printf("\t\t\tarr1");
for(i=0;i<4;i++)    /*Accessing  arr1  to  print  stored
values*/
printf("\nElement No %d of arr1 is d", i+1, arr1[i]);
printf("\t\t\tarr2");
for(i=0;i<5;i++)              /*Accessing arr2*/
printf("\nElement No %d of arr1 is d", i+1, arr2[i]);
getch();
}
```

NOTE: All the array elements are numbered. The first element in an array is numbered 0, so the last element is one less than the size of the array.

Output:



```
Enter an integer value: 12

Enter an integer value: 22

Enter an integer value: 32

Enter an integer value: 42


          arr1

Element No 1 of arr1 is  12
Element No 2 of arr1 is  22
Element No 3 of arr1 is  32
Element No 4 of arr1 is  42

          arr2

Element No 1 of arr2 is   1
Element No 2 of arr2 is   2
Element No 3 of arr2 is   3
Element No 4 of arr2 is   4
Element No 5 of arr2 is   5
```

Figure 8.1: Output

LABORATORY TASKS:

1. Write a program to convert a decimal number into its binary equivalent.

2. Read in 20 numbers, each of which is in between 10 and 100. As each number is read, print it only if it is not a duplicate of number already read.

LAB SESSION 09

To investigate the use of strings in C

Student Name: ______________________________

Roll Number: ____________________ **Batch:** ______________

Semester: ______________________ **Year:** ______________

Total Marks	
Marks Obtained	

Remarks (If Any): ______________________________

Instructor Name: ______________________________

Instructor Signature: __________________ **Date:** __________

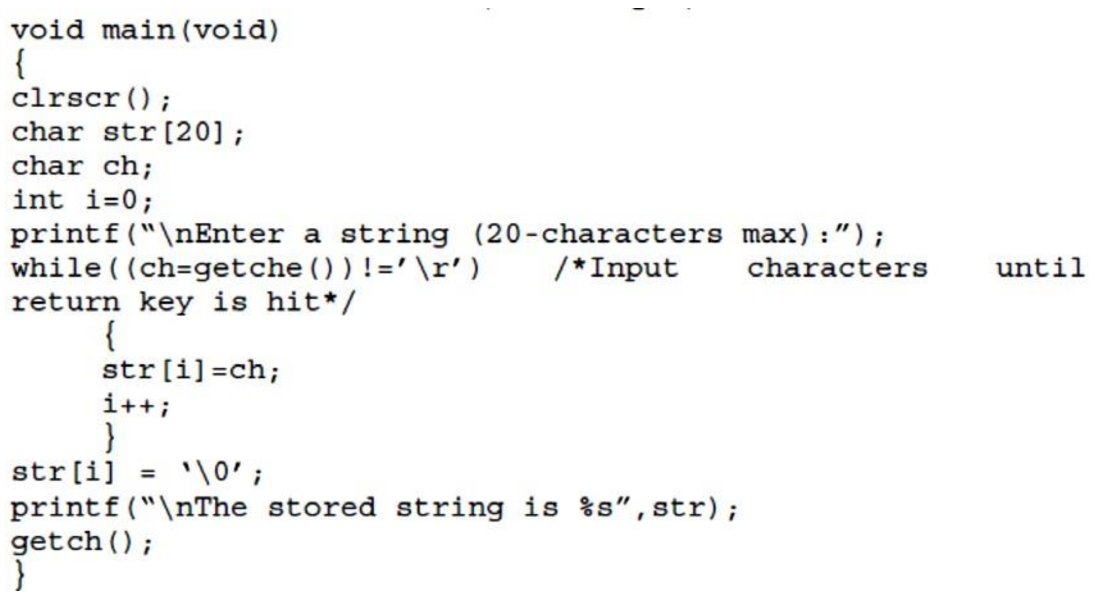
LAB SESSION 09

OBJECTIVE:
To investigate the use of strings in C

THEORY:
A string is an especial type of array of type char. Strings are the form of data used in programming languages for storing and manipulating text.
A string is a one dimensional array of characters. Following are some examples of string initializations
char str1[]={ N , E , D , \0 }; char str2[]={ NED };
char str3[]= NED ;

Each character in the string occupies one byte of memory and the last character is always a NULL i.e. \0, which indicates that the string has terminated. Note that in the second and third statements of initialization \0 is not necessary. C inserts the NULL character automatically.

Example:
Let us consider an example in which a user provides a string (character by character) and then the stored string is displayed on the screen.



```
void main(void)
{
clrscr();
char str[20];
char ch;
int i=0;
printf("\nEnter a string (20-characters max):");
while((ch=getche())!='\r')    /*Input    characters    until
return key is hit*/
      {
      str[i]=ch;
      i++;
      }
str[i] = '\0';
printf("\nThe stored string is %s",str);
getch();
}
```

NOTE: It is necessary to provide \0 character in the end. For instance if you make that statement a comment, you will observe erroneous results on the screen.

Output:

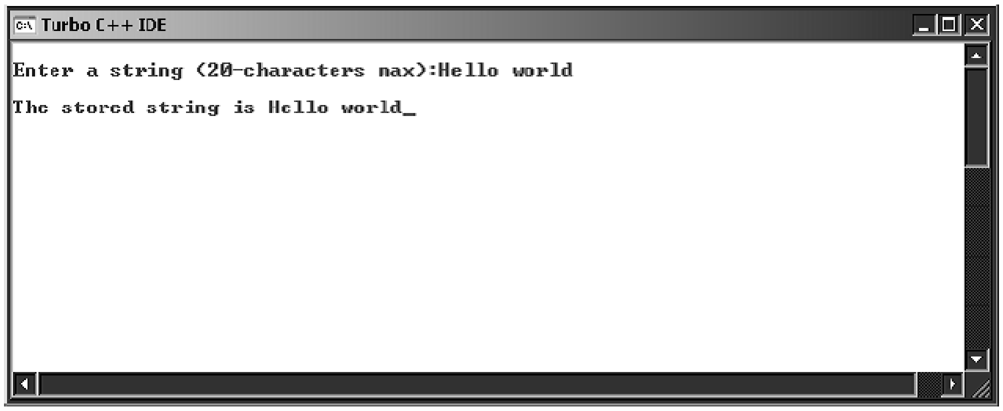

```
Enter a string (20-characters max):Hello world

The stored string is Hello world_
```

Figure 9.1: Output

Library Functions for Strings
There are many library functions for string handling in C. Some of the most common are listed below. In order to use these library functions you have to include header file named string.h

Functions	Use
strlen	Finds length of the string
strlwr	Converts a string to lowercase
strupr	Converts a string to uppercase
strcpy	Copies a string into another
strcmp	Compares two strings
strrev	Reverses string
gets	Input string from keyboard
puts	Output string on the screen

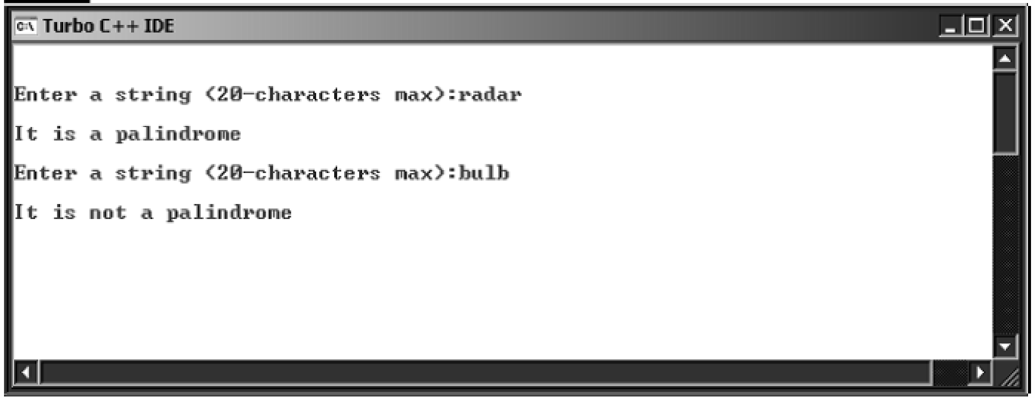
Table 9.1: Library functions for strings

Study all the above mentioned functions.

Example:
A palindrome is a string that is spelled the same way forward and backwards. Some examples of palindromes are: radar, mom and dad. Let s implement a program that that determines whether the string passed to is palindrome or not.

```
#include<conio.h>
#include<stdio.h>
#include<string.h>  /* Header file for string library
function*/
void main(void)
{
clrscr();
char str1[20];
char str2[20];
printf("\nEnter a string (20-characters max):");
gets(str1);          /*Input string*/
strcpy(str2,str1);  /*Equating the two strings*/
strrev(str2);       /*Reverses str2*/
if(strcmp(str1,str2)==0) /*Making decision*/
     printf("\nIt is a palindrome");
else
     printf("\nIt is not a palindrome");
getch();
}
```

Output:

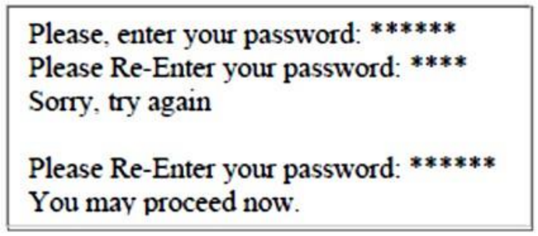


```
Enter a string (20-characters max):radar

It is a palindrome

Enter a string (20-characters max):bulb

It is not a palindrome
```

Figure 9.2: Output

LABORATORY TASK:

Carefully observe the output generated by a program. You are required to write the source code for the program.



```
Please, enter your password: ******
Please Re-Enter your password: ****
Sorry, try again

Please Re-Enter your password: ******
You may proceed now.
```

_

RESULT:

LAB SESSION 10

To explore the applications of structures

Student Name: ______________________________

Roll Number: ____________________ **Batch:** ______________

Semester: ______________________ **Year:** ______________

Total Marks	
Marks Obtained	

Remarks (If Any): ______________________________

Instructor Name: ______________________________

Instructor Signature: __________________ **Date:** __________

LAB SESSION 10

OBJECTIVE:
To explore the applications of structures

THEORY:
If we want a group of same data type we use an array. If we want a group of elements of different data types we use structures. For Example: To store the names, prices and number of pages of a book you can declare three variables. To store this information for more than one book three separate arrays may be declared. Another option is to make a structure. No memory is allocated when a structure is declared. It simply defines the "form" of the structure. When a variable is made then memory is allocated. This is equivalent to saying that there is no memory for "int", but when we declare an integer i.e. int var; only then memory is allocated.

Unions are also used to group a number of different variables together like a structure. But, unlike structures, union enables us to treat the same space in memory as a number of different variables. That is, a union is a way for a section of memory to be treated as a variable of one type on one occasion, and as a different variable, of a different type, on another occasion.

Example:

```
struct personnel
{
char name[50]; int agentno;
};
void main(void)
{
struct personnel agent1={"Mustafa",35}; printf("%s",agent1.name); printf("%d",agent1.agentno);
getch();
}
```

LABORATORY TASKS:

1. Declare a structure named employee that stores the employee id, salary and department.

2. Declare an array of 40 employees for the structure defined in question1. Also write statements to assign the following values to the employee [6].
Employee id = “Your_roll_no” salary = 30,000 and department = “IT dept”

3.Write a function that prints the highest salaried person amongst the employees defined in the above question.

4. How much memory is allocated for obj1 in the following code?

```
 union x
{
int i[(int)ceil(your_roll_number/2)]; //declare an array ,having as many elements as your half of your roll #
char c; float f;
} obj1;
```

5. Define a structure to represent a complex number in rectangular format i.e. real +**i** imag. Name it rect. Define another structure called polar that stores a complex number as polar format i.e. mag /angle. Write a function called convert that takes a complex number as input in rectangular format and returns the complex number converted in Polar form.

<u>RESULT:</u>

LAB SESSION 11

To apply the concept of pointers in C

Student Name: ______________________________

Roll Number: ________________ **Batch:** ____________

Semester: __________________ **Year:** ____________

Total Marks	
Marks Obtained	

Remarks (If Any): ____________________________

Instructor Name: ______________________________

Instructor Signature: ______________ **Date:** _________

LAB SESSION 11

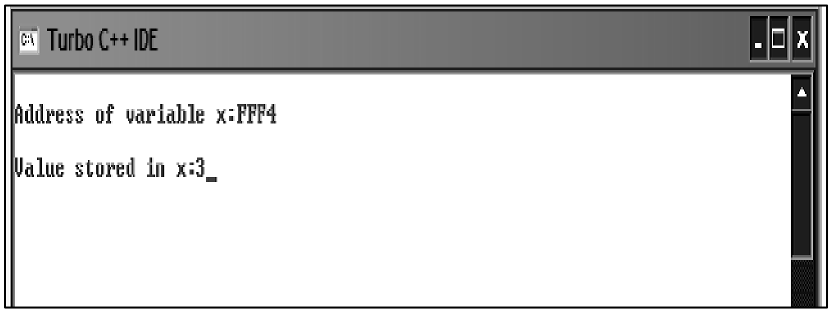
OBJECTIVE:
To apply the concept of pointers in C

THEORY:
Pointers are variables whose values are memory addresses. Normally, a variable directly contains a specific value. A pointer, on the other hand contains, an address of a variable that contains a specific value.
Pointers are used in situations when passing actual value is difficult or undesirable; like, returning more than one value from a function. The concept of pointers also provides an easy way to manipulate arrays and to pass an array or a string from one function to another.

Example:
Let s explore how we declare and initialize a pointer variable, using the following

```
void main(void)        /*Defining main()*/
{
int x = 3;
int *ptx; /*Declaring a pointer type variable, which will
point to an integer variable*/
ptx=&x;   /* Initializing the value of ptx*/
printf("Address of variable x: %p",ptx);
                /*prints the value stored in ptx*/
printf("Value stored in x :%d",*ptx);
       /*Referencing variable x using its pointer*/
getch();
}
```

Output:



```
Address of variable x:FFF4

Value stored in x:3_
```

Figure 11.1: Output

Address on your screen would be different, as they it is allocated when the program executes.

The Indirection Operator: *
The indirection unary operator is used to access the contents of the memory location pointed to. The name indirection Operator stems from the fact that the data is accessed indirectly. The same operator is sometimes called as dereference operator. Hence, * has several different uses

- ✓ Multiply Operator (binary)
- ✓ Indirection Operator (Unary)
- ✓ Used in declaration of a Pointer.

Each time you use * , the complier distinguishes its meaning by the context.

Pointers and Arrays

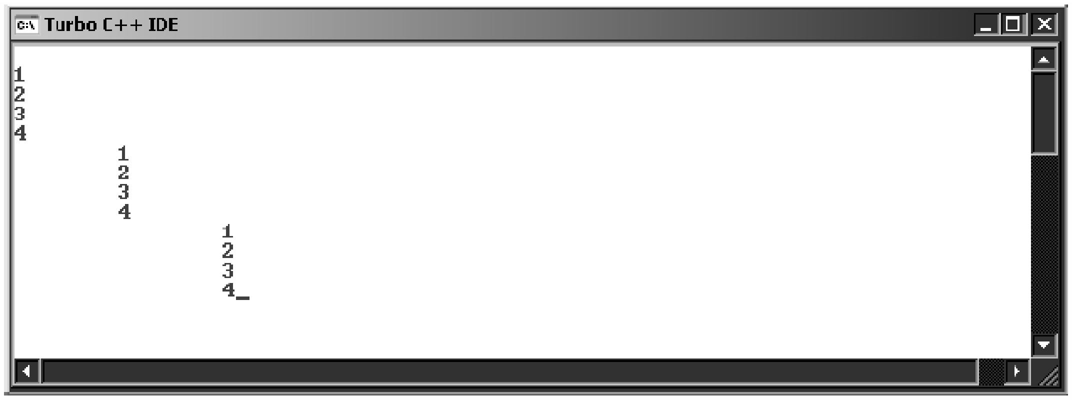
There is an inherent relationship between arrays and pointers; in fact, the compiler translates array notations into pointer notations when compiling the code, since the internal architecture of the microprocessor does not understand arrays.

An array name can be thought of as a constant pointer. Pointer can be used to do any operation involving array subscript. Let us look at a simple example.

Example:

```
void main(void)
{
int arr[4]={1,2,3,4};              /*Initializing 4-element
integer type array*/
for(int indx=0;indx<4;indx++)
printf("\n%d",arr[indx]);
for(int indx=0;indx<4;indx++)
printf("\n\t%d",*(arr+indx));/*arr is a constant pointer
referring to 1st element*/
int *ptr=arr;       /*ptr is a pointer variable, storing
base address of array*/
for(int i=0;i<4;i++)
printf("\n\t\t%d",*ptr++);/*ptr will be incremented(by 2-
byte) on the bases of its type*/
getch();
}
```

Output:



```
1
2
3
4
        1
        2
        3
        4
                1
                2
                3
                4_
```

Figure 11.2: Output

LABORATORY TASKS:

1. Using dynamic memory allocation, declare an array of the length user wants. Take input in that array and then print all those numbers, input by the user, which are even. The verification of whether a number is even or not should be done via macro.

2. Using pointers, write a program that takes a string as input from user and calculates the number of vowels in it.

3. Write pointer notation equivalent to the following array notations: i. arr[10] : _______
ii. arr2D[5][6] : ____

4. Give the function definition for the following function declarations:

i. void sort (char **x ,int no_of_strings);
// Sorts the strings in alphabetical order

ii. char* strstr(char *s1, char *s2);
//Returns the pointer to the element in s1 where s2 begins.

iii. int strlen (char *str);
// Determines length of string

iv. void swap (int *x, int *y);
// You can NOT declare any variable in the function definition

RESULT:

LAB SESSION 12

To perform Disk I/O using C (Files)

Student Name: ______________________________

Roll Number: ____________________ **Batch:** ______________

Semester: ______________________ **Year:** ______________

Total Marks	
Marks Obtained	

Remarks (If Any): ______________________________

Instructor Name: ______________________________

Instructor Signature: __________________ **Date:** __________

LAB SESSION 12

OBJECTIVE:

To perform Disk I/O using C (Files)

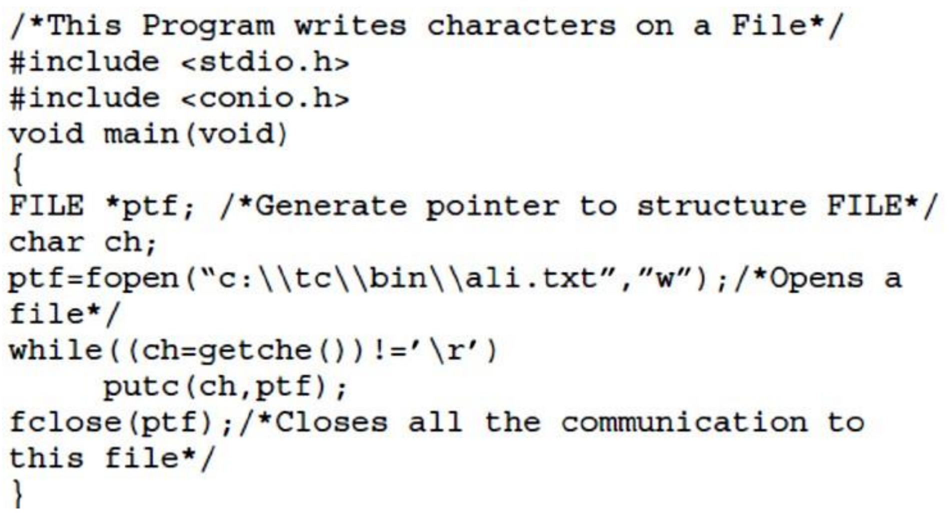
THEORY:

Storage of data in arrays and structure members is temporary; all such data are lost when a program terminates.

Files are used for permanent retention of large data The smallest data item in a computer can assume the value of 0 or the value of 1. Such a data item is called a bit. Programmer prefers to work with data in the form of decimal digits, letters and special symbols. These are referred as characters.
Since computers can only process 1s and 0s, every character is represented as a pattern of 1s and 0s called byte (group of 8 bits). Just as characters are composed of bits, fields are composed of charters. A field is a group of character that conveys meaning. A record is composed of several related fields.
A file is a group of related records. A group of related files is sometimes called as database. A collection of programs designed to create and manage database is called as a database management system.

Example 1:

Let s explore some of the basic functions and features of Standard I/O (a type of disk I/O) with the help of following program



```
/*This Program writes characters on a File*/
#include <stdio.h>
#include <conio.h>
void main(void)
{
FILE *ptf; /*Generate pointer to structure FILE*/
char ch;
ptf=fopen("c:\\tc\\bin\\ali.txt","w");/*Opens a
file*/
while((ch=getche())!='\r')
     putc(ch,ptf);
fclose(ptf);/*Closes all the communication to
this file*/
}
```

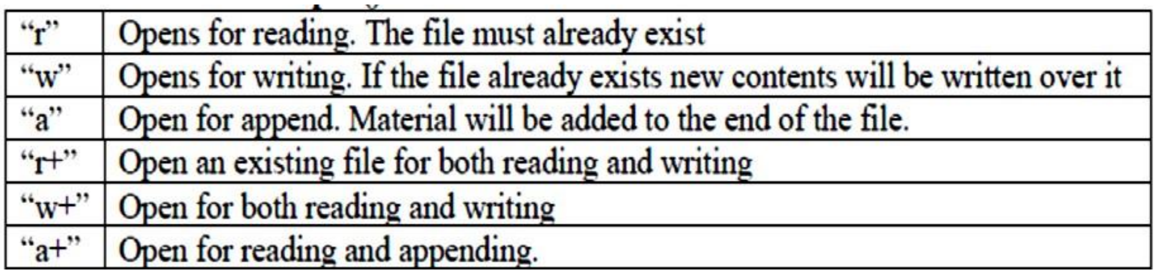
In the first line of main(), we have generated a pointer of type FILE. FILE is a structure that leads indirectly to the operating system s file control block. It is declared in the header file stdio.h . The FILE pointer name ptf shall be used latter to refer to a file. Each file must have a separate pointer.

We then make use of the function fopen to establish a line of communication with the file. The file pointer ptf is assigned a value corresponding to the file name ali.txt.

Function fopen takes two arguments: a file name with path (optional) and a file opening mode. The file open mode w indicates that the file is to be opened for writing. If a file does not exist, it will be created and opened.

Next two lines take characters form user and write it on the file using putc() function. The last statement closes the file. This will free the communication areas used by the file. The areas include FILE structure and the buffer.

<u>Different Modes of fopen() :</u>

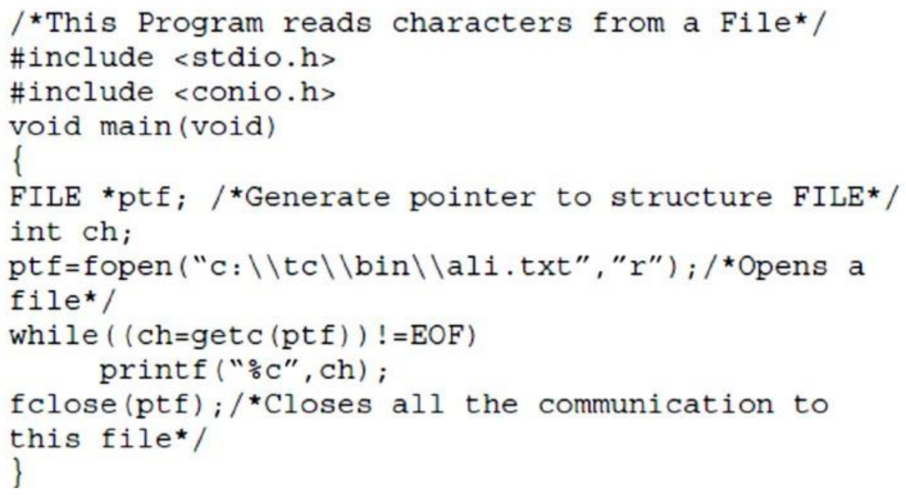


"r"	Opens for reading. The file must already exist
"w"	Opens for writing. If the file already exists new contents will be written over it
"a"	Open for append. Material will be added to the end of the file.
"r+"	Open an existing file for both reading and writing
"w+"	Open for both reading and writing
"a+"	Open for reading and appending.

Table 12.1: Different modes of fopen()

<u>Example 2:</u>

Now let us read the file



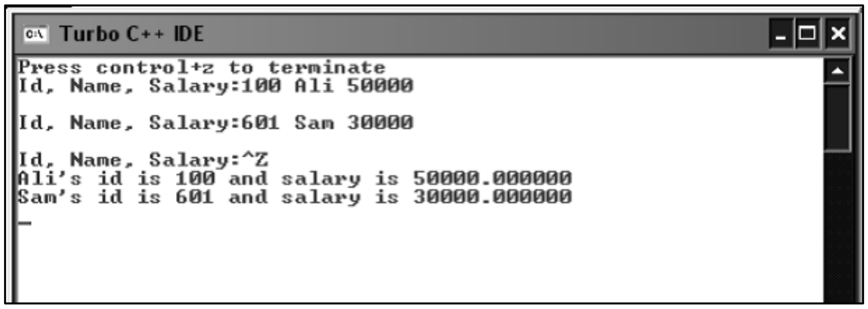
```
/*This Program reads characters from a File*/
#include <stdio.h>
#include <conio.h>
void main(void)
{
FILE *ptf; /*Generate pointer to structure FILE*/
int ch;
ptf=fopen("c:\\tc\\bin\\ali.txt","r");/*Opens a
file*/
while((ch=getc(ptf))!=EOF)
     printf("%c",ch);
fclose(ptf);/*Closes all the communication to
this file*/
}
```

The main difference in this program is that the reading program has to search the last character of the file. It does this by looking for the EOF (end of file) signal from the operating system.

Example of Formatted I/O:

```
/*This Program writes data on a File*/
#include<stdio.h>
#include<conio.h>
void read(void);
void write(void);
void main(void)
{
write();
read();
getch();
}
void write()
{
FILE *Wptr;
int id;
char name[20];
float sal;
Wptr=fopen("new1.dat","w");
printf("Press control+z to terminate\nId, Name, Salary:");
scanf("%d%s%f",&id,name,&sal);
while(!feof(stdin))
  {
  fprintf(Wptr,"%d %s %f ",id,name,sal);
  printf("\nId, Name, Salary:");
  scanf("%d%s%f",&id,name,&sal);
  }
fclose(Wptr);
}
void read(void)
{
FILE *Rptr;
Rptr=fopen("new1.dat","r");
int id;
float sal;
char name[20];
while(fscanf(Rptr,"%d%s%f",&id,name,&sal)!=EOF)
      printf("%s's id is %d and salary is
%f\n",name,id,sal);
fclose(Rptr);
}
```

Output:



```
Press control+z to terminate
Id, Name, Salary:100 Ali 50000

Id, Name, Salary:601 Sam 30000

Id, Name, Salary:^Z
Ali's id is 100 and salary is 50000.000000
Sam's id is 601 and salary is 30000.000000
_
```

Figure 12.1: Output

LABORATORY TASKS:

1. Write a program to store marks of students in a file. The program should take following inputs form the user: name, class roll number and marks. At the end of the entries, list of marks should be produced. The program should ask to append, replace or read the existing data.

2. Write a program to create a file test.txt in /tmp directory and write “This is testing” in that file.

3. A file record.txt contains 100 records of struct rec. Write down necessary statements to read the record # 55 only from the file.

LAB SESSION 13

To study Text and Graphic modes of display

Student Name: ______________________________

Roll Number: ________________ **Batch:** ____________

Semester: ________________ **Year:** ____________

Total Marks	
Marks Obtained	

Remarks (If Any): ______________________________

Instructor Name: ______________________________

Instructor Signature: ________________ **Date:** __________

LAB SESSION 13

OBJECTIVE:

To study Text and Graphics modes of Display

THEORY:

There are two ways to view the display screen in Turbo C graphics model:

- ✓ The Text Mode
- ✓ The Graphics Mode

The Text Mode:

In the Text Mode, the entire screen is viewed as a grid of cells, usually 50 rows by 80 columns.
Each cell can hold a character with certain foreground and background colors (if the monitor is capable of displaying colors). In text modes, a location on the screen is expressed in terms of rows and columns with the upper left corner corresponding to (1,1), the column numbers increasing from left to right and the row numbers increasing vertically downwards.

The Graphics Mode

In the Graphics Mode, the screen is seen as a matrix of pixels, each capable of displaying one or more color. The Turbo C Graphics coordinate system has its origin at the upper left hand corner of the physical screen with the x-axis positive to the right and the y-axis positive going downwards.

The ANSI Standard Codes

The ANSI – American National Standards Institute provides a standardized set of codes for cursor control. For this purpose, a file named ANSI.sys is to be installed each time you turn on your computer. Using the config.sys file, this job is automated, so that once you've got your system set up, you don't need to worry about it again. To automate the loading of ANSI.sys follow these steps:

1. Find the file ANSI.sys in your system. Note the path.
2. Find the config.sys file. Open this file and type the following: DEVICE = path_of_ANSI.sys
3. Restart your computer.

All the ANSI codes start by the character \x1B[after which, we mention codes specific to certain operation. Using the #define directive will make the programs easier to write and understand.

LABORATORY TASKS:

1. Write down program statements to initialize the graphics mode of operation.

2. Which header file is required to be included while working in (a) text mode (b) graphics mode?

3. Name the functions used to clear the screen in (a) text mode (b) graphics mode

RESULT:

LAB SESSION 14

To explore some of the basic graphic functions in C

Student Name: ______________________________

Roll Number: ________________ **Batch:** ____________

Semester: ________________ **Year:** ____________

Total Marks	
Marks Obtained	

Remarks (If Any): ______________________________

Instructor Name: ______________________________

Instructor Signature: ________________ **Date:** __________

LAB SESSION 14

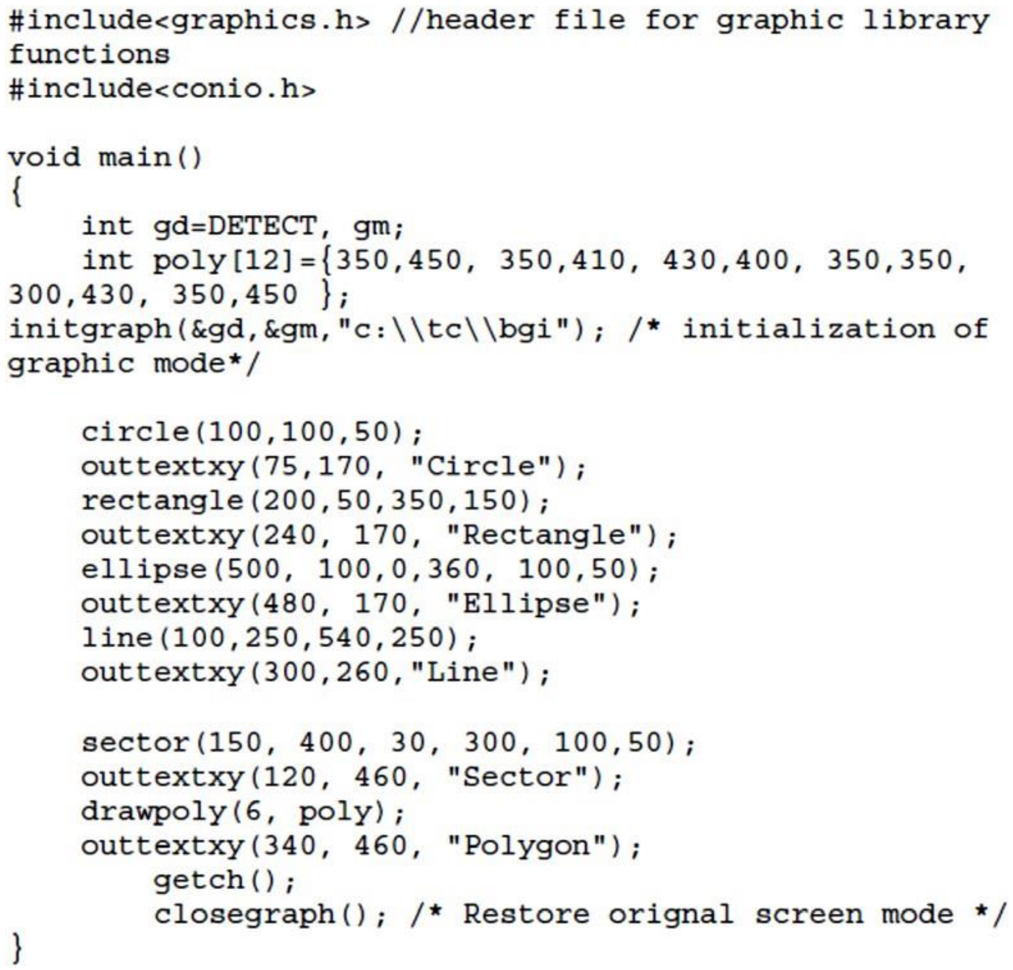
OBJECTIVE:

To explore some of the basic graphic functions in C

THEORY:

In C, graphics is one of the most interested & powerful future of C programming. All video games, animations & multimedia predominantly work using computer graphics. The aim of this lab is to introduce the basic graphics library functions.

Example:



```
#include<graphics.h> //header file for graphic library
functions
#include<conio.h>

void main()
{
    int gd=DETECT, gm;
    int poly[12]={350,450, 350,410, 430,400, 350,350,
300,430, 350,450 };
initgraph(&gd,&gm,"c:\\tc\\bgi"); /* initialization of
graphic mode*/

    circle(100,100,50);
    outtextxy(75,170, "Circle");
    rectangle(200,50,350,150);
    outtextxy(240, 170, "Rectangle");
    ellipse(500, 100,0,360, 100,50);
    outtextxy(480, 170, "Ellipse");
    line(100,250,540,250);
    outtextxy(300,260,"Line");

    sector(150, 400, 30, 300, 100,50);
    outtextxy(120, 460, "Sector");
    drawpoly(6, poly);
    outtextxy(340, 460, "Polygon");
        getch();
        closegraph(); /* Restore orignal screen mode */
}
```

To run this program, you need graphics.h header file, graphics.lib library file and Graphics driver (BGI file) in the compiler package for C. In graphics mode, all the screen co-ordinates are mentioned in terms of pixels. Number of pixels in the screen decides resolution of the screen. In the example, circle is drawn with x-coordinate of the center 100, y-coordinate 100 and radius 50 pixels. All the coordinates are mentioned with respect to top-left corner of the screen.

Library Functions:

initgraph():

This function

- ✓ Initializes the graphics system by loading a graphics driver from disk (or validating a registered driver) then putting the system into graphics mode.
- ✓ initgraph also resets all graphics settings (color, palette, current position, viewport, etc.) to their defaults, then resets graphresult to 0.

Declaration:

void far initgraph(int far *graphdriver, int far *graphmode, char far *pathtodriver);

Arguments:

*graphdriver: Integer that specifies the graphics driver to be used

*graphmode: Integer that specifies the initial graphics mode (unless *graphdriver = DETECT). If *graphdriver = DETECT, initgraph sets *graphmode to the highest resolution available for the detected driver.

pathtodriver: Specifies the directory path where initgraph looks for graphics drivers.

*pathtodriver: Full pathname of directory, where the driver files reside. If the driver is not found in the specified path, the function will search the current directory for the .BGI files.

closegraph():

This function switches back the screen from graphcs mode to text mode. It clears the screen also.
A graphics program should have a closegraph function at the end of graphics. Otherwise DOS screen will not go to text mode after running the program. Here, closegraph() is called after getch() since screen should not clear until user hits a key.

outtextxy():

Function outtextxy() displays a string in graphical mode. You can use different fonts, text sizes, alignments, colors and directions of the text. Parameters passed are x and y coordinates of the position on the screen where text is to be displayed.

Declaration:

void far outtextxy(int x, int y, char *text);

circle():

circle() function takes x & y coordinates of the center of the circle with respect to left top of the screen and radius of the circle in terms of pixels as arguments.

Declaration:

void far circle(int x, int y, int radius);

Arguments:
(x,y): Center point circle. radius: Radius of circle.

rectangle() & drawpoly():
To draw a border, rectangle and square use rectangle() in the current drawing color, line style and thickness.
To draw polygon with n sides specifying n+1 points, the first and the last point being the same.

Declaration:
void far rectangle(int left, int top, int right, int bottom); void far drawpoly(int numpoints, int far *polypoints);

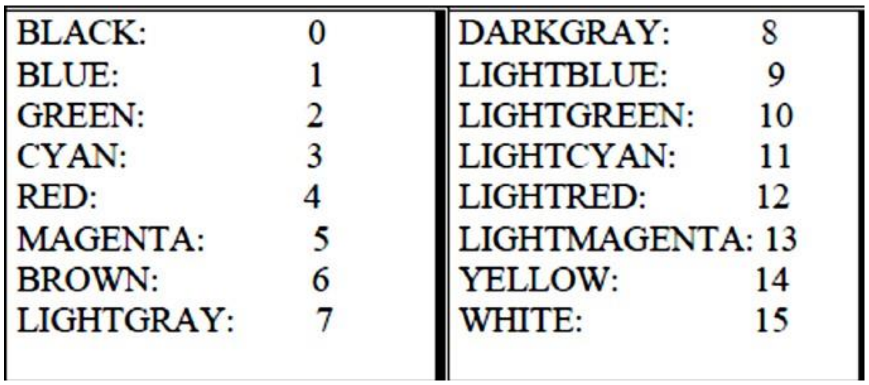
Arguments:
(left,top) is the upper left corner of the rectangle, and (right,bottom) is its lower right corner.

numpoints: Specifies number of points

*polypoints: Points to a sequence of (numpoints x 2) integers. Each pair of integers gives the x and y coordinates of a point on the polygon.

To draw a closed polygon with N points, numpoints should be N+1 and the array polypoints[] should contain 2(N+1) integers with first 2 integers equal to last 2 integers.

Setting Colors:
There are 16 colors declared in graphics.h as listed in Table 14.1



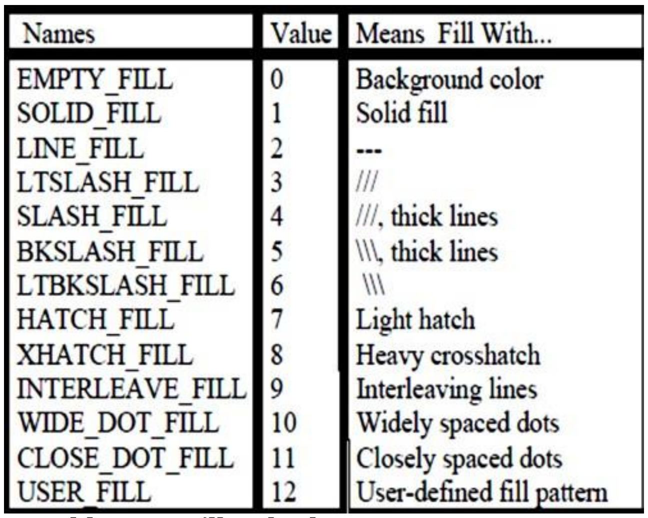
BLACK:	0	DARKGRAY:	8
BLUE:	1	LIGHTBLUE:	9
GREEN:	2	LIGHTGREEN:	10
CYAN:	3	LIGHTCYAN:	11
RED:	4	LIGHTRED:	12
MAGENTA:	5	LIGHTMAGENTA:	13
BROWN:	6	YELLOW:	14
LIGHTGRAY:	7	WHITE:	15

Table 14.1: Colour code chart

To use these colors, use functions setcolor(), setbkcolor() and setfillstyle().

- ✓ setcolor() function sets the current drawing color. If we use setcolor(RED); and draw any shape, line or text after that, the drawing will be in red color. You can either use color as defined above or number like setcolor(4)
- ✓ setbkcolor() sets background color for drawing.

✓ setfillstyle() sets fill pattern and fill colors. After calling setfillstyle, if we use functions like floodfill, fillpoly, bar etc, shapes will be filled with fill color and pattern set using setfillstyle. The parameter pattern in setfillstyle is describe in Table 14.2.



Names	Value	Means Fill With...
EMPTY_FILL	0	Background color
SOLID_FILL	1	Solid fill
LINE_FILL	2	---
LTSLASH_FILL	3	///
SLASH_FILL	4	///, thick lines
BKSLASH_FILL	5	\\\, thick lines
LTBKSLASH_FILL	6	\\\
HATCH_FILL	7	Light hatch
XHATCH_FILL	8	Heavy crosshatch
INTERLEAVE_FILL	9	Interleaving lines
WIDE_DOT_FILL	10	Widely spaced dots
CLOSE_DOT_FILL	11	Closely spaced dots
USER_FILL	12	User-defined fill pattern

Table 14.2: Fill style chart

Example:

```
#include "graphics.h"
#include "conio.h"
#include "stdlib.h"

void main()
{
    int gd,gm;
    gd=DETECT;

    initgraph(&gd, &gm, "");
    setcolor(3);
    setfillstyle(SOLID_FILL,RED);
    bar(50, 50, 590, 430);

    setfillstyle(1, 14);
    bar(100, 100, 540, 380);

    while(!kbhit())
    {
        putpixel(random(439)+101,
random(279)+101,random(16));
        setcolor(random(16));
        circle(320,240,random(100));
    }
    getch();
    closegraph();
}
```

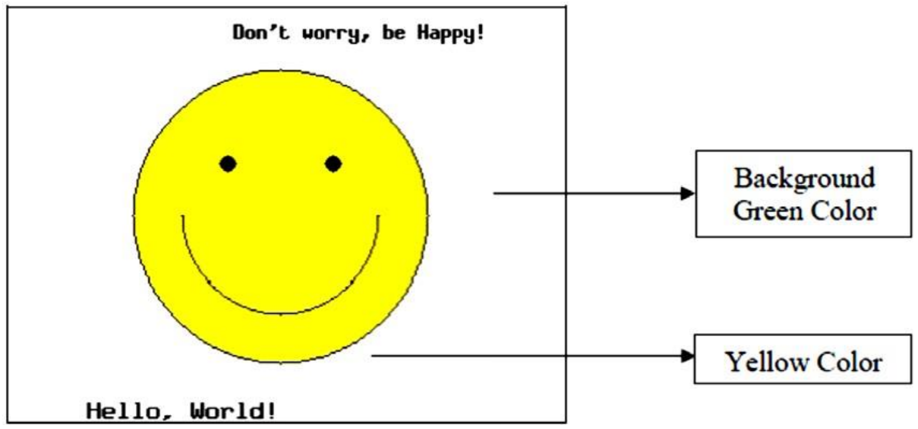
- ✓ random(no), defined in stdlib.h returns a random number between 0 an no. The effect is by drawing random radius, random color circles with same center and random pixels.

- ✓ kbhit(), defined in conio.h returns a nonzero value when a key is pressed in the keyboard. So, the loop will continue until a key is pressed.

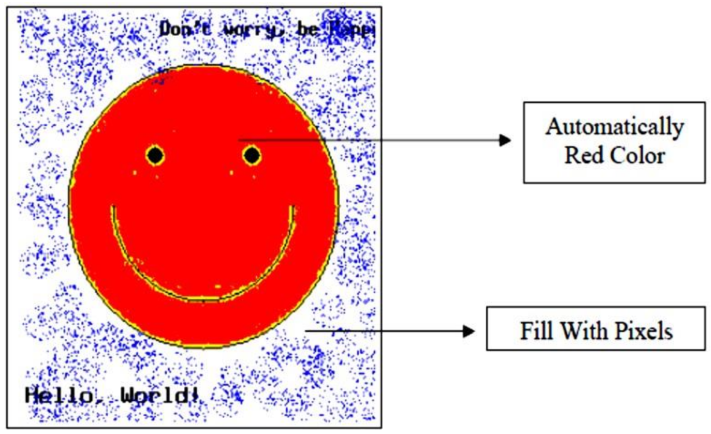
LABORATORY TASKS:

1) Use the graphics functions to construct the following output.

Task 1:



Task 2:



RESULT: