



**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 WORKBOOK
For The Course
TC-105 PROGRAMMING LANGUAGES

Prepared By:

Revised By:

Reviewed By:

Approved By:

**Board of Studies of Department of Telecommunications
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 Languages introduces the basic as well as advanced concepts of programming using C and C++ languages. 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). And C++ introduces Object Oriented Programming Concept into C.

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 workbook 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 No.	Date	Experiments	CLO	Page No	Signature
1		To explore Turbo C IDE and Programming Environment	3		
2		To study basic building blocks of C-language such as data types and input-output functions	3		
3		To study the different types of arithmetic and logical operators	3		
4		To Apply decision making	3		
5		To apply looping constructs	3		
6		To explore functions	3		
7		To study preprocessor directives	3		
8		To apply array concept	3		
9		To investigate the use of strings in C	3		
10		To apply the concept of pointers in C	3		
11		To introduce problem solving ability using Object Oriented Programming using C++	3		
12		To investigate different features of Objects in C++	3		
13		Experimenting Blinking LED with Arduino Board	3		

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.

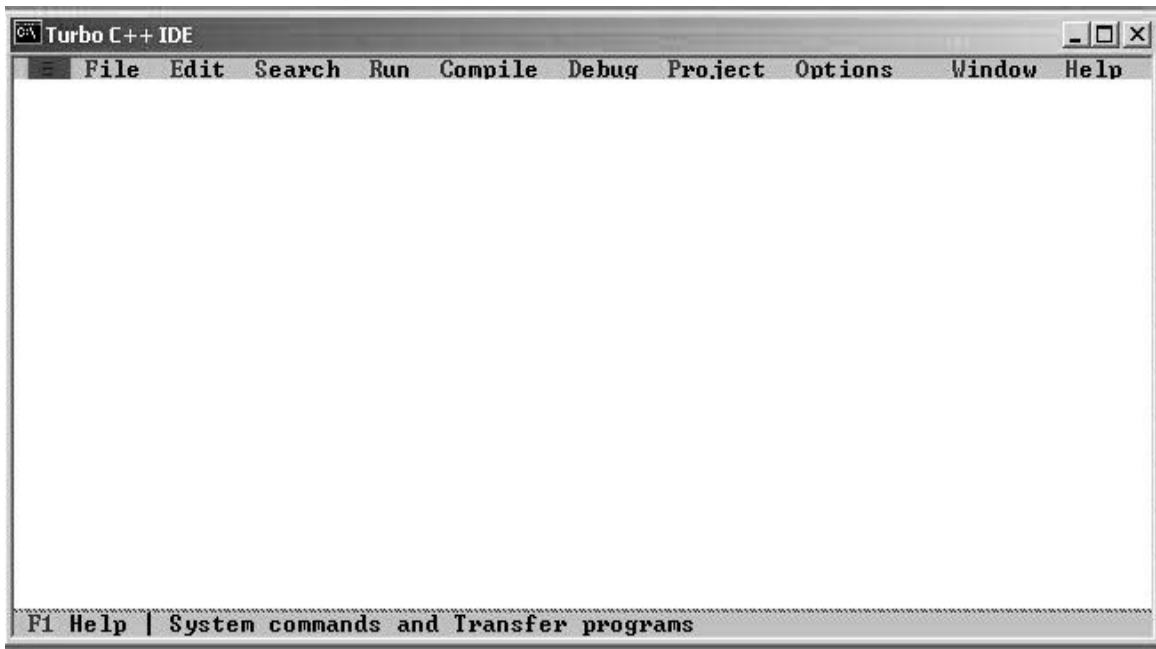


Figure 1.1: TURBO C IDE Environment

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.



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 the method to create an executable 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.

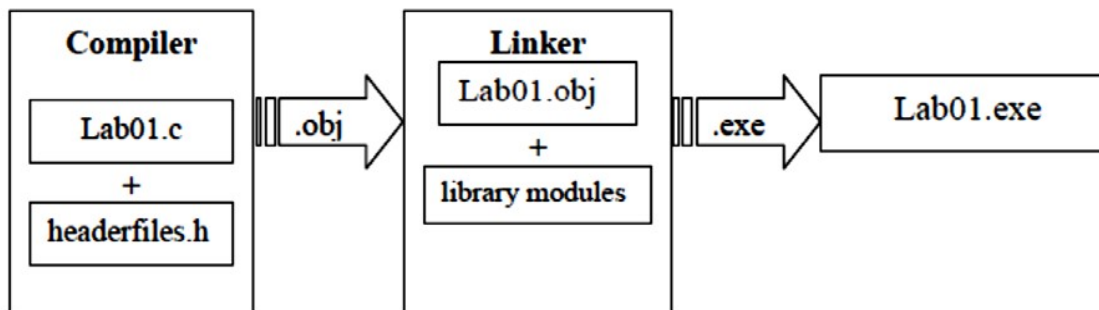


Figure 1.3: Making an executable file

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 code, 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.



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:

NED University of Engineering & Technology
Department of Telecommunications Engineering



Course Code and Title: TC-105 Programming Languages

Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with 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 include 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
<code>\n</code>	New Line
<code>\t</code>	Horizontal Tab
<code>\a</code>	Alert(Bell)
<code>\\</code>	Back Slash
<code>\"</code>	Quotation Mark
<code>\f</code>	Form feed
<code>\r</code>	Carriage Return
<code>\0</code>	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	4
Double	Number(Decimal) 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:

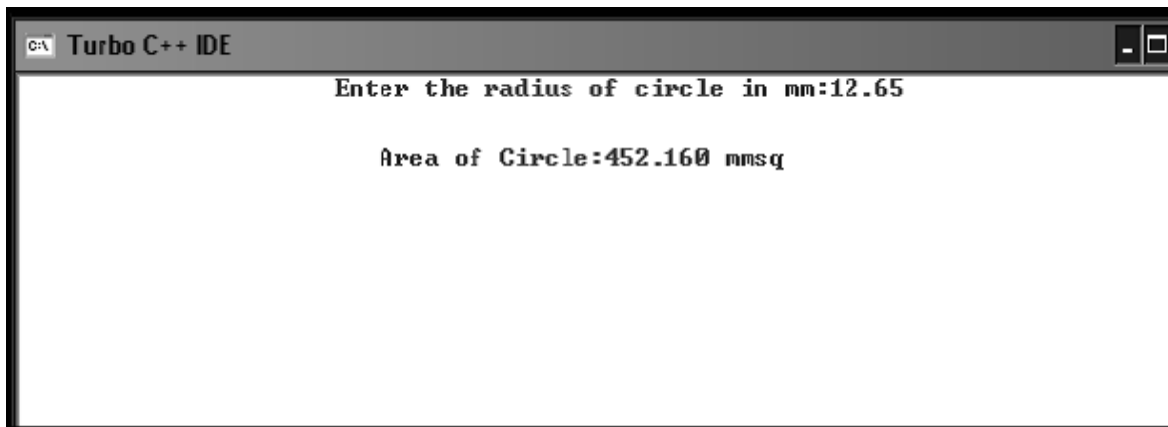


Figure 2.1: Output

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:

NED University of Engineering & Technology
Department of Telecommunications Engineering



Course Code and Title: TC-105 Programming Languages

Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with 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 Operator:

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 can be exemplified as in Table 3.2.

Operators	Operation	Explanation
++a	Pre-Increment	Increments a by 1, then uses new value in the expression in which a resides
a++	Post Increment	Uses the current value of a in the expression in which a resides then increments a by 1
--a	Pre-Decrement	Decrements a by 1, then uses new value in the expression in which a resides
a--	Post Decrement	Uses the current value of a in the expression in which a resides then decrements a 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`. Assumption `int c=3, d=5, e=4, f=6, g=12`

Assignment Operator	Sample Expression	Explanation	Assignment
Subtraction <code>-=</code>	<code>d -=4</code>	<code>d=d-4</code>	1 to d
Multiplication <code>*=</code>	<code>e*=5</code>	<code>e=e*5</code>	20 to e
Division <code>/=</code>	<code>f/=3</code>	<code>f=f/3</code>	2 to f
Remainder <code>% =</code>	<code>g%=9</code>	<code>g=g%9</code>	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.

Precedence	Operator	Description	Associativity
1	<code>++ --</code>	Suffix/postfix increment and decrement	Left-to-right
	<code>()</code>	Function call	
	<code>[]</code>	Array subscripting	
	<code>.</code>	Structure and union member access	
	<code>-></code>	Structure and union member access through pointer	
	<code>(type){list}</code>	Compound literal(C99)	
2	<code>++ --</code>	Prefix increment and decrement	Right-to-left
	<code>+ -</code>	Unary plus and minus	
	<code>! ~</code>	Logical NOT and bitwise NOT	
	<code>(type)</code>	Cast	
	<code>*</code>	Indirection (dereference)	
	<code>&</code>	Address-of	
	<code>sizeof</code>	Size-of	
3	<code>* / %</code>	Multiplication, division, and remainder	Left-to-right
4	<code>+ -</code>	Addition and subtraction	
5	<code><< >></code>	Bitwise left shift and right shift	
6	<code>< <=</code>	For relational operators <code><</code> and <code>≤</code> respectively	
	<code>> >=</code>	For relational operators <code>></code> and <code>≥</code> respectively	
7	<code>== !=</code>	For relational <code>=</code> and <code>≠</code> respectively	
8	<code>&</code>	Bitwise AND	

9	^	Bitwise XOR (exclusive or)	
10		Bitwise OR (inclusive or)	
11	&&	Logical AND	
12		Logical OR	
13	?:	Ternary conditional	
14	=	Simple assignment	Right-to-left
	+= -=	Assignment by sum and difference	
	*= /= %=	Assignment by product, quotient, and remainder	
	<<= >>=	Assignment by bitwise left shift and right shift	
	&= ^= =	Assignment by bitwise AND, XOR, and OR	
15	,	Comma	Left-to-right

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;
float 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) $x = 7 + 3 * 6 / 2$;

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:

NED University of Engineering & Technology
Department of Telecommunications Engineering



Course Code and Title: TC-105 Programming Languages

Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with 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 (> >= < <= && ||)

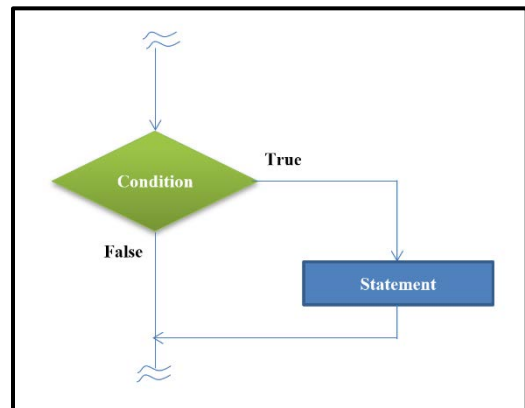


Figure 4.1: Flow Chart (If-statement)

Example:

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

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.

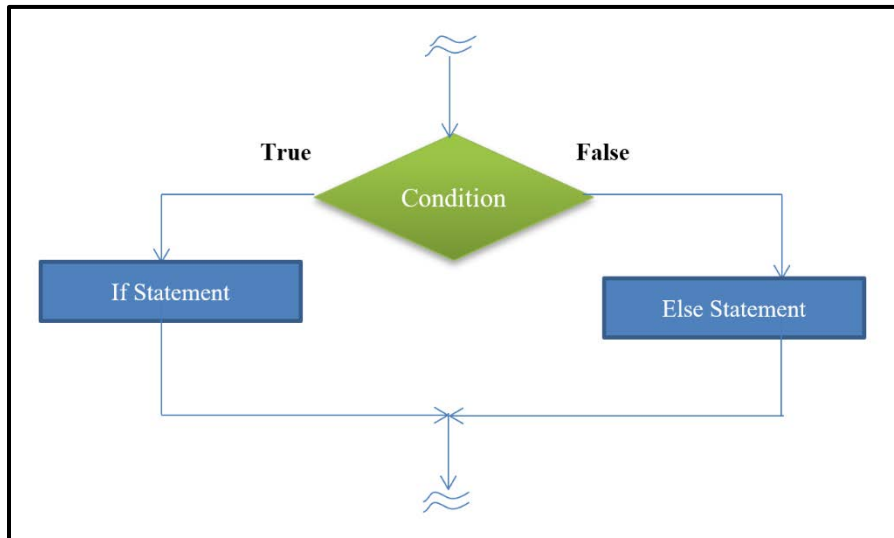


Figure 4.2: Flowchart (If-else statement)

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

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 allows 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 of 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:

NED University of Engineering & Technology
Department of Telecommunications Engineering



Course Code and Title: TC-105 Programming Languages

Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with 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

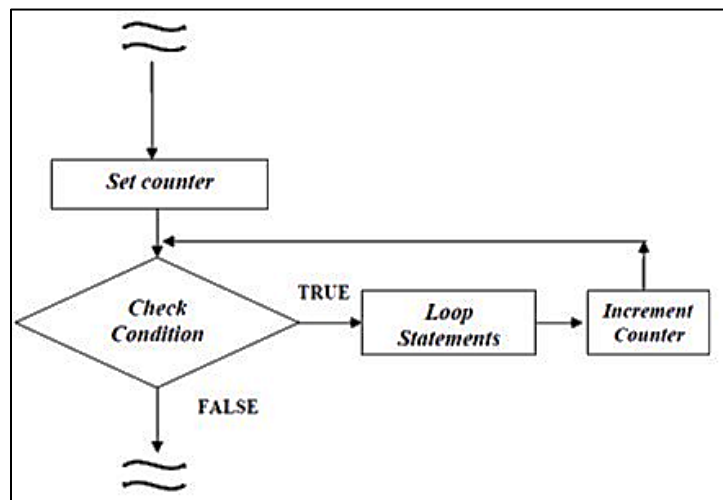


Figure 5.1: Flow Chart (for loop)

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:



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;
}
```

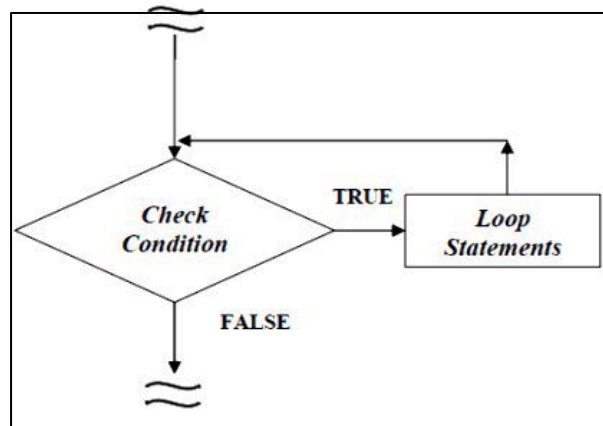


Figure 5.3: Flow chart (while Loop)

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:

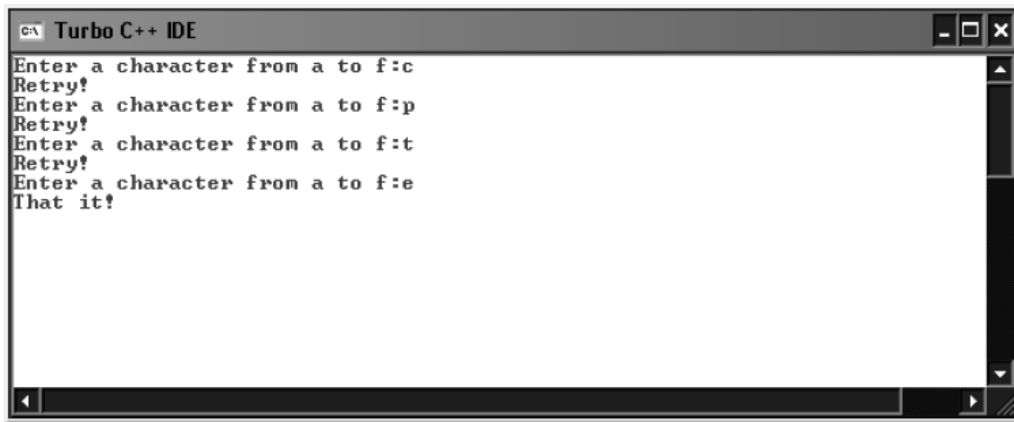


Figure 5.4: Output

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);
```

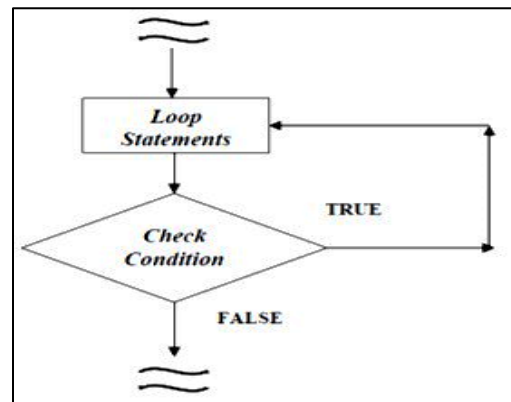


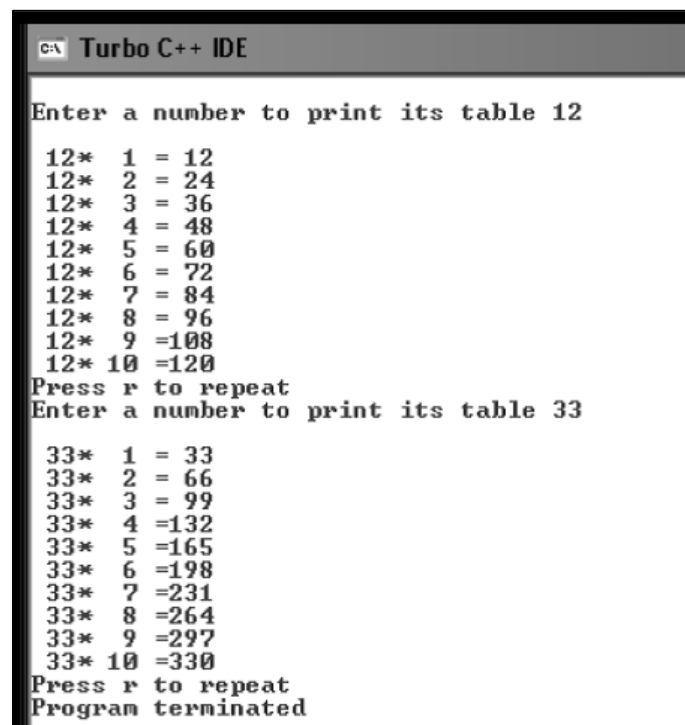
Figure 5.5: Flow chart(do while Loop)

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)
{
    clrscr();
    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:



```
c:\ Turbo C++ IDE

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

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

b. `int a = 10, b = 10; for(int i=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.

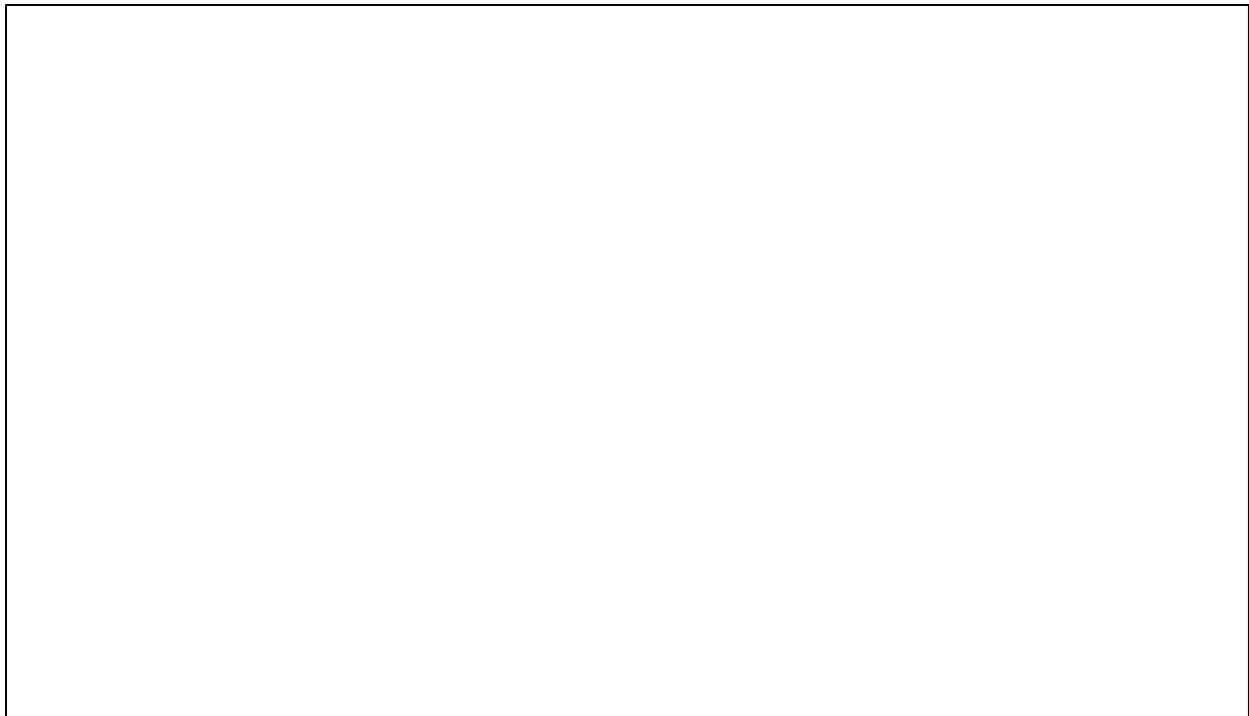
4. Write two programs, which may be used to input a sentence. Terminate when Enter key is pressed. (Use for and while loop).

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:

NED University of Engineering & Technology
Department of Telecommunications Engineering



Course Code and Title: TC-105 Programming Languages

Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with 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 is 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 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 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.

$n + (n-1) + (n-2) + \dots + 3 + 2 + 1$

```
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:

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

```
ii. #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:

NED University of Engineering & Technology
Department of Telecommunications Engineering



Course Code and Title: TC-105 Programming Languages

Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with 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

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>
```

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:

NED University of Engineering & Technology
Department of Telecommunications Engineering



Course Code and Title: TC-105 Programming Languages

Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

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.

RESULT:

NED University of Engineering & Technology
Department of Telecommunications Engineering



Course Code and Title: TC-105 Programming Languages

Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with 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:

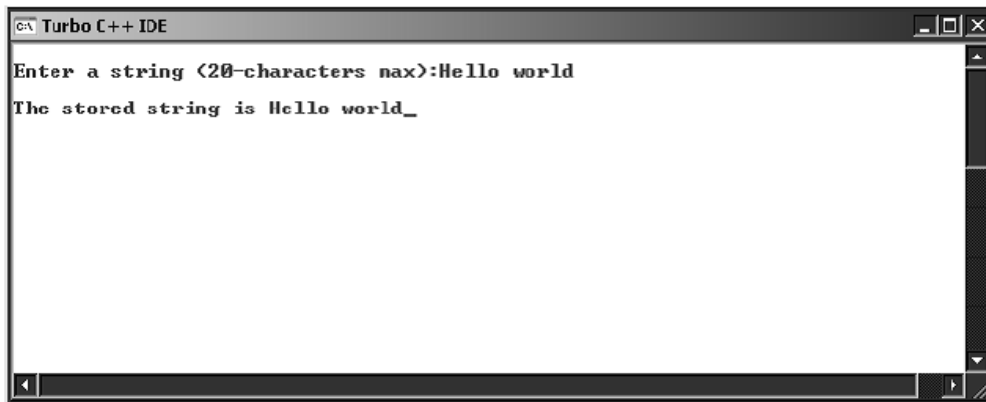
A screenshot of the Turbo C++ IDE window. The title bar reads "Turbo C++ IDE". The main text area contains two lines of text: "Enter a string <20-characters max>:Hello world" and "The stored string is Hello world_". The window has standard Windows-style controls (minimize, maximize, close) in the top right corner and a scroll bar on the right side.

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 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:

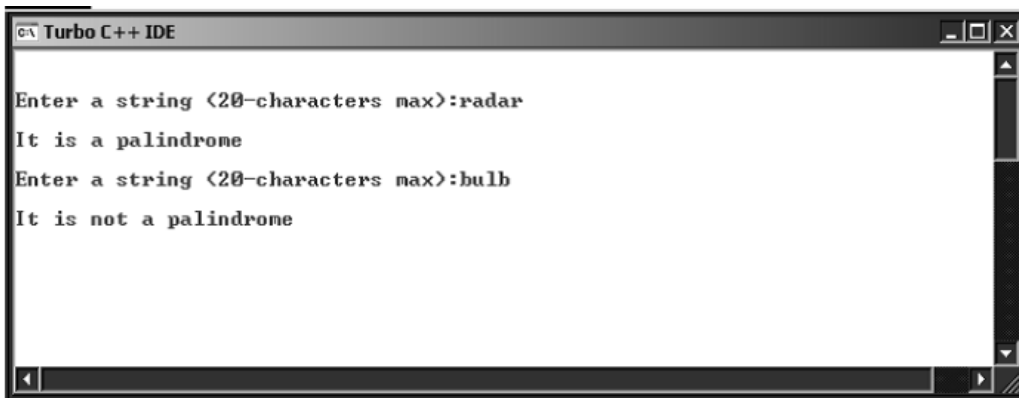


Figure 9.2: Output

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:

NED University of Engineering & Technology
Department of Telecommunications Engineering



Course Code and Title: TC-105 Programming Languages

Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

LAB SESSION 10

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.

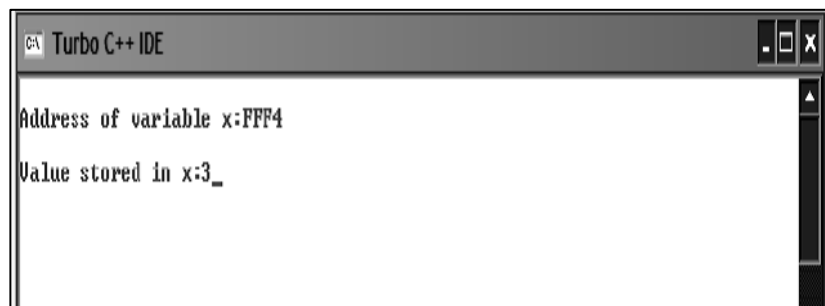
Pointers are used in situations when passing actual value is difficult or undesirable; like, returning more than one value from a function. The concept of pointers also provides an easy way to manipulate arrays and to pass an array or a string from one function to another.

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:

A screenshot of the Turbo C++ IDE's output window. The window title is "Turbo C++ IDE". The output text is as follows:

```
Address of variable x:FFF4
Value stored in x:3_
```

Figure 10.1: 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

1. Multiply Operator (binary)
2. Indirection Operator (Unary)
3. Used in declaration of a Pointer.

Each time you use * , the compiler 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 10.2: 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:

NED University of Engineering & Technology
Department of Telecommunications Engineering



Course Code and Title: TC-105 Programming Languages

Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

LAB SESSION 11

OBJECTIVE:

To introduce problem solving ability using Object Oriented Programming using C++.

THEORY:

C++ is an Object-oriented programming language. It is one of the most popular programming paradigms. The C++ programming language is one of many languages that supports object-oriented programming, alongside Java, C#, Python, and JavaScript. Any Object-Oriented programming-based language is based on classes and objects. The main OOP concepts involved are inheritance, encapsulation, abstraction, and polymorphism. **OOP mimics the real-world examples that consists of objects, rather than the variables and functions we use in software development.** Each object has properties and functions that it can perform. Objects that share properties and functions can be grouped into classes and subclasses.

The foundational OOP concepts are:

Inheritance: The ability of a class to inherit data and behaviors from another class. The class that inherits features is the derived class, while the class it inherits features from is the base class.

Encapsulation: The binding of data to the functions that can perform operations on them. Enables data hiding, another important OOP technique.

Abstraction: The masking of internal functions to present only high-level methods to the user.

Polymorphism: The ability to perform the same task in various ways.

Example:

Consider a sparrow and a penguin. Both have eyes, but they don't share the ability to fly. Not all birds share the function of flight, therefore the base class of birds wouldn't assign the function of flight, but the derived classes of flying birds would assign the flying function.

The syntax for creating a **class object** in C++ is:

```
class ClassName {  
  
    / *member variables and functions*/  
};  
  
int main () {  
    int x; // integer object  
    ClassName c; // ClassName object  
}
```

Example 1:

The following code presents a C++ program to find the sum of individual digits of a positive integer

```
#include<iostream.h>  
int sum_of_digits(int n)  
{  
    int digit,sum=0;  
    while(n!=0)
```

```

{
}
return sum;
}
int main()
{
digit=n%10;
sum=sum+digit;
n=n/10;
int number,digits_sum;
cout<<"Enter Positive integer within the range:";
cin>>number;
digits_sum=sum_of_digits(number);
cout<<"sum of digits of "<<number<<" is "<<digits_sum;
return 0;
}

```

Input:

Enter Positive integer within the range:4321

Output:

sum of digits of 4321 is 10

Example 2

The following C++ Program can be used to generate first n terms of Fibonacci sequence

```

#include<iostream.h>
void fib(int n)
{
int f0,f1,f,count=0;
f0=0;
f1=1;
while(count<n)
{
cout<<f0<<"\t";
count++;
f=f0+f1;
f0=f1;
f1=f;
}
}

int main()
{
int terms;
cout<<"Enter How many terms to be printed:";
cin>>terms;
fib(terms);
return 0;
}

```



```
}
```

Input:

Enter How many terms to be printed:10

Output:

0 1 1 2 3 5 8 13 21 34

LABORATORY TASKS:

1. Write a C++ Program to find both the largest and smallest number in a list of integers.

2. Write a C++ program to sort a list of numbers in ascending order.

3. Write a C++ Program to find both the largest and smallest number in a list of integers.

RESULT

NED University of Engineering & Technology
Department of Telecommunications Engineering
 Course Code and Title: TC-105 Programming Languages



Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors.	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs.	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

LAB SESSION 12

OBJECTIVE:

To investigate different features of Objects in C++.

Theory

In C++ language, everything is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has attributes, such as weight and color, and methods, such as drive and brake. Attributes and methods are basically variables and functions that belongs to the class. These are often referred to as "class members".

A class is a user-defined data type that we can use in our program, and it works as an object constructor, or a "blueprint" for creating objects.

Create a Class

To create a class, use the class keyword:

Example

Create a class called "MyClass":

```
class MyClass {    // The class
public:           // Access specifier
    int myNum;    // Attribute (int variable)
    string myString; // Attribute (string variable)
};
```

Explanation:

The class keyword is used to create a class called MyClass.

The public keyword is an access specifier, which specifies that members (attributes and methods) of the class are accessible from outside the class. You will learn more about access specifiers later.

Inside the class, there is an integer variable myNum and a string variable myString. When variables are declared within a class, they are called attributes.

At last, end the class definition with a semicolon ;.

Create an Object

In C++, an object is created from a class. We have already created the class named MyClass, so now we can use this to create objects.

To create an object of MyClass, specify the class name, followed by the object name.

To access the class attributes (myNum and myString), use the dot syntax (.) on the object:

Example

Create an object called "myObj" and access the attributes:

```
class MyClass {    // The class
public:           // Access specifier
    int myNum;    // Attribute (int variable)
    string myString; // Attribute (string variable)
};
```

```

int main() {
    MyClass myObj; // Create an object of MyClass

    // Access attributes and set values
    myObj.myNum = 15;
    myObj.myString = "Some text";

    // Print attribute values
    cout << myObj.myNum << "\n";
    cout << myObj.myString;
    return 0;
}

```

Example

/ Program to illustrate the working of // objects and class in C++ Programming

```

#include <iostream>
using namespace std;

// create a class
class Room {

    public:
    double length;
    double breadth;
    double height;

    double calculateArea() {
        return length * breadth;
    }

    double calculateVolume() {
        return length * breadth * height;
    }
};

int main() {

    // create object of Room class
    Room room1;

    // assign values to data members
    room1.length = 42.5;
    room1.breadth = 30.8;
    room1.height = 19.2;

    // calculate and display the area and volume of the room

```

```

    cout << "Area of Room = " << room1.calculateArea() << endl;
    cout << "Volume of Room = " << room1.calculateVolume() << endl;

    return 0;
}

// Program to illustrate the working of public and private in C++ Class

#include <iostream>
using namespace std;

class Room {

private:
    double length;
    double breadth;
    double height;

public:

    // function to initialize private variables
    void initData(double len, double brth, double hgt) {
        length = len;
        breadth = brth;
        height = hgt;
    }

    double calculateArea() {
        return length * breadth;
    }

    double calculateVolume() {
        return length * breadth * height;
    }
};

int main() {

    // create object of Room class
    Room room1;

    // pass the values of private variables as arguments
    room1.initData(42.5, 30.8, 19.2);

    cout << "Area of Room = " << room1.calculateArea() << endl;
    cout << "Volume of Room = " << room1.calculateVolume() << endl;

    return 0;
}

```

LABORATORY TASKS:

1. Write down program in C++ that calculates the roots of a quadratic equation by exploiting OOP features in C++.



RESULT:

NED University of Engineering & Technology
Department of Telecommunications Engineering
 Course Code and Title: TC-105 Programming Languages



Laboratory Session No. _____

Date: _____

Software Use Rubric					
Criterion	Level of Attainment				
	Below Average (1)	Average (2)	Good (3)	Very Good (4)	Excellent (5)
Identification of software menu (syntax, components, commands, tools, layout etc.).	Can't identify software menus.	Rarely identifies software menus.	Occasionally identifies software menus.	Able to identify software menus.	Perfectly able to identify software menus.
Skills to use software (schematic, syntax, commands, tools, layout) efficiently.	Can't use software efficiently.	Rarely uses software efficiently.	Occasionally uses software efficiently.	Often uses software efficiently.	Efficiently uses software (syntax, commands, tools, layout)
Adherence to safety procedures and handling of equipment (computing unit, peripheral devices, and other equipment in lab).	Doesn't handle equipment with required care and safety.	Rarely handles equipment with required care and safety.	Occasionally handles equipment with required care and safety.	Often handles equipment with required care and safety.	Handles equipment with required care and safety.
Ability to troubleshoot software errors (detection and debugging).	Not able to troubleshoot the errors	Rarely able to troubleshoot the errors	Occasionally able to troubleshoot the errors	Often able to troubleshoot the errors	Fully able to troubleshoot the errors
Analysis and interpretation of results/outputs.	Not able to analyze and interpret results/outputs	Rarely able to perform the analysis and interpretation.	Occasionally able to perform the analysis and interpretation.	Often able to perform the analysis and interpretation.	Perfectly able to perform the analysis and interpretation.

Weighted CLO (Score)	
Remarks	
Instructor's Signature with Date	

LAB SESSION 13

Open Ended Lab

OBJECTIVE:

Experimenting Blinking LED with Arduino Board

Theory and Experimentation

"Arduino is open-source electronics prototyping platform based on flexible, easy-to-use hardware and software. It is a handy board especially for artists, designers, hobbyists, and anyone interested in creating interactive objects or environments". Arduino can use sensors (for instance, temperature, tilt, and light) to test the environment and can control lights, motors, sound, and more. The Arduino Board can be segregated into three parts.

1. The hardware - Arduino
2. The Programming Environment - the Arduino IDE (Interactive Development Environment)
3. A Community and Philosophy



Figure 1. Segments of Arduino Board

Figure 1 is showing the key elements of an Arduino Board. The following two sections will discuss the hardware and the software.

Hardware. As mentioned earlier, Arduino consists of two major parts: the hardware (the Arduino board) and the software (the IDE). Let us consider hardware:

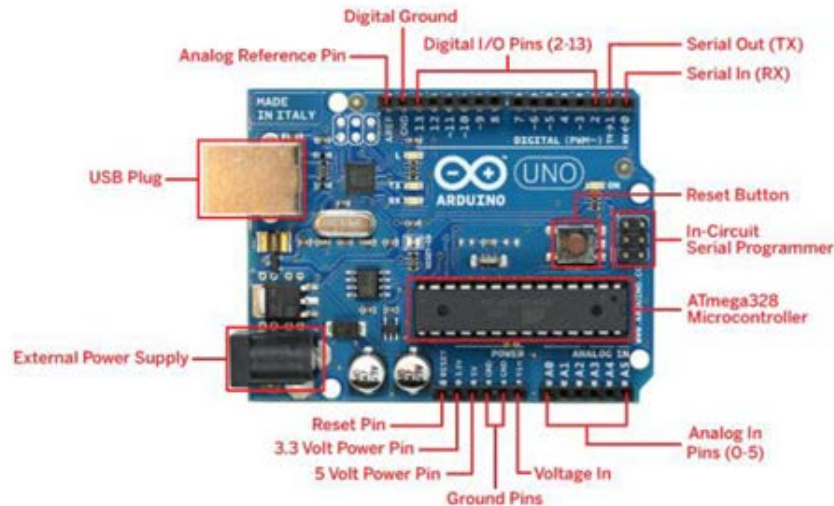


Figure 2. Hardware component of Arduino Board.

Figure 2 illustrates different components of hardware of Arduino Board.

With the appearance of digital technologies and microprocessors, these functions, which were once implemented with wires, were replaced by software programs. Software is easier to modify than hardware. With a few key presses, you can radically change the logic of a device and try two or three versions in the same amount of time that it would take you to solder a couple of resistors. The details of the board are illustrated:

5V is for power. Try to use a red wire to connect to this pin. If you follow this convention, it will be easier to figure out where wires are going to or coming from.

GND is for ground. Notice that there are two ground pins; it does not matter which one you use. Try to use a black wire to connect to this pin.

A0 to A5 are "Analog In" pins. Compared to binary input, which is either HIGH (pulled to +5 volts) or LOW (pulled to ground), the values are between 0 (for 0 volts) and 1023 (for 5 volts).

0 to 13 are Digital I/O Pins. Each of these can be used as an input or an output. You will notice that pins: 3, 5, 6, 9, 10, and 11 have a tilde (~) beside them representing PWM (Pulse Width Modulation). This means that you can output in the range of 0 (~0V) to 255 (~5V) instead of just outputting HIGH or LOW.

Software IDE

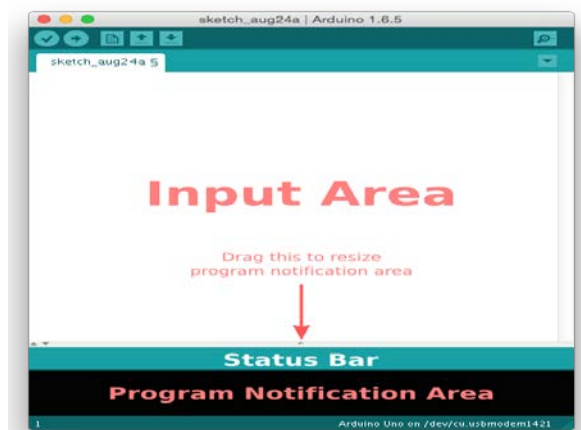


Figure 3. Software IDE

Figure 3 shows the software IDE of the Arduino Board.

There are three main regions:

Input Area - this is where you will type or edit code.

Status Bar - this is where you get information about the status of your code. For instance, "Done compiling." or "Done uploading." are two messages that appear in the Status Bar.

Program Notification Area - this is where you get additional details about the status of your code. Pay attention to this area if you notice that compiling or uploading are not working. Errors will be described here.

Settings

At this point, you should plug your Arduino devices into the USB Port for the Mac.

Under the Tools menu, select Serial Port and the option with (Arduino Uno) at the end. There may be slight differences in in what comes before this depending on your computer and which USB port you used.

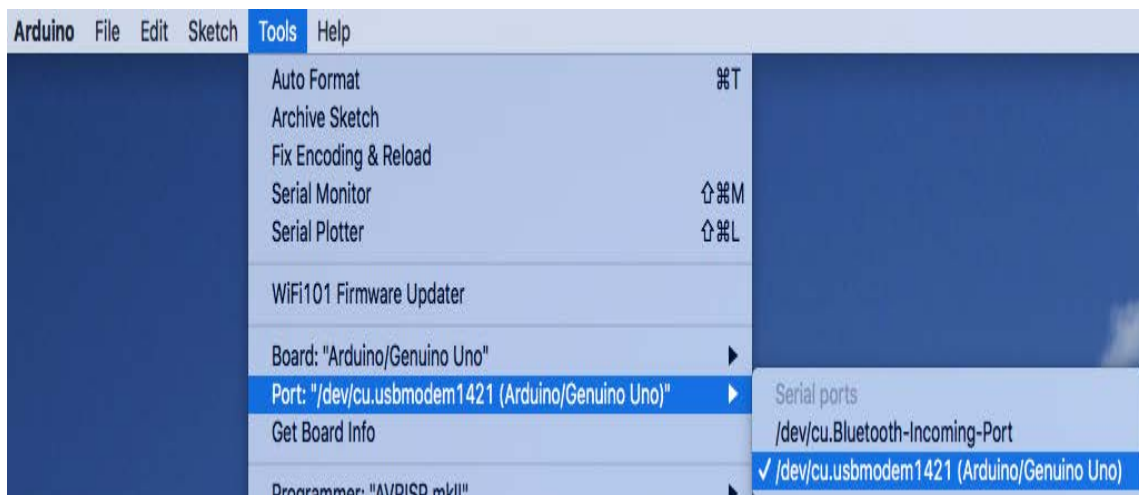


Figure 4 Shows IDE

Note: On Windows machines, the Ports are called COM ports (eg. COM3). Also note that on Windows machines the Tools menu and Upload buttons may be slow if your computer lists too many disconnected COM.ports.

Under the Tools menu, select Board and Arduino Uno

First Blinky Light

And now, for the moment that you have been waiting for, let us see something happening.

Under the File menu, choose: Examples | Basics | Blink. As shown in the diagram below.

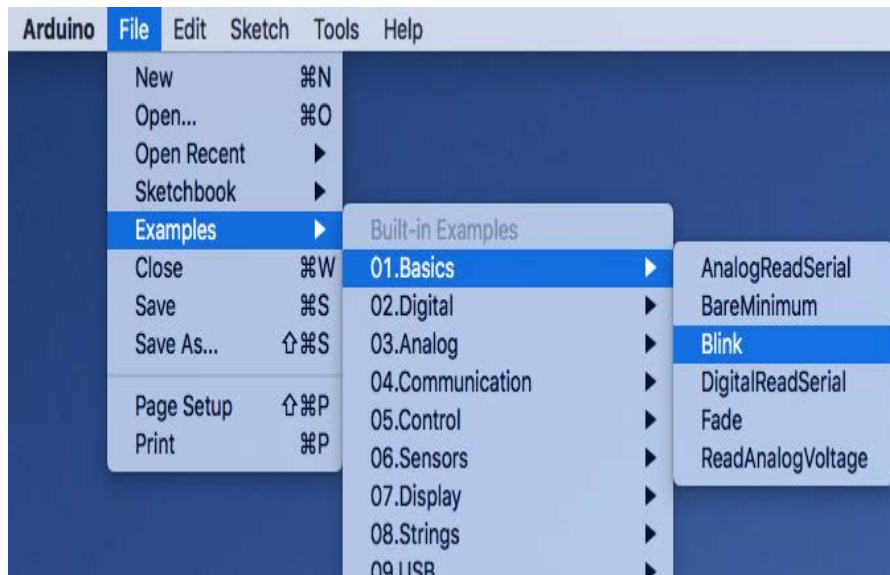



Figure 5 Showing the Implementation of LED Blink Program

A new window will open with code in the input area. You now have your first sketch, which is a special name that Arduino uses to mean a program, or a unit of code that will be run on the Arduino board (hardware). We will be looking at that code more later.




Figure 6. Software IDE Details

Let us try running the code:

Press the Verify button. 

You will notice messages "Compiling..." and then "Done compiling" in the Status bar (below the code).

Press the Upload button. 

You will notice a couple of things: messages of "Uploading to I/O Board.." and then "Done uploading" in the Status bar.

TX and RX lights will blink when the uploading occurs.

You should now notice a light (labeled “L”) blinking on your board. Congratulations, you have gotten your first sketch running on the Arduino board!

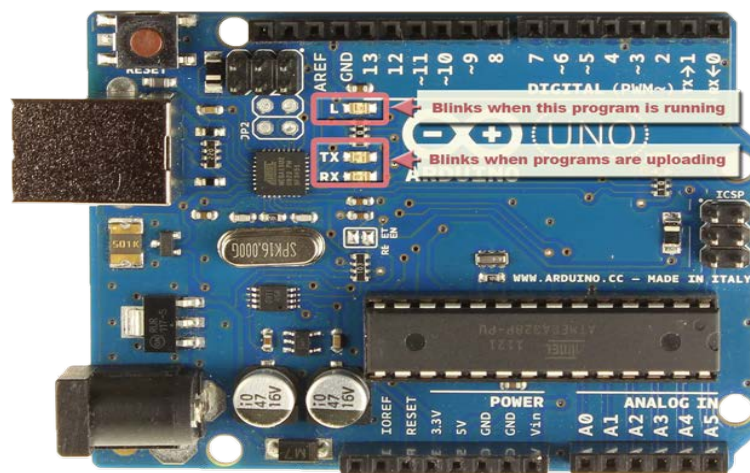


Figure 7. The LED Blinking on Arduino Board

A little more about these lights:

The TX and RX lights flash when data is being transmitted to or from the computer (using the USB connection).

L is a special built-in light that is connected to pin 13. When the pin is a HIGH value, the light is on, when the pin is LOW, it's off. This might not mean anything right now, but you will notice that L blinks on and off after you have uploaded the code.



F/OBEM 01/18/00

NED University of Engineering & Technology
Department of Telecommunications Engineering
Course Code & Title: TC-105 Programming Language
Assessment Rubric for CEP

Criterion	Level of Attainment				
	Below Average (0)	Average (1)	Good (2)	Very Good (3)	Excellent (4)

Student's Name: _____

Roll No.: _____

Total Score = _____

Instructor's Signature: _____