



**Department of Electronic Engineering
NED University of Engineering & Technology,**

PRACTICAL WORK BOOK

for the course

PROGRAMMING LANGUAGES
(EL-255)

Instructor name: _____

Student Name: _____

Roll no.: _____ **Batch:** _____

Semester: _____ **Year:** _____

Department: _____

**LABORATORY WORK BOOK
FOR THE COURSE**

EL-255 PROGRAMMING LANGUAGE

Prepared by:

(Dr. Yawar Rehman)

Reviewed by:

(Dr. Sadia Muniza Faraz)



Approved By:

**The Board of Studies of Department of Electronic
Engineering**

CONTENTS

Lab no.	Date	Title of Experiment	CLO	Page no.
1		To work and design programming solutions in the TURBO C IDE programming environment.	3	4
2		To use the C building blocks for designing the programming solutions.	3	10
3		To design programs using FOR and nested FOR loops to loop full or a part of the code in C.	3	14
4		To design program using WHILE and nested WHILE loops to loop full or a part of the code in C.	3	17
5		To design programs using the DO-WHILE for looping the code in C.	3	20
6		To design programs using IF and IF-ELSE statements for decisions in the C code.	3	23
7		To design programs using ELSE-IF and Switch statement for decisions in the C code.	3	26
8		To use the Debugging tool in the C code for designing the programming solutions.	3	30
9		To design programs using Function without input and output arguments. In addition, the students would also be able to use Function without input argument and with output argument.	3	34
10		To design programs using programming arrays in the C language.	3	39
11		To understand and design the programs using the concept of pointers with arrays in the C language.	3	44
12		To understand and implement the concept of pointers with functions in the C language for designing the programming solutions.	3	49
13		To apply their programming skills to design and develop solutions based on the trending and state-of-the-art technologies.	3	53

Lab # 01

Introduction of Turbo C IDE and Programming Environment

Objective: At the end of this lab exercise, students would be able to work and design programming solutions in the TURBO C IDE programming environment.

Theory: 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 as shown in Figure 01 may be used to invoke all the operations necessary to develop the program, including editing, compiling, linking, and debugging and program execution.

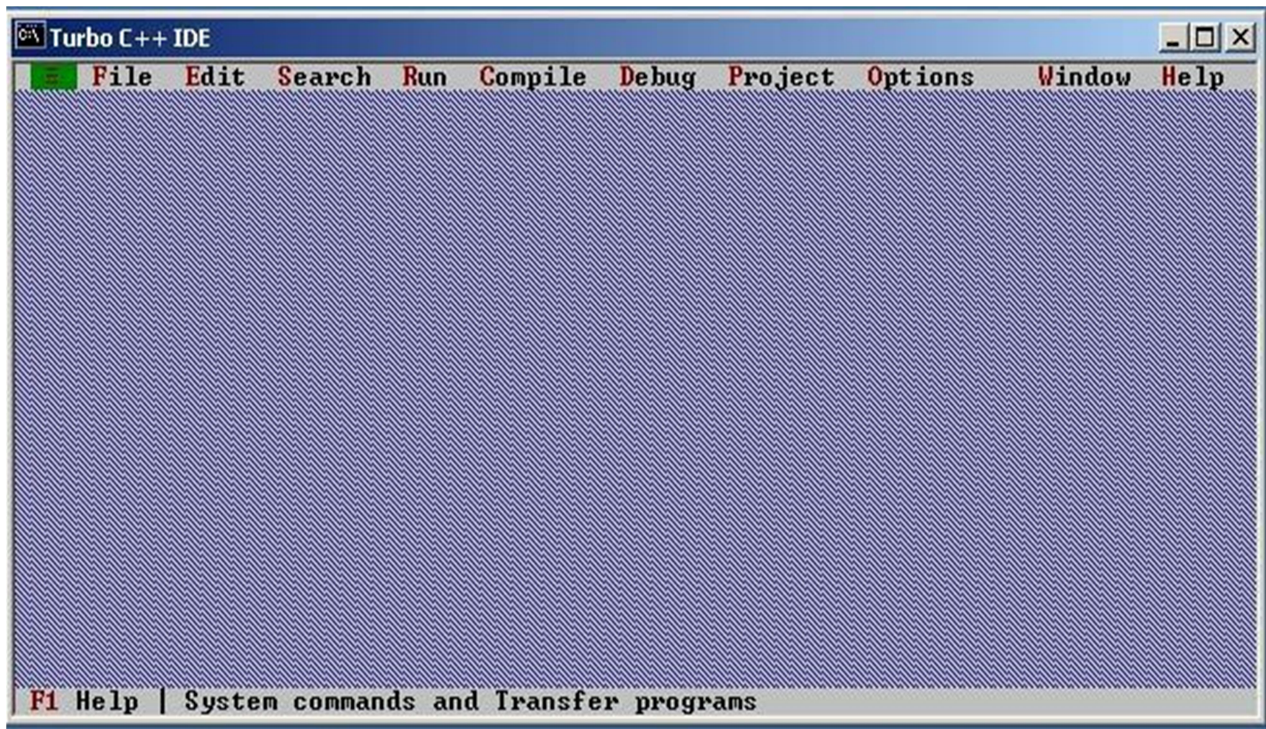


Figure 01: Turbo C IDE environment

Invoking the IDE: To invoke the IDE from the windows you need to double click the TC icon in the following path directory

"C:\TC\BIN" → Default Directory

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.

Using Menu: 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 as shown in Figure 02. For this, open file menu and click “new”. A window will appear on the screen where the program may be typed.



Figure 02: Opening new editor file

Writing a Program: When the Edit window is active, the program may be typed. Use the certain key combinations to perform specific edit functions.

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.

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.

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 relinking, and then rerunning the program. A sample program in Turbo C environment is shown in the Figure 03.

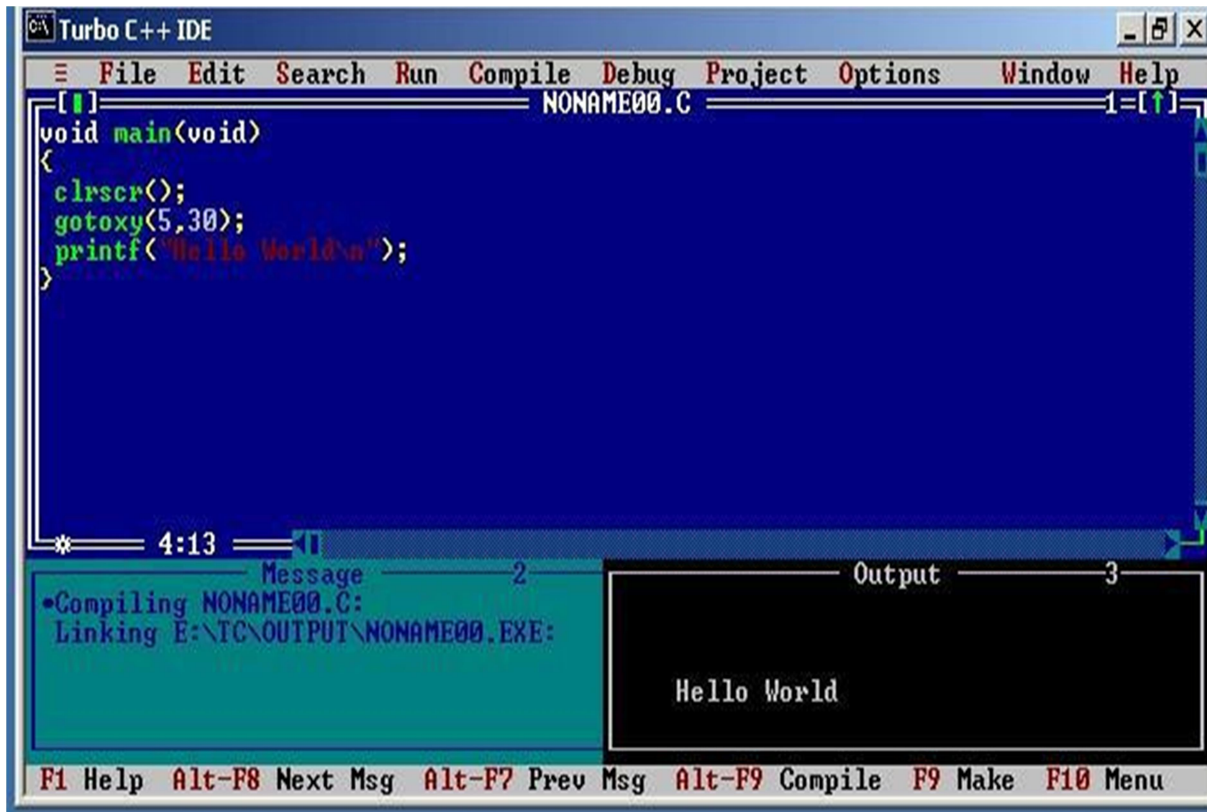


Figure 03: Sample program in Turbo C environment

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.

Exercise

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 ';'. And insert getch(); after printf command. What is the output?



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 01

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 # 02

C Building Blocks

Objective: At the end of this lab exercise, students would be able to use the C building blocks for designing the programming solutions.

Theory: In any programming language, there are building blocks that are used by the programmer to bolster or support the written code. Following are the types of building blocks.

- a) Constants
- b) Variables
- c) Operators
- d) Acquire input from the user (scanf(), getch(), getche())
- e) Displaying output (printf(), format specifiers, escape sequence)

Variables and Constants: If the value of an item is to be changed in the program then it is a variable. If it will not change then that item is a constant. The various variable types (also called data type) in C are: int, float, char, long, and double. They are also of the type signed or unsigned.

Format Specifiers: Format Specifiers tell the printf statement where to put the text and how to display the text. Few of the format specifiers are as follows.

- ✓ %d is used to represent integers
- ✓ %c is used to represent characters
- ✓ %f is used to represent floating numbers

Escape Sequences: Escape Sequence causes the program to **escape** from the normal interpretation of a string, so that the next character is recognized as having a special meaning. The back slash “\” character is also called Escape Character

- ✓ \n => new line
- ✓ \t => tab
- ✓ \b => back space
- ✓ \r => carriage return
- ✓ \" => double quotations
- ✓ \\ => back slash

Acquire Input From the User: The input from the user can be acquired by using the following built-in C functions.

- ✓ scanf()
- ✓ getch()
- ✓ getche()
- ✓ getchar()

Operators: There are various types of operators that may be placed in three categories:

- ✓ Basic: + - * / %
- ✓ Assignment: = += -= *= /= %=

- ✓ (++ , -- may also be considered as assignment operators)
- ✓ Relational: < > <= >= == !=
- ✓ Logical: && , || , !

Exercise

1. Write a program that shows the function of each escape sequence character.

For example

```
printf("alert ring bell rings like \a\a\a\a\a\a\a\a\a\a\a\a\a\a");  
printf("the tab is inserted like \t this");
```

2. What will be the output when

a=5,

```
printf("%d", ++a);  
printf("%d", a++);
```

3. Try making a program to check and validate the working of relational operator.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 02

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 # 03

Loops: FOR and Nested FOR loop

Objective: At the end of this lab exercise, students would be able to design programs using FOR and nested FOR loops to loop full or a part of the code in C.

Theory: Loops are used in a programming language to iterate a block of code for several times. The iterations can be predefined and can also be process dependent. The FOR loop, provides a utility to the user to repeat a block of code for predefined number of times. The syntax of FOR loop in C language is as follows.

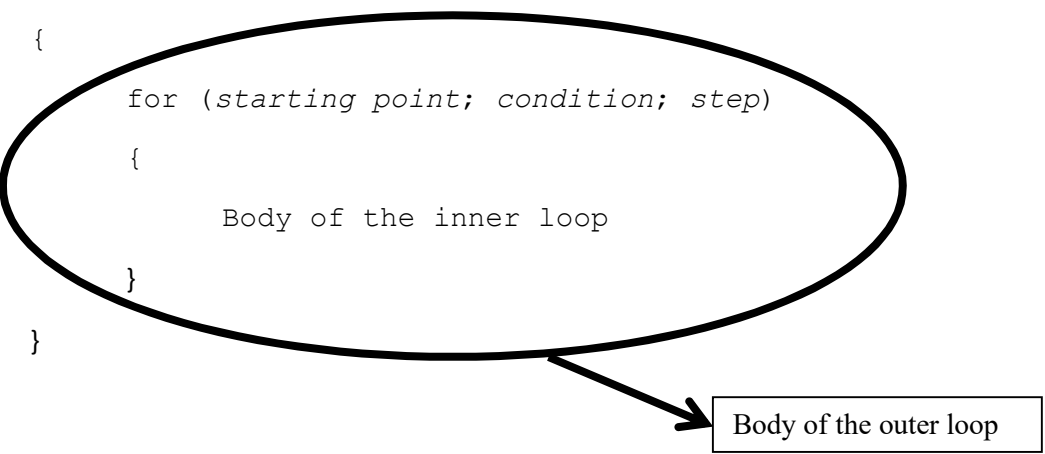
Syntax: The syntax or format for using the FOR loops in C language is as follows.

```
for (starting point; condition; step)
{
    Body of the loop
}
```

Commentary on the syntax: In the syntax of FOR loop, starting point indicate the position where the user wants to start the iteration. Condition specifies the amount of times user wants to iterate through the FOR loop. And finally, the step indicates the strides user wants to take from the starting point towards the end point as expressed by the defined condition.

Nesting the FOR loop: The term nesting the loop in programming languages means to initialize a loop inside the body of another loop. Loops can be nested to achieve the desired output. The format for the nested FOR loops can be expressed as follows.

```
for (starting point; condition; step)
{
    for (starting point; condition; step)
    {
        Body of the inner loop
    }
}
```



Body of the outer loop

Exercise

1. Make a program that iterates the following lines of code for twenty times using FOR loop.

```
printf("alert ring bell rings like \a\a\a\a\a\a\a\a\a\a\a\a\a\a");  
printf("the tab is inserted like \t this");
```

2. Make a program to that prints any number or character in 12 × 12 dimensional matrix using the nested FOR loop.

3. Make a program to print Chess board at the output using the nested FOR loop. The chess has a total of sixty four squares.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 03

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 # 04

Loops: WHILE and Nested WHILE loop

Objective: At the end of this lab exercise, students would be able to design program using WHILE and nested WHILE loops to loop full or a part of the code in C.

Theory: Loops are used in a programming language to iterate a block of code for several times. The iterations can be predefined and can also be process dependent. The WHILE loop, provides a utility to the user to repeat a block of code as per the process requirements. The process code is written in the body of the loop. Once, a condition is met in the process, the loop stops the iteration. The syntax of WHILE loop in C language is as follows.

Syntax: The syntax or format for using the WHILE loops in C language is as follows.

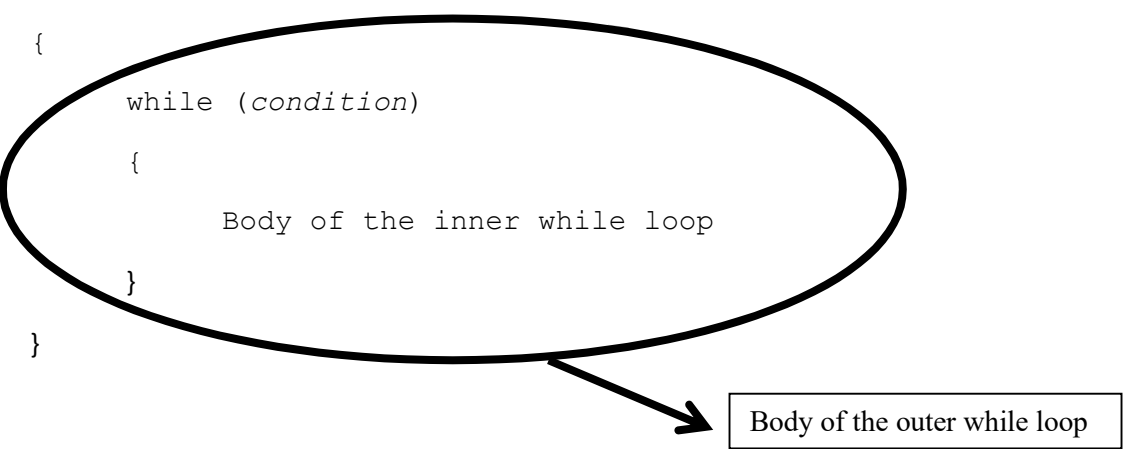
```
while (condition)  
{  
    Body of the loop  
    condition variable  
}
```

Commentary on the syntax: In the syntax of WHILE loop, condition indicates the compiler whether to quit or continue iterating the WHILE loop. When the condition is true, the body of the loop is iterated. When the condition becomes false, compiler exits the loop and iterations stop.

The two states of the condition in the WHILE loop are depended on the process. Process is the code that the user wrote inside the body of the loop. In each iteration, the condition variable defined inside the body of loop is updated. When the condition variable update is in-line with the user's intent, we say the condition is true and the body of the loop continues to iterate. However, when the condition variable update is not as desired by the user's intent, we say the condition became false and the compiler exits the body of the loop.

Nesting the WHILE loop: The term nesting the loop in programming languages means to initialize a loop inside the body of another loop. Loops can be nested to achieve the desired output. The format for the nested WHILE loops can be expressed as follows.

```
while (condition)  
{  
    while (condition)  
    {  
        Body of the inner while loop  
    }  
}
```



Body of the outer while loop

Exercise

1. Make a program that prints your name for ten times using WHILE loop.

2. Make a program that accepts a positive number from the user. Use the WHILE loop to reduce that number to zero.

3. Make a program that takes three characters from the user "abc". The program will not exit and keep on asking the user to enter the correct character unless the user enters the said characters in the same sequence. Use the nested WHILE loops to design such program.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 04

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 # 05

Loops: DO-WHILE loop

Objective: At the end of this lab exercise, students would be able to design programs using the DO-WHILE for looping the code in C.

Theory: Loops are used in a programming language to iterate a block of code for several times. The iterations can be predefined and can also be process dependent. Sometimes, it becomes necessary in a program to run the code for the first time and then check or validate the condition of the WHILE loop. DO-WHILE loop provide this flexibility of running the program code for the first time and then validate the defined condition.

Syntax: The syntax or format for using the DO-WHILE loop in C language is as follows.

```
do
{
    Body of the loop
    condition variable
} while (condition);
```

Commentary on the syntax: In the DO-WHILE loop, the body of the loop is defined first and then the condition of the DO-WHILE loop is defined. For this reason, the C compiler first runs the body of the loop for the first time and then check and validates the condition. If the condition is false the compiler will exit the loop, hence stopping the loop iterations. Whereas, if the condition of the DO-WHILE is true, the body of the loop is executed and then again the condition is checked and validated.

Exercise

1. Make a program that guesses a letter entered by the user (letter guessing game) using the DO-WHILE loop.

2. Make a program that accepts a positive number from the user. Use the DO-WHILE loop to reduce that number to zero.

3. Design a letter guessing game. Once the user has guessed the correct the program would ask the user to play again the game or not. Use the DO-WHILE loop for this program.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 05

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 # 06

Decisions: IF and IF-ELSE statement

Objective: At the end of this lab exercise, students would be able to design programs using IF and IF-ELSE statements for decisions in the C code.

Theory:

Normally, the program flows along line by line in the order in which it appears in your source code. However, it is sometimes required to execute a particular portion of code only if certain condition is true; or false i.e. we need to make a decision in the program. The C language provides the utility of using the decision statements for this purpose. In this lab, we will discuss and understand the working of IF and IF-ELSE decision statements.

The if statement: The if statement enables user to run a part of code when certain condition is true (such as whether two variables are equal). The program may be branch to different parts of the code, depending on the result or the conditions. The relational and logical operators can also be included. The syntax of the IF statement is as follows.

Syntax: The format or syntax of the if statement in the C language is as follows.

```
if (expression)
{
    Body of the if statement;
}
```

The if-else statement: Often the program may need to execute a block of code when the defined condition inside if statement is true and another block of code when the condition is false. For this scenario, the keyword 'else' can be used.

Syntax: The format or syntax of the if-else statement in the C language is as follows.

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

Exercise

1. Make a program that accepts an integer and classifies it as even or odd number with the help of if-else statement.

2. Make a letter guessing game using if statement.

3. Make a program that classifies the letter obtained from the user as vowel or consonant.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 06

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 # 07

Decisions: ELSE-IF and Switch statement

Objective: At the end of this lab exercise, students would be able to design programs using ELSE-IF and Switch statement for decisions in the C code.

Theory:

Normally, the program flows along line by line in the order in which it appears in your source code. However, it is sometimes required to execute a particular portion of code only if certain condition is true; or false i.e. we need to make a decision in the program. The C language provides the utility of using the decision statements for this purpose. In this lab, we will discuss and understand the working of ELSE-IF and Switch statements.

The ELSE-IF statement:

The IF-ELSE statement renders the program with flexibility in taking decisions. This is a very important statement that is used in almost all programming languages to incorporate the decisions. However, the use of IF-ELSE statement is limited. When in a program, the programmer is required to check multiple true or false conditions; ELSE-IF statement can be easily used. This ELSE-IF statement provides the programmer more control of the program.

Syntax: The format or syntax of the ELSE-IF statement in the C language is as follows.

```
if (expression)
{
    statement;
}
elseif (expression)
{
    statement;
}

elseif (expression)
{
    statement;
}
```

The Switch statement:

Switch statements allow you to branch on any of a number of different values. There must be break at the end of the statements of each case otherwise all the preceding cases will be executed including the default condition. The general form of the switch statement is as follows.

Syntax: The format or syntax of the Switch statement in the C language is as follows.


```
switch (identifier variable)
{
case identifier1: statement;
case identifier2: statement;
.
.
.
case identifierN: statement;
default: statement;
}
```

Exercise

1. Make a program for simple calculator using ELSE-IF statement.

2. Make a program for simple calculator using Switch statement.

3. Insert more calculation operation in the program of question 2.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 07

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 # 08

Debugging

Objective: At the end of this lab exercise, students would be able to use the Debugging tool in the C code for designing the programming solutions.

Theory:

The compiler of a programming language can detect the format or syntax errors in the program. The logical errors are human induced error in the programming logic that cannot be detected by the compiler. Such errors are called logical errors and there are separate tools in every language. These tools provide the programmer a utility of finding the logical error easily. These tools are commonly known as debugging tools.

One Step at a Time:

The first thing that the debugger can do is slow down the operation of the program. One trouble with finding logical errors is that a typical program executes in a few milliseconds, so all we may see is its final state. By invoking C++'s single-stepping capability, we may execute just one line of the program at a time. This way we can follow the program.

Consider the following program:

```
void main(void)
{
int number=-50, answer=-1;
if(number < 100)
{
if(number > 0)
answer = 1;
else
answer = 0;
}
printf("answer is %d\n", answer);
}
```

Our intention in this program is that when number is between 0 and 100, answer will be 1, when the number is 100 or greater, answer will be 0 , and when number is less than 0, answer will retain its initialized value of -1. When we run this program with a test value of -50 for number, we find that answer is set to 0 at the end of the program, instead of staying -1.

We can understand the problem, if we single step through the program. To do this, simply press the [F7] key. The first line of the program will be highlighted. This highlighted line is called the run bar . Press [F7] again. The run bar will move to the next program line. The run bar appears on the line about to be executed. You can execute each line of the program in turn by pressing [F7]. Eventually you'll reach the first if statement

```
if (number < 100 )
```

This statement is true (since number is -50); so, as we would expect the run bar moves to the second if statement.

```
if (num>0)
```

This is false. Because there's no else matched with the second if, we would expect the run bar to the printf() statement. But it doesn't! It goes to the line

```
answer = 0;
```

Now that we see where the program actually goes, the source of the bug should become clear. The else goes with the last if, not the first if as the indenting would lead us to believe. So, the else is executed when the second need to put braces around the second if statement is false, which leads to erroneous results. We if, or rewrite the program in some other way.

Watches

Single stepping is usually used with other features of the debugger. The most useful of these is the watch (or watch expression). This lets you see how the value of variable changes as the program runs. To add a watch expression, press [Ctrl+F7] and type the expression.

Breakpoints:

It often happens that debugged part of the program must deal with a bug in another section, and we don't want to single-step through all the statements in the first part to get to the section with the bug. Or we may have a loop with many iterations that would be tedious to step through. The way to do this is with a breakpoint.

A breakpoint marks a statement where the program will stop. If we start the program with [Ctrl][F9], it will execute all the statements up to the breakpoint, then stop. We may now examine the state of the variables at that point using the watch window.

Installing breakpoints:

To set a breakpoint, first position the cursor on the appropriate line. Then select Toggle Breakpoint from the Debug menu (or press [Ctrl][F8]). The line with the breakpoint will be highlighted. We may install as many breakpoints as we want. This is useful if the program can take several different paths, depending on the result of if statements or other branching constructs.

Removing Breakpoints:

We may remove a single breakpoint by positioning the cursor on the line with the breakpoint and selecting Toggle breakpoint from the Debug menu or pressing the [Ctrl][F8] combination (just as we did to install the breakpoint). The breakpoint highlight will vanish. We may all set Conditional Breakpoints that would break at the specified value only.

Exercise

1. Use breakpoints and single step debugging in the aforementioned program.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 08

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 # 09

Functions

Objective: At the end of this lab exercise, students would be able to design programs using Function without input and output arguments. In addition, the students would also be able to use Function without input argument and with output argument.

Theory:

In programming, there is a need to repeat the block of code at different places with different input and output variables in the programming code. The programmer may use the same lines of code inside the program at different places. However, to repeat the same lines of code would seem redundant. Therefore, the programmer may use the utility of functions in C language.

A subroutine can be made by wrapping the lines of code into a function. Those subroutines can be accessed by their call names. This becomes easier for the programmer to handle different snippets of code inside the program. The functions can be categorized into three categories. First one is the built-in functions that are provided by the language. In the C language, these functions are e.g. scanf(), printf(), and clrscr(). The second category of functions can be termed third party functions. These are available on the internet and can be added in the C language. The third category is the user defined function. The programmer may also make their own functions in the C language. The objective of this lab is to introduce the functions such that the users may build their own functions.

Syntax: The format or syntax of making a function in the C language is as follows.

```
OutputArguments functionName (inputArguments)
{
    function body;
}
```

Commentary: The output arguments are the variables we define for getting the output from the function. Function name signify the call name of the functions. The input arguments are the input variables given to the function for the processing in the function body.

The functions can be divided into four types.

1. Function without input and output arguments
2. Function with input and without output arguments
3. Function without input and with output arguments
4. Function with input and output arguments

Function without input and output arguments

These functions do not process any variable and do not give any output variable. The example of such functions is as follows.

```

void line(void);                //prototype declaration

void main(void)
{
    line();
    printf("EL255 programming languages");
    line();
}

void line(void)
{
    int i;
    for(i=1; i<=40; i++)
    {
        printf("\xDB");
    }
    printf("\n");
}

```

Function with input and without output arguments

These functions do process an input variable but do not return any output variable. The example of such functions is as follows.

```

void line(int);                //prototype declaration

void main(void)
{
    line(20);
    printf("EL255 programming languages");
    line(45);
}

void line(int k)
{
    int i;
    for(i=1; i<=k; i++)
    {
        printf("\xDB");
    }
    printf("\n");
}

```

Function without input and with output arguments

These functions do not take any input variable but returns an output.

```

int getmins(void);            //prototype declaration

void main(void)
{
    int mins1, mins2;

```

```

printf("type first time (like 3:22): ");
mins1 = getmins();
printf("type second time (like 4:22): ");
mins2 = getmins();
printf("the difference in time is %d minutes.",mins2-mins1);
}

int getmins(void)
{
int hours, minutes;

scanf("%d:%d",&hours,&minutes);
return( hours*60 + minutes);    //return statement is used for output
}

```

Function with input and output arguments

These functions accepts take input variable and returns an output variable.

Exercise

1. Make a program for simple calculator using the functions that can accept an input variable and return an output.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 09

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 # 10

Arrays

Objective: At the end of this lab exercise, students would be able to design programs using programming arrays in the C language.

Theory:

In programming, we may need to group multiple variables of same data type instead of declaring separate variables for each element. For example, the temperature of a week can be stored and processed using seven variables as follows.

```
int mon, tue, wed, thu, fri, sat, sun;
```

Looping through each of the variable would be difficult task. Secondly, it will be very cumbersome when temperature for a month needs to be entered and stored. The programming languages provide a utility of arrays to gather and process same data type easily. In C language an array of seven elements can be initialized using the square bracket as follows.

```
int temp[7];
```

This array is of integer class type; its name is 'temp'; and it has total of seven elements that can be accessed from 0 to 6. Now with the help of array we may easily process the data of same class type. Strings are also arrays. A particular example of initializing the string array is as follows.

```
char name[40];
```

Now using this array we may acquire the name of some other data of string class type into the variable 'name'. An example program for accessing the one dimensional array is as follows.

Program 1

```
main()
{
    int temp[4];
    clrscr();
    for (i=0;i<=3;i++)
    {
        printf("enter the temperature for day %d:",i);
        scanf("%d",&temp[i]);
    }
    system("pause");
}
```

The aforementioned program will acquire the temperature for four days and save those values in the array 'temp'. We may easily increase or decrease the size of array in this program. It should be noted that the size of array is pre-defined using for loop.

More than one dimensional array

In programming we may also use bigger arrays. Bigger array in a sense that an array may have more dimensions and data storage capacity. These arrays are known as multi-dimensional arrays. A two dimensional array can be declared as follows.

```
int rollno[10][2];
```

This array with the call name of rollno has two dimensions. There would be ten rows and two columns in this array. A simple program illustrating the control of accessing two dimensional arrays is as follows.

Program 2

```
#define rows 10          //define directive or macros
#define cols 2           //define directive or macros

main()
{
    float students[rows][cols];
    float number, expense;
    int index=0, outdex;
    printf("enter your rollno and weekly expenses (like 007 200):\n");
    printf("enter 0 0 to quit");
    do
    {
        printf("student rollno and weekly expense: ");
        scanf("%f %f",&number, &expense);
        students[index][0]=number;
        students[index][1]=expense;
    }
    while(students[index++][0] != 0);
    // for displaying the output
    for(outdex=0; outdex<index-1; outdex++)
    {
```



```
    printf("student %3.0f", students[outdex][0]);  
    printf("spent %7.2f", students[outdex][1]);  
}  
system("pause");  
}
```

Exercise

1. Complete the Program 1 by taking the average of the acquired temperatures.

2. Change for loop to the while loop in the Program 1.

3. Design a program that check the bounds of an array.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 10

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 # 11

Introduction to Pointers

Objective: At the end of this lab exercise, students would be able to understand and design the programs using the concept of pointers and use pointers with arrays in the C language.

Theory:

A pointer is a methodology in C language to access a variable let's say named 'x' without directly referring to the variable named 'x'. Let us understand this concept with the help of a simple analogy.

Analogy

Suppose Mr. Fareed went to a restaurant and sat on table no. 2. He then ordered a cup of tea. The restaurant owner without knowing the name of the customer will guide the server to deliver a cup of tea at the table no. 2. The server will then deliver or serve table no. 2 with a cup tea.

In the programming language, the pointers work in the similar way. In this analogy, we may swap Mr. Fareed as a variable 'x', the restaurant server as the program / compiler, and the server as the pointer. The program / compiler without the need of knowing the variable name 'x' can access the value using its address using the pointers.

Hence, we now may understand that pointer is a methodology in C language to access a variable let's say named 'x' without directly referring to the variable named 'x'.

There are various uses of pointer in the C language. The pointer may come handy to return more than one value from a function. They provide convenience of passing data such as variables or array from one function to another function. And the pointer can process or manipulate the arrays more easily. However, in this lab we will only touch the most important aspect of pointers i.e. to access the variable through its memory address and without really accessing through its name.

Program 1

```
// function that returns two values

void rets2(int *, int *); // prototype; * is indirection operator

void main(void)
{
    int x, y;
    rets2(&x, &y);
    printf("first is %d, second is %d",x,y);
}

void rets2(int *px, int *py)
{
```

```

*px = 3;

*py = 5;

}

```

Commentary

The Program 1 shows the use of pointers with functions. This program assigns the values of 3 and 5 to the variables x and y, respectively. It must be noted that without really accessing the variable name, their values has been accessed and changed. This is done with the help of address of variable and pointers.

The address of x and y has been provided in the rets2 function using the address operator '&'. In the rets2 function input *px and *py are declaration of the two pointer variables. We know a variable is a pointer with the help of * indirection operator. The address values are now assigned to the pointer variables. And in the function body of rets2 function, the content of the addresses in px and py as been set to 3 and 5, respectively.

Pointers with arrays

Let us now discuss the pointers with arrays. The use of pointers with arrays provides us a very powerful tool for accessing and managing data. In fact, a computer does not understands array, however, the compiler help to translate the arrays into the pointers notation before processing those to the machine.

The following program will illustrate the use of pointers with arrays to access the data inside arrays easily.

Program 2

```

void main(void)

{

int nums[] = {92, 81, 70, 68, 59};

int dex;

for(dex=0; dex<5; dex++)

{

    printf("%d\n", *(nums+dex));

}

system("pause");

}

```

Commentary

Program 2 illustrates a simple operation of displaying the array elements using pointers. In the program, an array of class int has been initialized and contains five elements. For loop variable is dex and its

starting from zero. In the body of the for loop, we may see that there is a single printf statement that contains the use of indirection operator i.e. pointers.

In the statement `*(nums+dex)`, the indirection operator informs the compiler about the use of the pointers. Hence, in the first iteration when `dex=0`; the first element of the array `num` will be printed at the output screen. Then in the second iteration, the `dex` value would be increased to 1. Here, the compiler knows that the array has been initialized as integer type and integer occupies two bytes. Therefore, the compiler will jump two memory location addresses to access the second element of the array `num` that is 81. This will be printed at the output. Similarly, in the next iteration the `dex` value would be increased to 2 and 70 will be printed at the output. This will go on until the entire array is printed at the output.

Exercise

1. Make a program that sets values of five variables using pointers.

2. Use the while loop in the Program 2 for accessing the array locations.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 11

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 # 12

Pointers with functions

Objective: At the end of this lab exercise, students would be able to understand and implement the concept of pointers with functions in the C language for designing the programming solutions.

Theory:

A pointer is a methodology in C language to access a variable let's say named 'x' without directly referring to the variable named 'x'. Let us understand this concept with the help of a simple analogy.

Now let us see an example of accessing the values of the variables using their memory address with the help of the pointers in the following section.

Pointers with functions

The use of pointers inside the functions is very important. The main advantage that the pointers provides in functions is that by using the pointer the programmer can yield multiple function output arguments in the C language. This functionality has not been provided in the C functions alone. The following program illustrate this property of pointers with functions.

Program 1

```
// function that returns two values

void addcon(int *, int *); // prototype; * is indirection operator

void main(void)
{
    int x=4, y=7;
    addcon(&x, &y);
    printf("first is %d, second is %d",x,y);
}

void addcon(int *px, int *py)
{
    *px = *px + 10;
    *py = *py + 10;
}
```

Commentary

The Program 1 shows the use of pointers with functions. This program adds the values of 10 to the variables x and y. It must be noted that without really accessing the variable name, their values has been accessed and changed. This is done with the help of address of variable and pointers.

The address of x and y has been provided in the addcon function using the address operator '&'. In the addcon function input *px and *py are declaration of the two pointer variables. We know a variable is a pointer with the help of * indirection operator. The address values are now assigned to the pointer variables. And in the function body of addcon function, the content of the addresses in px and py as been incremented by 10. Hence, after the addition of 10 in x and y address pointers, the values of the x and y has been changed to 14 and 17.

Exercise

1. Make a program that multiplies two with acquired integer values from the user using pointer with functions.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 12

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 # 13

Open Ended Lab Activity

Objective: This open ended lab activity will enable students to apply their programming skills for designing and developing solutions based on the trending and state-of-the-art technologies.

Theory: Artificial neural networks are the interconnected assemblies of simple processing elements called neurons, whose functionality is loosely based on the working principle of biological neurons. Let us first discuss the biological neurons. Biological neuron takes the input signal from branches called dendrites. The processing takes place at soma or cell body, which is the central part of the neuron. Nucleus is found in the cell body. Once, the electrochemical activity is greater than the threshold, the neuron fires and the output is given to the connected neurons through axon. Axon is a long tubular fiber used to provide output to other neurons as shown in the Figure 1.

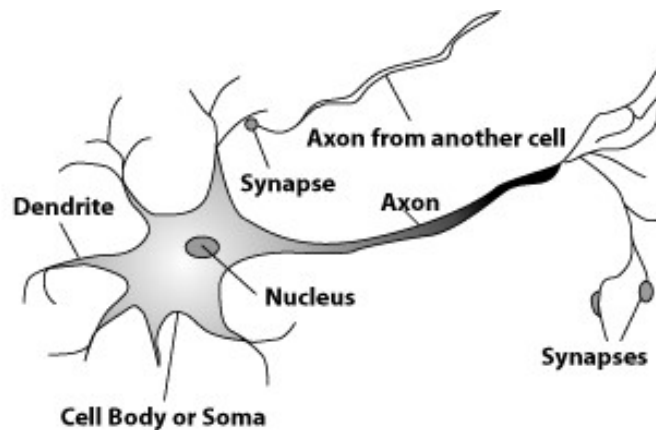


Figure 1: Biological neuron [1]

The neurons transfer the output in the form of electrochemical activity through connections known as synapse. Neurons are not interconnected physically; however, there is a gap between two neurons. The gap is known as synaptic cleft.

The neuron activity is measured in medical lab with the help of electrodes as shown in Figure 2.

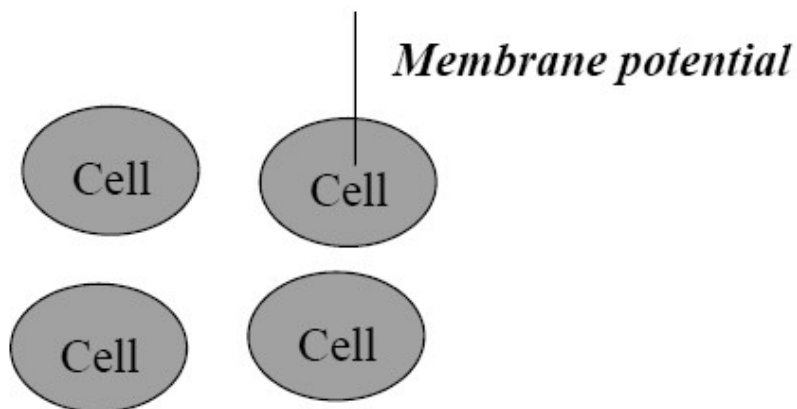


Figure 2: Biological neuron activity measurement [2]

The electrodes measure the membrane potential of a neuron with its surroundings. When the membrane potential is greater than threshold a spike is generated. The spike generation indicates that a neuron has communicated with other neuron as shown in Figure 3.

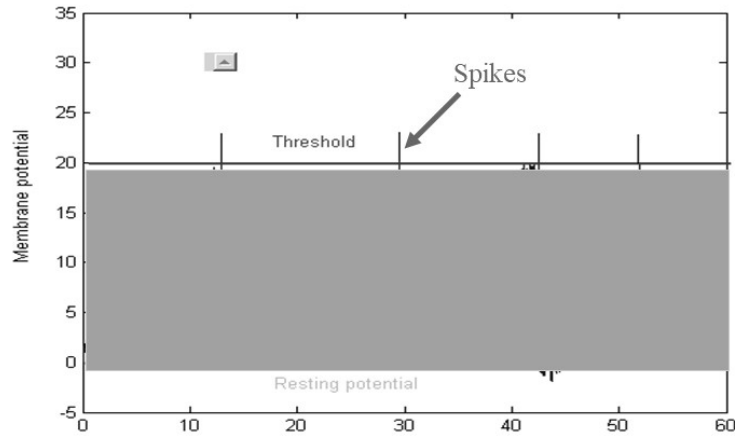


Figure 3: Detection of spikes using membrane potential [2]

The neuron activity below the set threshold is known as sub-threshold activity. We are not interested in the sub-threshold activity as the neuron communicates through spikes. It means that in the biological neuron the information is coded in spikes. If somehow we are able to understand the spiking time and space, we can decipher how the neurons communicate.

By looking and understanding the functioning of biological neuron, scientists have proposed an artificial neuron model that works loosely on the principle of the biological neuron as shown in the following Figure 4.

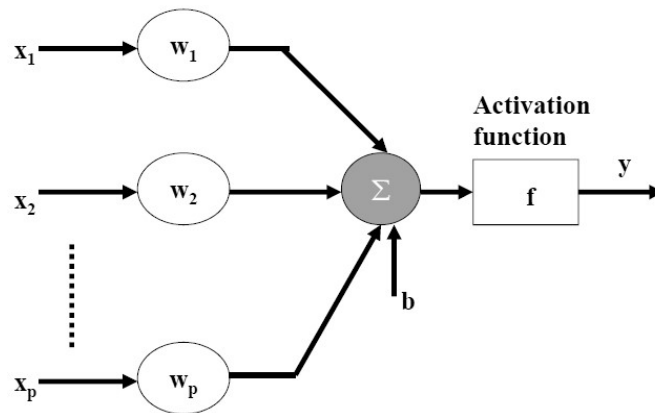


Figure 4: Artificial neuron model [2]

In Figure 4, x_1, x_2, \dots, x_p represents the inputs; w_1, w_2, \dots, w_p shows the associated weights with the inputs; Σ shows the summation of the product of input and weights; b is the added bias into the summation. Activation function or f represents a threshold deciding function. When the activity or the input of the activation function crosses the threshold, the artificial neuron fires and give its output to the connected neuron. A simple artificial neuron can be represented by the following Equation 1.

$$y = f\left(\sum_{i=1}^p w_i x_i + b\right) \quad (1)$$

If we compare the biological and artificial neurons, the dendrites are represented by the inputs in the artificial neuron, synapses are represented by the weights, function of the soma is represented by the summation, increase in biological electrochemical activity inside soma and its firing is represented by the activation function in the artificial neuron, and finally the axon is represented by the output y .

Activation functions: Activation function of an artificial neuron can be a linear or non-linear. A particular activation function is chosen to satisfy some specification of the problem that a neuron is attempting to solve. A variety of activation functions have been proposed for example hard limiter, symmetrical hard limiter, linear, Gaussian, sigmoid, and tangent sigmoid. We are going to address an electronics based problem in this open ended lab. Therefore, let us discuss hard limiter activation function. A hard limiter activation function can be expressed by the following Equation 2.

$$y = \begin{cases} 0, & u < 0 \\ 1, & u \geq 0 \end{cases} \quad (2)$$

This activation function sets the output y equal to zero when the activity u of the neuron is smaller than zero. And it sets the output y equal to one when the activity u of the neuron is greater than or equal to zero. This is also shown in the following Figure 5.

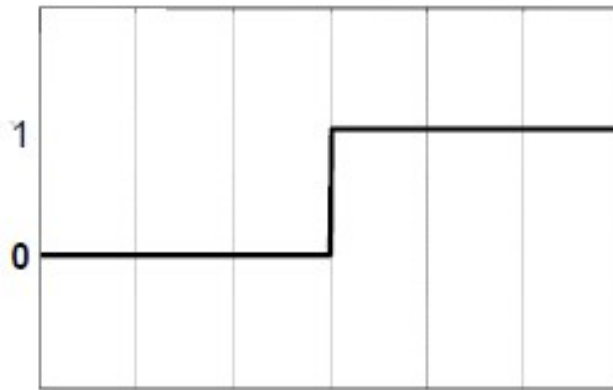


Figure 5: Hard limiter activation function[2]

Neuron working: Now that we know the artificial neuron proposed mathematical model equation and the work of activation function. Let's see this all in action. For example, if we are given with the weights of the neuron as $w = [1.2 \ 3.4]$, input to neuron as $x = [-3 \ 5]$, and bias as $b = 1.4$. What will be the total net activity and output of the neuron when the activation function is Hard limiter?

Net input to the neuron activation function is as follows.

$$y = f\left(\sum_{i=1}^p w_i x_i + b\right)$$

$$y = f(u)$$

$$u = [1.2 \ 3.4] \begin{bmatrix} -3 \\ 5 \end{bmatrix} + 1.4$$

$$u = 14.8$$

$$y = f(u) = 1$$

Hence, the net activity to the neuron is 14.8 and the output generated by the neuron is 1.

Error correction learning: Now that we are familiar with the working of a neuron, let us discuss the method of training weights for some specified purpose. In the aforementioned example, the weights were pre-defined. However, if we need to tune-in or learn the weights for some specific task, we will use a set of equations known as learning algorithm.

Error equation: Error correction learning algorithm provides us the basic understanding of the learning process through correction of errors. This algorithm first finds the error in system by subtracting the neuron output from the desired output as shown in Equation 3.

$$error = desired - neuron\ output \quad (3)$$

Cost function: Once all the system errors are calculated, a cost function is used to find the total system error. In the error correction learning algorithm, mean squared error is used as the cost function as shown in Equation 4.

$$J = \frac{1}{n} \sum e^2 \quad (4)$$

When the value of the cost function approaches infinity, it suggests that there is a lot of error in the system. Therefore, it is a bad system and we don't need this system. And when the value of the cost function approaches zero, it suggests that the total amount of error in the system is very less. Therefore, it may be a good system and we need this system.

Weight adjustment: After the calculation of total system error, we decide through a threshold value that whether to adjust the weights or keep those weights. Threshold is a hyper-parameter and can be set as per the requirement. If the cost function gives large value of error then the weights must be adjusted to get the desired output from the network. The following Equation 5 is used in error correction learning algorithm to adjust the weights.

$$\Delta w = \eta \times e_i \times x_i \quad (5)$$

In the Equation 5, η represents the rate of learning and its value ranges from 0 to 1. e_i represent the error calculated by the Equation 3 in sample i . And x_i represent the input i . Δw gives us the adjustment in weights.

Weight update: Now that we have adjusted the weight as per our requirements we need to update the weights. At this instance we have two weights i.e. old weights before adjustment and the adjusted weights. The error correction learning algorithm states that both can be added to make the updated weights as shown in the Equation 6.

$$w[n + 1] = w[n] + \Delta w \quad (6)$$

Once we have the updated weights, these can be used in the neuron model in Equation 1 to test and validate the output.

Programming structure: Following program structure shows the way to implement the Error correction learning algorithm in any programming language.

```
variable initializations;
```



```

while (condition: is cost function greater than threshold)
{
    for (starting point; condition; step)
    {
        neuron model equation;
        error equation;
    }
    cost function equation;
    if (condition: if total cost is greater than the threshold)
    {
        weight adjustment equation;
        weight update equation;
    }
}

```

Commentary: At the start of the program all the hyper-parameters needed to be initialized. Then while loop is used to check the value of cost function is greater than the threshold. In the first iteration, the value of the cost function is defined in the variable initialization. We may also use the do-while loop here. The using the for loop neuron model equation is calculated along with the error in each instance. Once all the error are calculated, we use those errors to find out the cost function value. Then by using if statement, we check the value of cost function is greater than or small than the defined threshold. If the value of cost function is greater than the defined threshold, we adjust and update the weights using corresponding equations. If not, the compile will exit the loop and the weight values obtained would be considered as final converged weight values.

References

- [1] Accessed from <http://www.neuralpower.com/technology.htm>
- [2] Haykin, S., Neural Networks – A Comprehensive Foundation, Second edition or latest, McMillan.

Exercise

1. Design and train the artificial neuron on the principle of two input AND gate using Error correction learning algorithm.



NED University of Engineering & Technology
Department of Electronic Engineering

Course Code and Title: EL-255 Programming Languages

Laboratory Session No. Lab # 13

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	