LABORATORY WORK BOOK
For The Course
CT-153 Programming Languages

Name : _____

Roll No. : _____

Batch : _____

Year : _____

Dept. : _____

Department of Computer Science and Information Technology
N.E.D. University of Engineering & Technology, Karachi –75270, Pakistan

LABORATORY WORK BOOK

For The Course

CT-153 Programming Languages

Prepared By:

Waseemullah (IT-Manager(Jr.)**)**

Approved By:

Chairman
Department of Computer Science and Information Technology

Introduction

This work book is specially designed to help the students to generate their own logic for the accomplishment of the assigned tasks. Ev ery lab is provided with the syntax of the statements or commands which will be used to make the programs of exercises. In order to facilitate the students some program segments are also provided explaining the use of commands. For a wide scope of usage of the commands several examples are also given so that the students can understand how to use the commands.

The conditions, limitations and memory allocation are also mentioned where necessary. This book starts the programming of C Language from the scratch and covers most of the programming structures of C-Language. For some commands or structures more than one lab are designed so that the students can thoroughly understand their use. This lab book leads the students to the Object Oriented Programming C++. One lab on Object Oriented Programming C++ is given in the last which gives the basic idea of OOP.

Programming with C-Language Laboratory

CONTENTS

Lab No. 01

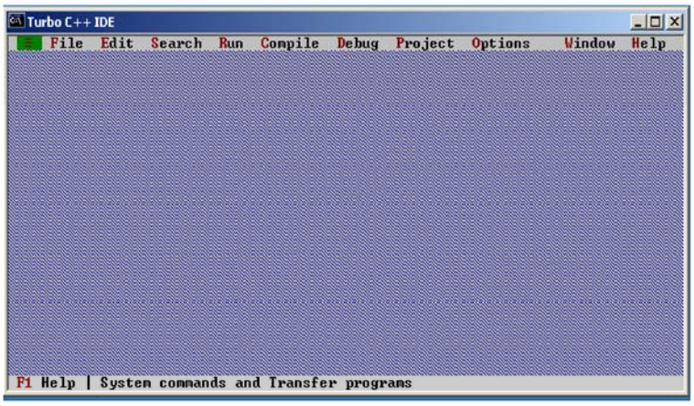
OBJECT

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.



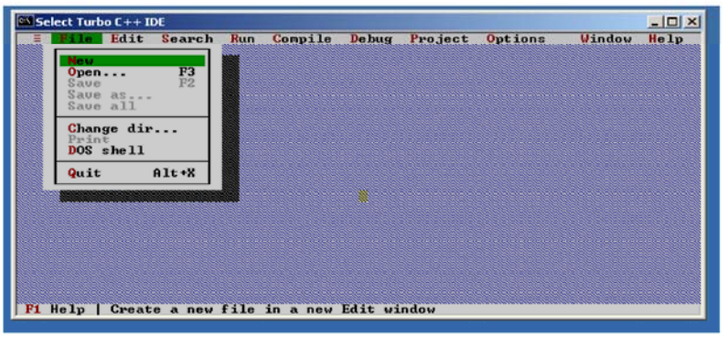
Invoking the IDE
To invoke the IDE from the windows you need to double click the TC icon in the directory c:\tc\bin.

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.



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.

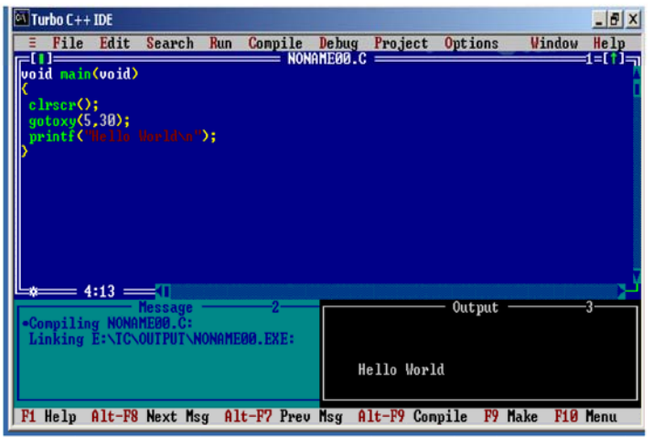
Creating an Executable File with the Linker
After your source code is compiled, an object file is produced. This file is often named with the extension .OBJ. This is still not an executable program, however. To turn this into an executable program, you must run your linker. C programs are typically created 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.



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 ';'.

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

Lab No. 02

OBJECT

C Building Blocks

THEORY

In any language there are certain building blocks:

- Constants
- Variables
- Operators
- Methods to get input from user(scanf( ), getch( ) etc.)
- Methods to display output (Format Specifier, Escape Sequences etc.) etc.

Format Specifiers
Format Specifiers tell the printf statement where to put the text and how to display the text. The various format specifiers are:

%d => integer
%c => character
%f => float etc.

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 ,double etc they are also of the type signed or unsigned.

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 called the **Escape Character"** . The escape sequence includes the following:

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

Taking Input From the User
The input from the user can be taken by the following techniques: scanf( ), getch( ), getche( ), getchar( ) etc.

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 which shows the function of each escape sequence character.
   eg 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");etc

2. Write down C statements to perform the following operations:
   i. $z = \frac{4.2(x+y)5/z - 0.52x/(y+z)}{(x+y)^2}$

   ii. $x = a^2 + 2ab + b^2$

3. what will be the out put of the mix mode use of integers and float.
   a=5/9;

```
b=5.0/9;
printf("%f,%f",a,b);
```

4 what will be the output if a=5,

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

5.write some simple state ments to check the working of logical and relational operators.

Lab No. 03

OBJECT

Looping constructs in C-Language

THEORY

Types of Loops
There are three 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

Nesting may extend these loops.

The for Loop

```
for(initialize(optional);condition(compulsory);increment(optio
nal)
{
        Body of the Loop;
}
```

This loop runs as long as the condition in the center is true. Note that there is no semicolon after the "for" statement. If there is only one statement in the "for" loop then the braces may be removed. If we put a semicolon after the for loop instruction then that loop will not work for any statements.

The while Loop

```
while(condition is true)
{
      Body of the Loop;
}
```

This loop runs as long as the condition in the parenthesis is true. Note that there is no semicolon after the "while" statement. If there is only one statement in the "while" loop then the braces may be re moved.

The do-while Loop

```
do
{
      Body of the Loop;
}
while(condition is true);
```

This loop runs as long as the condition in the parenthesis is true. Note that there is a semicolon after the "while" statement. The difference between the "while" and the "do-while" statements is that in the "while" loop the test condition is evaluated before the loop is executed, while in the "do" loop the test condition is evaluated after the loop is executed. This implies that statements in a "do" loop are executed at least once. However, the statements in the "while" loop are not executed if the condition is not satisfied..

EXERCISE

1. Write down the output of the following program statements

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

   ii.
   ```
   int a = 10, b = 10;
   for(inti=1;i<=a;i++)
   {
       a++;
       b--;
       printf("a = %d,b=%d\t",a,b);
   }
   ```

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

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

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

Lab No.04

OBJECT

Nested looping

THEORY

Types of Loops
There are three 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

Nesting may extend these loops.

The Nested for Loop

```
for(initialize;condition;increment)
{
        for(initialize;condition;increment)
        {
        Body of the loop;
        }
}
```

The inner loop runs as many times as there is the limit of the condition of the external loop. This loop runs as long as the condition in the parenthesis is true. We can nest many loops inside as there is the requirement.

The nested while Loop

```
while(condition is true)
{
       while(condition is true)
        {
        Body of the loop;
        }
Body of the loop;
}
```

The inner loop runs as many times as there is the limit of the condition of the external loop. This loop runs as long as the condition in the parenthesis is true. We can nest many loops inside as there is the requirement.

The Nested do-while Loop

```
do
{
        do
          {
          body of the loop;
          }
          while(condition is true);

body of the loop;
}
while(condition is true);
```

This loop runs as long as the condition in the parenthesis is true. Note that there is a semicolon after the "while" statement. The difference between the "while" and the "do-while" statements is that in the "while" loop the test condition is evaluated before the loop is executed, while in the "do" loop the test condition is evaluated after the loop is executed. This implies that statements in a "do" loop are executed at least once. The inner loop runs as many times as there is the limit of the condition of the external loop. This loop runs as long as the condition in the parenthesis is true. We can nest many loops inside as there is the requirement.

EXERCISE

1. Write down the output of the following program statements
   i.
```
for (int a=1;j=1; j<=5;j++)
        for (i=1; i<=5;i++)
          {
                printf("%d\n",a);
                a++;

          }
```

2. Write a program to print a series of first 50 prime numbers.

3. Write a program which Prints the following pattern up to 10 lines

```
0
111
22222
3333333
```

4. Write a program to print the following pattern up to Z only

```
A B  C  D
E F  G H
I J K L
```

Lab No.05

OBJECT

Decision making the if*and* if-else *structure*

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. The 'if' statement, the if-else statement, and the switch statement. Another less commonly used structure is the conditional operator.

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 or the conditions with relational and logical operators are also included..
The simplest form of an if statement is:

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

The if-else statement

Often your program will want to take one branch if your condition is true, another if it is false. If only one statement is to be followed by the if or else condition then there is no need of parenthesis. The keyword else can be used to perform this functionality:

```
if (expression)
        { statement/s;
        }

else
        {
        statement/s;
        }
```

EXERCISE

1.Write a program which takes three sides a, b and c of a triangle input and calculates its area if these conditions are satisfied a+b>c, b+c>a, a+c>b
(Help a= vs(s-a)(s-b)(s-c), where s=(a+b+c)/2

2. Write a program that inputs an integer – determine if it is even or odd.

3. Write a program which takes a character input and checks whether it is vowel or consonant

Lab No.06

OBJECT

Decision making the Switch case and conditional operator.

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. The 'if' statement, the if-else statement, and the switch statement. Another less commonly used structure is the conditional operator.

The switch Statement

Unlike if , which evaluates one value, 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:

```
switch (identifier variable)
{
case identifier One:        statement;
                               break;
case identifier Two:        statement;
                               break;
....
case identifier N:          statement;
                               break;
default: statement;
}
```

Conditional 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)

It replaces the following statements of if else structure

```
If(a>b)
c=a;
else
c=b;
```

can be replaced by

```
c=(a>b)?:a:b
```

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.

EXERCISE

1.Write a program which takes a text input counts total number of vowels, consonants and other special characters and prints the result

2. Write a program to make a simple calculator which should be able to do +,-,*,/,% operations

3. Write a program which takes 10 integers as input and prints the largest one.

Lab No. 07

OBJECT

Debugging and Single-Stepping of C Programs

THEORY

One of the most innovative and useful features of Turbo C++ is the integration of debugging facilities into the IDE.

Even if your program compiles perfectly, it still may not work. Such errors that cause the program to give incorrect results are called Logical Errors. The first thing that should be done is to review the listing carefully. Often, the mistake will be obvious. But, if it is not, you'll need the assistance of the Turbo C Debugger.

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

Consider the following program:

```
void main(void)
{
    int number, answer=-1;
    number = -50;
    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 where the problem is 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 (num < 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 **if** statement is false, which leads to erroneous results. We need to put braces around the second **if**, or rewrite the program in some other way.

Resetting the Debugger
Suppose you've single stepped part way through a program, and want to start over at the beginning. How do you place the run bar at the top of the listing? You can reset the debugging process and initialize the run bar by selecting the Program Reset option from the Run menu.

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 you've debugged part of your program, but must deal with a bug in another section, and you don't want to single-step through all the statements in the first part to get to the section with the bug. Or you 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 you start the program with [Ctrl][F9], it will execute all the statements up to the breakpoint, then stop. You can 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. You can install as many breakpoints as you 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
You can 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 you did to install the breakpoint). The breakpoint highlight will vanish.

You can all set **Conditional Breakpoints** that would break at the specified value only.

EXERCISE

1. Add watches to the entire variable in a program and follow their values line by line.

2. Type in the following program and find out the error using the Turbo C Debugger.

```
#include<stdio.h>
void main(void)
{
    int a=4,b=5,i;
    for(i=1;i<=10;i++);
        {
        a++;
        b--;
        }
        printf("%d",i);
        printf("%d",a);
        printf("%d",b);
}
```

Mention the error. Correct this program by locating the error through the debugger and rewrite the correct program statements

Lab No. 08

OBJECT

Functions in C-Language programming

THEORY

Functions are used normally in those program where some specific work is required to be done repeatedly and looping fails to do the same.
Three things are necessary while using a function.

i. Declaring a function or prototype:
The general structure of a function declaration is as follows:

```
return_type function_name(arguments);
```

Before defining a function, it is required to declare the function i.e. to specify the function prototype. A function declaration is followed by a semicolon ' ;'. Unlike the function definition only data type are to be me ntioned for arguments in the function declaration.

ii. Calling a function:
The function call is made as follows:

```
return_type = function_name(arguments);
```

ii. Defining a function:
All the statements or the operations to be performed by a function are given in the function definition which is normally given at the end of the program outside the main.
Function is defined as follows

```
return_type function_name(arguments)
{
Statements;
}
```

There are certain functions that you have already used e.g:getche( ), clrscr( ), printf( ), scanf( ) etc.

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.

A function that returns nothing must have the return type "void". If nothing is specified then the return type is considered as "int".

EXERCISE

1. Write a program to print the find the sum of the given series, take first 8 terms
A=1! +2! +3! +4! +…….

2. Write a program to find
a. Surface area ($A=4 \pi r^2$)
b. volume($v=4/3 \pi^3$)
of a sphere using functions make a function for finding powers of radius.

3. Write a program using functions to evaluate up to 8 terms

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} - \ldots\ldots$$

Lab No.09

OBJECT

Preprocessor Directives

THEORY

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

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

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

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

for 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.

1. wrte 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.

Lab No. 10

OBJECT

Arrays in C (one dimensional)

THEORY

An array is a collection of data storage locations, each of which holds the same type of data. Ea ch storage location is called an element of the array. You declare an array by writing the type, followed by the array name and the subscript. The subscript is the number of elements in the array, surrounded by square brackets. For example,

```
long LongArray[25];
```

declares an array of 25 long integers, named Long Array. When the compiler sees this declaration, it sets aside enough memory to hold all 25 elements. Because each long integer requires 4 bytes, this declaration sets aside 100 contiguous bytes of memory

EXERCISE

1. Write a program that takes input for array int a[5] and array int b[5] and exchanges their values.

2. Write a program that takes 10 integers as input and prints the largest integer and its location in the array.

3. Write a program which takes a string as input and counts total number of vowels in that.

4. Write a program to sort an integer array in descending order.

Lab No.11

OBJECT

Arrays in C (Multidimensional)

THEORY

A Multidimensional array is a collection of data storage locations, each of which holds the same type of data. Each storage location is called an element of the array. You declare an array by writing the type, followed by the array name and the subscript. The subscript is the number of elements in the array, surrounded by square brackets. For example,

```
int a[5][10]
```

declares an array of 50 integers, na med a. Its declaration shows that array a comprises of 5 one dimensional arrays and each one dimensional array contains 10 elemnts.
When the compiler sees this declaration, it sets aside enough memory to hold all 50 elements. Because each integer requires 2 bytes, this declaration sets aside 100 contiguous bytes of memory.

EXERCISE

2. Write a program that adds up two 4x4 arrays and stores the sum in third array.

2. Write a program which takes names of five countries as input and prints them in alphabetical order.

Lab No. 12

OBJECT

Learning Text and Graphics modes of display in C

THEORY

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

- The Text Mode
- The Graphics Mode.

The Text Mode
In the Text Mode, the entire screen is viewed as a grid of cells, usually 25 rows by 80 columns. Each cell can hold a character with certain foreground and background colors (if the monitor is capable of displaying colors). In text modes, a location on the screen is expressed in terms of rows and columns with the upper left corner corresponding to (1,1), the column numbers increasing from left to right and the row numbers increasing vertically downwards.

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

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

To automate the loading of ANSI.sys follow these steps:

i. Find the file ANSI.sys in your system. Note the path.

ii. Find the config.sys file. Open this file and type the following:

```
DEVICE = path_of_ANSI.sys
```

iii. Restart your computer.

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

EXERCISE

1. Write a program to pop up a window on the screen

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

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

4. Display the use of gettext and puttext functions.

Lab No. 13

OBJECT

Structures

THEORY

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

```
Struct books
{char bookname[20];
float price;
int pages;}
struct book[50];
```

the above structure can hold information of 50 books.

EXERCISE

1. Write a program to maintain the library record for 100 books with book name, author's name, and edition, year of publishing and price of the book.

2. Write a program to make a tabulation sheet for a class of 50 students with their names, seat nos, marks, percentages and grades.

3. Define a structure to represent a complex number in rectangular format.

3. Define a structure to represent a complex number in rectangular format.

Lab No. 14

OBJECT

Pointers in C-Language

THEORY

A pointer provides a way of accessing a variable without referring to the variable directly. The address of the variable is used.
The declaration of the pointer p,

```
int *p;
```

means that the expression *p is an int . This definition set aside two bytes in which to store the address of an integer variable and gives this storage space the name p. If instead of int we declare

```
char * p;
```

again, in this case 2 bytes will be occupied, but this time the address stored will b e pointing to a single byte.

EXERCISE

1. Write down the number of bytes allocated for the following pointer variables:

```
int *x;
```

_____

```
char *y;
```

_____

```
float *z;
```

_____

2. Determine the output of the following program:

```
void main(void)
{
        int q=2;
        int *p;
        p=&q;
        *p=100;
        printf("%d",q);
        printf("%p",p);
        printf("%d",*p);
        printf("%d",*q);
        printf("%p",&q);
        printf("%p",&p);
```

```
}
```

3. Determine the output of the following program:

```
void main(void)
{
        int x=3,y=4,z=6;
            int *p1,*p2,*p3;
            p1=&x;
            p2=&y;
            p3=&z;
            *p1=*p2+*p3;
            *p1++;
            *p2--;
            *p1=(*p2)*(*p3);
            *p2=(*p2)*(*p1);
            x=y+z;
            printf("%d",x);
            printf("%d",y);
            printf("%d",z);
            printf("%d",*p1);
            printf("%d",*p2);
            printf("%d",*p3);

}
```

4. Write a program which adds two arrays with the help of their pointers.

Lab No. 15

OBJECT

Pointers with arrays and function.

THEORY

A pointer is the most effective tool to pass an array to a function.
If pointers are involved than a function can return more than one values at a time.
We have to pass only the address and size of the array to the function and we can make as many changes in the function as we want. for example if we want to add 5 in each array element using functions. Then

```
void add(int *,int);
        void main(void)
        {int s[10],i;
        printf("enter ten integers");
        for(i=0,i<10,i++)
        {printf("\n enter integer no %d :",i+1);
          scanf("%d",&s[i]);
        }
        add(s,10);
        for(i=0,i<10,i++)
        {printf("\n integer no %d :%d",i+1,s[i]);

        }
        }
void add(int *p,int x)
{int j;
for (j=0;j<x;j++)
{
*p=*p+5;
p++;
}
}
```

Similarly string arrays and multidimensional arrays can also be passed to functions by their addresses and size.

EXERCISE

1. Write a program to pass an integer array of 10 elements to a function which returns the same array after sorting it in descending order. Print the array.

2. Write a program which passes a string to a function and the function changes its case without using any library function.

Lab No 16

OBJECT

Filing in C-Language

THEORY

Data files

Many applications require that infor mation be " **written & read"** fro m an auxiliary storage device.
This information is written in the form of **Data Files** .
Data files allow us to store information permanently and to access and alter that information whenever necessary.

Types of Data files
Standard data files .(stream I/O)
System data files. (low level I/O)

Standard I/O

Easy to work with, & have different ways to handle data.

Four ways of reading & writing data:

1. Character I/O.
2. String I/O.
3. Formatted I/O.
4. Record I/O.

File Protocol

1. fopen
2. fclose

fopen:-

It is used to open a file .

Syntax:

fopen (file name , access-mode).

"r" open a file for reading only.
"w" open a file for writing.
"a" open a file for appending .
"r+" open an existing file for reading & writing.
"w+" open a new file for reading &writing.
"a+" open a file for reading & appending & create a new file if it does exist.

Example:-

```
#include <stdio.h>
main
{
FILE*fpt;
```

```
fpt = fopen ("first.txt","w");
fclose(fpt);
}
```

Establish buffer area, where the information is temporarily stored before being transferred b/w the computer memory & the data file.

file is a special structure type that establishes the buffer area.
fpt is a pointer variable that indicates the beginning of buffer area.
fpt is called stream pointer.
fopen stands for File Open.
A data file must be opened before it can be created or processed.

Standard I/O:
Four way s of reading and writing data:
Character I/O.
String I/O.
Formatted I/O.
Record I/O.

Character I/O:
In normal C program we used to use getch, getchar and getche etc.
In filling we use putc and getc.

putc( );
It is used to write each character to the data file.
Putc requires specification of the stream pointer *fpt as an argument.

Syntax:
putc(c, fp);
c = character to be written in the file .
fp = file pointer .
putch or putchar writes the I/P character to the consol while putc writes to the file.

Example (1):

Reading the Data File:

```
#include < stdio.h >
main( )
{
    FILE*fpt;
    Char c;
    fpt = fopen ( " star.dat","r");
    if( fpt = = NULL);
    printf( "Error- cant open");
    else
    do
            putchar(c=getc(fpt) );
  while(c!='\n' );
```

```
    fclose( fpt );
}
```

Exercise:

1. What will be the output of the given program

```
#include < stdio.h >
main( )
{
      FILE*fpt;
      Char c;
      fpt = fopen ( " star.dat","n"); [ a new file is made
]
      do
                  putc((c=getchar( ) ); fpt );
   whilc(c!='\n' ); or '\r'
   fclose( fpt );
}
```

2. Write a program to store strings in a file

3. Write a program segment that writes an array to a file.

Lab No.17

OBJECT

Introduction to Object Oriented Programming C++.

THEORY

C++ i s a superset of C, that is, it incorporates all the operators and syntax of ordinary C, but adds additional features.

Object-Oriented Programming

Its important understands that object-oriented programming is an approach to organizing programs, it is not primarily concerned with the detail of program code; its focus is the overall structure of the program. A prerequisite for writing object oriented programs understands the philosophy of OOP. OOP has the greatest potential for simplify ing program conceptualization, coding, debugging, and maintenance.

Object-Oriented Organization

Object-oriented programming attempts to solve some of the procedural approach. OOP gives data a more central position in program organization, and ties the data more closely to the functions that act on it. In the procedural approach, the organizational units are functions. In OOP, the organizational units contain both the data and the functions that act on that data. These units are called objects.

Objects

In QOP, a program is organized into objects. As we noted, an object consists of data and the functions that operate on that data. In a typical situation only the functions in the object can access the object's data; it cannot be accessed by functions in other objects.

A New Vocabulary

The functions in an object are called methods, they do something to the objects data, and the binding together of the data with the functions that operate on it is called encapsulation. The fact that the data in an object cannot be accessed by other objects is called data hiding Encapsulation and data hiding are key elements in OOP.

Classes

In OOP, objects are memb ers of classes. A class in OOP has the same relationship to an object that the declaration of struct.

Inheritance

An important feature of C++ is inheritance. Inheritance means making objects out of other objects.

Classes and Objects

Our first example demonstrates a class and an object. It models a simp le expense ledger. Imagine you're running a lemonade stand, and every time you buy supplies you write down the amount: $10.50 for lemons, $4.12 for paper cups, and so on. Our program lets you record such expenses on your computer

Data

The central data structure in this program is an array of floating point numbers to hold the expenses.

Declaring a Class

We mentioned that declaring a class is so mewhat like declaring a structure. It specifies what objects of this class will look like. Here's the declaration of class ledger:

```
        Class ledge              // declaration of class ledger
{
        private:
                float list[100J;              // define array for items
                int count;                    //define numb er of items entered
                public:
                void initCountvoid;      // declare method
                void enterItem(voia);    // declare method
};
```

Defining Methods

Here are the definitions for the methods initCount( ), enterItem( ), and printTotal( ).

```
////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ledger:: initCount(void)                                  // a function in class ledger
{                                                                                  // initializes count to 0
count = 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////

vold ledger::enterItem(void)                                   // a function in class ledger
{                                                                                  // puts entry in list
cout << "\nEnter amount: ";
cin >>list[count++];
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////
void ledger::printTotal(void)                                  // a function in class ledger
{                                                                                  // prints total of all entries
float total=0.0;
```

```
for(int j=0; j<count; j++)
total += list[j];
cout <<"\n\nTotal is: " << total << "\n";
////////////////////////////////////////////////////////////////////////////////////////////////////
```

The expense.cpp Example

Lets put the C++ co mponents we discussed above into a complete C++ program. This example program permits the user to record amou nts in the list and print out the total.

```
////////////////////////////////////////////////////////////////////////////////////////////////////
// expense.cpp
// object oriented expense ledger
#include <iostream.h>.
// for cout, cin, etc.
////////////////////////////////////////////////////////////////////////////////////////////////////
class ledger                                                    //  declaration of, class ledger
{
private:
        float list[100];                                        // array for items
        int count;                                              // number of items entered
public:
        void initCount(void);                                   // declare function
        void enterItem( void);                                  // declare function
        void printTotal(void);                                  // declare function
        };
////////////////////////////////////////////////////////////////////////////////////////////////////
void ledger::initCount(void)                                    // a function in class ledger
{                                                               // initializes count to 0
        count = 0;
}
////////////////////////////////////////////////////////////////////////////////////////////////////
void ledger: :interItem(void)                                   // a function in c lass, ledger, '
{                                                               // puts entry in list
        cout <<"\nEnter amount: ";
        cin >> list[count++];
}
////////////////////////////////////////////////////////////////////////////////////////////////////
void ledger::printTotal(void)                                   // a function in class ledger
{                                                               // prints total of all entries
        float total=0.0;
        for( int j = 0; j<count; j++)
                total += list[j];
        tout <<"\n\nTotal is: " <<total <<"\n";
}
////////////////////////////////////////////////////////////////////////////////////////////////////
ledger expenses;                                                // an instance of class ledger
```

```
////////////////////////////////////////////////////////////////////////////////////////////////////
main( )                                                          // main --handles user interface
{                                                                   // uses object expenses
        char option;
        expenses.initCount( );                          // initialize count
        do
                {                                                                       // display options
                cout <<"\nOptions are: e --enter expense"
<<"\n                   E --print total expenses"
        <<"\n                   q --quit (terminate program)"
        <<"\nType option letter: ";
        cin >>option;                                   // get option from user
        switch (option)
                        {                                                       // act on Option selected
        case 'e': expenses.enterItem( ); break;
        case 'E': expenses.printTotal( ); break;
        case 'q': break;
        default: cout <<"\nUnknown co mmand\n";
        }
        } while(option != 'q');
        } // end main

        Options are:                            e --enter expense.
        E --print total expenses
        q --quit (terminate program)
Type option letter: e
        Enter amount: 10.97
        Options are:
        e --enter expense.
        E --print total expenses
        q --quit (terminate program
        Type option letter: E
        Total is: 99.44
```

EXERCISE:

1. Write a program to generate a simple calculator which can perform basic arithmetic operations.

2. Write a program to generate a tabulation sheet of students which includes their roll no, names, marks and their grades.