Practical Workbook

CS-109 Computer Programming (BM)

Name	: ______________________________
Year	: ______________________________
Batch	: ______________________________
Roll No	: ______________________________
Department:	______________________________
Teacher	: _________________________

Department of Computer & Information Systems Engineering

NED University of Engineering & Technology

INTRODUCTION

The Practical Workbook for "Computer Programming" introduces the basic as well as advance concepts of programming using C++ language. Each lab session begins with a brief theory of the topic. Many details have not been incorporated as the same is to be covered in Theory classes. The Exercise section follows this section.

The Workbook has been arranged as sixteen labs starting with a practical on the Introduction to programming environment and fundamentals of programming language. Next lab session deals with Familiarization with building blocks of C++ and conversion of mathematical expressions into an equivalent C++ statements. Next Lab discussed how to take multiple inputs from console & controlling output's position over it. Single stepping; an efficient debugging and error detection technique is discussed in Lab session 4. Next two lab sessions cover decision making in programming and its application. Lab session 7 and 8 introduce the concepts of loops with different examples to use them in programming.

Function declaration and definition concepts and examples are discussed in lab session 9 and 10. The next three experiments deal with the advance concepts like arrays, multidimensional and character arrays along with different exercises. Lab session 14 is about pointers and dynamic memory allocation. A pointer provides a way of accessing a variable without referring to the variable directly.

Next lab session discussed how to create a list of records using *structures* & its manipulation. These features enable the users to handle not only large amount of data, but also data of different types (integers, characters etc.) and to do so efficiently.

CONTENTS

Lab Session 01

OBJECT

Fundamentals of Computer Programming and
Familiarization with Programming Environment using Microsoft Visual C++ 2010

THEORY

Computer programming is the act of writing computer programs, which are a sequence of instructions written using a computer programming language to perform a specified task by the computer. There exist a large number of high level languages. Some of these include BASIC, C, C++, FORTRAN, Java, Pascal, Perl, PHP, Python, Ruby, and Visual Basic etc.

Introduction to C++

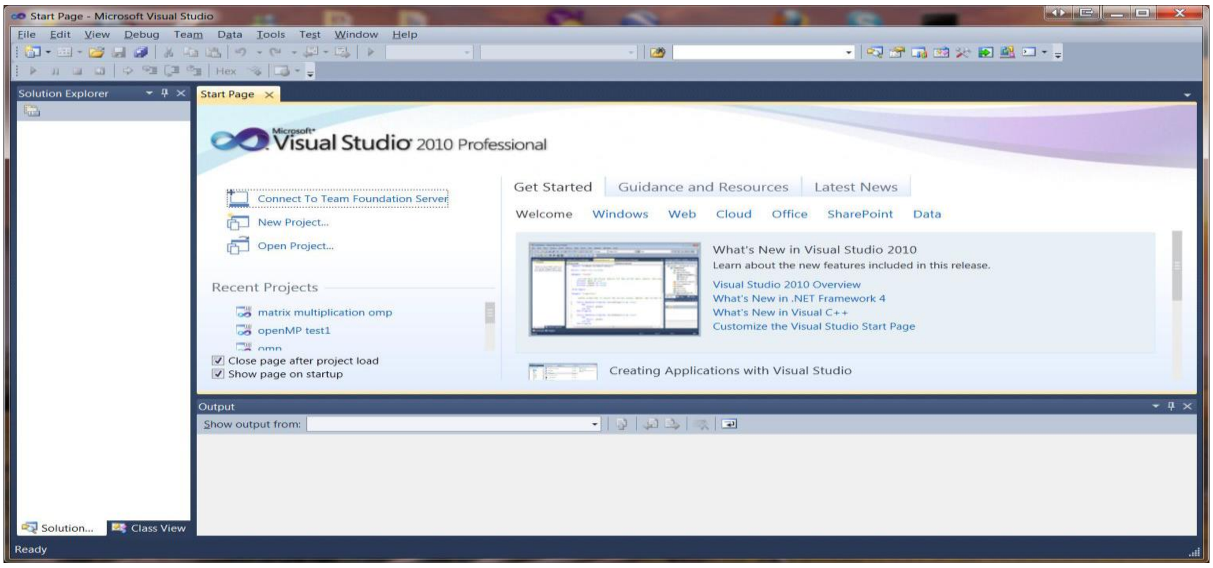
C++ is a general-purpose programming language providing a direct and efficient model of hardware combined with facilities for defining lightweight abstractions. It is a language for developing and using elegant and efficient abstractions. C++ is an object oriented programming (OOP) language, developed by Bjarne Stroustrup, and is an extension of C language. It is therefore possible to code C++ in a "C style" or "object-oriented style." In certain scenarios, it can be coded in either way and is thus an effective example of a hybrid language. Also, it is considered to be an intermediate level language, as it encapsulates both high and low level language features.

Program Development with Microsoft Visual C++ 2010

This lab session introduces the **Integrated Development Environment** (**IDE**) of Microsoft Visual C++ 2010 and shows how to enter, edit, save, retrieve, compile, link, and run a C++ program in such an environment.

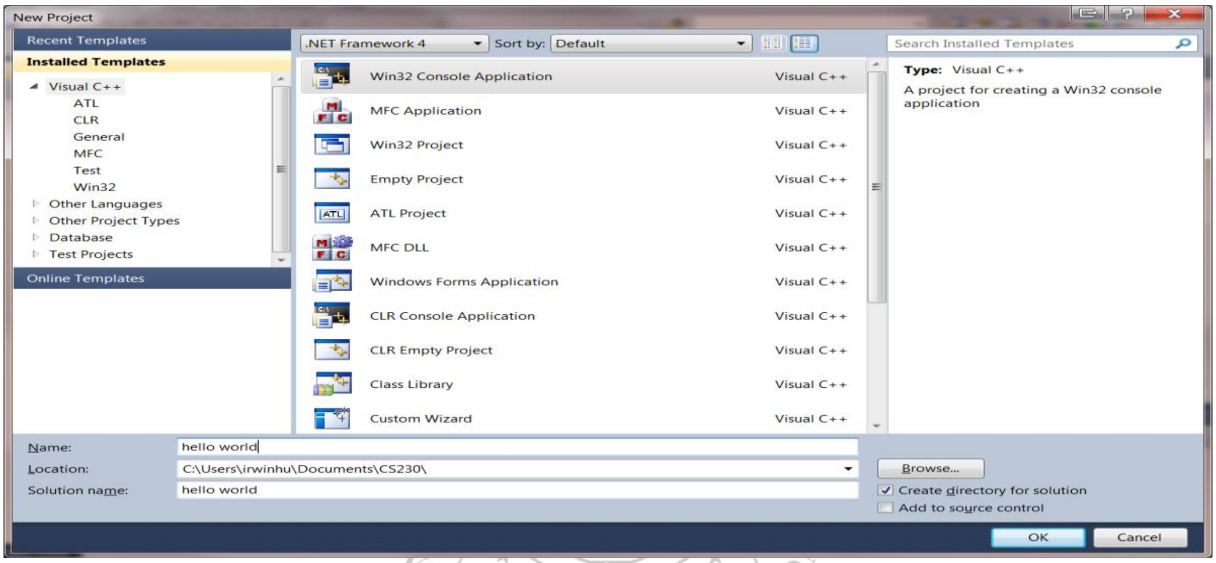
Single-file Project: The *Hello World* Program

Step 1: Launch the MS Visual C/C++ 2010 software from task bar. The main window of Visual Studio 2010 should be similar to the below display:


Start Page - Microsoft Visual Studio
File Edit View Debug Team Data Tools Test Window Help
Solution Explorer
Start Page
Visual Studio 2010 Professional
Connect To Team Foundation Server
New Project...
Open Project...
Recent Projects
matrix multiplication omp
openMP test1
Close page after project load
Show page on startup
Get Started
Guidance and Resources
Latest News
Welcome Windows Web Cloud Office SharePoint Data
What's New in Visual Studio 2010
Learn about the new features included in this release.
Visual Studio 2010 Overview
What's New in .NET Framework 4
What's New in Visual C++
Customize the Visual Studio Start Page
Creating Applications with Visual Studio
Output
Show output from:
Solution...
Class View
Ready

- Hereafter, all system defined terms including menu items such as File will appear in bold and all entries made by programmers such as a filename are italicized.
- If the Solution Explorer window on the left is not shown, click View in the menu bar and then click Solution Explorer to display it.

Step 2: In the menu bar, click **File →New → Projects…**to display the **New Project** dialog box shown below**.** In the **New** dialog box shown below, select by clicking **Visual C++** in the **Installed Template** pane and **Win32 Console Application** in the middle pane. Then enter a name for the project (e.g., *hello world* as shown) in the Name box, select the location or folder to store project files (e.g., C:\Users\irwinhu\Documents\CS230\ as shown) by clicking the **Browse…** Note that there is no need to enter a name in the Solution name box; the system fills the project name in it by default.

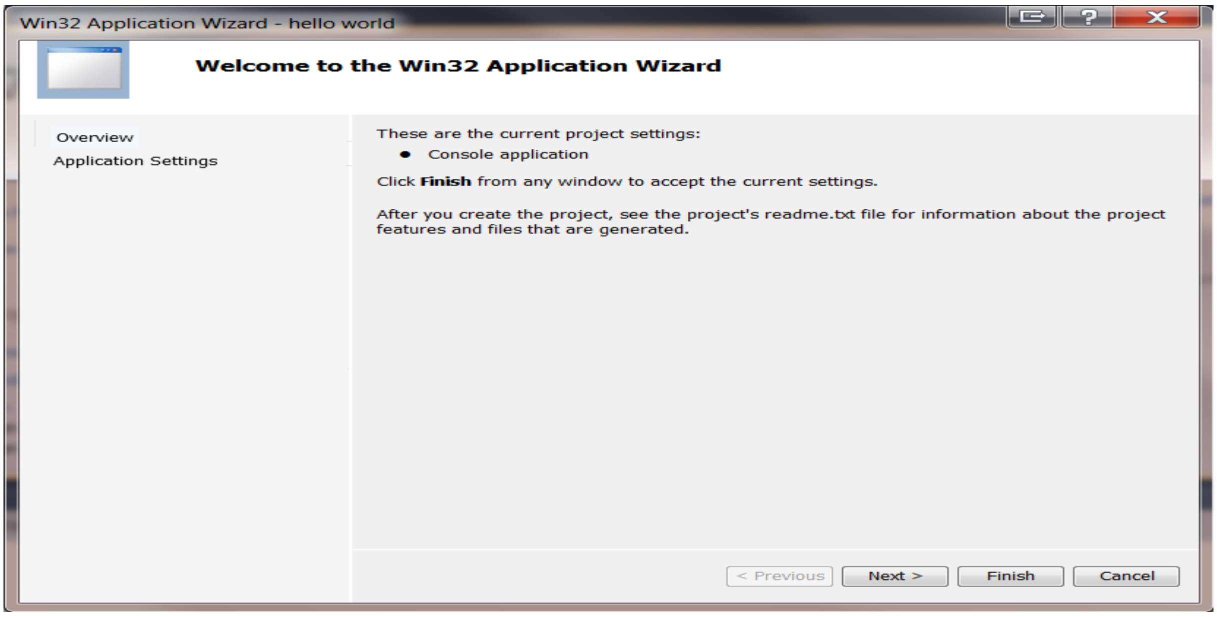


Win32 Console Application	Visual C++
MFC Application	Visual C++
Win32 Project	Visual C++
Empty Project	Visual C++
ATL Project	Visual C++
MFC DLL	Visual C++
Windows Forms Application	Visual C++
CLR Console Application	Visual C++
CLR Empty Project	Visual C++
Class Library	Visual C++
Custom Wizard	Visual C++

Type: Visual C++
A project for creating a Win32 console application

Name: hello world
Location: C:\Users\irwinhu\Documents\CS230\
Solution name: hello world
[✓] Create directory for solution
[ ] Add to source control

Click on the **OK** button to display the **Win32 Application Wizard –** ***hello world*** window shown below:



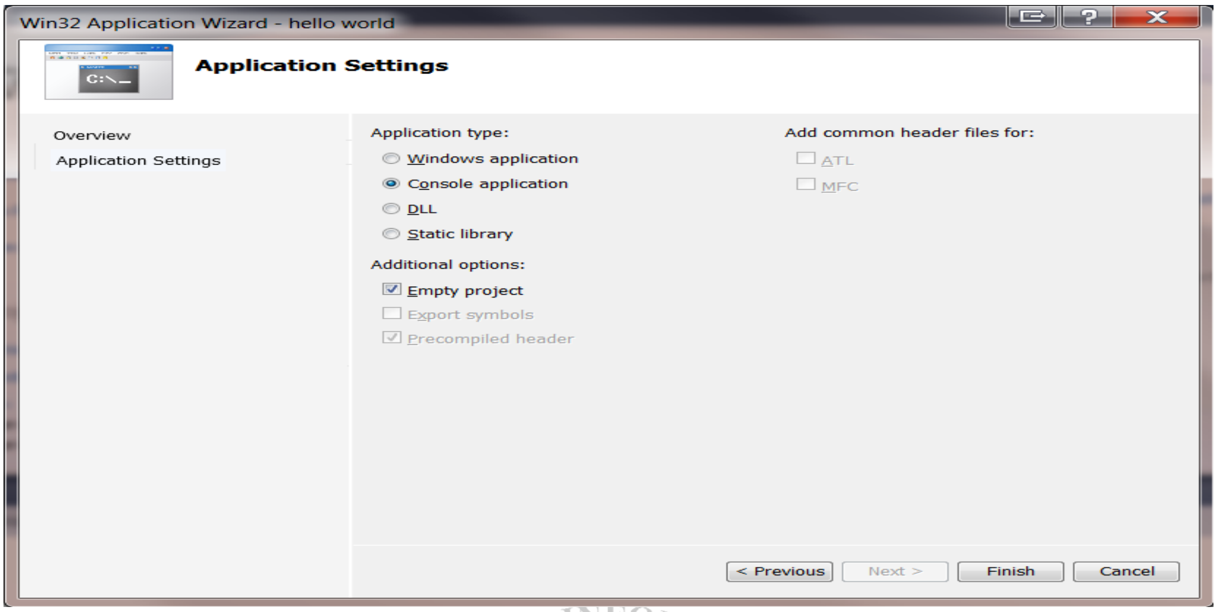
Welcome to the Win32 Application Wizard

These are the current project settings:
- Console application

Click **Finish** from any window to accept the current settings.

After you create the project, see the project's readme.txt file for information about the project features and files that are generated.

Click the **Next** button to display the following dialog box:



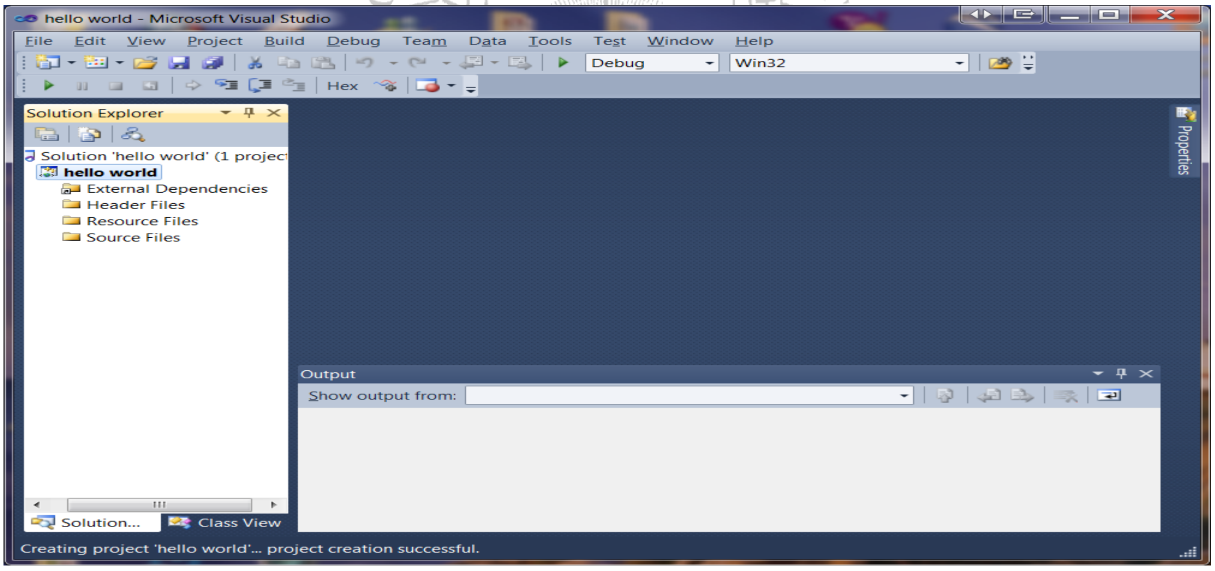
Application type:
- Windows application
- [✓ Console application]
- DLL
- Static library

Add common header files for:
- ATL
- MFC

Additional options:
- [✓ Empty project]
- Export symbols
- [✓ Precompiled header]

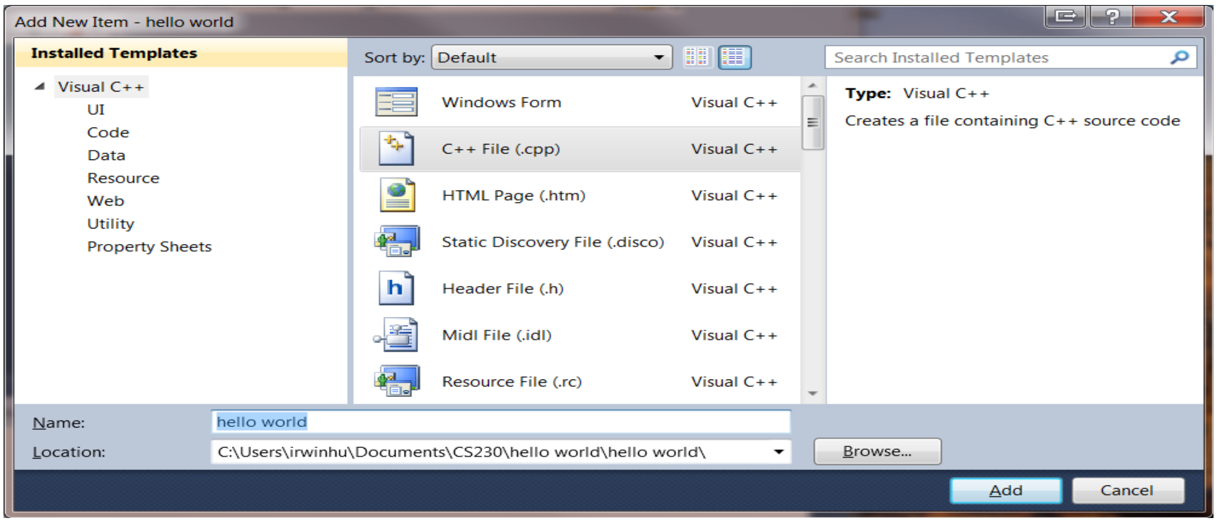
Check the **Empty project** box as shown above and click on the **Finish** button to proceed to the next step.

Step 3: Now the system displays the following window.



Creating project 'hello world'... project creation successful.

Right click on the **Source Files** folder in the **Solution Explorer** pane. In the popup menu, click **Add** then **New Item**… to display the following **Add New Item –** ***hello world*** dialog box:

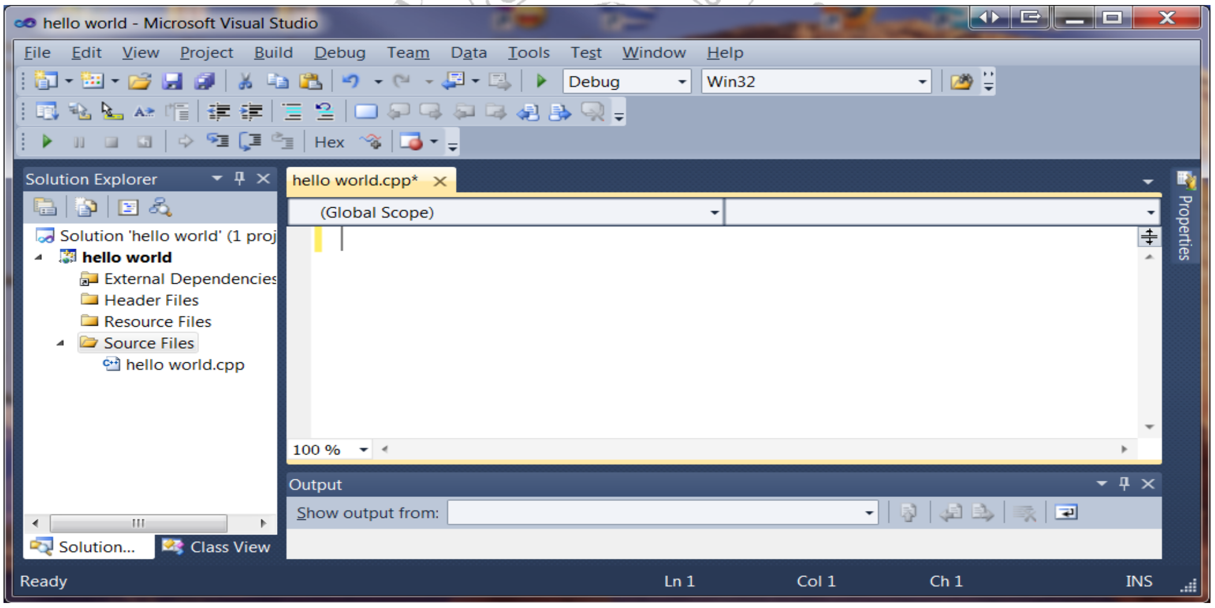

Add New Item - hello world
Installed Templates
Visual C++
UI
Code
Data
Resource
Web
Utility
Property Sheets
Sort by: Default
Search Installed Templates
Windows Form Visual C++
C++ File (.cpp) Visual C++
HTML Page (.htm) Visual C++
Static Discovery File (.disco) Visual C++
Header File (.h) Visual C++
Midl File (.idl) Visual C++
Resource File (.rc) Visual C++
Browse...
Add
Cancel

Type: Visual C++
Creates a file containing C++ source code

Name: hello world
Location: C:\Users\irwinhu\Documents\CS230\hello world\hello world\

Select **C++ Files (.cpp)** by clicking on it in the middle pane and enter an arbitrary file name (e.g., *hello world*). Click **Add** to proceed to the next step.

<u>*Step 4*</u>: The system displays the below window. The **Source Folder** in **Solution Explorer** pane contains the *hello world.cpp* file that was just added. The blank editing area/board is displayed with a *hello world.cpp* tab to enter the C++ source code.

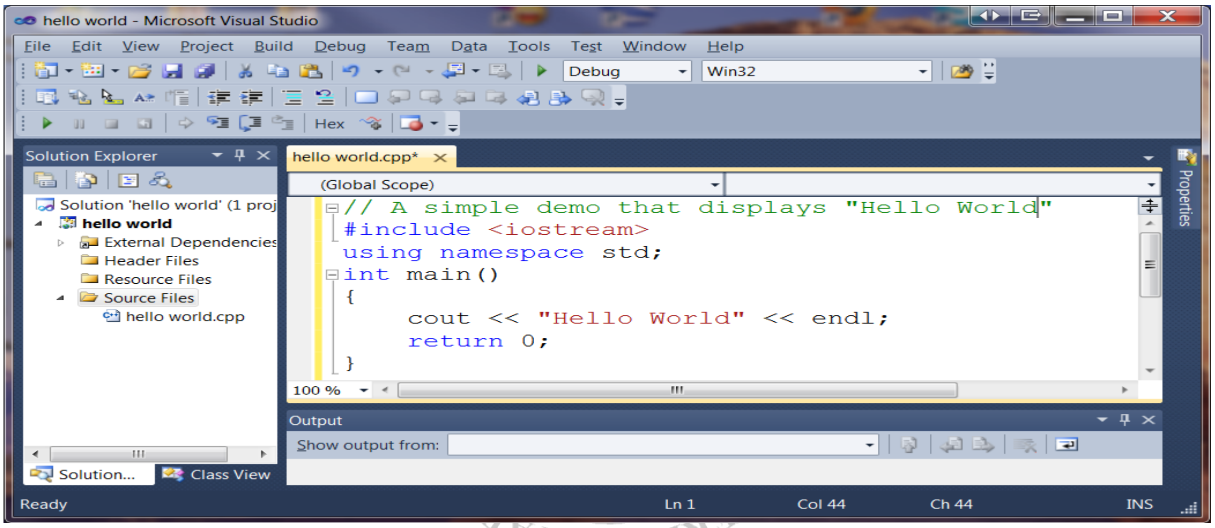


<u>Source Code</u>:

```
#include <iostream>
using namespace std;
int main()
{
cout << "Hello World!" << endl;
return 0;
}
```

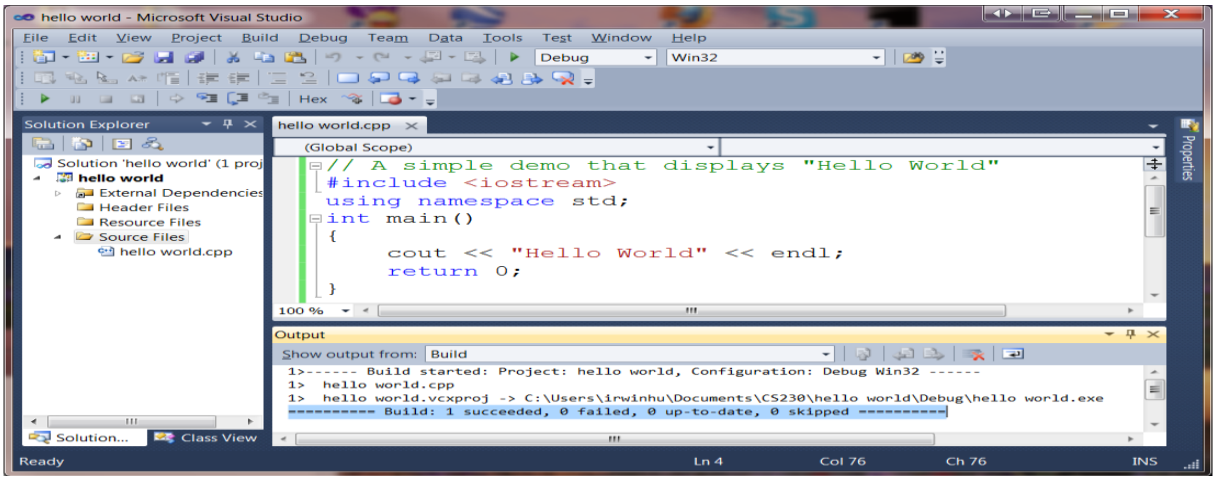
- #include: Lines beginning with a hash sign (#) are directives for the preprocessor. The directive #include tells the preprocessor to include the iostream standard file.
- using namespace std: All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name std.
- int main (): This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution. The word main is followed in the code by a pair of parentheses (()).
- cout << "Hello World!": This line is a C++ statement. cout represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters into the standard output stream.
- return 0: The return statement causes the main function to finish. return may be followed by a return code. A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution.

The same window after the source code is entered:



```
// A simple demo that displays "Hello World"
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World" << endl;
    return 0;
}
```

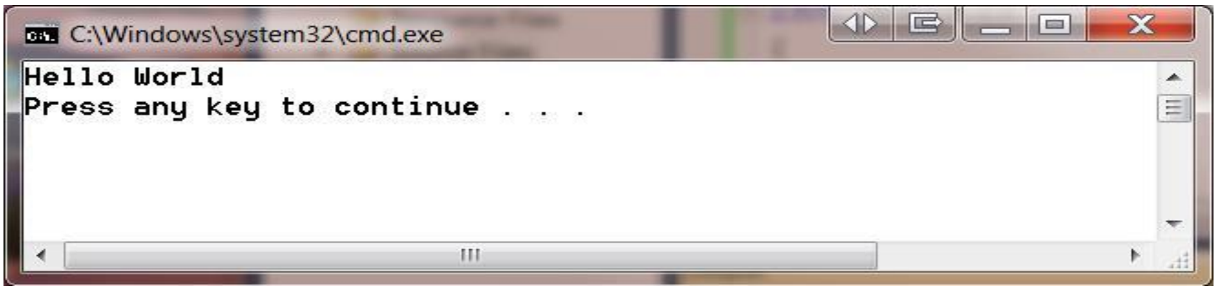
Step 5: To compile, link, load, and execute a program in a single step, click Debug in the menu bar and the click Start Without Debugging. If there is no error in your project, the below message `========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========` is displayed in the Output window as shown below.



```
// A simple demo that displays "Hello World"
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello World" << endl;
    return 0;
}
```

```
1>------ Build started: Project: hello world, Configuration: Debug Win32 ------
1>  hello world.cpp
1>  hello world.vcxproj -> C:\Users\irwinhu\Documents\CS230\hello world\Debug\hello world.exe
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

Also displayed in a separate window (C:\Windows\system32\cmd.exe) is the program output:



```
Hello World
Press any key to continue . . .
```

EXERCISE

1. What is a header file? Why the header file <iostream> is included in C++ project?

____

____

____

____

____

____

2. What is a namespace? Briefly describe the objective of the statement "using namespace std".

____

____

____

____

____

____

3. Use MS Visual C++ 2010 to create a project named, *Lab1* and then add a .CPP file named, *First* to this project. Now add given piece of code to file *Lab1* and compile this file. Execute this file afterwards and check the output of your program.

Code:

```
# include<iostream>
using namespace std;
int main ( )
{
  cout<<  ‟ My first Program in C++” ;
  return 0;
}
```

Attach output screens here : [**Hint:** *Use snipping tool to take screenshots*]

4. Use MS Visual C++ 2010 to create a project named, Prog2 and then add a .CPP file named, *Second* to this project. Now add given piece of code to file *Prog2* and compile this file. Execute this file afterwards and check the output of your program.

Code:

```
# include<iostream>
using namespace std;
int main ( )
{
  int x;
  x=5;
  cout<<  " x = " << x;
  return 0;
}
```

Attach output screens here : [**Hint:** *Use snipping tool to take screenshots*]

Lab Session 02

Familiarization with building blocks of C++ and conversion of mathematical expressions into an equivalent C++ statements

THEORY

Building Blocks of C++:

Variables and Constants

If the value of an item can 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/C++ are: `int, float, char, long, double` etc.

Operators

Basic: `+` `-` `*` `/` `%`
Assignment: `=` `+=` `-=` `*=` `/=` `%=`
(`++`, `--` may also be considered as assignment operators)
Relational: `<` `>` `<=` `>=` `==` `!=`

Typecasting

Typecasting allow a variable of one type to act like another for a single operation. In C++ typecasting is performed by mentioning the new data type followed by the variable name enclosed in parenthesis.

EXERCISE

1. Consider `num1`& `num2` as variables of type `int` , `average` as a `float` variable. Now mention error (if any) in your own words for given code fragments;

a- `average = num1 + num2 /2; //` num1 is 5 & num2 is 10

Error: ______________________________

b- `average = (num1 + num2 )/ 2; //` num1 & num2 both hold some odd values

Error: ______________________________

c- `average = (num1 + num2 )/ 2;` // num1 is 5 & num2 is 2

Error: ______________________________

d- `average = (num1 + num2 )/ 2.0;` // num1 is 5 & num2 is 2

Error: ______________________________

e- `average = float (num1 + num2 )/ 2;` // num1 is 5 & num2 is 2

Error: ______________________________

2. Why the following cannot be considered as valid C++ identifiers?

a- `int 3circles;`

Reason: ______________________________

b- `float big number;`

Reason: ______________________________

c- `char $name;`

Reason: ______________________________

d- `long  sum+sqr;`

Reason: ______________________________

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

a- `x = 7 + 3 * 6  / 2 - 1;`

Working :

b- `x = 2 % 2 + 2 * 2  - 2 / 2;`

Working :

c- `x =( 3 * 9 * ( 3 + (9 * 3 /  (3) ) ) );`

Working :

3. Mention output for following code fragment

```
cout<< " Size of char is : " <<sizeof (char) << " Bytes \n" ;
cout<< " Size of int is : " <<sizeof (int) << " Bytes \n" ;
cout<< " Size of long is : " <<sizeof (long) << " Bytes \n" ;
cout<< " Size of float is : " <<sizeof (float)<<" Bytes \n" ;
cout<< " Size of double is: " << sizeof(double) << "Bytes\n" ;
cout<< "Size of long int is:"<<sizeof(long int)<<"Bytes  \n";
```

Output:

4. Write equivalent C++ statements for given mathematical expressions.

a. root = $\frac{-b\pm\sqrt{b^2-4ac}}{2a}$

b. angle = $\tan^{-1} x/y$

c. z = $4\,x^5\,y^2$

d. y = $\log x$

e. y = $\sin^2 x - \cos^2 x$

f. z = $5\ln\,(x+y)$

g. z = $\frac{x+y}{x-y}$

5. Fill in the given chart.

S.No.	Data Type	Smallest Possible Value	Largest Possible Value
1	`char`		
2	`int`		
3	`unsigned int`		
4	`long`		
5	`float`		
6	`double`		
7	`long int`		

6. Make small programs of all parts (a- g) of exercise 4. Run those programs with some realistic values and attach hard copy of code here.

Lab Session 03a

Taking multiple inputs from console & controlling output's position

THEORY

Getting Input From the User

The input from the user can be taken by the following techniques; `cin, getch( ), getche( ), getchar( )` etc.

Displaying Output to the User

The output can be displayed on monitor with the help of following techniques; `cout, putch(), putchar` etc

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 escape sequence includes \n for new line, \b for back space, \r for carriage return, \" for double quotations, \' for single quotations & \\ for back slash etc.

Manipulators

Manipulators are operators, used with insertion operator to modify or to manipulate the way data is displayed. Two most common manipulators are `endl` (produces same effect as \n) and `setw`.

gotoxy(x,y)

gotoxy(x,y) is used to position the cursor on the character cell at a specified (col, row) coordinate in the current text window.

EXERCISE

1. What will be the output of following `cout` statements;

```
i- cout << " \n My name  is Syed Ali Abbas\b\b\baseen ";
```

____

```
ii-  cout<< "I \t am \t a \t good\n learner";
```

____

____

```
iii- cout<< " Course: " << endl << " \" Calculus \" ";
```

____

____

2. Write a single C statement with the help of `setw()` & `endl` manipulators to output the following on the screen:

```
Emp. ID     Name                        Salary
123456      Muhammad Ali                605000
789         Muhamamd Iqbal              70000
3456        Khanzada Liquat Ali Khan    99000
```

3. Write a single C statement to output the following starting from the 5th row and 10th column of the screen:

My name is "*Your Name*"
And my roll number is "00*Your_roll_no*"
I am a student of 'Computer and Information System Department'
In C++ language a tab is inserted by (\t)

4. Write necessary statements with appropriate variable declaration to take input the current time in the format hr:min from the user and display hours and minutes separately

Input:
Enter current time in the format hr:min 7:56

Output:
7 hours & 56 minutes

5. Write necessary statements with appropriate variable declaration to take the date of birth as input in the format DD-MM-YYYY from the user and display hours then calculates and displays the approximate age of the user in years.

6. Develop a program to display a similar output as given below. Use of gotoxy(x,y) is prohibited.

a.

```
         1
       2   2
      3  3   3
   4    4   4  4
5    5    5    5   5
```

b.

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

7. Attach hard copy of programs developed for exercise 4, 5 & 6 here.

Lab Session 03b

OBJECT

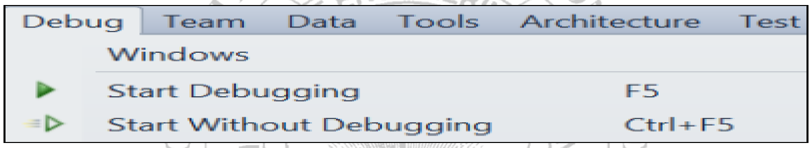
Debugging and Single-Stepping of Programs

THEORY

There are generally two types of errors namely syntax and logical errors. Syntax errors occur when a program does not conform to the grammar of a programming language, and the compiler cannot compile the source file. Logical errors occur when a program does not do what the programmer expects it to do. Syntax errors are usually easy to fix because the compiler can detect these errors. The logical errors might only be noticed during runtime. Because logical errors are often hidden in the source code, they are typically harder to find than syntax errors. The process of finding out defects (logical errors) in the program and fixing them is known as debugging. Debugging is an integral part of the programming process.

Program Debugging with Microsoft Visual C++ 2010

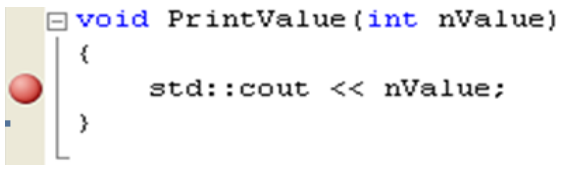
This lab session introduces the debugging feature of Microsoft Visual C++ 2010. The debugger can be started by clicking the leftmost button on the Debug toolbar, by selecting the Debug ➪ Start Debugging menu item, or by pressing F5.



The debugger has two primary modes of operation— it works through the code by single stepping (which is essentially executing one statement at a time), or runs to a particular point in the source code. The point in the source where the debugger is to stop is determined either by where the cursor has been placed or, more usefully, at a designated stopping point called a breakpoint. A breakpoint is a point in your program where the debugger automatically suspends execution when in debugging mode.

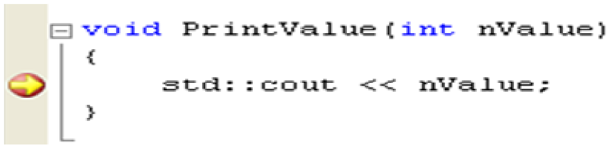
Creating a Breakpoint

In the VS editor, there is a margin on the left side. If this margin is clicked, VS will set a breakpoint on the matching code statement. Clicking a statement in code to give it the focus and pressing F9 sets a breakpoint too. A red dot is appeared in the margin and the matching statement highlighted in red.



```
void PrintValue(int nValue)
{
    std::cout << nValue;
}
```

To ensure VS stops on a breakpoint, the application must be running in debug mode. When the program hits the breakpoint, the breakpoint line will turn yellow and there will be a yellow arrow on the red dot in the margin. Clicking the Continue button (which is the same green arrow button used to start debugging) or pressing F5 will cause VS to resume execution. In order to stop debugging, select Debug | Stop Debugging, press F5, or click the Stop Debugging toolbar button (small blue square).



```
void PrintValue(int nValue)
{
    std::cout << nValue;
}
```

Customizing a Breakpoint

The preceding explanation described how to set a location breakpoint, where execution stops on a designated line in code. However, a program can be stopped executing based on various criteria, such as hit count, conditions, and more. There are several commands available in break mode as shown below.

Table 1. Options from the Breakpoint Context Menu

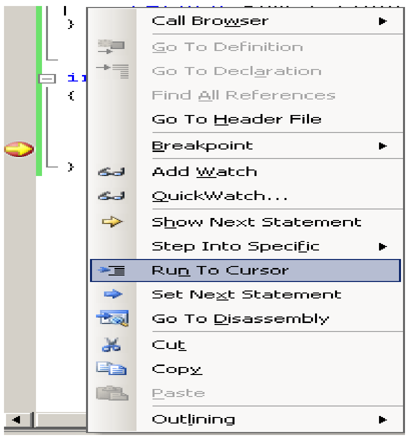
Option	**Meaning**
Delete Breakpoint	Removes the breakpoint.
Disable/Enable Breakpoint	It disables the breakpoint and then enables it later.
Condition	It allows entering an expression that can cause the program to stop if either the expression evaluates to true or the value of a variable has changed. The expression is based on variables in the code.
Hit Count	It makes the program break on that line every time, after a number of times the line has executed, when the count is a multiple of a number (i.e., every nth time), or when the number of hits is greater than or equal to a number.
Edit Labels	Breakpoints can be associated with labels to help organize breakpoints into groups.
Export	Exports breakpoints into an external XML file.

Stepping Through Code

It is the process of executing one or more lines of code in a controlled manner. At the most granular level, the code can be stepped through line-by-line. While moving line-by-line is often necessary, it could also be cumbersome, so there are ways to step over multiple lines of code, allowing them to silently and quickly execute. To step through code, open a project, set a breakpoint, and run with debugging until the program hits the breakpoint. At that point in time, various operations can be performed such as step, step over, and step out. Table 2 explains the stepping operations that are available.

Table 2. Step Operations

Operation	**Explanation**
Step Over	It executes the code in the current line and moves to the next line of code where it again pauses, waiting for programmer instruction. Perform a Step Over by selecting Debug | Step Over, pressing F10, or clicking the Step Over button in the toolbar.
Step Into Specific	When the current line is on a method call, a Step Into will move control to the first line of the method being called and execution will pause there. Perform the Step Into by selecting Debug | Step Into, pressing F11, or clicking the Step Into button in the toolbar.
Step Out	If the programmer is in a method, he can move back to the caller by performing a Step Out operation. Perform a Step Out by selecting Debug | Step Out, pressing SHIFT-F11, or clicking the Step Out button on the toolbar.
Run to Cursor	A quicker way when one doesn't want to keep a new breakpoint around is to right-click the line one want to stop at and select Run To Cursor. Again, no lines of code are skipped; the program will merely pause when it gets to the line the cursor is placed on.
Set Next Statement	It allows changing the path of execution of program while debugging. If the program is paused in a particular line and one wants to change the execution path, go to the particular line**,** right click on the line and select "Set Next Statement" from the context menu. The execution will come to that line without executing the previous lines of code. Shortcut key for Set Next Statement is Ctrl + Shift + F10**.**

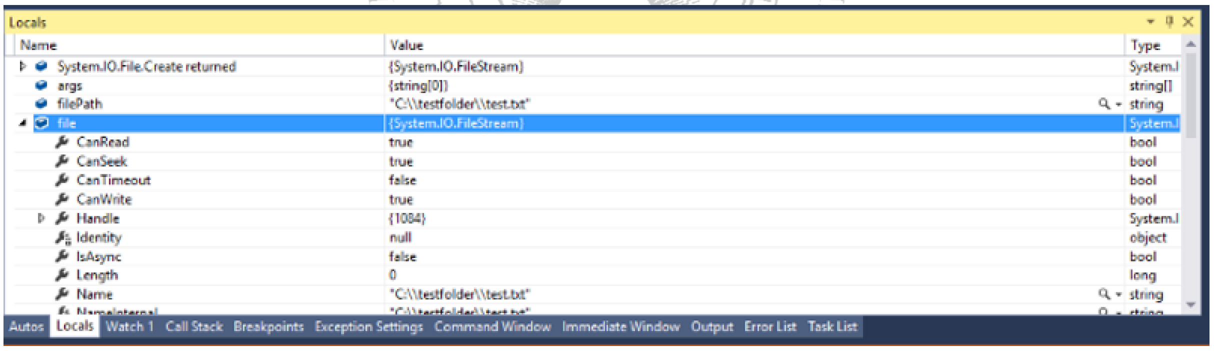


Inspecting Application State

Application state is the value of variables in the code, the current path of execution, or any other information that tells what the program is doing. While debugging, it's important to be able to view application state and compare what is really happening to what the programmer expected to happen.

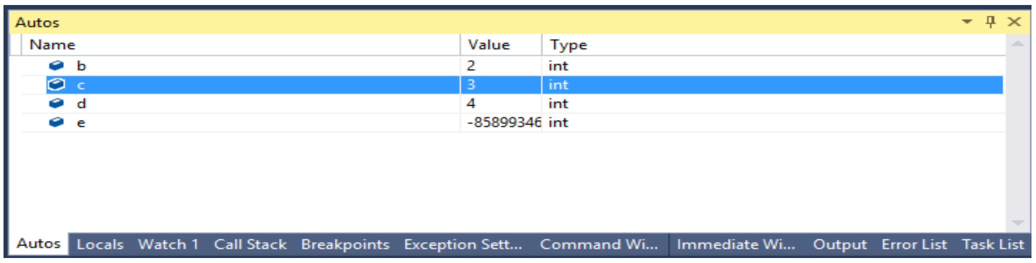
Locals and Autos Windows

The Locals and Autos windows show the variables in system at the current breakpoint. Locals give a list of all variables that the current statement could access (also referred to as in scope). The Autos window shows variables from the current and previous lines. The Locals and Autos windows can be opened from the Debug | Windows menu when the VS debug session is active and paused at a breakpoint. These windows may have already been placed at the bottom left of Visual Studio next to the Output window.



Locals

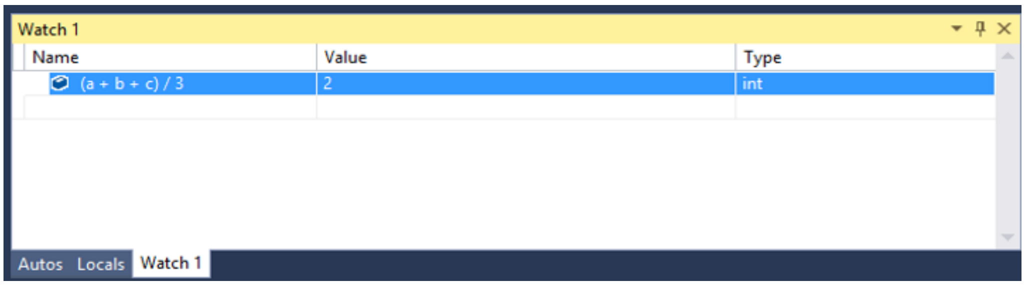
Name	Value	Type
System.IO.File.Create returned	{System.IO.FileStream}	System.I
args	{string[0]}	string[]
filePath	"C:\\testfolder\\test.txt"	string
file	{System.IO.FileStream}	System.I
CanRead	true	bool
CanSeek	true	bool
CanTimeout	false	bool
CanWrite	true	bool
Handle	{1084}	System.I
Identity	null	object
IsAsync	false	bool
Length	0	long
Name	"C:\\testfolder\\test.txt"	string



Autos

Name	Value	Type
b	2	int
c	3	int
d	4	int
e	-85899346	int

Watch Windows:
A Watch window allows you to create a custom list of variables to watch. The programmer can drag and drop variables from the editor or type a variable name in the Watch window.



Name	Value	Type
(a + b + c) / 3	2	int

Consider the following C++ codes containing logical errors. The errors are detected by debugging the code.

Source Code 1:
```
int count;
while (count < 100)
{
cout << count;
count++;
}
```

Logical Error
Uninitialized variables

Solution
```
int count = 0;
while (count < 100)
{
cout << count; count++;
}
```

Source Code 2
```
int a, b;
int sum = a + b;
cout << "Enter two numbers to add: ";
cin >> a;
cin >> b;
cout << "The sum is: " << sum;
```

Sample Run
Enter two numbers to add: 1 3 The sum is: -1393

Solution:
```
int a, b;
int sum;
cout << "Enter two numbers to add: ";
cin >> a;
cin >> b;
```

```
sum = a + b;
cout << "The sum is: "<< sum;
```

Source Code 3

```
char done = 'Y';
while (done = 'Y')
{ //...
cout << "Continue? (Y/N)";
cin >> done; }
```

Logical Error:
Infinite loop

Solution:

```
char done = 'Y';
while (done == 'Y')
{ //...
cout << "Continue? (Y/N)";
cin >> done;
}
```

EXERCISE

1. Fill out all the entities in table by their corresponding values by locals/autos window for all the variables and single stepping the program. (Consider **R** as your class roll number & **FL** as first letter of your name)

Code fragment	Before execution			After execution		
(i)	a	b		a	b	
`int a=5, b=10;`						
`b = a++;`						
(ii)	x	y		x	y	
`int x=10, y=20;`						
`y=++x;`						
(iii)	n1	n2	n3	n1	n2	n3
`int n1=10,n2=10,n3;`						
`n3= n1++ + ++n2;`						
(iv)	n	sqr		n	sqr	
`int n=20; long sqr;`						
`sqr= n*n ;`						

(v)
`char ch=`**`'FL'`**`;`
`    ch ++;`
	ch	▨	▨	ch	▨	▨

		▨	▨		▨	▨
		▨	▨		▨	▨

(vi)
`int x=R,y=R+50;float a;`
`a=(x+y)/2;`
	X	Y	a	X	Y	a

(vii)
`unsigned int x=-5;`
`x= x * x ;`
		x			x	

2. Add breakpoint at last line of given code and then single step the program.

```
void main( )
{
int  length, width;
long area;
length = 5;
cout << " Enter width of the door" ;
cin>> width;
area = length  * width ;
cout<<"Area of the door:"<<area<<"sqr cm"; // add breakpoint here
}
```

Before execution			After execution		
length	width	area	length	width	area

Lab Session 04

OBJECT

Decision making in programming (if-else & 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. Four decision making structures:

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

The `if` statement

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

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

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

The `if-else` statement

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

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

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

Conditional (Ternary) Operator

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

The conditional operator takes three expressions and returns a value:

```
(expression) ? (statement1(s) ) : (statement2(s));
```

This line is read as, "If expression is true, statement1(s) will be executed; otherwise, statement2(s) will be executed." Typically, this value would be assigned to a variable.

EXERCISE

1. Write a program that takes a positive integer as input from user and checks whether the number is even or odd, and displays an appropriate message on the screen. [**Note:** For negative numbers, program does nothing.]

a) Using if-else:

b) Develop the program mentioned in Q.1 using conditional operator [**Note:** Display some appropriate message for negative numbers]

2. The programs given below contain some syntax and/or logical error(s). By debugging and single stepping for all the variables, mention the error(s) along with their categorization into syntactical or logical error. Also write the correct program statements.

i- to check whether three numbers are equal or not

```
int a,b,c; char ch;
cout<< "enter three numbers in this format a,b,c" ;
cin>>a>> ch >> b >> ch >> c;
if (a= =b)

     if (b = = c)
```

```
          cout <<  a<<” , ” <<b<<” & “<<c << “ are equal;
     else
cout<< a << “ and  ” << b<< “ are different ”;
}
```

a- enter `3,4,5` as input & write the output
Output :

b- enter `3,3,5` as input & write the output
Output :

c- enter `5,10,10` as input & write the output
Output :

d- enter `10,10,10` as input & write the output
Output :

e- Nature of error(s) [Tick as appropriate] : _Syntactical / Logical_____________

f- Corrected Code:

ii- To check whether the number is divisible by 2 or not:

```
          int num;
          cout <<“enter any number” ;
          cin >>num;
          if(num%2=0)
                cout <<“Number is divisible by 2”;
          else
          cout <<“number is not divisible by 2”;
```

a) Run the code and mention the output, if user enters;

(i) 15 as input ____________________

(ii) 20 as input ____________________

b) Identify logical error in above code and mention in your own words.

____

____

c) What correction can be made to above code? Give corrected statement only.

____

3. Consider the given program.

```
void main ( )
{
int n1=10, n2=20, n3=30, max;
max = (n1>n2) ? (n1>n3 ? n1:n3)  : (n2>n3? n2:n3);// 2nd line
cout << "max of " <<n1<<","<<n2<<"&"<<n3<<" is " <<max;
}
```

(i) Add watches for given variables & expressions and fill up the given table.

Watch	Before Execution (of 2nd line)	After Execution (of 2nd line)
n1>n2		
n1>n3		
n2>n3		
n1>n3 ? n1:n3		
n2>n3 ? n2:n3		
max		

(ii) Now consider `n1 = 40, n2= 30, n3=10` and repeat (i)

Watch	Before Execution (of 2nd line)	After Execution (of 2nd line)
n1>n2		
n1>n3		
n2>n3		
n1>n3 ? n1:n3		
n2>n3 ? n2:n3		
max		

4) A programmer wants to display "Kaman Akmal" on screen, if score >30, Shoaib Akhtr, if 20<score <30, and Shahid Afreedi if 10<score <20. For this purpose, he has developed following piece of code;

```
void main(void)
{
int a=100;
if(a>10)
cout <<"Shahid Afridi";
else if(a>20)
cout<<"Shoaib Akhtar";
else if(a>30)
cout<< "Kamran Akmal";
          }
```

d) Run the code and mention the output for given cases:

Input	5	15	25	35
Output				

e) Identify logical error in above code and mention in your own words.

f) Give corrected code.

5. According to Russian Peasant Method (A fast multiplication algorithm, which is as old as 1700 B.C.), product of x & y can be found using the truth;

`x * y = 2x * (y/2)` if y is an even number &

`x * y = x + x * (y-1)` if y is an odd number.

Now write a program that takes two integers `x & y` as input and finds & displays their product by using Russian Peasant Method.

(i) Using conditional operator

(ii) Using if-else

Lab Session 05

OBJECT

Implementation of simple & scientific calculators using switch–case statements

THEORY

The `switch` Statement

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

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

`goto` Statement

goto statement can be used to branch towards any statement (forward or backward) which is marked by some label.
e.g.

```
      statement 1;
here: statement 2;
      ….
      ….

     goto here;
```

EXERCISE

1. Develop a four basic mathematical functions calculator that takes input form user and performs any one of the selected operations and displays the result. Use switch case approach to develop this program. Also make use of “goto” statement to continuously take input of the operator, if user enters some unknown operator (other than, +,-, *, /)

Mini Project:

Develop an advanced calculator that works in two modes; 1. Simple & 2. Scientific (log and anti-log, trigonometric & inverse trigonometric operation, square root, power etc). Program then takes operands as input form user and performs any one of the selected operations and displays the result. Use switch case approach to develop this program.

Submission: Mini Project is to be submitted in next lab session.

Attach source code here.

Lab Session 06

OBJECT

Generalization of tasks exploiting the benefit of for loops & their nesting

THEORY

Types of Loops

There are three types of Loops:

- for Loop
- while Loop
- do - while Loop

Nesting may extend these loops.
The for Loop

```
for(initialize; condition; increment)
{
     Do this;
}
```

This loop runs as long as the condition in the parenthesis 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.

EXERCISE

1 (a) Write a program that takes two integers `x & y` as input and finds & displays their product by performing repeated addition. For example: `3 x 5 = 3+3+3+3+3.`

1(b) Run the above code and mention the output for given cases. Also add watch for the loop decision variable `i` and give its value, when for loop is just terminated.

Input	3 x 4	0 x 12	5 x 0	5 x 2
Output				
Last value of 'i'				

2. Write a program to print the accurate average of 5 integer values, entered by user using for loop. [**Hint:** Apply concept of type casting]

3. What will be the output for given code fragment.

```
void main(void)
{
int i=10;
for(; ++i <=15 ;)
 cout<<"i= " << i << "\t";
}
```

Output:

4. Write a program that takes a number as input then checks and displays an appropriate message whether the number is a prime number or not.

5. Write a program that prints following as output.

```
                1
               2 2
              3 3 3
             4 4 4 4
            5 5 5 5 5
                :
                :
        9 9 9 9 9 9 9 9 9
```

6. Develop a program that takes a 5 digit positive number as input and displays the number of trailing zeros at the end of the number. (e.g. 50900 has two trailing zeros).

7. Make use of `continue` and `break` statement to develop a program that gives following as output.

```
Output:
        9,    8,   6,   5,   4,   3, countdown aborted!
```

____

____

____

____

____

____

8. Add a breakpoint at the last line of given code fragment, then first compile your program and then single step your program and mention the output. Add watches for all the variables of this program and mention their values before and after the execution of last `cout` statement.

```
for ( int n=1, count=1; n<=20; n+=2,count++) ;
       cout << "\t" << n;
cout << " loop executed for " <<count<< "number of times";
```

Output:	Before Execution		After Execution	
	n	count	n	count

9. Use nested for loop to generate the following as output.

```
         1      2      3      4      5
        ______________________________
1     | 1      2      3      4      5
2     | 2      4      6      8      10
3     | 3      6      9      12     15
4     | 4      8      12     16     20
5     | 5      10     15     20     25
```

____

____

Lab Session 07

OBJECT

Performing simple bank transactions as many times as user wants (Applying the concept of conditional break , while & do-while loops)

THEORY

Loops Revisited: The while Loop

```
while(condition is true)
{
    Do this;
}
```

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 removed.

The do-while Loop

```
do
{
    this;
}
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 necessarily executed.

EXERCISE

1. Develop a program to perform two simple transactions in a bank as long as user enters 'y. to continue.

Sample Output:

Enter your ID: ****	Main Menu *********** 1. Deposit Money 2. Withdraw Amount 3. Login as Different User Select your choice ….	*(after completing the selected transaction)* Do you want to continue? [y/Y] _ *(goes to Main Menu, if y/Y is pressed)*

2. Write only the necessary statements to print all the numbers in the range 100 - 150 which are divisible by 4 (tab separated), using ***while*** loop.

Output (of above code) :

3. Write a program that takes a positive integer (<9999) as input and expresses the number in words. Use ***do-while*** loop.

Input: Enter a positive integer (less than 9999) : 6342	
Output: 6 thousand 3 hundred 4 tens and 2 ones	

4. Write a program that keeps on taking characters as input from the user until he/she presses the enter key and displays the total number of words entered.

Lab Session 08

OBJECT

Implementing functions

THEORY

The general structure of a function declaration is as follows:

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

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

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

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

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

Consider a simple example of function declaration, definition and call.

```
void function1(void);
void function2(void) // definition of function 2
{
   cout<<"Writing in Function2 \n";
}
void main(void)
{
   cout<<"Writing in main \n";
   function1( ); // calling function 1
}
void function1(void) // definition of function 2
{
   cout<<"Writing in Function1 \n";
   function2( ); // calling function 2

}
```

Consider another example that adds two numbers using a function **sum()** .

```
void sum(void);
void main(void)
```

```
{
cout<<” \n Program to print sum of two numbers\n”;
sum(void);
}
void sum(void)
{
int num1,num2,sum;
char ch;
cout<<“Enter number1:number2 ”;
cin>> num1>>ch>>num2;
sum=num1+num2;
cout<<“Sum of <<num1<<” & “ <<num2 << “is” <<sum;
}
```

Built-in Functions

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

EXERCISE

1.Give function definition of given prototypes.

(i) `void SumOfComplex( int real1, int imag1, int real2, int imag2);`
// Takes two complex numbers as argument and prints their sum.

(ii) `void RepeatChar( char ch, int n);`

// displays character ‘ch’ , ‘n’ times on screen .

(ii) `char LowerCase(char ch);`

// Converts ch to lowercase , if ch is a uppercase alphabet

(iv) `long Factorial(int num);`

// returns factorial of some positive integer.

(v) `long Combination(int n, int r);`

// returns $^{n}C_{r}$ with the help of another function named `factorial ( )`

2. Single step the given code, and fill up the entities of given table, by adding watches for all variables of your program.

```
int x; // global variable
int LocalVar(int a,char
ch2;
void main (void)
{
int a=5, b=10;
char ch1= 'A';
cout<< "a = "<< a;
cout<<"\n ch1= "<< ch1;
cout<<"\n b= " << b;
x=30;

b= LocalVar( a, ch1);

cout<< "a = " << a;
cout<<"\n ch1= " << ch1;
cout<<"\n b= " << b;
x=90;
getch();

}
int LocalVar(int a,char
ch2)
{
int z=100;
a= a *10;
ch2 ++;
x=0;
cout<< "a = " << a;
cout<<"\n ch2= " << ch2;
cout<<"\n z= " << z;
return z;
}
```

Before Execution				After Execution			
a	b	ch1	x	a	b	ch1	x

Before Execution				After Execution			
a	b	ch1	x	a	b	ch1	x

3. Identify the errors (if any) in your own words in the given pieces of code:

a)
```
func(int a,float b)
{
return (5.75);
}
```

b)
```
int , float newfunction(void)
{
    return (7, 6.96);
}
```

4. Build a function that takes two natural numbers `x & y` as arguments, and returns their GCD (greatest common divisor) by applying given Euclid's algorithm;

a. Divide *x* by *y* and let *r* be the remainder.
b. If *r* is 0, *y* is the answer; if *r* is not 0, continue to step c.
c. Set *x* = *y* and *y* = *r*. Go back to step a.

5. main( ) is a function. Write a function void func( ) which calls main( ). What is the expected output of this program?

6. Build a function that takes two natural numbers `x & y` as arguments, and returns their LCM (Least Common Multiplier) by applying Euclid's algorithm. [**Hint:** The Least Common Multiple of two numbers is the smallest positive integer number that can be divided by the two number without producing a remainder & is equal to (GCD of x & y) / (product of x & y).

Lab Session 09

OBJECT

Implementing recursive functions

THEORY

Recursion

Ability of a function to call itself

e.g.

```
void function ( int n)
      {
      ….
      function(n);
      ….
      }
```

EXERCISE

1. Develop a program that calculates the factorial of a number using recursion.

2. Develop a program that takes a sentence as input from the user terminated by enter key and displays the entered sentence in reverse order.

Input: This program prints backward using recursion

Output: noisrucer gnisu drawkcab stnirp margorp sihT

3. Develop a program that calculates the following series using recursion.

n + (n-1) + (n-2) + ………… + 3 +2 + 1

Lab Session 10

OBJECT

Creating & editing lists using arrays

THEORY

An 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,

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

declares an array of 25 `long` integers, named `LongArray`. 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 necessary statements to store first ***n*** Fibonacci numbers in an array, where ***n*** is a value (< 50), taken as input from user.

2. Declare an array of length 10. Take input in the array using a function `input(int [])`. Use another function `largest (int [] )` to find the largest element in that array.

3. Write a program to search a number in an array of 10 elements and displays some appropriate message. Program should be able to keep asking user for other number search, till he presses ***y*** to continue.

Lab Session 11

OBJECT

Manipulating matrices using two dimensional arrays

THEORY

Two Dimensional Arrays

Multidimensional arrays can be described as "arrays of arrays". For example, a bi-dimensional array can be imagined as a two-dimensional table made of elements, all of them of a same uniform data type.

The C++ syntax for this is:

```
int array [3][5]; // it has 3 rows & 5 cols
```

Multidimensional arrays are not limited to two indices (i.e., two dimensions). They can contain as many indices as needed. Although be careful: the amount of memory needed for an array increases exponentially with each dimension. For example:

```
long BigArray [100][365][24][60][60];
// needs more than 123 billion bytes in memory
```

EXERCISE

3. Implement a Matrix Calculator, capable of performing simple operations on a matrix of size 3 x 3.

Sample Output:

Welcome Screen with student's name	Main Menu *********** 1. Addition 2. Multiplication 3. Transpose Select your choice ….	*(after completing the selected operation)* Do you want to continue? [y/Y] _ *(goes to Main Menu, if y/Y is pressed & terminates otherwise)*

2. Use two dimensional arrays to store given monthly sale chart of different employees during 1st quarter of year 2015.

	Jan	Feb	Mar	Apr
Emp1	3,20,000	5,56,000	4,98,000	4,79,000
Emp2	5,25,000	4,56,000	5,50,000	4,79,000
Emp3	4,30,000	3,90,000	3,75,000	3,20,000
Emp4	3,25,000	4,59,000	4,55,000	4,95,000
Emp5	4,80,000	5,00,000	4,35,000	4,40,000
Emp6	5,90,000	5,70,000	5,20,000	4,25,000

Monthly Sale Chart in Rs.

Now, give function definition only that;

(i) Searches the best employee with highest overall sale during first quarter, so that a letter of appreciation can be awarded to him.

```
void HighestSale (long [ ]  [4]);
```

(ii) Takes month number as input, and gives employee number with lowest sale, so that a warning letter can be issued to him to improve his performance.

```
void PoorPerformance(long [ ]  [4]);
```

Lab Session 12

OBJECT

Implementing Language Encoder Using Strings

THEORY

Strings

Array of characters can be used to store strings (character sequences).
e.g. `char title [10] = " Maths";`
or `char title [10] = { 'M', 'a', 't', 'h', 's', '\0' };`

Similarly, two dimensional char array can be used to store & manipulate multiple strings.

```
char tools [3][20]={ "Scissors", "Plier", "Cutter"};
```

EXERCISE

1. On a planet, a language named 'Hsilgne' with alphabets similar to English letters are used. Computer Scientists have discovered that one can convert any English language word to Hsilgne by simply following given rule;

Letter A is replaced with Z,
B with Y,
C with X & so on.
(All other digits & symbols remains same)

Now you are asked to implement a function that's converts any English word to Hsilgne word.

```
 void  EnglishToHsilgne (char engword[ ] , char hsiword[ ]);
```

2. Give function definition decode any Hsilgne signals back to English words.

```
void  HsilgneToEnglish (char hsiword [ ] , char engword [ ]);
```

3. Write a program that takes five names as input and sort them by their *lengths*. Use a separate function for sorting. [**Note:** `strlen( )` can't be used]

Lab Session 13

OBJECT

Working with Pointer variables & dynamic memory allocation

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 `ip`,

```
    int *ip;
```

means that the expression `*ip` 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 ip. If instead of int we declare

```
  char * ip;
```

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

EXERCISE

1. Consider the following piece of code;

```
void main (void){
int x,y;
int *px, *py;
x=  20;
y=40;      }
```

Now write necessary statements to accomplish the given task and also mention the output (if any)

i. Assign address of variable x to px, and display contents of x, &x, px, &px and *px.

_____________________ _____________________ _____________________

_____________________ _____________________ _____________________

_____________________ _____________________ _____________________

Output:

ii. Add `*px = *px *  *px;` to the above program and display the output.

Output:

iii. Add `py = px;` to above code and display contents of py, &py and *py.

Output:

iv. Add `*px++;` to your code and display new *px.

Output (Expected) :	Output (Actual) :

v. Add `*(px++)  ;` to your code and display new *px.

Output (Expected) :	Output (Actual) :

vi. Now assign address of y to variable px and display new *px.

Output:

vii. What will be the impact of adding `px  ++` to the above code? Display new px and *px.

Output (Expected) :	Output (Actual) :

2. Consider the following piece of code and mention the output;

```
void main (void)
{
char ch= ‘L’; char * ptrch=&ch;
float r= 7.69; float ptrr=&r;
cout << “\n ch =”<< ch;
cout << “\n &ch =”<< &ch;
cout << “\n ptrch =”<< ptrch;
cout << “\n &ptrch =”<< &ptrch;
cout << “\n *ptrch =”<< *ptrch;
ptrch++;
cout << “\n ptrch =”<< ptrch;
cout << “\n *ptrch =”<< *ptrch;

cout << “\n r =”<< r;
cout << “\n &r =”<< &r;
cout << “\n ptrr =”<< ptrr;
cout << “\n &ptrr =”<< &ptrr;
cout << “\n *ptrr =”<< *ptrr;
ptrr++;
cout << “\n ptrr =”<< ptrr;
cout << “\n *ptrr =”<< *ptrr;
getch();
}
```

```
Output:
```

3. Using dynamic memory allocation, declare an array of the length user wants. Take input in that array and then print all those numbers, input by the user, which are divisible by 3.

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

5. Write pointer notation equivalent to the following array notations:

    i. `arr[10]` : ______________________________

    ii. `arr2D[5][6]` : ______________________________

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

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

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

iii. `void swap (int *x, int *y );`
// Swaps values of x & y

Lab Session 14

OBJECT

Creating a list of records using structures and its manipulation

THEORY

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

EXERCISE

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

2. (i) Declare a structure named **employee** that stores the *employee id*, *name*, *salary* and *department*.

(ii) Also creates two different instances of structure employee, created in part (i). The name of first instance must be same as `your first name` and set second name as `dummy`

(iii) Write C++ statement using sizeof operator to check how much memory is allocated to instance dummy.

____ Output: _______________

(iv) Initialize `dummy` with an employee ID of `1259`, name of the employee should be `Hadi` , salary should be `45,000`/= and department name is “`Finance`”.

____

____

____

(v) Declare & initialize variable `newemp` with following attributes using a single statement. (ID: 2897, Name : Smith, Department: Marketing, Salary: 50,000)

____

____

(vi) Write necessary statement to copy `dummy` to some other structure employee.

____

____

(vii) Use size of operator to find how many bytes are allocated to struct employee `dummy`? Mention the statement used.

____

(viii) Declare an array of 5 employees for the structure defined in part(i) . Also write statements to assign the following values to the employee [3].
Employee id = “Your_roll_no” *salary* = 30,000 and *department* = “IT dept”

____

____

____

(ix) Write necessary statement to initialize all the elements of above array.

____

____

(x) Write a function to take input in above array of struct employee.

(xi) Write a function that prints the highest salaried person amongst the employees.

(xii) Write a function that search & display records of all those employees, whose salary in greater than 15000.

(xiii) Write a function that search & display records of all those employees, who are working in Finance department.