C-Style Strings

Lab Sections

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

C-style strings

1. Objectives

After you complete this experiment you will be able to implement and use a c-style string

2. Introduction

A string is a character array which ends with the null character. The string library provides several functions that can be used to manage string or character arrays.

3. Definitions & Important Terms

We will define several terms you need to know to understand c-style string. They are as follows:

a. A string is a character array.
b. The null character is represented using the '\0' character.
c. The [ ] operator is used to access the members of a string
d. The index/subscript of a cell in a string must be between 0 and the length of the string minus 1.
e. The capacity of an array is the number of elements it can hold.
f. Always include one extra cell in your capacity for the null character.

4. Declaration Syntax

Consider the following syntax when declaring strings in C:

```
a. char my_string[10]; //creates a static array of size 10
b. char my_string[ ] = "Hello"; //creates a string of size 6 including the
                                //null character
```

More information on c-style strings can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate the declaration, initialization and implementation of c-style strings. Enter, save, compile and execute the following program in MSVS. Call the new directory "cStyleStringsExp1" and the program "cStyleStrings1.cpp". Answer the questions below:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
	char my_name[20] = "James Madison";
	char her_name[ ] = "Michelle Obama";
	char his_name[20];

	cout<<"my_name = "<<my_name<<endl;
	cout<<"The length of my_name is "<<strlen(my_name)<<endl;
	cout<<"The capacity of my_name is "<<sizeof(my_name)<<endl;

	cout<<"her_name = "<<her_name<<endl;
	cout<<"The length of her_name is "<<strlen(her_name)<<endl;
	cout<<"The capacity of her_name is "<<sizeof(her_name)<<endl;

	strcpy(his_name,"Barack Obama");
	cout<<"his_name = "<<his_name<<endl;
	cout<<"The length of his_name is "<<strlen(his_name)<<endl;
	cout<<"The capacity of his_name is "<<sizeof(his_name)<<endl;

	return 0;

}
```

Question 1: Please explain why the capacity and the length values for each string are different in the output produced by the program in Step 1 above?

Question 2: What is the does statement "strcpy(his_name,"Barack Obama");" do?

Question 3: Change the library from "string" to "cstring". Does the program compile without producing any errors?

Step 2: In this experiment you will investigate the declaration, initialization and implementation of c-style strings. Enter, save, compile and execute the following program in MSVS. Call the new directory "cStyleStringsExp2" and the program "cStyleStrings2.cpp". Answer the questions below:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
	char his_name[20] = "James";
	char her_name[20];

	her_name[0]='M';
	her_name[1]='a';
	her_name[2]='r';
	her_name[3]='y';

	cout<<"his_name length equals "<<strlen(his_name)<<endl;
	cout<<"his_name is "<<his_name<<endl;
	cout<<endl<<endl;
	cout<<"her_name length equals "<<strlen(her_name)<<endl;
	cout<<"her_name is "<<her_name<<endl;

	return 0;
}
```

Question 4: Did the program produce any compiler errors or warnings?

Question 5: Was the output correct? Explain your answer?

Question 6: Can you state a rule that c-style strings must following when using the string library?

Step 3: In this experiment you will learn how to use functions in the string library. Enter, save, compile and execute the following program in MSVS. Call the new directory "cStyleStringsExp3" and the program "cStyleStrings3.cpp". Answer the questions below:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
	char s[ ] = "123abc456def789ghi";
	char r[40];

	cout<<"The string s contains "<<s<<endl;
	cout<<"The length of s equals "<<strlen(s)<<endl<<endl;

	strcpy(r,s);
	cout<<"The string r contains "<<r<<endl;
	cout<<"The length of r equals "<<strlen(r)<<endl<<endl;

	strncpy(r,"XXXXXX",3);
	cout<<"The string r contains "<<r<<endl<<endl<<endl;

	strcpy(r,"abcdef");
	cout<<"Now the string r contains "<<r<<endl;
	cout<<"The current length of r equals "<<strlen(r)<<endl<<endl;

	strcat(r,r);
	cout<<"Now the string r contains "<<r<<endl;
	cout<<"Now the current length of r equals "<<strlen(r)<<endl<<endl;

	return 0;
}
```

Question 7: What compiler warnings were given?

Question 8: What operation does the function "strcpy" perform? Please answer in detail?

Question 9: What operation does the function "strncpy" perform? Please answer in detail?

Question 10: What operation does the function "strcat" perform? Please answer in detail?

Step 4: In this experiment you will investigate the operation of the strcmp function. Enter, save, compile and execute the following program in MSVS. Call the new directory "cStyleStringsExp4" and the program "cStyleStrings4.cpp4". Answer the question below:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
	char s[ ] = "123";
	char r[ ] = "abc";
	char x[ ] = "124";
	char y[ ] = "abc";

	if (strcmp(r,y) == 0)
	{
		cout<<"String "<<r<<" and string "<<y
			<<" are equal."<<endl<<endl;
	}

	if (strcmp(s,x) == -1)
	{
		cout<<"String "<<s<<" has a lower lexicographical "
			<<"order than string "<<x<<"."<<endl<<endl;
	}

	if (strcmp(x,s) == 1)
	{
		cout<<"String "<<x<<" has a higher lexicographical "
			<<"order than string "<<s<<"."<<endl<<endl;
	}

	return 0;
}
```

Question 11: Explain the operation of the strcmp function and the output produced by the program in step 4.

Classes:
Copy Constructor

Lab Sections

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

Classes: Copy Constructor

1. Objectives

After you complete this experiment you will be able to implement a copy constructor.

2. Introduction

A copy constructor is called to make a copy of an object. The copy constructor is automatically called during one of the following events:

a. when an object is passed-by-value;
b. when an object is returned in a return statement;
c. when an object appears in a declaration/initialization statement.

3. Definitions & Important Terms

We will define several terms you need to understand before implementing copy constructors. They are as follows:

a. A **shallow copy** is a **dependent bit-by-bit** copy of an object. The objects involved will share the same dynamic memory;
b. A **deep copy** is an **independent** copy of an object. The objects involved will have their own copy of dynamic memory;
c. The **copy constructor** is used to make a deep copy of an object;
d. When an object does not use dynamic memory a copy constructor does not need to be implemented;
e. If you do not implement a copy constructor, one will be implemented for you. However, this copy constructor will perform a shallow copy of an object;
f. Whenever dynamic memory is allocated using the new function of a class, a **destructor** must be implemented to de-allocate that memory; it has the same name as the class, no formal parameters, no return type and is prefaced with a tilde character (~).
g. The **current object** is the object that called the copy constructor;
h. **this** is a pointer to the current object;
i. The **object that is being copied (source object) is passed explicitly** through the corresponding formal parameter of the copy constructor;
j. The **object that is being copied into (destination/current object) is passed implicitly through the "this" pointer**.

4. Declaration Syntax

Consider the following:

a. Class_name(const Class_name &); //prototype inside class declaration
b. Class_name::Class_name(const Class_name & Formal_parameter) //functon header

Notice that the copy constructor does not have a return type and has one constant formal parameter passed-by-reference of the same type as the class.

More information on the copy constructor can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate how an object is copied.

Enter, save, compile and execute the following program in MSVS. Call the new project "CopyConstructorExp1" and the program "CopyConstructor1.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

const int SIZE=5;

class ARRAY_CLASS
{
public:
	ARRAY_CLASS();//default constructor
	void Add(int); //mutator
	void Print(); //accessor
	int * Get_Address(); //accessor
private:

	int *A;
	int count;
};

ARRAY_CLASS::ARRAY_CLASS()
{
	cout<<"Default Constructor has been Called!\n";
	A = new int[SIZE];
	count = 0;
}

void ARRAY_CLASS::Add(int item)
{
	if (count<SIZE)
		A[count++]=item;
	else
		cout<<"Array Full\n";
}

void ARRAY_CLASS::Print()
{
	for(int i=0; i<count; i++)
		cout<<"A[i] = "<<A[i]<<endl;
}
```

```
int * ARRAY_CLASS::Get_Address()
{
	return A;
}

int main()
{
	ARRAY_CLASS B;

	B.Add(1);
	B.Add(2);
	B.Add(3);
	B.Add(4);
	B.Add(5);
	B.Print();

	cout<<endl<<endl;

	ARRAY_CLASS A = B;

	A.Print();

	cout<<"A holds address location = "<<A.Get_Address()
	<<" and B holds address location "<<B.Get_Address()<<endl;

	return 0;
}
```

Question 1: Referring to the declaration/initialization statement and the output of the last cout statement in the main function of the program of Step 1, what type of copy was implementated, deep or shallow.? Explain your answer.

Question 2: Was a copy constructor implemented in the program in Step 1? Explain you answer.

Step 2: In this experiment you will investigate how a copy constructor is implemented. Enter, save, compile and execute the following program in MSVS. Call the new project "CopyConstructorExp2" and the program "CopyConstructor2.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

const int SIZE=5;

class ARRAY_CLASS
{
public:
	ARRAY_CLASS();//default constructor
	ARRAY_CLASS(const ARRAY_CLASS &);//copy constructor
	~ARRAY_CLASS(); //destructor
	void Add(int); //mutator
	void Print(); //accessor
	int * Get_Address(); //accessor
private:

	int *A;
	int count;
};

ARRAY_CLASS::ARRAY_CLASS()
{
	cout<<"The Default Constructor has been Called!\n";
	A = new int[SIZE];
	count = 0;
}

ARRAY_CLASS::ARRAY_CLASS(const ARRAY_CLASS & Org)
{
	cout<<"The Copy Constructor has been Called!\n";

	count = Org.count;

	A = new int[SIZE];

	for(int i=0; i<count; i++)
	{
		A[i] = Org.A[i];
	}
}

ARRAY_CLASS::~ARRAY_CLASS()
{
	cout<<"The Destructor has been Called!\n";
	delete [ ] A;
	A=0;
	count = 0;
}
```

```
void ARRAY_CLASS::Add(int item)
{
	if (count<SIZE)
		A[count++]=item;
	else
		cout<<"Array Full\n";
}


void ARRAY_CLASS::Print()
{
	for(int i=0; i<count; i++)
		cout<<"A[i] = "<<A[i]<<endl;
}

int * ARRAY_CLASS::Get_Address()
{
	return A;
}

int main()
{

	ARRAY_CLASS B;

	B.Add(1);
	B.Add(2);
	B.Add(3);
	B.Add(4);
	B.Add(5);

	B.Print();

	ARRAY_CLASS A = B;

	A.Print();

	cout<<"A holds address location = "<<A.Get_Address()
	<<" and B holds address location "<<B.Get_Address()<<endl;

	return 0;
}
```

Question 3: Referring to the declaration/initialization statement and the output of the last cout statement in the program of Step 2, what type of copy was implemented? Explain your answer.

Question 4: Why was a copy constructor implemented in the program in Step 2?

Question 5: What is the purpose of a destructor, and why was one implemented in the program in Step 2.

Step 3: In this experiment you will investigate a program that allocates and de-allocates the dynamic memory used by a class. Enter, save, compile and execute the following program in MSVS. Call the new project "CopyConstructorExp3" and the program "CopyConstructor3.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

const int SIZE=5;

class ARRAY_CLASS
{
public:
	ARRAY_CLASS();//default constructor
	~ARRAY_CLASS(); //destructor
	void Add(int); //mutator
	void Print(); //accessor
	int * Get_Address(); //accessor
	void DeAllocate(); //mutator
private:

	int *A;
	int count;
};

ARRAY_CLASS::ARRAY_CLASS()
{
	cout<<"Default constructor has been called\n";
	A = new int[SIZE];
	count = 0;
}
```

```
ARRAY_CLASS::~ARRAY_CLASS()
{
	cout<<"The Destructor has been Called!\n";
	delete [ ] A;
	A=0;
	count = 0;
}

void ARRAY_CLASS::Add(int item)
{
	if (count<SIZE)
		A[count++]=item;
	else
		cout<<"Array Full\n";
}
void ARRAY_CLASS::Print()
{
	for(int i=0; i<count; i++)
		cout<<"A[i] = "<<A[i]<<endl;
}

int * ARRAY_CLASS::Get_Address()
{
	return A;
}

void ARRAY_CLASS::DeAllocate()
{
	delete [ ] A;
	A = 0;
	count = 0;
}

int main()
{

	ARRAY_CLASS B;

	B.Add(1);
	B.Add(2);
	B.Add(3);
	B.Add(4);
	B.Add(5);

	B.Print();

	ARRAY_CLASS A = B;

	cout<<"A holds address location = "<<A.Get_Address()
	    <<" and B holds address location "<<B.Get_Address()<<endl;

	B.DeAllocate();
	A.Print();

	return 0;
}
```

Question 6: What is the purpose of the DeAllocate function in the program of Step 3?

Question 7: What is the difference between the destrunctor and the DeAllocate functions?

Question 8: Does the program in Step 3 execute with errors? If so, name and explain the type(s) of execution error(s) you observed.

Array of Structures

Lab Sections

1. Objectives
2. Introduction
3. Definitions
4. Declaration Syntax
5. Experiments

Array of Structures

1. Objectives

After you complete this experiment you will be able to implement and use an array of structures.

2. Introduction

Arrays of structures are very easy to implement and use once you know how to implement as simple static array of integers. The operators all work the same. However, you must pay close attention to the subtle differences in the syntax needed.

3. Definitions

We will define several terms that you need to know to understand arrays. They are as follows:

1. A structure is a heterogeneous data type.
2. The dot "." operator is used to access the fields/members of a structure.
3. The name of a **static array** is a **constant** pointer to the first element in the array.
4. The **size/capacity** is the number of memory cells allocated to an array.
5. An **index /subscript** is used to access the memory cells in an array.
6. **[ ]** is called the subscript operator.
7. The **index** is a non-negative integer.
8. The **range** of an index is between 0 and the size-1.

4. Declaration Syntax

a. To declare a structure:

```
struct structure_name
{
      field_type_1  field_name_1;
      . . . .
      field_type_n  field_name_n;
};
```

Examples allocating memory for two Dynamic Arrays:

```
struct student_record
{
      string firstname, lastname;
      double age, income;
      int number_of_children;
      char sex;
};
```

b. To declare an array of structures:

structure_name array_of_structures[size];

More information on dynamic arrays can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will learn how to declare and use an array of structures. Enter, save, compile and execute the following program in MSVS. Call the new project "StaticArraysExp1" and the program "StaticArrays1.cpp". Answer the questions that follow.

```
#include <iostream>
#include <string>
using namespace std;

struct student_record
{
	string name;
	double age;
	char sex;
};

int main()
{
	student_record Student_DB[3] = {"Lofton",53,'M',"Thomas",55,'M',
					"Tami",25,'F'};

	cout<<endl;
	for (int i=0; i<3; i++)
	{
		cout<<Student_DB[i].name<<" "
			<<Student_DB[i].age<<" "
			<<Student_DB[i].sex<<endl<<endl;
	}

	Student_DB[0].name = "William";
	Student_DB[2].age = 100;
	Student_DB[1].sex = 'F';

	cout<<"++++++++++++++++++++++++++++++\n";

	cout<<endl;
	for (int i=0; i<3; i++)
	{
		cout<<Student_DB[i].name<<" "
			<<Student_DB[i].age<<" "
			<<Student_DB[i].sex<<endl<<endl;
	}
	return 0;
}
```

Question 1: Please explain how the array “Student_DB” was initialized?

Question 2: What is the purpose of the two “for” loops in the program (StaticArray1s.cpp) in Step 1? Explain your answer.

Step 2: In this experiment you will explain the output of a program that uses a dynamic array of structures. Enter, save, compile and execute the following program in MSVS. Call the new project "StaticArraysExp2" and the program "StaticArrays2.cpp". Answer the questions that follow.

```
#include <iostream>
#include <string>

using namespace std;

struct student_record
{
	string name;
	double age;
	char sex;
};

int main()
{
	student_record *Student_DB = new student_record[3];

	Student_DB[0].name = "Lofton";
	Student_DB[0].age = 53;
	Student_DB[0].sex = 'M';

	Student_DB[1].name = "Thomas";
	Student_DB[1].age = 55;
	Student_DB[1].sex = 'M';

	Student_DB[2].name = "Tami";
	Student_DB[2].age = 25;
	Student_DB[2].sex = 'F';

	cout<<endl;
	for (int i=0; i<3; i++)
	{
		cout<<Student_DB[i].name<<" "
			<<Student_DB[i].age<<" "
			<<Student_DB[i].sex<<endl<<endl;
	}

	Student_DB[0].name = "William";
	Student_DB[2].age = 100;
	Student_DB[1].sex = 'F';

	cout<<"++++++++++++++++++++++++++++++\n";

	cout<<endl;
	for (int i=0; i<3; i++)
	{
		cout<<Student_DB[i].name<<" "
			<<Student_DB[i].age<<" "
			<<Student_DB[i].sex<<endl<<endl;
	}

	delete [ ] Student_DB;
	return 0;
}
```

Question 3: Please state and explain differences in the source code of the program (StaticArrays1.cpp) in Step 1 and the program (StaticArrays2.cpp) in Step 2?

Question 4: What differences between static and dynamic arrays can you conclude from your observations?

Step 3: In this experiment you will investigate the output of a program that uses static and dynamic arrays. Enter, save, compile and execute the following program in MSVS. Call the new project "StaticArraysExp3" and the program "StaticArrays3.cpp". Answer the question that follows.

```
#include <iostream>
using namespace std;

int main()
{
	int static_Array[5];
	int *dynamic_Array;

	dynamic_Array = new int[5];

	int i;

	for(i=0; i<5; i++)
	{
		static_Array[i]=i;
		dynamic_Array[i]=5;
	}


	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	cout<<endl<<endl<<endl;

	dynamic_Array = static_Array;

	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	return 0;
}
```

Question 5: What action(s) does the program (StaticArrays3.cpp) in Step 3 perform?

Step 4: In this experiment you will explain the output of a program that uses static and dynamic arrays. Enter, save, compile and execute the following program in MSVS. Call the new project "StaticArraysExp4" and the program "StaticArrays4.cpp". Answer the question that follows.

```
#include <iostream>
using namespace std;

int main()
{
	int *static_Array = new int[5];
	int *dynamic_Array;

	dynamic_Array = new int[5];

	int i;

	for(i=0; i<5; i++)
	{
		static_Array[i]=i;
		dynamic_Array[i]=5;
	}


	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	cout<<endl<<endl<<endl;

	static_Array = dynamic_Array;

	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	return 0;

}
```

Question 6: Compare the programs that were presented in Steps 1, 2, 3 and 4. What are your observations?

Step 5: In this experiment you will explain the output of a program that uses static and dynamic arrays. Enter, save, compile and execute the following program in MSVS. Call the new project "StaticArraysExp5" and the program "StaticArrays5.cpp". Answer the question that follows.

```
#include <iostream>
using namespace std;

int main()
{
	int static_Array[5];
	int *dynamic_Array;

	dynamic_Array = new int[5];

	int i;

	for(i=0; i<5; i++)
	{
		static_Array[i]=i;
		dynamic_Array[i]=5;
	}


	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	cout<<endl<<endl<<endl;

	static_Array = dynamic_Array;

	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	return 0;

}
```

Question 7: Why does the program in Step 5 produce compilation errors?

Dynamic Arrays

Lab Sections

1. Objectives
2. Introduction
3. Definitions
4. Declaration Syntax
5. Experiments

Dynamic Arrays

1. Objectives

After you complete this experiment you will be able to implement and use dynamic arrays.

2. Introduction

There are two types of arrays in C++, dynamic and static. Dynamic arrays can grow in size while the program is executing, whereas static arrays cannot. Their size must be known before runtime. Because dynamic arrays can grow during runtime, it is unnecessary to estimate their maximum size before compile time. You just grow their size as needed.

3. Definitions

We will define several terms that you need to know to understand arrays. They are as follows:

1. The name of a **dynamic array** is a pointer to the first element in the array.
2. The name of a **static array** is a **constant** pointer to the first element in the array.
3. The **size/capacity** is the number of memory cells allocated to an array.
4. An **index /subscript** is used to access the memory cells in an array.
5. **[ ]** is called the subscript operator.
6. The **index** is a non-negative integer.
7. The **range** of an index is between 0 and the size-1.

4. Declaration Syntax

To declare a dynamic array of type:

```
type  *array_name;  // e.g.  char *ch;
```

Examples allocating memory for two Dynamic Arrays:

```
char *ch = new char[20];

int  *p;
p = new int[5];
```

More information on dynamic arrays can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will explain the output of a program that uses static and dynamic arrays. Enter, save, compile and execute the following program in MSVS. Call the new directory "DynamicArraysExp1" and the program "DynamicArrays1.cpp". Answer the question that follows.

```
#include <iostream>
using namespace std;

int main()
{
	int static_Array[5];
	int *dynamic_Array = new int[5];

	int i;

	for(i=0; i<5; i++)
	{
		static_Array[i]=i;
		dynamic_Array[i]=i;
	}

	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	return 0;
}
```

Question 1: Please explain each line of code and the output.

Step 2: In this experiment you will explain the output of a program that uses static and dynamic arrays. Enter, save, compile and execute the following program in MSVS. Call the new directory "DynamicArraysExp2" and the program "DynamicArrays2.cpp". Answer the question that follows.

```
#include <iostream>
using namespace std;

int main()
{
	int static_Array[5];
	int *dynamic_Array;

	dynamic_Array = new int[5];

	int i;

	for(i=0; i<5; i++)
	{
		static_Array[i]=i;
		dynamic_Array[i]=5;
	}


	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	cout<<endl<<endl<<endl;

	static_Array = dynamic_Array;

	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	return 0;
}
```

Question 2: Please list and explain any error message(s).

Step 3: In this experiment you will investigate the output of a program that uses static and dynamic arrays. Enter, save, compile and execute the following program in MSVS. Call the new directory "DynamicArraysExp3" and the program "DynamicArrays3.cpp". Answer the question that follows.

```
#include <iostream>
using namespace std;

int main()
{
	int static_Array[5];
	int *dynamic_Array;

	dynamic_Array = new int[5];

	int i;

	for(i=0; i<5; i++)
	{
		static_Array[i]=i;
		dynamic_Array[i]=5;
	}


	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	cout<<endl<<endl<<endl;

	dynamic_Array = static_Array;

	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	return 0;
}
```

Question 3: Why did the program in Step 3 compile and run without any errors, and the program in Step 2 did not? Explain your answer.

Step 4: In this experiment you will explain the output of a program that uses static and dynamic arrays. Enter, save, compile and execute the following program in MSVS. Call the new directory "DynamicArraysExp4" and the program "DynamicArrays4.cpp". Answer the question that follows.

```
#include <iostream>
using namespace std;

int main()
{
	int *static_Array = new int[5];
	int *dynamic_Array;

	dynamic_Array = new int[5];

	int i;

	for(i=0; i<5; i++)
	{
		static_Array[i]=i;
		dynamic_Array[i]=5;
	}


	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	cout<<endl<<endl<<endl;

	static_Array = dynamic_Array;

	for (i=0; i<5; i++)
	{
		cout<<"static_Array["<<i<<"] = "<<static_Array[i]<<endl;
		cout<<"dynamic_Array["<<i<<"] = "<<dynamic_Array[i]<<endl;
	}

	return 0;

}
```

Question 4: Compare the programs that were presented in Steps 1, 2, 3 and 4. What are your observations?

Introduction to Classes

Lab Sections

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

Introduction to Classes

1. Objectives

After you complete this experiment you will be able to implement a class.

2. Introduction

Classes encapsulate data and the functions that operate on that data. Classes follow the property "I can do everything for myself".

3. Definitions & Important Terms

We will define several terms you need to know to understand classes. They are as follows:

1. A **private member** is a member that can only be accessed while inside the class (within member functions of the class).
2. A **public member** is a member that can be accessed while inside or outside the class.
3. The **state** refers to the private data members.
4. The **behavior** refers to the pubic member functions.
5. Every class member function has access to the **this** pointer.
6. **Mutators** are member functions that change the state of a class.
7. **Accessors** are member functions that do not change the state of a class.
8. **Constructors** initialize the state of the class. Consider the following characteristics:
    - They have the same name as the class;
    - They have no return type;
    - The **default constructor** has no arguments/formal parameters; a class has only one default constructor;
    - The **explicit-value constructor** has arguments/formal parameters; a class can have many explicit-value constructors;
    - The **copy constructor** is used during a call-by-value, in a return statement and in an initialization/declaration statement.
9. **Destructors** de-allocate dynamic memory allocated by the class using the **new** operator.
10. **Helper functions** are private member functions. This means that they can only be used by member functions of the class.
11. The dot operator, "**.**", is used to access the members of a class.
12. An object is an instance of a class.
13. The scope resolution operator, "**::**", specifies ownership/membership.

4. Declaration Syntax

```
class Class_name
{
     public:
      constructors
      destructor
      member functions
            accessors
            mutators
      public data
    private:
      helper functions
      data
};
```

Example:

```
class Bank_Transaction
{
      public:
         Bank_Transaction( );  //initialize the state
         double Check_Balance( ); //return the dollar amount of balance
         void Deposit(double); //increase balance by a dollar amount
         void Withdrawal(double); //decrease balance by a dollar amount
      private:
         double balance;
};
```

More information on classes can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate the implementation of a class.
Enter, save, compile and execute the following program in MSVS. Call the new project "IntroClassesExp1" and the program "IntroClasses1.cpp". Answer the questions below:

```
#include <iostream>
using namespace std;

class Bank_Acct
{
public:
      Bank_Acct( );
      double Check_Balance( );
      void Deposit(double);
      void Withdrawal(double);
private:
      double balance;
};
```

```
Bank_Acct::Bank_Acct()
{
      balance = 0;
}

double Bank_Acct::Check_Balance()
{
      return balance;
}

void Bank_Acct::Deposit(double amount)
{
      balance = balance + amount;
}

void Bank_Acct::Withdrawal(double amount)
{
      balance = balance - amount;
}

int main()
{
      Bank_Acct my_Acct;

      cout<<"My Account Balance = "<<my_Acct.Check_Balance()<<endl;
      my_Acct.Deposit(2516.83);
      cout<<"My Account Balance = "<<my_Acct.Check_Balance()<<endl;
      my_Acct.Withdrawal(25.96);
      cout<<"My Account Balance = "<<my_Acct.Check_Balance()<<endl;
      return 0;
}
```

Question 1: Please list the elements that make up the state of the class "Bank_Acct" in the program in Step 1?

Question 2: Please list the element(s) that make up the behavior of the class "Bank_ Acct" in the program in Step 1?

Question 3: What kind of member function is Check_Balance in the program in Step 1?

Question 4: What kind of member function is Withdrawal in the program in Step 1?

Question 5: What kind of member function is Deposit in the program in Step 1?

Question 6: What kind of member function is Bank_Acct in the program in Step 1?

Question 7: Can you describe the operation of the dot operation in the program in Step 1?

Question 8: Referring to the first cout statement in the program in Step 1, when was the account balance set to 0? Explain your answer?

Step 2: Enter, save, compile and execute the following program in MSVS. Call the new project "IntroClassesExp2" and the program "IntroClasses2.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

class Bank_Transaction
{
public:
      Bank_Transaction( );  //default constructor
      Bank_Transaction(double);
      double Check_Balance( );
      void Deposit(double);
      void Withdrawal(double);
private:
      double balance;
};


Bank_Transaction::Bank_Transaction()
{
      balance = 0;
}

Bank_Transaction::Bank_Transaction(double amount)
{

      balance = amount;
}
```

```
double Bank_Transaction::Check_Balance()
{
	return balance;
}

void Bank_Transaction::Deposit(double amount)
{
	balance = balance + amount;
}

void Bank_Transaction::Withdrawal(double amount)
{
	balance = balance - amount;
}

int main()
{
	Bank_Transaction my_Acct;
	Bank_Transaction your_Acct(10340.85);

	cout<<"Your Account Balance = "<<your_Acct.Check_Balance()<<endl;
	your_Acct.Deposit(512.30);
	cout<<"Your Account Balance = "<<your_Acct.Check_Balance()<<endl;
	your_Acct.Withdrawal(8284.56);
	cout<<"Your Account Balance = "<<your_Acct.Check_Balance()<<endl;


	cout<<"My Account Balance = "<<my_Acct.Check_Balance()<<endl;
	my_Acct.Deposit(2516.83);
	cout<<"My Account Balance = "<<my_Acct.Check_Balance()<<endl;
	my_Acct.Withdrawal(25.96);
	cout<<"My Account Balance = "<<my_Acct.Check_Balance()<<endl;

	return 0;
}
```

Question 9: Write the statement(ie. the actual line of code) in the program in Step 2 that initializes the balance of the object “your_Acct”.

Question 10: Give the full name of the function(This is also referred to as the function header) and state the type of the constructor that initialized the object “your_Acct”.

Question 11: What happens if you add the statement “my_Acct.Balance = 0;” to the main function of the program in Step 2 after the object declarations? Explain your answer.

Question 12: What do we mean when we use the following phases?

a. Inside the class
b. Outside the class

Inheritance

Lab Sections:

1. Objectives
2. Introduction
3. Definition
4. Declaration Syntax
5. Experiment

Inheritance

1. Objectives

After this experiment you will be able to implement a class that inherits from another class.

2. Introduction

A class is composed of its state (private data) and behavior (public member functions). The public member functions operate on the private data. Symbolically, a class represents a physical object, like a person; the data of a class represents a description (characteristics) of that person; and the functions represent what the person will be able to do (its behavior). Inheritance implies there exists an "**is a**" relationship between two objects. For example, an employee "is a" person. Therefore, all the characteristics that apply to a person also apply to an employee, including any additional characteristics that are unique to an employee.

3. Definitions/Important Terms

We will define several terms you need to know to understand classes. They are as follows:

a. A **base class or parent class** is the class that is being inherited from.
b. The **derived class or child class** is the class that inherits from the base class. All the data and functions defined in the base class are accessible to the derived class.
c. **Protected** class members act as private members for both the base and inherited classes.
d. A **virtual** function is a function in the base class that may be replaced in the derived class. If replaced, the virtual function will provide its own implementation (behavior) for that function.
e. **Virtual functions allows 3 things**: a function in the base class can be replaced in the derived class with the exact same name and signature, it allows the new function in the derived class to have precedence when called, and it allows you to change the new function to suit your needs.

Syntax for a Class Inheritance

```
class Derived_Class : Base_Class
{
    Derived_Class members
};
```

4. Experiment

Step 1: In this experiment you will investigate the implementation of inheritance. Enter, save, compile and execute the following program in MSVS. Call the new directory "InheritanceExp1" and the program "Inheritance1.cpp". Answer the questions below:

```
#include <string>
#include <iostream>

using namespace std;

class Person
{
protected:
    string firstName;
    string lastName;
public:
    Person(void)
    {
          cout<<"In Person's constructor -- A Derived Class"<<endl;
          cout<<"Enter firstname: ";
          cin>>firstName;
          cout<<"Enter lastname: ";
          cin>>lastName;
    }
    string getFirstName(void)
    {
          return firstName;
    }
    string getLastName(void)
    {
          return lastName;
     }
};

class Employee : public Person
{
protected:
    float salary;
public:
    Employee()
     {
          cout<<"In Employee's constructor -- A Derived Class"<<endl;
          cout<<"Enter Salary: ";
          cin>>salary;


     }

    float getSalary(void)
     {
         return salary;
    }
};
```

```
int main (void)
{
   Employee Number_one; //calls Constructor of Person, then Constructor of employee
   Employee Number_two; //calls Constructor of Person, then Constructor of employee

   cout<<endl<<endl<<endl;
   cout<<"Retrieving Number_one's first name and last name from class Person\n";
   cout<<"   "<<Number_one.getFirstName()<<" "<< Number_one.getLastName()<<endl;
   cout<<"Retrieving Number_one's salary from class Employee\n";
   cout<<"   "<<Number_one.getSalary()<<endl;


   cout<<endl<<endl<<endl;
   cout<<"Retrieving Number_two's first name and last name from class Person\n";
   cout<<"   "<<Number_two.getFirstName()<<" "<< Number_two.getLastName()<<endl;
   cout<<"Retrieving Number_two's salary from class Employee\n";
   cout<<"   "<<Number_two.getSalary()<<endl;

   system("PAUSE");
   return 0;
}
```

Question 1: What are the names of the base and derived classes?

Question 2: Describe how the derived class accesses the properties (data and functions) of the base class.

Question 3: Based on the output of the code, which constructor is called first: the constructor in the derived class or the constructor in the base class?

Question 4: What is the purpose of making members of a class protected, as opposed to public or private? (This is a possible job interview question! Get it right and memorize it for life!)

Step 2: In this experiment you will investigate the implementation of inheritance. Enter, save, compile and execute the following program in MSVS. Call the new directory "InheritanceExp2" and the program "Inheritance2.cpp". Answer the questions below:

```
#include <string>
#include <iostream>

using namespace std;

class Person
{
protected:
	string firstName;
	string lastName;
public:
	Person(void)
	{
		cout<<"In Person's constructor -- A Derived Class"<<endl;
		cout<<"Enter firstname: ";
		cin>>firstName;
		cout<<"Enter lastname: ";
		cin>>lastName;
	}
	string getFirstName(void)
	{
		return firstName;
	}
	string getLastName(void)
	{
		return lastName;
	 }

	virtual void CalculateAndPrintPayrollInformation(void)
	{
		cout<<"This is Payroll Information for a Base Object"<<endl;
		cout<<"Gross Pay is 0"<<endl;
		cout<<"Income Tax is 0"<<endl;
		cout<<"Net Pay is 0"<<endl<<endl;
	}
};

class Employee : public Person
{
protected:
	float annual_salary;
	double gross_pay, hours_worked, hourly_rate, income_tax, net_pay;
public:
    Employee()
    {
		cout<<"In Employee's constructor -- A Derived Class"<<endl;
		cout<<"Enter Annual Salary: ";
		cin>>annual_salary;


    }
```

```
    float getSalary(void)
    {
        return annual_salary;
    }

      void CalculateAndPrintPayrollInformation(void)
      {
            cout<<"In Taxpayer's constructor -- A derived class of employe"<<endl;
            cout<<"Enter hours worked: ";
            cin>>hours_worked;
            cout<<"Enter hourly rate: ";
            cin>>hourly_rate;
            gross_pay = hours_worked * hourly_rate;
            income_tax = gross_pay * 0.25;
            net_pay = gross_pay - income_tax;
            cout<<"Gross Pay = "<<gross_pay<<endl;
            cout<<"Income Tax  = "<<income_tax<<endl;
            cout<<"Net Pay = "<<net_pay<<endl;
      }

};

int main (void)
{
      Person John;
      Employee Mary;

      cout<<endl<<endl<<endl;
      cout<<"Enter John's information\n";
      cout<<endl<<John.getFirstName()<<" "<<John.getLastName()<<endl;
      John.CalculateAndPrintPayrollInformation();

      cout<<endl<<endl<<endl;
      cout<<"Enter Mary's information\n";
      cout<<endl<<Mary.getFirstName()<<" "<<Mary.getLastName()<<" salary is "
            << Mary.getSalary()<<endl;
      Mary.CalculateAndPrintPayrollInformation();

      system("PAUSE");
      return 0;
}
```

Question 5: What are the code differences between the programs presented in Step 1 and Step 2? (Hint: it's a new virtual function)

Question 6: Referring to the output of the program in Step 2, how is the virtual function used in the Person and Employee classes?

Question 7: Change the type of John from Person to Employee in the main function of the program in Step 2. Execute this modified program and compare the output with the output of the original program in Step 2. Based on the 2 different outputs, do you think the original **base** class implementation of the virtual function is executed in this modified program? Why or why not?

Question 8: What is the purpose of the "virtual" keyword?

Classes:
Copy Constructor

Lab Sections

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

Classes: Copy Constructor

1. Objectives

After you complete this experiment you will be able to implement a copy constructor.

2. Introduction

A copy constructor is called to make a copy of an object. The copy constructor is automatically called during one of the following events:

a. when an object is passed-by-value;
b. when an object is returned in a return statement;
c. when an object appears in a declaration/initialization statement.

3. Definitions & Important Terms

We will define several terms you need to understand before implementing copy constructors. They are as follows:

a. A **shallow copy** is a **dependent bit-by-bit** copy of an object. The objects involved will share the same dynamic memory;
b. A **deep copy** is an **independent** copy of an object. The objects involved will have their own copy of dynamic memory;
c. The **copy constructor** is used to make a deep copy of an object;
d. When an object does not use dynamic memory a copy constructor does not need to be implemented;
e. If you do not implement a copy constructor, one will be implemented for you. However, this copy constructor will perform a shallow copy of an object;
f. Whenever dynamic memory is allocated using the new function of a class, a **destructor** must be implemented to de-allocate that memory; it has the same name as the class, no formal parameters, no return type and is prefaced with a tilde character (~).
g. The **current object** is the object that called the copy constructor;
h. **this** is a pointer to the current object;
i. The **object that is being copied (source object) is passed explicitly** through the corresponding formal parameter of the copy constructor;
j. The **object that is being copied into (destination/current object) is passed implicitly through the "this" pointer**.

4. Declaration Syntax

Consider the following:

a. Class_name(const Class_name &); //prototype inside class declaration
b. Class_name::Class_name(const Class_name & Formal_parameter) //functon header

Notice that the copy constructor does not have a return type and has one constant formal parameter passed-by-reference of the same type as the class.

More information on the copy constructor can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate how an object is copied.

Enter, save, compile and execute the following program in MSVS. Call the new project "CopyConstructorExp1" and the program "CopyConstructor1.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

const int SIZE=5;

class ARRAY_CLASS
{
public:
	ARRAY_CLASS();//default constructor
	void Add(int); //mutator
	void Print(); //accessor
	int * Get_Address(); //accessor
private:

	int *A;
	int count;
};

ARRAY_CLASS::ARRAY_CLASS()
{
	cout<<"Default Constructor has been Called!\n";
	A = new int[SIZE];
	count = 0;
}

void ARRAY_CLASS::Add(int item)
{
	if (count<SIZE)
		A[count++]=item;
	else
		cout<<"Array Full\n";
}

void ARRAY_CLASS::Print()
{
	for(int i=0; i<count; i++)
		cout<<"A[i] = "<<A[i]<<endl;
}
```

```
int * ARRAY_CLASS::Get_Address()
{
	return A;
}

int main()
{
	ARRAY_CLASS B;

	B.Add(1);
	B.Add(2);
	B.Add(3);
	B.Add(4);
	B.Add(5);
	B.Print();

	cout<<endl<<endl;

	ARRAY_CLASS A = B;

	A.Print();

	cout<<"A holds address location = "<<A.Get_Address()
	<<" and B holds address location "<<B.Get_Address()<<endl;

	return 0;
}
```

Question 1: Referring to the declaration/initialization statement and the output of the last cout statement in the main function of the program of Step 1, what type of copy was implementated, deep or shallow.? Explain your answer.

Question 2: Was a copy constructor implemented in the program in Step 1? Explain you answer.

Step 2: In this experiment you will investigate how a copy constructor is implemented. Enter, save, compile and execute the following program in MSVS. Call the new project "CopyConstructorExp2" and the program "CopyConstructor2.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

const int SIZE=5;

class ARRAY_CLASS
{
public:
	ARRAY_CLASS();//default constructor
	ARRAY_CLASS(const ARRAY_CLASS &);//copy constructor
	~ARRAY_CLASS(); //destructor
	void Add(int); //mutator
	void Print(); //accessor
	int * Get_Address(); //accessor
private:

	int *A;
	int count;
};

ARRAY_CLASS::ARRAY_CLASS()
{
	cout<<"The Default Constructor has been Called!\n";
	A = new int[SIZE];
	count = 0;
}

ARRAY_CLASS::ARRAY_CLASS(const ARRAY_CLASS & Org)
{
	cout<<"The Copy Constructor has been Called!\n";

	count = Org.count;

	A = new int[SIZE];

	for(int i=0; i<count; i++)
	{
		A[i] = Org.A[i];
	}
}

ARRAY_CLASS::~ARRAY_CLASS()
{
	cout<<"The Destructor has been Called!\n";
	delete [ ] A;
	A=0;
	count = 0;
}
```

```
void ARRAY_CLASS::Add(int item)
{
	if (count<SIZE)
		A[count++]=item;
	else
		cout<<"Array Full\n";
}


void ARRAY_CLASS::Print()
{
	for(int i=0; i<count; i++)
		cout<<"A[i] = "<<A[i]<<endl;
}

int * ARRAY_CLASS::Get_Address()
{
	return A;
}

int main()
{

	ARRAY_CLASS B;

	B.Add(1);
	B.Add(2);
	B.Add(3);
	B.Add(4);
	B.Add(5);

	B.Print();

	ARRAY_CLASS A = B;

	A.Print();

	cout<<"A holds address location = "<<A.Get_Address()
	<<" and B holds address location "<<B.Get_Address()<<endl;

	return 0;
}
```

Question 3: Referring to the declaration/initialization statement and the output of the last cout statement in the program of Step 2, what type of copy was implemented? Explain your answer.

Question 4: Why was a copy constructor implemented in the program in Step 2?

Question 5: What is the purpose of a destructor, and why was one implemented in the program in Step 2.

Step 3: In this experiment you will investigate a program that allocates and de-allocates the dynamic memory used by a class. Enter, save, compile and execute the following program in MSVS. Call the new project "CopyConstructorExp3" and the program "CopyConstructor3.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

const int SIZE=5;

class ARRAY_CLASS
{
public:
	ARRAY_CLASS();//default constructor
	~ARRAY_CLASS(); //destructor
	void Add(int); //mutator
	void Print(); //accessor
	int * Get_Address(); //accessor
	void DeAllocate(); //mutator
private:

	int *A;
	int count;
};

ARRAY_CLASS::ARRAY_CLASS()
{
	cout<<"Default constructor has been called\n";
	A = new int[SIZE];
	count = 0;
}
```

```
ARRAY_CLASS::~ARRAY_CLASS()
{
	cout<<"The Destructor has been Called!\n";
	delete [ ] A;
	A=0;
	count = 0;
}

void ARRAY_CLASS::Add(int item)
{
	if (count<SIZE)
		A[count++]=item;
	else
		cout<<"Array Full\n";
}
void ARRAY_CLASS::Print()
{
	for(int i=0; i<count; i++)
		cout<<"A[i] = "<<A[i]<<endl;
}

int * ARRAY_CLASS::Get_Address()
{
	return A;
}

void ARRAY_CLASS::DeAllocate()
{
	delete [ ] A;
	A = 0;
	count = 0;
}

int main()
{

	ARRAY_CLASS B;

	B.Add(1);
	B.Add(2);
	B.Add(3);
	B.Add(4);
	B.Add(5);

	B.Print();

	ARRAY_CLASS A = B;

	cout<<"A holds address location = "<<A.Get_Address()
	    <<" and B holds address location "<<B.Get_Address()<<endl;

	B.DeAllocate();
	A.Print();

	return 0;
}
```

Question 6: What is the purpose of the DeAllocate function in the program of Step 3?

Question 7: What is the difference between the destrunctor and the DeAllocate functions?

Question 8: Does the program in Step 3 execute with errors? If so, name and explain the type(s) of execution error(s) you observed.

Linked List

Estimated Time to Complete

60 minutes

Lab Sections

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

Linked List

1. Objectives

After you complete this experiment you will be able to implement a singly-linked list with a header node.

2. Introduction

A linked list is a dynamic structure. Remember, this means that the size of a linked list can change during runtime.

3. Definitions & Important Terms

We will define several terms you need to understand before implementing linked lists. They are as follows:

a. A **linked list** is composed of **nodes**.
b. **Nodes** are composed of a data element(s) and a pointer element.
c. The nodes are linked together using **pointers**.
d. A **pointer** to the front of the list is needed to access the list.
e. Lists are accessed **sequentially**.
f. The end of the list is represented with a null or 0.
g. There are several generic operators available to all lists. They include: **constructor, destructor, insert, remove, search** and **print**.
h. A **singly-linked list** is a list in which each node has only one pointer field pointing to its **successor node** in the list.

4. Declaration Syntax

Consider the following declarations that you will be using for the experiments:

```
// declaration for a node in the list
class LIST_NODE
{
public:
	int data; // data element
	LIST_NODE *next; // pointer element
};

//declaration of a list class
class LINKED_LIST_CLASS
{
public:
	LINKED_LIST_CLASS();// default constructor
	void Print(); // accessor
	bool Is_Empty(); // accessor
private:
	LIST_NODE *front;

};
```

Notice the data and pointer elements shown in the LIST_NODE declaration, and that a LINKED_LIST_CLASS has only one private element, a pointer called **front**.

More information on the linked lists can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate the implementation of a singly-linked list with a header node. Enter, save, compile and execute the following program in MSVS. Call the new project "LinkedListExp1" and the program "LinkedList1.cpp". Answer the questions below:

```
#include <iostream>
using namespace std;


class LIST_NODE
{
public:
	int data; // data element of node
	LIST_NODE *next; // pointer element of node
};

class LINKED_LIST_CLASS
{
public:
	LINKED_LIST_CLASS(); // default constructor
	~LINKED_LIST_CLASS();  // destructor
	void Add(int); // mutator
	void Print(); // accessor
```

```
private:

	LIST_NODE *front; // pointer to front of list (header node)

};

LINKED_LIST_CLASS::LINKED_LIST_CLASS()
{
	cout << endl << "Default constructor has been called.\n";
	front = new LIST_NODE;
	front->next = 0;  // initialize next field to null
	front->data = -10000;
}

LINKED_LIST_CLASS::~LINKED_LIST_CLASS()
{
	cout << endl << "The destructor has been called.\n";

	while (front->next != 0)
	{
		LIST_NODE *p = front->next;
		front->next = front->next->next;
		delete p;
	}
	delete front;
	front = 0;
}

void LINKED_LIST_CLASS::Add(int item)
{
	LIST_NODE *p = new LIST_NODE;

	p->data = item;

	if (front->next == 0) // empty list
	{
		front->next = p;
		p->next = 0;
	}
	else // list has information and is not empty
	{
		p->next = front->next;
		front->next = p;
	}
}

void LINKED_LIST_CLASS::Print()
{
	cout << endl;
	for(LIST_NODE *p = front->next; p != 0; p = p->next)
	{
		cout << p->data;
		if (p->next != 0)
		{
			cout << "-->";
		}
	}
	cout << endl << endl;
}


	int main ()
```

```
{
	LINKED_LIST_CLASS sample1;

	sample1.Add(4);
	sample1.Add(5);
	sample1.Add(6);
	sample1.Add(4);
	sample1.Add(7);
	sample1.Print();

	return 0;
}
```

Question 1: What is the output of the program in Step 1? Please show the values.

Question 2: Please draw an empty list.

Question 3: Please draw the list that is printed by the program in Step 1.

Question 4: Is the front of the list ever equal to 0 or null? Please explain your answer.

Question 5: What are the contents of the header node?

Question 6: Please list the order in which nodes are removed from the list by the destructor?

Step 2: In this experiment you will investigate the implementation of a singly-linked list with a header node. Enter, save, compile and execute the following program in MSVS. Call the new project "LinkedListExp2" and the program "LinkedList2.cpp". Answer the questions below:

```
#include <iostream>
using namespace std;

class LIST_NODE
{
public:
	int data; // data element of node
	LIST_NODE *next; // pointer element of node
};

class LINKED_LIST_CLASS
{
public:
	LINKED_LIST_CLASS(); // default constructor
	LINKED_LIST_CLASS(LINKED_LIST_CLASS &); // copy constructor
	~LINKED_LIST_CLASS(); // destructor
	void Add(int); // mutator
	void Print(); // accessor
private:

	LIST_NODE *front; // pointer to front of list (header node)

};

LINKED_LIST_CLASS::LINKED_LIST_CLASS()
{
	cout << endl << "The default constructor has been called.\n";
	front = new LIST_NODE;
	front->next = 0;  //initialize next field to null
	front->data = -10000;
}

LINKED_LIST_CLASS::LINKED_LIST_CLASS(LINKED_LIST_CLASS & org)
{
	cout << endl << "The copy constructor has been called.\n";

	front = new LIST_NODE;

	front->next = 0;
	front->data = -10000;

	LIST_NODE *p = org.front->next;
	LIST_NODE *back = 0;

	while(p!=0)
	{
		if (back == 0)
		{
			front->next = new LIST_NODE;
			back = front->next;
			back->next = 0;
			back->data = p->data;
		}
		else
```

```
		{
			back->next = new LIST_NODE;
			back = back->next;
			back->data = p->data;
			back->next = 0;
		}
		p=p->next;
	}

}

LINKED_LIST_CLASS::~LINKED_LIST_CLASS()
{
	cout << endl << "The destructor has been called.\n";

	while (front->next != 0)
	{
		LIST_NODE *p = front->next;
		front->next = front->next->next;
		delete p;
	}
	delete front;
	front = 0;
}

void LINKED_LIST_CLASS::Add(int item)
{
	LIST_NODE *p = new LIST_NODE;

	p->data = item;

	if (front->next== 0) // empty list
	{
		front->next = p;
		p->next = 0;
	}
	else // list has information and is not empty
	{
		p->next = front->next;
		front->next = p
;
	}
}

void LINKED_LIST_CLASS::Print()
{
	cout << endl;
	for(LIST_NODE *p = front->next; p != 0; p = p->next)
	{
		cout<<p->data;
		if (p->next != 0)
		{
			cout << "-->";
		}
	}
	cout << endl << endl;
}

int main()
{
	LINKED_LIST_CLASS L1;
```

```
        L1.Add(5);
        L1.Add(10);
        L1.Add(29);

        L1.Print();

        LINKED_LIST_CLASS L2 = L1;

        L2.Print();

        return 0;
}
```

Question 7: What is the output of the program in Step 2?

Question 8: Please draw the list object, L2, for the program in Step 2.

Question 9: What is the purpose of the back pointer in the copy constructor for the program in Step 2?

Question 10: Is the front of L2 ever equal to 0 or null? Please explain your answer.

Question 11: What are the contents of the header node for L2?

Step 3: In this experiment you will investigate the implementation of a singly-linked list with a header node. Enter, save, compile and execute the following program in MSVS. Call the new project "LinkedListExp3" and the program "LinkedList3.cpp". Answer the questions below:

```
#include <iostream>
using namespace std;

class LIST_NODE
{
public:
	int data; // data element of node
	LIST_NODE *next; // pointer element of node
};

class LINKED_LIST_CLASS
{
public:
	LINKED_LIST_CLASS(); // default constructor
	LINKED_LIST_CLASS(LINKED_LIST_CLASS &); // copy constructor
	~LINKED_LIST_CLASS(); // destructor
	void Add(int); // mutator
	void Print(); // accessor
	LIST_NODE * Search(int); // accessor
	void Remove(int); // mutator
	bool Is_Empty(); // accessor
private:
	LIST_NODE *front; // pointer to front of list

};

LINKED_LIST_CLASS::LINKED_LIST_CLASS()
{
	cout << endl << "The default constructor has been called.\n";
	front = new LIST_NODE;
	front->next = 0;  // initialize the next field to null
	front->data = -10000;
}

LINKED_LIST_CLASS::LINKED_LIST_CLASS(LINKED_LIST_CLASS & org)
{
	cout << endl << "The copy constructor has been called.\n";

	front = new LIST_NODE;

	front->next = 0;
	front->data = -10000;

	LIST_NODE *p = org.front->next;
	LIST_NODE *back = 0;

	while(p!=0)
	{
		if (back == 0)
		{
			front->next = new LIST_NODE;
			back = front->next;
			back->next = 0;
			back->data = p->data;
		}
```

```
		else
		{
			back->next = new LIST_NODE;
			back = back->next;
			back->data = p->data;
			back->next = 0;
		}
		p=p->next;
	}

}

LINKED_LIST_CLASS::~LINKED_LIST_CLASS()
{
	cout << endl << "The destructor has been called.\n";

	while (front->next != 0)
	{
		LIST_NODE *p = front->next;
		front->next = front->next->next;
		delete p;
	}
	delete front;
	front = 0;
}

void LINKED_LIST_CLASS::Add(int item)
{
	LIST_NODE *p = new LIST_NODE;

	p->data = item;

	if (front->next== 0) // empty list
	{
		front->next = p;
		p->next = 0;
	}
	else // list has information and is not empty
	{
		p->next = front->next;
		front->next = p;
	}
}

void LINKED_LIST_CLASS::Print()
{
	cout << endl;
	for(LIST_NODE *p = front->next; p != 0; p = p->next)
	{
		cout << p->data;
		if (p->next != 0)
		{
			cout << "-->";
		}
	}
	cout<<endl<<endl;
}



LIST_NODE * LINKED_LIST_CLASS::Search(int key)
```

```
{
	for(LIST_NODE *p = front->next; p!=0; p=p->next)
	{
		if (p->data == key)
			return p;
	}
	return 0; // key not found in list
}

void LINKED_LIST_CLASS::Remove(int key)
{
	LIST_NODE *p = Search(key);

	if (Is_Empty())
	{
		cout << key << " is not in the list. No removal performed!\n";
	}
	else
	{
		LIST_NODE *q = front;

		while (q->next->data != key)
		{
			q = q->next;
		}

		q->next = p->next; // CRITICAL STEP!!!!
		delete p;
	}

}

bool LINKED_LIST_CLASS::Is_Empty()
{
	return front->next == 0;
}

int main()
{
	LINKED_LIST_CLASS L1;

	L1.Add(5);
	L1.Add(10);
	L1.Add(29);

	L1.Print();

	LINKED_LIST_CLASS L2 = L1;

	L2.Print();

	L1.Remove(10);
	L1.Print();

	return 0;
}
```

Question 12: What is the output of the program in Step 3?

Question 13: Referring to the program in Step 3, please explain the order in which nodes are removed from a list.

Question 14: Is the **Search()** function called by any member functions of the linked list class for the program in Step 3?

Question 15: Please add a statement to the program in Step 3 that tests the **Search()** function. What did you add?

Question 16: Please add pre-condition, post-conditions and descriptions for each member function of the LINKED_LIST_CLASS. Answer with your new version of the program.

Member Operator Overloading

Lab Sections

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

Member Operator Overloading

1. Objectives

After you complete this experiment you will be able to overload an operator as a member function of a class.

2. Introduction

When you overload an operator you provide your personal implementation for the operator. It comes in handy when you create a new object and you need to refine the implementation of an existing operator to match your needs. Overloading operators is really effective when the original meaning of the operator can be ported to the new object. For example, the "+" operator should mean some type of addition will take place. For strings and chars addition is concatenation; for doubles and ints it is numeric summation and for a new type of array class it may be the summation of all the corresponding elements in two adjacent arrays. Overloading C++ operators is an optional topic. However, we think it is important that you understand how to implement operator overloading, so we will cover it.

3. Definitions & Important Terms

We will define several terms you need to know to understand classes. They are as follows:

a. The **arity** of an operator is the number of parameters (operands) it requires. The arity of an operator cannot change.
b. A **non-member** function of a class does not have access to the state (private area) of a class.
c. **Chaining** occurs when a C++ statement contains several instances of the same overloaded operator.
d. The **current object** is the object that performed the function invocation.
e. **"this"** is a pointer to the current object.
f. **All member functions of a class have access to the "this" pointer.**
g. **Non-member and friend functions of a class do not have access to the "this" pointer.**
h. **"*this"** is the current object.
i. When an object is passed **"implicitly"** it is passed through the "**this" pointer**.
j. When an object is passed **"explicitly"** it is passed through its corresponding formal parameter.

4. Declaration Syntax

Notice that the non-member function does not have a prototype in the class declaration and the syntax for the implementation of the non-member function body in the following code:

```
class Class_name
{
public:
      constructors
      destructor
      member functions
            accessors
            mutators
      public data
private:
      helper functions
      data
};
--------------

return_type Class_name::function_name(formal parameter list)
{
      body
}
```

More information on classes can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate the implementation for overloading the operator "+" as a member function of a class with chaining. Enter, save, compile and execute the following program in MSVS. Call the new project "MemberOpOverloadingExp" and the program "MemberOpOverloading.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

const int SIZE = 10;

class Bank_Acct
{
public:
      Bank_Acct( );  //default constructor
      Bank_Acct(double new_balance, string Cname); //explicit value
                                                   //constructor
      void Print( ); //accessor function
      Bank_Acct & operator+(double amount); //mutator function

private:
      double balance;
      string name;
};
```

```
Bank_Acct::Bank_Acct()
{
	balance = 0;
	name = "NoName";
}

Bank_Acct::Bank_Acct(double amount, string Cname)
{
	balance = amount;
	name = Cname;

}

void Bank_Acct::Print()
{
	cout<<endl<<"Object "<<name;
	cout<<endl<<"The new balance is "<<balance<<endl;
}

Bank_Acct & Bank_Acct::operator+(double amount)
{
	balance += amount;
	return *this;
}

int main()
{
	Bank_Acct my_Acct;

	cout.setf(ios::showpoint);
	cout.setf(ios::fixed);
	cout.precision(2);

	cout<<"Original balance of my_Acct"<<endl;
	my_Acct.Print( );

	//the following statement contains chaining
	my_Acct + 18.75 + 14.35 + 10054.96;

	cout<<"The balance of my_Acct after addition to balance 3 times"<<endl;
	my_Acct.Print();

	return 0;
}
```

Question 1: Referring to the program in Step 1, if the arity of the "+" operator is two, why is there only one formal parameter in operator+'s function header?

Question 2: What is the return type of the operator+ function?

Question 3: Referring to the operator+ function, what is the name of the Bank_Acct object it returned?

Question 4: Why is there a "Bank_Acct::" prefixed to the header of the operator+ function?

Question 5: Can you explain how chaining is performed for the operator+ function in the program in Step 1.

Question 6: Please explain why the operator "<<" cannot be overloaded as a member function? (hint: modify the code that was used in the laboratory "Classes: Part 2" to overload the "<<" operator as a friend member function.)

Friend Operator Overloading

Lab Sections

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

Friend Operator Overloading

1. Objectives

After you complete this experiment you will be able to overload an operator as a friend function of a class.

2. Introduction

When you overload an operator you provide your personal implementation for the operator. It comes in handy when you create a new object and you need to refine the implementation of an existing operator to match your needs. Overloading operators is really effective when the original meaning of the operator can be ported to the new object. For example, the "+" operator should mean some type of addition will take place. For strings and chars addition is concatenation; for doubles and ints it is numeric summation and for a new type of array class it may be the summation of all the corresponding elements in two adjacent arrays. Overloading C++ operators is an optional topic. However, we think it is important that you understand how to implement operator overloading, so we will cover it.

3. Definitions & Important Terms

We will define several terms you need to know to understand classes. They are as follows:

a. The **arity** of an operator is the number of parameters (operands) it requires. The arity of an operator cannot change.
b. A **friend** is a **non-member** function that has access to the state (private area) of a class. The class must give the function permission to be its friend.
c. **Chaining** occurs when a C++ statement contains several instances of the same overloaded operator.

4. Declaration Syntax

```
friend return_type function_name(formal parameter list);
//place this form of a prototype in your class declaration
```

Notice the location of the friend statement in the class declaration and the syntax for the implementation of the friend function body in the following code:

```
class Class_name
{
public:
      constructors
      destructor
      member functions
            accessors
            mutators
      friend return_type function_name(formal parameter list);_
      public data
```

```
private:
      helper functions
      data
};

--------------------------------

return_type function_name(formal parameter list)
{
      body
}
```

More information on classes can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate the implementation for overloading the operator "<<" as a friend function with chaining. Enter, save, compile and execute the following program in MSVS. Call the new project "FriendOpOverloadingExp1" and the program "FriendOpOverloading1.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

const int SIZE = 10;

class Bank_Acct
{
public:
      Bank_Acct( );  //default constructor
      Bank_Acct(double new_balance, string Cname); //explicit value
                                                   //constructor

        //overloading operator<< as a friend function with chaining
      friend ostream & operator<<(ostream & output, Bank_Acct & Org);
private:
      double balance;
      string name;
};

Bank_Acct::Bank_Acct()
{
      balance = 0;
      name = "NoName";
}

Bank_Acct::Bank_Acct(double amount, string Cname)
{
      balance = amount;
      name = Cname;

}

ostream & operator<<(ostream & output, Bank_Acct & Org)
```

```
{
	output<<endl<<"Object "<<Org.name;
	output<<endl<<"The new balance is "<<Org.balance<<endl;
	return output;
}

int main()
{
	Bank_Acct my_Acct;

	Bank_Acct DrB(2000.87, "Dr. Bullard");

	//the following statement contains chaining
	cout<<DrB<<endl<<my_Acct<<endl;

	return 0;
}
```

Question 1: Please explain how chaining works? (hint: Look at the cout statement in main)

Question 2: What is the arity of the "<<" operator?

Question 3: What variables are initialized by the explicit-value constructor? What values are the variables initialized to?

Question 4: What is the return type of the operator<< function?

Question 5: Why isn't there a "Bank_Acct::" prefixed to the header of the operator<< function?

Step 2: In this experiment you will investigate the implementation for overloading the operator "<<" as a friend function without chaining. Enter, save, compile and execute the following

program in MSVS. Call the new project "FriendOpOverloadingExp2" and the program "FriendOpOverloading2.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

const int SIZE = 10;

class Bank_Acct
{
public:
	Bank_Acct( );  //default constructor
	Bank_Acct(double new_balance, string Cname); //explicit value
	//constructor overloading operator<< as a friend function with chaining

	friend void operator<<(ostream & output, Bank_Acct & Org);
private:
	double balance;
	string name;
};

Bank_Acct::Bank_Acct()
{
	balance = 0;
	name = "NoName";
}

Bank_Acct::Bank_Acct(double amount, string Cname)
{
	balance = amount;
	name = Cname;

}

void operator<<(ostream & output, Bank_Acct & Org)
{
	output<<endl<<"Object "<<Org.name;
	output<<endl<<"The new balance is "<<Org.balance<<endl;

}

int main()
{
	Bank_Acct my_Acct;

	Bank_Acct DrB(2000.87, "Dr. Bullard");

	//the following statement does not contain chaining
	cout<<DrB;

	return 0;
}
```

Question 6: Did the program in Step 2 execute without any errors? Make a statement about chaining and this program.

Question 7: Replace the cout statement in the main function of the program in Step 2 with the following statement:

```
cout<<DrB<<endl<<my_Acct<<endl;
```

Are there any errors when you try to compile the program? Explain your observations.

Question 8: Referring to the program in Step 1, please re-write new cout statements representing the one cout statement in the main function that was used in chaining.

Question 9: Please remove the cout statement in the program in Step 2 and replace it with the cout statements you wrote for Queston 8. Execute the new program and explain (if any) errors that may have occured.

Non-Member Operator Overloading

Lab Sections

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

Non-member Operator Overloading

1. Objectives

After you complete this experiment you will be able to overload an operator as a non-member function of a class.

2. Introduction

When you overload an operator you provide your personal implementation for the operator. It comes in handy when you create a new object and you need to refine the implementation of an existing operator to match your needs. Overloading operators is really effective when the original meaning of the operator can be ported to the new object. For example, the "+" operator should mean some type of addition will take place. For strings and chars addition is concatenation; for doubles and ints it is numeric summation and for a new type of array class it may be the summation of all the corresponding elements in two adjacent arrays. Overloading C++ operators is an optional topic. However, we think it is important that you understand how to implement operator overloading, so we will cover it.

3. Definitions & Important Terms

We will define several terms you need to understand in order to implement operator overloading as a non-member function. They are as follows:

a. The **arity** of an operator is the number of parameters (operands) it requires. The arity of an operator cannot change.
b. A **non-member** function of a class does not have access to the state (private area) of a class.
c. **Chaining** occurs when a C++ statement contains several instances of the same overloaded operator.

4. Declaration Syntax

Notice the location of the non-member function of the class in the class declaration and the syntax for the implementation of the non-member function body in the following code:

```
class Class_name
{
public:
      constructors
      destructor
      member functions
            accessors
            mutators
      public data
private:
      helper functions
      data
};
---------------------
```

```
return_type function_name(formal parameter list)
{
      body
}
```

More information on classes can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate the implementation for overloading the operator "<<" as a non-member function of a class with chaining. Enter, save, compile and execute the following program in MSVS. Call the new project "NonMemberOpOverloadingExp" and the program "NonMemberOpOverloading.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

const int SIZE = 10;

class Bank_Acct
{
public:
      Bank_Acct( );  //default constructor
      Bank_Acct(double new_balance, string Cname); //explicit value
                                                   //constructor
      void Print(ostream & out); //accessor function

private:
      double balance;
      string name;
};

Bank_Acct::Bank_Acct()
{
      balance = 0;
      name = "NoName";
}

Bank_Acct::Bank_Acct(double amount, string Cname)
{
      balance = amount;
      name = Cname;

}

void Bank_Acct::Print(ostream & output)
{
      output<<endl<<"Object "<<name;
      output<<endl<<"The new balance is "<<balance<<endl;
}
```

```
ostream & operator<<(ostream & output, Bank_Acct & Org)
```

```
{
	Org.Print(output);
	return output;
}



int main()
{
	Bank_Acct my_Acct;

	Bank_Acct DrB(2000.87, "Dr. Bullard");

	//the following statement contains chaining
	cout<<DrB<<endl<<my_Acct<<endl;

	return 0;
}
```

Question 1: Please explain how chaining works? (hint: look at the cout statement in main)

Question 2: What is the arity of the "<<" operator in the program?

Question 3: What is the return type of the operator<< function in the program?

Question 4: What is the purpose of the Print function in the program?

Question 5: Why isn't there a "Bank_Acct::" prefixed to the header of the operator<< function?

Question 6: What are the similarities and differences you observed in overloading the operator<< as a friend function and as a non-member function of a class? **[Answer after completing the Friend Operator Overloading lab.]**

Function Templates

Lab Sections

1. Objectives
2. Introduction
3. Declaration Syntax
4. Experiments

Function Templates

1. Objectives

After you complete this experiment you will be able to implement templates for functions and know when it is appropriate to implement templates for functions

2. Introduction

Templates allow functions to use and operate on generic data types. They provide a mechanism to parameterize a data type. Whenever you notice that two or more functions have identical code but operate on data of different types, you should consider templates if possible.

3. Declaration Syntax

a. To declare a class template with one type parameter:

```
template <class Type_Parameter>
    function declaration
   //temple <class Type_Parameter> is referred to as the "template prefix" which
   //informs the compiler that "Type_Parameter" is a type parameter.
```

b. To declare a class template with one type parameter (alternative form):

```
template <typename Type_Parameter>
   function declaration
   //This declaration operates identically to the first declaration.
```

c. To declare a class template with multiple type parameters:

```
template <class Type_Parameter1, class Type_Parameter2, ..., class Type_Parameter)
    function declaration
    //This declaration contains several type parameters
```

Consider the following **function template** declaration for a function that checks to see if two items have the same value:

```
template <class a_type>
bool Is_Equal(a_type a, a_type b)
{
      return a == b;
}
```

This function contains the type parameter "a_type" which represents a type that has not been specified.

To call the function Is_Equal consider the following:

```
string x="XXXX", y="YYYY";

cout<<Is_Equal(x,y)<<endl;
```

More information on templates can be found in your course textbook and on the web.

4. Experiments

Step 1: In this experiment you will investigate a program that contains two functions which are perfect candidates to be implemented as a function template. Enter, save, compile and execute the following program in MSVS. Call the new project "FunctionTemplateExp" and the program "FunctionTemplate.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

void swap(int &a, int &b)
{
	int temp = a;
	a = b;
	b = temp;
}

void swap(string &a, string &b)
{
	string temp = a;
	a = b;
	b = temp;
}

void swap(char &a, char&b)
{
	char temp = a;
	a = b;
	b = temp;
}

int main( )
{
	string x = "first", y = "second";
	int m = 10, n = 20;
	char q = 'Q', r = 'R';

	cout<<"x before swap called = "<<x<<" and y before swap called = "<<y<<endl;
	swap(x,y);
	cout<<"x after swap called = "<<x<<" and y after swap called = "<<y<<endl<<endl;

	cout<<"m before swap called = "<<m<<" and n before swap called = "<<n<<endl;
	swap(m,n);
	cout<<"m after swap called = "<<m<<" and n after swap called = "<<n<<endl<<endl;

	cout<<"q before swap called = "<<q<<" and r before swap called = "<<r<<endl;
	swap(q,r);
	cout<<"q after swap called = "<<q<<" and r after swap called = "<<r<<endl<<endl;

	return 0;
}
```

Question 1: Please explain the output of the program in Step 1?

Question 2: Please examine the three different swap functions in Step 1, and state any similarities and/or differences you observe?

Question 3: Write a function template declaration for a function with the following characteristics:

a. the function name is “Tester3”;
b. the function return type is character;
c. there are two template parameters, a_type and b_type;
d. there are four formal parameters: x (a_type), y (b_type), w (b_type) and m (b_type).

Question 4: Write a function template declaration replacing the three different swap functions in Step 1:

Class Templates

Lab Sections

1. Objectives
2. Introduction
3. Declaration Syntax
4. Experiments

Class Templates

1. Objectives

After you complete this experiment you will be able to implement and know when it is appropriate to implement a class template.

2. Introduction

Templates allow classes to use and operate on generic data types. They provide a mechanism to parameterize data types. Whenever you notice that two or more classes have identical code but operate on data of different types, you should consider templates if possible.

3. Declaration Syntax

The class definition and all its member functions must be prefaced with the following:

```
template <class Type_Parameter>
```

The implementation for a class template is the same as the implementation for ordinary classes. However, you must remember to preface all member functions with the template heading, even those functions that do not use the generalized type specified in the type parameter.

Consider the following class definition:

```
template <class New_Type>
class Array_Class
{
public:
      Array_Class();
      ~Array_Class();
      void Add(New_Type item);
      int Search(New_Type item);
      void Print();
private:
      New_Type *A;
      int count;
};
```

Carefully observe the locations of the “New_Type” parameter.

Now consider the following program that includes a main function and the definitions of the member functions for the class Array_Class:

```
#include <iostream>
#include <string>

using namespace std;

const int SIZE=5;

/*************************************************************************/
// Class declaraton for Array_Class
/*************************************************************************/

template <class New_Type>
class Array_Class
{
public:
      Array_Class();
      ~Array_Class();
      void Add(New_Type item);
      int Search(New_Type item);
      void Print();
private:
      New_Type *A;
      int count;
};

/*************************************************************************/
// Class definitions for the member function of Array_Class
/*************************************************************************/

template <class New_Type>
Array_Class<New_Type>::Array_Class()
{
      cout<<"You are inside the default constructor\n";
      cout<<"New_Type has a size of "<<sizeof(New_Type)<<" bytes\n\n";
      count=0;
      A = new New_Type[SIZE];
}

template <class New_Type>
Array_Class<New_Type>::~Array_Class()
{
      cout<<"The Destructor has been called\n\n";
      delete [] A;
      count = 0;
      A = 0;
}
```

```
template <class New_Type>
void Array_Class<New_Type>::Add(New_Type item)
{
	if (count<SIZE)
	{
		A[count++] = item;
	}
	else
	{
		cout<<"Array is Full\n";
	}
}


template <class New_Type>
int Array_Class<New_Type>::Search(New_Type item)
{
	int i;

	for(i=0; i<count; i++)
	{
		if (item == A[i])
		{
			return i;
		}
	}
	return -1;
}

template <class New_Type>
void Array_Class<New_Type>::Print()
{
	int i;

	for(i=0; i<count; i++)
	{
		cout<<"A[i"<<i<<"] = "<<A[i]<<endl;
	}
}

int main()
{
	return 0;
}
```

Notice that every member function is prefaced with the template heading, even functions that do not use the type parameter.

More information on templates can be found in your course textbook and on the web.

4. Experiments

Step 1: In this experiment you will investigate a template array class. Enter, save, compile and execute the following program in MSVS. Call the new project "ClassTemplatesExp" and the program "ClassTemplates.cpp". Answer the questions below:

Create a source file which contains the program defined in the previous section (Declaration Syntax). Call the source file "template_tester.cpp". Add code to the main function for the following:

a. declaring an Array_Class object of strings called "my_String";
b. declaring an Array_Class object of integers called "my_Ints";
c. declaring an Array_Class object of characters called "my_Chars";
d. adding the strings "Hello", "GoodBye", "ComeHere", "SayNo" and "SayYes" to my_String;
e. adding the integers 1,2,3,4 and 5 to my_Ints;
f. adding the characters 'a', 'b', 'c' and 'd' to my_Chars;
g. printing the Array_Class objects my_String, my_Ints and my_Chars;
h. searching my_String for the string "SayYes" and "No".

Execute the program and explain its output by answering the following questions:

Question 1: Why do the output statements which contain "New_Type" have different sizes printed?

Question 2: How many times was the destructor called in the program?

Question 3: What happened when parts (d), (e) and (f) were performed by the program?

Question 4: How many items were printed when part (g) was performed by the program?

Question 5: What happened when part (h) was performed by the program?

Question 6: Why was the type parameter included in the class name but not in the function name? Consider the following: void Array_Class<New_Type>::Print( );

Question 7: Why does the Search function return an integer type and not a New_Type type?

Question 8: Are there any types that cannot be parameterized by New_Type? Explain.

Question 9: What happens if you add code to the main function to add 'e', 'f' and 'g' to the "my_Chars" array? Explain.

Question 10: Please implement the following functions for Array_Class.

a. A Boolean function called "Is_Empty" that returns true if the array A is empty; otherwise it returns false.
b. A Boolean function called "Is_Full" that returns true if the array A is full; otherwise it returns false.

c. A void function called “Remove” that deletes an item from the array A. Remove has one formal parameter of type New_Type which holds a copy of the item to be deleted.
d. Add code to the main function to test these functions.

Vectors

Lab Sections

1. Objectives
2. Introduction
3. Definitions
4. Declaration Syntax
5. Experiments

Vectors

1. Objectives

After you complete this experiment you will understand vectors and be able to use them in your programs.

2. Introduction

Vectors are implemented as dynamic arrays. Their elements are stored adjacent or contiguous in memory. Elements are randomly or directly accessed using the subscript operator, pointers or iterators. However, the storage of vectors is handled automatically. This means that you cannot access elements outside the memory range of the vector. Elements can be added to and removed from a vector as necessary.

3. Definitions

We will define several terms that you need to know to understand vectors. They are as follows:

1. **Vector** is part of the STL. . It represents a stack of elements that changes in size while a program is running.
2. **Containers** store other objects in a specific order.
3. **push_back( )** adds an element in the next available position.
4. **size()** returns the total number of elements in a vector or list.
5. **Iterators** provide a means for accessing data stored in a vector regardless of its type.
6. **begin()** returns the iterator pointing to the beginning of the vector.
7. **end()** returns the iterator pointing to the end of the vector.
8. **pop_back()** deletes the last element in a vector.

4. Declaration Syntax

To create a vector of size zero:

```
vector<type> vector_name;
```

To create a vector of a specified size:

```
vector<type> vector_name(size);
```

To create a vector of a specified size with each cell initialized to a value:

```
vector<type> vector_name(size, value);
```

Rule: If **type** is a class it must have a **default constructor defined**.

More information on vectors can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will write declarations for different types of vectors. Enter, save, compile and execute the following program in MSVS. Call the new directory "VectorsExp1" and the program "Vectors1.cpp". Answer the questions below:

Question 1: Can you write a statement to declare a vector of characters called "line" with an initial size of zero?

Question 2: Can you write a statement to declare a vector of integers called "Ages" with an initial size of 20 and each element has a value of 20?

Question 3: Can you declare a vector of characters called "sentence" with a size of 60?

Question 4: Can you declare a vector of "sentences" (declared in Question 3) called "page" with a size of 50?

Question 5: Can you declare a vector of "pages" (declared in Question 4) called "Book" with a size of 0?

Step 2: In this experiment you will investigate a program that uses a vector.
Step 1: In this experiment you will write declarations for different types of vectors.
Enter, save, compile and execute the following program in MSVS. Call the new directory "VectorsExp2" and the program "Vectors2.cpp". Answer the questions below:

```
#include <vector>
#include <iostream>
using namespace std;

class coordinate
{
   public:
      double x,y;
};

int main()
{
      vector<coordinate> moves;
      coordinate C;

      for (unsigned int i=0; i<10; i++)
      {
            C.x =  i * 2.386;
            C.y = i * 4.87;
           moves.push_back( C );
      }

      for (unsigned int j=0; j<moves.size( ); j++)
      {
            cout<<"Coordinate in cell  "<< j << moves[j].x<<",
"<<moves[j].y<<endl;
      }

       moves.clear( );

       return 0;

}
```

Question 6: Please execute the program in Step 2? Explain any complier messages and the output produced by the program?

Question 7: Remove the "unsigned" modifier in the two "for" loop headers and execute the program in Step 2. Explain any compiler messages and the output produced by the program?

Recursion

Lab Sections

1. Objectives
2. Introduction
3. Definitions
4. Declaration Syntax
5. Experiments

Recursion

1. Objectives

After this experiment you will be able to implement a simple recursive function.

2. Introduction

There are two types of recursion, direct and indirect. Direct recursion occurs when a function calls itself. Indirect recursion occurs when a function calls another function, which calls another function, which calls another function, until eventually the original function is called. Many solutions to problems are easily implemented recursively.

All recursive algorithms divide a large problem into smaller less complex sub-problems. This process is known as "Divide-and-conquer". Each sub-problem is also divided into smaller sub-problems until eventually one of those sub-problems can be solved. This solution is then returned to its predecessor (parent problem), which uses this solution to solve its problem. This procedure is continued until all sub-problems have been solved. Once this occurs, the solution to the original problem can be determined. Each recursive call represents a new totally independent call from all the previous function calls. Memory must be allocated for its formal parameters, return address and so forth.

All recursive algorithms can be implemented iteratively. Iteratively means they can be implemented using loops (while, for, etc.).

3. Definitions/Important Terms

We will define several terms you need to know to understand recursion. They are as follows:

a. A **base, sometimes referred to as the anchor or trivial case,** is a very easy solution that does not require too much thought to solve.
b. **Recursive case** is the step in the algorithm where the problem is divided into smaller less complex problems.

4. Syntax for a Recursive Function

Following is a simple "multiple-alternative if" statement that can be used to implement a recursive function. However, please keep in mind that recursive functions may have different formats. To write a recursive function usually requires two steps. First, identify the base case. Secondly, identify any recursive cases.

```
if (base case)
     return trivial solution
else //(recursive case 1)
     return (solution from recursive call)
.....
else //(recursive call)
   return (solution from recursive call)
```

More information on recursion can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will learn how to declare functions.
Enter, save, compile and execute the following program in MSVS. Call the new directory "recursionExp1" and the program "recursion1.cpp". Answer the questions below:

```
#include <iostream>
using namespace std;

void recursive_countdown(int count)
{
	if (count == 0)
		cout<<"count="<<count<<endl;
	else
	{
		cout<<"count="<<count<<endl;
		recursive_countdown(--count);
	}
}


int main(void)
{
	int count = 10;

	recursive_countdown(count);

	return 0;
}
```

Question 1: What does the program do?

Question 2: Please write the base case?

Question 3: How many times is "recursive_countdown" called recursively?

Question 4: Please rewrite the program in Step 1 using an iterative function to countdown? Call the function "iterative_countdown".

Question 5: Which type of function (recursive or iterative) executes the fastest? Please explain?

Step 2: In this experiment you will learn how to declare functions.

Enter, save, compile and execute the following program in MSVS. Call the new directory "recursionExp2" and the program "recursion2.cpp". Answer the question below:

```
#include <iostream>
using namespace std;

int R_power(int count, const int & base)
{
	if (count == 0)
		return 1;
	else
		return base * R_power(count-1,base);
}

int I_power(int count, const int & base)
{
	int multiend = 1;

	while (count > 0)
	{
		multiend *= base;
		count--;
	}
	return multiend;
}

int main()
{
	int count = 10;
	int base = 2;

	cout<<R_power(count,base)<<endl;

	cout<<I_power(count,base)<<endl;

	return 0;
}
```

Question 6: Please explain the operation of the functions "R_power" and "I_Power"?

Step 3: In this experiment you will learn how to declare functions.

Enter, save, compile and execute the following program in MSVS. Call the new directory "recursionExp3" and the program "recursion3.cpp". Answer the questions below:

Question 7: Please write an iterative function to reverse a string? For example, if a string is "hello" then the function will return "olleh".

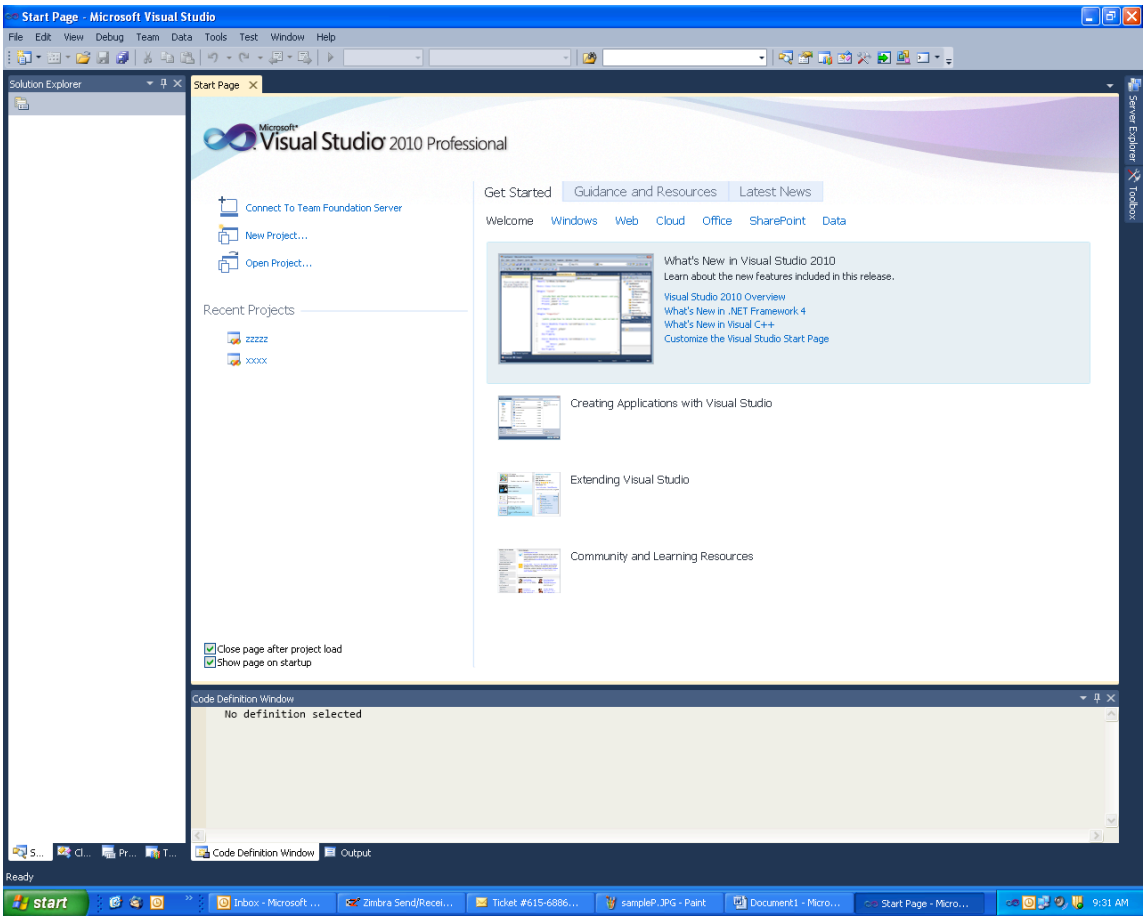
Question 8: Please write a recursive function to reverse a string? For example, if a string is "hello" then the function will return "olleh".

Compiling a C++ Program in Microsoft Visual Studio 2010

Objective: After completing this experiment you will be able to:

a. create an empty C++ project;
b. generate a source file to hold a C++ program;
c. compile (build) a C++ program;
d. execute a C++ program;

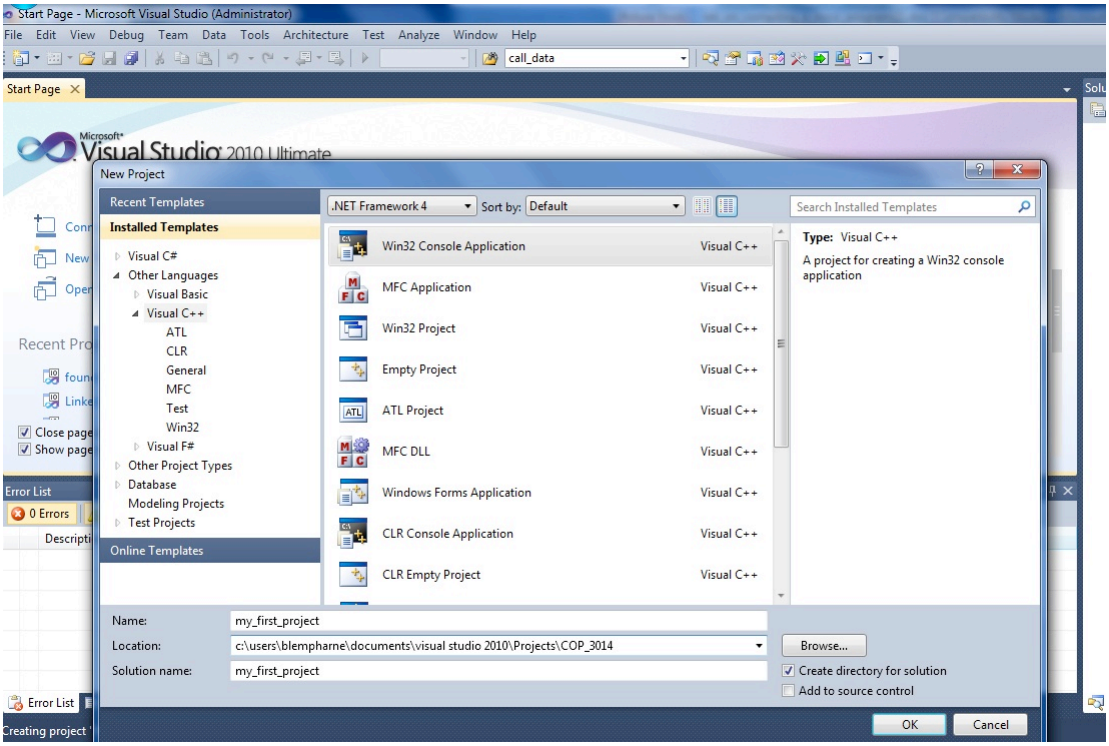
Step 1: Open Visual Studio 2010. You should see a screen similar to the one shown below. Now, select New Project, and continue to the next step.



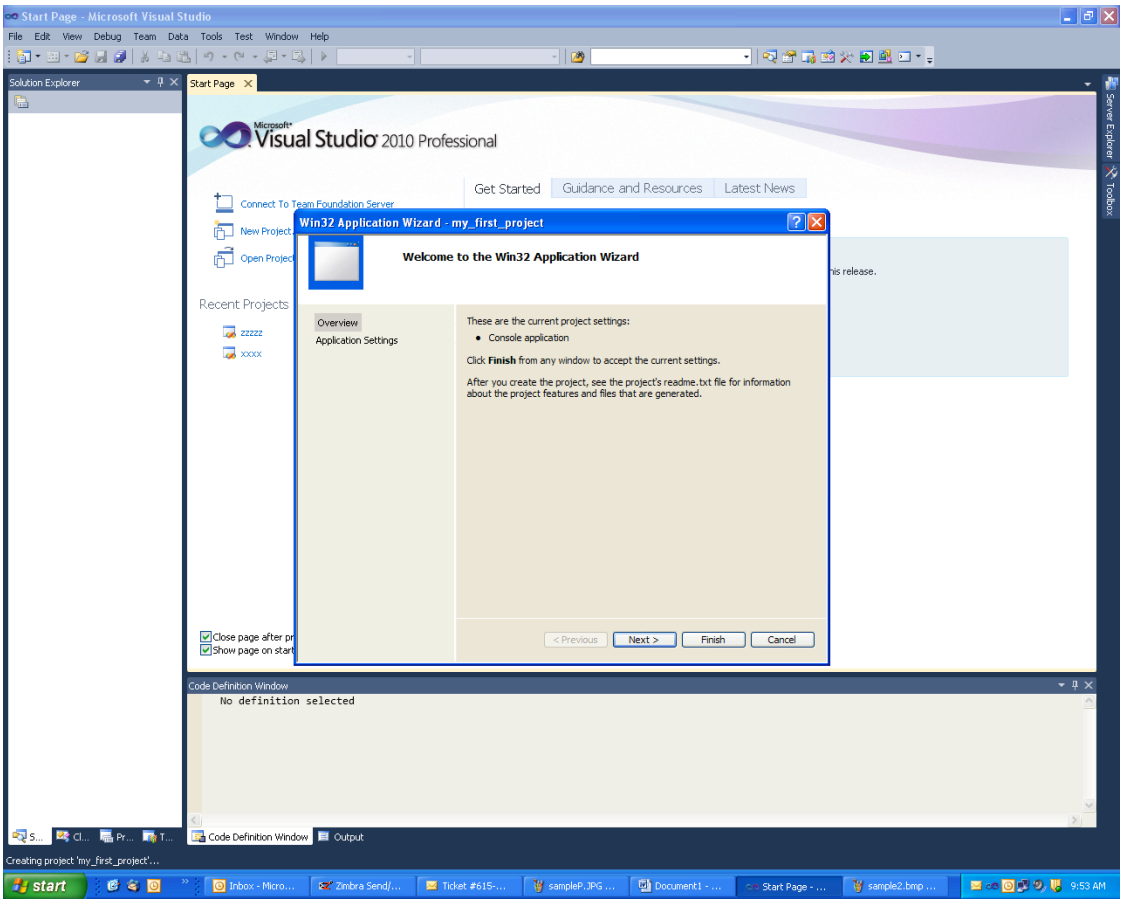
Step 2: Your screen should look similar to the one below. Perform the following actions:

a. Choose Visual C++ in the “Installed Templates” area;
b. Choose Win32 Console Application in the middle section;
c. Name the project “my_first_project”;
d. Create a folder called “COP3014_Laboratory” by provide the location where the folder “COP3014_Laboratory” will reside.

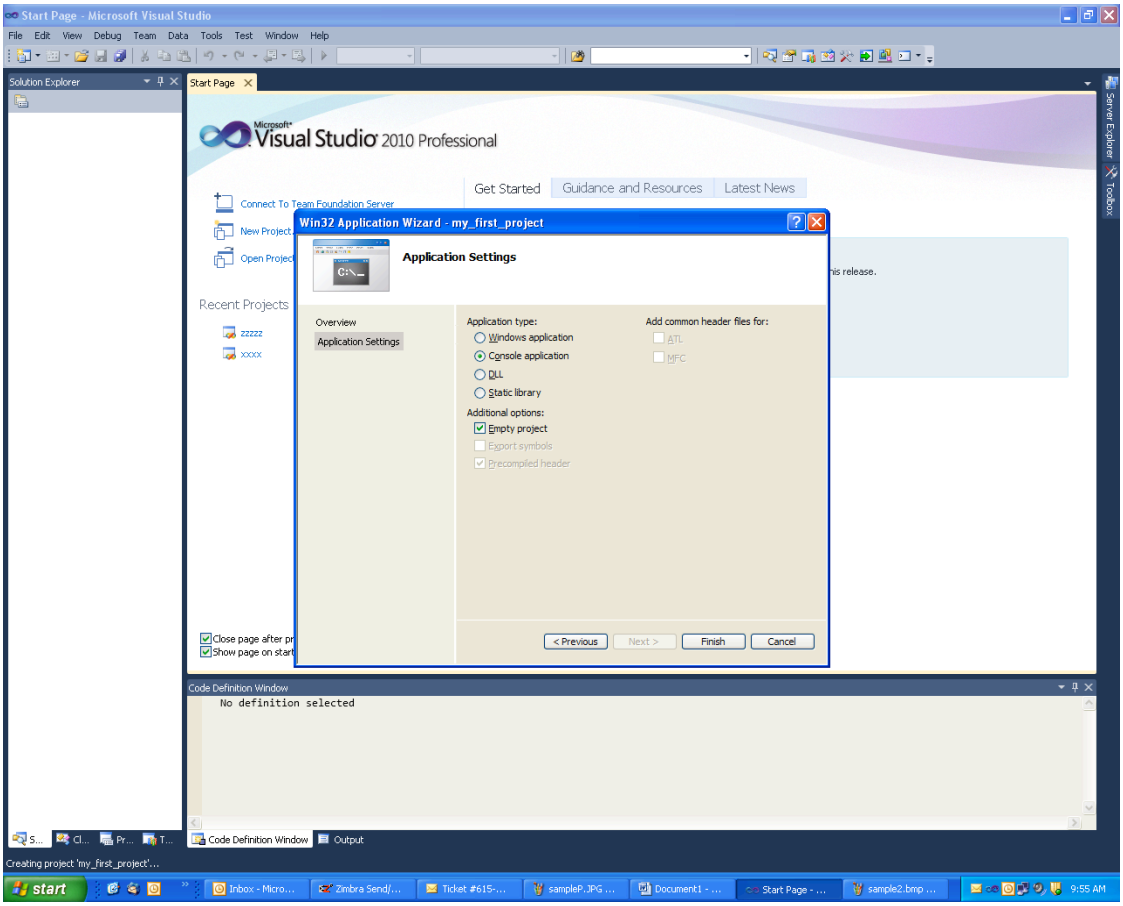
Continue to the next step.



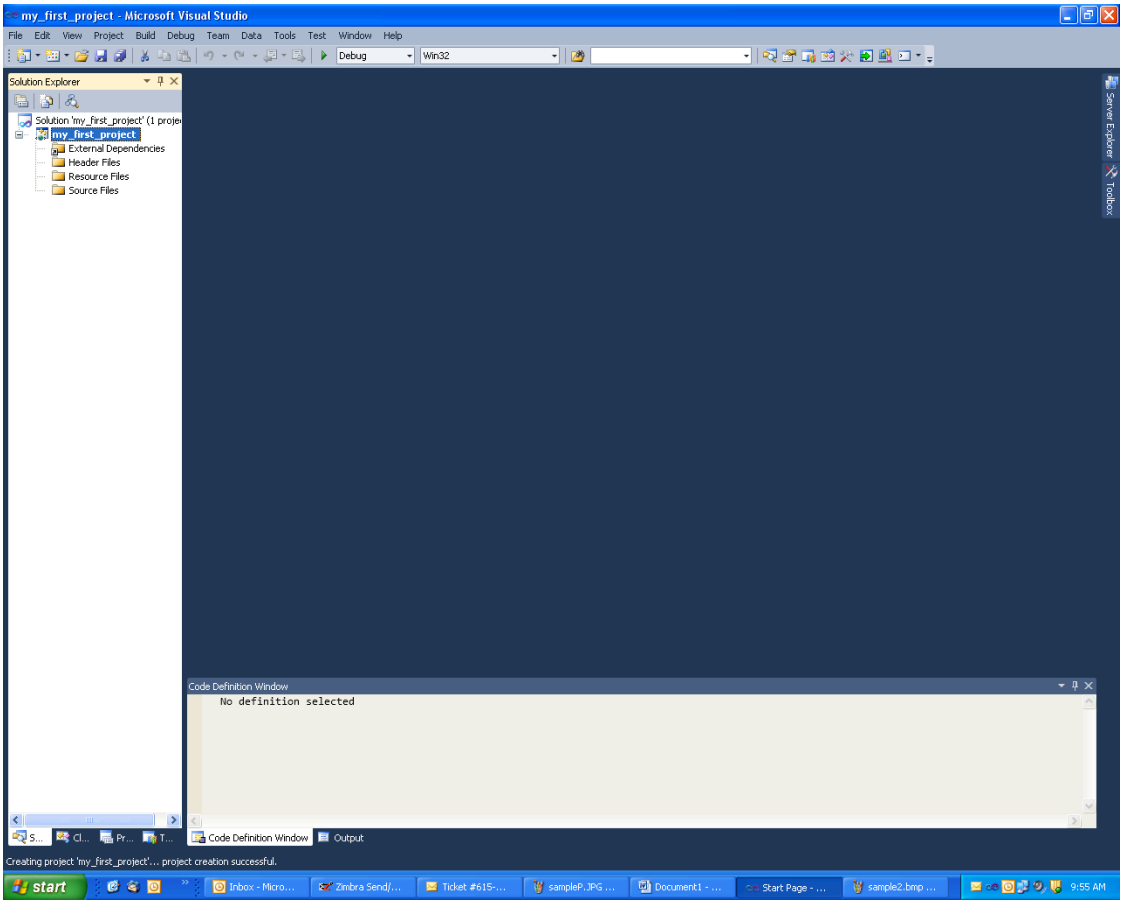
Step 3: Your screen should look similar to the screen shown below. Choose next, and continue to the next step.



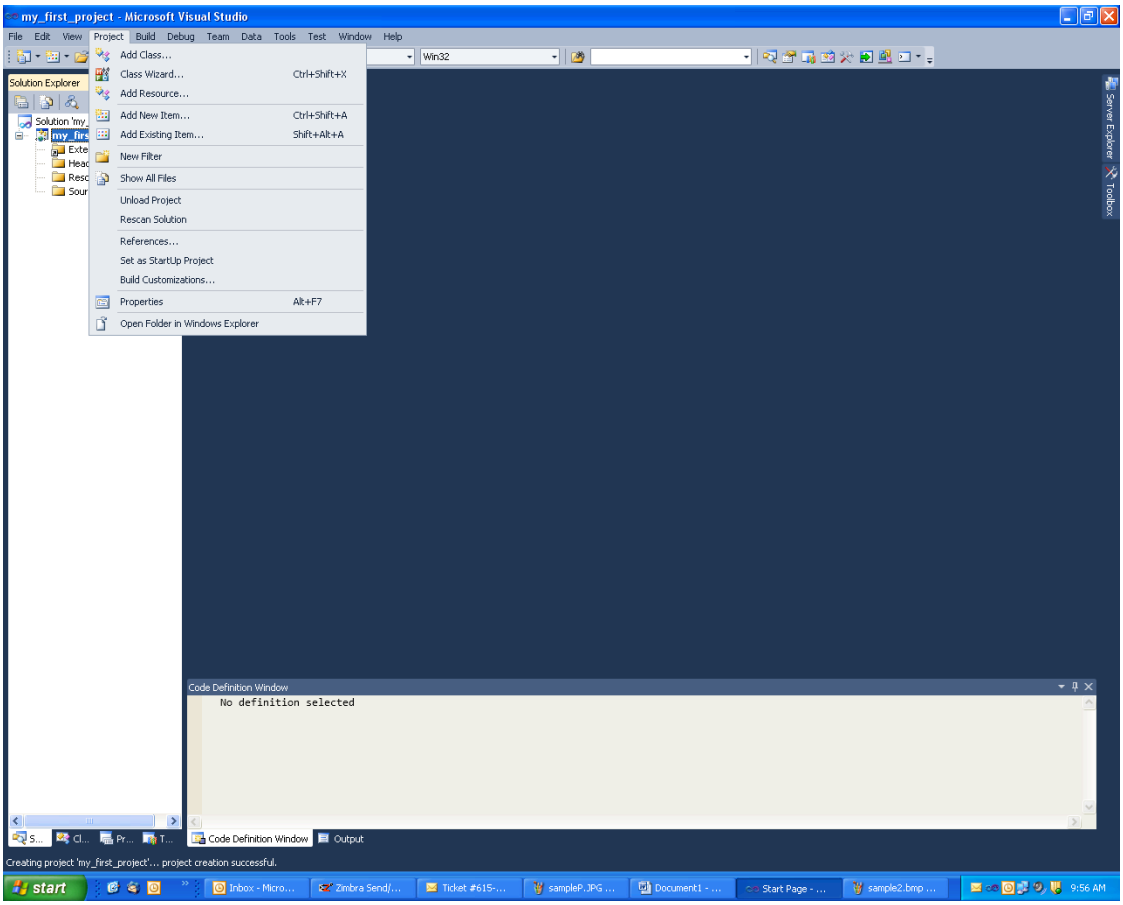
Step 4: Your screen should look similar to the screen shown below. Choose “empty Project”. The Application type should be “console application”. Now select Finish, and continue to the next step.



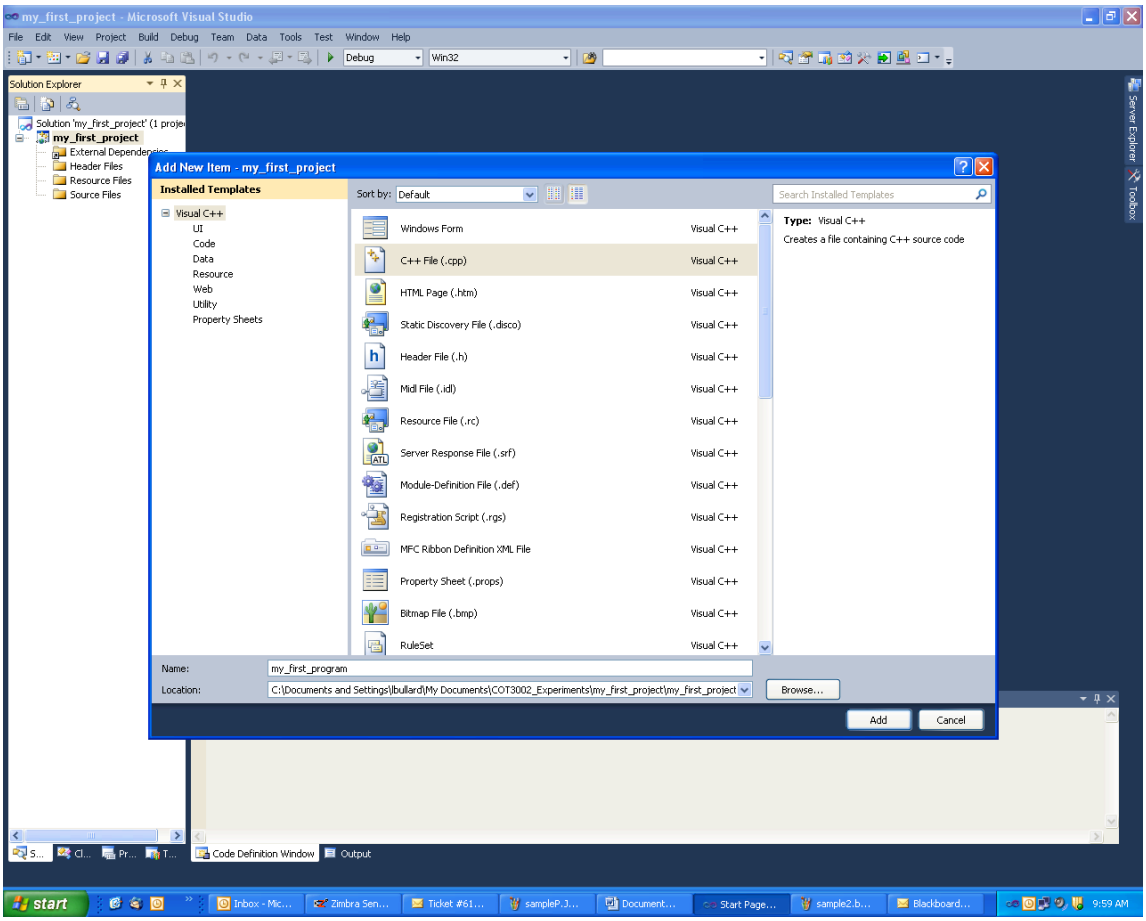
Step 5: Your screen should look similar to the blank screen shown below. Continue to the next step.



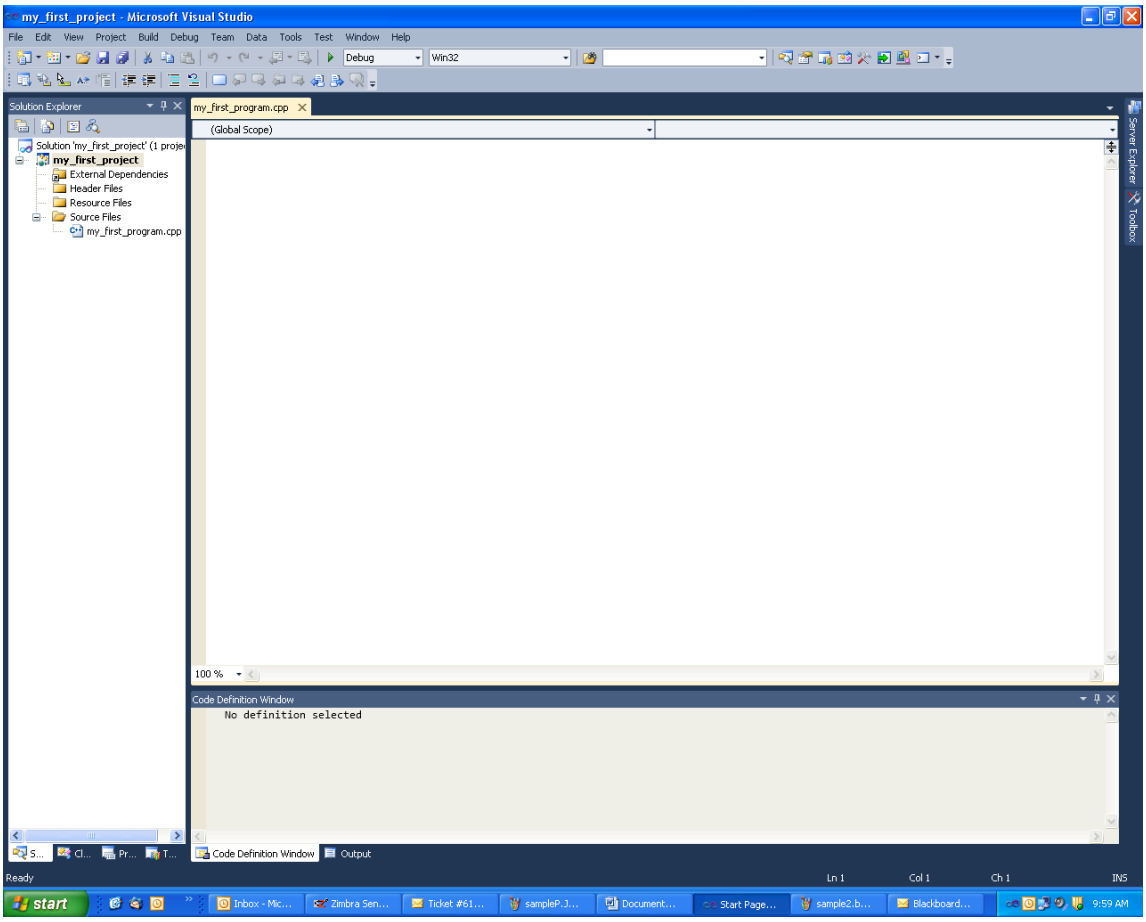
Step 6: Select the Project pull down menu, and choose “Add New Item”. Continue to the next step.



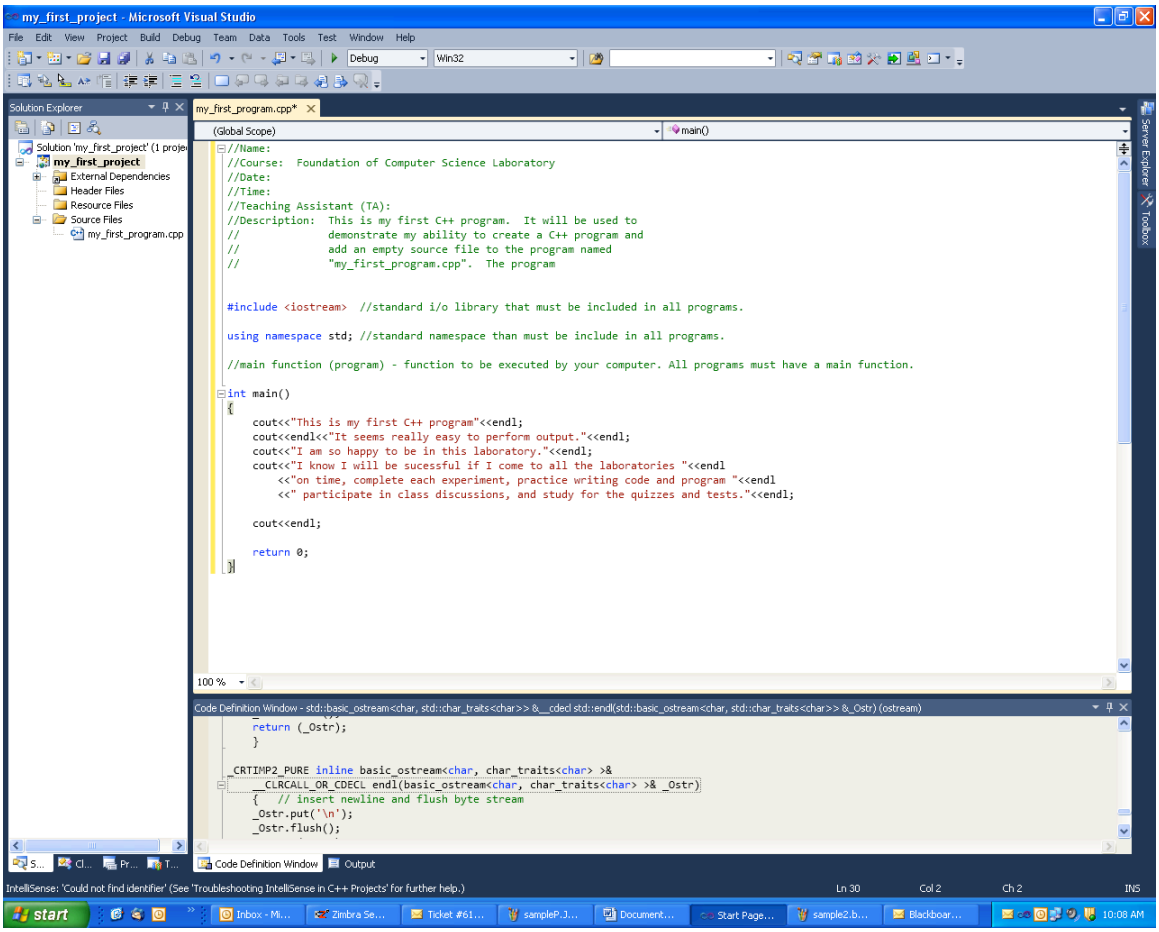
Step 7: Your screen should look similar to the one shown below. In the middle section select “C++ File(.cpp)”. Name the source file “my_first_program” in the name field at the bottom. The location field should already be set to the location of your project. Choose Add, and continue to the next step.



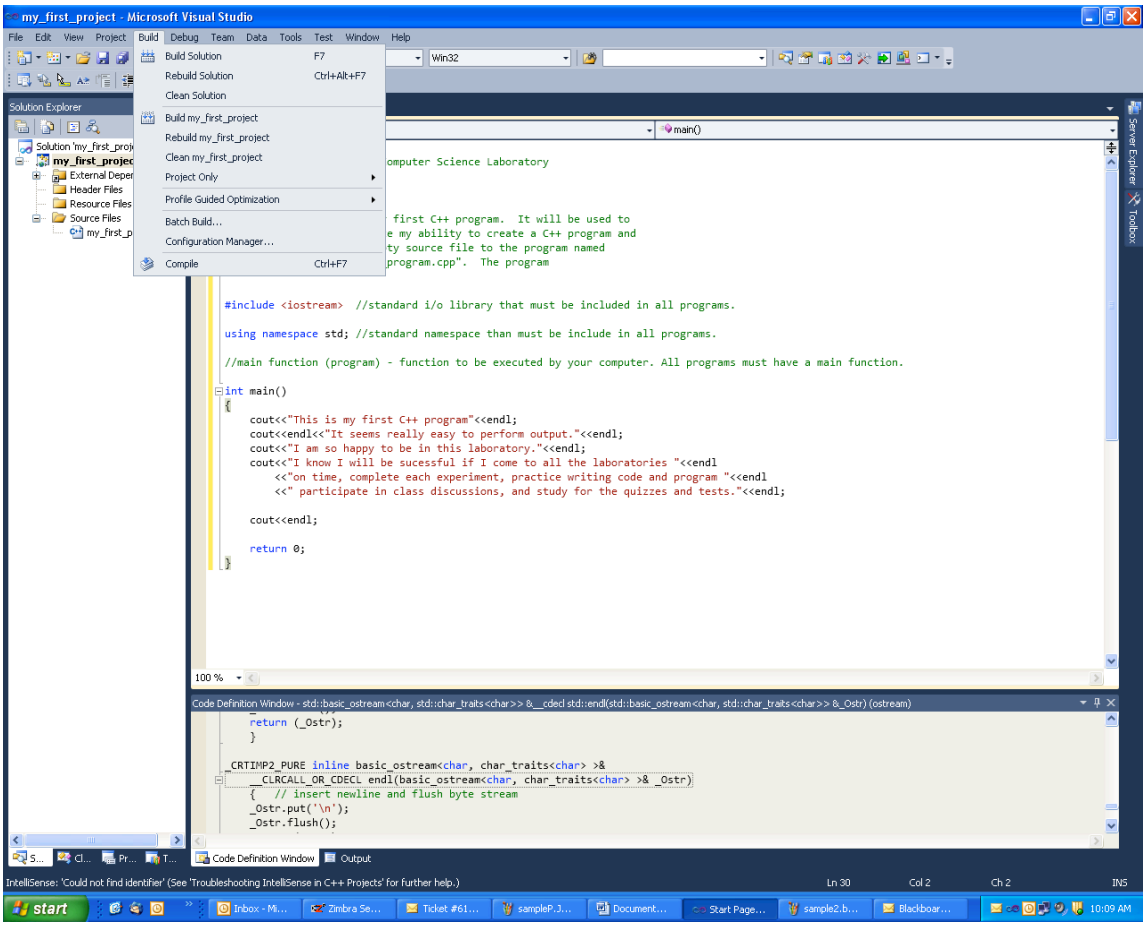
Step 8: Your screen should look similar to the one shown below. Continue to the next step.



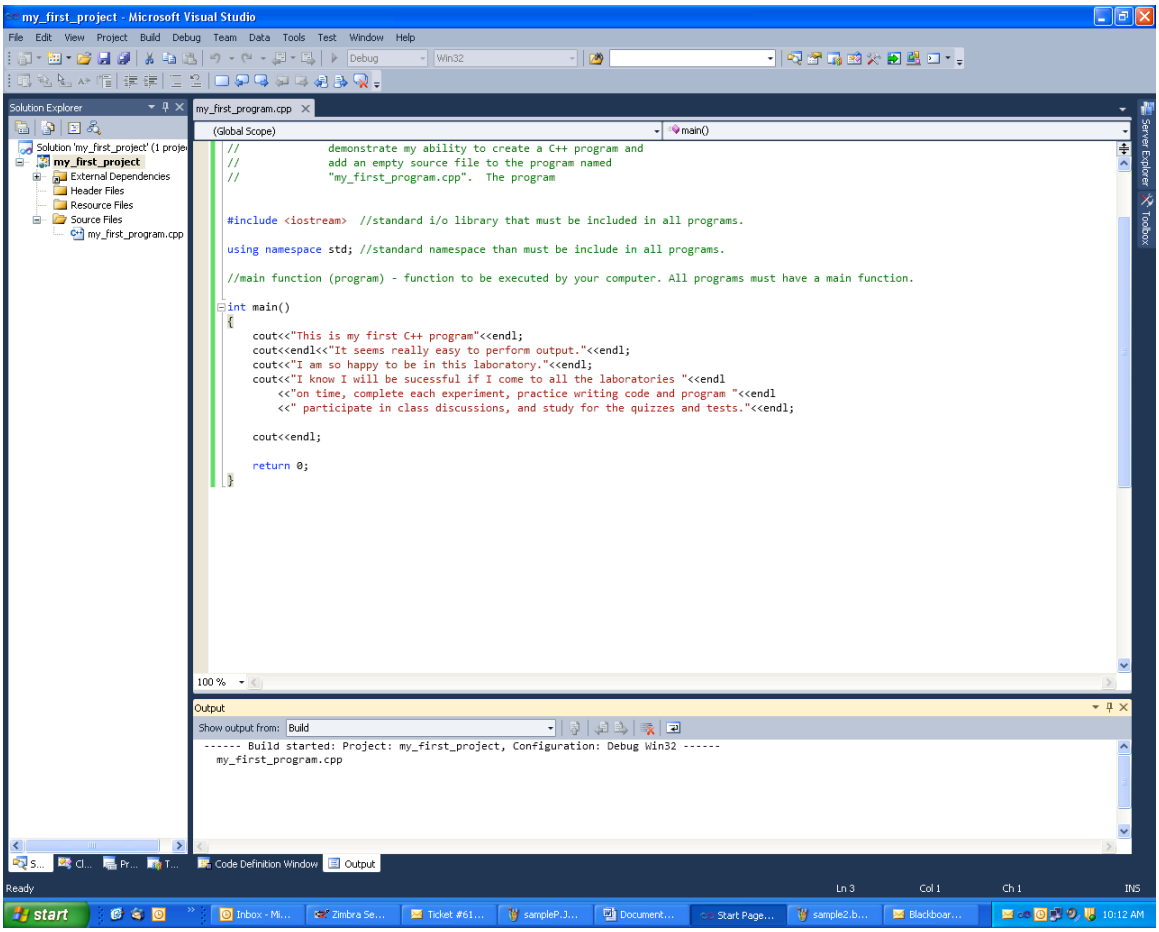
Step 9: Enter the C++ program shown below. Be careful or you will get errors when you try to compile and execute the program. Pay close attention to the syntax. If you enter the program correctly, it will compile without any errors. Continue to the next step after you enter the program.



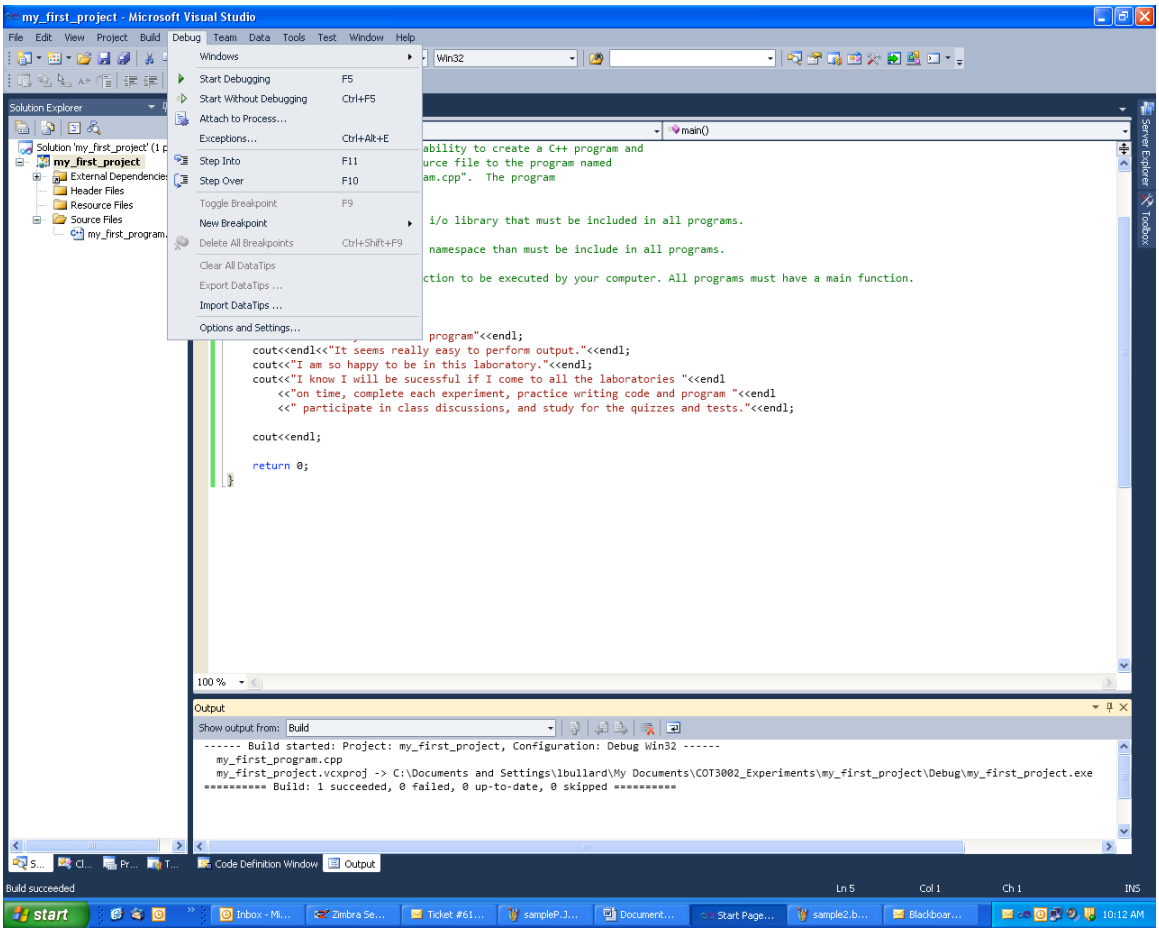
Step 10: Choose the Build pull down menu, and select Build Solution. During this step the C++ compiler will compile your program. Continue to the next step.



Step 11: Your screen should look like the one shown below. If you get any errors, try to fix them. If you can't, speak to your laboratory TA immediately.



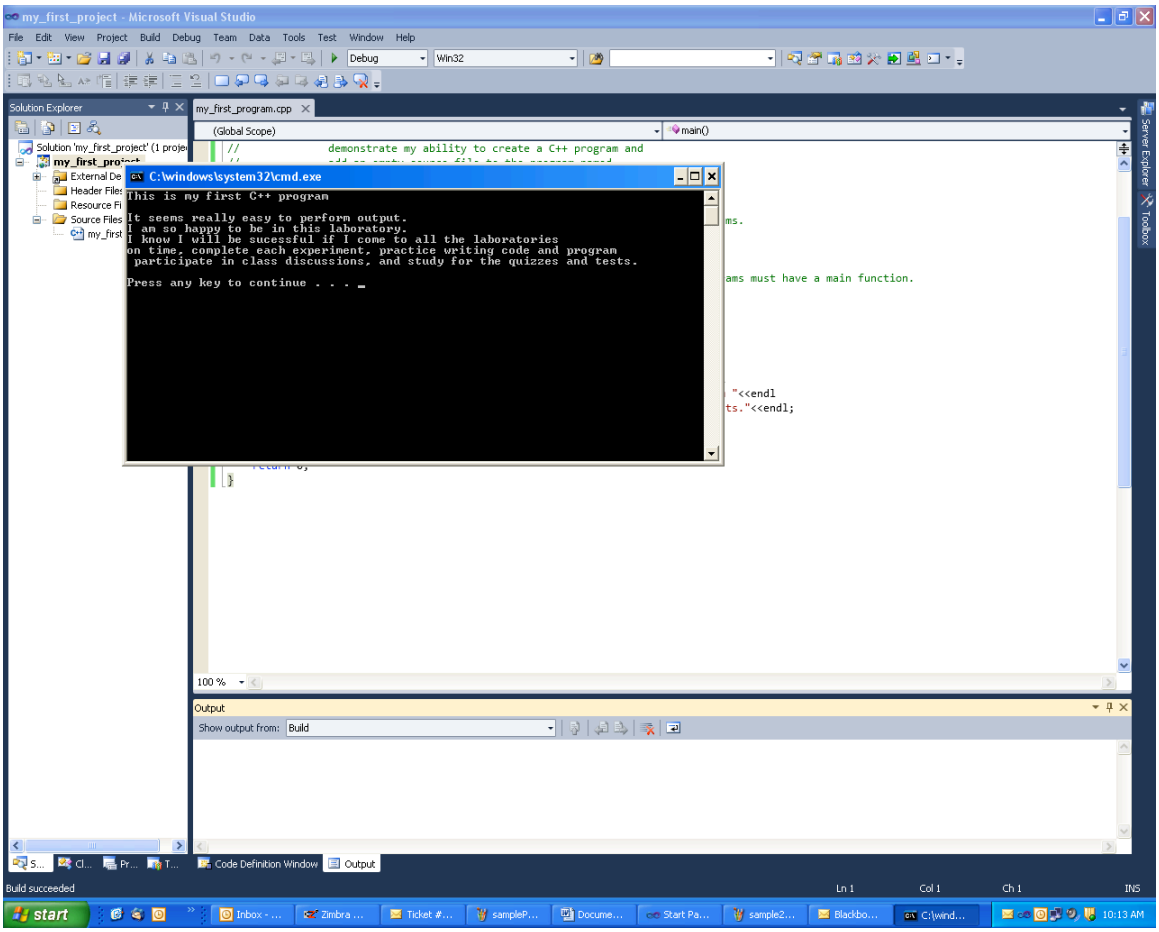
Step 12: Choose the Debug pull down menu, and select “Start Without Debugging”. Your program should execute without any problems. If your program does not execute, speak to your laboratory TA. Continue to the next step.



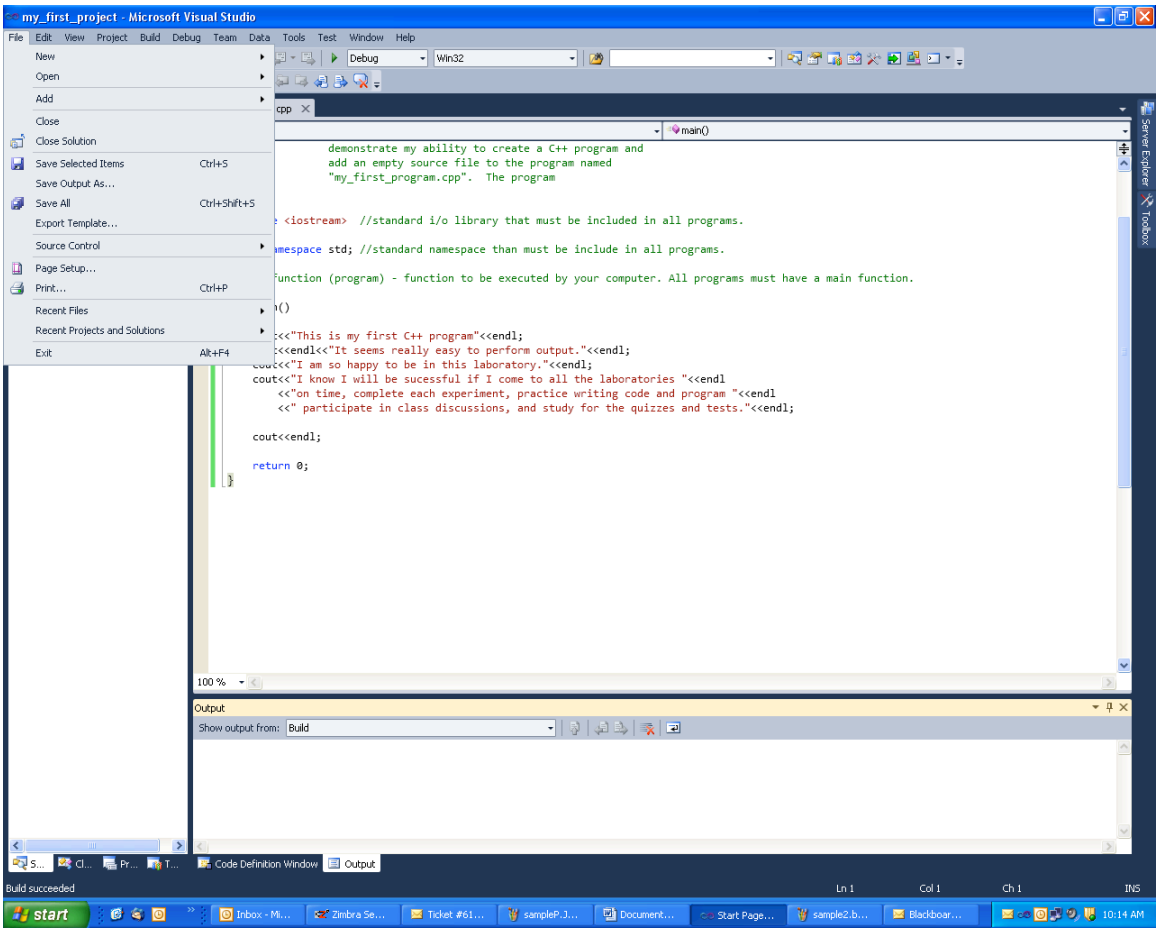
Important: If this is your first time using Visual Studios 2010, the black console window may appear to flash briefly then disappear. If this is the case you need to setup your Output to Console. To do this:

1. Go to “Project” then choose the last item: “my_first_project Properties…”
2. Choose the ‘+’ and open up Configuration Properties.
3. Choose the ‘+’ and open up Linker.
4. Choose System. This opens up a number of options on the right side.
5. Under SubSystem, go to right side of the pull down menu and choose:
   Console (/SUBSYSTEM:CONSOLE)
6. Choose OK to save your changes. Then rebuild and run your program.

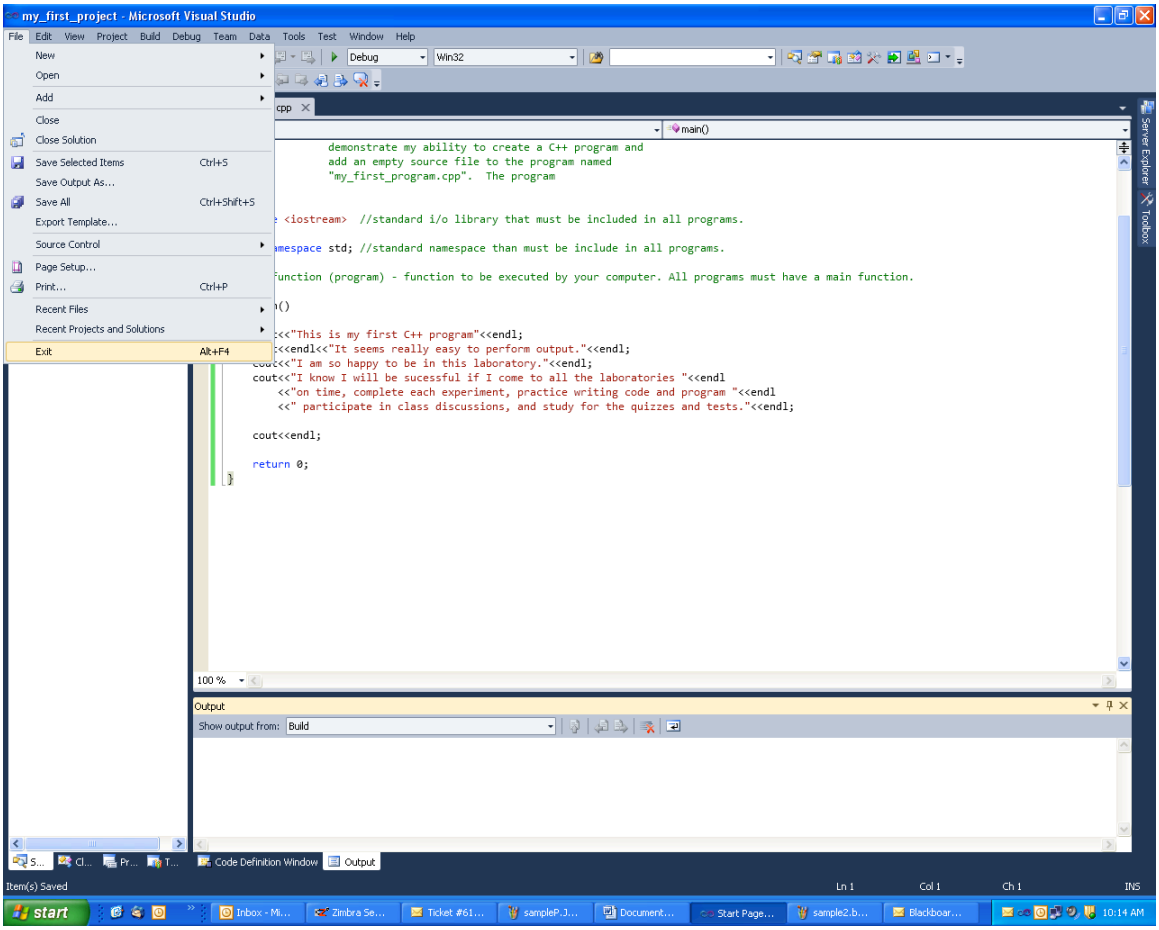
Step 13: Your screen should look like the screen shown below. The black screen shows the output produced by your program. Continue to the next step.



Step 14: Choose the File pull down menu, and select “Save All”. Continue to the next step.



Step 15: Choose the File pull down menu, and select “Exit”. Continue to the next step.



Step 16: You have complete your first experiment.

Simple Types

Lab Sections

1. Objectives
2. Introduction
3. Definitions
4. Experiments

Declarations of Simple Types

1. Objectives

After you complete this experiment you will be able to:

a. **discuss the amount of memory a Boolean, character, integer, float and double require**
b. **declare a Boolean, character, integer, float and double**
c. **compare Booleans, characters, integers, floats and doubles**

2. Introduction

Variable declarations tell the compiler how much memory to use when assigning values to a variable, and what operations can be performed on that variable.

3. Definitions

We will define several variable types that you will be using early in the semester. They are as follows:

1. **bool** refers to a Boolean type. If it holds a value equal to 0 it is false; otherwise it is true.
2. **char** refers to character type. It can hold any integer that represents a character (symbol).
3. **int** refers to integer type. It can hold any integer of a size specified by your system.
4. **float** refers to float type. The float type is a single-precision, floating-point number (positive or negative).
5. **double** refers to double type. The double type is a double-precision, floating-point number (positive or negative). It can be larger or equal to the size of a float.

More information on these and other variable types can be found in your course textbook and on the web.

4. Experiments

Step 1: In this experiment you will determine how much memory a simple variable requires. Enter, save, compile and execute the following program in MSVS. Call the new project "SimpleTypeExp1" and the program "simpleTypeDecls1.cpp". Answer the questions below:

```
#include <iostream>
using namespace std;

int main()
{
	bool response;
	char character;
	int integer;
	float single_precision_number;
	double double_precision_number;
```

```
	cout<<"A boolean uses "<<sizeof(bool)<<" bytes."<<endl;
	cout<<"A character uses "<<sizeof(char)<<" bytes."<<endl;
	cout<<"An integer uses "<<sizeof(int)<<" bytes."<<endl;
	cout<<"A float uses "<<sizeof(float)<<" bytes."<<endl;
	cout<<"A double uses "<<sizeof(double)<<" bytes."<<endl;

	return 0;
}
```

Question 1: Referring to the program in Step 1 (simpleTypeDecls1.cpp), please discuss the output(if any), and any errors or warnings your compiler gives.

Question 2: Please try running this program on another computer, if possible. Explain your observations.

Question 3: What is the magnitude of the largest positive value you can place in a bool? a char? an int? a float? a double? (hint: Wikipedia "limits.h".)

Step 2: Enter, save, compile and execute the following program in MSVS. Call the project"SimpleTypeExp2" and the program "simpleTypeDecls2.cpp". Answer the questions below:

```
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
	bool response = 234;
	char character = 68;
	int integer = 123.456789;
	float single_precision_number = 1234.567890123456789;
	double double_precision_number = 1234.567890123456789;

	cout<<"response =  "<< response <<endl;
	cout<<"character =  "<< character <<endl;
	cout<<"integer =  "<< integer <<endl;
cout<<"single_precision_number =  "<< setprecision (17)
<<single_precision_number<<endl;
cout<<"double_precision_number =  "<< setprecision (17)
<<double_precision_number<<endl;

	return 0;
}
```

Question 4: Please explain each line of output.

Question 5: Make the following changes to the program in Step 2 (simpleTypeDecls2.cpp) and explain the output you get.

- change the value of “response” to 0 (zero).
- change the value of “character” to ‘A’

Structures

Lab Sections

1. Objectives
2. Introduction
3. Definitions
4. Declaration Syntax
5. Experiments

Declarations of Structures

1. Objectives

After you complete this experiment you will be able to:

a. **determine the amount of memory a structure uses;**
b. **declare a structure;**
c. **discuss the properties of a structure.**

2. Introduction

A **structure** is a heterogeneous data type. We say it is heterogeneous because it may be composed of members that all have different types. The members of a structure are accessed using the dot operator, and all members are public by default.

3. Definitions

We will define several terms that you need to know to really understand structures. They are as follows:

a. The **dot operator** is used to access the members of a structure. The symbol, '**.**', is used to represent the dot operator.
b. The **total amount of memory** a structure uses is equal to the sum of all the memory (bytes) used by its members.
c. A structure is a **user defined data type**.
d. A structure is a **complex data type** because it may be composed of other structures and data types.
e. A structure is a heterogeneous data type because it may be composed of members that all have different types.

4. Declaration Syntax

To declare a structure:

```
struct structure_name
{
      field_type_1  field_name_1;
      . . . .
      field_type_n  field_name_n;
};
```

Notice that the member fields are enclosed in braces and that the right brace is followed by a semicolon.

More information on structures can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will learn how to declare structures and access their members. Enter, save, compile and execute the following program in MSVS. Call the new project "StructuresExp1" and the program "structureDecls1.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

struct student_record
{
	string firstname, lastname;
	double age, income;
	int number_of_children;
	char sex;
};

int main()
{
	student_record Mary;

	cout<<"Enter the firstname and lastname: ";
	cin>>Mary.firstname;
	cin>>Mary.lastname;
	cout<<"Enter age: ";
	cin>>Mary.age;
	cout<<"Enter income: ";
	cin>>Mary.income;
	cout<<"Enter number of children: ";
	cin>>Mary.number_of_children;
	cout<<"Enter sex: ";
	cin>>Mary.sex;

	cout<<Mary.firstname<<"    "<<Mary.lastname<<endl;
	cout<<Mary.age<<endl;
	cout<<Mary.income<<endl;
	cout<<Mary.number_of_children<<endl;
	cout<<Mary.sex<<endl;

	return 0;
}
```

Question 1: Please discuss the output (if any) and any compiler errors or warnings?

Question 2: Please move the declaration of the structure "student_record" in the program in Step 1 inside the main program body (before the declaration "student_record Mary;"). Explain your observations.

Step 2: In this experiment you will learn how to declare structures and access their members. Enter, save, compile and execute the following program in MSVS. Call the new project "StructureExp2" and the program "StructureDecls2.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

struct student_record
{
	string firstname, lastname;
	double age, income;
	int number_of_children;
	char sex;
};

int main()
{

	student_record Mary;
	student_record Susan;

	cout<<"Enter the firstname and lastname: ";
	cin>>Mary.firstname;
	cin>>Mary.lastname;
	cout<<"Enter age: ";
	cin>>Mary.age;
	cout<<"Enter income: ";
	cin>>Mary.income;
	cout<<"Enter number of children: ";
	cin>>Mary.number_of_children;
	cout<<"Enter sex: ";
	cin>>Mary.sex;

	Susan = Mary;

	cout<<Susan.firstname<<"    "<<Mary.lastname<<endl;
	cout<<Susan.age<<endl;
	cout<<Susan.income<<endl;
	cout<<Susan.number_of_children<<endl;
	cout<<Susan.sex<<endl;

	return 0;
}
```

Question 3: What can you say about the assignment operator shown in the program in Step 2 when the left and right operands are structures?

Step 3: In this experiment you will investigate how the =, ==, <= and >= operators are used to compare two different structures. Enter, save, compile and execute the following program in MSVS. Call the new project "StructureLogOpsExp" and the program "StructureLogOps.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

struct student_record
{
	string firstname, lastname;
	double age, income;
	int number_of_children;
	char sex;
};

int main()
{

	student_record Mary;
	student_record Susan;

	cout<<"Enter the firstname and lastname: ";
	cin>>Mary.firstname;
	cin>>Mary.lastname;
	cout<<"Enter age: ";
	cin>>Mary.age;
	cout<<"Enter income: ";
	cin>>Mary.income;
	cout<<"Enter number of children: ";
	cin>>Mary.number_of_children;
	cout<<"Enter sex: ";
	cin>>Mary.sex;

	Susan = Mary;

	if (Susan == Mary)
	{
		cout<<Susan.firstname<<"    "<<Mary.lastname<<endl;
		cout<<Susan.age<<endl;
		cout<<Susan.income<<endl;
		cout<<Susan.number_of_children<<endl;
		cout<<Susan.sex<<endl;
	}
	return 0;
}
```

Question 4: What can you say about the logical equaility operator when the left and right operands are structures?

Question 5: What happens when you replace the logical equality operator (==) with the <= operator in the program in Step 3? Discuss your observations.

Question 6: What happens when you replace the logical equality operator (==) with the >= operator in the program in Step 3? Discuss your observations.

Question 7: What would you do to correct the program in Step 3, if necessary?

Pointers & References

Lab Sections

1. Objectives
2. Introduction
3. Definitions
4. Declaration Syntax
5. Experiments

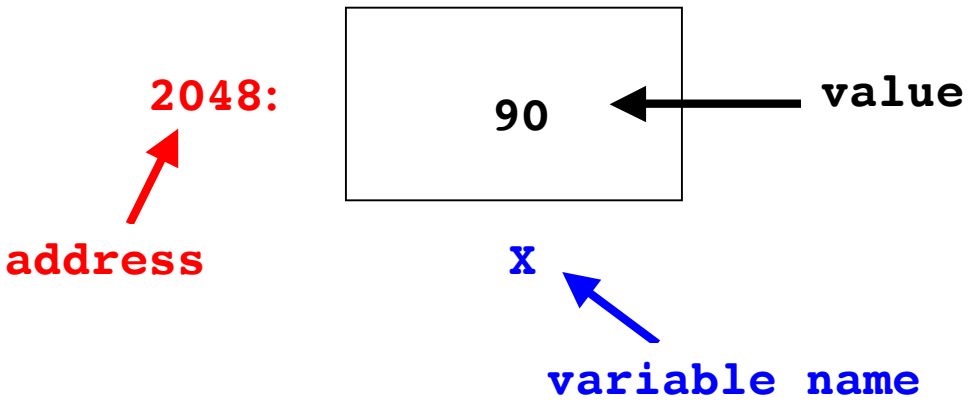
Pointers & References – The Basics

1. Objectives

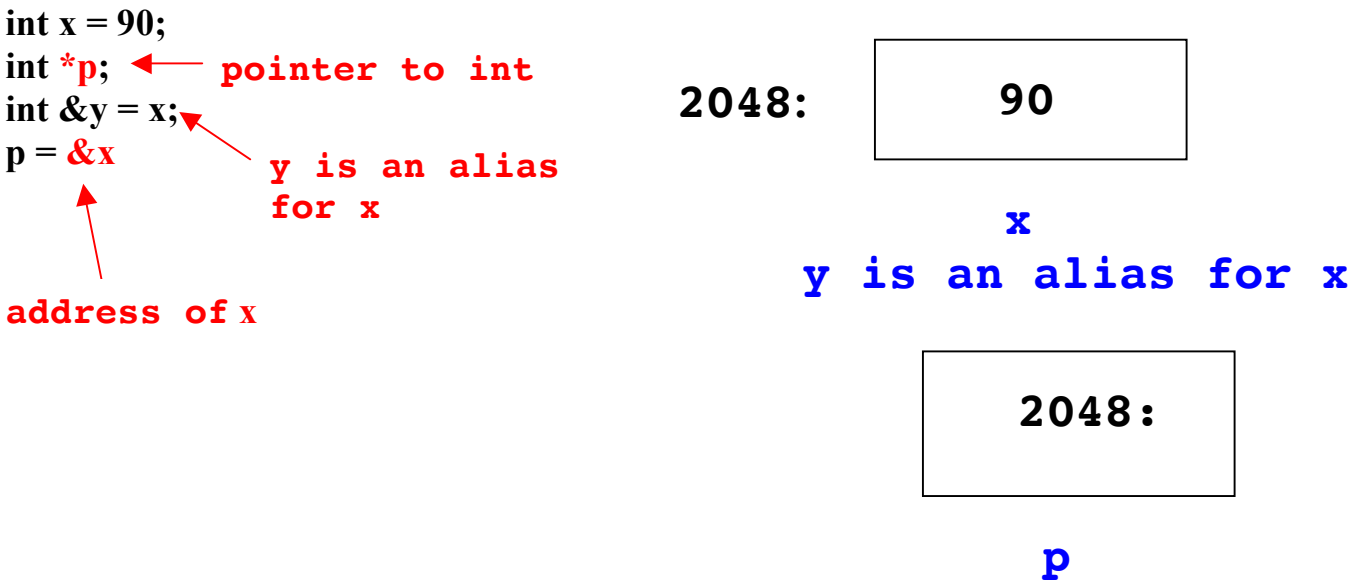
After you complete this experiment you will be able to declare and use pointers and references in a C++ program.

2. Introduction

Variables name the memory locations where values are stored. Declarations tell the compiler how much memory to use when assigning values to a variable, and what operations can be performed on a variable. Consider the following figure:



A pointer holds the address of a memory location. In other words, a pointer contains a reference to another variable. A reference in C++ is a variable that references a memory address. References are often referred to as "aliases" of other variables. Consider the following figure:



3. Definitions

We will define several terms to help you understand pointers. They are as follows:

1. **Pointers** hold memory addresses.
2. When you **de-reference** a pointer, you retrieve the contents of the memory address that is stored in the pointer.
3. A **reference** is an alias for memory that is allocated elsewhere.
4. The "*****" operator is used to declare and de-reference a pointer. Please pay close attention to the context in which the operator is used.
5. The "**&**" operator is called the "address of" operator.
6. The "**&**" operator is also used to declare references in C++.

4. Declaration Syntax

Declaration for a pointer:

```
type  * pointer_name;
```

Declaration for a reference:

```
type & reference_name;
```

More information on pointers can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate the use of pointers and references in a C++ program. Enter, save, compile and execute the following program in MSVS. Call the new project "PtrsAndRefsExp1" and the program "ptrsAndrefs1.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

int main()
{
	int i;
	int *p;

	cout<<&i<<endl;
	cout<<p<<endl;
	cout<<&p<<endl<<endl;

	i=90;
	p = &i;

	cout<<i<<endl;
	cout<<(*p)<<endl;
```

```
	return 0;
}
```

Question 1: Please discuss the output (if any), and any errors or warnings your compiler gives. (Hint: Describe how the different operators are being used in different statements. Be as complete as possible in your description.)

Step 2: In this experiment you will investigate the use of pointers and references in a C++ program. Enter, save, compile and execute the following program in MSVS. Call the new project "PtrsAndRefsExp2" and the program "ptrsAndrefs2.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

int main()
{
	int i;
	int *p=0;
	int &temp=i;

	cout<<&i<<endl;
	cout<<p<<endl;
	cout<<&temp<<endl<<endl;

	i=90;
	p = &i;

	cout<<i<<endl;
	cout<<(*p)<<endl;
	cout<<temp<<endl;

	return 0;
}
```

Question 2: Please discuss the output (if any), and any errors or warnings your compiler gives. (Hint: Describe how the different operators are being used in different statements. Be as complete as possible in your description.)

Functions: Declarations

Lab Sections

1. Objectives
2. Introduction
3. Definitions
4. Declaration Syntax
5. Experiments

Declarations of Functions

1. Objectives

After you complete this experiment you will be able to:

a. **declare a function**
b. **understand the alternate syntax for a function declaration**
c. **understand why function prototypes are important**

2. Introduction

Using functions in programs has several advantages which include: 1) allows for code reuse; 2) allows programs to be decomposed into smaller components (modularization); 3) they make debugging easier; 4) they make testing easier; 5) they allow for the implementation of recursion; 6) allow for generalizing a program; 7) allow information hiding.

3. Definitions

We will define several terms you need to know to really understand functions. They are as follows:

a. Formal parameters reside inside a function header. They allow the actual arguments to be passed to a function. The scope of formal parameters is local to the function where they are declared.
b. Actual arguments reside in the function that did the invocation (call). Their values are passed to the called function through the formal parameters.
c. The header of a function includes the function return type, the function name, and the formal parameter list enclosed in parenthesis.
d. The signature of a function includes the function name and the formal parameter list enclosed in parenthesis.
e. The function prototype includes the function return type, the function name, and the formal parameter list enclosed in parenthesis. The prototype ends with a semi-colon.
f. Function declarations (prototypes) provide information to the compiler to aid in the setup for a function call.

4. Declaration Syntax

There are two ways to code a function declaration:

```
Function_return_type   Function_name (Type1 FF1, Type2 FF2, …, TypeN FFN);
```

or the alternative syntax

```
Function_return_type   Function_name (Type1, Type2, …, TypeN);
```

Note:

a. Type1, Type2, …, TypeN refer to the data types of the formal parameters.

b. FF1, FF2, …, FFN refer to the names of the formal parameters
c. All declarations end with a ";".

More information on functions can be found in your course textbook and on the web.

5. Experiments

Question 1: What are the differences between the two function declarations shown above?

Question 2: Based on our previous lab, we learned about data types and the size of space allocated in memory [for instance an **int** may take up 4 bytes of space]. Based on this knowledge and different ways to write a prototype, what observations can you make regarding how a compiler "knows" how much space to allocate?

Question 3: What specific information does the compiler need to perform a successful function call?

Question 4: Write the header and signature for the following function. Also write the two forms of its prototype.

```
double Print_Message(double balance, double limit)
{
	if (limit > 0)
	{
		cout<<"Your Balance is $"<<balance<<endl;
		cout<<"Your have money.  Proceed with your transaction\n";
	}
	else
	{
		cout<<"Your Balance is $"<<balance<<endl;
		cout<<"Your transaction has been cancelled\n";
	}

	return balance
}
int main()
{
	double balance = 99.99, limit = 2000.22;

	cout<<"Balacne and Limt have the following values before Print_Message is
called"<<endl;
	cout<<"balance= "<<balance<<" and limit = "<<limt<<endl;
	Print_Message(balance, limit);
	cout<<"a and b have the following values after swap is called"<<endl;
	cout<<"Balance = "<<balance<<" and Limit = "<<limit<<endl;
	return 0;
}
```

Function Overloading

Lab Sections

1. Objectives
2. Introduction
3. Experiments

Function Overloading

1. Objectives

After you complete this experiment you will be able to implement function overloading in a C++ program.

2. Introduction

Function overloading occurs when two or more functions have the same name but different formal parameter lists. The compiler only uses the function signatures to identify a function in function overloading. Never consider the function return type.

More information on function overloading can be found in your course textbook and on the web.

3. Experiments

Step 1: In this experiment you will learn how to implement function overloading.
Enter, save, compile and execute the following program in MSVS. Call the new project "FunctionOverloadingExp" and the program "FunctionOverloading.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;
void swap(string &a, string &b)
{
	string temp = a;

	a=b;
	b=temp;
}

void swap(int &a, int &b)
{
	int temp = a;

	a=b;
	b=temp;
}

void swap(char &a, char &b)
{
	char temp = a;

	a=b;
	b=temp;
}
```

```
int main()
{
	string x="111", y="222";
	char r='1', s='2';
	int a=111, b=222;

	cout<<"original strings: x = "<<x<<" y = "<<y<<endl;
	cout<<"original integers: a = "<<a<<" b = "<<b<<endl;
	cout<<"original characters: r = "<<r<<" s = "<<s<<endl;

	swap(x,y);
	swap(r,s);
	swap(a,b);

	cout<<"swap with strings: x = "<<x<<" y = "<<y<<endl;
	cout<<"swap with integers: a = "<<a<<" b = "<<b<<endl;
	cout<<"swap with characters: r = "<<r<<" s = "<<s<<endl;

	return 0;
}
```

Question 1: Please execute the program in Step 1 and explain how function overloading was used?

Question 2: Without changing the original program, would the following function cause an error if added in the source file that contains the program in Step 1? If so, list the error message(s), and explain why the error(s) occurred.

```
double swap(string &a, string &b)
{
	string temp = a;

	a=b;
	b=temp;

	return 0;
}
```

Function Calls

Lab Sections

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

Function Calls

1. Objectives

After you complete this experiment you will be able to:

a. **Understand input and output parameters**
b. **compare and contrast call-by-value and call-by-reference**

2. Introduction

During a function call memory is allocated for the formal parameters and local variables. Memory is also allocated for the address of the instruction to return to after the function has executed. There are two ways arguments are passed to a function during a function call, call-by-value and call-by-reference. Being able to call functions has many advantages which include: 1) allows us to write complex programs; 2) allows the user to decompose a program into smaller manageable components; 3) makes it easier to find errors; 4) makes it easier to maintain the program.

3. Definitions & Important Terms

We will define several terms that you need to know to understand function calls. They are as follows:

1. **Actual arguments** are the arguments used in the function call statement.
2. **Formal parameters** are the parameters used in the function header.
3. During a **call-by-value** a copy of the actual argument is made and placed into its corresponding formal parameter.
4. During a **call-by-reference** the address of the actual argument is copied into the formal parameter, when pointers are used.
5. During a **call-by-reference** the formal parameter is an alias for the actual argument when a reference is used in C++.
6. A formal parameter is called an **output parameter** when it is passed-by-reference. A formal parameter is called an **input parameter** if it is passed-by-value
7. If the value of a formal parameter is changed inside a function, and if this changed value is needed outside the function, it should be labeled as an output parameter or as a parameter that is passed by reference.
8. If the value of a formal parameter is only used inside a function, it should be called an input parameter or a parameter that is passed by value.
9. A **prototype** is a function declaration.
10. **References** are implemented internally as pointers

4. Declaration Syntax

```
return_type  function_name(formal_parameter_list);
```

Example:

```
void input(int &a, double c, char * m);
```

Explanation: a and m are passed-by-reference, and c is passed-by-value.

More information on function calls can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate the call-by-value mechanism.
Enter, save, compile and execute the following program in MSVS. Call the new project "FunctionCallsExp1" and the program "FunctionCalls1.cpp". Answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

void swap(int a, int b)
{
	int temp = a;
	cout<<"a and b have the following values at the start of swap"<<endl;
	cout<<"a = "<<a<<" and b = "<<b<<endl;
	a=b;
	b=temp;
	cout<<"a and b have the following values at the end of swap"<<endl;
	cout<<"a = "<<a<<" and b = "<<b<<endl;

}
int main()
{
	int a=111, b=222;

	cout<<"a and b have the following values before swap is called"<<endl;
	cout<<"a = "<<a<<" and b = "<<b<<endl;
	swap(a,b);
	cout<<"a and b have the following values after swap is called"<<endl;
	cout<<"a = "<<a<<" and b = "<<b<<endl;
	return 0;
}
```

Question 1: Are the variables a and b used in the function main the same a and b variables that are used in the function swap in the program in Step 1? Explain your answer.

Question 2: Were the values stored in the variables a and b in the function main swapped after the execution of the function swap in the program in Step 1? Explain your observations.

Step 2: In this experiment you will investigate the call-by-reference mechanism.

Execute the following program and answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

void swap(int & a, int & b)
{
	int temp = a;
	cout<<"a and b have the following values at the start of swap"<<endl;
	cout<<"a = "<<a<<" and b = "<<b<<endl;
	a=b;
	b=temp;
	cout<<"a and b have the following values at the end of swap"<<endl;
	cout<<"a = "<<a<<" and b = "<<b<<endl;

}
int main()
{
	int a=111, b=222;

	cout<<"a and b have the following values before swap is called"<<endl;
	cout<<"a = "<<a<<" and b = "<<b<<endl;
	swap(a,b);
	cout<<"a and b have the following values after swap is called"<<endl;
	cout<<"a = "<<a<<" and b = "<<b<<endl;
	return 0;
}
```

Question 3: Are the variables a and b used in the function main the same a and b variables that are used in the function swap in the program in Step 2 (hint: are they pass by value or by address)? Please explain your answer.

Question 4: Were the values stored in the variables a and b in the function main swapped after the execution of the function swap in the program in Step 2? Explain your observations.

Step 3: In this experiment you investigate the call-by-reference mechanism that uses pointers.

Execute the following program and answer the questions below:

```
#include <iostream>
#include <string>

using namespace std;

void swap(int * a, int * b)
{
	int temp = *a;
	cout<<"the contents of the memories pointed to by a and b have the following
values at the start of swap"<<endl;
	cout<<"*a = "<<*a<<" and *b = "<<*b<<endl;
	*a=*b;
	*b=temp;
	cout<<"the contents of the memories pointed to by a and b have the following
values at the end of swap"<<endl;
	cout<<"*a = "<<*a<<" and *b = "<<*b<<endl;

}
int main()
{
	int a=111, b=222;

	cout<<"a and b have the following values before swap is called"<<endl;
	cout<<"a = "<<a<<" and b = "<<b<<endl;
	swap(&a,&b);
	cout<<"a and b have the following values after swap is called"<<endl;
	cout<<"a = "<<a<<" and b = "<<b<<endl;
	return 0;
}
```

Question 5: What is the purpose of the '&' and '*' symbols in the main and swap functions in the program in Step 3? Explain your answer.

Question 6: Are the variables a and b used in the function main the same a and b variables that are used in the function swap in the program in Step 3? Explain your answer.

Question 7: Were the values stored in the variables a and b in the function main swapped after the execution of the function swap in the program in Step 3? Explain your observations.

Question 8: What types of function calls (reference, value, etc.) were used in the programs in Steps 1, 2 and 3?

Using the Visual C++ Debugger

Acknowledgements: This document is modified from a Web page named *CSE/ENGR 142 Debugger Tips* written by staff at the University of Washington's Computer Science department. We're very grateful!

Use the debugger on the program **msvs_exp.cpp**. You'll need to use Visual C++ to create and name a new project directory, then save this C++ file into that directory, and then add the file to the project using the menu sequence Project->Add to Project->Files.

1. **What is a debugger?**
2. **How to start the debugger**
3. **How to stop the debugger**
4. **Breakpoints**
5. **What a debugger session "looks" like**
6. **Step Over/Into/Out**
7. **Use the variable window**
8. **Use the watch window**
9. **Displaying expressions**
10. **Contexts**
11. **Displaying chars and strings**
12. **Dereferencing pointers**
13. **Displaying arrays and structures**
14. **Possible problems getting the debugger to work**

---

1. **What is a debugger?** A debugger is a tool which helps a programmer find errors ("bugs") in a program. You can make the program stop when it reaches a particular statement: this is called a "breakpoint." While at a **breakpoint**, you can examine the values of your variables and investigate the state of your program. Your C++ program is visible in a window at this time. Then you can continue executing from the breakpoint, perhaps to another breakpoint. Along the way you can set "**watches**" to view the values of variables you are interested in.
2. **How to start the debugger:** The Visual C++ (VC++) debugger is built-in to the Visual C++ development environment. **You can start the debugger by selecting "Debug/Start Debugging"**. Alternatively you can press "F5" to start debugging also.

Before starting the debugger, you probably should set a breakpoint (see below), or your program will simply run to completion as usual. When you started the debugger and it's paused at certain line, your whole VC++ window takes on a totally different look, as you'll see in a minute.

3. **How to stop the debugger:** During the debugging process, you can stop it by selecting "**Debug\Stop Debugging**". Alternatively you can press "Alt+F5" to stop debugging.

4. **Breakpoints:** If you want to stop the program execution at a particular line, use the mouse to left click on that line. When you see a red circle showing up next to that line, you have successfully put a breakpoint there. See the figure below.
5. If you ever want to remove a breakpoint, put the mouse cursor on that line and left click.

   Some subtle things you may need to know: You **cannot** put a breakpoint on a line that **only** defines a variable or function; put it on the first line afterwards that assigns or "executes" something. When you create a breakpoint while the program is running in debug mode, it is not actually activated until the next time the program reaches that statement.

6. **What a debugger session "looks" like:** Your session should look similar to the image of the debugger session shown below. You'll see a big window with the code of the current file (with breakpoints shown). Below that, you'll see the *variable window* on the left, and the *watch window* on the right. Right above the variable window you'll see a textbox labeled *Context* with the name of the current function, *main()*. Finally, notice there's a new set of icons up in the top-right window that look like this:
   More on these later.
7. **Step Over/Into/Out:** One basic use of the debugger is for you to set one or more breakpoints, then when the program pauses at that line, make the program execution continue one line at a time while you watch what happens. We call this ***single-stepping***. You can tell which line is being executed because VC++ puts a yellow arrow next to the line. Debuggers usually provide several commands for this; in VC++ they are:
   - **Step Into:** executes the next line. If this is a function call, go to the next line **inside** the called function.
   - **Step Over:** executes the next line, **without** entering into a called function. Stay in the same function.
   - **Step Out:** Used when inside a function; this executes all the remaining lines in the function, and then stops at the end of the function.

When the program is paused (**at a breakpoint)**, you access these functions from the Debug menu at the top of your window.

Again, when single-stepping a program, and the next statement to be executed is a function call, "**Debug/ Step Over**" will execute the entire function, whereas "**Debug/Step Into**" will step you **into** the function. Use "**Step Into**" for your own not-quite-debugged functions, and "**Step over**" for system functions like printf, or debugged functions of your own. To run a function until its end and then return to the caller, click "**Debug/Step Out**."

If you have stepped as far as you want and now you want the entire program to run to its normal completion, you can click "**Terminate All**" and then click "**Debug/Start Debugging** "to finish execution. (Or, you can click the same button that started the debugger running to do this.) There is also a handy "Run to Cursor" command that will execute all lines between the current line and the line where you've clicked the mouse. As for all of these commands, you can see the function-key and button short-cuts for doing "Step Into","Step Over" and "Step Out" and the others by looking at the "Debug" menu. For example, F8 can be used for "Step Into".

Really useful hint: In C++, often functions you didn't write get called automatically. For example, in the example shown, the function call code `pr_message("Hello world!")` actually calls a string class constructor to convert a string literal into a string object. If you use "Step Into" on this line, you'll unexpectedly find yourself in a function you didn't write and may not know anything about. **Solution:** Just execute "Step Out" to finish up that function and get back to code you've written.

8. **Use the variable window:** The variable window will display variable values local to a function. (If it's not visible, then select "View/Variables" will pop up the watch window.) As you single-step your program, you'll see the values change - very useful! There are three tabs in the lower left corner of the variable window. You can click on different tabs to display different information:
    - The Auto tab displays information about variables used in the current statement and the previous statement.
    - The Locals tab displays information about variables local to the current function.
    - The Threads tab will not be discussed
    - The Modules tab will not be discussed
9. **Use the watch window:** Watch window displays information about the variables that you are interested in. (If it's not visible, then select "View/Watch"

will pop up the watch window.) You can enter variable names in the "**Name**" column of the watch window, the debugger fills in
the **Type** and **Value** columns with the corresponding information as your program runs. You use the watch window to focus only on the variables you're interested in.
To remove a variable from the watch window, just use the mouse to high-light the variable's name in the "**Name**" column and hit backspace or delete to erase the name.

10. **Displaying expressions:** You can display more than just simple variables in the watch window. You can enter expressions there, and their values will be displayed. This can be useful, for example, in displaying the values of logical expressions controlling `if` statements and loops.
11. **Displaying chars and strings:** If you declare a variable of type `char` or `char *`, Visual C++ will display the variable value as well as the charater or character string it points to in the "Value" column.
12. **Dereferencing pointers:** If you have a pointer variable in the watch window, there will be a small button next to the variable name. By clicking on the button, you can see the memory location it points to in the "Value" column.
13. **Displaying arrays and structures:** To display the contents of an array or struct, enter the variable name into the watch window as usual. You will see a small button appears next to the variable name. By clicking on the button, you can expand or contract your view of the variable. The button displays a plus sign (+) when the variable is displayed in contracted-form, and a minus sign when it is displayed in expanded form.
14. **Possible problems getting the debugger to work:** If you find the debugger just won't start for you, here are several things you can try. (But these trouble-shooting tips are from an older version of VC++.)
    - First, you need to be sure that the active configuration is set properly. Pull down the "**Build**" menu and select "**Configuration Manager**." Be sure that "**Active Solution Configuration"** is set to "**Debug**" and "**Active solution platform**" is set to Win32. Now, re-compile your program. You should now be able to use the debugger.
    - When you wish to debug, be sure that you either start the program with Build -> Debug -> Start Debugger, or by hitting F5. If you start the program by selecting Build -> Start Without Debugging any breakpoints will not be recognized and the debugger will not work.

Laboratory Questions

Directions: Copy the program "msvs_exp.cpp" into Microsoft Visual Studios. Then follow the steps below and answer the following questions.

1. **Question 1:** Place a breakpoint at the line "int x = 1, y = 1;" and run the program using Debug, Start Debugging. The program stops at the first breakpoint. What are the values of x and y at this point in the program?

2. **Question 2:** Use the Step Over command once. What now are the values of x and y at this point in the program?

3. **Question 3:** In your own words, explain why the change in the values of x and y that occurred in Step 2 was not visible at the breakpoint in Step 1.

4. **Question 4:** The yellow arrow should be pointing to the line "pr_message("Hello world!");". Use the Step Into command to enter this function. Explain what you see.

5. **Question 5:** Now use the Step Out command to return to "pr_message("Hello world!");". What is the command you used for this action?

6. **Question 6:** Now in the main function, place a breakpoint on the line "x = y / x;". Keep an eye on the window as you continously use the Step Over command until you reach this breakpoint. What line in the program sets the denominator to cause the divide-by-zero error?

7. **Question 7:** Based on your answer in Question 6, correct the code to allow the program to run to completion.

Introduction to Unix

Lab Sections

1. Objectives
2. Introduction
3. Experiments

Introduction to Unix

1. Objectives

After you complete this experiment you will be able to:

a. **log into unix;**
b. **log out of unix;**
c. **display the content of a file to the screen;**
d. **edit a file using pico;**
e. **list the contents of a directory;**
f. **copy the contents of a file to another file;**
g. **delete a file for a directory;**
h. **change the working directory;**
i. **compile a C++ program using g++;**
j. **print the current/working directory;**
k. **determine individuals who are currently logged on;**
l. **print the date and time;**
m. **create a subdirectory;**
n. **use the "man" command to get information.**

2. Introduction

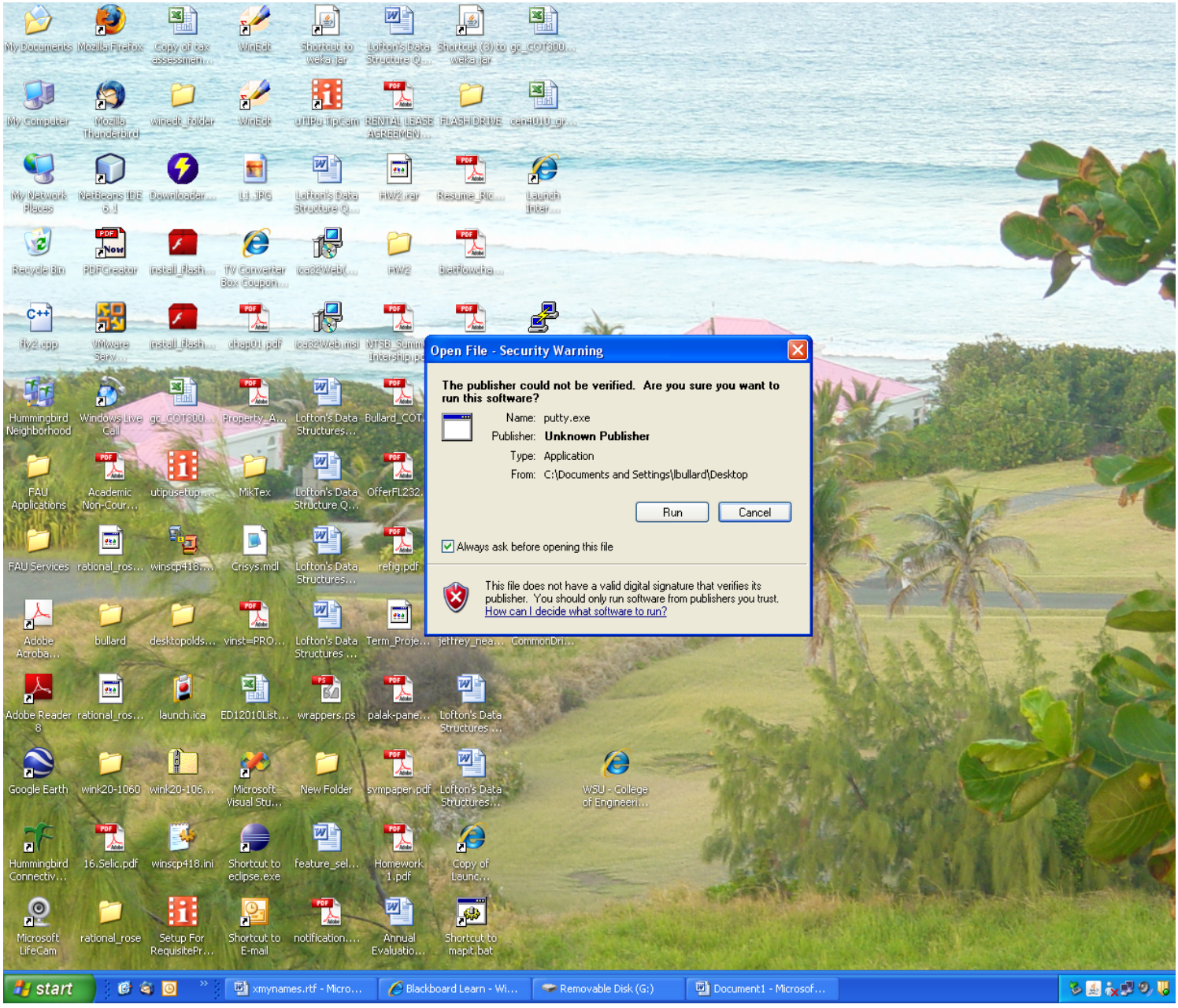
Unix is an operating system that was create by Bell Labs in 1969. Many internet applications were written using Unix. An operating system allocates resources to the processes that are executing on a computer. The resources can hardware and software that is managed by the operating system. There are many distributions (versions) of Unix, and it remains one of the most used operating systems. This is due to the fact of the many advantages it provides which include openness, portability, multiuser environment, multitasking, networking, and prevalence.

More information on Unix can be found on the web. A Unix Quick Reference Sheet is also provided with this laboratory. See the files unix.pdf and vi.pdf.

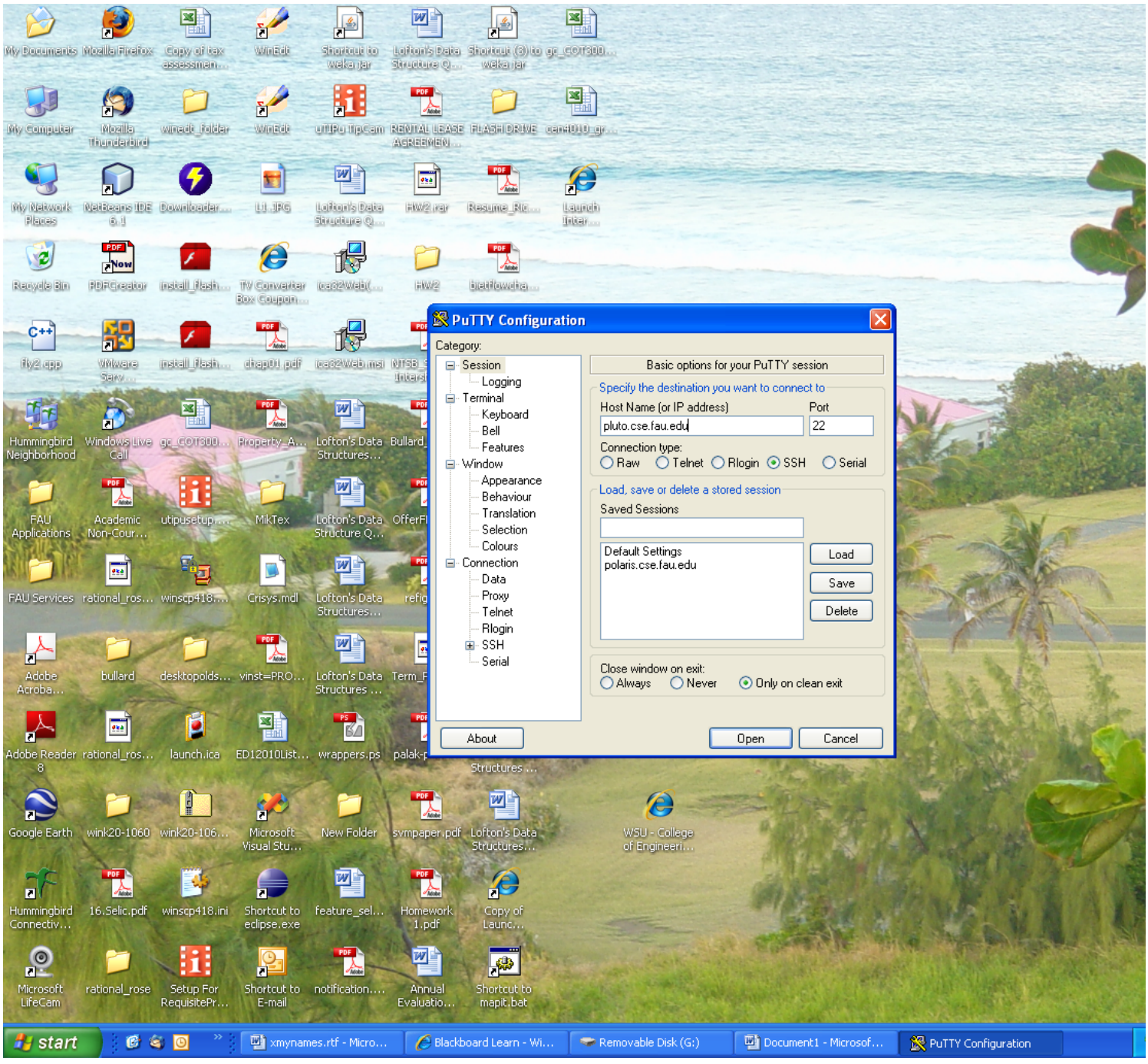
3. Experiments

Step 1: Locate the "putty.exe" icon on the desktop of your computer. Double click on it. If you can't see it, ask your TA for help.

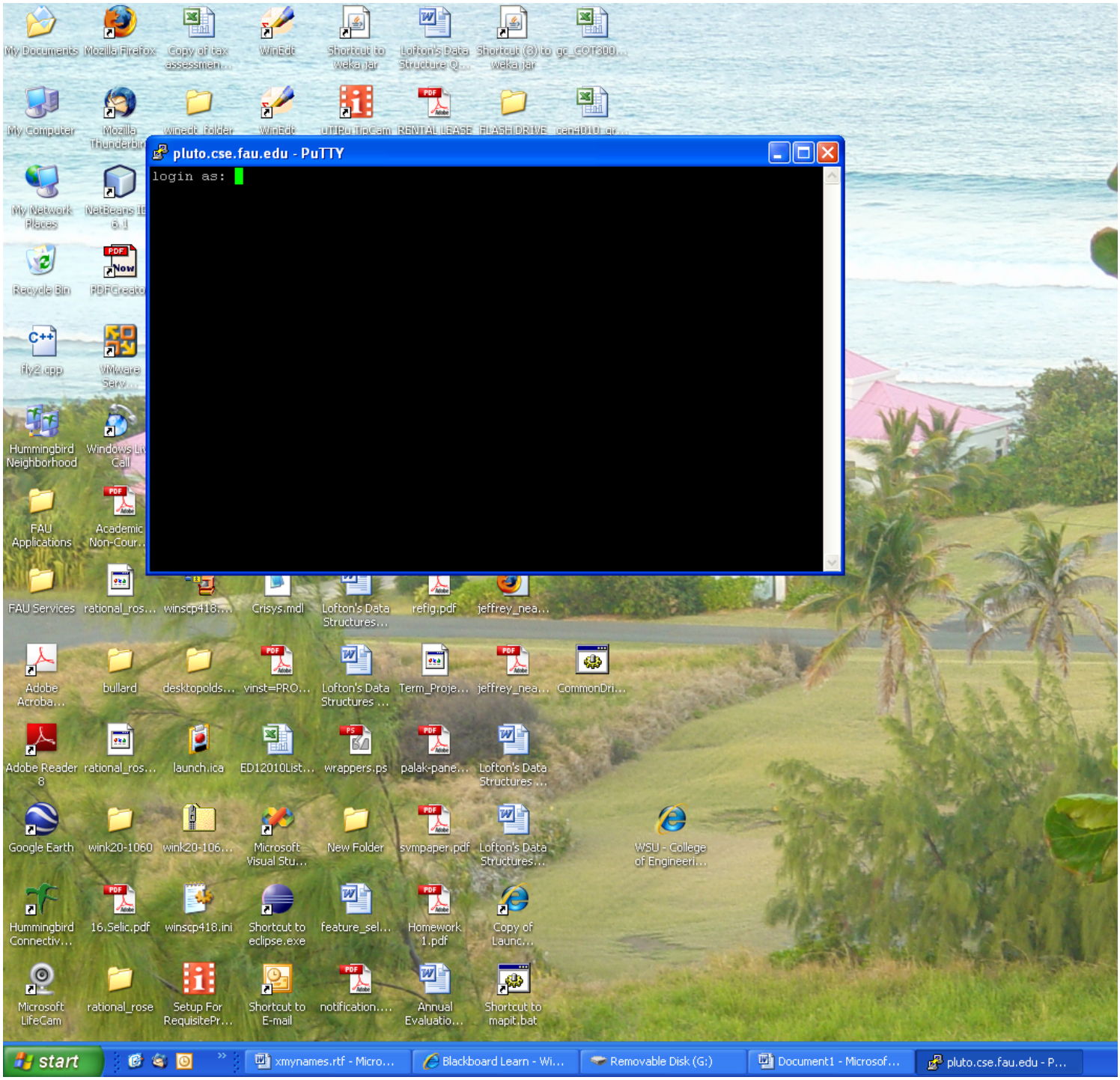
Step2. After you click on the putty.exe icon, your screen should look similar to the one below. Click run.



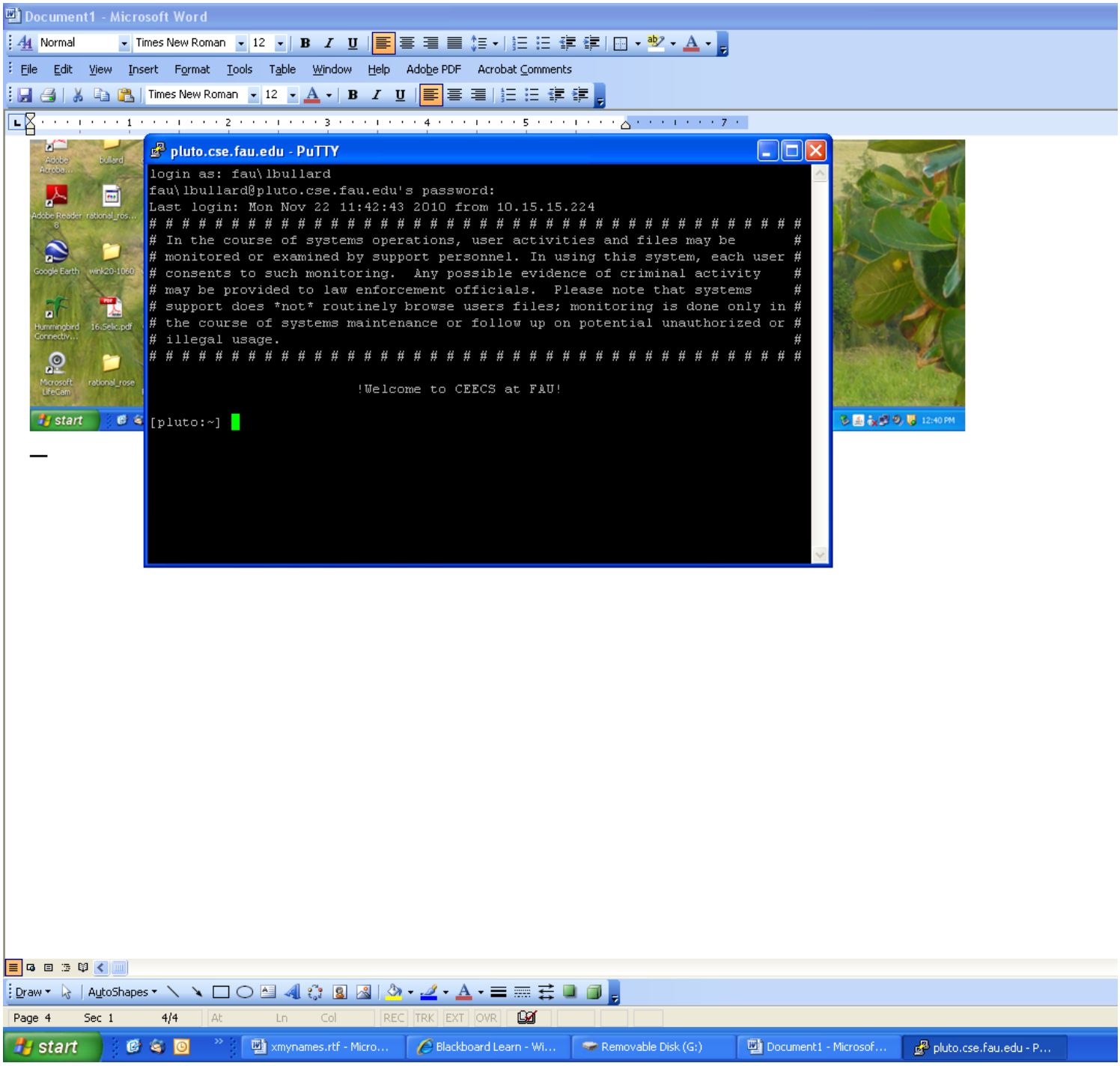
Step 3: Your screen should look like the one below. For Host Name enter "pluto.cse.fau.edu". For Connection type choose SSH. Click on Open.



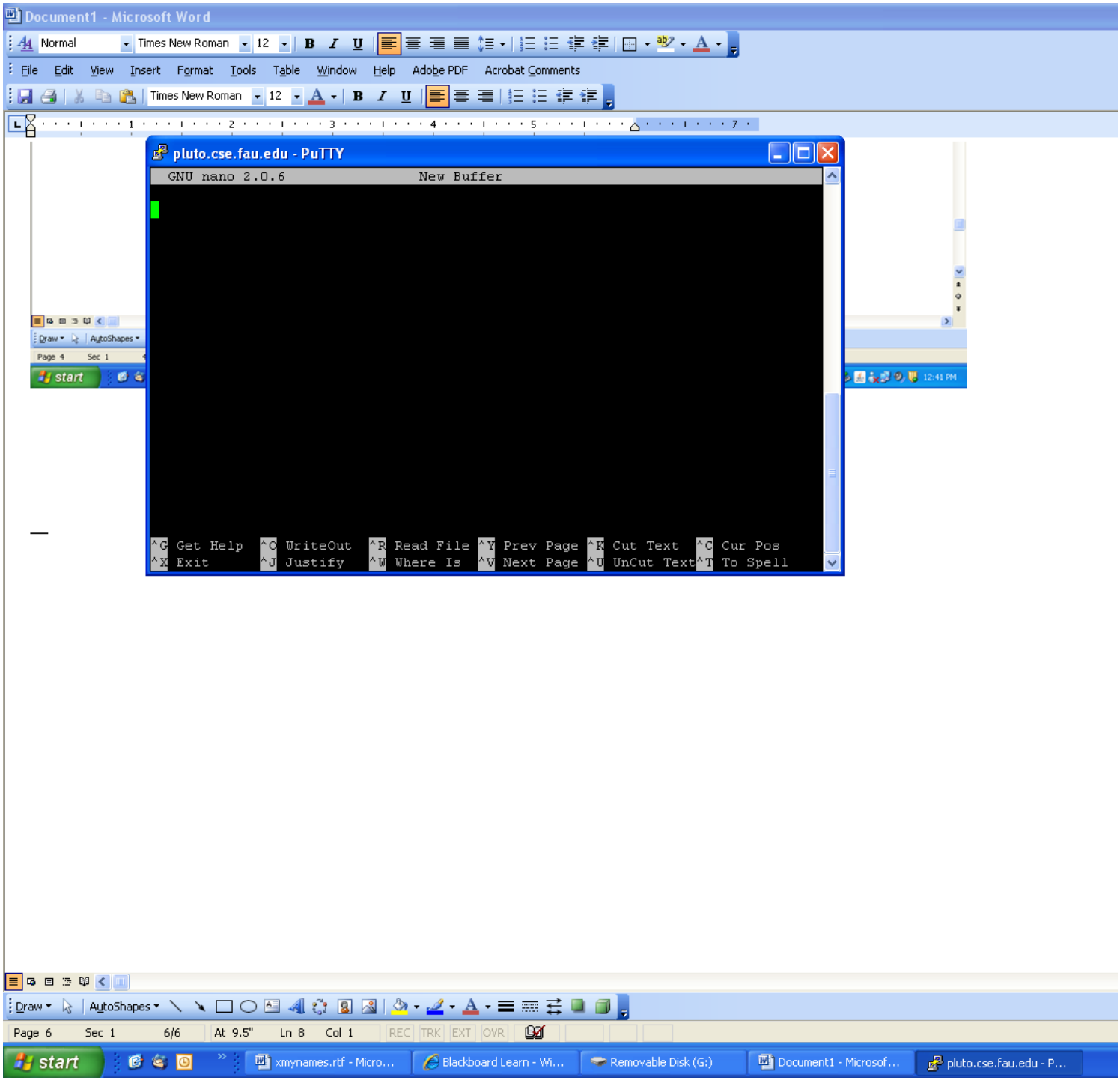
Step 4: **Now you are ready to log into unix. Your unix screen should look like the one below. You should type fau\your_FAU_id for the username. Press enter and type your password. If you are having difficulties ask your TA for help.**



Step 5: Congratulations you have successfully logged into Pluto. Your screen should look similar to the one shown below.



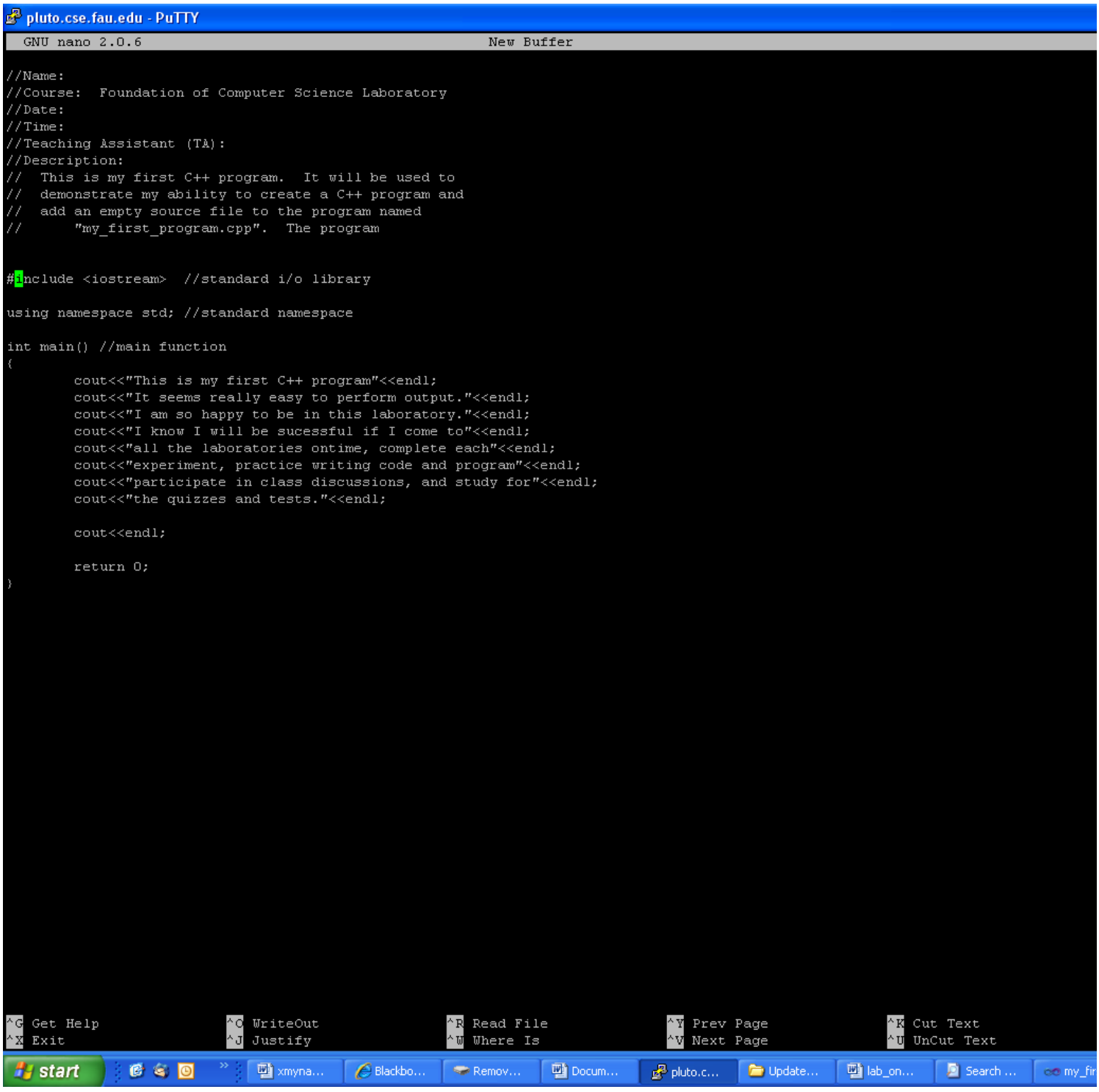
Step 6: From your unix command line prompt, enter the command "pico myFirstProgram.cpp". Your screen should look like the one below. If you are having any problems please ask your TA for help.



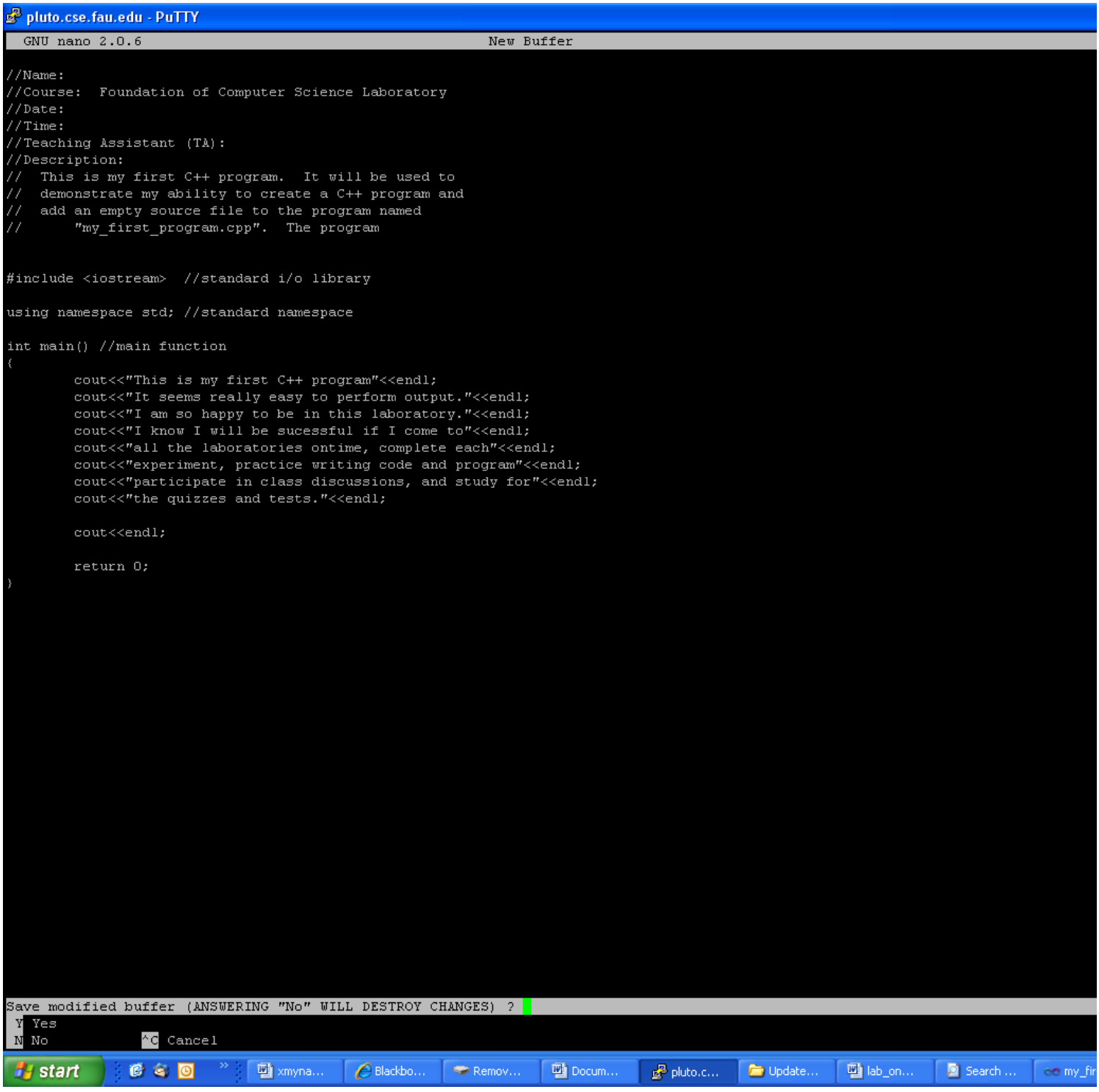
Step 6: Type the following program into the pico editor.

```
//Name: Firstname Lastname
//Course:  Foundation of Computer Science Laboratory
//Date: Today date
//Time: The current time
//Teaching Assistant (TA): Your TAs name
//Description:
//  This is my first C++ program.  It will be used to
//  demonstrate my ability to create a C++ program and
//  add an empty source file to the program named
//  "my_first_program.cpp".  The program


#include <iostream>  //standard i/o library

using namespace std; //standard namespace

int main() //main function
{
       cout<<"This is my first C++ program"<<endl;
       cout<<"It seems really easy to perform output."<<endl;
       cout<<"I am so happy to be in this laboratory."<<endl;
       cout<<"I know I will be sucessful if I come to"<<endl;
       cout<<"all the laboratories ontime, complete each"<<endl;
       cout<<"experiment, practice writing code and program"<<endl;
       cout<<"participate in class discussions, and study for"<<endl;
       cout<<"the quizzes and tests."<<endl;

       cout<<endl;

       return 0;
}
```

Step 7: When you have finished entering the program into pico, your screen should look like the one shown below.



Step 8: Now press Ctrl X to exit, press Y to save, and save the program titled "myFirstProgram.cpp", then hit enter. Your screen should look similar to the one below.



Step 9: From the unix prompt, enter the "ls" command to get a directory listing. Notice how the program "myFirstProgram.cpp" appears in the directory listing.

Step 10: Now enter the command "ls -l" to get a long directory listing.

Question 1: How does this listing differ from the one you observed in Step 9?

Question 2: Do you know the meaning of each field/column in the listing? If not, try using Google to find the field names.

Step 11: g++ is the name of the GNU C++ compiler. Enter the command "g++ myFirstProgram.cpp". Do you get any compiler errors?

If you observe any compiler errors, you must correct them before you proceed. Load your program into pico and correct the compiler errors. Ask you TA for help if you are confused.

Step 12: Now that your program has compiled without any errors, enter the command "./a.out".

Question 3: Can you describe the output you observe?

Question 4: Do you know what "a.out" represents?

Step 13: Enter the command "g++ -o first myFirstProgram.cpp". Then enter the command "ls –l".

Question 5: Can you explain the directory listing you observe?

Question 6: Do you see the file called "first"?

Step 14: Enter the command "./first".

Question 7: Can you explain what you observed?

Step 15: Enter the command "cat myFirstProgram.cpp".

Question 8: Can you explain what you observed?

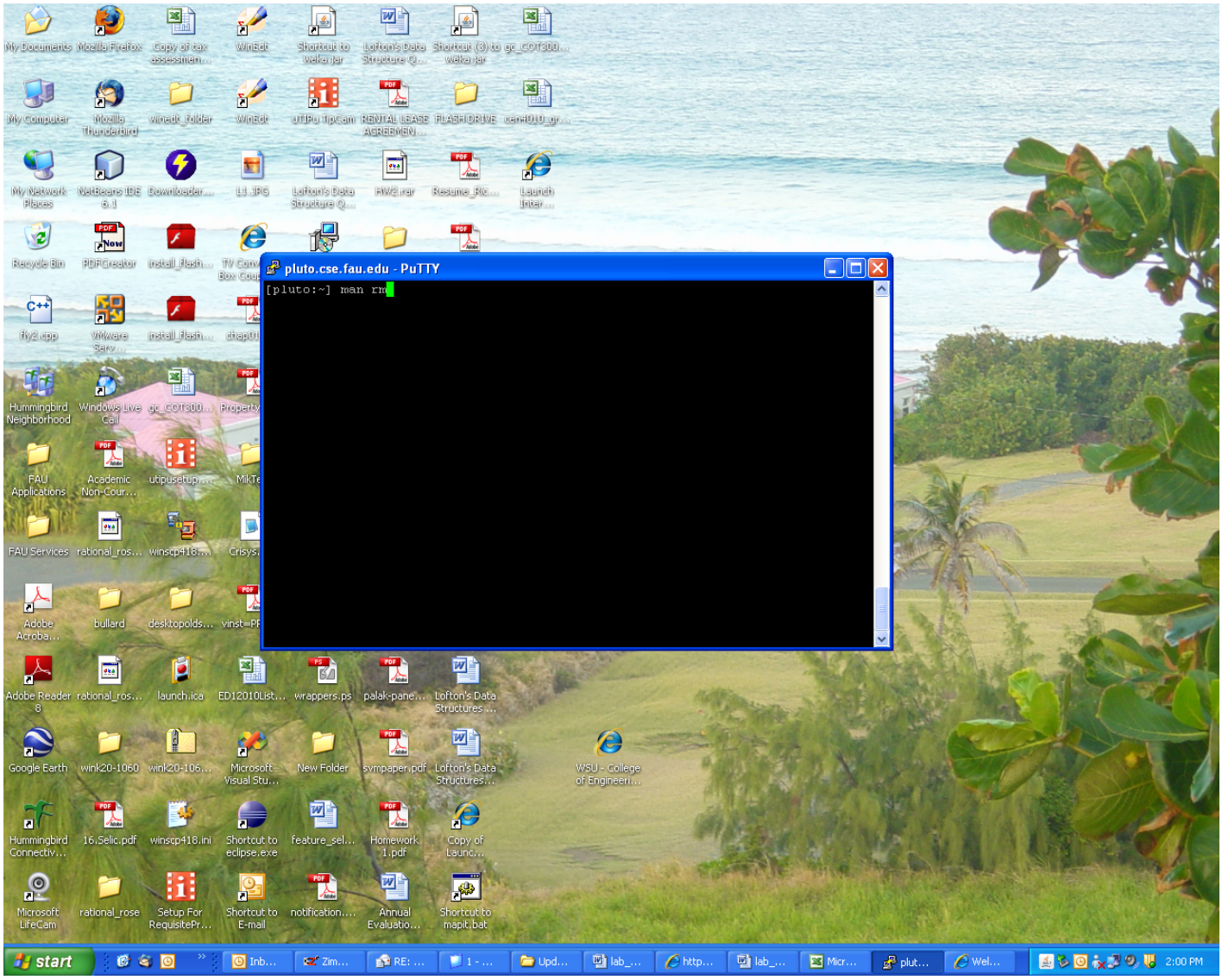
Step 16: Enter the command "rm a.out"; then enter "ls –l".

Question 9: Can you explain what you observed?

Question 10: What is the purpose of the rm command ?

Step 17: Enter the command "man rm". To move the cursor up press "u"; to move the cursor down press "d". To exit press "q". To page forward press "enter".

Question 11: Do you have any observations you would like to share?



Step 18: Enter the command "cp myFirstProgram.cpp mySecondProgram.cpp". Now enter the command "cat mySecondProgram.cpp".

Question 12: What is the purpose of the "cp" command?

Step 19: Enter the command "mkdir COP3014L". Then enter the command "ls –l".

Question 13: What did you observe?

Question 14: What is the purpose of the "mkdir" command?

Step 20: Enter the command "cd COP3014L". Then enter the command "ls –l".

Question 15: What did you observe?

Question 16: What is the purpose of the "cd" command?

Step 21: Enter the command "cd". Then enter the command "ls –l".

Question 17: What did you observe?

Question 18: What is the purpose of the "cd" command used by itself?

Step 22: Enter the command "cd ~". Then enter the command "ls –l".

Question 19: What did you observe?

Question 20: What is the purpose of "cd ~"?

Step 23: Enter the command "cd ..". Then enter the command "ls –l".

Question 21: What did you observe?

Question 22: What is the purpose of "cd .."?

Step 24: Enter the command "pwd".

Question 23: What did you observe?

Question 24: What is the purpose of "pwd"?

Step 25: Enter the command "who".

Question 25: What did you observe?

Question 26: What is the purpose of "who"?

Step 26: Enter the command “date”.

Question 27: What did you observe?

Step 27: Go back into your home directory (cd ~) and Enter the command “cat first”.

Question 28: What did you observe?

LAB: Advanced MSVS Debugger example using a real-life network failure

Some of the text and code used in this exercise was **borrowed *Deep C Programming: Deep C Secrets*, by Peter Van Der Linden (SunSoft Press, Prentice Hall, 1984), pp. 38-39.** The program that will be used in this exercise is called "**att_bug.cpp**", which is an included document for the experiments of this lab.

The code is based on a program that caused a major disruption of AT&T phone service throughout the U.S. This failure made national news at the time and is one of the best-known software failures.

AT&T's nationwide network was mostly unusable for about nine hours starting on the afternoon of January 15, 1990. Telephone exchanges, also known as "switching systems", are computer systems that allow multiple phone calls to be transferred concurrently over one wire. This type of software was running on the hardware controlling the AT&T network. The failure of the network was due to the bug that will be illustrated in this example; The bug led to the first major network problem in AT&T's 114-year history. The bug caused a problem that led to a chain-reaction that spread across the network, bringing down AT&T's entire long distance network. More information on the bug can be found in the January 22, 1990 issue of *Telephony* magazine.

You should be able to understand the logic of the code (**att_bug.cpp**). Certain **values are used to call various functions which initialize** certain **variables to make the system execute correctly**. Different initialization functions are called depending on the values of the initialized variables. A **switch** statement, **if** statements, and **break**statements are used to implement the correct initialization logic. In this simplified example, all **you need to know is that both global integer variables, val1** and **val2**, **in the program must be set to non-negative values before the function** `route_that_call()` is called.

Testing Section:

To use the code in this exercise, simply compile and execute it, and enter 3 integer values. The program will print a message stating whether "the phone system" is OK or crashes for the 3 integer values you entered. The program will repeatedly ask for 3 more values until it gets to the end of file or invalid inputs.

Step 1: Compile and execute the program, att-bug.cpp, with various inputs and see what happens.

Step 2: Study the logic of the code. Answer the following questions:

Question 1: For what values does the function `init_values_x1()` get invoked?

Question 2: For what values does `init_values_part1A()` get called?

Question 3: For what values does `init_values_part1B()` get called?

Question 4: For what values does `init_values_part2()` get called?

Question 5: For what values does `init_default()` get called?

Step 3: Find values for i, x and y that make the phone network crash? Hint: Try each set of data values you identified in Step 2 to make the program perform each of the possible initializations. Together these sets of values are good test data, since each one causes the program to behavior differently. They cover all the behavior caused by the different initialization values.

Question 6: What set of 3 values (**i,x,y**) caused the phone network to crash?

Debugger Section:

Now that you know what data is causing the crash, you may still not know why those data values cause the problem. In the steps that follow you will use the **MSVS 2010 debugger** and answer the following questions. **Build** the program with the **Build Solution option** and then carry out the following steps.

Step 4: Put a breakpoint at the start of the function, `phone_network()`. Execute the program, by choosing "Debug" and then "Start Debugging".

Question 7: After entering the 3 values, what is displayed when the execution of the program stops at the breakpoint you set at the line that contain the name of the function?

Step 5: Type "Ctrl-F10" to resume execution.

Question 8: Does the program stop at the breakpoint again?

Question 9: How do you remove the breakpoint you set?

Step 6: If you removed the breakpoint, put it back. Execute the program again. When you reach the breakpoint, choose "F11". This executes the program one line at a time. Execute "F11" a few times to see how "F11" works. Type "Ctrl-F10" to continue on until the breakpoint is reached again.

Now that you are back at the initial breakpoint again, do the same thing but this time choose "F10" instead of "F11" to go to the next line. Do this a few times to observe what happens.

***Question 10**:* What differences in the execution of the **MSVS** commands, **"F11"** and **"F10",** did you observe?

Step 7: Using the same breakpoint from Step 6, run the "F11" command to step "into" a function. Now enter "Shift-F11". The "Shift-F11" command terminates the execution of the function and pauses.

Question 11: Where does execution stop when you enter "**Shift-F11**"?

Step 8: Using the same breakpoint, stop the execution process by choosing "Stop Debugging" and then restart the execution process by choosing "Start Debugging".

Now, in the "Watch" window underneath the code window, enter the variables "val1" and "val2" per each row under Name. Once you've done this, run the "F10"command a few times by typing "F10" repeatedly. Then type "Ctrl-F10" to run the "continue" command.

Question 12: What does **MSVS** do with "**Watch**" variables?

Step 9: OK, about that bug.... Set a breakpoint at the line that contains `phone_network()` or at the switch statement. Enter the 3 data values for i, x, and ythat caused the phone network to crash. From the breakpoint, use the "F10" command to step through the program.

Question 13: Did you observe anything that surprises you about the behavior of the program? Was the execution of any lines skipped that you thought should have been executed?

Question 14: Can you explain the behavior you observed?

MS Visual C++ Debugger Reference Card

1. **Set/Remove Breakpoint: F9**
   This command sets/removes a breakpoint at the current cursor position.

2. **Start debugger: F5**
   This command starts your program under the debugger. Execution stops at the first breakpoint.

3. **Stop debugger: Shift-F5**
   (Works only when debugger is running).

4. **Run to cursor: Ctrl-F10**
   This command forces the debugger to execute your program until either current cursor position or a breakpoint are reached.

5. **Step over: F10**
   This command executes one statement treating functions as single steps (thus the name "step over").

6. **Step into: F11**
   Same as "Step Over", but steps into functions.

7. **Step out: Shift-F11**
   Steps out of the function you are currently in.

Scope of Function Names

Lab Sections

1. Objectives
2. Introduction
3. Experiments

Scope of Function Names

1. Objectives

After you complete this experiment you will be able to determine the scope of a function name.

2. Introduction

The scope is the region in a program where a name (identifier) has meaning. As you already know, all identifiers must be declared before they can be used

More information on scope can be found in your course textbook and on the web.

3. Experiments

Step 1: In this experiment you will discover the scope rules that are used in a program. Enter, save, compile and execute the following program in MSVS. Call the new project "FunctionNameScopeExp1" and the program "FunctionNameScope1.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

void Function_One()
{
	cout<<"You are in Function One"<<endl;
	cout<<"Function_One called no one"<<endl<<endl;
}

void Function_Two()
{
	cout<<"You are in Function_Two"<<endl;
	cout<<"Function_Two will call Function_One"<<endl<<endl;
	Function_One();
}

void Function_Three()
{
	cout<<"You are in Function_Three"<<endl;
	cout<<"Function_Three will call Function_Two"<<endl<<endl;
	Function_Two();
}

int main()
{
	Function_Three();
	Function_Two();
	Function_One();

	return 0;
}
```

Question 1: Does the program in Step 1 compile without any errors? Explain.

Question 2: We didn't use prototypes. When would the program in Step 1 need prototypes? Explain your answer.

Question 3: Please state a rule about **where** functions must be declared (ie. scope) when compared to the **location** of the function call that you observed in the program in Step 1?

Step 2: In this experiment you will discover the scope rules that are used in a C++ program. Enter, save, compile and execute the following program in MSVS. Call the new project "FunctionNameScopeExp2" and the program "FunctionNameScope2.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

void Function_One()
{
	cout<<"You are in Function One"<<endl;
	cout<<"Function_One will call Function_Two"<<endl<<endl;
	Function_Two();
}

void Function_Two()
{
	cout<<"You are in Function_Two"<<endl;
	cout<<"Function_Two will call Function_Three"<<endl<<endl;
	Function_Three();
}

void Function_Three()
{
	cout<<"You are in Function_Three"<<endl;
	cout<<"Function_Three calls no one"<<endl<<endl;
}

int main()
{
```

```
	Function_Three();
	Function_Two();
	Function_One();

	return 0;
}
```

Question 4: Does the program in Step 2 compile without any errors? If not, what are the error message(s)?

Question 5: Does the program in Step 2 need prototypes? Explain your answer.

Question 6: Please state a rule about scope that you observed in the program in Step 2?

Step 3: In this experiment you will discover the scope rules that are used in a program. Enter, save, compile and execute the following program in MSVS. Call the new project "FunctionNameScopeExp3" and the program "FunctionNameScope3.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

void Function_One();
void Function_Two();
void Function_Three();

void Function_One()
{
	cout<<"You are in Function One"<<endl;
	cout<<"Function_One will call Function_Two"<<endl<<endl;
	Function_Two();
}

void Function_Two()
{
	cout<<"You are in Function_Two"<<endl;
	cout<<"Function_Two will call Function_Three"<<endl<<endl;
	Function_Three();
}

void Function_Three()
{
	cout<<"You are in Function_Three"<<endl;
	cout<<"Function_Three calls no one"<<endl<<endl;
}
```

```
int main()
{

	Function_Three();
	Function_Two();
	Function_One();

	return 0;
}
```

Question 7: Does the program in Step 3 compile without any errors? If not, what are the error message(s)?

Question 8: What are the differences between the programs in Step 2 and Step 3?

Question 9: Please state a rule about scope that you observed in the program in Step 3.

Step 4: In this experiment you will discover the scope rules that are used in a program. Enter, save, compile and execute the following program in MSVS. Call the new project "FunctionNameScopeExp4" and the program "FunctionNameScope4.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

void Function_One(int);
void Function_Two();
void Function_Three();


void Function_One(int Function_Two)
{
	cout<<"You are in Function One"<<endl;
	cout<<"Function_One will call Function_Two"<<endl<<endl;
	Function_Two();
}

void Function_Two()
{
	cout<<"You are in Function_Two"<<endl;
	cout<<"Function_Two will call Function_Three"<<endl<<endl;
	Function_Three();
}

void Function_Three()
{
	cout<<"You are in Function_Three"<<endl;
	cout<<"Function_Three calls no one"<<endl<<endl;
}
```

```
int main()
{

	Function_Three();
	Function_Two();
	Function_One(5);

	return 0;
}
```

Question 10: Does the program in Step 4 compile without any errors? If not, what are the error message(s)?

Question 11: State a rule about scope that you observed in the program in Step 4.

Scope of Variable Names

Lab Sections

1. Objectives
2. Introduction
3. Experiments

Scope of Variable Names

1. Objectives

After you complete this experiment you will be able to determine the scope of a variable name.

2. Introduction

The scope is the region in a program where a name (identifier) has meaning. As you already know, all identifiers must be declared before they can be used.

More information on scope can be found in your course textbook and on the web.

3. Experiments

Step 1: In this experiment you will investigate the operation of scope rules in a program. Enter, save, compile and execute the following program in MSVS. Call the new project "VariableScopeExp1" and the program "VariableNameScope1.cpp". Answer the questions below:

```
#include <iostream>
using namespace std;

void Function_One();
void Function_Two();
void Function_Three(int);

int i=100;

void Function_One()
{
	cout<<"i in Function_One = "<<i<<endl;
}
void Function_Two()
{
	int i = 555;
	cout<<"i in Function_Two = "<<i<<endl;
}
void Function_Three(int i)
{
	cout<<"i in Function_Three = "<<i<<endl;
}
int main()
{
	int i = 777;
	Function_Three(666);
	Function_Two();
	Function_One();
	cout<<"i in main = "<<i<<endl;
	return 0;
}
```

Question 1: What are the scope rules you observed in the program in Step 1?

Step 2: In this experiment you will investigate the operation of scope rules in a program. Enter, save, compile and execute the following program in MSVS. Call the new project "VariableScopeExp2" and the program "VariableNameScope2.cpp". Answer the questions below:

```
#include <iostream>

using namespace std;

int i = 111;

int main()
{
	{
		int i = 222;
		{
			int i = 333;
			cout<<"i = "<<i<<endl;
			{
				int i = 444;
				cout<<"i = "<<i<<endl;
					{
						cout<<"i = "<<i<<endl;
					}
			}
		}

		cout<<"i = "<<i<<endl;
	}

	cout<<"i = "<<i<<endl;

	return 0;
}
```

Question 2: What are the scope rules you observed in the program in Step 2?

C++ Strings

Lab Sections

1. Objectives
2. Introduction
3. Definitions & Important Terms
4. Declaration Syntax
5. Experiments

Strings

1. Objectives

After you complete this experiment you will be able to use the standard string class

2. Introduction

Strings in C++ are objects of the string class. This means that the programmer does not need to manage the memory that a string uses as in c-style strings.

3. Definitions & Important Terms

Following is some preliminary information you need to know to understand the string class:

a. The string library includes all the functions necessary to use the string class.
b. The string class has a default, explicit-value and copy constructor.
c. The [ ] operator is used to access the members of a string
d. The index/subscript of a cell in a string must be between 0 and the length of the string minus 1.
e. The capacity of a string is the number of elements it can hold..

4. Declaration Syntax

Consider the following syntax when declaring strings in C:

```
a. string s1; //creates a empty string
b. string s2 = "hello"; //creates a string initialized to "hello".
c. string s3("hello"); //creates a string initialized to "hello";
                       //an explicit-valued constructor called.
d. string s4 = s2; //declaration and initialization; the copy constructor
                   //called.
```

More information on the string class can be found in your course textbook and on the web.

5. Experiments

Step 1: In this experiment you will investigate the declaration, initialization and implementation of C++ styles. Enter, save, compile and execute the following program in MSVS. Call the new directory "cPlusPlusStringsExp1" and the program "cPlusPlusStrings1.cpp". Answer the questions below:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
	string my_name = "Thomas Jefferson";
	string her_name("Michelle Obama");
	string his_name = my_name;

	cout<<"my_name = "<<my_name<<endl;
	cout<<"The length of my_name is "<<my_name.length()<<endl;
	cout<<"The size of my_name is "<<sizeof(my_name)<<endl<<endl;

	cout<<"her_name = "<<her_name<<endl;
	cout<<"The length of her_name is "<<her_name.length()<<endl;
	cout<<"The size of her_name is "<<sizeof(her_name)<<endl<<endl;

	his_name = "Barack Obama";
	cout<<"his_name = "<<his_name<<endl;
	cout<<"The length of his_name is "<<his_name.length()<<endl;
	cout<<"The size of his_name is "<<sizeof(his_name)<<endl<<endl;

	string their_name;

	their_name = my_name + " " + her_name
		+ " " + his_name + " " + my_name + " ";
	cout<<"their_name = "<<their_name<<endl;
	cout<<"The length of their_name is "<<their_name.length()<<endl;
	cout<<"The size of their_name is "<<sizeof(their_name)<<endl;

	return 0;

}
```

Question 1: Please explain why the size and the length values for each string are different (or the same) in the output produced by the program in Step 1 above?

Question 2: Why is the size the same for each string in the program in Step 1?

Question 3: Please write the statements from the program in Step 1 in which a constructor is used? Name each constructor.

Question 4: Please name the string operations that were preformed in the program?

Step 2: In this experiment you will investigate performance of the subscript and the relational operators on strings. Enter, save, compile and execute the following program in MSVS. Call the new directory "cPlusPlusStringsExp2" and the program "cPlusPlusStrings2.cpp". Answer the questions below:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
	string my_name = "Thomas Jefferson";
	string her_name("Michelle Obama");
	string his_name = "Joe Biden";

	for (int i=0; i < her_name.length(); i++)
	{
		cout<<her_name[i];
	}
	cout<<endl<<endl;

	if (my_name == "Thomas Jefferson")
	{
		cout << "my_name is equal to Thomas Jefferson"
			<< endl << endl;
	}

	if (my_name > her_name)
	{
		cout << my_name <<" is lexicographically greater than "
			<< her_name << endl << endl;
	}
```

```
        if (my_name < his_name)
        {
              cout << my_name <<" is lexicographically less than "
                   << his_name <<endl <<endl;
        }

        if (my_name >= her_name)
        {
              cout << my_name <<" is lexicographically greater than or equal to "
                   << her_name << endl << endl;
        }

        if (my_name <= his_name)
        {
              cout << my_name <<" is lexicographically less than or equal to "
                   << his_name <<endl <<endl;
        }

        return 0;

}
```

Question 5: What does the for loop do in the program in Step 2?

Question 6: Please explain the operation of each relational operator used in the program in Step 2?

Step 3: In this experiment you will investigate the performance of the find and substring functions. Enter, save, compile and execute the following program in MSVS. Call the new directory "cPlusPlusStringsExp3" and the program "cPlusPlusStrings3.cpp". Answer the questions below:

```
#include <iostream>
#include <string>
using namespace std;

int main()
{
	string s1 = "123abc456de111ef789eehij111ee";
	string s2;
	int loc = 0;

	for (int i =0; i<s1.length(); i++)
	{
		loc = s1.find("1",loc);
		if (loc != -1)
		{
			cout<<"1 appears at location "<<loc<<endl;
		loc++;
		}
	}
	cout << endl << endl;

	s2 = s1.substr(0,3);
	cout << "s2 equals " << s2<<endl;

	s2 = s1.substr(3,3);
	cout << "s2 equals " << s2<<endl;

	s2 = s1.substr(0,500);
	cout << "s2 equals " << s2 << endl << endl;


	loc = 0;
	s2 = "";

	for (int i =0; i<s1.length(); i++)
	{
		loc = s1.find("e",loc);
		if (loc != -1)
		{
			s2+= s1.substr(loc,1);
			loc++;
		}
	}
	cout << "s2 equals " << s2 << endl <<endl;

	return 0;

}
```

Question 7: What compiler warnings were given?

Question 8: What does the first **for** loop perform in the program in Step 3? Explain by observing the output of the program.

Question 9: Please explain **the first three statements** that use the substr function?

Question 10: Please explain the last **for** loop in the program?