**Lab Sections**

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

|  |
| --- |
|  |

Classes:

Copy Constructor

**Classes: Copy Constructor**

1. **Objectives**

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

1. **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:

1. when an object is passed-by-value;
2. when an object is returned in a return statement;
3. when an object appears in a declaration/initialization statement.

1. **Definitions & Important Terms**

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

1. A **shallow copy** is a **dependent bit-by-bit** copy of an object. The objects involved will share the same dynamic memory;
2. A **deep copy** is an **independent** copy of an object. The objects involved will have their own copy of dynamic memory;
3. The **copy constructor** is used to make a deep copy of an object;
4. When an object does not use dynamic memory a copy constructor does not need to be implemented;
5. 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;
6. 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 (~).
7. The **current object** is the object that called the copy constructor;
8. **this** is a pointer to the current object;
9. The **object that is being copied (source object) is passed explicitly** through the corresponding formal parameter of the copy constructor;
10. The **object that is being copied into (destination/current object) is passed implicitly through the “this” pointer**.
11. **Declaration Syntax**

Consider the following:

1. Class\_name(const Class\_name &); //prototype inside class declaration
2. 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.

1. **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;

}

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.

1. 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;

}

1. 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.
2. Why was a copy constructor implemented in the program in Step 2?
3. 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;

}

1. What is the purpose of the DeAllocate function in the program of Step 3?
2. What is the difference between the destrunctor and the DeAllocate functions?
3. Does the program in Step 3 execute with errors? If so, name and explain the

type(s) of execution error(s) you observed.