

BOOK No. _____

CONCORDIA UNIVERSITY

10/08

NUMBER OF BOOKS USED _____ ADDITIONAL BOOK (to be placed inside back cover of first book)

FILL IN THE FOLLOWING:

NAME Naser Mohammed 6769268
(Please Print) SURNAME GIVEN NAMES

SUBJECT COMP 345
(Course and Number)

COURSE GIVEN BY Dr. J. Paquet

EXAMINATION SUPERVISED _____

DATE WRITTEN 19 Oct. 2013

10 IA 3

A. Output: $40 \llcorner 30$ (2)
 $40 \llcorner 40$ (2)
 $10 \llcorner 20$ (3)

P_2 contains the memory address of l_2 . We are trying to set the value of an int pointer (P) to another pointer which points to an int, which is NOT an int pointer in that case, We expected a memory address of an int.

B. If no constructor is defined in derived constructor, the constructor of all base classes are called in order from topmost. For example:

```
class A ...  
class B : public A  
class C : public B
```

The constructor of A will be called first, then B and then C if we create a new C object.

Continued on next page →

80%

good

CONCORDIA UNIVERSITY
BOOK NO. _____
NUMBER OF BOOKS LISTED _____
ADDITIONAL BOOK (to be placed inside back cover of first book)
FILL IN THE FOLLOWING

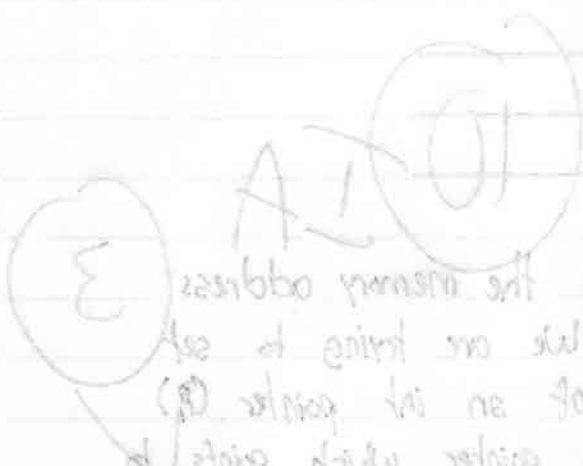
NAME: Mohammed

SUBJECT: Comp 3MS

COURSE GIVEN BY: Dr. J. Pappert

EXAMINATION SUPERVISOR: _____

DATE WRITTEN: 12 Oct 2013



A contains the memory address of B. We are trying to get the value of an int pointer (B) to another pointer which points to an int. Which is not an int pointer in that case. We expected a memory address of an int.

A: Output: 1073016
1074016
1073016

B. If a constructor is defined in derived constructor the constructor of all base classes are called in order from topmost. For example:

```
class A {
class B : public A {
class C : public B {
```

The constructor of A will be called first then B and then C. If we create a new C object.

← constructor of next base →

If the constructor is implicitly defined, but not called, it will be called anyways, example of such constructor

```
B::B(int foo_, int bar_) : foo(foo_), bar(bar_) { ... }
```

Which one?

The above would still call the constructor for A. However it can implicitly be called too:

```
B::B(int foo_, int bar_) : A(), foo(foo_), bar(bar_) { ... }
```

If the constructor has different signature it would need to be implemented manually ~~explicitly~~

```
B::B(int x, int foo_, int bar_) : A(x), foo(foo_), bar(bar_) { ... }
```

C. These operators must be implemented by a derived class or explicitly defined to use the base class (such as using virtual for ~~constructor~~ destructor) to avoid

missing calls? if the item is being accessed by a pointer related to the base object. pointer members deep copies

D. The diamond problem is when having a class with multiple inheritance and ~~functions~~ from base classes which are all duplicated in memory.

When using virtuals the code for these ~~methods~~ is pointed to directly instead of being duplicated.

virtual inheritance ≠ virtual methods.

DayOfYear.cpp

#include "DayOfYear.h"

DayOfYear.h

using namespace std;

~~#include "DayOfYear"~~

#ifndef DAYOFYEAR_H

#define DAYOFYEAR_H

5

class DayOfYear {

{

public:

void input();

void output();

void set(int newMonth, int newDay);

int getMonthNumber();

int getDay();

private:

int month;

int day;

};

#endif

void DayOfYear::input() {

{

month = newMonth;

day = newDay;

}

}

Continued →

DayOfYear.cpp

#include "DayOfYear.h"

#include <iostream>
using namespace std;

0/2

void DayOfYear::input()

{
int m, d;

cout << "Please enter month: ";

cin >> m;

cout << "\n";

cout << "Please enter day: ";

cin >> d;

cout << "\n";

set(m, d);

}

void DayOfYear::output()

{

cout << "The day is " << day << " and the month is "
<< month << "\n";

}

void DayOfYear::set(int newMonth, int newYear)

{

month = newMonth;

year = newYear;

}

Continued →

```

int DayOfYear :: getMonthNumber (d)
{
    return month;
}

```

```

int DayOfYear :: getDay (d)
{
    return day;
}

```

```

int get Size (s)
+ MyArray ();

```

DayOfYear Driver. cpp

```

#include "DayOfYear.h"

```

```

int main ()

```

```

{
    size = 0;

```

```

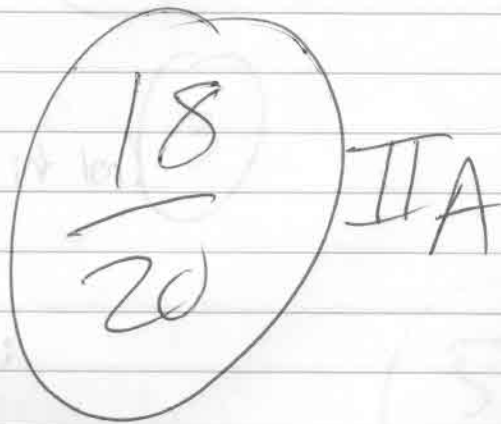
    MyArray = new int[size];

```

```

}

```



```

for (size = 0; size < len; size++)
    MyArray[size] = new int[size];

```

```

for (int i = 0; i < len; i++) MyArray[i] = arr[i];

```

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

```
class MyArray
{
```

```
public:
```

```
MyArray();
```

```
MyArray(int[], int);
```

```
getArray int* getArray();
```

```
int getSize();
```

```
~MyArray();
```

```
private:
```

```
int size;
```

```
int* theArray;
```

```
};
```

```
MyArray::MyArray()
```

```
{
```

```
size = 0;
```

```
theArray = new int[0];
```

```
}
```

```
MyArray::MyArray(int arr[], int len)
```

```
{
```

```
size = len;
```

```
theArray = new int[len];
```

```
for(int i = 0; i <= len; i++) theArray[i] = arr[i];
```

```
}
```

5

5

5

3

5

```
int* MyArray::getArray()
{
    return theArray;
}
```

5

```
int MyArray::getSize()
{
    return size;
}
```

2

```
MyArray::~MyArray()
{
    size = NULL;
    delete [] theArray;
    theArray = NULL;
}
```

3/5

40

#B

```
int main()
{
```

~~MyArray~~

```
int[] staticArray = {1, 2, 3};
```

```
MyArray dynamicArray(staticArray, 3);
```

```
int size = dynamicArray.getSize();
```

```
int* arr = dynamicArray.getArray();
```

```
cout << "Size: " << size << "\n";
```

```
cout << "Items: ";
```

```
for(int i = 0; i <= size; i++) cout << arr[i] << " ";
```

```
cout << "\n";
```

```
}
```

3

3

1

3;