BOOK No. _____ **CONCORDIA UNIVERSITY**

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 15 Oct. 2013

10 - A

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

P_2 contains the memory address of l_2. We are trying to set the value of an int pointer (P_1) 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. (3)

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 USED ____ ADDITIONAL BOOK (to be placed inside back cover of first book)

FILL IN THE FOLLOWING

NAME Nacer Mohammed 5749268

SUBJECT COMP 345

COURSE GIVEN BY Dr. J. Paquet

EXAMINATION SUPERVISED ______

DATE WRITTEN 15 Oct. 2013

10

1 A

3

A. Output: 40 30 (2)
40 40 (2)
10 20 (1)

p2 contains the memory address of p1. We are trying to set the value of an int pointer (p1) 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 →

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 implicitely be called too:

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

8/10

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

```
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 explicitely 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 casted to the base object. pointer members must deep copies

2/10

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 virtual the code for these ~~methods~~ is pointed to directly instead of being duplicated.

2/10

virtual inheritance ≠ virtual methods.

DayOfYear.h

~~#include "DayOfYear~~

```
#ifndef DAYOFYEAR_H
#define DAYOFYEAR_H

class DayOfYear
{
public:
    void input();
    void output();
    void set(int newMonth, int newDay);
    int getMonthNumber();
    int getDay();
private:
    int month;
    int day;
};

#endif
```

5

✓

✓

3

DayOfYear.cpp

#include "DayOfYear.h"

0/2

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

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;
}
```

5

Continued →

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

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

DayOfYearDriver.cpp

```
#include "DayOfYear.h"

int main()
{
   ....
}
```

5

18/20

II A

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

class MyArray
{
public:
    MyArray();
    MyArray(int[], int);
    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];
}
```

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

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

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

int main()
{
    MyArray
    int[] static_array = {1, 2, 3};
    MyArray dynamic_array(static_array, 3);

    int size = dynamic_array.getSize();
    int* arr = dynamic_array.getArray();

    cout << "Size: " << size << "\n";
    cout << "Items: ";
    for(int i = 0; i <= size; i++) cout << arr[i] << ", ";
    cout << "\n";
}
```

5

2

315

40

3

3

1

3