CS 2413 001 Summer 2000 Homework #1

Quiz to be held in class 1:20pm Friday 16 June 2000

You may need extra sheets of notebook paper to write your answers on.

1. What is the **output** of each of the following programs? (Mark programs that won't compile **WON'T COMPILE**, and explain why.) If you aren't confident of an answer, type in, compile and run the program to test it.

   (a)

```
#include <iostream.h>

int func1 (int a, int b)
{ // func1
    a = a * 2;
    b = b * 3;
    return a + b;
} // func1

int main ()
{ // main
    int x = 5, y = 10, z;

    z = func1(x, y);
    cout << "x = " << x << ", y = " << y << ", z = " << z << endl;
} // main
```

   (b)

```
#include <iostream.h>

int func1 (int& a, int& b)
{ // func1
    a = a * 2;
    b = b * 3;
    return a + b;
} // func1

int main ()
{ // main
    int x = 5, y = 10, z;

    z = func1(x, y);
    cout << "x = " << x << ", y = " << y << ", z = " << z << endl;
} // main
```

   (c)

```
#include <iostream.h>

int func1 (const int& a, const int& b)
{ // func1
    a = a * 2;
    b = b * 3;
    return a + b;
} // func1

int main ()
{ // main
    int x = 5, y = 10, z;

    z = func1(x, y);
    cout << "x = " << x << ", y = " << y << ", z = " << z << endl;
} // main
```

2. For this question, you will need to read the document "The C++ Preprocessor," which is available from the CS 2413 webpage:

   `http://www.cs.ou.edu/~cs2413/`

   Consider the following set of C++ source files:

   - Point.h

```
#ifndef Point_h
#define Point_h

class Point {
    protected:
        double _x, _y;
    public:
        Point();
        Point(double x, double y);
        virtual ~Point();
        void display();
        Point operator+(const Point& b);
}; // class Point

#endif // #ifndef Point_h
```

   - Point.cpp

```
#include <iostream.h>
#include "Point.h"

#ifndef DEBUGGING_VERBOSITY
#  define DEBUGGING_VERBOSITY  0
#endif // #ifndef DEBUGGING_VERBOSITY

Point::Point ()
{ // Point::Point
#if DEBUGGING_VERBOSITY >= 2
    cout << "Enter Point()" << endl;
#endif // #if DEBUGGING_VERBOSITY >= 2

    _x = 0; _y = 0;

#if DEBUGGING_VERBOSITY >= 2
    cout << "Exit  Point()" << endl;
#endif // #if DEBUGGING_VERBOSITY >= 2
} // Point::Point

Point::Point (double x, double y)
{ // Point::Point
#if DEBUGGING_VERBOSITY >= 2
    cout << "Enter Point(" << x << "," << y << ")" << endl;
#endif // #if DEBUGGING_VERBOSITY >= 2

    _x = x; _y = y;

#if DEBUGGING_VERBOSITY >= 2
    cout << "Exit  Point(" << x << "," << y << ")" << endl;
#endif // #if DEBUGGING_VERBOSITY >= 2
} // Point::Point

Point::~Point ()
{ // Point::~Point
#if DEBUGGING_VERBOSITY >= 2
    cout << "~Point called" << endl;
#endif // #if DEBUGGING_VERBOSITY >= 2
} // Point::~Point
```

```
void Point::display ()
{ // Point::display
#if DEBUGGING_VERBOSITY >= 2
    cout << "Enter Point.display()" << endl;
#endif // #if DEBUGGING_VERBOSITY >= 2

    cout << "x = " << _x << ", y = " << _y << endl;

#if DEBUGGING_VERBOSITY >= 2
    cout << "Exit  Point.display()" << endl;
#endif // #if DEBUGGING_VERBOSITY >= 2
} // Point::display

Point Point::operator+ (const Point& b)
{ // Point::operator+
#if DEBUGGING_VERBOSITY >= 2
    cout << "Point.operator+() called," << "  b._x = " << b._x <<
        ", b._y = " << b._y << endl;
#endif // #if DEBUGGING_VERBOSITY >= 2
    return Point(_x + b._x, _y + b._y);
} // Point::operator+
```

- Point.main.cpp

```
#include <iostream.h>
#include "Point.h"

int main ()
{ // main
    Point p1(1,10), p2(2, 20);
    Point psum = p1 + p2;
    psum.display();
    return 0;
} // main
```

(a) Give the **output** of the program if you compile using:
`g++ Point.main.cpp Point.cpp`

(b) Give the **output** of the program if you compile using:
`g++ -DDEBUGGING_VERBOSITY=2 Point.main.cpp Point.cpp`

If you aren't confident of an answer, type in, compile and run the program to test it.

3. For this question, you will need to read the document "C++ Arrays & Strings," which is available from the CS 2413 webpage:

   `http://www.cs.ou.edu/~cs2413/`

   The Depressingly Dull Corporation (DDC) has many employees. Each employee has:

   - a first (given) name;
   - a last (family) name;
   - a Social Security number (a 9 digit integer);
   - an hourly pay rate in dollars (which may include cents after the decimal point);
   - a number of unused vacation days;
   - a number of unused sick days.

   Employees come in two kinds, laborers and managers, each of which has an extra piece of information:

   - Laborers have the ID number of the part that they assemble.
   - Managers have the number of papers shuffled.

   For this question:

   (a) Declare a class `Employee` with the appropriate fields; the fields should have the appropriate level of privacy. The class should also have appropriate methods, including:
      - a default constructor;
      - a constructor that takes values for all of the fields;
      - a copy constructor;
      - a destructor;
      - a display method that outputs all of the employee's data;
      - a method that returns the amount that the employee should be paid this week, given the number of hours that they worked;
      - an output stream function (i.e., a friend).

   (b) Declare a class `Laborer` that inherits all of the properties of `Employee`, but has its own constructors and destructor, as well as a display method that overrides the display method of `Employee`.

   (c) Declare a class `Manager` that inherits all of the properties of `Employee`, but has its own constructors and destructor, as well as a display method that overrides the display method of `Employee`.

   Note: for this question, you do not need to implement the methods of these classes; you only need to declare the classes (i.e., you need to write the `Employee.h` file, but not the `Employee.cpp` file).

4. Implement the following methods from the previous question:

   (a) The copy constructors for each of the three classes.

   (b) The destructor for the `Employee` class.

   (c) The display methods for each of the three classes.

   Your implementations should not have any redundant code in them; that is, if a method of the `Employee` class has code to perform a task for its data fields, then the methods for the `Laborer` and `Manager` classes should not contain the same code; they should instead invoke the associated method from the `Employee` class and include code to operate on their subclass-specific fields.