CS 7B - Spring 2014 - Midterm 1 Solutions. 3/11/14

Write responses on separate paper.

1. What will be printed by the following program segment? Use the following data:
   `Betty 100000 0.1`

```
class Slacker {
public:
   string name;
   double rate, hours, pay;
};

int main() {
    Slacker lazyBoy;
    cout.setf(ios::fixed,ios::floatfield);
    cout.precision(2);
    cin >> lazyBoy.name >> lazyBoy.rate >> lazyBoy.hours;
    lazyBoy.pay = lazyBoy.rate * lazyBoy.hours;
    cout << lazyBoy.pay << endl;

    return 0;
}
```

   Solution: From `http://www.cplusplus.com/reference/ios/ios_base/precision/`
   "The floating-point precision determines the maximum number of digits to be written on insertion operations to express floating-point values. How this is interpreted depends on whether the floatfield format flag is set to a specific notation (either fixed or scientific) or it is unset (using the default notation, which is not necessarily equivalent to either fixed nor scientific).
   "Using the default floating-point notation, the precision field specifies the maximum number of meaningful digits to display in total counting both those before and those after the decimal point. Notice that it is not a minimum, and therefore it does not pad the displayed number with trailing zeros if the number can be displayed with less digits than the precision.
   "In both the fixed and scientific notations, the precision field specifies exactly how many digits to display after the decimal point, even if this includes trailing decimal zeros. The digits before the decimal point are not relevant for the precision in this case."
   In short, the numbers will be printed out like money, with the hundredth of a dollar (penny) value least significant.
   The `cin >> lazyBoy.name >> lazyBoy.rate >> lazyBoy.hours;` command will read the string "Betty" into `lazyBoy.name`, 100000 into `lazyBoy.rate` and 0.1 into `lazyBoy.hours`. The command `lazyBoy.pay = lazyBoy.rate * lazyBoy.hours;` then computes earnings of $10000 for 6 minutes' work. The is print to the console by the line `cout<<lazyBoy.pay<<endl;` Thus the output is `10000.00`.

2. Consider the following definition of a `Vector` class:

```
#include <iostream>
class Vector {
// friends
   friend Vector operator*(double n, const Vector & a);
   friend std:ostream &
       operator<<(std::ostream & os, const Vector & v);
public:
   enum Mode {RECT, POL};
   //RECT for recdtangular, POL for polar modes
   Vector();
   Vector(double n1, double n2, Mode form = RECT);
   void reset(double n1, double n2, Mode form = RECT);
   ~Vector();
```

```
14    double xval() const {return _x;}      // report x
      double yval() const {return _y;}      // report y
16    double magval() const {return _mag;} // report length
      double angval() const {return _ang;} // report angle
18    void polar_mode();                    // set polar mode
      void rect_mode();                     // set rectangular mode
20    // operator overloading
      Vector operator+(const Vector & b) const;
22    Vector operator-(const Vector & b) const;
      Vector operator-() const;
24    Vector operator*(double n) const;
private:
26   double _x;      // horizontal position
     double _y;      // vertical postion
28   double _mag;    // length of vector
     double _ang;    // direction of vector in degrees
30   Mode mode;      // RECT or POL
     // private methods for setting values
32   void set_mag();
     void set_ang();
34   void set_x();
     void set_y();
36 };
```

Complete the definition of the functions in parts a,b,c.

(a) Recall that the magnitude of a vector, (x, y) is $\sqrt{x^2 + y^2}$ and assume that the `cmath` library has been included.
`void Vector::set_mag() { //code here };`
Solution: You could make this more complicated, but the simplest definition is

```
void Vector::set_mag() {
2     _mag = sqrt(_x*_x+_y*_y);

4 }
```

(b) Recall that the polar angle of a vector is either $\arctan(y/x)$ or $\pi + \arctan(y/x)$ depending on which quadrant the vector lies in. Also, the `cmath` function `atan2(y,x)` figures that out, except that $(x, y) = (0, 0)$ needs to be handled as a special case.
`void Vector::set_ang() { //code here };`

```
void Vector::set_mag() {
2     _ang = atan2(_y,_x);

4 }
```

(c) Recall that the horizontal coordinate of a vector can be recovered from the polar angle and length of a vector by the formula $x = r\cos(\theta)$.
`void Vector::set_x() { //code here };`

(d) Define the default constructor to create the zero vector (length = 0 and angle = 0) in RECT mode.

```
Vector::Vector(double n1, double n2, Mode form) {
2     _mode = form;
      if(form==RECT)
4     {
          _x = n1;
6         _y = n2;
          _mag = sqrt(_x*_x+_y*_y);
```

```
        _ang = atan2(_y,_x);
    }
    else {
        _mag = n1;
        _ang = n2;
        _x = _mag*cos(_ang);
        _y = _mag*sin(_ang);
    }
}
```

In fact, using the `atan2()` function takes care of the special (0,0) case, so, no worries!

(e) Overload the constructor to take three parameters: the length, angle and mode and to set these accordingly.
Solution: The constructor defined above does all this.

(f) Define the reset function to change from polar to rectangular mode if form is POL or from rectangular to polar if form is RECT.
Solution: The given code had a parameter list that was misleading. To accomplish the specified task, the code below will do with the prototype: `void reset();`

```
void Vector::reset() {
    if(mode == RECT) mode = POL;
    else mode = RECT;
}
```

(g) Define the addtion (+), subtraction (-), and negation (-) and scalar multiplication (*) opeartors.

```
Vector Vector::operator+(const Vector & b) const {
    double x = _x + b._x;
    double y = _y + b._y;
    Vector temp(x,y,RECT);
    return temp;
}

Vector Vector::operator-(const Vector & b) const {
    double x = _x - b._x;
    double y = _y - b._y;
    Vector temp(x,y,RECT);
    return temp;
}

Vector Vector::operator-() const {
    double x = -_x;
    double y = -_y;
    Vector temp(x,y,RECT);
    return temp;
}
```

The treatment of the scalar multiplier is a bit tricky. The problem is the in scalar*Vector = k*V, the calling operator (lhs = k) is not a vector. The * operator takes two operands: a scalar and vector, and since the implicit operand is the scalar in k * V, we're stuck. C++ provides for this in two ways.
1. Define V * k and create a global operator for the commutative case.
2. Use a friend function.

Below is the relevant code for the first of these. Note how the global operator calls the member operator.

```
// This is the (Vector * scalar) operator
// defined inside the Vector class interface
```

```
Vector operator * (double n ) //const
{
   /* multiply your vector (*this) by the scalar (k)
      here and put the result in v*/
   this->_x *= n;
   this->_y *= n;
   Vector v(*this);
   return v;
}

//global operator defined outside Vector interface
Vector operator*(double n, Vector& v) {
    return v*n;
}
```

3. Trace the following code, showing each value of each variable:

```
int a[] = {20, 30, 40, 50};
int* p = &a[1]; // assume that p gets the value 0x3fffe0f here
int n = *p;
++(*p);
++p;
int* q = &a[2];
*(--q) = 60;
p -= 2;
n = q - p;
```

step	a	p	q	n	$*p$	$*q$
0	$\{20, 30, 40, 50\}$	-random-	-random-			
1		0x3fffe13			30	
2	$\{20, 31, 40, 50\}$			30	31	
3		0x3fffe17			40	
4			0x3fffe17			40
5	$\{20, 60, 40, 50\}$		0x3fffe13			60
6		0x3fffe0f			20	
7				1		

4. Assume `ip` is a pointer to an `int`. Write a statement that will dynamically allocate an integer variable and store its address in `ip`. Write a statement that will free the memory allocated in the statement you wrote.
Solution: This is almost like just declaring an integer variable, the difference is that the memory is allocated on the heap in a way that in can later be deleted:

```
    int *ip = new int; // dynamically allocate an integer
    *ip = 17; // assign 17 to this integer
    delete ip;
```

5. Consider the following array defintion.
`char str[] = "237.89";`
`double value;`
Write a statement that converts the string in `str` to a double and stores the result in `value`.
Solution: "A statement," was a bit misleading. I think you need more than one statement...a "compound statement," perhaps. Here's one solution:

```
char str[] = "237.89";
char* sp = str;
double value = 0, pow10 = 10.;
bool dec = false;
for(;*sp && *sp!='.'; ++sp) value = 10*value + int(*sp-'0');
++sp;
for(;*sp;++sp) {
    value = value + int(*sp-'0')/pow10;
    pow10 *= 10.;
}
cout << "\nvalue = " << value;
```