

NAME (print): _____ Lecture # _____ Lab # _____

Honor Acknowledgment (signature): _____

DO NOT SPEND MORE THAN 10 MINUTES ON ANY OF THE QUESTIONS! If you don't see the solution to a problem right away, move on to another problem and come back to it later.

Before starting, make sure your test contains 10 pages including a blank page at the end.

If you think there is a syntax error, then ask. You may assume any include statements are provided.

	value	grade
Problem 1	10 pts.	
Problem 2	16 pts.	
Problem 3	16 pts.	
Problem 4	16 pts.	
Problem 5	20 pts.	
Optional	(8) pts.	
TOTAL:	78 pts.	

PROBLEM 1 : (*Daffynition. recurse: to curse again again again ...* (10 pts))

Part A (7 points)

Trace the recursive function `Pow` that is shown in its entirety below. Assume that it is invoked with the arguments 4.0 and 5. For example we might use $y = \text{Pow}(4.0, 5)$;

Trace this recursive routine for these arguments by listing each case for which `Pow` is called (directly as above and recursively) and showing both the arguments and what value is returned.

As a trivial example, with no recursion is involved, if we had asked you to trace this for $y = \text{Pow}(7.5, 0)$; Your answer would simply have been

`Pow(7.5, 0)` returns 1.0

Again, you do the trace for `Pow(4.0, 5)`;

```
double Pow(double base, int expo)
// pre: expo >= 0
// post: -----
{
    if (0 == expo)
    {
        return 1.0;                // correct for zero
    }
    else
    {
        double semi = Pow(base, expo/2);    // build answer from this

        if (expo % 2 == 0)                // even
        {
            return semi*semi;
        }
        else                                // odd
        {
            return base*semi*semi;
        }
    }
}
```

Part B (3 points)

Complete the postcondition for the function shown above.

PROBLEM 2 : (*They get you coming and going:* (16 points))

Complete the function *isPalindrome* whose header is included below.

A palindrome is a word, phrase, or sentence that, when punctuation, spaces, and capitalization are ignored reads the same when the order of its characters is reversed. For example the phrase **"Madam, I'm Adam"** is a palindrome. Therefore *isPalindrome("MadamImAdam")* should return a **true**. Note that your function does not have to deal with the punctuation marks and blanks (they've already been removed). However, you do have to deal with case.

```
bool isPalindrome(const string & candidate)
// pre: candidate is a string representing a word, phrase or sentence,
//       but having had all blanks and punctuation marks removed.
// post: returns true if candidate is a palindrome or string is
//       empty. Returns false otherwise.
{
```

PROBLEM 3 : (*Life has its ...* : (16 points))

Complete the function *upsAndDowns* whose header is included below.

People who follow the stock market often make note of how many times the stock price changes direction over a period. Biologists studying populations of animals in the wild might also note the rise and fall of the numbers of certain species. The function you are to write should count how often the numbers in a vector "change direction" and return that value.

For example, given the vector *x*:

4.3	5.0	9.2	8.8	7.4	12.1	14.3	19.7	20.0	18.6	16.3	11.4
-----	-----	-----	-----	-----	------	------	------	------	------	------	------

upsAndDowns(x, 12) returns **3** because the data changes direction three times. (This is also the number of peaks and valleys if you graphed this, ignoring the endpoints as possible peaks and valleys.) You may assume no two adjacent points will be identical. Note that *upsAndDowns(x, 4)* returns **1** because there is only one direction change in the first four numbers and *upsAndDowns(x, 3)* returns **0**.

```
int upsAndDowns(const Vector<double> & data, int length)
// pre: data is a partially filled vector of doubles of which only the
//       first length elements contain meaningful values
// post: returns the number of times the sequence of numbers, data[0] ...
//       data[length-1], changes from ascending to descending or from
//       descending to ascending. Returns 0 if length is 0, 1, or 2.
{
```

PROBLEM 4 : (*Sum it up and down:* (16 points))

For this problem, you will be computing the sum of a set of numbers.

Part A (6 points)

Write the function *SumItUp* whose header is given below. When given a Vector of integers, *SumItUp* returns a single integer that represents the sum of the first `count` numbers in the Vector. For this part of the problem, you must implement your function iteratively (i.e., with a loop).

```
int SumItUp(const Vector<int> & numbers, int count)
// pre:  numbers contains count integers
// post: returns sum of first count integers in numbers
{
```

Part B (10 points)

Complete the function *SumItUp* whose header is given below. When given a Vector of integers, *SumItUp* returns a single integer that represents the sum of the first `count` integers in the Vector. For this part of the problem, we have started a recursive implementation that identifies the base and the general case. You must complete the code given so that it returns the correct sum.

```
int SumItUp(const Vector<int> & numbers, int count)
// pre:  numbers contains count integers
// post: returns sum of first count integers in numbers
{
    if (count == 0) // base: no more numbers to sum
    {
        return _____ ;
    }
    else           // general: add last to sum of smaller vector
    {
        return numbers[count - 1] + SumItUp(numbers, _____ );
    }
}
```

PROBLEM 5 : (*Redux your magazines:* (20 points))

For this problem, information about a subscriber is stored in a struct defined below:

```
struct Subscriber
{
    string name;
    string magazine;
    int    yearToRenew;
};
```

Part A (8 points)

The code shown on the next page prompts for the name of a file containing name, magazine, and renewal year information for an arbitrary number of subscribers. It then tries to use this file to fill a Vector of **Subscribers** with the information stored in the file. For example, if the function was written correctly, the following data file:

```
Kenny Science 1999
Lisa Games 2002
```

would generate the following Vector of subscribers:

Kenny	Lisa
Science	Games
1999	2002

However, it generates this Vector of subscribers:

Science
1999
0

So, even though each subscriber entry is correctly formatted, all names are only one word, and always in the correct order; the data is loaded incorrectly.

Explain the error in the code shown below and suggest a solution.

```
int main()
{
    Vector<Subscriber> subscribers(1);
    ifstream input;

    // assume file exists and can be read
    string fileName = PromptString("enter name of file: ");
    input.open(fileName);

    int k = 0;
    string word;
    while (input >> word)
    {
        input >> subscribers[k].name;
        input >> subscribers[k].magazine;
        input >> subscribers[k].yearToRenew;

        k++;
        if (k == subscribers.length())
        {
            subscribers.SetSize(2 * k);
        }
    }

    return 0;
}
```

Part B (12 points)

Write the function *NeedsToRenew* whose header is given below. *NeedsToRenew* returns the name of the subscriber that needs to renew the most (i.e., whose year to renew is the smallest). If multiple subscribers need to renew in the same year, return the **last** one in the Vector. For this problem, you may assume the data contained in the given Vector was loaded correctly.

```
string NeedsToRenew(const Vector<Subscriber> & subscribers, int count)
// pre: subscribers contains count Subscribers
// post: returns name of subscriber who most urgently needs to renew
{
```

PROBLEM 6 : (*Duped again* : (8 points))

OPTIONAL * EXTRA CREDIT *** OPTIONAL**

Part A (6 points)

Write the function *FindDupe* whose header is given below. *FindDupe*, when given `count` integers, each within a range of 1 to `count - 1` inclusive, returns the integer within the collection that appears twice. It is guaranteed that one and only one number within the collection of integers is repeated.

```
int FindDupe(const Vector<int> & data, int count)
// pre:  data contains count integers each within the range: 1 <= data[k] <= count - 1
// post: returns the value of the repeated integer
{
```

Part B (2 points)

Explain what is guaranteed about the numbers that appear in the collection if one and only one number within the collection of integers is repeated.