PROBLEM 1:  *(Up the Descending Staircase: 8 points)*

Some characters are "descenders" when printed, that is part of the character descends below the line. For example, the characters 'g', 'j', and 'y' are descenders. Write the function `NumDescenders` whose header is given below. `NumDescenders` returns the number of characters in the string parameter `s` that are descenders.  

You may call the function `IsDescender` whose prototype is given below; assume that the function `IsDescender` exists. DO NOT write `IsDescender`!

```c
bool IsDescender(char ch) // postcondition: returns true if ch is a descender, otherwise false
```

Complete `NumDescenders` below.

```c
int NumDescenders(string s) // postcondition: returns number of characters in s that are descenders
{
}
```

PROBLEM 2:  *(An Orderly Way: 8 points)*

Write the function `IsInOrder` whose header is given below. `IsInOrder` returns `true` if the elements of the vector `list` are in non-decreasing order.

For example, consider the following arrays:

```
A: 3 4 4 5 5 7 8 8
B: 1 4 4 5 6 7 6 8
```

`IsInOrder(A, 8)` should return `true`, but `IsInOrder(B, 8)` should return `false` since 7 6 is not in non-decreasing order. However, `IsInOrder(B, 5)` should return `true`.

Complete the function `IsInOrder` below.
bool IsInOrder(const Vector<int> & list, int numInts)
// precondition: numInts is the number of integers in list
// postcondition: returns true if the integers in list are
// in nondecreasing order. Otherwise, returns false.
{

}

PROBLEM 3: (Scope: 8 points)

What is the output of the following program?

#include <iostream.h>

int x = 1;
int y = 2;

int Nancy(int & a, int & b)
{
    a = 5;
    b = x;
    if (b > 0)
    {
        int y = 9;
    }
    return y;
}

void Nathan(int & y, int z)
{
    y = 7;
    z = 4;
}

main ()
{
    int x, y, j, k;
    x = 2; y = 3;
    Nathan(x,y);
    cout << x << " " << y << endl;

    x = 2; y = 3;
    Nathan(y,y);
    cout << x << " " << y << endl;

    j = 8; k = 0; x = 2; y=3;
    cout << Nancy(j,k) << endl;
    cout << x << " " << y << " " << j << " " << k << endl;
}

2
Complete the program below to compute the total number of words of length 1, of length 2, of length 3, etc. Assume no word has more than 30 characters. The program should generate output similar to that shown below; for each different word length print the length followed by the number of words of that length. **Do not print anything for lengths for which there are no words.**

For the file: 

<table>
<thead>
<tr>
<th>Length</th>
<th>Occurrences</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
</tbody>
</table>

```
#include <iostream.h>
#include <fstream.h>
#include "CPstring.h"
#include "vector.h"

main ()
{
    ifstream input;
    string filename, word;
    int k;

    cout << "Enter name of file: ";
    cin >> filename;
    input.open(filename);    // assume file opens successfully

    cout << "Length" << "\t" << "Occurrences" << endl
    cout << "=" << endl;
    while (input >> word)
    {

    }
}
```

**PROBLEM 5 : (Snail Mail: 12 points)**

Letters at the post office are electronically scanned and sorted. Information about the number of letters for each family on a street is stored in a struct `Mail` defined below.
struct Mail
{
    string name;         // family name
    string street;       // street where family lives
    int numLetters;      // number of letters
};

Write the function `MaxLetterStreet` which returns the name of the street with the most letters to deliver. You may call the function `CountLetters` whose prototype is given below; assume that the function `CountLetters` exists. DO NOT write `CountLetters`!

```c
int CountLetters(const Vector<Mail> & list, int size, string street)
// precondition: size is the number of elements in list
// postcondition: returns the number of letters to deliver on the
// street specified by the parameter street
```

For example, in the array A below, `CountLetters(A, 5, "Cornwallis")` returns 9 (3+6), and `MaxLetterStreet(A,5)` should return "Main" which is the street with the most letters (12 letters).

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Smith&quot;</td>
<td>&quot;Thomas&quot;</td>
<td>&quot;Hou&quot;</td>
<td>&quot;Raja&quot;</td>
<td>&quot;Tan&quot;</td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>2</td>
<td>8</td>
<td>6</td>
</tr>
</tbody>
</table>

Complete the function `MaxLetterStreet` below.

```c
string MaxLetterStreet(const Vector<Mail> & list, int size)
// precondition: size > 0, size is equal to the number of entries in list
// postcondition: returns the street which has the most letters
```

{ }

**PROBLEM 6:**  (Expansion: 12 points)

The function `Expand` expands a vector with consecutive copies of its original values. For example, `Expand(A,3)` applied to the array A on the left, results in the modified A on the right.

```
 4 8 3 2
<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>8</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>
```

Complete the function `Expand` below.

```c
void Expand(Vector<int> & list, int number)
// precondition: the number of integers in list is equal to list.Length()
// number > 1
// postcondition: list contains "number" consecutive copies of the original vector
```

```c
int len = list.Length();  // the length of list when Expand begins
list.SetSize(list.Length() * number);  // list now has room for
// consecutive copies of itself
```
EXTRA: NOT REQUIRED!

PROBLEM 7:  (Extra Credit: Space Cadet: 6 points)

In some text-processing it is necessary to insert extra white-space in a line. A function `StringSpace` creates a vector of characters (this is NOT the same thing as a string) from a list of strings by copying each string into the vector of characters and inserting spaces between each string. For example, consider the vector of strings below.

![Example Vector]

Inserting two spaces between each string yields the vector of characters shown.

```
Chocolate  is  addictive
```

Creating the vector of characters is done in two steps:

1. The size of the vector is calculated by summing the lengths of all words and adding the total number of spaces required. The size of the character vector (number of elements) is set using `SetSize`.

2. The strings are copied, one character at a time, into the character vector with the appropriate number of spaces added between each string.

Complete the function `StringSpace` that fills the parameter `spaceChars` with characters according to the description above. The vector of characters shown above could be generated by the function call `StringSpace(list, 3, spaceChars, 2)` if the vector of strings shown above was named list.

```c
void StringSpace(const Vector<string> & list, int numStrings,
    Vector<char> & spaceChars, int numSpaces)
    // precondition: numStrings is the number of strings in list, numStrings > 0, numSpaces > 0
    // postcondition: spaceChars contains all characters from the strings
    //    in list, with numSpaces spaces inserted between each string
{
```