This page contains information you might need in the following problems.

The Vector Class

```
// Vector() -- default, vector of size 0 (no entries)
// Vector(int size) -- vector with size entries
// Vector(int size, Item fillValue) -- vector w/ size entries all == fillValue
// int Length() -- returns size of vector (capacity)
// void SetSize(int newSize) -- resizes the vector to newSize elements
// void Fill(Item fillValue) -- set all entries == fillValue
// operator = -- assignment operator works properly
// operator [] -- indexes vector
```

file streams

```
string file, word;
file = "testfile";

ifstream input(file); // to declare an input file stream

while(input >> word) // to use a file stream
```

Useful Function

```
double fabs(double num); // returns absolute value of num
```

**PROBLEM 1**: *(Scope and Recursion: (12 pts))*

What is the output of the following program?

```c
int Mystery(int x)
{
    if (x < 2)
        return 1;
    else
        return 1 + Mystery(x-2);
}

void Something(string & a, string b)
{
    a = "MO";
    b = "LO";
}

void Thing(string & a, string & b)
```
```cpp
{
    a = "SG";
    b = "TG";
}

int main()
{
    string a, b;
    cout << Mystery(6) << endl;
    cout << Mystery(5) << endl;
    a = "RO";  b = "PO";
    Something(a,b);
    cout << a << b << endl;
    a = "RO";  b = "PO";
    Thing(b,b);
    cout << a << b << endl;
}

PROBLEM 2:  (Who is the Youngest?: (12 pts))

Write the function Youngest whose header is given below. Youngest reads in birth dates of people from a file in the form month day year, and sets the parameters month, day and year to the birth date of the youngest person.

For example, if the file family contained the following birth dates:

   2 4 1954
   6 10 1963
   4 17 1963

Then Youngest(family, month, day, year) would set month to 6, day to 10 and year to 1963. Note the youngest person has the largest year. Hint: See the information on filestreams on page 2.

Complete the function Youngest below the following header.

```cpp
template<class T>
void Youngest(string filename, int & month, int & day, int & year)
    // precondition: filename contains birth dates of people in format above,
    // all years are greater than 0, there is at least one date in filename
    // postcondition: year, month and day are the birth date for the youngest in filename
{
}
```cpp

PROBLEM 3:  (Where are the Hot Spots?: (12 pts))

Write the function TempInfo whose header is given below. TempInfo is given a vector of temperatures, and it calculates the average temperature, and the maximum difference between the average and other temperatures.
For example, consider the following vector `info`:

```
info:  46.7  54.3  52.9  51.7
```

The call `TempInfo(info, 4, avg, maxdiff)` sets `avg` to 51.4, and `maxdiff` to 4.7 (the differences between each temp and the average are 4.7, 2.9, 1.5, 0.3, with the maximum difference equal to 4.7).

Complete `TempInfo` below the following header.

```c
void TempInfo(const Vector<double> & temps, int num, double & average, double & diff) {
  // precondition: num is the number of temperatures in "temps", num > 0
  // postcondition: average is the average temperature in "temps", diff is the maximum
  // difference between the average temperature and temperatures in "temps"
```

**PROBLEM 4:** (Am I the only one?: 20 pts)

Consider the following definition for this problem and assume the field `unique` has no value.

```c
struct Member {
  string name;  // person’s name
  bool unique;  // has no value initially
};
```

**PART A** (10 pts) Write the function `IsUnique` whose header is given below. `IsUnique` is given a Vector of Members containing names, and a person’s name, and returns true if the name is unique (appears at most once) in the Vector. The `unique` field in the Vector has NOT been set (you will set this value in Part B).

For example, in the Vector named `list` shown below, `IsUnique(list, 8, ”Kyle”)` returns false, `IsUnique(list, 8, ”Alice”)` returns false, and `IsUnique(list, 8, ”Jeremy”)` returns true.

```
name:  ”Jeremy” ”Kyle” ”Alice” ”Kyle” ”Sharon” ”John” ”Kyle” ”Alice”
unique: - - - - - - - -
```

Complete `IsUnique` below the following header.

```c
bool IsUnique(const Vector<Member> & group, int num, string name) {
  // precondition: num > 0 is the number of elements in "group", name appears in "group"
  // precondition: returns true if name occurs at most once in "group",
  // otherwise returns false.
```

**PART B** (10 pts) Write the function `SetUnique` whose header is given below. `SetUnique` is given a Vector of Members with names, and sets the unique field associated with each name, indicating whether the name is unique (appears at most once in the Vector).
For example, in the Vector named list shown on the previous page, SetUnique(list, 8) modifies the Vector by setting the unique fields, resulting in the Vector shown below.

<table>
<thead>
<tr>
<th></th>
<th>&quot;Jeremy&quot;</th>
<th>&quot;Kyle&quot;</th>
<th>&quot;Alice&quot;</th>
<th>&quot;Kyle&quot;</th>
<th>&quot;Sharon&quot;</th>
<th>&quot;John&quot;</th>
<th>&quot;Kyle&quot;</th>
<th>&quot;Alice&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>name:</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>unique:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

You may use the function IsUnique you wrote in PART A. Assume IsUnique works correctly, regardless of what you wrote in PART A.

Complete SetUnique below the following header.

```cpp
void SetUnique(Vector<Member> & group, int num)
// precondition: num > 0 is the number of elements in "group",
// postcondition: unique field has been calculated for each element in "group"
{
```

**PROBLEM 5 :** *(Cashing in on the Wedding: (20 pts))*

The class Registry is designed to keep track of information for a bridal registry at a store, including the couples names and the items they have registered for. This class is an Iterator class, allowing one to iterate over each of the couples registered, once the member function First is called. For each couple, one can determine if the couple has selected a particular item.

```cpp
class Registry
{
public:
    Registry(); // constructor
    void First(); // set to first couple in registry
    string Current(); // returns bride-groom name of current couple
    void Next(); // advance to next couple
    bool IsDone(); // true if iterator done
    bool HasSelected(string); // true if current couple has selected item
    // rest not shown
};
```

For example, consider the code segment shown below.

```cpp
Registry Macys;
Macys.First();
if (Macys.HasSelected("bread machine"))
    cout << Macys.Current() << " selected a bread machine " << endl;
```

If the first couple in the registry is Smith-Jones and they have selected a bread machine, then "Smith-Jones selected a bread machine" is printed.

**PART A (10 pts)** Write the function SetCounts whose header is given on the next page. SetCounts is given a Vector of type Item (defined below) in which names of items one can register for have
already been placed into the Vector, but there are no values for count. \textit{SetCounts} computes the number of couples in the registry that have registered for each item, and sets the associated count field to this number.

\begin{verbatim}

struct Item
{
    string name;  // name of Item
    int count;    // count of item
};

For example, consider the Vector \textit{list} shown below that is passed to \textit{SetCounts}.

\begin{verbatim}

<table>
<thead>
<tr>
<th>name</th>
<th>toaster</th>
<th>blender</th>
<th>rake</th>
<th>microwave</th>
<th>beach towel</th>
<th>10gal cooler</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

\end{verbatim}

Note that there are no values yet in the count fields. \textit{SetCounts} should access the registry and fill in the missing counts. Suppose that the registry for the store Sears has the number of couples registered for each item as: "toaster" is 9, "blender" is 22, "rake" is 0, "microwave" is 5, "beach towel" is 7, and "10gal cooler" is 11. (you’ll have to calculate each of these numbers by seeing if each couple has registered for each item) After calling \textit{SetCounts}(\textit{list}, 6, Sears), the counts would be set and list would look like:

\begin{verbatim}

<table>
<thead>
<tr>
<th>name</th>
<th>toaster</th>
<th>blender</th>
<th>rake</th>
<th>microwave</th>
<th>beach towel</th>
<th>10gal cooler</th>
</tr>
</thead>
<tbody>
<tr>
<td>count</td>
<td>9</td>
<td>22</td>
<td>0</td>
<td>5</td>
<td>7</td>
<td>11</td>
</tr>
</tbody>
</table>

\end{verbatim}

Complete \textit{SetCounts} below the following header. Note the registry has already been created and is passed as a parameter.

\begin{verbatim}

void SetCounts(Vector <Item> & things, int num, Registry & Sears)
{// precondition: num > 0 is the number of elements in "things"
// postcondition: calculates the count of each item in the registry and
//                 sets this value in "things"
{

\end{verbatim}

\textbf{PART B: (10 pts)}

Write the function \textit{CoupleWithPopularItem} whose header is given below. \textit{CoupleWithPopularItem} takes as input a Vector of Items with names and counts (assume they have already been set by calling \textit{SetCounts} in the previous problem) and returns the name of a couple that has selected the most popular item.

For example, consider the vector \textit{list} from the previous problem. If the couple "Roden-Hammer" has selected a blender, then this couple name (or one of the other 21 couple names that selected blender) would be returned. Hint: You must first determine which item is the most popular item. Complete \textit{CoupleWithPopularItem} below the following header.

\begin{verbatim}

\end{verbatim}
string CoupleWithPopularItem(const Vector<Item>& things, int num, Registry& Sears) 
// precondition: num > 0 is the number of elements in "things",
// postcondition: returns a couple name who has registered for the most popular item
{