String class

int length();
// post: returns number of characters in string

string substr(int pos, int len);
// pre: 0 <= pos < length()
// post: returns substring of len chars, starting at pos
// (as many characters as possible if len too large)

int find(string s)
// post: returns first position at which s begins
// (returns string::npos if s does not occur)

tvector class

    int size() const; // # elements constructed/stored
    void push_back(const itemType & t); // put an item in, same type as array
    void pop_back(); // remove last item put in

tmatrix class

    // Examples of use:
    //
    //   tmatrix<double> dmat( 100, 80 ); // 100 x 80 matrix of doubles
    //   tmatrix<double> dzmat( 100, 80, 0.0 ); // initialized to 0.0

    int numrows() const; // number of rows
    int numcols() const; // number of columns
    void resize( int newRows, int newCols ); // resizes matrix to newRows x newCols

getline and istrstream

    // example usage of getline
    getline(in, s); // reads from stream in and puts into string s

    // example declaration of a string stream
    istrstream input(s.c_str()); // binds string stream input to string s

PROBLEM 1 :  (Output: (8 pts))

Consider the following function.

void Mystery(int value)
{
    if (value > 5)
What is output for the call \texttt{Mystery(11)}?

**PROBLEM 2 : (Count Those Words: (20 pts))**

This problem is concerned with reading words from a data file and keeping track of information about certain words.

**PART A (12 pts):**

Write the function \texttt{GetCountsPerLine} whose header is given below. \texttt{GetCountsPerLine} is given an input stream that is ready for reading, a character letter and an array \texttt{counts}. \texttt{GetCountsPerLine} reads input from the input stream and counts the number of words on each line that start with the character letter, storing these counts in the array.

For example, if the data file was the following:

\begin{verbatim}
carl campbell eats cookies
carl campbell loves to crunch cookies
yum yum yum
chocolate chip cookies are his favorite
\end{verbatim}

Then the call \texttt{GetCountsPerLine(input, 'c', counts)} would store the counts of the number of words on each line that start with the letter 'c', resulting in (line 0 has 3 such words, line 1 has 4 such words, line 2 has 0 such words, etc.):

\begin{verbatim}
   3 4 0 3
\end{verbatim}

\texttt{void GetCountsPerLine(istream & input, char letter, tvector<int> & counts)}
// precondition: input is open and ready for reading, counts.size() == 0
// postcondition: counts contains the count of words on each line that start
// with the character letter
{

**PART B (8 pts):**

Write the function \texttt{LineOfKthWord} whose header is given below. \texttt{LineOfKthWord} is given an array containing the number of words per line for special words and a number k and returns the line number of the kth special word. Line numbers start with 0.
For example, using \texttt{counts}, which contains the number of words per line that start with the letter 'c' from Part A, then \texttt{LineOfKthWord(counts,3)} is 0, since the 3rd word counted ("cookies") in the array counts is on line 0 (the first line is 0). \texttt{LineOfKthWord(counts,8)} is 3, since the 8th word counted ("chocolate") in the array counts is on line 3.

Complete the function \texttt{LineOfKthWord} below.

\begin{verbatim}
int LineOfKthWord(const tvector<int> & counts, int k)
// precondition: counts.size() > 0, 0 < k <= total number of words counted in counts
// postcondition: returns the line number of the kth word counted in counts
{

PROBLEM 3 : (Numbers in a Matrix (20 pts))

The following problem refers to integers in a matrix.

\textbf{PART A} (9 pts):
Write the function \texttt{LargestInRow} whose header is given below. \texttt{LargestInRow} returns the largest value in the specified row.

For example, consider the matrix \texttt{A}:

\begin{verbatim}
2 3 7 6
3 1 3 2
1 3 7 3
\end{verbatim}

The call \texttt{LargestInRow(A,0)} returns 7, and the call \texttt{LargestInRow(A,1)} returns 3.

Complete the function \texttt{LargestInRow} below.

\begin{verbatim}
int LargestInRow(const tmatrix <int> & numbers, int row)
// precondition: numbers.numcols() > 0, 0 <= row < numbers.numrows()
// postcondition: returns the maximum value in the specified row
{

}\end{verbatim}

\textbf{PART B} (11 pts):
Write the function \texttt{Replace} whose header is given below. \texttt{Replace} replaces all entries of the number 3 by the number 4 EXCEPT if the number 1 is in the entry position immediately to the left.

For example, consider the matrix \texttt{A}:

\begin{verbatim}
2 3 7 6
3 1 3 2
1 3 7 3
\end{verbatim}

After the call \texttt{Replace(A)}, \texttt{A} has 3 of its 3's replaced by a 4. Two 3's were not replaced because there was a 1 in the entry to the immediate left of the 3. Here is the resulting matrix \texttt{A}:

\begin{verbatim}
3
\end{verbatim}
Complete the function Replace below.

```cpp
void Replace(tmatrix<int> & numbers)
// precondition: numbers.numcols() > 0, numbers.numrows() > 0
// postcondition: Replace all 3's by 4's except if 1 comes immediately
// before 3 in the row
{
}
```

**PROBLEM 4:** *(Lurking in the Meal Hall: (26 pts))*

A class `Meals` is designed to keep track of food items that Duke students have eaten. A struct `Food` saves information about a food item: the name of the food item, the ID of the student who purchased it and the date it was purchased.

Here are the definitions of the `Food` struct and `Meals` class.

```cpp
struct Food
{
    string itemName; // item name of food
    string DukeID;   // Student ID purchasing food item
    string date;     // date food item was purchased format: mm/dd/yyyy

    Food();
    Food(const string & name, const string & ID, const string & d);
};

class Meals
{
    public:
    Meals();
    int QuantityItemOnDate(const string & item, const string & date);
    bool ItemEatenOnDate(const string & item, const string & studentID,
                         const string & date);
    bool ItemsEatenSameDay(const string & item1, const string & item2,
                           const string & studentID);

    // other member functions not shown

    private:
```
Below is an example usage of this class that prints the number of people eating Shell Pasta on 1/20/2001.

Meals Market;
    cout << "Number eating Shell Pasta on 01/20/2001 is ";
    cout << Market.QuantityItemOnDate("Shell Pasta", "01/20/2001") << endl;

The output corresponding to this example might be (assuming 50 people ate Shell Pasta on this date):

Number eating Shell Pasta on 01/20/2001 is 50

Note that the vector myFood contains one entry for each item of food purchased. For example, here are a four of the entries stored in myFood, including 3 entries for a student with id 0278452.

<table>
<thead>
<tr>
<th>Asparagus</th>
<th>Shell Pasta</th>
<th>Cheese and Tomato Pizza</th>
<th>Shell Pasta</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>0278452</td>
<td>0265781</td>
<td>0278452</td>
<td>0278452</td>
<td>...</td>
</tr>
<tr>
<td>01/25/2001</td>
<td>03/05/2001</td>
<td>02/12/2001</td>
<td>01/25/2001</td>
<td>...</td>
</tr>
</tbody>
</table>

PART A (8 pts):
Write the function QuantityItemOnDate whose header is given below. QuantityItemOnDate is given a food item and a date and returns the number of such food items that were purchased on the given date.

The example on the previous page, the call Market.QuantityItemOnDate("Shell Pasta", "01/20/2001") outputs the number of entries of Shell Pasta for 1/20/2001.

Complete QuantityItemOnDate below.

```cpp
int Meals::QuantityItemOnDate(const string & item, const string & date)
// precondition: date must be in the format: mm/dd/yyyy
// postcondition: returns the number of servings of item on given date
{
}
```

PART B (8 pts):
Write the function ItemEatenOnDate whose header is given below. ItemEatenOnDate is given a food item, a student ID and a date and returns true if the student ate the food item on the given date. Otherwise, it returns false.

For example, if we show just part of the array myFood below, then the call Market.ItemEatenOnDate("Shell Pasta", 0278452, "01/25/2001") returns true.
Complete ItemEatenOnDate below.

```cpp
bool Meals::ItemEatenOnDate(const string & item, const string & studentID, const string & date)
// precondition: date is in the format mm/dd/yyyy
// postcondition: returns true if item was eaten by student with studentID on given date
{
}
```

**PART C (10 pts):**

Write the function ItemsEatenSameDay whose header is given below. ItemsEatenSameDay is given two food items and a student id and returns true if the student with the student id ever purchased item1 and item2 on the same day.

For example, if we show just part of the array myFood below, then the call

```cpp
Market.ItemsEatenSameDay("Shell Pasta", "Asparagus", "0278452")
```

returns true since the student purchased both on 1/25/2001.

<table>
<thead>
<tr>
<th>Asparagus</th>
<th>Shell Pasta</th>
<th>Cheese and Tomato Pizza</th>
<th>Shell Pasta</th>
</tr>
</thead>
<tbody>
<tr>
<td>0278452</td>
<td>0265781</td>
<td>0278452</td>
<td>0278452</td>
</tr>
<tr>
<td>01/25/2001</td>
<td>03/05/2001</td>
<td>02/12/2001</td>
<td>01/25/2001</td>
</tr>
</tbody>
</table>

In writing ItemsEatenSameDay you may call the function ItemEatenOnDate from Part B. You may assume ItemEatenOnDate works correctly regardless of what you wrote in Part B.

```cpp
bool Meals::ItemsEatenSameDay(const string & item1, const string & item2, const string & studentID)
// post: returns true if student with ID studentID ever ate item1 and item2 on the same day. Otherwise returns false.
{
}
```