

Dr. Rodger

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

```

// examples of use
//     tvector<int> v1;           // 0-element vector
//     tvector<int> v2(4);       // 4-element vector
//     tvector<int> v3(4, 22);   // 4-element vector, all elements == 22.

int length() const;           // returns capacity
void resize( int newSize );   // change size to newSize

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 istringstream

```

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

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

```

PROBLEM 1 : (*Mystery Mystery Mystery*: (10 pts))

Give the output of the following program.

```
int Mystery(int num)
{
    if (num > 0)
    {
        if (Mystery(num-2) > 0)
        {
            cout << num << " ";
        }
    }
    return 2;
}

int Something(int num)
{
    if (num < 0)
    {
        cout << num << " ";
        return Something(num+5);
    }
    return 1;
}

int main()
{
    cout << "Mystery(3)" << endl;
    cout << Mystery(3) << endl;
    cout << "Something(-12)" << endl;
    cout << Something(-12) << endl;
    return 0;
}
```

Output

PROBLEM 2 : (*Programming Contest*: (18 pts))

This problem uses 2-dimensional arrays to represent teams at a programming contest. Each row represents a team, and each column represents a problem to solve. A value of 1 in the array means the team solved that problem, and a 0 in the array means the team did not solve that problem.

PART A (*8 pts*):

Complete the function *NumSolved* whose header is shown below. The function *NumSolved* is given a 2-dimensional array and a row number representing a team, and it returns the number of programming problems solved by this team.

Consider the following 2d-array named *teams* representing 5 teams (rows 0 through 4) and 8 problems (cols 0 through 7).

```

0 0 0 0 1 0 0 0
0 1 0 1 0 0 0 1
1 0 0 0 1 0 0 0
0 1 0 0 1 0 0 1
0 1 0 0 0 0 0 0

```

The call `NumSolved(teams, 1)` would return 3 (row 1 contains 3 1's), and the call `NumSolved(teams, 2)` would return 2.

```

int NumSolved(const tmatrix<int> & teams, int row)
// pre: all entries in teams are 0 or 1, 1 indicates the problem was solved
// post: returns the number of problems solved by the team in row
{

}

```

PART B (10 pts):

Complete the function *MaxNumSolved* whose header is shown below. The function *MaxNumSolved* is given a 2-dimensional array and it returns the maximum number of problems solved by any team.

Using the 2d-array named *teams* in Part A, the call `MaxNumSolved(teams)` would return 3, the maximum number of problems solved by any team.

In writing `MaxNumSolved`, consider calling the function `NumSolved` that you wrote in Part A. Assume `NumSolved` works correctly, regardless of what you wrote in Part A.

```

int MaxNumSolved(const tmatrix<int> & teams)
// post: returns the maximum number of problems solved by any team
{

}

```

PROBLEM 3 : (*Is there a Winner?:* (40 pts))

In this problem we will tally student votes for President. A struct named *Person* has been declared for each student to keep track of the name of the student and the name of the candidate the student voted for. A class called *Poll* has been declared to determine information such as the number of candidates and how many votes per candidate.

```

struct Person
{
    string cand;    // candidate
    string first;  // first name of student
    string last;   // last name of student

    Person();      // constructor
    Person(string cand, string firstname, string lastname); // constructor
};

```

```

Person::Person()
{ }

Person::Person(string candidate, string firstname, string lastname)
    : cand(candidate), first(firstname), last(lastname)
{
    // all work in the initializer list
}

class Poll
{
public:
    Poll();                // constructor
    void ReadIn();         // read in data
    int Count(const string & name); // returns number of votes for name
    tvector<string> Candidates(); // returns array of candidates
    bool IsCandidate(const tvector<string> & allnames, const string & name); // returns true if name is a candidate

private:
    tvector <Person> myStudents; // array of student info
    int myNumStudents;         // number of students
};

```

Below is a sample data file containing info on six students. Each line has the candidate first (one word) followed by the student's name on the rest of the line.

```

Bush Lauren Adele Mary Sue Gallagher
Bush Michael Grayson France
Gore Yozu Kelly Jo Lee
Rodger Jason Daniel Patrick Cole Jeffers Robins
Gore Arjun Menon
Gore Charles Patton Wilkinson

```

Here is a sample usage of this class that prints the candidates and the number of votes.

```

int main()
{
    Poll Duke;
    int k;
    Duke.ReadIn();

    tvector<string> cand = Duke.Candidates();
    int num = cand.size();
    cout << "Candidates for President are: " << endl;
    for (k=0; k < num; k++)
    {
        cout << cand[k]<< " " << Duke.Count(cand[k]) << " votes" << endl;
    }
    cout << endl;
}

```

```
    return 0;
}
```

Below is the corresponding output:

```
Candidates for President are:
Bush 2 votes
Gore 3 votes
Rodger 1 votes
```

PART A (4 pts):

The *Poll* constructor should initialize all the values in the private section. Complete the constructor function.

```
Poll::Poll()
```

PART B (10 pts):

Complete the member function *ReadIn* whose header is shown below. The function *ReadIn* should read in the data from *cin* and place the information in the *myStudents* array.

Each line of data has the candidate first (one word) followed by the student's name on the rest of the line. A student's name contains 2 or more words. Only the first and last name of each student should be stored in the array. For example, for the first line of data on the previous page *Bush* is the candidate, *Lauren* is the first name, and *Gallagher* is the last name. Hint: Use a string stream.

```
void Poll::ReadIn()
// post: reads in data into myStudents. For each student puts in
// the candidates name (one word) and the first and last name only
// of the student. Updates myNumStudents.
{
}
}
```

PART C (8 pts):

Complete the member function *Count* whose header is shown below. *Count* returns the number of student votes for the name.

For example, for the data file at the beginning of this problem, *Count("Gore")* would return 3.

```
int Poll::Count(const string & name)
// post: returns the number of votes for name
{
}
}
```

PART D (8 pts):

Complete the member function *IsCandidate* whose header is shown below. The function *IsCandidate* is given an array and a name and it returns true if the name appears in the array, otherwise it returns false.

```
bool Poll::IsCandidate(const tvector<string> & allnames, const string & name)
// post: returns true if name appears in allnames, otherwise returns false
{

}
}
```

PART E (10 pts):

Complete the member function *Candidate* whose header is shown below. The function *Candidate* returns an array containing the names of all the candidates students voted for, each candidate name appears only once in the array returned. In the main function example at the beginning of this problem, an array with 3 names is returned (note: `cand.size()` has the value 3).

You may call any of the previous functions you wrote for the class. Assume they work correctly, regardless of what you wrote.

Hint: create a vector, fill it using `push_back`, and return it.

```
tvector<string> Poll::Candidates()
// post: returns an array of candidate names, one entry per candidate,
// the array contains size() entries
{

}
}
```