CSCI 135 Software Design and Analysis, C++
Homework 2
Due 2/21/2014

Saad Mneimneh
Hunter College of CUNY

Problem 1: Walking in Manhattan (well, sort of...)
Consider the following triangular grid that grows indefinitely. You start at the bottom left corner $(0, 0)$ and you follow a path by making a series of UP and RIGHT steps on the grid. Your goal is to get to a point (n, n).

```
                            (7,7) o
                                  |
                        (6,6) o---o
                              |   |
                    (5,5) o---o---o
                          |   |   |
                (4,4) o---o---o---o
                      |   |   |   |
            (3,3) o---o---o---o---o
                  |   |   |   |   |
        (2,2) o---o---o---o---o---o
              |   |   |   |   |   |
    (1,1) o---o---o---o---o---o---o
          |   |   |   |   |   |   |
(0,0) o---o---o---o---o---o---o---o
```

Write a program to do the following:

(a) reads from the input the value of n.

(b) reads $2n$ characters, each of which is either u or U (for UP), or r or R (for RIGHT).

Note 1: while this will require a loop to execute a cin statement $2n$ times, you can still input all the characters in one shot as a string of length $2n$. Regardless of how you do it, the program will block until you have entered $2n$ characters.

(c) outputs exactly one of the following in order of priority:

- "the path contains invalid steps" if any of the $2n$ characters is not in the set {'u', 'U', 'r', 'R'}.
- "you fell off the edge", if the path takes you outside the grid.
- whether the path takes you to your destination (n, n) or not.

Note 2: you don't have to store $2n$ characters (we did not cover arrays yet). In fact, you will use the same variable to read all $2n$ characters. So you will do something like:

```
char c;
for( ... ) {
  cin>>c;

  ...

}
```

But you will have to keep track of some quantities as you go along. An important technique for this problem is *flags*. Flags are boolean variables that tell you if something happens or not. For example:

```
char c;
bool flag=false;
for ( ... ) {
  cin>>c;
  if ( ... ) //some condition on c
    flag=true;

  ...

}

//done with reading all characters
if (flag) {

 //do whatever you want

}
```

For this problem, think of three flags:

- is the path valid?
- did I fall off the edge
- did I reach my destination?

Solution:

```
#include <iostream>
using std::cout;
using std::cin;

int main() {
  int n;
  cout<<"input the grid size: ";
  cin>>n;
  cout<<"input your path consisting of "<<2*n<<" steps: ";
  char c;

  //keep separate counts for up and right moves
  int u=0; //counts up moves
  int r=0; //counts right moves

  //some flags
  bool invalid=false;
  bool crossed=false;
  bool destination=false;

  //read 2n characters
  for (int i=1; i<=2*n; i=i+1) {
    cin>>c;
    if (c!='r' && c!='R' && c!='u' && c!='U') //path not valid
      invalid=true;
    if (c=='u' || c=='U') //increment up
      u=u+1;
    if (c=='r' || c=='R') //increment right
      r=r+1;
    if (u>r) //at any time, more ups than rights means we crossed the grid
      crossed=true;
  }
  if (u==r)  //reached destination, u=r=n
    destination=true;

  if (invalid)
    cout<<"invalid steps in path\n";
  else if (crossed)
    cout<<"you crossed the diagonal\n";
  else if (!destination)
    cout<<"you did not reach your destination\n";
  else
    cout<<"congratulations, you reached your destination\n";
}
```

Another way:

```
#include <iostream>
using std::cout;
using std::cin;

int main() {
  int n;
  cout<<"input the grid size: ";
  cin>>n;
  cout<<"input your path consisting of "<<2*n<<" steps: ";
  char c;

  //keep separate counts for up and right moves
  int u=0; //counts up moves
  int r=0; //counts right moves

  bool crossed=false;

  //read 2n characters
  for (int i=1; i<=2*n; i=i+1) {
    cin>>c;
    if (c=='u' || c=='U') //increment up
      u=u+1;
    if (c=='r' || c=='R') //increment right
      r=r+1;
    if (u>r) //at any time, more ups than rights means we crossed the grid
      crossed=true;
  }

  if (u+r<2*n) //some steps where neither u/U nor r/R
    cout<<"invalid steps in path\n";
  else if (crossed)
    cout<<"you crossed the diagonal\n";
  else if (u!=r)
    cout<<"you did not reach your destination\n";
  else
    cout<<"congratulations, you reached your destination\n";
}
```

Problem 2: Doomsday
For this problem, you will implement John Conway's Doomsday algorithm to determine which day of the week a given date is. The algorithm is based on the observation that all of these dates fall on the same day (John Conway calls it doomsday):

Regular year:
1/3, 2/0, 3/0, 4/4, 5/9, 6/6, 7/11, 8/8, 9/5, 10/10, 11/7, 12/12.

Leap year:
1/4, 2/1, 3/0, 4/4, 5/9, 6/6, 7/11, 8/8, 9/5, 10/10, 11/7, 12/12.

where 0 simply means the last day of the previous month.

If one can determine the doomsday in a given year, it will be easy to determine on which day of the week a given date falls. For example, the doomsday for 2014 is Friday. Also, 2014 is a regular year (not leap). So 1/3, 2/0, 3/0, 4/4, 5/9, 6/6, 7/11, 8/8, 9/5, 10/10, 11/7, and 12/12 are all Fridays. Given this information, let's say we want to know what day 2/15/2014 is. We identified 2/0/2014 as a Friday, so 2/14/2014 must be also a Friday (Happy Valentine's day), so 2/15/2014 must be a Saturday. It is a simple computation of offset modulo 7 in terms of days. Here the offset between 15 and 0 is 15, which is 1 modulo 7. So we add one day to Friday, we get Saturday.

(a) Implement a function called dayofmonth that accepts the month and the year as parameters and returns the doomsday corresponding to that month (as shown above). *Hint*: you will need to check if the year is a leap year or not (review lab exercise).

```
bool leapyear(int y) {
  return (y%4==0 && (y%100!=0 || y%400==0));
}
```

```
int dayofmonth(int m, int y) {
  if (m==3)
    return 0;
  else if (m==4 || m==6 || m==8 || m==10 || m==12)
    return m;
  else if (m==5)
    return 9;
  else if (m==9)
    return 5;
  else if (m==7)
    return 11;
  else if (m==11)
    return 7;
  else if (leapyear(y))
    if (m==1)
      return 4;
    else
      return 1;
  else
    if (m==1)
      return 3;
    else
      return 0;
}
```

(b) Assume Sunday is 0, Monday is 1, Tuesday is 2, Wednesday is 3, Thursday is 4, Friday is 5, and Saturday is 6. Write a function called doomsday that takes the year as a parameter and returns the doomsday for that year. Here's the algorithm (all divisions are integer divisions).

$$a \leftarrow (\mathit{year}/100) \bmod 4$$

$$b \leftarrow \mathit{year} \bmod 100$$

$$c \leftarrow b/12$$

$$d \leftarrow b \bmod 12$$

$$e \leftarrow d/4$$

$$\mathit{doomsday} \leftarrow (c + d + e + 5a + 2) \bmod 7$$

Solution:

```
int doomsday(int y) {
  int a=(y/100)%4;
  int b=y%100;
  int c=b/12;
  int d=b%12;
  int e=d/4;
  return (c+d+e+5*a+2)%7;
}
```

(c) Write a function called dayofweek that accepts the month, the day, and the year as parameters and returns the day of the week. Your function will make use of the previous two, and you will need to compute the offset between the given day and the result of dayofmonth. To avoid a negative offset, add a large enough multiple of 7; e.g. 14 (the modulo operator in C++ does not work correctly for negative numbers).

Solution:

```
int dayofweek(int m, int d, int y) {
  return (doomsday(y)+d-dayofmonth(m, y)+14)%7;
}
```

(d) If everything works fine, have fun with it in main! For instance, try to find out on what day you were born.

Solution:

```
int main() {
  int m;
  int d;
  int y;
  cout<<''input a date: month day year'';
  cin>>m>>d>>y;
  cout<<''this date falls on a ``<<dayofweek(m, d, y)<<'\n';
}
```

Problem 3: The 3n+1 problem (revisited)
We have seen the collatz function in class:

```
int collatz(int n) {
  if (n%2==0)
    return n/2;
  else
    return 3*n+1;
}
```

```
int main() {
  int n;
  cin>>n;
  while (n!=1)
    n=collatz(n);
}
```

It is conjectured that, if we start with any positive integer, repeatedly applying the collatz function will eventually brings us to 1. So the condition to exit the while loop will always be met. For instance, if we start with 10, we will go through the following sequence:

$$10 \rightarrow 5 \rightarrow 16 \rightarrow 8 \rightarrow 4 \rightarrow 2 \rightarrow 1$$

(a) Try the above program and input a value for n equal to 113383. The program seems to enter an infinite loop. Press Ctrl-C to stop it. Is this a counter example that disproves the conjecture? Modify the program to output the intermediate values in the sequence and try to understand what is happening and how to fix it.

Solution: the problem is due to the fact that, before reaching 1, some numbers on the way are larger than the maximum positive integer that int can represent. So numbers become negative. Once this happens, the collatz function can cycle indefinitely without reaching 1. This can be fixed if we use long unsigned int instead of int.

```
long unsigned int collatz(long unsigned int n) {
  if (n%2==0)
    return n;
  else
    return 3*n+1;
}
```

(b) Write a function called collatzCount that accepts an integer n and returns the number of steps needed to reach 1. For instance, collatzCount(10) should return 6.

Solution:

```
int collatzCount(int n) {
  int count=0;
  long unsigned int m=n;
  while (m!=1) {
    m=collatz(m);
    count=count+1;
  }
  return count;
}
```

(c) Using part (b), write a program in main to discover which number in $[1, 1000000]$ takes the longest to reach 1. How many steps does it require?

Solution:

```
int main() {
  int max=-1;  //maximum number of steps seen so far
  int i;  //the corresponding number
  for (int n=1; n<1000000; n=n+1) {
    int c=collatzCount(n);
    if (max<c) {
      max=c;
      i=n;
    }
  }
  cout<<i<<" takes the longest: "<<max<<" steps\n";
}
```