EECE 230 Introduction to Programming, Sections 3,4, and 12 Programming Assignment 10

Tuesday Dec 12, 2012

- This programming assignment consists of 4 problems.
- Never due, but material included on the final exam.
- Related material: classes, pointers and classes, templates, and 2-dimensional arrays.
- *Lab structure and regulations:*
  - ⋆ The 3 hours Lab session is on Tuesdays in Lab rooms 1,2 and 5 from 2:00 pm to 5:00 pm. It consists of three parts:
    - *Occasional Solving Session (not graded but attendance mandatory)*
    - *Programming Assignment (graded)*
    - *Occasional graded weekly quiz*
  - ⋆ You are supposed to submit your own work. Cheating will not be tolerated and will be dealt with severely: zero grades on the programming assignments, disciplinary committee, Dean's warning.
  - ⋆ Lab attendance is mandatory. Violating this rule can lead to a failing grade.

Problem 1. Quadratic polynomials class

In this problem, we are interested in quadratic polynomials, i.e., polynomials of degree 2.

A *quadratic polynomial* $p(x)$ on the variable x is of the form $p(x) = ax^2 + bx + c$, where a, b, c are real numbers. Thus p is uniquely specified by the three parameters a, b, and c.

Design a class *quadraticPolynomial* that defines a quadratic polynomial as an Abstract Data Type.

Include the member functions:

- a constructor which takes the values of a, b, and c as input parameters.
- default constructor which sets $a = b = c = 0$.
- *evaluate* function which given a real number v, returns the value of the polynomial on $x = v$, i.e., $av^2 + bv + c$.
- *print* function which prints the parameters of the polynomial in the following format:

  ```
  a*x^2 + b*x + c
  ```

- *realRoots* function which finds the real roots of the polynomial. This function returns an integers and it takes two reference parameters x_1 and x_2 as explained below.

  Recall that the roots of $p(x) = ax^2+bx+c$ are the solutions of the equation $p(x) = 0$. This equation has real solutions if and only if $\Delta = b^2 - 4ac \geq 0$.

  If $a \neq 0$ and $\Delta \geq 0$, the roots are given by $\frac{-b\pm\sqrt{\Delta}}{2a}$. In this cases the function *realRoots* returns 2 and it sets the reference parameters x_1 and x_2 to the values of the two roots (we set them to the same value if $\Delta = 0$).

  If $a \neq 0$ and $\Delta < 0$, *realRoots* returns 0.

  If $a = 0$ and $b \neq 0$, the polynomial has only one root $-c/b$. In this cases the *realRoots* returns 1 and it sets the reference parameter x_1 to the single root of the polynomial.

  If both a and b are zero, then the equation has either no solution if $c \neq 0$ or infinitely many solutions if $c = 0$. In the first case *realRoots* returns 0, and in the second it returns 999.

- *isSquare* function which returns TRUE if the polynomial is the square of a degree-1 polynomial and FALSE otherwise.
- *derivative* function which returns the derivative of the polynomial, i.e., the polynomial given by $2ax + b = 0x^2 + 2ax + b$.

Include also the (non-member) function:

- *sum* which takes two quadratic polynomials and returns their summation.

  Note that if $p(x) = ax^2 + bx + c$ and $q(x) = a'x^2 + b'x + c'$, then their summation is the polynomial given by $(a + a')x^2 + (b + b')x + (c + c')$.

Test your class using the following program:

```
int main()
{
   quadraticPolynomial p(1,5,3),q(1,2,1), r;
   cout<<"p: ";p.print();
   cout<<"q: ";q.print();
```

```
    cout<<"r: ";r.print();
    cout<<"p(2)="<<p.evaluate(2)<<endl;
    r = sum(p,q);
    cout<<"p+q: ";r.print();
    r = p.derivative();
    cout<<"derivative r of p: ";r.print();
    double x1,x2;
    if(p.realRoots(x1,x2)==2) cout<<"roots of p : "<<x1<<","<<x2<<endl;
    if(q.realRoots(x1,x2)==2) cout<<"roots of q : "<<x1<<","<<x2<<endl;
    if(r.realRoots(x1,x2)==1) cout<<"root of r : "<<x1<<endl;
    if(q.isSquare())  cout<<"q is a square";
    else cout<<"q is not a square";
    cout<<endl;
    if(p.isSquare())  cout<<"p is a square";
    else cout<<"p is not a square";
    cout<<endl;
    return 0;
}
```

You should get

```
p: 1*x^2 + 5*x + 3
q: 1*x^2 + 2*x + 1
r: 0*x^2 + 0*x + 0
p(2)=17
p+q: 2*x^2 + 7*x + 4
derivative r of p: 0*x^2 + 2*x + 5
roots of p : -0.697224,-4.30278
roots of q : -1,-1
root of r : -2.5
q is a square
p is not a square
```

Problem 2. myDynamicArray class revisited: Copy constructor and the Overload the assignment operator

Recall the class myDynamicArray [Programming Assignment 9, Problem 3]. Modify it as follows:

- Make A and *size* private
- Introduce interface public member functions to manipulate A and *size*:
    i. *get* function to read the i'th entry of A
    ii. *set* function to set the i'th entry of A
    iii. *getSize* function to read *size*

- Add a copy constructor

  ```
          myDynamicArray::myDynamicArray(const myDynamicArray & other);
  ```

  Note that the copy constructor is essential to: 1) pass instances of this class by value to functions and 2) enable functions to return instances of this class.

- Overload the assignment operator

  ```
  const myDynamicArray& myDynamicArray::operator=(const myDynamicArray & other);
  ```

Write a program to test your class.

Problem 3. Templates

This problem is a based on the code of various sorting functions we did in class or in previous Programming Assignments.

Write the template versions of the insertion sort, selection sort, merge sort, and randomized quick sort functions (parametrize the array type). Put the functions (with their implementations) in a header file called "mySortingAlgorithms.h". Add also the template version of the array print function to the header file.

All what you have to do is copy, paste, find, and replace....

Write a test program which includes the sorting algorithms header file and sorts arrays of various types (e.g., *int* and *double*) using each of the above algorithms.

Problem 4 Magic Square

Do Programming Exercise 9.13 [Malik, page 547] (page 494 in the second edition, page 547 in the third edition).

The problem statement is below.

12. **(Airplane Seating Assignment)** Write a program that can be used to assign seats for a commercial airplane. The airplane has 13 rows, with 6 seats in each row. Rows 1 and 2 are first class; the remaining rows are economy class. Also, rows 1 through 7 are nonsmoking. Your program must prompt the user to enter the following information:

    a. Ticket type (first class or economy class)
    b. For economy class, the smoking or nonsmoking section
    c. Desired seat

    Output the seating plan in the following form:

```
          A     B     C     D     E     F
Row  1    *     *     X     *     X     X
Row  2    *     X     *     X     *     X
Row  3    *     *     X     X     *     X
Row  4    X     *     X     *     X     X
Row  5    *     X     *     X     *     *
Row  6    *     X     *     *     *     X
Row  7    X     *     *     *     X     X
Row  8    *     X     *     X     X     *
Row  9    X     *     X     X     *     X
Row 10    *     X     *     X     X     X
Row 11    *     *     X     *     X     *
Row 12    *     *     X     X     *     X
Row 13    *     *     *     *     X     *
```

    Here, * indicates that the seat is available; X indicates that the seat is occupied. Make this a menu-driven program; show the user's choices and allow the user to make the appropriate choices.

13. **(Magic Square)** For this exercise, the functions written in parts a through e should be general enough to apply to an array of any size. (That is, the work should be done by using loops.)

    a. Write a function `createArithmeticSeq` that prompts the user to input two numbers, `first` and `diff`. The function then creates a one-dimensional array of `16` elements ordered in an arithmetic sequence. It also outputs the arithmetic sequence. For example, if `first` = `21` and `diff` = `5`, the arithmetic sequence is: `21 26 31 36 41 46 51 56 61 66 71 76 81 86 91 96`.

b. Write a function `matricize` that takes a one-dimensional array of 16 elements and a two-dimensional array of 4 rows and 4 columns as parameters. (Other values, such as the sizes of the arrays, must also be passed as parameters.) It puts the elements of the one-dimensional array into the two-dimensional array. For example, if `A` is the one-dimensional array created in part a and `B` is a two-dimensional array, then after putting the elements of `A` into `B`, the array `B` is:

```
21 26 31 36
41 46 51 56
61 66 71 76
81 86 91 96
```

c. Write a function `reverseDiagonal` that reverses both the diagonals of a two-dimensional array. For example, if the two-dimensional array is as in part b, after reversing the diagonals, the two-dimensional array is:

```
96 26 31 81
41 71 66 56
61 51 46 76
36 86 91 21
```

d. Write a function `magicCheck` that takes a one-dimensional array of size 16, a two-dimensional array of 4 rows and 4 columns, and the sizes of the arrays as parameters. By adding all the elements of the one-dimensional array and dividing by 4, this function determines the `magicNumber`. The function then adds each row, each column, and each diagonal of the two-dimensional array and compares each sum with the magic number. If the sum of each row, each column, and each diagonal is equal to the `magicNumber`, the function outputs "`It is a magic square`". Otherwise, it outputs "`It is not a magic number`". Do not print the sum of each row, each column, and the diagonals.

e. Write a function `printMatrix` that outputs the elements of a two-dimensional array, one row per line. This output should be as close to a square form as possible.

f. Test the functions you wrote for parts a through e using the following function `main`:

```
const int ROWS = 4;
const int COLUMNS = 4;

const int LIST_SIZE = 16;
...
```

```
int main()
{
    int list[LIST_SIZE];
    int matrix[ROWS][COLUMNS];

    createArithmeticSeq(list, LIST_SIZE);
    matricize(list, matrix, ROWS);
    printMatrix(matrix, ROWS);
    reverseDiagonal(matrix, ROWS);
    printMatrix(matrix, ROWS);
    magicCheck(list, matrix, LIST_SIZE, ROWS);

    return 0;
}
```