Homework #1

Oct. 2nd (Wed.) ~ Oct. 16th (Wed.) 23:59

*** Be sure to read the note on the last page.**

P1 (29pt). Implement n-ary large number operations using <u>operator overloading</u>.

Files to submit : HW1_NaryBigNum.h, HW1_NaryBigNum.cpp

NaryBigNum Class

- **Member Variable** (define as **Private**)
  - **char *number** : A pointer to large number stored in array
  - **int size** : The number of digits stored in array *number*
  - **int base** : The base (the number of unique digits) ranging from 2 to 36
- **Constructor**
  - **NaryBigNum()**
    - Initialize *number* to NULL, *size* to 0, and *base* to 10
  - **NaryBigNum(int n, string num)**
    - Convert *num* of string type to char array to store in *number,* and store its length in *size*
      - It is converted to uppercase for lowercase letters and converted to '0' for non-alphabetic, non-numeric characters, and characters out of range of expression; the converted results are stored in *number*

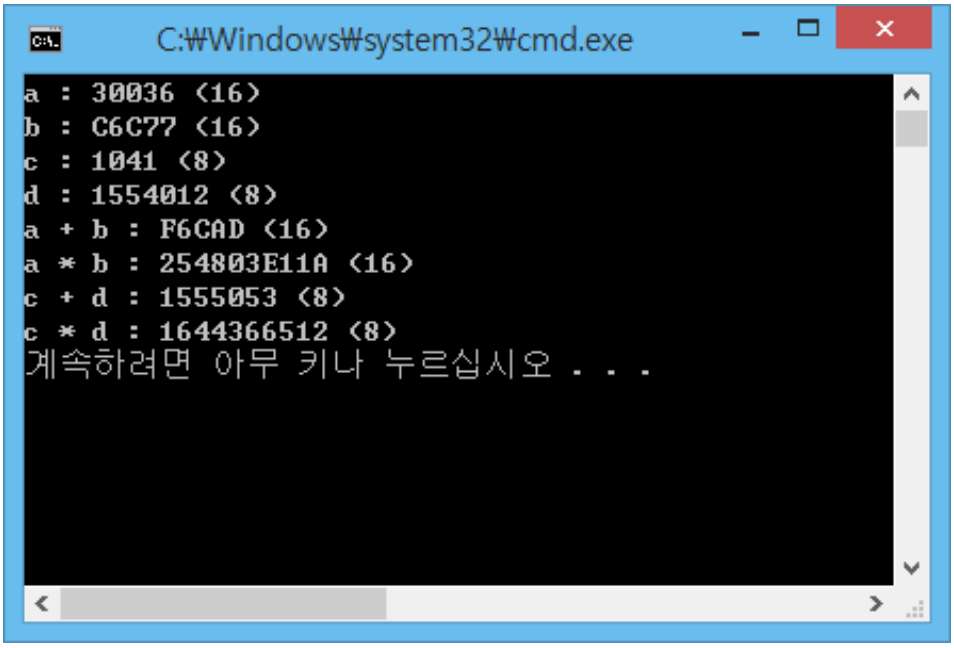
        e.g.) *num* = "1234abCD@", *n* = 13 → *number* = "1234ABC00"
      - The leftmost element of *number* cannot be zero, in other words, the size of *number* must be exactly the length of the number
    - Store *n* in *base*
      - *n* must be between 2 and 36; if *n* is out of range, 10 is stored in *base*

        e.g.) *n* = 40 → *base* = 10
    - Assume that *number* only accepts positive integers (excluding negative, zero, decimal, etc.)
  - **NaryBigNum(const NaryBigNum& nbn)**
    - Copy constructor; create a new object with the same number as *nbn*
- **Destructor**
  - **~NaryBigNum()**
    - Delete *number* (deallocate), and initialize *size* to 0 and *base* to 10

- **Operator Overloading**
  - **Operator =** : **NaryBigNum& operator= (const NaryBigNum& nbn)**
    - Assignment op.; delete existing data and create new object equal to *nbn*
  - **Operator +** : **NaryBigNum operator+ (const NaryBigNum& nbn)**
    - Return addition result of two NaryBigNum variables as **NaryBigNum** object
    - Assume that both operands have the same base
    - The leftmost element of the result cannot be zero
  - **Operator *** : **NaryBigNum operator* (const NaryBigNum& nbn)**
    - Return multiplication result of two NaryBigNum variables as **NaryBigNum** object
    - Assume that both operands have the same base
    - The leftmost element of the result cannot be zero
  - **Operator << : ostream& operator<< (ostream& os, const NaryBigNum& nbn)**
    - Print the number stored in *nbn* as follows:

      e.g.) *base* = 16, *number* = "123ABF" → 123ABF (16)

When HW1_NaryBigNum_Test.cpp is executed, the result is as follows.



P2 (20pt). Implement a Complex class and a global function that satisfies the following conditions.

Files to submit : HW1_Complex.h, HW1_Complex.cpp

Complex Class (template <class T>)

- **Definition**
  - A class representing a complex number $(a + bi)$
  - Assume that **T** only accepts numerical data types such as int, float and double
- **Member Variable** (define as **Private**)
  - **T re** : Real part (a of $a + bi$)
  - **T im** : Imaginary part (b of $a + bi$)
- **Constructor**
  - Take two numbers, one real and one imaginary, as parameters

    e.g.) **Complex<double>**(1.5, 3.4) stands for $1.5 + 3.4i$
  - **Default Constructor** : $0 + 0i$

    e.g.) **Complex<int>**() stands for $0 + 0i$
- **Operator Overloading**
  - **Operator +** : Return addition result of two Complex variables as **Complex<T>** object

    e.g.) $(3 + 2i) + (5 + 3i) = (8 + 5i)$
  - **Operator -** : Return subtraction result of two Complex variables as **Complex<T>** object

    (this computation doesn't satisfy commutative property)

    e.g.) $(3 + 2i) - (5 + 3i) = (-2 - i), (5 + 3i) - (3 + 2i) = (2 + i)$
  - **Operator *** : Return multiplication result of two Complex variables as **Complex<T>** object

    e.g.) $(3 + 2i) * (5 + 3i) = (9 + 19i)$
  - **Operator ~**: Return conjugate form of the complex number as **Complex<T>** object

    e.g.) $\sim(3 + 2i) = 3 - 2i$
  - **Operator <<** : Print in the form of $(a - bi)$ if $\mathrm{b} < 0$, otherwise $(a + bi)$

    e.g.) $(1 + 2i), (-1.08 + 3.24i), (-5 - 2i), (4.4 - 9.28i)$

Global function

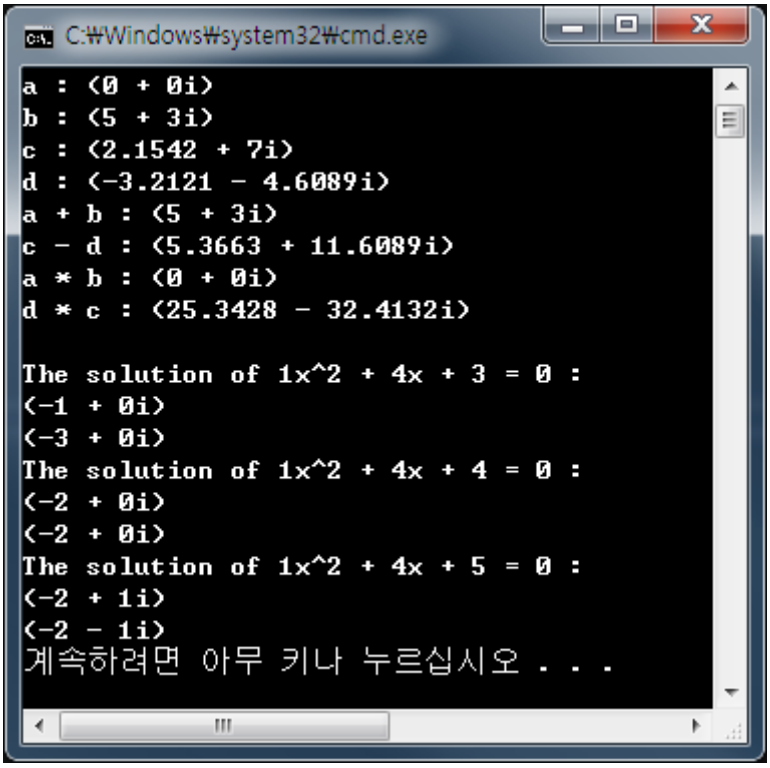
void solveQuadratic (Complex<double>& x1, Complex<double>& x2, int a, int b, int c)

- **Definition**
  - A function solving quadratic equation, $ax^2 + bx + c = 0$
- **Conditions**
  - Assume $a \neq 0$
  - Use quadratic formula for finding two solutions

$$ax^2 + bx + c = 0 \leftrightarrow x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

  - If the discriminant $D = b^2 - 4ac > 0$, *x1* and *x2* are real number, so *im* of *x1* and *x2* is 0
  - Else if $D = b^2 - 4ac = 0$, *x1* and *x2* are the same
  - Otherwise, *x1* and *x2* are complex numbers
  - Include 'math.h' library and use 'sqrt' functions for calculating square root

When HW1_Complex_Test.cpp is executed, the result is as follows.



P3 (23pt). Using given Swap() function, Implement an Array2D class that satisfies the following conditions.

Files to submit : HW1_Array2D.h, HW1_Array2D.cpp

Array2D Class

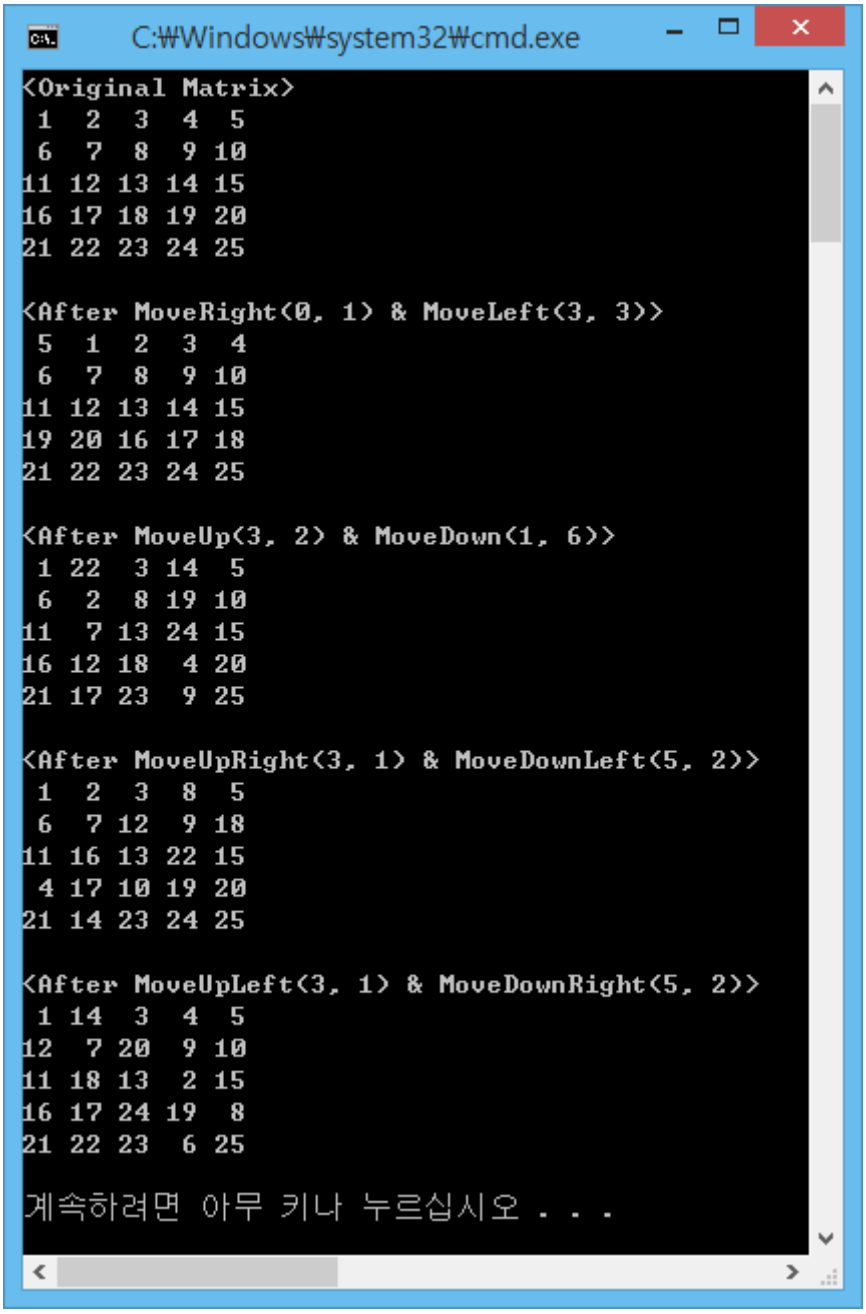
- **Member Variable** (define as **Private**)
  - **int **m_array** : 2D square array pointer
  - **int m_size** : The number of row (or column) of *m_array*
- **Constructor**
  - **Array2D(int size)**
    - Initialize *m_size* to *size*
    - Create a 2D int array of (*size* X *size*) in *m_array* and set each element value from 1 to (*size* X *size*) in order

      e.g.) *size* = 2 → *m_array* [0][0], [0][1], [1][0], and [1][1] are 1, 2, 3, and 4, respectively
- **Destructor**
  - **~Array2D()**
    - Delete *m_array* (deallocate), and initialize *m_size* to 0
- **Operator Overloading**
  - **Operator << : ostream& operator<< (ostream& os, const Array2D& arr)**
    - Print the array in the form of 2D array as follows:

      e.g.) *size* = 2 → **1 2**

      3 4
- **Member Function**
  - **void Swap (int* a, int* b)**
    - **int temp = *a; *a = *b; *b = temp;**
  - **void moveRight(int r, int dist) / void moveLeft(int r, int dist)**
    - Move r^{th} row right/left by *dist* using Swap() function ($0 \le r \le size - 1$)
  - **void moveUp(int c, int dist) / void moveDown(int c, int dist)**
    - Move c^{th} column up/down by *dist* using Swap() function ($0 \le c \le size - 1$)
  - **void moveUpRight(int d, int dist) / void moveDownLeft(int d, int dist)**
    - Move d^{th} diagonal ↗/↙ way by *dist* using Swap() function ($1 \le d \le 2 \cdot size - 3$)
  - **void moveUpLeft(int d, int dist) / void moveDownRight(int d, int dist)**
    - Move d^{th} diagonal ↖/↘ way by *dist* using Swap() function ($1 \le d \le 2 \cdot size - 3$)

When HW1_Array2D_Test.cpp is executed, the result is as follows.



P4 (26pt). Implement Figure class and its derived class, Rectangle and Circle, which satisfies the following conditions.

Files to submit : HW1_Figure.h, HW1_Figure.cpp

Figure Class

- **Member Variable** (define as **Protected**)
  - **float area** : Area of figure
- **Constructor**
  - **Figure()**
    - Initialize *area* to 0
- **Destructor**
  - **~Figure()**
    - Delete nothing but it should allow destructor of derived class to be executed
- **Operator Overloading**
  - **Operator << :** Print information of figure by calling print function
- **Member Function : It should allow overridden functions of derived class to be executed**
  - **void shift(int _x, int _y)**
  - **ostream& print(ostream& os)**

Rectangle Class – derived class of Figure as public

: Assume that its side are parallel with x- and y-axis

- **Member Variable** (define as **Private**)
  - **int width** : Width of rectangle
  - **int height** : Height of rectangle
  - **int* point** : Coordinate of its diagonal $(x1, y1, x2, y2)$ where $x1 < x2, y1 < y2$
- **Constructor**
  - **Rectangle()**
    - Initialize *width* and *height* to 0, and *point* to nullptr
  - **Rectangle(int x1, int y1, int x2, int y2)**
    - Calculate *width*, *height* and *area*, and store coordinates $(x1, y1, x2, y2)$ to *point*
- **Destructor**
  - **~Figure()**
    - Delete *point* (deallocate), and initialize *width* and *height* to 0

- **Member Function**
  - **void shift(int _x, int _y) :** Shift rectangle by *_x*, *_y* in the x- and y-directions respectively

    e.g.) *point* : (1, 2), (3, 4) ---shift(-1, -2)---> *point* : (0, 0), (2, 2)
  - **ostream& print(ostream& os) const :** Print information of rectangle

    e.g.) *width* : 2, *height* : 3, *point* : (0, 0), (2, 3) →

    Figure: rectangle

    Width: 2, Height: 3 -> Area: 6

    The coordinate of diagonal line: (0, 0), (2, 3)

Circle Class – derived class of Figure as public

- **Member Variable** (define as **Private**)
  - **int radius** : Radius of circle
  - **int* center** : (x, y) which representing the coordinate of center
- **Constructor**
  - **Circle()**
    - Initialize *radius* to 0, and *center* to nullptr
  - **Circle(int cx, int cy, int r)**
    - Store coordinates (cx, cy) to *center* and radius *r* to *radius*
    - Calculate *area* assuming $\pi = 3.14$
- **Destructor**
  - **~Circle()**
    - Delete *center* (deallocate), and initialize *radius* to 0
- **Member Function**
  - **void shift(int _x, int _y) :** Shift circle by *_x*, *_y* in the x- and y-directions respectively

    e.g.) *center* : (1, 2) ---shift(3, 1)---> *center* : (4, 3)
  - **ostream& print(ostream& os) :** Print information of rectangle

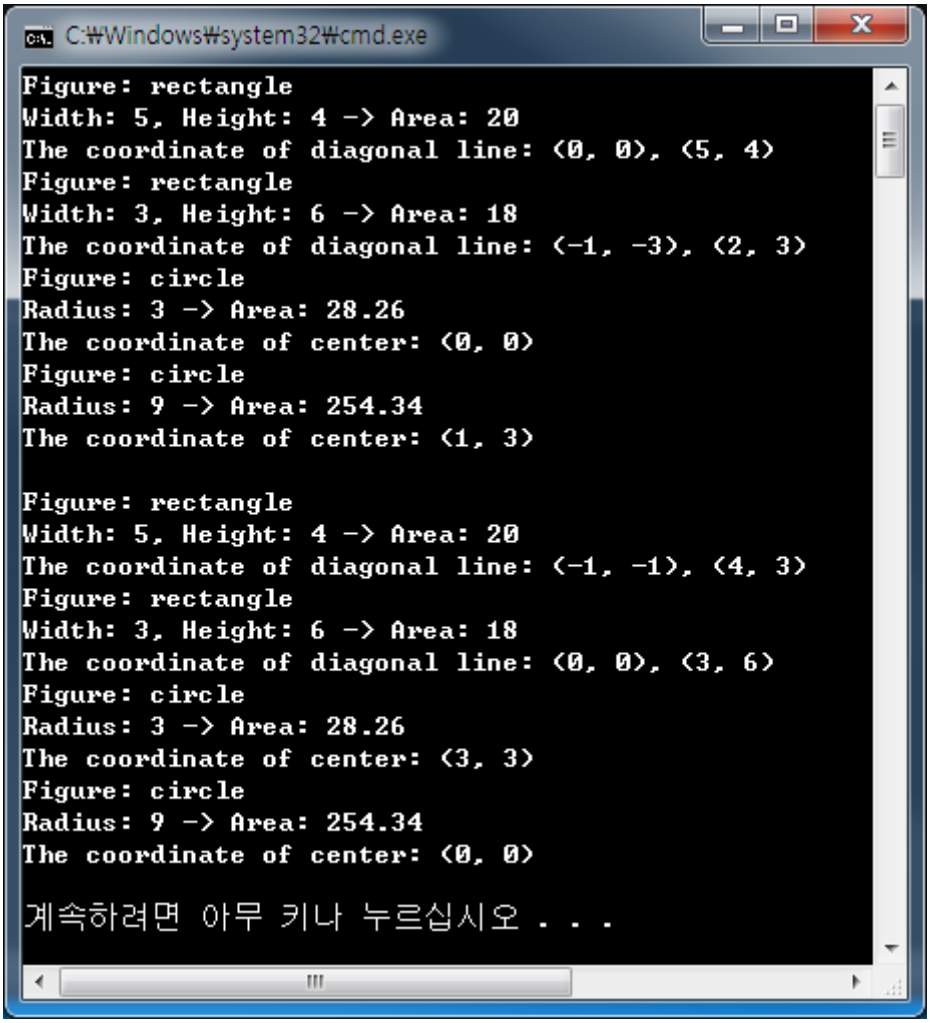
    e.g.) *radius* : 3, *center* : (1, 2) →

    Figure: Circle

    Radius: 3 -> Area: 28.26

    The coordinate of center: (3, 3)

When HW1_Figure_Test.cpp is executed, the result is as follows.



- **Note**
  - **Submit the files with the exact file names given! Otherwise score will be deducted.**
  - **Output should be in the same form as the example given! Otherwise score will be deducted.**
  - **Scoring will be done with more complex case than the example test case given.**
  - **No plagiarism! If plagiarism is detected, 0 points will be given for the assignment and it will be notified to the professor.**
  - **If you have any question, ask questions via e-mail only if they are not resolved after sufficient search has been done.**
- **How to submit**
  - Write code for each problem in each .h / .cpp file.
  - Files should be saved **with exact file names** given in each problem.
  - Compress your code into **one compressed file**.
    - The compressed file name should be "**HW1_(name)_(student ID).zip**" using zip compression.
    - e.g.) "**HW1_김태환_2017-11111.zip**"
  - Submit the compressed file to the "Assignment 1" on the eTL course page.
- **Deadline for submission**
  - **By Wednesday, October 16th, 11:59 pm.**
  - Submit the assignment **via e-mail within the deadline** if there is any problem when you submit it on eTL.
    E-mail : ds@snucad.snu.ac.kr
  - **No delay submission (both eTL / e-mail). 0 points for late submission or no submission.**
  - **Make sure that the file is attached and submitted! 0 points for submission without attachment.**