Program 4

Due Dec 8 by 11:59pm **Points** 30 **Submitting** a text entry box or a file upload
Available Nov 25 at 12am - Dec 8 at 11:59pm 14 days

This assignment was locked Dec 8 at 11:59pm.

Queues **- Application: Very Long Integer**

Purpose

This programming assignment implements a **LongInt** class to facilitate very long integers that cannot be represented with ordinary C++ **int** variables, using Deque.

Deque

Deque is an abbreviation for a "double ended queue". It allows a new data item to be enqueued not only to the back but also to the front. Similarly, it allows both the front and the back item to be retrieved and dequeued. The specifications of **Deque** class can be found **deque.h**

Only the difference from the ordinary FIFO queues is that **Deque** has three additional methods:

1. **getBack( )**: retrieves the tail object.
2. **removeBack( )**: removes the tail object.
3. **addFront(object)**: enqueues a new object to the front.

All the other methods are functionally identical to those of the FIFO queues.

deque.cpp partially includes the methods. You have to complete by implementing 4 missing methods, addFront, addBack, removeFront, and removeBack.

Very Long Integers

Some real-world applications such as public/private-key cryptography need to use integers more than 100 digits long. Those integer values are obviously not maintained in C++ **int** variables. In this programming assignment, you will design a **LongInt** class that maintains a very long integer with a **Deque** of charaters, (i.e., **char**s). In other words, each **char**-type item in **Deque** represents a different digit of a given integer; the front item corresponds to the most significant digit; and the tail item maintains the least significant digit. Look at the specifications from **longint.h**

Data Members

LongInt includes two private data members such as **digits** and **negative**. The former is a **Deque** instance that maintains a series of char-type data items, namely representing a very long integer. The latter indicates the sign. If **negative** is true, the given integer is less than 0. If the integer is 0, we don't care about **negative**.

Input

Your operator>> must read characters '0' through '9' from the keyboard as well as accept '-' if it is the first character. Skip all the other characters. Assume the following code segment:

```
LongInt a, b;
cout << "enter for a: ";
cin >> a;
cout << "enter for b: ";
cin >> b;
```

A user may type:

```
enter for a: 123
enter for b: -456
```

In this case, your operator>> should substitute a and b with 123 and -456 respectively.

Output

Your operator<< must print out a given **LongInt** object's digits as an integer. If the **LongInt** object's digits is empty or 0, it should be printed out as 0 without a negative sign. Wrong outputs include:

```
-0
00000
-000
```

Constructors/Destructor

Implement three constructions: (1) the one that reads a string to convert to a **Deque**, (2) the copy constructor, and (3) the default constructor that initializes the object to 0. The destructor should deallocate all **Deque** items.

Arithmetic Binary Operators

Implement operators + and -.

1. Operator+:
   First, consider four different cases:
   - **positive lhs + positive rhs** means ans = lhs + rhs.

- **positive lhs + negative rhs** means ans = lhs - rhs. You should call operator-.
- **negative lhs + positive rhs** means ans = rhs - lhs. You should call operator-.
- **negative lhs + negative rhs** means ans = -(lhs + rhs). Apply operator+ and thereafter set a negative sign.

- Examine **lhs** and **rhs** digit by digit from the tail of their deque, (i.e., from the tail of their **digits** data member) to the top. Prepare a zero-initialized integer named **carry**. Let **lhs'** i^{th} item from the deque tail be **lhs[i]**. Similarly, let **rhs** and **ans'** corresponding i^{th} item from the tail be **rhs[i]**and **ans[i]** respectively. The computation will be

```
ans[i] = ( lhs[i] + rhs[i] + carry ) % 10;
carry  = ( lhs[i] + rhs[i] + carry ) / 10;
```

2. Operator-:
First, consider four different cases:

- **positive lhs - positive rhs** means ans = lhs - rhs.
- **positive lhs - negative rhs** means ans = lhs + rhs. You should call operator+.
- **negative lhs - positive rhs** means ans = -(lhs + rhs). This corresponds to the 4th case of operator+.
- **negative lhs - negative rhs** means ans = rhs - lhs.

Examine **lhs** and **rhs** digit by digit from the tail of their deque, (i.e., from the tail of their **digits** data member) to the top. Prepare a zero-initialized integer named **borrow**. Consider by yourself how to compute each digit of **ans**, (i.e., **ans[i]**) with **lhs[i]**, **rhs[i]**, and **borrow**.

Assignment Operators

Copy **the digits** and the **sign**.

Logical Binary Operators

Implement operators <, <=, >, >=, ==, and !=. A comparison between a left-hand and a right-hand operands follows the steps below:

1. Compare their negative sign. If their signs are different, the operand with a negative sign is a smaller integer.
2. Compare their deque size. If their signs are both positive but their sizes are different, the operand with a larger deque size is a larger integer. If their signs are both negative, the operand with a larger deque size is a smaller integer.
3. Compare their deque elements from the front as removing them. The operand with a larger deque element is a larger integer in a positive sign but a smaller integer in a negative sign.

Statement of Work

There are two tasks you have to finish.

1. Complete deque class

- Copy the following files: (Files-->Progrms/Program4/)
  - **deque.h** :the header file of the **Deque** class
  - **deque.cpp** :the incompleted implementation of the **Deque** class
  - **deque_test.cpp** : Deque class test driver
- Complete Deque class by implementing the four missing methods. (addFront, addBack, removeFront, removeBack)
- Compile: **g++ deque_test.cpp**
- Save the output as deque_test_out.txt
- You should see the result as the following

```
deque1:
9
8
7
6
5
0
1
2
3
4
deque2:
10
4
3
2
1
0
5
6
7
8
9
```

2. Implement LongInt class using Deque class

- Copy the following files: (Files-->Progrms/Program4/)
  - **longint.h**: the header file of the **LongInt** class
  - **driver.cpp**: a main program
- Put them in the same folder with **deque.h**, and **your completed deque.cpp**
- Implement the **LongInt** class in **longint.cpp**
- Compile the program with **g++ longint.cpp driver.cpp**
- Run the **driver** program to verify your implementation. The driver.cpp asks you to type in four long integers, each assigned to the variable **a**, **b**, **c**, and **d**. Use **input.txt** for those values. That is,

  a=00012345678901234567890123456789 0

  b=-00012345678901234567890123456788 9

  c=-00000000000000000000000000000000 1

  d=00012345678901234567890123456788 9

- Check if your output is the same as **output.txt.**
- Copy and past your result from the screen to longint_test_out.txt
- Create your own testing driver, LongintDriver.cpp, that includes more testing cases than driver.cpp, and tests all methods thoroughly.
- Compile and run with your LongintDriver.cpp, and provide your output as MyTesting.txt

What to Turn in

Clearly state any other assumptions you have made. Your soft copy must include:

(1) deque.h, deque.cpp (completed deque), deque_test.cpp, deque_test_out.txt

(2) longint.h, longint.cpp, driver.cpp, LongintDriver.cpp, longint_test_out.txt, MyTesting.txt

(3) optional) If you have to provide your own input file, please submit it as well.

Grading Guide

Click the following grading guide to see how your homework will be graded.

```
Program 4 Grade Guideline

1. File submission (1 pts) :
     You are supposed to submit 3 output files.
     Any missing file: 0 pt
```

```
2. Testing cases. Will check LongintDriver.cpp and the output (2pts)
   Thorough testing cases (2pts)  Incomplete testing cases (1pt)   No additional testing cases (0pt)

3. Correctness (25 pts) (If any missing file causes compilation error, you won't get any points)
Compilation errors ( 0 pts)

   Successful compilation
  + Correct implementation of Deque methods ( 4 pts)
  From the LongInt class
  + Correct constructors (3 pts)
  + Correct destructor ( 1 pt)
  + Correct Operator+ (2 pts)
  + Correct  Operator-  (2 pts)
  + Correct Operator= (1 pt)
  + Correct  Operator< (2pt)
  + Correct  Operator<= (1pt)
  + Correct  Operator>  (2 pt)
  + Correct   Operator>= (1 pt)
  + Correct   Operator== (1 pt)
  + Correct Operator!= (1 pt)
  + Correct Operator>> (2 pt)
  + Correct Operator<< (2 pt)

3. Program Organization (2pts)
   Write comments to help the professor or the grader understand your
   pointer operations.

   Proper comments
   Good (1pts)       Poor/No explanations(0pt)

   Coding style (proper indentations, blank lines, variable names, and non-redundant code)
   Good (1pt)        Poor/No explanations(0pt)
```