CSS342 Data Structures, Algorithms, and Discrete Mathematics I

Autumn 2017

Assignment 6: Queues

Due date: Friday 15 Dec

Purpose

Using **Deque (**Double-ended queue), this programming assignment implements a **LongInt** class to facilitate very long integers that cannot be represented with ordinary C++ **int** variables.

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. Attached is **LongInt**'s header file (i.e., 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:
   1. **positive lhs + positive rhs** means ans = lhs + rhs.
   2. **positive lhs + negative rhs** means ans = lhs - rhs. You should call operator-.
   3. **negative lhs + positive rhs** means ans = rhs - lhs. You should call operator-.
   4. **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:
   1. **positive lhs - positive rhs** means ans = lhs - rhs.
   2. **positive lhs - negative rhs** means ans = lhs + rhs. You should call operator+.
   3. **negative lhs - positive rhs** means ans = -(lhs + rhs). This corresponds to the 4th case of operator+.
   4. **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 pointer 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 signes are both negative, the operand with a larger deque size is a smaller integer.
3. Compare their deque elemens 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

Files can be found in Canavas.

1. Copy the following files:
   - **deque.h**: the header file of the **Deque** class
   - **deque.cpp.h**: partial implementation of the **Deque** class (you need to implement the four add/remove functions)
   - **longint.h**: the header file of the **LongInt** class
   - **driver.cpp**: a main program
2. Implement the **LongInt** class in **longint.cpp**
3. Compile all the programs.
4. 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. Check if your output is the same as output.txt

What to Turn in

Clearly state any other assumptions you have made. Your soft copy must include: (1) all .h and .cpp files. The professor and the grader will compile your program with "g++ driver.cpp longint.cpp " for grading your work. (2) execution outputs in a .doc, .docx, or text file.

Grading Guide and Answers

Click the following grading guide to see how your homework will be graded. Key answer will be made available after the due date through Solution page.

```
Grade Guideline

1. Documentation (2pts)
   A 1-page output
   A correct output (2pts)    An output with a few errors (1pt)   An output
with many errors/No output (0pt)

2. Correctness (15pts)
Compilation erros ( 0 pts)

   Successful compilation (1 pts)
  + Correct Operator+  (2pts)
  + Correct Operator-  (2 pts)
```

+ Correct Operator< (1pt)
+ Correct Operator<= (1pt)
+ Correct Operator> (1 pt)
+ Correct Operator>= (1 pt)
+ Correct Operator== (1 pt)
+ Correct Operator!= (1 pt)
+ Correct Operator>> (1 pt)
+ Correct Operator<< (1 pt)
+ Correct Operator= (1 pt)
+ Correct LongInt(const string str) (1 pt)

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

   Proper comments
   Good (2pts) Poor(1pt) No explanations(0pt)

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