C++ Exercise 5: Circularly Linked Lists

Deadline: Monday, October 12, 2020 by 23:59:59 EEST Timezone.
How to submit: Check your code into your GitHub repository, the default `master` branch, tag it as `ex05submit` (all lowercase, no dashes).
Grading: This exercise is worth 30‰(or 3%) of the total grade.

Develop a software that receives a list of integers and adds them to a circular linked list (implemented as pointers). See Section 3.4.1 in the textbook.

Input
The first line contains number N (the number of integers to add into the initial circularly linked list). After that the input contains exactly N integers (not necessarily different) that fit into the `int` data type. All these integers are written on a single line. After that there are one or more operations that insert or delete things at certain places in the circularly linked list (we assume that the cursor does not change once the circularly linked list becomes nonempty).

Output
The output is a list of integers (starting from the circularly linked list's "front" element - pointed to by the cursor) after all the insert and delete operations performed.

Implementation Details

1. Implement files `CircleList.cpp` (and `CircleList.h`) that ensure circularly linked list of integers. Also implement `CircleListMain.cpp` to read input and write to output.
2. If you need any other C++ classes or structures (for a single Circularly Linked List node, custom exception `OutOfBoundsException` etc.) add these structures to your `CircleList.cpp` and `CircleList.h` respectively. We do not test any of them separately.
3. Implement the functions of a circularly linked list to support the ADT of this abstract data structure (see Figure 1).
4. Even if you do not need some of the functions (for insertion or deletion), make sure that they are still implemented - as we might use them in the next lab that will be continued next week.
5. Ensure that your code has destructors for all the data structures you dynamically create and that your code has no memory leaks.
6. If the command cannot be performed (insert or delete position is larger than the current size of the circularly linked list), thrown a custom exception `OutOfBoundsException`.

Sample input **test01in.txt**:

```
6
11 12 13 14 15 16
INS 2 1000
```

Expected output **test01expected.txt**:

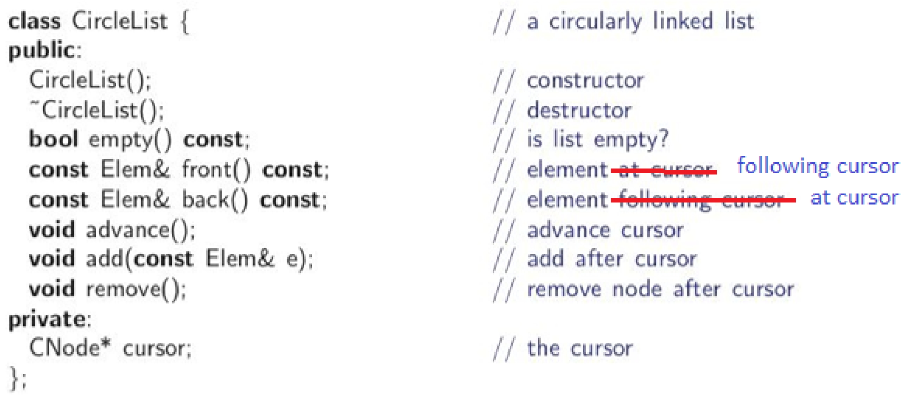
```
11 12 1000 13 14 15 16
```

Sample input **test02in.txt**:

```
6
11 12 13 14 15 16
DEL 2
```

Expected output **test02expected.txt**:

```
11 12 14 15 16
```



```
class CircleList {                          // a circularly linked list
public:
  CircleList();                             // constructor
  ~CircleList();                            // destructor
  bool empty() const;                       // is list empty?
  const Elem& front() const;                // element ~~at cursor~~ following cursor
  const Elem& back() const;                 // element ~~following cursor~~ at cursor
  void advance();                           // advance cursor
  void add(const Elem& e);                  // add after cursor
  void remove();                            // remove node after cursor
private:
  CNode* cursor;                            // the cursor
};
```

Figure 1: Circularly linked lists.

Sample input **test03in.txt**:

```
6
11 12 13 14 15 16
INS 0 100
DEL 2
```

Expected output **test03expected.txt**:

```
100 11 13 14 15 16
```

Sample input **test04in.txt**:

```
6
11 12 13 14 15 16
DEL 6
```

Expected output **test04expected.txt**:

```
OutOfBoundsException
```

Sample input **test05in.txt**:

```
6
11 12 13 14 15 16
INS 6 101
```

Expected output **test05expected.txt**:

```
11 12 13 14 15 16 101
```

Sample input **test06in.txt**:

```
6
11 12 13 14 15 16
INS 7 101
```

Expected output **test06expected.txt**:

```
OutOfBoundsException
```

Running Unit tests (Check2)

Inserting and deleting items from lists of integers can be implemented in many ways. The operations that we want (INS and DEL at certain positions in the list) can be easily expressed using the circularly linked list ADT (Figure 1).

To distinguish "high-level mistakes" in our code (e.g. handling input/output incorrectly or misinterpreting where we need to insert and delete) from "low-level mistakes" (wrongly implemented data structure), we suggest that you run Unit tests on `CircleList` classes

In fact, out of 30 points for this exercise, 10 points will be given for satisfying the unit tests. A few notes before you apply the `CircleList` class to the EX05 problem (inserting/deleting integers at certain positions):

1. The function `front()` returns integer value following the cursor, but `back()` returns the

integer value currently under the cursor. (This is explained in the textbook too, just the two comments have switched places on p.130).

2. In order to throw `OutOfBoundsException` whenever we insert at position that does not currently exist (and also is not next to one that currently exists), we need to know the current size of the circularly linked list. Therefore the `getSize()` method in the UML diagram (Figure 2). The same function is needed as we delete from that list.

3. Adding and removing elements only happen next to the head/cursor. If you need to add/remove in any other place, you need to call `advance()` certain number of times. And after the operation you need to return the list to the original cursor position. It can be achieved in two ways: Either you call `advance()` many times to run around the whole list. Or you can also implement `retreat()` that works in the opposite direction than `advance()` (head travels back using `prev` pointer in `CNote`.)

Our Unit tests are implemented using Catch2 framework; see `https://github.com/catchorg/Catch2`. It means that you download `catch.hpp` and place it in your project directory. Tests are organized into testcases and sections. Each of testcases (or sections therein) can call your `CircleList` implementation.
Since this file (`Catch2TestRunner.cpp`) creates its own `main()` method, you cannot run it in the same executable as the one created by the `CircleListMain.cpp`. Please refer to the `Makefile` and the testcases `Catch2TestRunner.cpp` for details - under **EX05** lab section in
`http://linen-tracer-682.appspot.com/data-structures/assignments.html`.

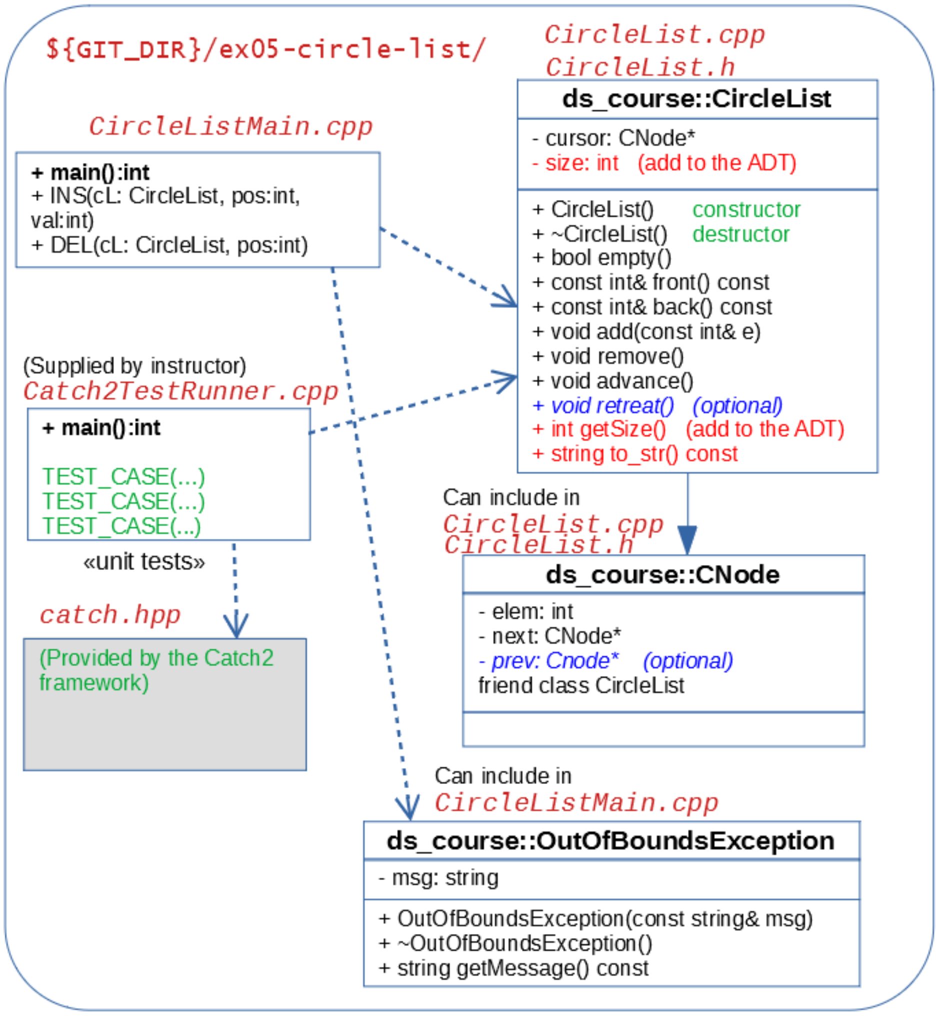

${GIT_DIR}/ex05-circle-list/
CircleList.cpp
CircleList.h
ds_course::CircleList
- cursor: CNode*
- size: int (add to the ADT)
+ CircleList() constructor
+ ~CircleList() destructor
+ bool empty()
+ const int& front() const
+ const int& back() const
+ void add(const int& e)
+ void remove()
+ void advance()
+ void retreat() (optional)
+ int getSize() (add to the ADT)
+ string to_str() const
CircleListMain.cpp
+ main():int
+ INS(cL: CircleList, pos:int, val:int)
+ DEL(cL: CircleList, pos:int)
(Supplied by instructor)
Catch2TestRunner.cpp
+ main():int
TEST_CASE(...)
TEST_CASE(...)
TEST_CASE(...)
«unit tests»
catch.hpp
(Provided by the Catch2 framework)
Can include in
CircleList.cpp
CircleList.h
ds_course::CNode
- elem: int
- next: CNode*
- prev: Cnode* (optional)
friend class CircleList
Can include in
CircleListMain.cpp
ds_course::OutOfBoundsException
- msg: string
+ OutOfBoundsException(const string& msg)
+ ~OutOfBoundsException()
+ string getMessage() const

Figure 2: Classes for EX05.