CSCE 221 Spring 2021

Programming Assignment 2

Lists, Stack, and Queue

Approved Includes

```
<cstddef>
<iostream>
<stdexcept>
<utility>
"array_list.h"
"doubly_linked_list.h"
"stack.h"
"queue.h"
```

Code Coverage

You must submit a test suite for each task that, when run, covers at least 90% of your code. You should, at a minimum, invoke every function at least once. Best practice is to also check the actual behavior against the expected behavior, e.g. verify that the result is correct.

Starter Code

```
array_list.h
array_list_tests.cpp
doubly_linked_list.h
doubly_linked_list_tests.cpp
stack.h
stack_tests.cpp
queue.h
queue_tests.cpp
compile_test.cpp
Makefile
```

Files to Submit

```
array_list.h
array_list_tests.cpp
doubly_linked_list.h
doubly_linked_list_tests.cpp
stack.h
stack_tests.cpp
queue.h
queue_tests.cpp
```

Task 1: Array List

Implement a list using an array.

Requirements

Files

`array_list.h` - contains the template definitions
`array_list_tests.cpp` - contains the test cases and test driver (main)

Class

```
template <typename Object>
class ArrayList;
```

Functions (public)

`ArrayList()` - makes an empty list
`explicit ArrayList(size_t)` - makes a list with the specified initial capacity

```
+--Rule of Three-----------------------------------------------+
| ArrayList(const ArrayList&) - constructs a copy of the given list  |
| ~ArrayList() - destroys this list                                  |
| ArrayList& operator=(const ArrayList&) - assigns a copy of the given list |
+--------------------------------------------------------------+
```

`size_t size() const` - returns the number of elements in the list
`Object& operator[](size_t)` - returns a reference to the element at the specified index or throws `std::out_of_range` if the index is out of bounds.
`void insert(size_t, const Object&)` - insert the given object at the specified index or throws `std::out_of_range` if the index is out of bounds
`void remove(index)` - remove the object at the specified index or throws `std::out_of_range` if the index is out of bounds

Optional

`ArrayList(ArrayList&&)` - move-constructs a “copy” of the given (rvalue) list
`ArrayList& operator=(ArrayList&&)` - move-assigns a “copy” of the given (rvalue) list
`void insert(size_t, Object&&)` - insert the given (rvalue) object at the specified index or throws `std::out_of_range` if the index is out of bounds
`const Object& operator[](size_t) const` - returns a constant reference to the element at the specified index or throws `std::out_of_range` if the index is out of bounds.
`Object* begin()` – returns a pointer to the beginning of the list
`const Object* begin() const` – returns a pointer to the beginning of the list

`Object*  end()` – returns a pointer to the end of the list
`const Object* end() const` – returns a pointer to the end of the list

Example

```
// make an empty list
ArrayList<int> list;

// insert 3 values at the end of the list
list.insert(0, 1);
list.insert(1, 2);
list.insert(2, 3);

// get the size
size_t size = list.size();

// remove the middle element
list.remove(1);

// access the element at index 1
int value = list[1];
```

Task 2: Doubly Linked List

Implement a list using a doubly linked list.

Requirements

Files

`doubly_linked_list.h` - contains the template definitions
`doubly_linked_list_tests.cpp` - contains the test cases and test driver `(main)`

Class

```
template <typename Object>
class DoublyLinkedList;
```

Functions (public)

`DoublyLinkedList()` - makes an empty list

```
+--Rule of Three-------------------------------------------------+
| DoublyLinkedList(const DoublyLinkedList&) - constructs a copy of the given list|
| ~DoublyLinkedList() - destroys this list                                        |
| DoublyLinkedList& operator=(const DoublyLinkedList&) - assigns a copy           |
| the given list                                                                  |
+---------------------------------------------------------------------------------+
```

`size_t size() const` - returns the number of elements in the list
`Object& operator[](size_t)` - returns a reference to the element at the specified index or throws `std::out_of_range` if the index is out of bounds.
`void insert(size_t, const Object&)` - insert the given object at the specified index or throws `std::out_of_range` if the index is out of bounds
`void remove(size_t)` - remove the object at the specified index or throws `std::out_of_range` if the index is out of bounds

Optional

`DoublyLinkedList(DoublyLinkedList&&)` - move-constructs a “copy” of the given (rvalue) list
`DoublyLinkedList& operator=(DoublyLinkedList&&)` - move-assigns a “copy” of the given (rvalue) list
`void insert(size_t, Object&&)` - insert the given (rvalue) object at the specified index or throws `std::out_of_range` if the index is out of bounds
`const Object& operator[](size_t) const` - returns a constant reference to the element at the specified index or throws `std::out_of_range` if the index is out of bounds.

`iterator begin()` – returns an iterator that points to the beginning of the list
`const_iterator begin() const` – returns an iterator that points to the beginning of the list
`iterator end()` – returns an iterator that points to the end of the list
`const_iterator end() const` – returns an iterator that points to the end of the list

Example

```
// make an empty list
DoublyLinkedList<int> list;

// insert 3 values at the end of the list
list.insert(0, 1);
list.insert(1, 2);
list.insert(2, 3);

// get the size
size_t size = list.size();

// remove the middle element
list.remove(1);

// access the element at index 1
int value = list[1];
```

Task 3: Stack

Implement a stack using a list. You should use your `ArrayList` or `DoublyLinkedList`.

Requirements

Files

`stack.h` - contains the template definitions
`stack_tests.cpp` - contains the test cases and test driver (main)

Class

```
template <typename Object>
class Stack;
```

Functions (public)

`Stack()` - makes an empty stack

```
+--Rule of Three-----------------------------------------+
| Stack(const Stack&) - constructs a copy of the given stack |
| ~Stack() - destroys this stack                             |
| Stack& operator=(const Stack&) - assigns a copy of the given stack |
+------------------------------------------------------------+
```

`void push(const Object&)` - add the given object to the top of the stack
`void pop()` - remove the top element from the stack, or throw `std::out_of_range` is the stack is empty.
`Object& top()` - return a reference to the element on top of the stack or throw `std::out_of_range` if the stack is empty.

Optional

`Stack(Stack&&)` - move-constructs a “copy” of the given (rvalue) stack
`Stack& operator=(Stack&&)` - move-assigns a “copy” of the given (rvalue) stack
`void push(Object&&)` - add the given (rvalue) object to the top of the stack
`const Object& top() const` - returns a constant reference to the element on top of the stack or throws `std::out_of_range` if the stack is empty.
`size_t size() const` - returns the number of elements in the stack

Example

```
// make an empty stack
Stack<int> stack;

// push 3 values onto the stack
stack.push(1);
stack.push(2);
stack.push(3);

// remove the top element
stack.pop();

// access the top element
int value = stack.top();
```

Task 4: Queue

Implement a queue using a list. You should use your `ArrayList` or `DoublyLinkedList`.

Requirements

Files

`queue.h - contains the template definitions`
`queue_tests.cpp - contains the test cases and test driver (main)`

Class

```
template <typename Object>
class Queue;
```

Functions (public)

`Queue()` - makes an empty stack

```
+--Rule of Three-------------------------------------------+
| Queue(const Queue&) - constructs a copy of the given queue |
| ~Queue() - destroys this queue                             |
| Queue& operator=(const Queue&) - assigns a copy of the given stack |
+-----------------------------------------------------------+
```

`void enqueue(const Object&)` - add the given object to the back of the queue
`Object dequeue()` - remove and return the front element from the queue, or throw `std::out_of_range` if the queue is empty.
`Object& front()` - return a reference to the element at the front of the queue or throw `std::out_of_range` if the queue is empty.

Optional

`Queue(Queue&&)` - move-constructs a “copy” of the given (rvalue) queue
`Queue& operator=(Queue&&)` - move-assigns a “copy” of the given (rvalue) queue
`void enqueue(Object&&)` - add the given (rvalue) object to the queue
`const Object& front() const` - returns a constant reference to the element at the front of the queue or throws `std::out_of_range` if the queue is empty.
`size_t size() const` - returns the number of elements in the queue

Example

```
// make an empty queue
Queue<int> queue;

// enqueue 3 values into the queue
queue.enqueue(1);
queue.enqueue(2);
queue.enqueue(3);

// remove the front element
queue.dequeue();

// access the front element
int value = queue.front();
```

How To Measure Coverage with Gcov

Compile with coverage

```
g++ -std=c++17 -g --coverage <source files>
```

Run

```
./a.out
```

Generate coverage report

```
gcov -mr <source file>
```

View coverage report

```
cat <source file>.gcov
```

```
‘-’ means the line is not executable (does not count for coverage)
‘#####’ means the line is executable but was executed 0 times
‘126’ means the line was executed 126 times
```

Identify lines which are not covered

```
grep “#####” <source file>.gcov
```

Clean up before next measurement

```
rm -f *.gcov *.gcno *.gcda
```