Programming Assignment 1

C++ Programming Review

Task 1: Selection

Write a program to determine (select) the *k*-th largest value in a list of *N* values.

Requirements

Files	`selection.cpp`
	`selection.h`
	`selection_tests.cpp`

Function	`int select(size_t k, const int* list, size_t N);`
	
	Input
	`k := the rank of the desired value, 0 < k <= N`
	`list := the collection of values from which to select`
	`N := the number of values in the list, N > 0`
	
	Output
	`The value which is the k-th largest in the list.`
	
	Exceptions
	`Throws std::invalid_argument if the arguments are invalid, e.g. k is out of bounds.`

Approved Includes	`cstddef, iostream, stdexcept, selection.h`
Tests	`You must submit a test suite that, when run, covers at least 90% of your code.` See How To Measure Coverage with Gcov

Examples

Consider the list `A = [8, 6, 7, 5, 3, 0, 9]`.

```
select(2, A, 7)  returns   8
select(4, A, 7)  returns   6
select(3, A, 7)  returns   7
select(5, A, 7)  returns   5
select(6, A, 7)  returns   3
select(7, A, 7)  returns   0
select(1, A, 7)  returns   9
select(0, A, 7)  throws    std::invavlid_argument
select(8, A, 7)  throws    std::invavlid_argument
```

Task 2: Collection<T>

Write a class template, `Collection`, that stores an unordered collection of `Objects`[1]..

Requirements

Files	`collection.h` `collection_tests.cpp`
Class	`template <typename Object>` `class Collection;`

Member Functions	`Collection();`
	
	`The default constructor makes an empty Collection.`
	
	`----------`
	
	`Collection(const Collection&);`
	`~Collection();`
	`Collection& operator=(const Collection&);`
	
	`The Rule of Three copies and destroys Collections.`
	
	`----------`
	
	`size_t size() const;`
	
	Input
	None.
	
	Output
	The number of elements in the collection.
	
	Exceptions
	None.
	
	`----------`
	
	`bool is_empty() const;`

[1] `Object` is a generic type which is assumed to have a zero-parameter (default) constructor, an `operator=`, and a copy constructor (and, therefore, also a destructor).

	Input
	None.
	
	Output
	Boolean true if and only if the collection is empty.
	
	Exceptions
	None.
	
	---------- 	
	
	void make_empty();
	
	Input
	None.
	
	Output
	None.
	Side effect: the collection is now empty.
	
	Exceptions
	None.
	
	---------- 	
	
	void insert(const Object& obj);
	
	Input
	obj := the value to insert
	
	Output
	None.
	Side effect: the collection now contains the value of *obj*.
	
	Exceptions
	None.
	
	---------- 	
	
	void remove(const Object& obj);

	Input obj := the value to remove Output None. Side effect: at most one element which has the same value as *obj* is removed. Exceptions None. ---------- bool contains(const Object& obj) const; Input obj := the value to look for Output Boolean true if and only if an *Object* that is equal to *obj* is present in the collection. Exceptions None.
Approved Includes	cstddef, iostream, stdexcept, collection.h
Tests	You must submit a test suite that, when run, covers at least 90% of your code. See How To Measure Coverage with Gcov

Examples

Consider the following sequence of operations with post conditions:

1. Make a new collection of `int`s : `Collection<int> the_collection;`
    a. A variable of type `Collection<int>` exists.
    b. The collection is empty.
2. Insert `8, 6, 7, 5, 3, 0, 9 : the_collection.insert(8); …`
    a. The collection has 7 elements: {`8,6,7,5,3,0, 9`}[2]
    b. The elements are the digits of Jenny's phone number
3. Remove `8 : the_collection.remove(8);`
    a. The collection has 6 elements: {`6, 7, 5, 3, 0, 9`}[2]
    b. The collection is not empty
4. Make empty : `the_collection.make_empty();`
    a. The collection is empty

[2] The order of the elements in the collection does not matter.

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
```

Example

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

$ g++ -std=c++17 -g --coverage selection.cpp selection_tests.cpp

$ ./a.out
passed 56 / 56
ALL TESTS PASSING

$ gcov -mr selection.cpp
File 'selection.cpp'
Lines executed:97.83% of 46
Creating 'selection.cpp.gcov'

$ grep “#####” selection.cpp.gcov
      #####:   42:     throw "whoop"; // this line is not covered

$ cat selection.cpp.gcov
        -:    0:Source:selection.cpp
        -:    0:Graph:selection.gcno
        -:    0:Data:selection.gcda
        -:    0:Runs:1
    ... [snip] ...
      126:   41:if (k == 2007) {
    #####:   42:      throw "whoop"; // this line is not covered
        -:   43:}
    ... [snip] ...
```