Exceptions

Lab Assignment 1: Step counter

A pedometer treats walking 2,000 steps as walking 1 mile. Write a StepsToMiles() function that takes the number of steps as an integer parameter and returns the miles walked as a double. The StepsToMiles() function throws a runtime_error object with the message "Exception: Negative step count entered." when the number of steps is negative. Complete the main() function that reads the number of steps from a user, calls the StepsToMiles() function, and outputs the returned value from the StepsToMiles() function. Use a try-catch block to catch any runtime_error object thrown by the StepsToMiles() function and output the exception message.

Output each floating-point value with two digits after the decimal point, which can be achieved by executing **cout << fixed << setprecision(2)**; once before all other cout statements.

Ex: If the input of the program is:

In [ ]:
```
5345
```

the output of the program is:

In [ ]:
```
2.67
```

Ex: If the input of the program is:

In [ ]:
```
-3850
```

the output of the program is:

In [ ]:
```
Exception: Negative step count entered.
```

main.cpp

```
#include <iostream>
#include <iomanip>
#include <stdexcept>
using namespace std;

/* Define your function here */

int main() {

   /* Type your code here. */

   return 0;
}
```

Assignment 1 Tests:

Apply the following 2 tests for 4 points

1. Compare output (2 points)

When input is

```
5345
```

Standard output exactly matches

```
2.67
```

2. Compare output (2 points)

When input is

```
-3850
```

Standard output exactly matches

```
Exception: Negative step count entered.
```

Templates

Lab Assignment 2: Ordered lists

An OrderedList is a vector that keeps elements in sorted order.

Complete **template <typename TheType> class OrderedList** by defining the following functions:

1. **int Size()**
   - Return the size of the list
2. **TheType At(int index)**
   - Return the element of the list at parameter index.
3. **int Find(TheType value)**
   - Return the index of the first element in the list equal to parameter value.
   - Return -1 if parameter value is not found in the list.
4. **bool Remove(TheType value)**
   - Search the list for parameter value. Hint: Use Find().
   - If parameter value is found in the list, remove the element found by moving the subsequent elements towards the beginning of the list. Decrement list size and return true.
   - Return false if parameter value is not found in the list.

Hint: Use any vector functions to simplify the implementations.

The template code provides the implementations of the following functions:

1. **void Insert(TheType value)**
   - Search the list for an element that is greater than parameter value.
   - If an element is found, increment list size and move the element and all subsequent elements towards the end of the list to make room for parameter value. Copy parameter value at the location that was occupied by the first element greater than parameter value.
   - If no such element is found, increment list size and add parameter value at the end of the list.
2. **void Print()**
   - Output the list, separated by a space character.

A main program is provided as a sample test in the develop mode. Unit tests will be used during a submission.

Ex: if the given main() is executed, the output of the program is

In [ ]:
```
Size is correct
List is in correct order: 3 7 11
Index of 11 is correct -- 2
7 was removed correctly
```

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

/* ---------- Template class OrderedList declaration ---------- */
template<typename TheType> class OrderedList {
  public:
    int Size();                         // Number of elements in the list
    TheType At(int index);              // Return the element at index

    int Find(TheType value);            // Return index of first occurrence
                                        // of value or -1 if not found
    void Insert(TheType value);         // Insert value at its sorted index
    bool Remove(TheType value);         // Find the first occurrence of value
                                        // and remove the value; true if success
    void Print();

  private:
    vector<TheType> list;               // Elements are stored in list
};

/* ---------- OrderedList function implementations ---------- */

template<typename TheType>
int OrderedList<TheType>::Size() {
  /* Type your code here. */

}

template<typename TheType>
TheType OrderedList<TheType>::At(int index) {
  /* Type your code here. */

}

template<typename TheType>
int OrderedList<TheType>::Find(TheType value) {
  /* Type your code here. */

}

template<typename TheType>
void OrderedList<TheType>::Insert(TheType newItem) {
  unsigned int j;                       // Vector size is unsigned int
  unsigned int k;                       // Vector size is unsigned int
  if (list.size() == 0) {
    list.push_back(newItem);
    return;
  }

  j = 0;
  while (j < list.size() && newItem > list.at(j)) {
    ++j;
  }

  list.resize(list.size() + 1);

  // Now all items after index j are >= newItem
  if (j == list.size()) {
    // If newItem is > last element, just add at end of list
    list.push_back(newItem);
  } else {
    // Now move backwards from the end of the list copying elements to
    // the next higher position; stop at j, where newItem will go
    for (k = list.size() - 1; k > j; --k) {
      list.at(k) = list.at(k-1);
    }

    // Finally, insert newItem
    list.at(k) = newItem;
  }
}

template<typename TheType>
bool OrderedList<TheType>::Remove(TheType oldItem) {
  unsigned int j;
  int indx = Find(oldItem);

  /* Type your code here. */

}

template<typename TheType>
void OrderedList<TheType>::Print() {
  for (int j = 0; j < Size(); ++j) {
    cout << list.at(j);
    if (j < Size()) {
      cout << " ";                              // No end line after last element
    }
  }
}

/* --------- End OrderedList function implementations ---------- */

// Demonstration of functions
int main() {
  OrderedList<int> intList;

  intList.Insert(11);
  intList.Insert(3);
  intList.Insert(7);

  if (intList.Size() == 3) {
    cout << "Size is correct" << endl;
  }
  else {
    cout << "Size should be 3" << endl;
  }

  if (intList.At(0) == 3 && intList.At(1) == 7 && intList.At(2) == 11) {
    cout << "List is in correct order: ";
  }
  else {
    cout << "List is out of order: ";
  }
  intList.Print();
  cout << endl;

  int indx = intList.Find(11);
  if (indx == 2) {
    cout << "Index of 11 is correct -- " << indx << endl;
  }
  else {
    cout << "Index of 11 is incorrect -- " << indx << endl;
  }

  intList.Remove(7);
  if (intList.At(0) == 3 && intList.At(1) == 11) {
    cout << "7 was removed correctly" << endl;
  }
  else {
    cout << "7 was not removed correctly" << endl;
  }
}
```

Assignment 2 Tests:

Apply the following 5 tests for 10 points

1. Unit test (2 points)

Test insert 3 integers. OrderedList.Size() should return 3.

2. Unit test (2 points)

Test insert 3 integers. Result list should be sorted: [3 7 11]

3. Unit test (2 points)

Test Find(11.1) with [3.3 7.711.1 13.3]. Should return 2.

4. Unit test (2 points)

Test Find(9.9) with [1.1 3.3 5.5 7.7]. Should return -1.

5. Unit test (2 points)

Test At(1) with ["apple" "banana" "mango" "watermelon"]. Should returns "banana".

Submissions

Note: Do not forget to submit the TWO assignments and their corresponding test outputs to receive full credit.

1. Name your C++ files FirstName_Lastname_Ordered_List.cpp and FirstName_Lastname_Step_Counter.cpp.
2. Prepare your report in docx or pdf format and name it Firstname_Lastname.docx or Firstname_Lastname.pdf.
3. Add the screenshot of your codes to the report and provide a description for them. All tests should be performed and the result screenshot be included in the report.