San Francisco Bay University

CS360L - Programming in C and C++ Lab
Lab Assignment #5

Due day: 4/5/2024

Replit link: https://replit.com/@TASMITA-TANJIMT/CSLAB5

Github link: https://github.com/tasmita0131/CSLAB5

1. Write a function that takes a vector of integers as argument and reverses its elements.

```
void rvrs(Vector<int>& vct){
      //Complete your program
}
```

CODE:

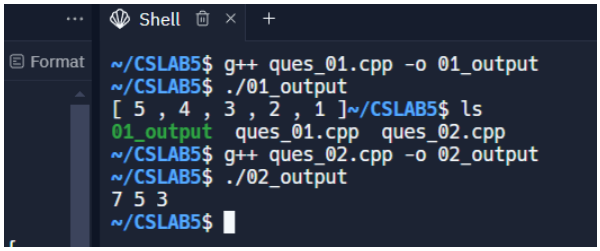
```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

void rvrs(vector<int>& vct) {
   reverse(vct.begin(), vct.end());
}

int main() {
   vector<int> myVector = {1, 2, 3, 4, 5}; // Example vector
   rvrs(myVector); // Reverses the vector

   // Printing the reversed vector
   cout << "[ ";
   for (size_t i = 0; i < myVector.size(); ++i) {
      cout << myVector[i];
      if (i != myVector.size() - 1) {
         cout << " , ";
      }
   }
   cout << " ]";

   return 0;
}
```

```
~/CSLAB5$ g++ ques_01.cpp -o 01_output
~/CSLAB5$ ./01_output
[ 5 , 4 , 3 , 2 , 1 ]~/CSLAB5$
```

2. **Find a function with one argument, vector of vectors named *vals*, for coordinates of one of its elements in *row* and *col* to print the values that lie on the lower-left to upper-right diagonal of *vals*. After that, verify it in *main* function.**

CODE:

```
#include <iostream>
#include <vector>

using namespace std;

void printDiagonal(const vector<vector<int>>& vals, int row, int col) {
    while (row >= 0 && col < vals[row].size()) {
        cout << vals[row][col] << " ";
        row--; // Move up a row
        col++; // Move right a column
    }
    cout << endl;
}

int main() {
    vector<vector<int>> vals = {
        {1, 2, 3},
        {4, 5, 6},
        {7, 8, 9}
    };

    // Coordinates of the element
    int row = vals.size() - 1; // Start at the last row
    int col = 0;               // Start at the first column

    // Call the function to print the diagonal starting from (row, col)
    printDiagonal(vals, row, col);

    return 0;
}
```



3. **Create a class *Tensor* with a method *sort* to sort a vector input argument and print it out. Please verify this correctness in *main* function**

CODE:

```
#include <algorithm>
#include <iostream>
#include <vector>

using namespace std;

class Tensor {
public:
  void sort(vector<int> &vec) {
    std::sort(vec.begin(), vec.end());
    for (int i : vec) {
      cout << i << " ";
    }
    cout << endl;
  }
};

int main() {
  Tensor tensor;
  vector<int> myVector = {47, 20, 32, 11, 50};

  cout << "Original vector: ";
  for (int i : myVector) {
    cout << i << " ";
  }
  cout << endl;

  cout << "Sorted vector: ";
  tensor.sort(myVector); // The sort method will print the sorted vector

  return 0;
}
```

```
Shell
~/CSLAB5$ g++ ques_03.cpp -o 03_output
~/CSLAB5$ ./03_output
Original vector: 47 20 32 11 50
Sorted vector: 11 20 32 47 50
~/CSLAB5$
```

4. Find the errors in the following class and explain how to correct them. Please test it in main function

```
class Example{
  public:
    Example( int y = 10 ): data( y ){
      // empty body
    } // end Example constructor
    int getIncrementedData() const{
      return data++;
    } // end function getIncrementedData
    static int getCount(){
      cout << "Data is " << data << endl;
      return count;
    } // end function getCount
  private:
    int data;
    static int count;
}; // end class Example
```

CORRECTED VERSION OF THE CODE:

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

class Example {
public:
   // Constructor: If you don't specify a value, y defaults to 10.
   Example(int y = 10): data(y) {
      // Every time we create an Example, let's up the count by one.
      count++;
   }

   // Instead of incrementing data when we just want to see it,
   // let's have a simple method to get the data's current value.
   int getData() const {
      return data;
   }
```

```
    // Need to increment data? This method bumps it up by one and gives it back to you.
    int incrementData() {
        return ++data;
    }

    // Wanna know how many Example objects we've got? This will tell you and print the count.
    static int getCount() {
        cout << "Count is " << count << endl;
        return count;
    }

private:
    int data; // Each Example knows its own data.
    static int count; // But all Examples share the same count.
};

// Don't forget to actually give count a starting value!
int Example::count = 0;

int main() {
    // Let's create a couple of Example objects to see this in action.
    Example ex1, ex2(20);

    // What's the initial data for each object?
    cout << "ex1 data: " << ex1.getData() << endl;
    cout << "ex2 data: " << ex2.getData() << endl;

    // Now let's increment the data and see what we get.
    cout << "ex1 incremented data: " << ex1.incrementData() << endl;
    cout << "ex2 incremented data: " << ex2.incrementData() << endl;

    // And how many Example objects have we made? This should say 2.
    Example::getCount();

    return 0;
} #include <iostream>
using namespace std;

class Example {
public:
    // Constructor: If you don't specify a value, y defaults to 10.
    Example(int y = 10): data(y) {
        // Every time we create an Example, let's up the count by one.
        count++;
    }

    // Instead of incrementing data when we just want to see it,
    // let's have a simple method to get the data's current value.
    int getData() const {
        return data;
    }
```

```
    // Need to increment data? This method bumps it up by one and gives it back to you.
    int incrementData() {
        return ++data;
    }

    // Wanna know how many Example objects we've got? This will tell you and print the count.
    static int getCount() {
        cout << "Count is " << count << endl;
        return count;
    }

private:
    int data; // Each Example knows its own data.
    static int count; // But all Examples share the same count.
};

// Don't forget to actually give count a starting value!
int Example::count = 0;

int main() {
    // Let's create a couple of Example objects to see this in action.
    Example ex1, ex2(20);

    // What's the initial data for each object?
    cout << "ex1 data: " << ex1.getData() << endl;
    cout << "ex2 data: " << ex2.getData() << endl;

    // Now let's increment the data and see what we get.
    cout << "ex1 incremented data: " << ex1.incrementData() << endl;
    cout << "ex2 incremented data: " << ex2.incrementData() << endl;

    // And how many Example objects have we made? This should say 2.
    Example::getCount();

    return 0;
}
```

```
~/CSLAB5$ g++ ques_04.cpp -o 04_output
~/CSLAB5$ ./04_output
ex1 data: 10
ex2 data: 20
ex1 incremented data: 11
ex2 incremented data: 21
Count is 2
```