EE538 HW2 Solution

University of Southern California

Instructor: Arash Saifhashemi

Q1

```
#include "q.h"

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

std::string CPPLib::GetFullName(std::string& first_name,
                                std::string& last_name) {
  std::string result;
  if (first_name.empty() && last_name.empty()) {
    return result;
  }

  if (first_name.empty()) {
    return last_name;
  }

  if (last_name.empty()) {
    return first_name;
  }

  result = first_name + std::string(" ") + last_name;

  return result;
}

int CPPLib::CountCharacters(std::string& input, std::vector<char>
characters) {
  int result = 0;
  std::vector<char> characters_lower_case(characters.size());

  std::transform(characters.begin(), characters.end(),
                 characters_lower_case.begin(),
                 [](unsigned char c) { return std::tolower(c); });

  for (auto c : input) {
    if (std::find(characters_lower_case.begin(),
characters_lower_case.end(),
                  std::tolower(c)) != characters_lower_case.end()) {
      result++;
    }
  }
```

```
  return result;
}
```

Q2

```
#include "q.h"

#include <cctype>
#include <iostream>
#include <string>
#include <vector>
// Returns the index of the fist space character in a string.
int CPPLib::IndexOfFirstSpace(std::string& input) {
  for (size_t i = 0; i < input.size(); i++) {
    if (input[i] == ' ') {
      return i;
    }
  }
  return -1;
}

// Given the full_name, it returns the first_name and last_name.
// Note: you should use IndexOfFirstSpace function and cannot use any
other
// std:: functions.
void CPPLib::SeparateFirstAndLastNames(std::string& full_name,
                                       std::string& first_name,
                                       std::string& last_name) {
  auto index_of_first_space = IndexOfFirstSpace(full_name);

  if (index_of_first_space != -1) {
    first_name = full_name.substr(0, index_of_first_space);
    last_name = full_name.substr(index_of_first_space + 1);
  } else {
    first_name = full_name;
  }
}

// Returns the number of vowels (a, e, i, o, u) in a string.
// Your algorithm should be case insensitive.
int CPPLib::NumberOfVowels(std::string& input) {
  std::vector<char> vowels = {'a', 'e', 'i', 'o', 'u'};
  int count = 0;
  for (auto e : input) {
    auto l = std::tolower(e);
    if ((l >= 'a' && l <= 'z') &&
        std::find(vowels.begin(), vowels.end(), l) != vowels.end()) {
      count++;
    }
  }
```

```
  return count;
}

// Returns the number of consonants (letters that are not a, e, i, o, u)
in a
// string.
// Returns the number of vowels (a, e, i, o, u) in a string.
// Your algorithm should be case insensitive.
int CPPLib::NumberOfConsonants(std::string& input) {
  std::vector<char> vowels = {'a', 'e', 'i', 'o', 'u'};
  int count = 0;
  for (auto e : input) {
    auto l = std::tolower(e);
    if ((l >= 'a' && l <= 'z') &&
        std::find(vowels.begin(), vowels.end(), l) == vowels.end()) {
      count++;
    }
  }
  return count;
}

// Returns the revers of a string.
// Example input: 'ted', output: 'det'.
// Note: You cannot use any std:: functions.
int CPPLib::Reverse(std::string& input) {
  for (size_t i = 0; i < input.size() / 2; i++) {
    auto temp = input[i];
    input[i] = input[input.size() - 1 - i];
    input[input.size() - 1 - i] = temp;
  }

  return 0;
}
```

Q3

q.h

```
enum class Operation {
  kAdd,
  kSubtract,
  kDivision,
  kMultiplication,
  kBitwise_AND,
  kBitwise_OR,
  kBitwise_XOR,
  kShift_right,
  kShift_left
};

class CPPLib {
```

```
 public:
  // Given two integers, returns the result of the operation based on the
given
  // operator.
  float Calculate(int a, int b, Operation operation);
};
```

q.cc

```
#include "q.h"

// Given two integers, returns the result of the operation based on the
given
// operator.
float CPPLib::Calculate(int a, int b, Operation operation) {
  switch (operation) {
    case Operation::kAdd:
      return a + b;
      break;
    case Operation::kSubtract:
      return a - b;
      break;
    case Operation::kDivision:
      if (b != 0) {
        return a / b;
      } else {
        return -1;
      }
      break;
    case Operation::kMultiplication:
      return a * b;
      break;
    case Operation::kBitwise_AND:
      return a & b;
      break;
    case Operation::kBitwise_OR:
      return a | b;
      break;
    case Operation::kBitwise_XOR:
      return a ^ b;
      break;
    case Operation::kShift_right:
      return a >> b;
      break;
    case Operation::kShift_left:
      return a << b;
      break;
    default:
      return -1;
  }
}
```

Q4

```cpp
#include "q.h"

#include <string>

// A function that capitalize the first letter of a string.
// If it was possible, returns true, otherwise false.
bool CPPLib::CapitalizeFirstLetter(std::string &input) {
  if (input.empty()) {
    return false;
  }

  if (input[0] >= 'a' && input[0] <= 'z') {
    input[0] = input[0] - ('a' - 'A');
    return true;
  }

  if (input[0] >= 'A' && input[0] <= 'Z') {
    return true;
  }

  return false;
}
```

Q5

```cpp
#include "q.h"

#include <iostream>
#include <vector>

// Concatenate two dynamic arrays.
// Example:
// array_1 = {1, 2}, size_1 = 2
// array_2 = {2, 3, 4}, size_12 = 3
// Output: {1, 2, 2, 3, 4}.
// Question 1: Why did we have to proved size_1, size_2 as an input?
// Question 2: How can we know the size of the output?
int* CPPLib::Concatenate(int* array_1, int size_1, int* array_2, int
size_2) {
  int size = size_1 + size_2;
  int* result = new int[size];

  for (int i = 0; i < size_1; i++) {
    result[i] = array_1[i];
  }

  for (int i = 0; i < size_2; i++) {
```

```
    result[i + size_1] = array_2[i];
  }

  return result;
}
//------------------------------------------------------------------------
-----
// Concatenate two dynamic vectors.
// Example:
// vector_1 = {1, 2}
// vector_2 = {2, 3, 4}
// Output: {1, 2, 2, 3, 4}.
// Question 1: Why didn't we provide the sizes?
// Question 2: We have two functions with the name of Concatenate. Is this
ok?
std::vector<int> CPPLib::Concatenate(std::vector<int>& vector_1,
                                     std::vector<int>& vector_2) {
  std::vector<int> result;
  for (auto e : vector_1) {
    result.push_back(e);
  }

  for (auto e : vector_2) {
    result.push_back(e);
  }
  return result;
}
//------------------------------------------------------------------------
-----
```

Q6

Q1: We should not index an item in an empty vector.

Q2: We should not dereference a pointer that is not allocated using new.

Q3: Delete pointers a, b at the end.

Q4: We increment the pointers and then dereference them. Also, we need to delete a, b.

Q5: The loop never ends

code

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

// Returns element at i index.
int ReturnElementI(std::vector<int>& input, int i) { return input[i]; }

// - What is wrong with each piece of code below?
```

```
// - For each case modify the code so that the issue is fixed:
int main() {
  // Question 1:
  {
    std::vector<int> elements;

    // Number of values to read from the input
    int number_of_items;
    std::cin >> number_of_items;

    // Reading elements from the input.
    for (int i = 0; i < number_of_items; i++) {
      int element;
      std::cin >> element;
      elements.push_back(element);
    }
  }

  // Question 2:
  {
    int* a = new int;
    std::cin >> (*a);
    (*a)++;
    std::cout << "(*a): " << (*a) << std::endl;
  }

  // Question 3:
  {
    int* a = new int;
    int* b = new int;
    std::cin >> (*a);
    std::cin >> (*b);

    std::cout << "(*a) + (*b): " << (*a) + (*b) << std::endl;

    delete a;
    delete b;
  }

  // Question 4:
  {
    int* a = new int;
    int* b = new int;
    std::cin >> (*a);
    std::cin >> (*b);
    std::cout << "(*a) + (*b): " << (*a) + (*b) << std::endl;
    delete a;
    delete b;

    a = new int;
    b = new int;
    std::cin >> (*a);
    std::cin >> (*b);
    std::cout << "(*a) + (*b): " << (*a) + (*b) << std::endl;
```

```
  }

  // Question 5:
  {
    for (int i = 0; i < 10; i++) {
      std::cout << "i: " << i << std::endl;

      std::cout << "i - 1: " << i - 1 << std::endl;
    }
  }

  return 0;
}
```

Q7

```
#include "q.h"

// Write two functions for swapping variables:

// Example :
// Before: x = 10, y = 15
// We call Swap(x,y)
// After: x = 15, y = 10

void CPPLib::SwapByPointer(float *input1, float *input2) {
  if (input1 == nullptr || input2 == nullptr) {
    return;
  }

  float tmp;
  tmp = *input1;
  *input1 = *input2;
  *input2 = tmp;
}

void CPPLib::SwapByReference(float &input1, float &input2) {
  float tmp;
  tmp = input1;
  input1 = input2;
  input2 = tmp;
}
```

Q8

```
#include "q.h"

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

```cpp
#include <set>
#include <vector>

// Write a function that takes a vector of positive integers as input. The
// output is the same vector where all duplicates are removed. Note that
the
// output is the same vector means the function's return type should be
void and
// do the modifications on the input vector. Example: before: v=[1, 2, 2,
4],
// after : v=[1, 2, 4] Solve this for the following cases: You cannot use
// std::set
void CPPLib::UniqueVectorNotBySet(std::vector<int> &input) {
  std::vector<int> result;
  for (auto &e : input) {
    if (std::find(result.begin(), result.end(), e) == result.end()) {
      result.push_back(e);
    }
  }
  input = result;
}
// You can use std::set
void CPPLib::UniqueVectorBySet(std::vector<int> &input) {
  std::set<int> my_set(input.begin(), input.end());

  input.clear();
  for (auto &e : my_set) {
    input.push_back(e);
  }
}

// Write a function that takes two vectors v1 and v2 and returns a new
vector
// that is the intersection of the values in v1 and v2. All the values in
return
// vector should be unique.
// Example: input: v1={1, 2, 2, 3}, v2={3, 4, 4, 5},
// output = {1, 2, 3, 4, 5}
std::vector<int> CPPLib::IntersectVectors(std::vector<int> &input1,
                                          std::vector<int> &input2) {
  std::vector<int> result;
  std::vector<int> v1_unique = input1;
  std::vector<int> v2_unique = input2;

  UniqueVectorNotBySet(v1_unique);
  UniqueVectorNotBySet(v2_unique);

  for (auto &e : v1_unique) {
    if (std::find(v2_unique.begin(), v2_unique.end(), e) !=
v2_unique.end()) {
      result.push_back(e);
    }
  }
```

```
    return result;
  }
```