EE538: Computing Principles for Electrical Engineers

Discussion 4: Solve C++ Problems

University of Southern California
06/08,06/09 Summer 2022
Instructor: Arash Saifhashemi
TA and Mentors: Yijun Liu, Aditi Bodhankar, Zixu Wang

Outline

- String
- Vector
- Set / Unordered Set
- HW2 Hints

std::string

- Strings are objects that represent sequences of characters.
- Use string instead of char array!

```
std::string name;
std::cout << "What is your name? ";
getline (std::cin, name);
std::cout << "Hello, " << name << "!\n";
```

https://www.cplusplus.com/reference/string/string/

std::string

- Methods
  - Operator[ ]
  - Operator+
  - push_back()
  - pop_back()
  - insert()
  - erase()
  - c_str()
  - size()

```
std::string str1 = "USC";
std::string str2 = "EE538";
std::cout << str1[0] << std::endl;
str1 = str1 + str2;
std::cout << str1 << std::endl;
str1.push_back('x');
std::cout << str1 << std::endl;
str1.pop_back();
std::cout << str1 << std::endl;
std::cout << str1.size() << std::endl;
const char* cstr = str1.c_str();
std::cout << cstr << std::endl;
```

std::string

- Methods
  - Operator[ ]
  - Operator+
  - push_back()
  - pop_back()
  - insert()
  - erase()
  - c_str()
  - size()

```
 std::string str="to be question";
  std::string str2="the ";
  std::string str3="or not to be";
  std::string::iterator it;
  // used in the same order as described above:
  str.insert(6,str2);                 // to be (the )question
  str.insert(6,str3,3,4);             // to be (not )the question
  str.insert(10,"that is cool",8);    // to be not (that is )the question
  str.insert(10,"to be ");            // to be not (to be )that is the question
  str.insert(15,1,':');               // to be not to be(:) that is the question
  it = str.insert(str.begin()+5,','); // to be(,) not to be: that is the question
  str.insert (str.end(),3,'.');       // to be, not to be: that is the question(...)
  str.insert (it+2,str3.begin(),str3.begin()+3); // to be, (or )not to be: that is
the question...

  std::cout << str << '\n';
```

std::string

- Methods
  - Operator[ ]
  - Operator+
  - push_back()
  - pop_back()
  - insert()
  - erase()
  - c_str()
  - size()

```
std::string str ("This is an example sentence.");
 std::cout << str << '\n';
                                  // "This is an example sentence."
 str.erase (10,8);                       //            ^^^^^^^^
 std::cout << str << '\n';
                                  // "This is an sentence."
 str.erase (str.begin()+9);              //         ^
 std::cout << str << '\n';
                                  // "This is a sentence."
 str.erase (str.begin()+5, str.end()-9);  //      ^^^^^
 std::cout << str << '\n';
                                  // "This sentence."
```

std::string

- Methods
  - Operator[ ] O(1)
  - Operator+ O(n)
  - push_back() O(1)
  - pop_back() O(1)
  - insert() O(n)
  - erase() O(n)
  - c_str() O(1)
  - size() O(1)

std::vector

- Methods
  - Operator[ ]
  - push_back()
  - pop_back()
  - insert()
  - erase()
  - size()

Guess the complexity of each function! Note that vector is similar to a dynamic array!

std::vector

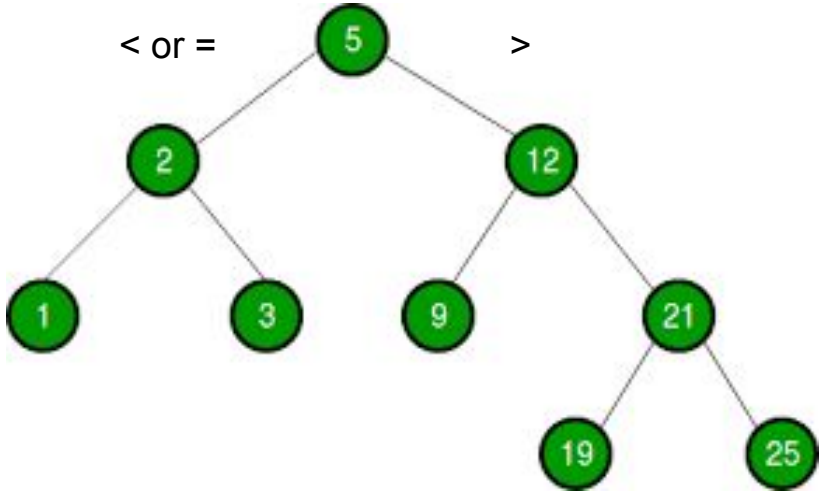
- Methods
  - Operator[ ]: O(1) Constant
  - push_back(): O(1) Constant (amortized time, reallocation may happen).
  - pop_back(): O(1) Constant
  - insert(): O(m+n) Linear on the number of elements (*m*) inserted (copy/move construction) plus the number of elements (*n*) after position (moving).
  - erase(): O(m+n) Linear on the number of elements (*m*) erased (destructions) plus the number of elements (*n*) after the last element deleted (moving).
  - size(): O(1) Constant

https://www.cplusplus.com/reference/vector/vector/ for vector

https://www.youtube.com/watch?v=d605guaOH3A for amortized analysis (not covered in exams)

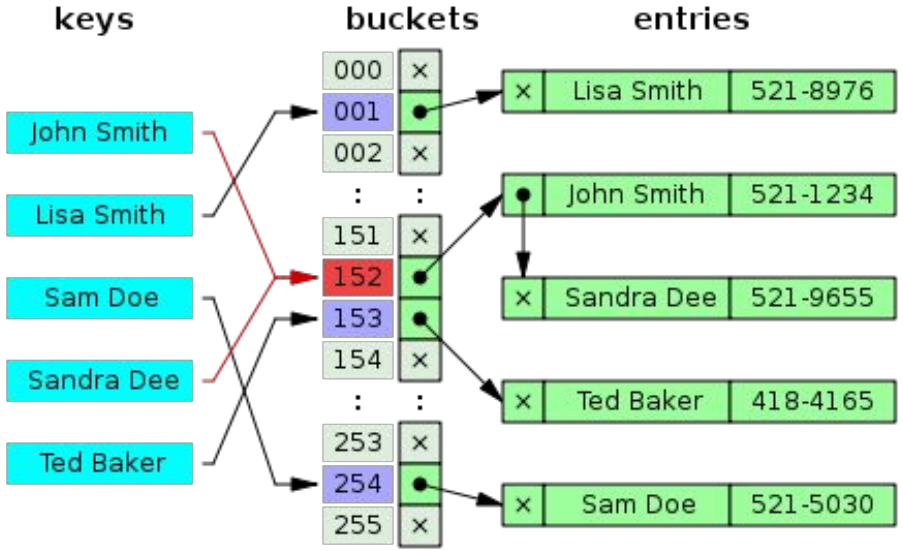
std::set

- Methods
  - insert() O(logn)
  - erase() O(logn)
  - size() O(1)
  - find() O(logn)



http://www.cplusplus.com/reference/set/set/

std::unordered_set



Hash collision!

- Methods
  - insert() Average case: constant. Worst case: linear in container size.
  - erase() Average case: constant. Worst case: linear in container size.
  - size() O(1) Constant
  - find() Average case: constant. Worst case: linear in container size.

https://www.cplusplus.com/reference/unordered_set/unordered_set/

Outline

- String
- Vector
- Set / Unordered Set
- HW2 Hints

HW2 Q1

```
class CPPLib {
 public:
  // Given two separate strings for first and last names, return the full name.
  std::string GetFullName(std::string& first_name, std::string& last_name);

  // Given a string and a list of characters, find how many times any of those
  // characters appeared in the input string. Make the comparison case
  // insensitive. Example: Input: "This is a test", characters = {'t','h'}
  // Output: 4

  // YOU CANNOT USE ANY LIBRARY FUNCTIONS FROM std::
  int CountCharacters(std::string& input, std::vector<char> characters);
};
```

HW2 Q1

Get Full name:

std::string input1;
std::string input2;

Hint: result = input1 + input2;

int CPPLib::CountCharacters(std::string& input, std::vector<char> characters)

int char_count;
for each ch in characters:
Char_count = std::count(input.begin(), input.end(), ch)

How to run code:

```
bazel run files/1:q
```

No need to test

HW2 Q2

```
class CPPLib {
 public:
  // Returns the index of the fist space character in a string.
  int IndexOfFirstSpace(std::string& input);

  // 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 SeparateFirstAndLastNames(std::string& full_name,
                                 std::string& first_name,
                                 std::string& last_name);

  // Returns the number of vowels (a, e, i, o, u) in a string.
  // Your algorithm should be case insensitive.
  int NumberOfVowels(std::string& input);

  // Returns the number of consonants (letters that are not a, e, i, o, u) in a
  // string.
  // Your algorithm should be case insensitive.
  int NumberOfConsonants(std::string& input);

  // Returns the revers of a string.
  // Example input: 'ted', output: 'det'.
  // Note: You cannot use any std:: functions.
  int Reverse(std::string& input);
```

HW2 Q2

// Returns the index of the fist space character in a string.

int CPPLib::IndexOfFirstSpace(std::string& input){

check the input.size();

for(i = 0 -> input.size()){

If i == “ ”;

return i;

}

How to test the code?

`bazel test files/2:student_test`

HW2 Q2

// 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)

Check the coner case

Use the *IndexOfFirstSpace()* function to separate the name.

HW2 Q2

// Returns the number of vowles (a, e, i, o, u) in a string.

int CPPLib::NumberOfVowels(std::string& input)

```
std::string dic = “aeiou”;
Int res = 0;
for(auto i : input)
    traverse the dic:
        if i in dic: res++
        else continue;
```

HW2 Q2

// Returns the number of consonants (letters that are not a, e, i, o, u) in a

// string.

int CPPLib::NumberOfConsonants(std::string& input)

Use the NumberOfVowels() function to calculate the vowels.
Then calculate the consonants.

How to test the code?

```
bazel test files/2:student_test
```

HW2 Q2

// Returns the reverse of a string.

// Example input: 'ted', output: 'det'.

// Note: You cannot use any std:: functions.

int CPPLib::Reverse(std::string& input)

```
    for(i = input.size(); i > 0; i-1)
        res.push_back(input[i]);
    input = res;
```

How to test the code?

```
bazel test files/2:student_test
```

HW2 Q3

```
// Define a new C++ enum type called Operation that has members for add,
// subtract, division,  subtraction, bitwise AND, bitwise OR, bitwise XOR, shift
// right, and shift left:
enum class Operation {

};

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);
};
```

HW2 Q3

```
// Define a new C++ enum type called Operation that has members for add,
// subtract, division, subtraction multiplication, bitwise AND, bitwise OR, bitwise XOR,
shift // right, and shift left (please use the following names: subtract, division,
// multiplication, bitwise_AND, bitwise_OR, bitwise_XOR, shift_right, shift_left):
enum class Operation {

};

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

How to test the code?

```
bazel test files/3:student_test
```

HW2 Q3

Enum Class

- Scoped enumeration
  - Strongly typed
  - Strongly scoped
- Use **enum class** instead of just **enum**!

```
// Enum type in C:
enum ColorPallet1 { Red, Green, Blue };
enum ColorPallet2 { Yellow, Orange, Red };

// Enum Class in C++
// Declaration
enum class ColorPalletClass1 { Red, Green, Blue };
enum class ColorPalletClass2 { Yellow, Orange, Red };

// Assignment
ColorPalletClass1 col1 = ColorPalletClass1::Red;
ColorPalletClass2 col2 = ColorPalletClass2::Red;
```

HW2 Q4

```
class CPPLib {
    public:

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

HW2 Q4

// A function that capitalize the first letter of a string.

// If it was possible, returns true, otherwise false.

bool CPPLib::CapitalizeFirstLetter(std::string &input)

1. Make all characters lower case - std::tolower()
2. Then capitalize the first letter - std::toupper()

HW2 Q4: use character handling functions

Character classification functions

isalnum	Check if character is alphanumeric (function)
isalpha	Check if character is alphabetic (function)
isblank C++11	Check if character is blank (function)
iscntrl	Check if character is a control character (function)
isdigit	Check if character is decimal digit (function)
isgraph	Check if character has graphical representation (function)
islower	Check if character is lowercase letter (function)
isprint	Check if character is printable (function)
ispunct	Check if character is a punctuation character (function)
isspace	Check if character is a white-space (function)
isupper	Check if character is uppercase letter (function)
isxdigit	Check if character is hexadecimal digit (function)

Character conversion function

tolower	Convert uppercase letter to lowercase (function)
toupper	Convert lowercase letter to uppercase (function)

HW2 Q4

CapitalizeFirstLetter

```
If the length of the input is 0 or the first letter is not an alphabet
     Return false
Else
     input[0] = toupper(input[0])
     Return true
```

How to test the code?

```
bazel test files/4:student_test
```

HW2 Q5

```
// 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* Concatenate(int* array_1, int size_1, int* array_2, int size_2);
//------------------------------------------------------------------------
// 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> Concatenate(std::vector<int>& vector_1,
                             std::vector<int>& vector_2);
```

HW2 Q5

int* CPPLib::Concatenate(int* array_1, int size_1, int* array_2, int size_2)

```
Create a new dynamic array with size size_1+size_2

Assign values in array_1 to the new array
Assign values in array_2 to the new array
```

std::vector<int> CPPLib::Concatenate(std::vector<int>& vector_1, std::vector<int>& vector_2) {

```
Create a new vector
Push back values in array_1 to the new vector
Push back values in array_2 to the new vector
```

HW2 Q6

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

// - For each case modify the code so that the issue is fixed:

// Returns element at i index.

```
int ReturnElementI(std::vector<int>& input, int i) { return input[i]; }
```

HW2 Q6

```
// Question 1:
 {
   std::vector<int> elements;
   int number_of_items;
   // Number of values to read from the input
   std::cin >> number_of_items;
   // Reading elements from the input.
   for (int i = 0; i < number_of_items; i++) {
     int element;
     std::cin >> elements[i];
   }
 }
```

HW2 Q6

```
// - What is wrong with each piece of code below?
// - For each case modify the code so that the issue is fixed:
// Question 2:
 {
   int* a;
   std::cin >> (*a);
   (*a)++;
   std::cout << "(*a): " << (*a) << std::endl;
 }
```

HW2 Q6

```
// - What is wrong with each piece of code below?
// - For each case modify the code so that the issue is fixed:
// Question 3:
 {
   int* a = new int;
   int* b = new int;
   std::cin >> (*a);
   std::cin >> (*b);

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

HW2 Q6

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

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

HW2 Q6

```
// Question 5:
  {
    for (int i = 0; i < 10; i++) {
      std::cout << "i: " << i << std::endl;
      i = i - 1;
      std::cout << "i: " << i << std::endl;
    }
  }
  return 0;
}
```

HW2 Q7

```
class CPPLib {
 public:
  // Write two functions for swapping variables:

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

  void SwapByPointer(float *input1, float *input2);

  void SwapByReference(float &input1, float &input2);
};
```

HW2 Q7

See Discussion 4: page 34, 35, 36

How to test the code?

```
bazel test files/7:student_test
```

Practice

Write 2 functions that will swap the values of the inputs by pointers and references.

- pass by pointers

```
void SwapByPointer(double *input1, double *input2);
```

- pass by references

```
void SwapByReference(double &input1, double &input2);
```

Example :
Before: x = 9.9, y = 7.5
Call Swap(x,y)
After: x = 7.5, y = 9.9

Solution - Pass by pointer

```
void swap (double* first, double* second)
{
    double temp = *first;
    *first = *second;
    *second = temp;
}
```

```
int main()
{
    double a = 2, b = 3;
    std::cout << a << " " << b << std::endl;
    swap(&a, &b);
    std::cout << a << " " << b << std::endl;
    return 0;
}
```

Solution - Pass by reference

```
void swap (double& first, double& second)
{
    double temp = first;
    first = second;
    second = temp;
}
```

```
int main()
{
    double a = 2, b = 3;
    std::cout << a << " " << b << std::endl;
    swap(a, b);
    std::cout << a << " " << b << std::endl;
    return 0;
}
```

HW2 Q8

```
class CPPLib {
 public:
  // 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 UniqueVectorNotBySet(std::vector<int> &input);
  // You can use std::set
  void UniqueVectorBySet(std::vector<int> &input);

  // 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> IntersectVectors(std::vector<int> &input1,
                                    std::vector<int> &input2);
};
```

HW2 Q8

1. void CPPLib::UniqueVectorNotBySet(std::vector<int> &input)
2. void CPPLib::UniqueVectorBySet(std::vector<int> &input)
3. std::vector<int> CPPLib::IntersectVectors(std::vector<int> &input1, std::vector<int> &input2)

Note: Numbers in return vector could be in any order.
std::count: https://en.cppreference.com/w/cpp/algorithm/count

HW2 Q8

UniqueVectorNotBySet

```
Create a result vector
for x in input vector
      count number of occurrences of x in result
      If number of occurrences is 0
            push x to result vector
Set input as the result
```

UniqueVectorBySet

```
std::set<int> res_set(input.begin(),
input.end());


std::vector<int> result(res_set.begin(),
res_set.end());

Set input as the result
```

How to test the code?

```
bazel test files/8:student_test
```

HW2 Q8

IntersectVectors

```
Create a result vector
std::set<int> input1_set(input1.begin(), input1.end());
std::set<int> input2_set(input2.begin(), input2.end());

For x in input1_set
      If number of occurrences in input2_set is 1
            Push x to the result vector
Return the result vector
```