Smart Pointers & C++ STL

Lab 13

TA : Hyuna Seo, Kichang Yang, Minkyung Jeong, Jaeyong Kim

SEOUL NATIONAL UNIVERSITY

COPY CHECK Announcement

Announcement

- You should finish the lab practice and submit your job to eTL before the next lab class starts(Wednesday, 7:00 PM).
- The answer of the practice will be uploaded after the due.

Overview

- The concept of smart pointer
- Recap: C++ STL
  - Container: Vector, Map
  - Algorithm
- Practice : Restaurant Rating System (Part 1&2)

Objectives

- Learn concepts of smart pointers
- Practice C++ STL
  - How to use vectors and algorithms.

Smart Pointers

- In large programs with many programmers, it is hard to track all the pointers.
- Failing to handle pointers can lead to memory leak. Sometimes it causes fatal problems.

Smart Pointers

- C++ introduced **smart pointers** to avoid memory leak problems.
- Smart pointers are used to make sure that an object is deleted if it is no longer referenced. Programmers don't have to care about deleting memories manually.
- There are three kinds of smart pointers;

  `unique_ptr`, `shared_ptr`, and `weak_ptr`
- You may get detailed information here:

  https://en.cppreference.com/book/intro/smart_pointers
- Source code is in the skeleton zip.

Unique Pointers

- A `unique_ptr` can be owned by only one owner.
- Cannot be copied or shared.

```
#include <iostream>
#include <memory>
using std::unique_ptr; using std::make_unique;
int main() {
    unique_ptr<Test> test_unique1(new Test(1));
    unique_ptr<Test> test_unique2 = make_unique<Test>(2);
    //unique_ptr<Test> test_unique3 = test_unique2; // this is not allowed
    std::cout << "id : " << test_unique1->test_id << std::endl;
    std::cout << "id : " << test_unique2->test_id << std::endl;
    return 0;
}
```

Shared Pointers

- A `shared_ptr` can be owned by multiple owners.
- When no owner is using the object, it is destructed.

```
using std::shared_ptr; using std::make_shared;

shared_ptr<Test> test_shared() {
    shared_ptr<Test> test_shared1(new Test(1));
    shared_ptr<Test> test_shared2 = make_shared<Test>(2);
    shared_ptr<Test> test_shared3 = test_shared2;
    std::cout << "id : " << test_shared1->test_id << std::endl;
    std::cout << "id : " << test_shared2->test_id << std::endl;
    return test_shared3;
}
```

Shared Pointers (continued)

```
int main() {

    shared_ptr<Test> ptr = test_shared();

    std::cout << "id : " << ptr->test_id <<
std::endl;

    return 0;

}
```

Output

```
constructed

constructed

id : 1

id : 2

destructed

id : 2

destructed
```

Weak Pointers

- If two shared pointers point to each other, they are never released.
- `weak_ptr` pointing to a resource doesn’t affect the resource's reference count.
- When the last `shared_ptr` pointing the resource is destroyed, the resource will be freed, even if there are `weak_ptr` objects pointing to that resource.

Weak Pointers

```
int main() {
    shared_ptr<Test> test_shared1(new Test(1));
    shared_ptr<Test> test_shared2 = test_shared1;
    std::cout << "use count before : " << test_shared1.use_count() << std::endl;
    weak_ptr<Test> test_weak = test_shared1;
    std::cout << "id : " << test_weak.lock()->test_id << std::endl;
    std::cout << "use count after : " << test_weak.use_count() << std::endl;
    return 0;
}
```

Output

```
constructed
use count before : 2
id : 1
use count after : 2
destructed
```

Recap: Standard Template Library

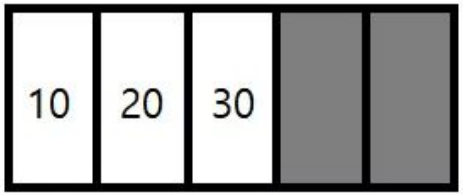
- A library consisting of a set of C++ template classes to provide data structures and functions such as lists, stacks, arrays.
  - You can consider this similar to JAVA Collections framework.
- In this practice, we will mainly learn how to use **vectors** and **algorithms**.
- You may find these links helpful:

  https://en.cppreference.com/w/cpp/container/vector

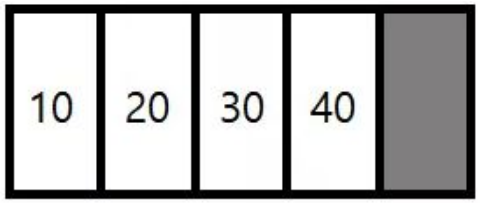
  https://en.cppreference.com/w/cpp/algorithm

Recap: `vector`

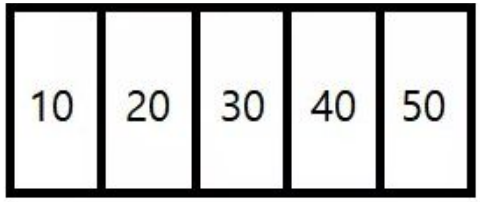
- A “dynamic” contiguous array.
  - Similar to ArrayList in Java

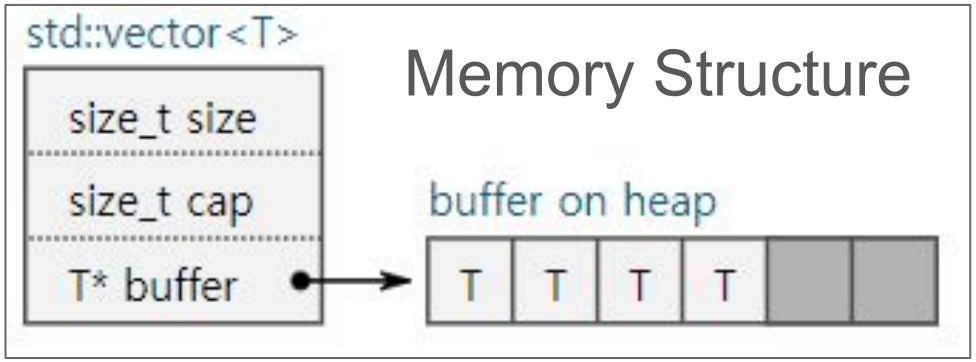


vec.push_back(40);

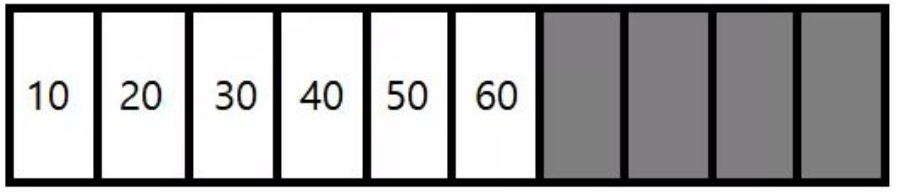


vec.push_back(50);





vec.push_back(60);



Recap: vector methods

- **push_back() : pushes** the elements into a vector from the back
- **pop_back() :** is used to **pop or remove** elements from a vector from the back
- **front() :** returns a reference to the **first** element in the vector
- **back() :** returns a reference to the **last** element in the vector
- **erase() :** is used to remove elements from a container from the specified position or range.

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

int main() {
  std::vector<int> vec;
  vec.push_back(10);
  vec.push_back(20);
  vec.push_back(30);
  vec.push_back(40);

  for (std::vector<int>::size_type i = 0;
  i < vec.size(); i++) {
    std::cout << vec[i] >> " ";
  }
  std::cout << std::endl;

    while(vec.size() > 0){

        std::cout << vec.back() << " ";

        vec.pop_back();

    }

    return 0;

}
```

Output:

```
10 20 30 40

40 30 20 10
```

Recap: vector & algorithm

- combination of erase() and remove() is recommend. remove() is in algorithm library.
- **vector.remove(first, last, value) :** delete the element same with the parameter *value.* This method doesn't resize the vector, so that you have to use **erase()** to completely delete the element. .

```
int main()
{
    std::vector<int> vec;
    vec.push_back(10);
    vec.push_back(20);
    vec.push_back(30);
    vec.push_back(40);

    std::cout<<"vector size = "<<vec.size()<<std::endl;

    vec.erase(std::remove(vec.begin(), vec.end(), 20),
                                                vec.end());

    std::cout<<"vector size = "<<vec.size()<<std::endl;

    for (int i : vec)
        std::cout<<i<<" ";
    return 0;
}
```

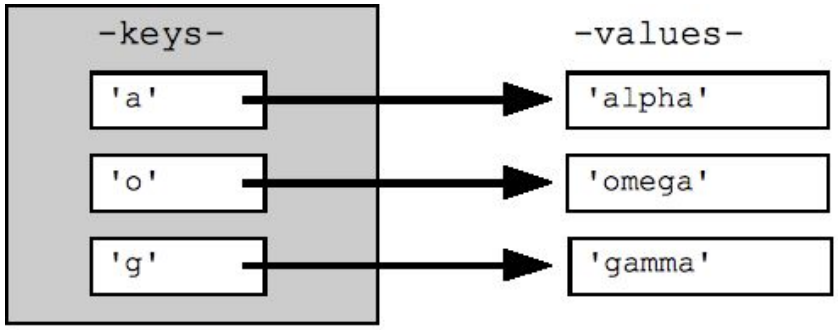
Output:

```
vector size = 4
vector size = 3
10 30 40
```

Recap: `map`

- `map` is a collection of key-value pairs, sorted by unique keys.

```
map <int, string> id_name;
// The type of key and value are specified in template
```



Recap: `map` Example (1/3)

```
#include <iostream>
#include <iterator>
#include <map>

using namespace std;

void printmap(map <int, int> g) {
    cout << "\tKEY\tELEM\n";
    for(auto itr=g.begin();itr!= g.end();++itr) {
        cout<<'\t'<<itr->first <<'\t'<<itr->second<<'\n';
    }
    cout << endl;
}
```

Recap: `map` Example (2/3)

```
int main() {
    // empty map container
    map<int, int> dict1;

    // insert elements in random order
    dict1.insert(pair<int, int>(1, 40));
    dict1.insert(pair<int, int>(2, 30));
    dict1.insert(pair<int, int>(3, 60));
    dict1.insert(pair<int, int>(4, 20));
    dict1.insert(pair<int, int>(5, 50));

    // printing map dict1
    printmap(dict1);
...
```

Recap: `map` Example (3/3)

```
...
// copying the elements
    map<int, int> dict2(dict1.begin(), dict1.end());

    // remove all elements up to key=3
    dict2.erase(dict2.begin(), dict2.find(3));
    cout << "\ndict2 after removal of";
    printmap(dict2);

    // remove all elements with key = 4
    dict2.erase(4);
    cout << "\ndict2.erase(4) : ";
    printmap(dict2);
    return 0;
}
```

Recap: `map` Example Output

```
    KEY  ELEM
    1    40
    2    30
    3    60
    4    20
    5    50


dict2 after removal of KEY  ELEM
    3    60
    4    20
    5    50


dict2.erase(4) :   KEY  ELEM
    3    60
    5    50
```

Recap: algorithm

- **sort(first, last, value) :** sorts the elements in the range [first,last) into ascending order.

```
int main()
{
    std::vector<int> vec;
    vec.push_back(10);
    vec.push_back(30);
    vec.push_back(40);
    vec.push_back(5);


    for (int i : vec)
        std::cout<<i<<" ";
    std::cout<<std::endl;

    std::sort(vec.begin(), vec.end());
    for (int i : vec)
        std::cout<<i<<" ";
    return 0;
}
```

Output:

```
10 30 40 5
5 10 30 40
```

Practice - Restaurant Rating System

- Let's implement a restaurant rating system.
- There are six functions that you have to implement.
  - RATE, LIST, SHOW, AVE, DEL, CHEAT

Commands of Restaurant Rating System:

```
Welcome to Restaurant Rating System
Commands : (EXIT to exit)
      RATE <name> <X> | LIST | SHOW <name>
      AVE <name> | DEL <name> <X> | CHEAT <name> <X>
```

Practice - Restaurant Rating System

- Use `std::map<string, shared_ptr<vector<int>>> restaurants` in `restaurante_app` class to store the pair of restaurant name and the rating data.
- Use `shared_ptr<vector<int>> find_restaurant(string target)` method which is already implemented in `restaurante_app` class.

- Assume that there are no duplicates in input restaurant names.
- Your program should get console inputs repeatedly and do the appropriate action each time.
- You don't have to consider wrong input types. Assume that `<name>` contains only alphanumeric characters, and `X` is always integer.

Restaurant Rating System: *Part 1*

1. `RATE <name> <X>`

- Implement **`void RestaurantApp::rate(string target, int rate)`**
- **Details**
  - Find the restaurant with <name> and add the rating data to the restaurant <name> by int <X>.
  - If the the restaurant with <name> doesn't exist, add restaurant <name> to your data.
  - Use `std::sort()` method in algorithm to sort the rating data in ascending order.

5 mins

Restaurant Rating System: *Part 1*

2. LIST

- Implement **`void RestaurantApp::list`**
- **Details**
  - Print all the restaurant names you have in one line.
  - Use range based for loop (auto).

- **Output Example**

Console Inputs

```
RATE McDonalds 10
RATE McDonalds 50
RATE McDonalds 35
RATE BurgerKing 20
LIST
```

Console Outputs

```
McDonalds BurgerKing
```

5 mins

3. SHOW <name>

- Implement **`void RestaurantApp::show(string target)`**
- **Details**
  - Print all the ratings of the restaurant with <name>.
  - The printed ratings should be sorted in ascending order.
  - If there is no restaurant with <name>, print **“<name> does not exist.”**.
- **Output Example**

Console Inputs

```
SHOW McDonalds
RATE McDonalds 10
RATE McDonalds 50
RATE McDonalds 35
RATE BurgerKing 20
LIST
SHOW McDonalds
```

Console Outputs

```
McDonalds does not exist.
McDonalds BurgerKing
10 35 50
```

Output Example

Console Inputs

```
SHOW McDonalds
RATE McDonalds 10
RATE McDonalds 50
RATE McDonalds 35
RATE BurgerKing 20
LIST
SHOW McDonalds
RATE BurgerKing 15
SHOW Lotteria
SHOW BurgerKing
RATE Lotteria 30
SHOW Lotteria
```

Console Outputs

```
McDonalds does not exist.
McDonalds BurgerKing
10 35 50
Lotteria does not exist.
15 20
30
```

5 mins

Restaurant Rating System: *Part 2*

1. AVE <name>

- Implement **`void RestaurantApp::ave(string target)`**
- **Details**
  - Print the average value of the rating of restaurant with <name>.
  - Round to the second digit below the decimal point.
  - If there is no restaurant with <name>, print **"<name> does not exist."**.
- **Output Example**

Console Inputs

```
RATE McDonalds 10
RATE McDonalds 50
RATE McDonalds 35
SHOW McDonalds
AVE McDonalds
```

Console Outputs

```
10 35 50
31.67
```

5 mins

Restaurant Rating System: *Part 2*

2. `DEL <name> <X>`

- Implement **`void RestaurantApp::del(string target, int rate)`**
- **Details**
  - Delete all the rating values same with int <X> from restaurant with <name>.
  - If there is no restaurant with <name>, print **“<name> does not exist.”**.

Restaurant Rating System: *Part 2*

`3. CHEAT <name> <X>`

- Implement **`void RestaurantApp::cheat(string target, int rate)`**
- **Details**
  - Delete all the rating values *less than* int <X> from restaurant with <name>.
  - If there is no restaurant with <name>, print **"<name> does not exist."**.

Output Example

Console Inputs

```
RATE McDonalds 10
RATE McDonalds 50
RATE McDonalds 35
RATE McDonalds 15
RATE McDonalds 5
RATE McDonalds 70
RATE McDonalds 25
SHOW McDonalds
DEL McDonalds 5
SHOW McDonalds
DEL McDonalds 20
SHOW McDonalds
CHEAT McDonalds 30
SHOW McDonalds
```

Console Outputs

```
5 10 15 25 35 50 70
10 15 25 35 50 70
10 15 25 35 50 70
35 50 70
```

Submission

- Compress your final project directory into a zip file.
- Rename your zip file as 20XX-XXXXX_{name}.zip - for example, 2021-12345_HyunaSeo.zip
- Upload it to eTL - Lab 12 assignment.
- You don’t have to consider about wrong input types not in this assignment.
- Today, we provide you the solution. You may just choose to follow the solution.