

Advanced Programming Concepts with C++



CSI 2372

Tutorial # 10 **Selected exercises** **from chapters 9 & 11**

Exercise 9.1:



- Which is the most appropriate—a vector, a deque, or a list—for the following program tasks? Explain the rationale for your choice. If there is no reason to prefer one or another container, explain why not.
 - a) Read a fixed number of words, inserting them in the container alphabetically as they are entered. We'll see in the next chapter that associative containers are better suited to this problem.
 - b) Read an unknown number of words. Always insert new words at the back. Remove the next value from the front.
 - c) Read an unknown number of integers from a file. Sort the numbers and then print them to standard output.
- **Answer:**
 - a) `std::set` is the best. now, we can select list, better than vector or deque, cause we may need to insert elements in the middle frequently to keep sorted alphabetical.
 - b) deque. If the program needs to insert or delete elements at the front and the back, but not in the middle, use a deque
 - c) vector, no need that insert or delete at the front or back. and If your program has lots of small elements and space overhead matters, don't use list or ***forward_list***.



Exercise 9.4:

- Write a function that takes a pair of iterators to a `vector<int>` and an `int` value. Look for that value in the range and return a `bool` indicating whether it was found.

```
bool find(vector<int>::iterator beg, vector<int>::iterator end, int value)
{
    for (auto iter = beg; iter != end; ++iter)
        if (*iter == value)
            return true;
        else
            return false;
}
```

Exercise 9.5

- Rewrite the previous program to return an iterator to the requested element. Note that the program must handle the case where the element is not found.

- **Answer:**

```
vector<int>::iterator find(vector<int>::iterator beg, vector<int>::iterator end, int value)
{
    for (auto iter = beg; iter != end; ++iter)
        if (*iter == value)
            return iter;
    else
        return end;
}
```



Exercise 9.7 & Exercise 9.8

- **9.7**
 - What type should be used as the index into a vector of *ints*?
- **Answer:**
 - `vector<int>::size_type`
- **9.8**
 - What type should be used to read elements in a list of strings? To write them?
- **Answer:**
 - `list<string>::iterator` or `list<string>::const_iterator` `// read`
 - `list<string>::iterator` `// write`

Exercise 9.15:

- Write a program to determine whether two `vector<int>`s are equal.

- **Answer:**

```
#include <iostream>
```

```
#include <vector>
```

```
int main()
```

```
{
```

```
    std::vector<int> vec1{ 1, 2, 3, 4, 5 };
```

```
    std::vector<int> vec2{ 1, 2, 3, 4, 5 };
```

```
    std::vector<int> vec3{ 1, 2, 3, 4 };
```

```
    std::cout << (vec1 == vec2 ? "true" : "false") << std::endl;
```

```
    std::cout << (vec1 == vec3 ? "true" : "false") << std::endl;
```

```
    return 0;
```

```
}
```



Exercise 9.16:



- Repeat the previous program, but compare elements in a `list<int>` to a `vector<int>`.

- **Answer:**

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

```
int main()
```

```
{
```

```
    std::list<int>    li{ 1, 2, 3, 4, 5 };
```

```
    std::vector<int>  vec2{ 1, 2, 3, 4, 5 };
```

```
    std::vector<int>  vec3{ 1, 2, 3, 4 };
```

```
    std::cout << (std::vector<int>(li.begin(), li.end()) == vec2 ? "true" : "false") << std::endl;
```

```
    std::cout << (std::vector<int>(li.begin(), li.end()) == vec3 ? "true" : "false") << std::endl;
```

```
    return 0;
```

```
}
```

Exercise 9.22:

- Assuming iv is a vector of ints, what is wrong with the following program? How might you correct the problem(s)?

```
vector<int>::iterator iter = iv.begin(), mid = iv.begin() + iv.size()/2;
while (iter != mid)
    if (*iter == some_val)
        iv.insert(iter, 2 * some_val);
```

- Answer:**

- Problems:

1. It's an endless loop. iter never equals mid.
2. mid will be invalid after the insert.



Amswer (fixing):

```
#include <iostream>
#include <vector>
void double_and_insert(std::vector<int>& v, int some_val)
{
    auto mid = [&]{ return v.begin() + v.size() / 2; };
    for (auto curr = v.begin(); curr != mid(); ++curr)
        if (*curr == some_val)
            ++(curr = v.insert(curr, 2 * some_val));
}
int main()
{
    std::vector<int> v{ 1, 9, 1, 9, 9, 9, 1, 1 };
    double_and_insert(v, 1);
    for (auto i : v)
        std::cout << i << " ";
    std::cout << std::endl;
}
```

2 1 9 2 1 9 9 9 1 1

Exercise 9.26:

- Using the following definition of `ia`, copy `ia` into a vector and into a list. Use the single-iterator form of `erase` to remove the elements with odd values from your list and the even values from your vector.
 - `int ia[] = { 0, 1, 1, 2, 3, 5, 8, 13, 21, 55, 89 };`

- **Answer:**

```
#include <vector>
#include <list>
#include <iterator>
#include <iostream>
int main() {
    int ia[] = {0, 1, 1, 2, 3, 5, 8, 13, 21, 55, 89};
    std::vector<int> iv(std::begin(ia), std::end(ia));
    std::list<int> il(std::begin(ia), std::end(ia));
    std::cout << "Before erase:" << std::endl;
    std::cout << "vector iv: ";
    for (const auto &i: iv)
        std::cout << i << " ";
    std::cout << std::endl;
```

```

std::cout << "list  il: ";
for (const auto &i: il)
    std::cout << i << " ";
std::cout << std::endl;
for (auto it = iv.begin(); it != iv.end(); )
    if (*it % 2)
        ++it;
    else
        it = iv.erase(it);
for (auto it = il.begin(); it != il.end(); )
    if (*it % 2)
        it = il.erase(it);
    else
        ++it;
std::cout << "After erase:" << std::endl;
std::cout << "vector iv: ";
for (const auto &i: iv)
    std::cout << i << " ";
std::cout << std::endl;
std::cout << "list  il: ";
for (const auto &i: il)
    std::cout << i << " ";
std::cout << std::endl;
return 0;

```

Before erase:
vector iv: 0 1 1 2 3 5 8 13 21 55 89
list il: 0 1 1 2 3 5 8 13 21 55 89
After erase:
vector iv: 1 1 3 5 13 21 55 89
list il: 0 2 8



Exercise 11.7:

- Define a map for which the key is the family's last name and the value is a vector of the children's names. Write code to add new families and to add new children to an existing family.

Answer: 11.7

```
#include <iostream>
#include <map>
#include <string>
#include <algorithm>
#include <vector>
using namespace std;
using Families = map<string, vector<string>>;
auto make_families(){
    Families families;
    for (string ln; cout << "Last name:\n", cin >> ln && ln != "@q";)
        for (string cn; cout << "|-Children's names:\n", cin >> cn && cn != "@q";)
            families[ln].push_back(cn);
    return families;
}
auto print(Families const& families){
    for (auto const& family : families){
        cout << family.first << ":\n";
        for (auto const& child : family.second)
            cout << child << " ";
        cout << "\n";
    }
}
int main(){
    print(make_families());
    return 0;
}
```



uOttawa

Ahmedou Jreivine



Exercise 11.17:

- Assuming `c` is a multiset of strings and `v` is a vector of strings, explain the following calls. Indicate whether each call is legal:

`copy(v.begin(), v.end(), inserter(c, c.end()));`

`copy(v.begin(), v.end(), back_inserter(c));`

`copy(c.begin(), c.end(), inserter(v, v.end()));`

`copy(c.begin(), c.end(), back_inserter(v));`

- Answer:**

- `copy(v.begin(), v.end(), inserter(c, c.end()));` // legal
- `copy(v.begin(), v.end(), back_inserter(c));` // illegal, no `push_back` in `set`.
- `copy(c.begin(), c.end(), inserter(v, v.end()));` // legal.
- `copy(c.begin(), c.end(), back_inserter(v));` // legal.



Exercise 11.21:

- Assuming `word_count` is a map from `string` to `size_t` and `word` is a `string`, explain the following loop:
 `while (cin >> word)`
 `++word_count.insert({word, 0}).first->second;`

Answer:

This code can be explained like this pseudocode:

```
while reading into word
    if word_count has key word:
        word_count[word] += 1
    else:
        word_count[word] = 0
        word_count[word] += 1
```

Exercise 11.24-25:



Exercise 11.24:

What does the following program do?

```
map<int, int> m;  
m[0] = 1;
```

Answer:

it adds a key-value pair { 0, 1 } into the map.

Exercise 11.25:

Contrast the following program with the one in the previous exercise

```
vector<int> v;  
v[0] = 1;
```

Answer:

Undefined Behavior, since it's trying to dereference an item out of range.

Exercise 11.26:



What type can be used to subscript a map? What type does the subscript operator return? Give a concrete example—that is, define a map and then write the types that can be used to subscript the map and the type that would be returned from the subscript operator.

Answer:

```
#include <iostream>
#include <map>
#include <string>
#include <typeinfo>
using namespace std;
int main()
{
    // ex11.26
    map<int, string> m = { { 1,"ss" }, { 2,"sz" } };
    using KeyType = map<int, string>::key_type;
    cout << "type to subscript: " << typeid(KeyType).name() << endl;
    cout << "returned from the subscript operator: " << typeid(decatype(m[1])).name() << endl;
    return 0;
}
```

Exercise 11.31:



- Write a program that defines a multimap of authors and their works. Use *find* to find an element in the multimap and erase that element.
 - Be sure your program works correctly if the element you look for is not in the map.

```

#include <map>
#include <string>
#include <iostream>
using std::string;
int main()
{
    std::multimap<string, string> authors{ { "alan", "DMA" },
                                           { "pezy", "LeetCode" },
                                           { "alan", "CLRS" },
                                           { "wang", "FTP" },
                                           { "pezy", "CP5" },
                                           { "wang", "CPP-Concurrency" }

};
// want to delete an element that author is [Alan], work is [112].
string author = "pezy";
string work = "CP5";
auto found = authors.find(author);
auto count = authors.count(author);
while (count) {
    if (found->second == work) {
        authors.erase(found);
        break;
    }
    ++found;
    --count;
}
for (const auto &author : authors)
    std::cout << author.first << " " << author.second << std::endl;

```





Exercise 11.34:

- What would happen if we used the subscript operator instead of find in the transform function?

- **Answer:**

Say the code has been changed like below:

```
const string& transform(const string &s, const map<string, string> &m)
{
    return m[s];
}
```

The above code won't compile because the subscript operator might insert an element (when the element with the key `s` is not found), and we may use subscript only on a map that is not const.



Exercise 11.35:

- In *buildMap*, what effect, if any, would there be from rewriting:
 `trans_map[key] = value.substr(1);` as
 `trans_map.insert({ key, value.substr(1) })`?
- **Answer:**
 - use subscript operator: if a word does appear multiple times, our loops will put the last corresponding phrase into `trans_map` use `insert`: if a word does appear multiple times, our loops will put the first corresponding phrase into `trans_map`

Exercise 11.37:

- What are the advantages of an unordered container as compared to the ordered version of that container? What are the advantages of the ordered version?
- **Answer:**
- **Ordered Associative Container**
 - Standard Traversal encounters elements in sorted order
 - Order predicate may be specified
 - Default order predicate is "less than", defined using operator< for the element type
 - Popular implementations: `OrderedVector`, `BinarySearchTree`
 - Search operations required to have $O(\log n)$ runtime
 - Insert, Remove operations should either be seldom used or have $O(\log n)$ runtime
- **Unordered Associative Container**
 - Standard Traversal encounters elements in unspecified order
 - Search, Insert, Remove operations should have average-case constant runtime
 - Popular implementations use hashing.



Refereces



Accreditation:

- This presentation is prepared/extracted from the following resources:
 - C++ Primer, Fifth Edition.
Stanley B. Lippman Josée Lajoie Barbara E. Moo
 - <https://github.com/jaege/Cpp-Primer-5th-Exercises>
 - <https://github.com/Mooophy/Cpp-Primer>
 - <https://github.com/pezy/CppPrimer>