Exercises week 5: Containers - Revision

Klaas Isaac Bijlsma
s2394480

David Vroom
s2309939

January 10, 2018

Exercise 36

Learn to select the right container for the task at hand

In the previous attempt, we used a multiset. Now we have used a map to construct a program that prints a sorted list of all different words appearing at its standard input, together with the number of times is was entered as input.

main.cc

```
#include <iostream>
#include <map>
#include <string>

using namespace std;

int main()
{
    map<string, size_t> object;

    cout << "Enter a sequence of words (enter \"done\" when done):\n";
    string word;
    while (cin >> word)
    {
        if (word == "done")
            break;

        if (object.count(word))
```

```cpp
            object[word] += 1;  // if word already entered, increment value
        else                    // new word, insert and set value to 1
            object.insert(pair<string, size_t>{ word, 1 });
    }

    for (auto iter = object.begin(); iter != object.end(); ++iter)
        cout << iter->first << ' ' << iter->second << '\n';
}
```

Exercise 37

Learn to shed excess capacity from a vector

In the previous attempt, we added the elements of the set to the vector one by one. This is improved now by initializing the vector using iterators.

The output of the program of exercise 35 is now stored in a vector. Then one additional word is added to the vector. If needed, its excess capacity is shedded. Below is our code and the output it generates when initally two (different) input words are given. It can be seen that the capacity is nicely reduced from 4 to 3.

main1.cc

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

using namespace std;

int main()
{
    set<string> setObj;

    cout << "Enter a sequence of words (enter \"done\" when done):\n";
    string word;
    while (cin >> word)
    {
        if (word == "done")
            break;
        setObj.insert(word);
    }

    vector<string> vecObj(setObj.begin(), setObj.end());

    cout << "1. Size: " << vecObj.size() << '\n'
         << "1. Capacity: " << vecObj.capacity() << '\n';

    vecObj.push_back("new");

    cout << "2. Size: " << vecObj.size() << '\n'
         << "2. Capacity: " << vecObj.capacity() << '\n';

    vector<string>(vecObj).swap(vecObj);

    cout << "3. Size: " << vecObj.size() << '\n'
         << "3. Capacity: " << vecObj.capacity() << '\n';
}
```

Output of main1.cc

```
Enter a sequence of words (enter "done" when done):
1. Size: 2
1. Capacity: 2
2. Size: 3
2. Capacity: 4
3. Size: 3
3. Capacity: 3
```

Now essentially the same is done, but by using a class VectorData that has a `vector<string>` data member. The output (from input ”2 1 done”) shows that again excess capacity is shedded.

vectordata/vectordata.h

```
#ifndef EX37B_VECTORDATA_H
#define EX37B_VECTORDATA_H

#include <vector>
#include <set>

class VectorData
{
    typedef std::set<std::string>::iterator setStrIter;

    std::vector<std::string> d_data;

    public:
        VectorData(setStrIter first, setStrIter last);

        void add(std::string const &word);
        void swap(VectorData &other);
        size_t size() const;
        size_t capacity() const;
};

inline VectorData::VectorData(setStrIter first, setStrIter last)
:
    d_data(first, last)
{}

inline void VectorData::add(std::string const &word)
{
    d_data.push_back(word);
}

inline void VectorData::swap(VectorData &other)
{
    d_data.swap(other.d_data);
}

inline size_t VectorData::size() const
{
    return d_data.size();
}

inline size_t VectorData::capacity() const
{
    return d_data.capacity();
}

#endif
```

main.cc

```
#include <iostream>
#include <set>
#include "vectordata/vectordata.h"

using namespace std;

```

```cpp
int main()
{
    set<string> setObj;

    cout << "Enter a sequence of words (enter \"done\" when done):\n";
    string word;
    while (cin >> word)
    {
        if (word == "done")
            break;
        setObj.insert(word);
    }

    VectorData vecObj(setObj.begin(), setObj.end());

    cout << "1. Size: " << vecObj.size() << '\n'
         << "1. Capacity: " << vecObj.capacity() << '\n';

    vecObj.add("new");

    cout << "2. Size: " << vecObj.size() << '\n'
         << "2. Capacity: " << vecObj.capacity() << '\n';

    VectorData(vecObj).swap(vecObj);

    cout << "3. Size: " << vecObj.size() << '\n'
         << "3. Capacity: " << vecObj.capacity() << '\n';
}
```

Output of main.cc

```
Enter a sequence of words (enter "done" when done):
1. Size: 2
1. Capacity: 2
2. Size: 3
2. Capacity: 4
3. Size: 3
3. Capacity: 3
```

Exercise 38

Learn to fine-tune the unordered_multimap::count member

In the previous attempt, we made a function that computes in a bit complex way the number of unique keys. No we used a simpler way to determine this number.

main.cc

```
#include <iostream>
#include <string>
#include <unordered_map>

using namespace std;

int main(int argc, char **argv)
{
    typedef unordered_multimap<string, string> container;

    container object({{ "Netherlands", "Amsterdam" },
                      { "Belgium", "Brussels" },
                      { "Belgium", "Antwerp" },
                      { "France", "Nice" },
                      { "France", "Paris" }});

    size_t nUniqueKeys = unordered_map<string, string>
                        (object.begin(), object.end()).size();
    cout << "There are " << nUniqueKeys
         << " unique keys in the container\n";
}
```