Programming in C/C++
Exercises set four: containers

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

December 7, 2016

Exercise 22, Containers solving complex tasks

We are asked to order all words obtain by the standard input and print them in the screen.

Code listings

Listing 1: main.cc

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

int main()
{
    std::vector<std::string> vec;
    std::string stringTemp;

    while (std::cin >> stringTemp)
        vec.push_back(stringTemp);

    sort(vec.begin(), vec.end());
    vec.erase(unique(vec.begin(), vec.end()),
        vec.end());

    for (std::size_t ind = 0; ind < vec.size(); ++ind)
        std::cout << ind  << ".\t" << vec[ind]
            << std::endl;
}
```

Exercise 23, vectors and shrinking

So we experimented with slicing off extra capacity with vectors and a class with a vector as a data member.

Output

```
size: 10 capacity: 16
size: 11 capacity: 16
size: 11 capacity: 11

size: 11 capacity: 16
size: 12 capacity: 16
size: 12 capacity: 12
```

Code listings

Listing 2: main.ih

```
#include "main.h"

using namespace std;
```

Listing 3: main.h

```
#ifndef MAIN_H_
#define MAIN_H_

#include <iostream>
#include "uwl/uniquewordlist.h"

void reader(std::istream &stream,
  std::vector<std::string> &wordList);
void printer(std::ostream &stream,
  std::vector<std::string> const &wordList);
void printer(std::ostream &stream,
  UniqueWordList const &wordList);
```

```

#endif
```

Listing 4: main.cc

```
#include "main.ih"

int main(int argc, char **argv)
{
  vector<string> wordList;
  reader(cin, wordList);
  cout << "size: " << wordList.size()
    << " capacity: " << wordList.capacity() << '\n';

  wordList.push_back("test");
  printer(cout, wordList);

  wordList.shrink_to_fit();
  printer(cout, wordList);

  UniqueWordList uwl;
  for (auto it = wordList.begin(); it != wordList.end
   (); ++it)
    uwl.addWord(*it);
  cout << '\n';

  printer(cout, uwl);

  uwl.addWord("west");
  printer(cout, uwl);

  uwl = uwl;
  printer(cout, uwl);
}
```

Listing 5: printer1.cc

```
#include "main.ih"

void printer(ostream &stream,
  vector<string> const &wordList)
```

```
{
  stream << "size: " << wordList.size()
    << " capacity: " << wordList.capacity() << '\n';
}
```

Listing 6: printer2.cc

```
#include "main.ih"

void printer(ostream &stream,
  UniqueWordList const &wordList)
{
  stream << "size: " << wordList.size()
    << " capacity: " << wordList.capacity() << '\n';
}
```

Listing 7: reader.cc

```
#include "main.ih"

#include <algorithm>

void reader(istream &stream, vector<string> &wordList)
{
  string word;
  while (stream >> word)
  {
    if (!findWord(wordList, word))
      wordList.push_back(word);
  }
}
```

UniqueWordList

Listing 8: uniquewordlist.ih

```
#include "uniquewordlist.h"

using namespace std;
```

Listing 9: uniquewordlist.h

```
#ifndef UNIQUEWORDLIST_H_
```

```cpp
#define UNIQUEWORDLIST_H_

#include <vector>
#include <string>

class UniqueWordList
{
  std::vector<std::string> d_list;

  public:
    UniqueWordList &operator=(
      UniqueWordList const &uwl);

    void swap(UniqueWordList &uwl);

    void addWord(std::string word);

    std::size_t size();
    std::size_t capacity();

    std::size_t size() const;
    std::size_t capacity() const;
};

bool findWord(std::vector<std::string> &wordList,
  std::string word);

#endif
```

Listing 10: addword.cc

```cpp
#include "uniquewordlist.ih"

#include <algorithm>

void UniqueWordList::addWord(string word)
{
  if (!findWord(d_list, word))
    d_list.push_back(word);
}
```

Listing 11: capacity.cc

```
#include "uniquewordlist.ih"

size_t UniqueWordList::capacity()
{
  return d_list.capacity();
}
```

Listing 12: capacityconst.cc

```
#include "uniquewordlist.ih"

size_t UniqueWordList::capacity() const
{
  return d_list.capacity();
}
```

Listing 13: findword.cc

```
#include "uniquewordlist.ih"

bool findWord(vector<string> &wordList,
  string word)
{
  for (auto it = wordList.begin();
    it != wordList.end(); ++it)
  {
    if (*it == word)
      return true;
  }

  return false;
}
```

Listing 14: operator=.cc

```
#include "uniquewordlist.ih"

UniqueWordList &UniqueWordList::operator=(
  UniqueWordList const &uwl)
{
  UniqueWordList copy(uwl);
  swap(copy);
  return *this;
}
```

Listing 15: size.cc

```
#include "uniquewordlist.ih"

size_t UniqueWordList::size()
{
  return d_list.size();
}
```

Listing 16: sizeconst.cc

```
#include "uniquewordlist.ih"

size_t UniqueWordList::size() const
{
  return d_list.size();
}
```

Listing 17: swap.cc

```
#include "uniquewordlist.ih"

#include <cstring>

void UniqueWordList::swap(UniqueWordList &uwl)
{
  char bytes[sizeof(UniqueWordList)];
  memcpy(bytes, this, sizeof(UniqueWordList));
  memcpy(this, &uwl, sizeof(UniqueWordList));
  memcpy(&uwl, bytes, sizeof(UniqueWordList));
}
```

Exercise 24, Containers solving complex tasks

Now, we are asked to count the number of repetitions of each word, this is a continuation from exercise 22.

Code listings

Listing 18: main.cc

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

int main()
{
    std::vector<std::string> vec;
    std::string stringTemp;

    while (std::cin >> stringTemp)
        vec.push_back(stringTemp);

    sort(vec.begin(), vec.end());

    for (std::size_t ind = 0; ind < vec.size(); ++ind)
        std::cout << ind  << ".\t" << vec[ind]
            << std::endl;

    std::cout << std::endl;
    //End algorithm from exercise 22

    std::vector<std::string> vecUn;
    vecUn = vec;
    vecUn.erase(unique(vecUn.begin(), vecUn.end()),
        vecUn.end());

    for (std::size_t ind = 0; ind < vecUn.size();
        ++ind)
    {
        std::size_t count = std::count (vec.begin(),
            vec.end(), vecUn[ind]);

        std::cout << "The element " << vecUn[ind]
            << " is repited " << count - 1 << " times"
            << std::endl;
    }

}
```

Exercise 25, unique keys

We made a snippet of code to count the number of unique keys in an `unordered_multimap`. Never again.

Code listings

Listing 19: main.cc

```cpp
#include <unordered_map>
#include <set>
#include <algorithm>
#include <string>
#include <iostream>

using namespace std;

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

  // fill the container with data
  // (no need to implement this)


  set<string> keys;
  for (size_t bucket = 0;
    bucket != container.bucket_count(); ++bucket)
  {
    for (auto it = container.begin(bucket);
      it != container.end(bucket); ++it)

      keys.insert(it->first)
  }
  size_t nUniqueKeys = keys.size();

  cout << "There are " << nUniqueKeys
    << " in the container\n";
}
```

Exercise 26, signal handling

We made the class interface for the Signal class and made a TestHandler class that inherits from the class SignalHandler.

Code listings

Listing 20: signal.h

```
#include "signal.h"
#include <iostream>
#include <signal.h>

using namespace std;
```

Listing 21: signal.h

```
#ifndef SIGNAL_H
#define SIGNAL_H

#include <map>

class Signal
{
  // map to store pair of signal with
  // set of signalhandlers
  map<size_t,
    set<SignalHandler>> d_signalHandlerMap;
  static Signal *s_instance = NULL;

  public:
    Signal(Signal const &other) = delete;
    static Signal &instance();

  private:
    Signal();
    ~Signal();
    // calls the signalhanders for the
    // given signal it is linked to all
    // required signals using sigaction
    void (*processSignal)(size_t signum);
    void add(size_t signum,
      SignalHandler &object);
    void remove(size_t signum,
      SignalHandler &object);
    void ignore(size_t signum);
    void reset(size_t signum);
};

#endif
```

Listing 22: signalhandler.ih

```
#include "signalhandler.h"
#include <iostream>

using namespace std;
```

Listing 23: signalhandler.h

```
#ifndef SIGNALHANDLER_H
#define SIGNALHANDLER_H

class SignalHandler
{
    friend class Signal;

    public:
        virtual ~SignalHandler();
    private:
        virtual void signalHandler(size_t signum) = 0;
};

#endif
```

Listing 24: testhandler.h

```
#ifndef TESTHANDLER_H
#define TESTHANDLER_H

class TestHandler: public SignalHandler
{
    friend class Signal;

    public:
        TestHandler();
        virtual ~TestHandler() override;
    private:
        virtual void signalHandler(
            size_t signum) override;
};

#endif
```

Listing 25: testhandler.cc

```
#include "signalhandler.ih"

TestHandler::TestHandler()
{
    Signal.instance().add(SIGINT, *this);
}
```

Listing 26: destructor testhandler.cc

```
#include "signalhandler.ih"

virtual void TestHandler::~TestHandler()
{
    Signal.instance().remove(SIGINT);
}
```

Exercise 27, implementing singleton functionality

We have implemented the member function that belong to the singleton property of the class Signal.

Code listings

Listing 27: instance.cc

```
#include "signal.ih"

static Signal &Signal::instance();
{
    return (s_instance == NULL ? s_instance = new
   Signal : *Signal)
}
```

Listing 28: destructor of signal

```
#include "signal.ih"

Signal::~Signal()
{
}
```