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

Christiaan Steenkist
Jaime Betancor Valado
Remco Bos

December 1, 2016

## 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 1: main.ih

```
#include "main.h"

using namespace std;
```

Listing 2: 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 3: main.cc

```
#include "main.ih"
#include "uwl/uniquewordlist.h"

int main(int argc, char **argv)
{
  vector<string> wordList;
  reader(cin, wordList);
  printer(cout, wordList);

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

  wordList = vector<string>(wordList);
  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 4: printer1.cc

```
#include "main.ih"

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

Listing 5: printer2.cc

```
#include "main.ih"

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

Listing 6: reader.cc

```
#include "main.ih"

#include <algorithm>

void reader(istream &stream, vector<string> &wordList)
{
  string word;
  while (stream >> word)
  {
    if (find(wordList.begin(), wordList.end(), word)
      == wordList.end())

      wordList.push_back(word);
  }
}
```

**UniqueWordList**

Listing 7: uniquewordlist.ih

```
#include "uniquewordlist.h"

using namespace std;
```

Listing 8: uniquewordlist.h

```
#ifndef UNIQUEWORDLIST_H_
#define UNIQUEWORDLIST_H_

#include <vector>
#include <string>

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

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

    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;
};
```

```

#endif
```

Listing 9: addword.cc

```
#include "uniquewordlist.ih"

#include <algorithm>

void UniqueWordList::addWord(string word)
{
  if (find(d_list.begin(), d_list.end(), word)
    == d_list.end())

    d_list.push_back(word);
}
```

Listing 10: capacity.cc

```
#include "uniquewordlist.ih"

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

Listing 11: capacityconst.cc

```
#include "uniquewordlist.ih"

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

Listing 12: `operator=`.cc

```
#include "uniquewordlist.ih"

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

Listing 13: size.cc

```
#include "uniquewordlist.ih"

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

Listing 14: sizeconst.cc

```
#include "uniquewordlist.ih"

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

Listing 15: 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 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 16: main.cc

```cpp
#include "main.ih"
#include "uwl/uniquewordlist.h"

int main(int argc, char **argv)
{
  vector<string> wordList;
  reader(cin, wordList);
  printer(cout, wordList);

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

  wordList = vector<string>(wordList);
  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);
}
```

## 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 17: signal.h

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

using namespace std;
```

Listing 18: 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 19: signalhandler.ih

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

using namespace std;
```

Listing 20: 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 21: 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 22: testhandler.cc

```
#include "signalhandler.ih"

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

Listing 23: 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 24: instance.cc

```
#include "signal.ih"

static Signal &Signal::instance();
{
    if (s_instance == NULL)
        s_instance = new Signal;

    return *Signal;
}
```

Listing 25: destructor of signal

```
#include "signal.ih"

Signal::~Signal()
{
    delete s_instance;
}
```