Codebeispiele

C++

Julian Klaiber und Severin Dellsperger

Hochschule für Technik Rapperswil

2. Mai 2019

Lizenz
"THE BEER-WARE LICENSE" (Revision 42): Julian and Severin wrote this file. As long as you retain this notice you can do whatever you want with this stuff. If we meet some day, and you think this stuff is worth it, you can buy us a beer in return.

Inhaltsverzeichnis

1 Drill questions

namespace	headerfile	variable	function	type	description
std::cin	iostream	x			Represent standard input stream
std::endl	ostream		x		Inserts a newline character into the output sequence os and flushes it as if by calling.
std::tolower	cctype		x		Converts given character to lowercase (A -> a).
std::string	string			x	Stores and manipulates sequences of char-like objects.
std::distance	iterator		x		Returns the number of hops from first to last.
std::istream_iterator	iterator			x	Read successive objects of type T from the basic_istream.
std::size_t	cstddef			x	Unsigned integer type of the result of the sizeof operator.
std::vector	vector			x	Sequence container that encapsulates dynamic size arrays.

!

1.1 Element Iteration

	const: • element cannot be changed	**non-const:** • element can be changed

reference:
• element in vector is accessed
	`for (auto const & cref : v) {`
	`  std::cout << cref << '\n';`
	`}`
		`for (auto & ref : v) {`
		`  ref *= 2;`
		`}`

copy:
• loop has own copy of the element
	`for (auto const ccopy : v) {`
	`  std::cout << ccopy << '\n';`
	`}`
		`for (auto copy : v) {`
		`  copy *= 2;`
		`  std::cout << copy << '\n';`
		`}`

!

2 Stack and Queue

```cpp
#include <stack>
#include <queue>
#include <iostream>
#include <string>

int main() {
   std::stack<std::string> lifo{};
   std::queue<std::string> fifo{};
   for (std::string s : { "Fall", "leaves", "after", "leaves", "fall" }) {
      lifo.push(s);
      fifo.push(s);
   }
   while (!lifo.empty()) { // fall leaves after leaves Fall
      std::cout << lifo.top() << ' ';
      lifo.pop(); } std::cout << '\n';
      while (!fifo.empty()) {// Fall leaves after leaves fall
         std::cout << fifo.front() << ' ';
         fifo.pop();
      }
   }
}
```

3 MAP

3.1 counting word

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

int main(){
   std::map<std::string, size_t> words{};
   std::string s{};

   while (std::cin >> s) {
   ++words[s];
   }
   for(auto const & p : words) {
   std::cout << p.first << " = "<< p.second << '\n';
   }
}
```

4 Istream and Ostream

4.1 robust reading of an int value

```
int inputAge(std::istream & in) {
   while (in.good()) {
      std::string line{};
      getline(in, line);
      std::istringstream is{line};
      int age{-1};
      if (is >> age) {
      return age;
      }
   }
   return -1;
}
```

4.2 Implementing Read

```
class Date {
   int year, month, day;
public:
   std::istream & read(std::istream & is) {
         int year{-1}, month{-1}, day{-1};
         char sep1, sep2;
         //read values
         is >> year >> sep1 >> month >> sep2 >> day;
      try {
         Date input{year, month, day};
         //overwrite content of this object (copy-ctor)
         (*this) = input;
         //clear stream if read was ok
         is.clear();
      } catch (std::out_of_range const & e) {
         //set failbit
         is.setstate(std::ios::failbit);
      }
      return is;
   }
};
```

4.3 Implementing Print

Date.h

```
#include <ostream>

class Date {
   int year, month, day;
public:
   std::ostream & print(std::ostream & os) const {
   os << year << "/" << month << "/" << day;
   return os;
  }
};
inline std::ostream & operator<<(std::ostream & os, Date const & date) {
   return date.print(os);
```

```
13 }
```

Any.cpp

```
1 #include "Date.h"
2 #include <iostream>
3
4 void foo() {
5    std::cout << Date::myBirthday;
6 }
```

5 Count

5.1 non-whitespace Chars

char.h

```h
#include <iosfwd>
int charcount(std::istream &in);
```

char.cpp

```cpp
int charcount(std::istream &in){
   std::string line{};
   std::getline(input, line);
   line.erase(remove(line.begin(), line.end(),' '),line.end());
   return line.size();
}
```

main.cpp

```cpp
#include <charc.h>
#include <iostream>
int main(){
   std::cout << charc(std::cin) << '\n';
}
```

without loop char.cpp

```cpp
int charcount(std::istream &in){
  using initer = std::istream_iterator<char>;
  return std::vector<char>{initer{in},initer{}}.size();
}
```

5.2 all Chars

header

```h
int allcharcount(std::istream &in){
}
```

cpp

```cpp
void allcharc(std::istream & in, std::ostream & out) {
   char c { };
   int counter { 0 };
   while (in.get() >> c && in.eof() == false) {
      counter++;
   }
   out << counter;
}
```

without loop cpp

```cpp
int allcharcount(std::istream &in){
}
void allcharc(std::istream & in, std::ostream & out) {
   std::noskipws (in);
   std::istream_iterator<char> input {in};
   std::istream_iterator<char> eos{};
   out << std::distance(input, eos);
}
```

5.3 Words

header

```
int wordcount(std::istream &in){
}
```

cpp

```
void wc(std::istream & in, std::ostream & out) {
   std::string word{};
   int counter{0};
   while (in >> word) {
      counter++;
   }
   out << counter;
}
```

5.4 Lines

header

```
int linecount(std::istream &in){
}
```

cpp

```
void lc(std::istream & in, std::ostream & out) {
   std::string line{};
   int counter{0};
   while (!in.eof()) {
   std::getline(in, line);
   counter++;
   }
   out << counter;
}
```

6 Testat 1

6.1 calc.cpp

```cpp
#include "calc.h"
#include <exception>
#include <istream>
int calc(int lhs, int rhs, char op) {
   switch(op) {
      case '+' : return lhs + rhs;
      case '-' : return lhs - rhs;
      case '*' : return lhs * rhs;
      case '/' :
      case '%':
         if(rhs == 0)
         {
            throw std::invalid_argument{"Modulo by zero"};
         }
         else
         {
            if(op == '/')
            {
               return lhs / rhs;
            }
            else
            {
               return lhs % rhs;
            }
         }
      default: throw std::invalid_argument{"Invalid Argument"};
   }
}
int calc(std::istream& in)
{
   int lhs{}, rhs{};
   char op{};
   if(in >> lhs >> op >> rhs)
   {
      return calc(lhs, rhs, op);
   }

   throw std::invalid_argument{"Invalid Argument!"};
}
```

6.2 calc.h

```cpp
#ifndef CALC_H_
#define CALC_H_
#include <iosfwd>

int calc(int lhs, int rhs, char op);
int calc(std::istream& in);

#endif /* CALC_H_ */
```

6.3 pocketcalculator.cpp

```
#include "pocketcalculator.h"
#include "calc.h"
#include "sevensegment.h"
#include <iostream>
#include <exception>
#include <string>
#include <sstream>

const unsigned MAXDIGITLENGTH{8};

void startCalculator(std::ostream &out, std::istream &in)
{
   std::string line{};
   while(getline(in, line))
   {
    std::istringstream argument{line};
      try {
         int result = calc(argument);
         std::string digits = std::to_string(result);
         if(digits.length() <= MAXDIGITLENGTH)
         {
            printLargeNumber(result, out);
         }
         else
         {
            throw std::length_error{"Too many digits"};
         }
      } catch (...) {
         printErrorMessage(out);
      }
   }
}
```

6.4 pocketcalculator.h

```
#ifndef POCKETCALCULATOR_H_
#define POCKETCALCULATOR_H_
#include <iosfwd>

void startCalculator(std::ostream &out, std::istream &in);

#endif /* POCKETCALCULATOR_H_ */
```

6.5 sevensegment.cpp

```
#include "sevensegment.h"
#include <ostream>
#include <algorithm>
#include <vector>
#include <string>
void printLargeNumber(int i, std::ostream &out)
{  std::string number = std::to_string(i);
   if(i > 0)
   {
         for(int line = 0; line < 5; line++)
         {

```

```
13             std::for_each(number.begin(), number.end(), [&out, line](auto digit){
14                digit = digit - '0';
15                out << digitsvector.at(digit).at(line);
16             });
17             out << '\n';
18          }
19    }
20    else
21    {
22       for(int line = 0; line < 5; line++)
23             {
24
25                std::for_each(number.begin(), number.end(), [&out, line](auto symbol){
26                   if(symbol == '-' )
27                      {
28                         out << minusvector.at(line);
29                      }
30                   else
31                      {
32                         symbol = symbol - '0';
33                         out << digitsvector.at(symbol).at(line);
34                      }
35
36                });
37                out << '\n';
38             }
39    }
40
41 }
42 void printErrorMessage(std::ostream &out)
43 {
44    std::for_each(errorvector.begin(), errorvector.end(), [&out](auto line)
45          {
46             out << line << '\n';
47          });
48 }
49 void printLargeDigit(int i, std::ostream &out)
50 {
51    std::vector<std::string> number = digitsvector.at(i);
52
53             std::for_each(std::begin(number), std::end(number), [&out](auto line) {
54                out << line << '\n';
55             });
56 }
```

6.6 sevensegment.h

```
1  #ifndef SEVENSEGMENT_H_
2  #define SEVENSEGMENT_H_
3  #include <iosfwd>
4  #include <vector>
5  #include <string>
6
7     const std::vector<std::vector<std::string>> digitsvector{
8        std::vector<std::string>{" - ", "| |", " ", "| |", " - "},
9        std::vector<std::string>{" ", "  | ", " ", "  | ", " "},
10       std::vector<std::string>{" - ", "  |", " - ", "|  ", " - "},
11       std::vector<std::string>{" - ", "  |", " - ", "  |", " - "},
12       std::vector<std::string>{" ", "| |", " - ", "  |", " "},
13       std::vector<std::string>{" - ", "|  ", " - ", "  |", " - "},
```

```
14          std::vector<std::string>{" - ", "| ", " - ", "| |", " - "},
15          std::vector<std::string>{" - ", " |", " ", " |", " "},
16          std::vector<std::string>{" - ", "| |", " - ", "| |", " - "},
17          std::vector<std::string>{" - ", "| |", " - ", " |", " - "}
18      };
19
20      const std::vector<std::string> errorvector
21      {
22
23          {" -         "},
24          {"|          "},
25          {" - - - - - "},
26          {"| | | | || "},
27          {" -     -   "}
28      };
29
30      const std::vector<std::string> minusvector
31          {
32          {" "},
33          {" "},
34          {"-"},
35          {" "},
36          {" "}
37          };
38
39  void printLargeDigit(int i, std::ostream &out);
40  void printLargeNumber(int i, std::ostream &out);
41  void printErrorMessage(std::ostream &out);
42
43  #endif /* SEVENSEGMENT_H_ */
```

7 Testat 2

7.1 word.h

```
#ifndef WORD_H_
#define WORD_H_

#include <algorithm>
#include <iosfwd>
#include <string>
#include <cctype>
#include <vector>

namespace word {
   class Word {
      std::string word {};
   public:
      Word() noexcept = default;
      explicit Word(std::string const &s);
      std::istream & read(std::istream & is);
      std::ostream & print(std::ostream & os) const;
      bool operator==(Word const & rhs) const;
      bool operator<(Word const & rhs) const;
   private:
      bool static isValidWord(std::string s);
   };

   std::istream & operator>>(std::istream & is, Word & word);
   std::ostream & operator<<(std::ostream & os, Word const & word);
   std::ostream & operator<<(std::ostream & os, std::vector<Word> const & l);

   inline bool operator>(Word const & lhs, Word const & rhs) {
      return rhs < lhs;
   }
   inline bool operator>=(Word const & lhs, Word const & rhs) {
      return !(lhs < rhs);
   }
   inline bool operator<=(Word const & lhs, Word const & rhs) {
      return !(rhs < lhs);
   }
   inline bool operator!=(Word const & lhs, Word const & rhs) {
      return !(rhs == lhs);
   }
}

#endif /* WORD_H_ */
```

7.2 word.cpp

```
#include "word.h"

#include <algorithm>

#include <cctype>
#include <iosfwd>
#include <stdexcept>
#include <string>
#include <iterator>

```

```
11 namespace word {
12 Word::Word(std::string const &s) {
13    if (s.empty()) {
14       throw std::invalid_argument("Word cannot be initialized with an empty string!");
15    }
16    if (!isValidWord(s)) {
17       throw std::invalid_argument("Word must contain only alphanumeric chars!");
18    }
19    word = s;
20 }
21
22 bool Word::isValidWord(std::string s) {
23    return std::all_of(s.begin(), s.end(), [](char c) {return std::isalpha(c);});
24 }
25
26 std::istream & Word::read(std::istream & is) {
27    while (!isalpha(is.peek()) && !is.eof()) {
28       is.ignore();
29    }
30    std::string temp { };
31    while (isalpha(is.peek()) && !is.eof()) {
32       temp += is.get();
33    }
34    if (!temp.empty()) {
35       if (isValidWord(temp)) {
36          word.clear();
37          word = temp;
38          is.clear();
39       } else {
40          is.setstate(std::ios::failbit);
41       }
42    }
43    return is;
44 }
45 bool Word::operator==(Word const & rhs) const {
46
47    return std::equal(word.begin(), word.end(), rhs.word.begin(), rhs.word.end(),
       [](auto left, auto right)
48       {
49          return tolower(left) == tolower(right);
50       });
51 }
52 bool Word::operator<(Word const & rhs) const {
53    return std::lexicographical_compare(word.begin(), word.end(), rhs.word.begin(),
       rhs.word.end(), [](char a, char b) {
54       return tolower(a) < tolower(b);
55    });
56 }
57
58 std::ostream & Word::print(std::ostream & os) const {
59    os << word;
60    return os;
61 }
62
63 std::istream & operator>>(std::istream & is, Word & word) {
64    return word.read(is);
65 }
66
67 std::ostream & operator<<(std::ostream & os, Word const & word) {
68    return word.print(os);
69 }
70 }
```

7.3 kwic.cpp

```cpp
#include "kwic.h"
#include "word.h"
#include <iosfwd>
#include <iterator>
#include <ostream>
#include <sstream>
#include <string>
#include <vector>
#include <algorithm>

using namespace word;

namespace word {
std::ostream & operator<<(std::ostream & os, std::vector<Word> const & l) {
   std::copy(std::begin(l), std::end(l), std::ostream_iterator<Word>{os, " "});
   return os;
}
}

namespace kwic {
   std::vector<std::vector<Word>> rotate(std::vector<std::vector<Word>> const &
       input) {
      std::vector<std::vector<Word>> rotated {};
      std::for_each(begin(input), end(input), [&](std::vector<Word> toRotate) {
         for (auto it = begin(toRotate); it != end(toRotate); it++) {
            std::vector<Word> rotatedLine { toRotate.size() };
            std::rotate_copy(
               std::begin(toRotate), it, std::end(toRotate),
               std::begin(rotatedLine));
            rotated.push_back(rotatedLine);
         }
      });
      std::sort(rotated.begin(), rotated.end());
      return rotated;
   }
   void kwic(std::istream & in, std::ostream & out) {
      std::vector<std::vector<Word>> lines {};
      std::string inputLine {};
      while(std::getline(in, inputLine)) {
         std::istringstream lineStream { inputLine };
         std::istream_iterator<Word> wordIterator { lineStream }, eof {};
         lines.push_back(std::vector<Word> { wordIterator, eof });
      }
      std::vector<std::vector<Word>> rotatedLines = rotate(lines);

      std::copy(
         std::begin(rotatedLines),
         std::end(rotatedLines),
         std::ostream_iterator<std::vector<Word>>(out, "\n"));

//    std::for_each(begin(rotatedLines), end(rotatedLines), [&out](std::vector<Word>
    line) {
//       //out << line << '\n';
//    });
   }
}
```

7.4 kwic.h

```
#ifndef KWIC_H_
#define KWIC_H_
#include <iosfwd>

namespace kwic {
   void kwic(std::istream & in, std::ostream & out);
}

#endif /* KWIC_H_ */
```

8 Testat 3

8.1 indexableSet.h

```h
#ifndef SRC_INDEXABLESET_H_
#define SRC_INDEXABLESET_H_

#include <functional>
#include <set>
#include <iterator>
#include <stdexcept>

template<typename T, typename COMPARE = std::less<T>>
struct indexableSet: std::set<T, COMPARE> {
   using container = std::set<T,COMPARE>;
   using container::container;
   using const_reference = typename container::const_reference;

public:
   const_reference at(int index) const {
      if (index < 0) {
         index += this->size();
      }
      if (static_cast<unsigned>(index) >= this->size()) {
         throw std::out_of_range { "index out of range!" };
      }

      return *std::next(this->begin(), index);
   }

   const_reference operator[](int index) const {
      return at(index);
   }

   const_reference front() const {
      return at(0);
   }

   const_reference back() const {
      return at(-1);
   }
};
#endif /* SRC_INDEXABLESET_H_ */
```

8.2 Test.cpp

```cpp
#include "cute.h"
#include "ide_listener.h"
#include "xml_listener.h"
#include "cute_runner.h"
#include "indexableSet.h"

#include <functional>
#include <iterator>
#include <string>
#include <vector>
#include <algorithm>
#include <cctype>

```

```
14 void emptyContainerTest() {
15    indexableSet<int, std::less<int>> empty { };
16    ASSERT_EQUAL(0, empty.size());
17 }
18
19 void resultingSizeTest() {
20    std::vector<std::string> input { "Banana", "Apple", "Strawberry", "Ananas" };
21    indexableSet<std::string> const fruits { std::begin(input), std::end(input) };
22    ASSERT_EQUAL(4, fruits.size());
23 }
24
25 void ascendingOrderRangeConstructorTest() {
26    indexableSet<std::string> fruits { "Banana", "Apple", "Strawberry", "Ananas" };
27    std::vector<std::string> expected { "Ananas", "Apple", "Banana", "Strawberry" };
28    std::vector<std::string> actual { std::begin(fruits), std::end(fruits) };
29    ASSERT_EQUAL(expected, actual);
30 }
31
32 void descendingOrderRangeConstructorTest() {
33    indexableSet<std::string> fruits { "Banana", "Apple", "Strawberry", "Ananas" };
34    std::vector<std::string> expected { "Strawberry", "Banana", "Apple", "Ananas" };
35    std::vector<std::string> actual { std::rbegin(fruits), std::rend(fruits) };
36    ASSERT_EQUAL(expected, actual);
37 }
38
39 void copyConstructorTest() {
40    indexableSet<std::string> const fruits { "Banana", "Apple", "Strawberry", "Ananas"
       };
41    indexableSet<std::string> const copy { fruits };
42    ASSERT_EQUAL(4, copy.size());
43 }
44
45 void atTooLargeIndex() {
46    indexableSet<std::string> const fruits { "Banana", "Apple", "Strawberry", "Ananas"
       };
47    ASSERT_THROWS(fruits.at(4), std::out_of_range);
48 }
49
50 void atTooSmallIndex() {
51    indexableSet<std::string> const fruits { "Banana", "Apple", "Strawberry", "Ananas"
       };
52    ASSERT_THROWS(fruits.at(-8), std::out_of_range);
53 }
54
55 void initializerConstructorTest() {
56    indexableSet<std::string> const fruits { "Banana", "Apple", "Strawberry", "Ananas"
       };
57    ASSERT_EQUAL(4, fruits.size());
58 }
59
60 void atPositiveIndicesTest() {
61    indexableSet<std::string> const fruits { "Banana", "Apple", "Strawberry", "Ananas"
       };
62    ASSERT_EQUAL("Strawberry", fruits.at(3));
63 }
64
65 void atNegativeIndicesTest() {
66    indexableSet<std::string> const fruits { "Banana", "Apple", "Strawberry", "Ananas"
       };
67    ASSERT_EQUAL("Banana", fruits.at(-2));
68 }
69
```

```
70  void frontEmptyExceptionTest() {
71     indexableSet<std::string> const empty { };
72     ASSERT_THROWS(empty.front(), std::out_of_range);
73  }
74
75  void frontNotEmptyTest() {
76     indexableSet<std::string> const fruits { "Banana", "Apple", "Strawberry", "Ananas"
           };
77     ASSERT_EQUAL("Ananas", fruits.front());
78  }
79
80  void backEmptyExceptionTest() {
81     indexableSet<std::string> const empty { };
82     ASSERT_THROWS(empty.back(), std::out_of_range);
83  }
84
85  void backNotEmptyTest() {
86     indexableSet<std::string> const fruits { "Banana", "Apple", "Strawberry", "Ananas"
           };
87     ASSERT_EQUAL("Strawberry", fruits.back());
88  }
89
90  void iteratorOrderTest() {
91     indexableSet<std::string, std::greater<>> fruits { "Banana", "Apple",
           "Strawberry", "Ananas" };
92     std::vector<std::string> expected { "Strawberry", "Banana", "Apple", "Ananas" };
93     ASSERT_EQUAL_RANGES(std::begin(expected), std::end(expected), std::begin(fruits),
           std::end(fruits));
94  }
95
96  void returningByIndexAndValueTest() {
97     indexableSet<int> const values { 15 };
98     ASSERT_EQUAL(&values.at(0), &values[0]);
99  }
100
101 void FrontBackReferenceTest() {
102    indexableSet<int> const values { 15 };
103    ASSERT_EQUAL(&values.front(), &values.back());
104 }
105
106 struct caselessCompare {
107    bool operator()(std::string const &left, std::string const &right) const {
108       return std::lexicographical_compare(left.begin(), left.end(), right.begin(),
              right.end(), [](char c1, char c2)
109       {
110          return std::tolower(c1) < std::tolower(c2);
111       });
112    }
113 };
114
115 void caselessComparatorTest1() {
116    indexableSet<std::string, caselessCompare> set { "a", "B", "c" };
117    ASSERT_EQUAL("a", set[0]);
118 }
119
120 void caselessComparatorTest2() {
121    indexableSet<std::string, caselessCompare> set { "A", "B", "c" };
122    ASSERT_EQUAL("B", set[1]);
123 }
124
125 void caselessComparatorTest3() {
126    indexableSet<std::string, caselessCompare> set { "A", "B", "c" };
```

```
127     ASSERT_EQUAL("c", set[2]);
128 }
129
130 void caselessComparatorTest4() {
131     indexableSet<std::string, caselessCompare> set {"A", "a"};
132     ASSERT_EQUAL(1, set.size());
133 }
134
135 bool runAllTests(int argc, char const *argv[]) {
136     cute::suite s { };
137     //TODO add your test here
138     s.push_back(CUTE(emptyContainerTest));
139     s.push_back(CUTE(copyConstructorTest));
140     s.push_back(CUTE(resultingSizeTest));
141     s.push_back(CUTE(ascendingOrderRangeConstructorTest));
142     s.push_back(CUTE(atTooLargeIndex));
143     s.push_back(CUTE(atTooSmallIndex));
144     s.push_back(CUTE(initializerConstructorTest));
145     s.push_back(CUTE(descendingOrderRangeConstructorTest));
146     s.push_back(CUTE(frontNotEmptyTest));
147     s.push_back(CUTE(frontEmptyExceptionTest));
148     s.push_back(CUTE(backEmptyExceptionTest));
149     s.push_back(CUTE(backNotEmptyTest));
150     s.push_back(CUTE(atPositiveIndicesTest));
151     s.push_back(CUTE(atNegativeIndicesTest));
152     s.push_back(CUTE(iteratorOrderTest));
153     s.push_back(CUTE(returningByIndexAndValueTest));
154     s.push_back(CUTE(FrontBackReferenceTest));
155     s.push_back(CUTE(caselessComparatorTest1));
156     s.push_back(CUTE(caselessComparatorTest2));
157     s.push_back(CUTE(caselessComparatorTest3));
158     s.push_back(CUTE(caselessComparatorTest4));
159
160     cute::xml_file_opener xmlfile(argc, argv);
161     cute::xml_listener<cute::ide_listener<>> lis(xmlfile.out);
162     auto runner = cute::makeRunner(lis, argc, argv);
163     bool success = runner(s, "AllTests");
164     return success;
165 }
166
167 int main(int argc, char const *argv[]) {
168     return runAllTests(argc, argv) ? EXIT_SUCCESS : EXIT_FAILURE;
169 }
```