Programmiersprachen
Sommersemester 2015

Aufgabensammlung 3

Die Aufgaben werden am **18. Mai** in der Übung bewertet. Diese Aufgabensammlung beschäftigt sich mit den Grundlagen zu Templates, der Standard Template Library (STL), einem Teil der Standard C++ Library und dem Überladen von Operatoren.
Es gelten die Ausführungshinweise der vorherigen Aufgabenblätter (`const`-Korrektheit, Member-initialization-list, Header/Source, Tests, ...). Nutzen Sie neben dem Vorlesungsskript ausschließlich aktuelle Fachliteratur oder Online-Referenzen, z.B.

- Stroustrup, B.: Einführung in die Programmierung mit C++ (2010)
- http://en.cppreference.com/
- http://www.cplusplus.com/

Bei Fragen und Anmerkungen schreiben Sie bitte ein Email an andreas.bernstein@uni-weimar.de .

Aufgabe 3.1

Erstellen Sie für dieses Aufgabenblatt ein neues CMake-Projekt, wie im ersten Aufgabenblatt beschrieben. Legen Sie dazu einen Ordner mit dem Namen *Aufgabenblatt3* und erstellen Sie die Datei `CMakeLists.txt`:

```
cmake_minimum_required (VERSION 2.8)
project(Aufgabenblatt3)
set(CMAKE_CXX_FLAGS "-std=c++0x")

add_executable(aufgabe1 aufgabe1.cpp)
```

Vergessen Sie nicht die Zeile `set(CMAKE_CXX_FLAGS "-std=c++0x")`! Kopieren Sie außerdem die Datei catch.hpp in den Ordner und kompilieren Sie folgendes Programm.

```
#include <cstdlib>   // std::rand()
#include <vector>    // std::vector<>
#include <list>      // std::list<>
#include <iostream>  // std::cout
#include <iterator>  // std::ostream_iterator<>
#include <algorithm> // std::reverse, std::generate
```

```cpp
int main()
{
  std::vector<int> v0(10);

  for (std::vector<int>::iterator i=v0.begin();
       i!=v0.end();
       ++i) {
    *i = std::rand();
  }
  std::copy(std::begin(v0), std::end(v0),
            std::ostream_iterator<int>(std::cout, "\n"));

  std::list<int> l0(v0.size());
  std::copy(std::begin(v0), std::end(v0), std::begin(l0));

  std::list<int> l1(std::begin(l0), std::end(l0));
  std::reverse(std::begin(l1), std::end(l1));
  std::copy(std::begin(l1), std::end(l1),
            std::ostream_iterator<int>(std::cout, "\n"));

  l1.sort();
  std::copy(l1.begin(), l1.end(),
            std::ostream_iterator<int>(std::cout, "\n"));

  std::generate(std::begin(v0), std::end(v0), std::rand);
  std::copy(v0.rbegin(), v0.rend(),
            std::ostream_iterator<int>(std::cout, "\n"));

  return 0;
}
```

~~Analysieren Sie das Programm und erläutern Sie dessen Funktionsweise.~~ Benennen Sie die *Typen* aller Variablen. **[5 Punkte]**

Aufgabe 3.2

~~Erstellen Sie ein neues Programm mit dem Namen `aufgabe2bis4.cpp` und fügen Sie es zur Datei `CMakeLists.txt` hinzu.~~ Instanziieren Sie eine ~~`std::list`~~ mit ~~`unsigned int`~~ und füllen Sie diese mit ~~100 Zufallszahlen zwischen 0 und 50.~~ ~~Erzeugen Sie einen `std::vector` und kopieren Sie mit `std::copy` die Elemente der Liste in den `std::vector`.~~ **[5 Punkte]**

Rückgabe noch etwas ungünstig…

Aufgabe 3.3

Bestimmen Sie, wieviele unterschiedliche Zahlen in der `std::list` aus Aufgabe 3.2 sind und geben Sie die Zahlen von 0 bis 50 aus, die nicht in der Liste sind.

[10 Punkte]

Aufgabe 3.4

Ermitteln Sie die Häufigkeit jeder Zahl in der `std::list` aus Aufgabe 3.2. Erklären Sie, warum sich `std::map` für dieses Problem anbietet? Geben Sie die Häufigkeit aller Zahlen in der Form *Zahl : Häufigkeit* auf der Konsole aus.

[10 Punkte]

Aufgabe 3.5

Erstellen Sie ein neues Programm `aufgabe5.cpp`.

```
#define CATCH_CONFIG_RUNNER
#include "catch.hpp"
#include <cmath>
#include <algorithm>

TEST_CASE("describe_factorial", "[aufgabe3]")
{
  // ihre Loesung :
  // ...

  REQUIRE(std::all_of(v.begin(), v.end(), is_even));
}

int main(int argc, char* argv[])
{
  return Catch::Session().run(argc, argv);
}
```

Füllen Sie einen `std::vector` von `unsigned int` mit 100 Zufallszahlen zwischen 0 und 50. Entfernen Sie alle ungeraden Zahlen. Verwenden Sie dafür einen passenden STL-Algorithmus.
Testen Sie danach mit `std::all_of` aus `<algorithm>`, ob alle Elemente im vector gerade sind. Schreiben Sie dafür eine Hilfsfunktion `is_even`.

[10 Punkte]

Aufgabe 3.6

Implementieren Sie ein Programm `aufgabe6`, welches den Namen und die dazugehörige Adresse einer beliebigen Anzahl von Personen von der Konsole einliest und in einer `std::map` speichert. Suchen Sie im Anschluss eine Person und geben Sie deren Adresse auf der Konsole aus. Erklären Sie Vorteile und Einschränkungen bei der Verwendung einer Map. **[10 Punkte]**

Aufgabe 3.7

vector, array, list, queue, deque, stack
lineare Anordnung
eindeutige Schlüssel
map(multi), set(multi)
balancierte Binärbäume → logarithmisch

Erklären Sie die Unterschiede zwischen *sequentiellen* und *assoziativen* Containern. Wählen Sie für folgende Anwendungsfälle einen Container und erklären Sie ihre Wahl:

- Speichern der Punkte eines Polygons — vector —> lineare Verarbeitung (abarbeitung), Key nicht erforderlich
- Zuordnung von Farbnamen und entsprechenden RGB-Werten — map —> Farbnamen können direkt als Key zugegriffen und gesucht werden
- FIFO-Warteschlange von Druckaufträgen — queue (Warteschlange) —> keine Itteratoren, neues Element am Ende

[5 Punkte]

Aufgabe 3.8

Erstellen Sie ein neues Programm `aufgabe8`. Am besten verwenden Sie das Codefragment aus Aufgabe 3.5.
Objekte der Klasse `Circle` sollen in einem STL-Container gespeichert und der Radiusgröße nach sortiert werden. Um Objekte in einem Container sortieren zu können, müssen Sie vergleichbar sein. Überladen Sie die Operatoren `operator<`, `operator>` und `operator==` für Objekte vom Typ `Circle`, füllen Sie einen Container mit Objekten vom Typ `Circle` und sortieren Sie diesen.

```
REQUIRE(std::is_sorted_of(container.begin(), container.end()));
```

[10 Punkte]

Aufgabe 3.9

Implementieren Sie ein Funktionstemplate `swap`, zum Vertauschen zweier Objekte gleichen Typs. Erklären Sie die Wahl der Parameterübergabe. Schreiben Sie einen `TEST_CASE` dafür. **[8 Punkte]**

Aufgabe 3.10

Mit Konkatenation bezeichnet man das Zusammenfügen zweier sequentieller Container ohne die Reihenfolge der Elemente zu verändern.

Definieren Sie dazu ein Funktionstemplate `concatenate`, welches den Inhalt zweier STL-Container in einem Dritten zusammenführt.
Diese Funktion hat als Parameter zwei Container die zusammengeführt werden sollen. Die Funktion gibt einen neuen Container zurück, in welchem der Inhalt des erstens gefolgt vom Inhalt des zweiten steht.
Testen Sie ihre Funktion für verschiedene Container mit `TEST_CASE`.

[5 Punkte]

Aufgabe 3.11

Schreiben Sie ein Funktionstemplate `filter`, welches als ersten Parameter einen sequentiellen Container und als zweiten Parameter ein Prädikat hat. Die Funktion soll einen neuen Container gleichen Typs zurückgeben. Dieser soll nur Werte enthalten, die das Prädikat erfüllen. Verwenden kann man die Funktion dann wie im folgenden Beispiel.

```
std::vector<int> v{1,2,3,4,5,6};
std::vector<int> alleven = filter(v, is_odd);
```

Am besten verwenden Sie das Codefragment aus Aufgabe 3.5 um ein Testprogramm zu schreiben.

[10 Punkte]

Aufgabe 3.12

Erklären Sie, warum es bei folgendem Programmsegment zu Problemen kommen kann:

```
std::map<string,int> matrikelnummern;
//Hinzufueggen von vielen Studenten
matrikelnummern["Max Mustermann"] = 12345;
matrikelnummern["Erika Mustermann"] = 23523;
//...
exmatrikulation(matrikelnummer["Fred Fuchs"]);
```

überschreibt vorhandene. besser per std::insert. liefert false wenn bereits vorhanden

fürht dazu, dass wen Eintrag nicht vorhanden ist, wird dieser erstellt. Value wird Standardkonstruiert

Wie sollten Sie vorgehen, um dieses Problem zu vermeiden? **[2 Punkte]**

Schlüsselwörter 3.13

- Freispeicher/Freestore — abspeicherung dynamisch erzegter Variablen, muss explizt freigegeben werden, zugriff über zeiger, große Datenstrukturen und Container
- Template — generische Programmierung, Funktionen unabhängig von Datentypen
- STL — Standard Template Library - hocheffiziente Sammlung Template Klassen
- Sequenzielle Container — lineare Speicherung, behalten Reihenfolge bei.
- Assoziative Container — nach Schlüsseln sortiert

- Iteratoren nützliche Objekte ähnlich wie Pointer, zum Zugriff und das Bewegen über einen bestimmten Bereich
- STL-Algorithmen effiziente Sammlung von Funktionen auf Template Klassen, arbeiten mit Itteratoren
- Komplexität `std::vector` (siehe Vorlesung)
- Komplexität `std::list`
- Komplexität `std::set`
- Komplexität `std::map`

[10 Punkte]

Bei Fragen und Anmerkungen schreiben Sie bitte ein Email an andreas.bernstein@uni-weimar.de .

vector
Einfügen und löschen am Ende in O(1) - effizient
Einfügen und löschen überall in O(N)
Suchen in O(N)

list
Einfügen und löschen überall in O(1) - effizient
Suchen in O(N) (ohne Itterator!!)

set
Einfügen und Löschen überall in O(logN)
Suchen in O(logN)

map
Einfügen und Löschen überall in O(logN)
Suchen in O(logN)