technische universität dortmund

fakultät für informatik

Jurij Kuzmic

Dortmund, den 16. Januar 2020

Übung zur Vorlesung EidP (WS 2019/20)

Blatt 11

Block grün

Es können 4 Punkte und 3 Bonuspunkte erreicht werden.

Abgabedatum: 23. Januar 2020, 23:59 Uhr

Hinweise

- Bitte beachten Sie die aktuellen Hinweise unter

  `https://ls11-www.cs.tu-dortmund.de/teaching/ep1920uebung/`

- Für die Abgabe sind die jeweils genannten Dateien zu erstellen.
- Stellen Sie sicher, dass alle von Ihnen abgegebene Dateien reine Textdateien im UTF-8-Format sind.
- Für die Kompilierung des Programms muss der C++14-Standard aktiviert sein. Dies kann im Referenzkompiler GCC 6.3 durch den Schalter `-std=c++14` sichergestellt werden. Es sollen zudem die Parameter `-pedantic` und `-Werror` genutzt werden. Der Befehl zum Kompilieren soll somit wie folgt aussehen:

  ```
  g++-6 -pedantic -Werror -std=c++14 Aufgabe.cpp -o Aufgabe
  ```

- Für die Programmieraufgaben kopieren Sie immer die Ergebnisse als Block-Kommentar an das Ende der Datei, welche das jeweilige Hauptprogramm enthält.
- **Achtung:** Dies ist das letzte Aufgabenblatt im grünen Block und insgesamt das letzte Übungsblatt zur Vorlesung. Zum Bestehen dieses Blocks sind 8 Punkte erforderlich.

Aufgaben

Aufgabe 1: Grundlagen (1 Punkt)

Legen Sie für Ihre Antworten eine Text-Datei `Aufgabe_11_1.txt` an.

a) Skizzieren Sie mit den Schlüsselwörtern `try` und `catch` den typischen Ausnahmebehandlungsblock. (0.2 Punkte)

b) Welchen Vorteil bietet eine Ausnahmebehandlung gegenüber dem Zurückliefern eines Wertes, welcher einen Fehler darstellen soll? (0.2 Punkte)

c) Nennen Sie zwei Situationen, in denen eine Ausnahmebehandlung dem Beenden eines Programms vorzuziehen ist. (0.2 Punkte)

d) Wie sieht der `catch`-Block aus, der jede beliebige im vorherigen `try`-Block geworfene Ausnahme fängt? (0.2 Punkte)

e) Welche Voraussetzungen müssen vorliegen, damit eine Exception vom Typ `E` im *catch*-Block mit Parametertyp `H` gefangen wird? (0.2 Punkte)

Aufgabe 2: Ausnahmebehandlung (3 Punkte)

Legen Sie für Ihre Antworten die Dateien `Aufgabe_11_2a.txt`, `Aufgabe_11_2b.cpp` bzw. `Aufgabe_11_2c.cpp` an.

a) Um Polymorphie beim Fangen von Ausnahmen zu ermöglichen, haben Sie in der Vorlesung das Konzept *Throw by value, catch by reference* kennengelernt. Erklären Sie, warum Polymorphie beim Fangen von Ausnahmen ein sinnvolles Konzept ist. Welche Probleme können beim *catch by value* entstehen? Verdeutlichen Sie das Problem mit einem Beispiel, bestehend aus zwei Ausnahme-Klassen, einer `main` Funktion und einer Funktion, welche die Ausnahme mit `throw` auslöst.

Hinweis: Sie brauchen keine .cpp-Datei anzulegen; schreiben Sie Ihre Implementierung stattdessen in die Datei `Aufgabe_11_2a.txt`. Die Ausnahme-Klassen brauchen nur eine print-Funktion, um eine Ausgabe auf der Konsole zu ermöglichen. Um das Beispiel möglichst klein zu halten, brauchen Sie keine Konstruktoren zu implementieren. (1 Punkt)

b) Betrachten Sie das unten stehende Programm:

```
1  /*** Aufgabe_11_2b.cpp ***/
2  #include <iostream>
3  using namespace std;
4
5  int PosProdInt(unsigned int size, int const array[]) {
6    int posProd = 1;
7    for (unsigned int i = 0; i < size; ++i) {
8      if (array[i] > 0) posProd *= array[i];
9    }
10   return posProd;
11 }
12
13 double PosProdDbl(unsigned int size, const double array[]) {
14   double posProd = 1;
15   for (unsigned int i = 0; i < size; ++i) {
16     if (array[i] > 0) posProd *= array[i];
17   }
18   return posProd;
19 }
20
21 int main() {
22   int const iA[] = {1, -1, 7, -6, -3, 0};
23   double const dA[] = {1.3, -3.2, 0.1, -2.7, 0.4};
24   cout << PosProdInt(6, iA) << ' ' << PosProdDbl(5, dA) << endl;
25   return 0;
26 }
```

```
27  /*** Ende Aufgabe_11_2b.cpp ***/
```

Offensichtlich existieren hier die zwei Funktionen `PosProdInt` und `PosProdDbl`, die zwar das gleiche Verhalten haben, jedoch auf verschiedenen Datentypen arbeiten.
Kommentieren und implementieren Sie mittels der Schablonentechnik aus C++ eine Funktionsschablone `PosProd`, die das Produkt aller positiven Elemente im übergebenen Feld zurückgibt und für jeden eingebauten numerischen Datentyp instantiiert werden kann. Ändern Sie das Hauptprogramm entsprechend ab, so dass es die neue Funktion verwendet. Kopieren Sie das berechnete Ergebnis als Blockkommentar ans Ende der Datei. (1 Punkt)

c) Ändern Sie die Funktionsschablone so, dass sie als Arraygröße eine Variable vom Datentyp `int` erhält, aber dann negative Arraygrößen und Arrays mit der Größe null durch den Mechanismus der Ausnahmebehandlung mit `try`, `throw` und `catch` behandelt. Verwenden Sie für jede Fehlerart eine eigene Fehlerklasse und einen eigenen `catch`-Block. Kopieren Sie das berechnete Ergebnis als Blockkommentar ans Ende der Datei. (1 Punkt)

Aufgabe 3: Kryptografie durch Rotation (3 Bonuspunkte)

Hinweis: Die Bearbeitung dieser Aufgabe ist optional. Mit dieser Aufgabe können Sie, wie in der Probeklausur, bis zu 3 Punkte auch für zurückliegende Aufgabenblätter beziehungsweise Blöcke erwerben.

Die Caesar-Verschlüsselung, die auf Gaius Julius Caesar zurückgeht, verwendet für das Ver- und Entschlüsseln eine Rotation des Alphabets um n Positionen. Es gibt 25 Möglichkeiten, denn eine Rotation um 26 Positionen verändert das Alphabet nicht. Bei Beschränkung auf die 26 Großbuchstaben kann der Zielbuchstabe einer Caesar-Verschlüsselung wie folgt berechnet werden:

```
1  cout << (char)( 'A' + ( c - 'A' + n ) % 26 );
```

Dabei ist c der Buchstabe aus dem Klartext und n die Anzahl der Zeichen, um die rotiert wird. Die Implementierung der Caesar-Verschlüsselung ist auf der Website der Übung zu finden. Laden Sie für Ihre Abgabe die Datei `Aufgabe_11_3.cpp` von der Website der Übung herunter. Schreiben Sie Ihre Antworten in die Datei `Aufgabe_11_3.cpp`.

1. Erklären Sie zunächst anhand eines Beispiels, warum die obige Berechung für die Großbuchstaben das richtige Ergebnis liefert.
2. Wie muss das n gewählt werden, damit das Ver- *und* Entschlüsseln von Großbuchstaben mit derselben Berechnung funktioniert? Wie stattdessen für Ziffern?
3. Warum können mit $n = 65$ sowohl Buchstaben als auch Ziffern ver- *und* entschlüsselt werden?
4. Erweitern Sie das Programm so, dass Kleinbuchstaben und Ziffern aus dem Klartext ebenfalls verschlüsselt werden. Für die Überprüfung auf Kleinbuchstaben und Ziffern stellt die `<cctype>`-Bibliothek unter anderem Funktionen wie `islower(char)` und `isdigit(char)` zur Verfügung.