Lösungen zu Übungsblatt 10

Ralph Gauges Ursula Rost Katja Wegner

16.01.2008

Aufgabe 1:

Schreiben Sie die Funktion zur Berechnung der Fakultät aus Aufgabe 2 vom 31.10.2007 so um, dass diese eine Ausnahme (Exception) erzeugt, wenn das übergebene Argument nicht im erlaubten Wertebereich liegt. Die Funktionsdeklaration soll auch angeben, dass die Funktion diese Ausnahme auslösen kann.
Das Hauptprogramm, welches die Zahlen vom Benutzer einliest, soll diese Ausnahme abfangen und eine entsprechende Fehlermeldung generieren. Die C++ Bibliothek bietet bereits eine Ausnahme, welche sich für diesen Zweck eignen würde. Der Name der Ausnahmen-Klasse ist *invalid_argument* und sie ist in der Header Datei <stdexcept> zu finden. Sie können jedoch auch eine eigene Klasse schreiben und als Ausnahme verwenden.

```
1 #include <stdexcept>
2 #include <sstream>
3
4 ...
5
6 using namespace std;
7
8 double fakultaet(unsigned int n) throw(invalid_argument)
9 {
10   if(n<1 || n>20)
11   {
12     ostringstream os;
13     os  << n << " liegt nicht zwischen 1 und 20.";
14     throw invalid_argument(os.str());
15   }
```

```
16   double result=1.0;
17   unsigned int x;
18   for(x=2; x<=n; ++x)
19   {
20     result*=x;
21   }
22   return result;
23 }
24
25 int main()
26 {
27   cout << "Bitte geben Sie eine zweistellige Zahl ein:";
28   char zahl[3];
29   cin.getline(zahl,3);
30   if(istEineZahl(zahl[0],zahl[1])==true)
31   {
32     unsigned int zahlenWert=umwandelnInZahl(zahl[0],zahl[1]);
33     try
34     {
35       cout <<  "Fakultaet von " << zahlenWert;
36       cout << " ist " << fakultaet(zahlenWert) << ".\n";
37     }
38     catch(invalid_argument e)
39     {
40       cerr << "ERROR: " << e.what() << endl;
41     }
42   }
43   else
44   {
45     cerr << "Es handelt sich bei der Eingabe nicht um eine Zahl.\n";
46   }
47   return 0;
48 }
```

Die Funktionen *istEineZahl* und *umwandelnInZahl* bleiben unverändert. Die Funktion *fakultaet* überprüft in dieser Fassung, ob die übergebenen Zahl im erlaubten Wertebereich liegt und falls nicht, wird eine Ausnahem (Exception) (*invalid_argument*) ausgelöst. Die Fehlermeldung für die Exception wird dabei über eine Instanz von *std::ostringstream* generiert, da sich so Zahlen einfach in Zeichenketten umwandeln lassen. In der *main* Funktion wird die Berechnung der Fakultät nun innerhalb eines **try** Blocks ausgeführt und mit einer anschließenden **catch** Anweisung wird die *invalid_argument* Exception abgefangen und eine Fehlermeldung ausgegeben.

Aufgabe 2:

Schreiben Sie die Klasse für Komplexe Zahlen aus Aufgabe 3 vom 28.11.2007 so um, dass alle Methoden, in denen eine Division durch die Zahl 0 stattfinden könnte, eine Ausnahme (Exception) generieren. Die Methoden, in denen dies auftreten kann, sollen diese Ausnahme bei der Deklaration angeben.
Schreiben Sie anschließend ein kurzes Programm, welches überprüft, ob die Ausnahme auch tatsächlich generiert wird.

——— Änderungen an ComplexNumber.h ———————-

```
1 #include "DivisionByZero.h"
2
3 class ComplexNumber
4 {
5     ...
6     public:
7     ...
8         ComplexNumber& operator/=(const ComplexNumber& right)
9                 throw(division_by_zero);
10
11        const ComplexNumber operator/(const ComplexNumber& right)
12                const throw(division_by_zero);
13    ...
14 };
```

Lediglich die Methoden für die Division und die Zuweisung mit Division enthalten in dieser Implementierung Code, der zu einer Division durch 0 führen kann. Dementsprechend wurden die Deklarationen dieser zwei Methoden dahingehend erweitert, dass die Ausnahmen, welche von den Methoden ausgelöst werden können angegeben sind.

——— Änderungen an ComplexNumber.cpp ———————-

```
1 ...
2
3 ComplexNumber& ComplexNumber::operator/=(const ComplexNumber& right)
4   throw(division_by_zero)
5 {
6     double divisor = right.realPart*right.realPart;
7     divisor += right.imagPart*right.imagPart;
8     double temp1 = this->realPart*right.realPart;
9     temp1 += this->imagPart*right.imagPart;
10    double temp2 = this->imagPart*right.realPart;
```

```
11      temp2 -= this->realPart*right.imagPart;
12      if(divisor==0.0)
13      {
14          throw division_by_zero();
15      }
16      this->realPart=temp1/divisor;
17      this->imagPart=temp2/divisor;
18      return (*this);
19  }
20
21  ...
22
23  const ComplexNumber ComplexNumber::operator/(const ComplexNumber& right)
24     const throw(division_by_zero)
25  {
26      ComplexNumber result=(*this);
27      try
28      {
29        result/=right;
30      }
31      catch(division_by_zero)
32      {
33          throw;
34      }
35      return result;
36  }
37  ...
```

Der Code für die Zuweisung mit Division enthält eine Division, d.h. an dieser Stelle müssen wir testen, ob der Wert des Divisors 0 ist. Ist dies der Fall, lösen wir eine *division_by_zero* Ausnahme aus.
Der Code für die einfache Division benutzt zur Berechnung die Zuweisung mit Division. Die Ausnahme, welche von dieser ausgelöst werden kann, wird nun abgefangen. Da es aber keine Möglichkeit gibt, den Fehler zu beheben, wird die Ausnahme im **catch** Block erneut ausgelöst. Das Abfangen der Ausnahme und das erneute Auslösen ist im Grunde unnötig und dient hier nur dazu den Umgang mit Ausnahmen zu veranschaulichen.
Der Code für die Ausnahme selbst ist sehr einfach. Es wird lediglich eine Klasse *division_by_zero* von der Klasse *runtime_error* abgeleitet. Die Klasse hat einen Konstruktor, welches den Konstruktor der Oberklasse aufruft und dabei den String für die Fehlermeldung setzt. Die Klasse ist so einfach, dass es sich hier sogar angeboten hätte, die Definition auch in die Header Datei aufzunehmen.

———— DivisionByZero.h ————

```
1  // HEADER
2  #ifndef DIVISIONBYZERO_H__
3  #define DIVISIONBYZERO_H__
4
5  #include <stdexcept>
6
7  class division_by_zero : public std::runtime_error
8  {
9      public:
10         division_by_zero();
11 };
12
13 #endif // DIVISIONBYZERO
```

———— DivisionByZero.cpp ————

```
1  #include "DivisionByZero.h"
2
3  division_by_zero::division_by_zero()
4      :std::runtime_error("Division by zero."){};
```