Praxis der Programmierung

Liste, Ausnahmefehler, Template-Funktionen und -Klassen

Institut für Informatik und Computational Science
Universität Potsdam

Henning Bordihn

Stack als einfach verkettete Liste

Aufgabe 1

1. Kopieren Sie aus `/home/rlehre/W13` die Dateien `intstack.cpp` und `testintstack.cpp`. Analysieren Sie die Quellcodes.

2. Ergänzen Sie `intstack.cpp` um die Implementierungen von `top()` und `pop()`. Testen Sie mit `testintstack.cpp`.

3. Beantworten Sie die Fragen von Aufgabe 1 auf dem Übungsblatt.

Stack-Methoden

```
int Stack::top() {
   return first->value;
}
```

```
int Stack::pop() {
      int res = first->value;
      StackElement * ontop;
      ontop = first;
      first = first->next;
      delete ontop;
      return res;
}
```

Ausnahmefehler

Begriff Ausnahmefehler

- Ausnahmefehler sind Laufzeitfehler oder logische Fehler,
  die ein unerwartetes Verhalten oder einen Absturz zur Laufzeit verursachen

- Beispiele:
  - Zugriff auf Array- oder Listenelemente ausserhalb des alloziierten Bereichs
  - Zugriff auf Datenelemente von Instanzvariabeln, die auf kein Objekt zeigen
  - Aufruf von Methoden mit Instanzvariabeln, die auf kein Objekt zeigen
  - Division durch 0
  - Öffnen einer Datei, die nicht voranden ist
  - Lesen aus einer Datei, die während des Zugriffs gelöscht wird

- Ausnahmefehler = Exception

Behandlung von Ausnahmefehlern

- Ausnahmefehler können vom Programmierer *abgefangen* werden
  - $\rightsquigarrow$ kein Programmabbruch
  - $\rightsquigarrow$ kontrolliertes, vom Programmierer festgelegtes Verhalten im Ausnahmefall
  - $\rightsquigarrow$ Trennung von normalem Verhalten und Fehlerbehandlung

```
try {
   // Versuch, das Programm fehlerfrei auszufuehren
   //
   // Anweisungen, die einen Fehler ausloesen koennten
}
catch(...) {  // ... muss dort stehen!
   // Anweisungen, die ausgefuehrt werden,
   // falls ein Ausnahmefehler aufgetreten ist
}
```

Vordefinierte Ausnahmefehler

- in Klassen der Satndardbibliothek
- Basisklasse: `exception`
- mehrere abgeleitete Klassen, *z.B.*
  - `ios_base::failure` – Fehlerklasse der Stream-Klassen
  - `length_error` – maximale Größe wird überschritten
  - `out_of_range` – Zugriff mit unzulässigem Index
  - `bad_alloc` – `new` kann keinen Speicher anfordern
  - ...
- Man kann eigene Unterklassen von `exception` definieren
  $\rightsquigarrow$ Überschreiben der Methode `what()`:
  $\rightsquigarrow$ liefert einen Pointer auf einen C-String mit Fehlerinformationen

Unterscheidung von Ausnahemefehlern

- mehrere `catch`-Blöcke zu einem `try`-Block
- erst spezielle Fehler (Vererbungshierarchie beachten!), dann allgemeinere
- `catch(...)` zuletzt (Behandlung "aller weiteren" Ausnahmefehler)

```
try { // Anweisungen }
catch(MyException&) {
      // Behandlung von Fehlern eines selbst definierten Typs
}
catch(ios::failure&)
      // Behandlung von Fehlern bei Stream-Nutzung
}
catch(...) {
      // wenn sonst etwas schief geht
}
```

`throw`: Ausnahmefehler ohne Fehlerklasse erzeugen

```
void foo(int problem) {
   if (problem > 0)
      throw 0;      // erzeugt eine Exception
   // ...
}

int main() {
   try {
      foo(1);
   }
   catch(...) {
      cout << "Ein Fehler beim Aufruf von foo(int)."
   }
```

Übergabe der Fehlernummer?!

`throw`: Ausnahmefehler ohne Fehlerklasse erzeugen

```
void foo(int problem) {
   if (problem == 1)
      throw 1;       // erzeugt eine Exception
   if (problem == 2)
      throw 2;
   if (problem > 2)
      throw (char *) "message";
}

int main() {
   int n = 0;                    catch(int i)   { // Anweisungen fuer int }
   cin >> n;                     catch(char* s) { // Anweisungen fuer char* }
   try { foo(n); }               catch(...)     { // Anweisungen sonst }
}
```

Aufgabe 2

1. Ergänzen Sie in `intstack.cpp` die Implementierungen von `top()` und `pop()` um eine geeignete Exceptionbehandlung.

2. Entfernen Sie aus `testintstack.cpp` die `if`-Statements und testen Sie Ihre Fehlerbehandlung.

Template-Funktionen

Minimumfunktion und offene Typen

- Aufruf `min(x,y)` sollte für möglichst alle Datentypen funktionieren, für die eine Ordnungsrelation definiert ist

- Keine Code-Verdopplung!
  $\rightsquigarrow$ Überladen der Funktion ungünstig
  $\rightsquigarrow$ Funktion mit offenem Typ definieren

- Template-Funktionen
  - verwenden offene Typen
  - definieren eine Schablone einer Funktion, die typunabhängiges Verhalten hat
  - Typen werden beim Aufruf der Funktion festgelegt
    $\rightsquigarrow$ Compiler erzeugt aus der Schablone eine zu den Typen passende Funktion

Beispiel: Minimumfunktion

```
template <typename T>   // eine Schablone mit dem Typparameter T
T min(T a, T b) {       // T kann jetzt wie ein Typ verwendet werden

     return (a < b) ? a : b;

}

int main() {
   int i = 19;
   int j = 66;
   int a = min(i,j);  // Funktion mit T = int wird jetzt erzeugt
}
```

- Statt `typename` kann synonym auch `class` verwendet werden.
- mehrere Typparameter durch Komma getrennt, *z.B.* `<class T, class S>`

Festlegung der Zieltypen

- entweder durch Parameterübergabe
- oder durch explizite Angabe der Zieltypen:

```
template <typename T1, typename T2, typename T3>
T3 foo(T1 x, T2 y) {
   // ...
}

int main() {
   cout << foo<short,long,int>(23,11);  // Rueckgabetyp int
}
```

Aufgabe 3

Definieren Sie eine Template-Funktion `myswap()`, die die Werte der ihr übergebenen Parameter vertauscht. Der Datentyp beider Variablen ist identisch aber offen.

- Denken Sie an call-by-value!
- Verwenden Sie diesmal aber keine Pointer!
  Was bietet sich dann an?

Template-Klassen / generische Typen

Definition von Template-Klassen

```
// Schablone einer Klasse mit einem Typparameter T
template <typename T> class Cls {
   // jetzt kann T wie ein Datentyp verwendet werden, z.B.:
   T var;
   T foo(int n, T& t);
};

template <typename T> T Cls<T>::foo(int n, T& t) {
   // ...
}

int main() {
   Cls<int> instance;  // Cls<int> ist ein generischer Typ
                       // Compiler legt generischen Typ erst bei der
   // ...              // Definition eines Objekts an
}
```

Aufgabe 4

- Kopieren Sie `intstack.cpp` und `testintstack.cpp` in neue Dateien `stack.cpp` bzw. `testStack.cpp`.

- Modifizieren Sie die Kopien so, dass der abstrakte Datentyp Stack als generischer Typ definiert wird.