Zahlen in einem Stapel (Stack) speichern

Die Datenstruktur Stack

Erinnern Sie sich an Karol: Er hat jederzeit Ziegelsteine zur Hand, die er irgendwo hinlegen kann. Sie können sich vorstellen, dass er immer einen großen Rucksack voller Ziegelsteine dabei hat.

In C++ gibt es verschiedene solcher „Rucksäcke", die große Datenmengen speichern können. Man nennt sie **Datenstrukturen**. Eines der einfachsten Modelle ist ein Stapel oder Stack: Er speichert Werte quasi übereinander und man kommt immer nur an der obersten heran - so wie bei einem Stapel Papier, der sich in einer Schachtel befindet. Es ist nur das Blatt zugänglich, das ganz oben liegt.

Einen Stack für anlegen

Wenn wir solche Stacks verwenden möchten, müssen wir das dem C++-Compiler erst ankündigen. Dafür kommt ganz oben nach `#include <iostream>` eine Zeile dazu:

```
#include <stack>
```

Sie gehört nicht zum eigentlichen Hauptprogramm und wird nicht mit einem Strichpunkt abgeschlossen.

Danach geht es an der gewohnten Stelle, also nach der öffnenden geschweiften Klammer, weiter:

```
stack<int> stapel;
```

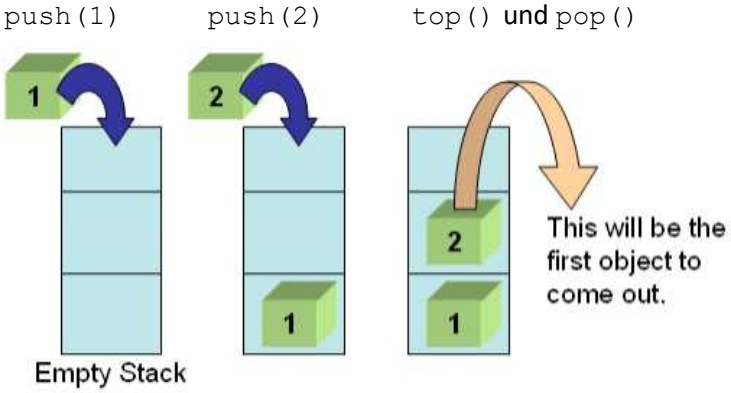
Diese Zeile erzeugt einen leeren Stack mit dem Namen `stapel`. Das können Sie sich vorstellen wie eine leere Schachtel, die dafür bereit steht, Blätter aufzunehmen.

Allerdings muss die Größe der Schachtel zu den Blättern passen. Dafür dient der Zusatz in Spitzen Klammern: Das Schlüsselwort int (kurz für *Integer*) bedeutet, dass ganze Zahlen gespeichert werden sollen. Bei einem echten Stapel würden Sie wahrscheinlich stattdessen DIN A4 oder DIN A3 schreiben.

C++ kann mit vielen verschiedenen Arten von Zahlen umgehen. Man nennt sie **Datentypen**. Am Ende dieses Skripts finden Sie eine Übersicht mit den wichtigsten Datentypen. Einstweilen genügen zwei: `int` für ganze Zahlen und `double` für Zahlen mit Nachkommastellen.

Mit einem Stack arbeiten

Auf einen Stack oder Stapel kann man nur von oben zugreifen[1].



Mit `push(...)` legt man eine Zahl auf den Stack, mit `top()` liest man die oberste Zahl und mit `pop()` nimmt man sie weg[2]. Vor diese Anweisungen, die man **Methoden** nennt, müssen Sie den Namen des Stapels und einen Punkt setzen.

In manchen Sprachen, auch in C++, kennen Stacks noch die Methode `size()`. Sie gibt an, wie viele Werte auf dem Stack liegen.

1 Bild: http://www.codeproject.com/KB/dotnet/Stacks/Stack.jpg (01.06.2014)
2 In vielen anderen Programmiersprachen heißen die Methoden anders. In Java übernimmt zum Beispiel `pop()` das Lesen und Wegnehmen in einem. `peek()` hat in Java die gleiche Funktion wie in C++ `top()`.

Beispiel mit dem oben angelegten Stack:

```
stapel.push(5);      // 5 auf den Stack legen
stapel.push(42);     // 42 auf den Stack legen

cout << "So viele Zahlen liegen auf dem Stack: " << stapel.size() << endl;

cout << "Oberste Zahl:" << stapel.top() << endl;     // oberste Zahl ausgeben

stapel.pop();                                        // oberste Zahl wegnehmen
cout << "Zahl weggenommen." << endl;

cout << "Oberste Zahl: " << stapel.top() << endl;    // oberste Zahl ausgeben

stapel.pop();                                        // oberste Zahl wegnehmen
cout << "Zahl weggenommen" << endl;

cout << "So viele Zahlen liegen auf dem Stack: " << stapel.size() << endl;
```

Stacks mit Wiederholungen (Schleifen) verwenden

Die Werte, die `top()` und `size()` liefern, können Sie nicht nur auf dem Bildschirm ausgeben, sondern auch als Wiederholungsbedingung für Schleifen verwenden.

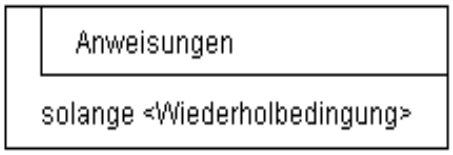
Wiederholung mit Endbedingung (do-while-Schleife)

Statt `wenn … *wenn` oder `wenn … endewenn` wie in Karol gilt in C++ die folgende Schreibweise:

Syntax

```
do
{
  <Anweisungen>;
}
while (<Wiederholungsbedingung>);
```

Struktogramm



Der zwischen `do` und `while` in geschweiften Klammern angegebene **Block** wird wiederholt ausgeführt, solange die Wiederholungsbedingung erfüllt ist, mindestens aber einmal (fußgesteuerte Schleife).

Die Bedingung ist wie bei Karol ein Ausdruck, der `true` oder `false` ergeben muss. Sie könne dafür die folgenden **Vergleichsoperatoren** verwenden:

==	gleich (zwei =-Zeichen!)
>	größer
<	kleiner
>=	größer oder gleich
<=	kleiner oder gleich
!=	ungleich

Beispiel: Der Wert 17,5 (in C++ 17.5) wird so lange immer wieder auf einen Stack gelegt, wie dieser weniger als 100 Werte enthält:

```
stack<double> kommastapel;

do
{
    kommastapel.push(17.5);
} while (kommastapel.size() < 100);

cout << "So viele Zahlen liegen auf dem Stack: " << kommastapel.size() <<endl;
```

Die letzte Zeile gibt den Wert 100 aus.

Wiederholung mit Anfangsbedingung

Der nach `while` angegebene Anweisungsblock wird wiederholt ausgeführt, solange die Anfangsbedingung erfüllt ist (kopfgesteuerte Schleife).

Syntax:

```
while  (<Wiederholungsbedingung>)
{
   <Anweisungen>;
}
```

Struktogramm:

Beispiel: Der Stack aus dem vorherigen Beispiel soll komplett abgebaut werden. Dafür nimmt man so lange Zahlen weg, wie er mehr als null Elemente enthält.

```
while (kommastapel.size()>0)
{
    kommastapel.pop(); // eine Zahl wegnehmen
}
```

```
cout << "So viele Zahlen liegen auf dem Stack: " << kommastapel.size() <<endl;
```

Die letzte Zeile gibt den Wert 0 aus.

Die wichtigsten Datentypen für Zahlenwerte

Ganze Zahlen

Datentyp	Größe (Visual Studio, 32 Bit)	Wertebereich (Visual Studio, 32 Bit)
`short` oder `short int`	16 Bit	-2^{15} bis $+2^{15}-1$
`int`	32 Bit	-2^{31} bis $+2^{31}-1$
`long` oder `long int`	32 Bit	-2^{31} bis $+2^{31}-1$
`long long` oder `long long int`	64 Bit	-2^{63} bis $+2^{63}-1$

Meistens verwendet man `int`, weil dieser Typ für den Prozessor am einfachsten zu verarbeiten ist.

Die Größenangaben gelten für den C++-Compiler von Microsoft, wenn er 32-Bit-Programme erzeugt. Allgemein ist `short` höchstens so groß wie `int` und `long` mindestens so groß. `long long` hat mindestens 64 Bit.

Den Ganzzahltypen kann man das Wort `unsigned` voranstellen. Dann speichern sie nur positive Zahlen. `unsigned int` (oder kurz nur `unsigned`) hat zum Beispiel bei 32 Bit einen Zahlenbereich von 0 bis $2^{32}-1$.

Reelle Zahlen (Gleitkommazahlen)

Datentyp	Größe	Gültige Stellen	Zahlenbereich
`float`	32 Bit	7	$-3{,}4\bullet10^{38}$ bis $-3{,}4\bullet10^{-38}$ und $+3{,}4\bullet10^{-38}$ bis $+3{,}4\bullet10^{38}$
`double`	64 Bit	15	$-1{,}7\bullet10^{308}$ bis $-1{,}7\bullet10^{-308}$ und $+1{,}7\bullet10^{-308}$ bis $+1{,}7\bullet10^{308}$

Auch hier gelten alle Werte für den Microsoft-Compiler. Man verwendet in der Regel `double` und weicht nur auf `float` aus, wenn der Speicherplatz knapp ist.