Übungen zu EDV für Physikerinnen und Physiker (physik131) WS 2011/2012

Jörg Pretz und Daniel Elsner

9. Übung Woche: 12.12.-16.12.2011

Lernziele

Programmieren in C++: Zeiger (Pointer)

Präsenzübungen

Zeiger (Pointer)

- Einfache Zeiger:
  Zur Wiederholung der Vorlesung betrachten Sie folgende Programmzeilen und machen Sie sich deren Bedeutung klar:
  `int i=5;`
  `int* pt;`
  `pt = &i;`
  Die Speicheradresse von `i` lautet etwa `0xb80de650`. Welche Ausgabe erzeugt die Programmzeile
  `cout << *pt << ‘‘ ‘‘ << pt << endl;` ?
  Was ist die Ausgabe von `&i` und `*(&i)`?

- Dynamische Speicherallokation
  Bisher haben wir Arrays immer mit einer festen vordefinierten Anzahl von Elementen definiert, z.B. int data[100]. Dies kann dazu führen, dass man viel mehr Speicher alloziert hat als notwendig, oder umgekehrt, während der Laufzeit ein größeres Array benötigt. Die dynamische Speicherallokation kann hierbei helfen, denn sie erlaubt es, während der Programmausführung festzustellen, wieviel Speicher vom Programm benötigt wird. Das heisst, Sie geben nicht mehr statisch zum Kompiliervorgang vor, wieviel Speicher reserviert werden muss, sondern Ihr Programm ermittelt dynamisch zur Laufzeit, wieviel Speicher benötigt wird und beschafft werden muss.

```
#include <string>
#include <iostream>
using namespace std;

// Hier wird eine Datenstruktur adresse zu Speichern von einem
// Variablenset angelegt.
// Dies entspricht im Prinzip einer sehr einfachen Klasse.
class adresse
```

```
{
  public:
    string Name;
    string Ort;
};

int main()
{
  int i;

  cout << "Wie viele Adressen moechten Sie speichern? " << flush;
  cin >> i;

  // Hier wird ein Array in der benoetigten Groesse [i] angelegt
  adresse* Adressen = new adresse[i];

  // Das Array wird j-fach immer wieder mit dem gleichen Inhalt gefuellt
  for (int j = 0; j < i; ++j)
  {
    Adressen[j].Name = "Franz Kafka"; //oder hier Eingabe abfragen
    Adressen[j].Ort = "Prag"; //oder hier Eingabe abfragen
  }

  // Hier wird der Speicherbereich wieder freigegeben.
  delete[] Adressen;
}
```

Im obigen Code-Beispiel wird zuerst gefragt, wie viele Adressen man speichern möchte. Diese Zahl wird in der Variablen i gespeichert. Der Wert in der Variablen i wird dann verwendet, um genau die benötigte Menge an Speicherplatz zu reservieren (adresse* Adressen = new adresse[i];). Der hier neue Operator **new** rechts vom Gleichheitszeichen ermöglicht es dynamisch Speicher zu reservieren. Das new wird also immer im Zusammenhang mit der dynamischen Speicherallokation verwendet. Der Operator new wird so angewendet, dass Sie hinter new angeben müssen, wie viel Speicher eigentlich reserviert werden soll. Sie bekommen von new einen Zeiger zurück - oder anders gesagt eine Positionsnummer - der auf den reservierten Speicherbereich im RAM zeigt.
(Bemerkung: Sie sehen an der Art des Zugriffs, dass der Zeiger *Adressen* merkwürdigerweise so angewendet werden kann als würde es sich um ein ganz normales Array handeln. Der Datentyp des Zeigers *Adressen* ist zwar adresse*, dennoch meckert der Compiler anscheinend nicht, wenn Sie auf Elemente in diesem Speicherbereich mit Adressen[i] zugreifen. Der Grund, warum dies funktioniert, ist, dass Arrays und Zeiger sehr eng miteinander verwandt sind, soll an dieser Stelle nicht tiefer verfolgt werden.)
Am Ende der Funktion main() enthält das Code-Beispiel folgende Zeile:

```
delete[] Adressen;
```

Mit dem delete Operator wird reservierter Speicher, der nicht mehr benötigt wird, an das Betriebssystem zurückzugeben, d.h. Speicher der mit new reserviert werden kann,

kann mit dem Operator delete wieder freigegeben werden. In diesem Beispiel muss hinter delete eine eckigen Klammern [] angeben werden, da Sie zuvor mit new ein Array dynamisch reserviert haben. Eine einfacheres Bespiel ist folgendes:

```
std::string *s = new std::string;
*s = "Hallo, Welt!";
delete s;
```

- Überprüfungsaufgabe

  Im Folgenden ist *feld* ein int-Array und *p* ein int-Zeiger. Welche der folgenden Zuweisungen sind zulässig, welche nicht? Probieren Sie es gegebenenfalls aus.

```
p = feld;
feld = p;
p = &feld[3];
feld[2] = p[5];
```

  Hier sind *p1* und *p2* zwei int-Zeiger und *i* eine int-Variable. Welche Zuweisungen wird der Compiler akzeptieren, welche nicht? (Ggf. ausprobieren!)

```
p1 = p2 + i;
p1 = i + p2;
i = p1 * p2;
i = p1 - p2;
i = p1 + p2;
```

- Zeiger und Funktionen:
  Schreiben Sie eine Funktion (z.B. swap()) , die zwei Werte bekommt und diese vertauscht. Das Programm könnte folgendermassen aufgebaut sein:

```
#include <iostream>
using namespace std;

void swap(?,?){
?
}
int main(){
int x=2, y=3;
 cout << x << " " << y << endl;
 swap(?,?);
 cout << x << " " << y << endl;
return 0;
}
```

  Ersetzen Sie die Fragezeichen, so dass die Ausgabe folgendermassen aussieht:
  `x=2,y=3`
  `x=3,y=2`
  Überlegen Sie zunächst warum es geschickt ist, für diese Funktion auf Zeiger zurück zu greifen.

Eine Möglichkeit ist es nun an die Funktion `swap()` die Variablen `x,y` zu übergeben. In der Funktion selbst übergeben Sie aber die Positionsnummern (`void swap(&x, &y){}`), d.h. die Referenz auf die Variablen. Nun benötigen Sie nur noch einen lokale Variable vom Typ `int` in der Funktion swap um die beiden übergebenen Variablen zu vertauschen.
Eine zweite Möglichkeit ist es an die Funktion `swap()` nicht die Variablen `x,y` selbst, sondern deren Positionsnummern (`&x, &y`) zu übergeben (`swap(&x, &y);`). In der Funktion (`void swap(**xx ,*yy){}`) kann auf mit Hilfe des Dereferenzierungsoperators (`*`) auf die Variablen zugegriffen werden, deren Positionsnummern hier z.B. in xx und yy gespeichert sind. Die Umsetzung der eigentlichen Vertauschung geschieht dann genauso wie in der erstgenannten Möglichkeit.
Zusammengefasst: Obwohl in diesem Beispiel die Variablen `x,y` nur innerhalb von main() verwendet werden kann, kann durch die Weitergabe der Adresse dieser Variablen an eine andere Funktion, diese ebenfalls mit `x,y` arbeiten.

Berichtsaufgaben

- Parameter bei Programmstart übergeben:
  Erweitern Sie das Programm von Übungszettel Nr. 7 zum Einlesen einer Datenreihe und Ausgabe der Ergebnisse in eine Datei indem Sie die Dateinamen nicht im Quellcode fest programmieren, sondern die Namen bei Programmstart üebergeben. Folgender Aufruf liest die Daten aus der Datei `data_bericht.dat` ein und schreibt die Ergebnisse in die Datei `ergebnisse.dat`:
  `meinprogramm data_bericht.dat ergebnisse.dat`