Dr. Peter Lachmann
Martin Pietsch

SS 2015
02.06.2015

Programmieren - Blatt 4

Diskussion:

- Was ist ein Array? Wieso sagt man, dass Arrays als *call-by-reference* übergeben werden?
- Wenn ein Pointer eine Speicher-Adresse ist, was ist dann der Unterschied zwischen z.B. **int*** und **double***?
- Erklären Sie jeweils die Bedeutung des Symbols &:
  - **double f(int& a)...**
  - **int* a= &b;**
- Wie werden mehrdimensionale Arrays angelegt? Wie werden sie im Speicher abgelegt? Was bedeutet das für die Übergabe eines mehrdimensionalen Arrays als Parameter einer Funktion?
- Was ist der Unterschied zwischen **int a[3][6]**, **int* a[3]**, **int** a**?

Präsenz-Aufgabe 1: *(5-min-Aufgabe zum Warmwerden)*
Auf dem letzten Übungsblatt haben Sie ein Programm geschrieben, welches die Zahlen zweier Variablen (a=5, b=1), an eine Funktion via Referenz übergibt. Schreiben Sie das gleiche Programm und arbeiten Sie nun mit Zeigern.

Präsenz-Aufgabe 2: *(Arrays und Vektoren)*

(a) Schreiben Sie eine Funktion, um das Skalarprodukt zweier Vektoren $x, y \in \mathbb{N}^n$ auszurechnen. Testen Sie Ihre Funktion, indem Sie im Hauptprogramm die Vektoren

$$x = (1, ..., n)^T$$
$$y = (1, ..., 1)^T$$

anlegen, wobei die Länge n der beiden Vektoren als Kommandozeilenparameter eingelesen werden soll.

(b) Wie Sie in Teil (a) vielleicht gemerkt haben, ist der Umgang mit Arrays oft umständlich. Insbesondere muss immer die Länge des Arrays mitgeschleppt werden. Eine sehr nützliche C++-Bibliothek ist die sogenannte *Standard Template Library* (STL), in der unter anderem verschiedene sogenannte Container-Klassen bereitgestellt werden (also Datenstrukturen wie Listen, Bäume etc.).
Für uns interessant ist die Klasse *vector*, welche dynamische Arrays implementiert. Die im Vektor enthaltenen Elemente werden dabei genau wie bei üblichen Arrays im Speicher hintereinander abgelegt, so dass man auch wieder mit Pointern arbeiten kann, wenn man möchte. Vektoren sind aber Objekte und haben noch ein paar hilfreiche Methoden (unter anderem die Funktion **size**, welche die Größe des Vektores zurückgibt.).

```
#include <vector>

using namespace std;

int main(){
  // Vektor der Laenge 10 anlegen
  vector<int> my_vec(10);

  // Vektor mit Werten fuellen
  for (unsigned int i = 0; i < my_vec.size(); ++i){
    my_vec[i] = i;
  }

  return 0;
}
}
```

Schreiben Sie Ihr Programm aus (a) so um, dass es mit einem Vektor arbeiten kann.

Präsenz-Aufgabe 3: *(Mehrdimensionale Arrays und Vektoren, dynamische Speicherverwaltung)*

(a) Schreiben Sie ein Programm zur Matrix-Vektor-Multiplikation, hier zunächst ohne dynamische Speicherverwaltung. Dabei sollte

- im Hauptprogramm eine Matrix $A \in \mathbb{R}^{nxm}$ und ein Array $x \in \mathbb{R}^m$ angelegt und mit WErten gefüllt werden,
- die Matrix-Vektor-Multiplikation $A \cdot x$ in einer Funktion **mv** implementiert werden.

Definieren Sie sich zunächst die größen n, m am Anfang des Dokuments mittels

```
#define n 3
#define m 4
```

Überlegen Sie sich, wie das Ergebnis der Multiplikation aussieht und wie dieses zurückgegeben werden kann. Geben Sie das Ergebnis auf der Console aus.

(b) Modifizieren Sie Ihr Programm aus (a) so, dass Sie keine globalen Variablen mehr benötigen. Hierzu benötigen Sie explizit Pointer, sowie das Kommando **new**. Denken Sie daran, dynamisch angelegte Datenstrukturen auch wieder freizugeben (in C++ mit delete bzw. delete[]).