ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Informatik I - D-ITET

Wintersemester 2003/2004

Prof. Markus Gross, Bruno Heidelberger, Richard Keiser, Nicky Kern, Edouard Lamboray, Christoph Niederberger, Tim Weyrich, Felix Eberhard, Manuel Graber, Nathalie Kellenberger, Marcel Kessler, Lior Wehrli

Uebung 6 - Pointers

Ausgabe:	8. Dezember 2003
Abgabe:	15. Dezember 2003
Autor:	Bruno Heidelberger

1. Programmanalyse — 4 Punkte

Gegeben sei folgendes Programmsegment:

```
1     int a = 10;
2     int b = 0;
3     int *c_ptr = 0;
4     int *d_ptr = 0;
5     int e[] = {0, 1, 2, 3, 4};
6
7     c_ptr = &a;
8     d_ptr = &b;
9     *c_ptr = e[4];
10    e[1] = *c_ptr;
11
12    *d_ptr = *c_ptr;
13    d_ptr = &(e[2]);
14    c_ptr = e;
15    *c_ptr = 7;
16
17    *c_ptr++ = b + *d_ptr;
18    *++c_ptr = a + e[3];
19    *e = *c_ptr;
20    *d_ptr = *++c_ptr;
21    *c_ptr = b;
```

Analysiere dieses Programm und gebe für alle Zeilen von 7 bis 21 an, was passiert und wie die Variablenbelegung ist. Stelle in einer Tabelle die Werte der Variablen `a` und `b` und den Inhalt des Arrays dar. Zeige ebenfalls an, auf welchen Wert die Pointer `c_ptr` und `d_ptr` in jeder Zeile zeigen.

Hinweis: `a++` inkrementiert `a` nach der Ausführung der Anweisung, wohingegen das Inkrement bei `++a` vor der Ausführung durchgeführt wird.

2. n-dimensionale Vektoren — 3 Punkte

In dieser Aufgabe sollen Vektoren einer beliebigen Grösse dynamisch erzeugt werden und die Vektoraddition und das Skalarprodukt berechnet werden.

a) Erzeugung Vektoren

Die zu implementierenden Vektoren sollen *n* float-Werte aufnehmen können, wobei der Benutzer die Grösse *n* zu Beginn des Programms eingeben muss. Danach sollen mittels dem `new` Operator drei Vektoren erzeugt werden, wobei zwei mit aufsteigenden Werten initialisiert werden sollen:
z.B. für n=3: `a=[1.0, 2.0, 3.0], b=[4.0, 5.0, 6.0], c=[0.0, 0.0, 0.0]`

b) Addition

Nun sollen die beiden ersten Vektoren addiert werden und das Resultat in den dritten geschrieben werden. Mit obigem Beispiel würden die Vektoren danach folgende Werte beinhalten: `a=[1.0, 2.0, 3.0], b=[4.0, 5.0, 6.0], c=[5.0, 7.0, 9.0]`

c) Skalarprodukt

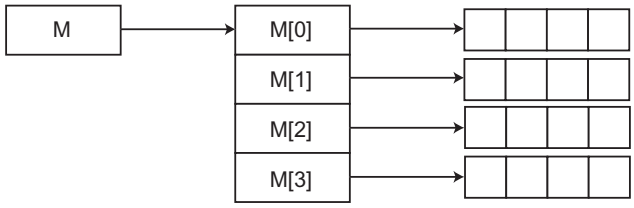
Berechne nun das Skalarprodukt der ersten zwei Vektoren und schreibe das Resultat wieder in Vektor `c`. Das Skalarprodukt ergibt sich aus der Summe der elementweisen Multiplikation: `1.0 * 4.0 + 2.0 * 5.0 + 3.0 * 6.0 = 32.0`

d) Speicher freigeben

Zum Schluss des Programms müssen die dynamisch erzeugten Vektoren wieder gelöscht werden.

3. Dynamische mehrdimensionale Arrays 3 Punkte

Das Ziel dieser Aufgabe ist es, ein Programm zu schreiben, welches ganzzahlige, quadratische Matrizen beliebiger Dimension zur Laufzeit erzeugt. Ein zweidimensionales Array der Grösse 4x4 können wir uns wie folgt vorstellen:



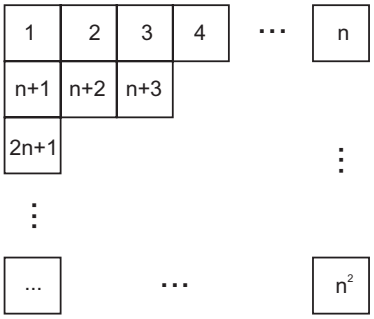
M ist der Ursprung der Matrix und ist ein Pointer auf ein Array von Pointern (senkrecht: M[0] bis M[3]), welche wiederum auf ein Array von Matrix-Werten (waagrecht) zeigen.
Um eine solche Matrix zu erzeugen, muss also zuerst M definiert werden und darauf ein Array von Pointern alloziert werden. Danach muss für jeden Pointer in M (M[0] bis M[3]) jeweils ein Array von `int` alloziert werden.
Ueberlege genau, welchen Typ M haben muss und von welchem Typ die Elemente M[i] sein müssen.

a) Erzeugen der Matrix

Lies die Seitenlänge n als Benutzereingabe von der Kommandozeile und alloziere den Speicher für die Matrix gemäss obigem Beispiel für eine nxn Matrix.

b) Füllen der Matrix

Traversiere nun die Matrix mittels for-Schleifen und fülle sie zeilenweise mit einem aufsteigenden Index auf:



c) Ausgabe

Gib jetzt den Inhalt der Matrix in der Shell aus.

d) Freigeben der Matrix

Gib am Schluss den allozierten Speicher wieder frei. Beachte, dass du zuerst die Zeilen-Arrays löschen musst, bevor du das Spalten-Array löschst.

4. Unix-Einführung: Prozesse (fakultativ)

In Unix läuft jedes Programm und jedes Kommando in einem eigenen Prozess ab. Die Prozesse werden über eine ganze Zahl, den sogenannten Process Identifier (PID), referenziert. Die wichtigsten Kommandos um die aktuellen Prozesse unter Unix anzuzeigen sind:

- **`top`**: zeigt eine Liste aller Prozesse, geordnet nach Resourcennutzung. Schau dir die Manualpages zu **`top`** an oder gebe **`h`** in einer Shell ein, in der gerade **`top`** läuft.
- **`ps`**: zeigt eine Liste der Prozesse, die aus der entsprechenden Shell gestartet wurden. Die Option **`-f`** zeigt mehr Details zu den Prozessen an, **`-e`** zeigt alle Prozesse an und **`-u username`** alle Prozesse eines bestimmten Users.

Falls ein Programm mit dem Zusatzzeichen **&** gestartet wird, läuft es als Hintergrundprozess ab. Das heisst, während der Programmausführung kann die entsprechende Shell weiterhin normal genutzt werden. Wird ein Programm im Vordergrund gestartet, ist die entsprechende Shell bis zur Beendung des Programms blockiert. Durch die Tastenkombination **`Ctrl-Z`** kann ein Programm unterbrochen und mit **`fg`** oder **`bg`** in einen Vordergrund- respektive Hintergrundprozess verwandelt werden.

Mit dem Befehl
`> kill pid`
oder
`> kill -9 pid`
kann ein Programm mit dem PID ***pid*** abgebrochen werden. Dies ist besonders nützlich, falls ein Programm blockiert und nicht mehr normal beendet werden kann. Die Tastenkombination **`Ctrl-C`** in einer Shell bewirkt das gleiche, vorausgesetzt das entsprechende Programm läuft als Vordergrundprozess.

Experimentiere mit den genannten Kommandos und einem Programm deiner Wahl.