ETH
Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo

computer graphics lab

Departement Informatik
Dr. R. Peikert - Christian Sigg

12. Dezember 2003

37-847 Informatik Übung 8 WS03/04

Hinweis: In der ersten Woche im neuen Jahr finden Schnellübungen statt.

1) Programmanalyse

Gegeben sei folgendes Programmsegment:

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

Analysieren Sie dieses Programm und geben Sie für alle Zeilen von 6 bis 20 an, was passiert und wie die Variablenbelegung ist. Stellen Sie in einer Tabelle die Werte der Variablen `i` und `j` und den Inhalt des Arrays dar. Zeigen Sie ebenfalls an, auf welchen Wert die Pointer `m_ptr` und `n_ptr` in jeder Zeile zeigen.

2) n-dimensionale Vektoren

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]
```

b) Addition

Schreiben Sie eine Funktion, welche die ersten beiden Vektoren addiert und das Resultat in den dritten Vektor schreibt. 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

Schreiben Sie eine Funktion, welche das Skalarprodukt von zwei Vektoren berechnet. Das Skalarprodukt ergibt sich aus der Summe der elementweisen Multiplikation:

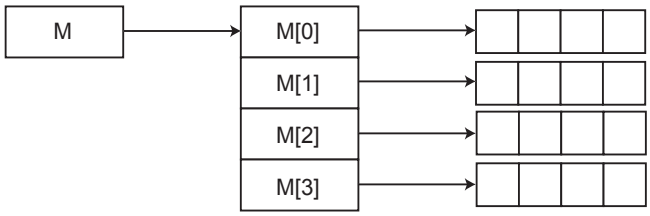
```
1.0 * 4.0 + 2.0 * 5.0 + 3.0 * 6.0 = 32.0
```

d) Speicher freigeben

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

3) Dynamische mehrdimensionale Arrays

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:



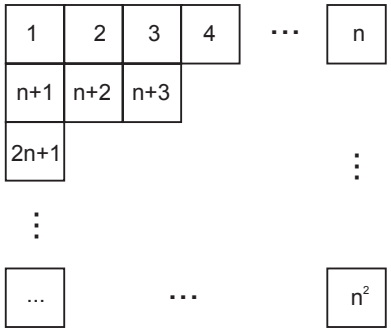
M ist der Ursprung der Matrix und ist ein Pointer auf ein Array von Pointern (senkrecht: M[0] bis M[3]), welche wiederum auf ein Array von Matrix-Werten (waagrecht) zeigen. Um eine solche Matrix zu erzeugen, muss also zuerst M definiert werden und darauf ein Array von Pointern alloziert werden. Danach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

a) Erzeugen der Matrix

Lesen Sie die Seitenlänge n als Benutzereingabe von der Kommandozeile und allozieren Sie den Speicher für die Matrix gemäss obigem Beispiel für eine nxn Matrix.

b) Füllen der Matrix

Traversieren Sie nun die Matrix mittels for-Schleifen und füllen Sie sie zeilenweise mit einem aufsteigenden Index auf:



c) Ausgabe

Geben Sie jetzt den Inhalt der Matrix nach `std::cout` aus.

d) Freigeben der Matrix

Geben Sie am Schluss den allozierten Speicher wieder frei. Beachten Sie, dass zuerst die Zeilen-Arrays gelöscht werden müssen, bevor das Spalten-Array gelöscht wird.

Abgabetermin: 4. Januar 2004