Serie 3

Arbeiten mit Zeigern und dynamischem Speicher

Übungsaufgaben

Aufgabe 1 – Zeiger durchdacht

Führen Sie das nachfolgende Programm in Gedanken aus! Skizzieren Sie worauf die Zeiger jeweils vor der Ausgabe verweisen! Welche Ausgabe wird erzeugt?

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

int x = 0;
int y = 1;

void f(int* p) {
     p = &y;
     cout << *p << endl;
}

int main() {
     int* p = &x;
     cout << *p << endl;

     f(p);
     cout << *p << endl;

     int* q = &y;
     p = q;
     cout << *p << endl;
}
```

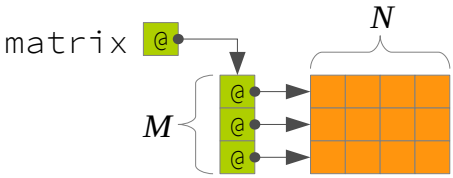
Aufgabe 2 – Dynamischer Speicher

Schreiben Sie ein kleines C++ Programm zum Üben des Umgangs mit dynamischem Speicher (Heap). Folgende Aspekte soll das Programm beinhalten:

1. Reservieren von Speicher für eine vom Nutzer einzugebende Anzahl von ganzen Zahlen
2. Belegen des Speichers mit zufälligen Werten
3. Ausgaben der Werte sowie deren Adressen im Speicher
4. Freigeben des Speichers

Aufgabe 3 – Matrizenverwaltung

Schreiben Sie Funktionen zum Arbeiten mit beliebigen $M \times N$ Matrizen (M Zeilen und N Spalten). Der Speicher für Matrizen soll dynamisch zugewiesen werden. Eine Matrix soll mittels Zeiger-auf-Zeiger modelliert werden. Das heißt, eine Matrix entspricht einem Zeiger, der auf M Zeiger zeigt (für M Zeilen), die wiederum auf N Zahlenwerte zeigen (für N Spalten).



Zunächst ist folgende Funktion für dieses Modell zu entwickeln:

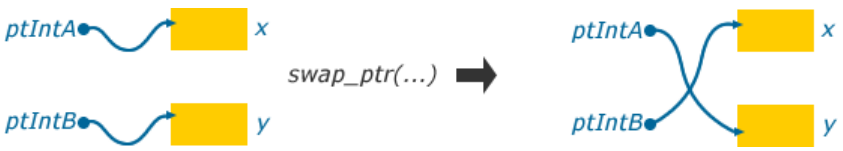
1. `.. allocateMatrix(..)` zum Reservieren des notwendigen dynamischen Speichers für eine $M \times N$ Matrix und Rückgabe des Zeigers auf den Speicher.

Zusatzaufgabe – Zeigertausch

Schreiben Sie eine C++ Funktion `swap_ptr(..)`, die zwei Zeiger vertauscht (es sollen nicht die Werte vertauscht werden)! Folgendes Beispiel verdeutlicht, wie die Funktion angewendet wird:

```
int main() {
     int x = 10;
     int y = 20;
     int* ptIntA = &x;
     int* ptIntB = &y;

     swap_ptr(...);
}
```



Nach dem Aufruf der Funktion `swap_ptr` soll `ptIntA` auf `y` zeigen und `ptIntB` auf `x`.

Hausaufgaben

Aufgabe 1 – Arbeiten mit Matrizen (Punkte: 3)

Setzten Sie Aufgabe 3 der Übung fort und schreiben Sie weitere C++ Funktionen für das Arbeiten mit Matrizen:

2. `.. freeMatrix(..)` Der für die zu übergebene $M \times N$ Matrix reservierte Speicher soll durch diese Funktion korrekt wieder freigegeben werden.
3. `.. readMatrix(..)` Die Funktion soll eine $M \times N$ Matrix im Speicher reservieren, deren Inhalt von der Tastatur einlesen und die so erstellte Matrix zurückgeben.
4. `.. printMatrix(..)` Der Inhalt einer zu übergebenen $M \times N$ Matrix soll durch diese Funktion auf dem Bildschirm ausgegeben werden.

Aufgabe 2 – Rechnen mit Matrizen (Punkte: 4)

Schreiben Sie zwei weitere C++ Funktionen für einfache Rechenoperationen mit Matrizen:

5. `.. transposeMatrix(..)` Zu einer übergebenen $M \times N$ Matrix A soll diese Funktion die transponierte Matrix A^T erstellen, berechnen und zurückgeben.
6. `.. multiplyMatrices(..)` Zu einer übergebenen $M \times N$ Matrix A und einer übergebenen $N \times P$ Matrix B soll diese Funktion das Produkt der Matrizen A und B berechnen und in einer neu anzulegenden Matrix $C = AB$ speichern und zurückgeben.

Aufgabe 3 – Testen der Matrixfunktionen (Punkte: 3)

Schreiben Sie ein C++ Hauptprogramm zum Testen der Matrixfunktionen. Ihr Programm soll zunächst die Dimensionen M und N und dann eine $M \times N$ Matrix A einlesen. Anschließend ist die Transponierte A^T zu berechnen sowie das Produkt AA^T. Anschließend sind A, A^T sowie AA^T auf dem Bildschirm auszugeben. Vor dem Beenden des Programms sollen alle dynamisch reservierten Speicherbereiche wieder korrekt freigegeben werden.

Vermerken Sie auf Ihrem Lösungsblatt bitte auch die Matrix A, mit der Sie getestet haben, sowie die Ergebnisse A^T und AA^T.