Klausur
Kompaktkurs Einführung in die Programmierung
Dr. T. Weinzierl & M. Sedlacek
25. März 2011

Wichtig:

- Die Angabe besteht aus 10 Seiten — prüfen Sie bitte Ihr Exemplar auf Vollständigkeit.
- Kennzeichen Sie jedes Blatt lesbar mit Ihrem Namen und dieses Deckblatt zusätzlich mit Ihrer **Matrikelnummer und Unterschrift**; trennen Sie die Heftung nicht auf.
- Bearbeiten Sie die Aufgaben auf den Angabenblättern. Sollte der Platz für einen Aufgabenteil nicht reichen, benutzen Sie bitte die Rückseite der Angabenblätter und machen Sie einen entsprechenden Vermerk bei der Aufgabe.
- Insgesamt können Sie 40+4 Punkte erreichen (4 Punkte Überhang), bei jeder Aufgabe ist die maximale Anzahl an erreichbaren Punkten angegeben. Zum Bestehen sind 17 Punkte erforderlich.
- Es ist alles außer elektronische Geräte als Hilfsmittel zugelassen.
- Eigene Funktionen dürfen in den folgenden Teilaufgaben weiterverwendet werden; von den Standardfunktionen sind, soweit nicht bei der Aufgabe anders angegeben, nur die zur Ausgabe zulässig (`std::cout` bzw. `printf`, was immer Ihnen davon lieber ist).
- Die Bearbeitungszeit beträgt 75 Minuten.

Aufg.	1)	2)	3)	4)
Pkt.	/11	/9	/10	/10(+4)

Summe	/40(+4)
Note	

1 Warm Up (max. 11 Punkte)

a) Was ist der Inhalt von `a` und `b` nach Ausführung dieser Anweisungen?

```
int a = 0;
int b = -3;
if (++a != 0 && b == -3)
    a = a + b;
b = --a;
```

b) Was ist die Ausgabe des folgenden Programmstücks? Achten Sie auf exakte Darstellung der Ausgabe (Zeilenumbrüche?), wie sie bei Ausführung sichtbar wird.

```
int feld[5];
for(int i = 4; i >= 0; i--)
    feld[i] = i + i*i;
for(int i = 0; i < 5; i++)
    std::cout << feld[i] << " ";
```

c) Was ist der Inhalt von `a` und `b` nach Ausführung dieser Anweisungen?

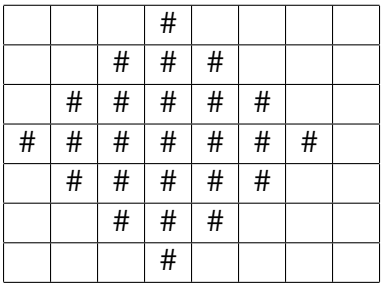
```
int a = 3;
int b = 7;
a == 3 ? b == 4 ? b = 3 : a = 4, b = -3 : (a = 8, b = -1);
```

d) Entscheiden Sie jeweils, ob die Anweisung in der Zeile der Kästchen korrekt ist (kompiliert) oder nicht (kompiliert nicht).

Programm	korrekt	nicht korrekt
`int main()`		
`{`		
`  float z = -4.5;`		
`  int i = 0;`		
`  char c = 'd';`		
`  double ***pppd;`	☐	☐
`  double *pd = z;`	☐	☐
`  int j = c-4;`	☐	☐
`  z++;`	☐	☐
`  c++ = ++i;`	☐	☐
`  return 0;`		
`}`		

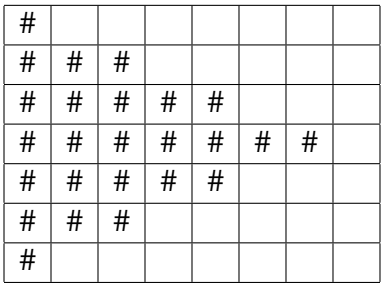
2 Schleifen und Ausgabe (max. 9 Punkte)

Im Folgenden soll eine Reihe von Funktionen realisiert werden, die ein Rautenmuster wie im Bild rechts druckt. Dabei soll die Höhe der Raute von einer Variablen `hoehe` abhängen. Im Bild rechts hat die Raute `hoehe = 4`, da sich die längste Zeile, von oben gezählt, an vierter Position befindet. Alle nachfolgenden Zeilen sind wieder stufenweise kürzer.



a) Schreiben Sie eine Funktion `void drucke_zeile(int len)`, die `len` mal das Zeichen '#' und abschließend einen Zeilenumbruch druckt. Hierbei soll noch nicht auf das Rautenmuster eingegangen werden.

b) Schreiben Sie nun eine Funktion `void raute(int hoehe)`, die die Raute linksbündig wie im Bild rechts druckt. Nehmen Sie hierzu Ihre Funktion aus Teilaufgabe a) zu Hilfe.
Hinweis: Es sollen linksbündige Rauten beliebiger `hoehe` gedruckt werden können und nicht nur für `hoehe = 4`.



c) Schreiben Sie nun eine Funktion `void einruecken(int z)`, die `z` mal eine Leerposition (whitespace) druckt. Vervollständigen Sie anschließend Ihre Funktion `void raute(int hoehe)` aus Teilaufgabe b) so, dass die Raute in richtiger Form gedruckt wird. Hierzu müssen Sie ihre Funktion aus Teilaufgabe b) nochmals abschreiben und mit der Funktion `void einruecken(int z)` korrekt vervollständigen.
Hinweis: Es sollen Rauten beliebiger `hoehe` gedruckt werden können und nicht nur für `hoehe = 4`.

3 Rang-Eins-Matrizen und Ausgabe (max. 10 Punkte)

Im Folgenden soll eine Reihe von Funktionen zur Ausgabe von Vektoren bzw. Matrizen sowie eine Funktion zur Erstellung einer Rang-Eins-Matrix realisiert werden. Eine *Rang-Eins-Matrix* ist eine Matrix, die aus der Multiplikation xy^T zweier Vektoren $x \in \mathbb{R}^n$ und $y \in \mathbb{R}^n$ hervorgeht, d.h. für die Elemente der resultierenden Matrix gilt:

$$\left(xy^T\right)_{ij} = x_i \cdot y_j.$$

a) Skizzieren Sie grob das Speicherdiagramm bei der Ausführung folgenden Programmcodes. Skizzieren Sie ebenfalls Zeigerzuweisungen und fiktive Adressen neben den Speicherzellen.
Hinweis: Es ist keine explizite Funktion gefordert, sondern nur ein grobes Speicherdiagramm, wie es in den Vorlesungen (und Übungen) benutzt wurde.

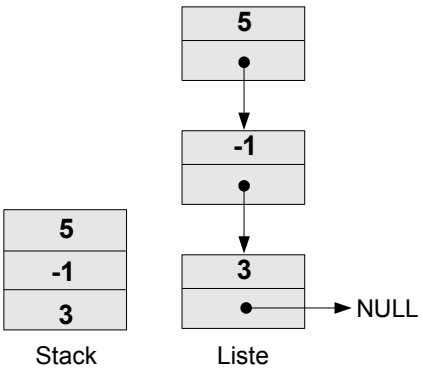
```
int n = 3;
double **mtx = new double*[n];
for (int i = 0; i < n; i++)
    mtx[i] = new double[n];
mtx[0][2] = -7.0;
mtx[2][1] = 5.0;
```

b) Schreiben Sie eine Funktion `void drucke_matrix(double **mtx, int zeilen, int spalten)`, oder mehrere ineinandergreifende Funktionen, zur Ausgabe einer korrekt initialisierten Matrix. Dabei soll die Matrix, die über ein mehrdimensionales Feld `**mtx` übergeben wird, in lesbarer Form ausgegeben werden, d.h. jede Zeile der Matrix soll in einer eigenen Zeile ausgegeben werden und zwischen den einzelnen Elementen sollen Leerzeichen stehen.

c) Schreiben Sie eine Funktion `double** rangeins(double* x, double* y, int n)`, die als Funktionswert die Rang-Eins-Matrix zu den entsprechenden Vektoren x und y liefert. Mit dem Parameter n werde die Länge der Vektoren x und y übergeben. Der Speicherplatz für die zu erzeugende Ergebnismatrix ist innerhalb der Funktion dynamisch (mit `new`) anzulegen, d.h. die Matrix soll innerhalb der Funktion, vor der eigentlichen Berechnung, korrekt als Nullmatrix initialisiert werden.

4 Stack und Listen (max. 10 (+4) Punkte)

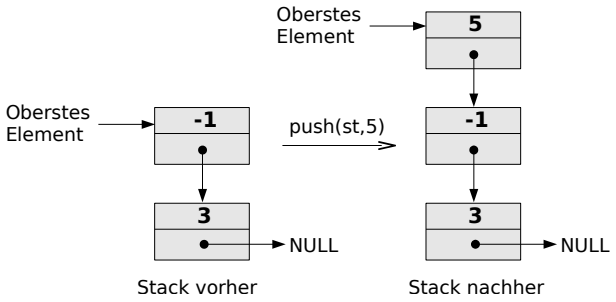
Ein Stack (Keller- oder Stapelspeicher) ist eine listenähnliche Datenstruktur, in die nur an einem Ende eingefügt oder gelöscht werden kann. Bildlich werden die Elemente übereinander gestapelt und in umgekehrter Reihenfolge vom Stapel genommen. Dies wird auch *Last In First Out*-Prinzip (LIFO) genannt. Dabei werden die Elemente immer vorne an die Liste angehängt. Das Bild rechts stellt einen Stack und die dazu korrespondierende Listenstruktur von drei Elementen dar. Im Folgenden sollen eine Reihe von Funktionen geschrieben werden, die auf einem Stack operieren.



Hierbei soll keine ObjektOrientierung verwendet werden. Halten Sie sich an die vorgegebenen Funktionssignaturen!

a) Definieren Sie einen rekursiven Strukturtyp `struct stackelement`, der nach dem Bild oben rechts ein Datum sowie einen Zeiger auf ein vorheriges Stackelement hat.

b) Schreiben Sie eine Funktion `stackelement* push(stackelement* st, int data)`, die auf dem Stack ein neues Element ablegt. Dabei soll der Zeiger auf das oberste Element aktuell gehalten werden. Siehe als Beispiel Bild rechts. Sie können davon ausgehen, dass der übergebene Zeiger `st` bereits auf das oberste Stackelement zeigt.



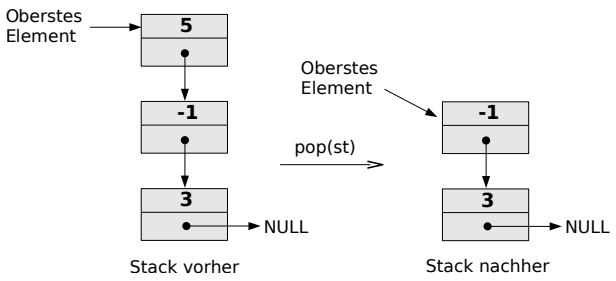

```
stackelement* push(stackelement* st, int data) {
     //Fügen Sie hier code ein




}
```

c) Schreiben Sie eine Funktion `int top(stackelement* st)`, die das Datum des obersten Stackelements zurückgibt.

d) Schreiben Sie eine Funktion `stackelement* pop(stackelement* st)`, die das oberste Element des Stacks löscht und den Zeiger auf das neue oberste Element aktualisiert. Siehe als Beispiel Bild rechts. Für den Fall, dass der Stack leer ist, soll eine Fehlermeldung ausgegeben und `NULL` zurückgegeben werden.



e) Schreiben Sie eine Funktion die einen Stack übergeben bekommt und den gesamten Stack druckt. Dabei sollen die Elemente von oben nach unten ausgegeben werden. Das heißt, dass das zuletzt hinzugefügte Element als erstes gedruckt werden soll.

f) Schreiben Sie nun eine **objektorientierte** Variante der Funktion `pop(...)`, die das oberste Element des Stacks löscht, analog zu Teilaufgabe d).