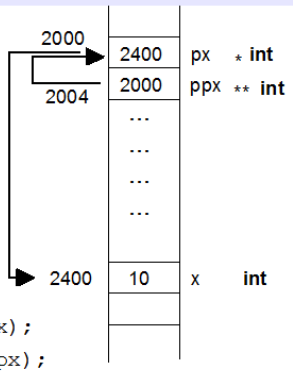
ÜBUNG: Einfache Zeigeroperationen

Gegeben ist folgendes Programm:

```
int x = 10;
int *px;
int **ppx;

px  = &x;
ppx = &px;

printf("%d %d %d", x,    &x,     px);
printf("%d %d %d", &px,  *px,    *ppx);
printf("%d %d %d", &ppx, **ppx,  ppx);
```



Variable, die eine Adresse enthält!

x = 10   &x = 2400   px = 2400

&x = 2000   *px = 10   *ppx = 2400

&ppx = 2004   **ppx = 10   ppx = 2000

ÜBUNG: Umgang mit Zeigern

```
int a[]={1,2,3,4,5,6,7};

int main(){
        int *p1;
        int **p2;

        p1=&a[2];
        p2=&p1;

        printf("%d\n",a[3]);
        printf("%d\n",*p1);
        printf("%d\n",p1);
        printf("%d\n",&p1);
        printf("%d\n",p2);
        printf("%d\n",&p2);
        printf("%d\n",p2[0][2]);

        return 0;
}
```

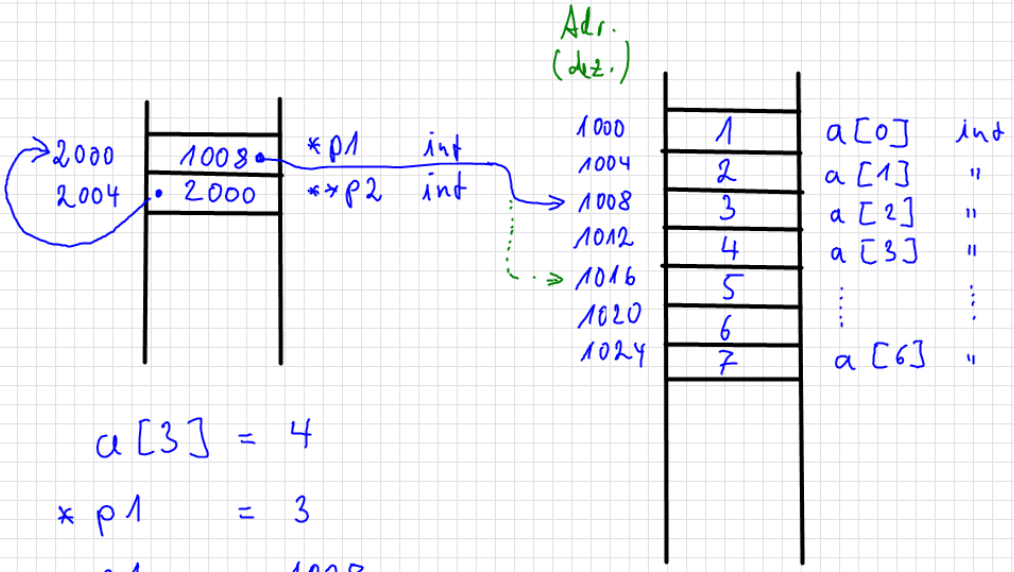
Folgende Annahmen gelten:

Das Feld a beginne bei Adresse 1000 (dezimal).

Der Zeiger p1 stehe bei Adresse 2000 (dez.) gefolgt vom Zeiger p2.

Achtung:

Das Beispiel hat didaktischen Wert, zeugt aber nicht gerade von gutem Stil !



a[3] = 4

*p1 = 3

p1 = 1008

&p1 = 2000

p2 = 2000

&p2 = 2004

p2[0][2] = 5

Ein Zeiger auf ein Zeiger muss doppelt dereferenziert werden !

Ein Einfachzeiger wird einfach dereferenziert !

p1[2] = 5

ÜBUNG: Beispiele zu Feldern und Strings

1. Schreiben Sie ein C-Programm, welches eine 4x3-Matrix mit einem Vektor multipliziert.

Vorüberlegungen:

$$\underline{A} = \begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \\ a_{41} & a_{42} & a_{43} \end{pmatrix} \qquad \vec{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$$

Matrix 4 × 3 — Vektor

$$\underline{A} \cdot \vec{x} = \vec{b}$$

$[4,3] \cdot [3,1] \rightarrow [4,1]$

Verkettbar

$$\vec{b} = \begin{pmatrix} a_{11} \cdot x_1 + a_{12} \cdot x_2 + a_{13} \cdot x_3 \\ a_{21} \cdot x_1 + a_{22} \cdot x_2 + a_{23} \cdot x_3 \\ a_{31} \cdot x_1 + a_{32} \cdot x_2 + a_{33} \cdot x_3 \\ a_{41} \cdot x_1 + a_{42} \cdot x_2 + a_{43} \cdot x_3 \end{pmatrix}$$

$$b_r = \sum_{c=1}^{3} a_{r,c} \cdot x_c$$

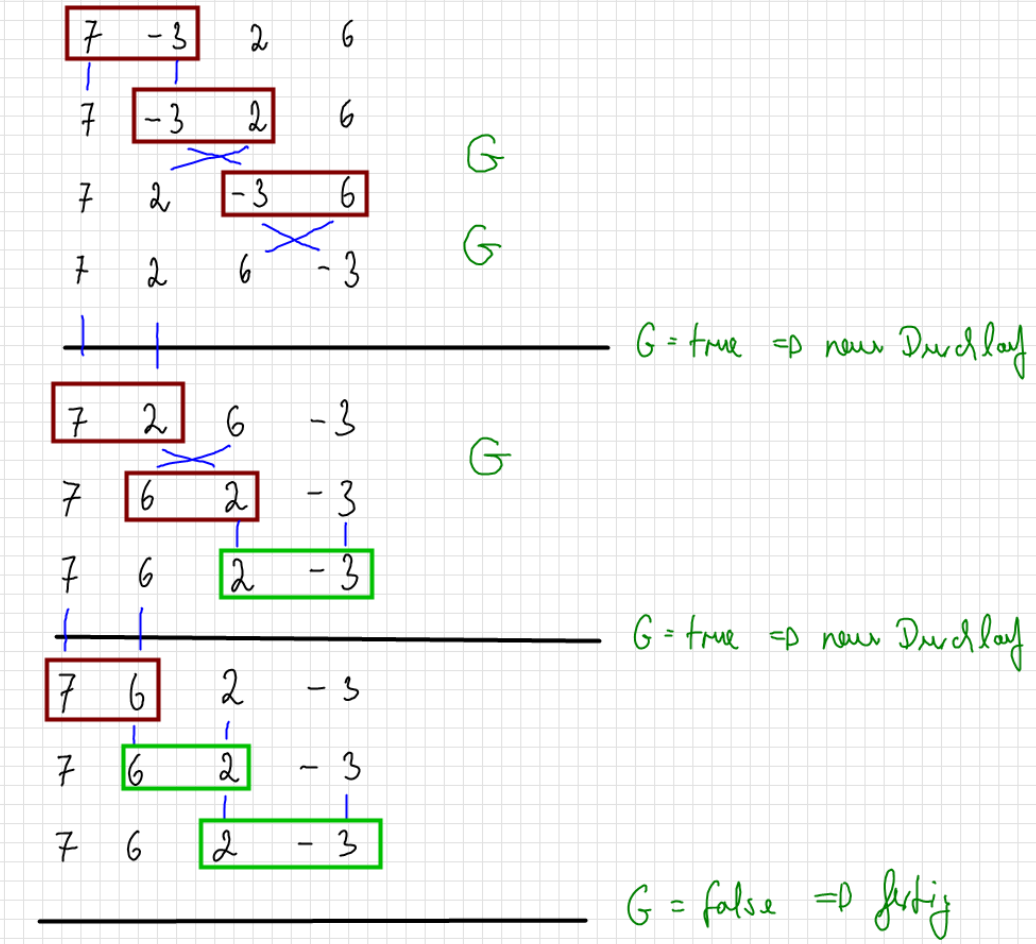
r = row (Zeile)
c = column (Spalte)
r = 1 ... 4

```c
int A[][3] =  { 1,2,4,
                4,5,3,
                3,3,1,
                2,4,9 };

int x[] = { 2 ,5, 9 };
int b[] = { 0, 0, 0, 0 };

int main(){
        int r, c; // Row u. Column

        for(r=0; r<4; r++){  // alle Zeilen
                for(c=0; c<3; c++){ // Summe über alle Spalten
                        b[r] = b[r] + A[r][c] * x[c];
                }
        }

        printf("%d  %d  %d  %d\n", b[0], b[1], b[2], b[3]);

        return EXIT_SUCCESS;
}
```

2. Schreiben Sie ein C-Programm, welches die Grossbuchstaben eines Strings zählt und ausgibt.

```c
#include <stdio.h>
#include <stdlib.h>


int main(){
        char            txt[] = "AbcDDrRTgGzuJ";
        char            currChar;
        unsigned int    i=0, cap=0;

        while((currChar = txt[i++]) != '\0'){
                if(currChar >= 'A'  &&  currChar <= 'Z'){
                        cap++;
                }
        }

        printf("cap = %d\n", cap);

        return EXIT_SUCCESS;
}
```

Bubble - Sort



* kann entfallen

```c
#include <stdio.h>
#include <stdlib.h>


/* Funktionsdeklaration */
void    SortInteger(int *IArray, unsigned int Anzahl);


/* Variablen */
int x[8]        = {7, -3, 5, -4, -1, 9, 3, 2};
unsigned int n  = 8;

int main(){
        int i;

        SortInteger(x, n);

        for(i=0 ; i<n; i++)
                printf("x[%d]=%d\n", i, x[i]);

        return EXIT_SUCCESS;
}
```

Kann auch entfallen

3

1

2

```c
/* Funktionsdefinition */
void SortInteger(int *a, unsigned int m){
        char    Getauscht = 1;
        int     i, temp;

        while(Getauscht){
                Getauscht=0;
                for(i=0; i<m-1; i++){
                        if(a[i] < a[i+1]){
                                temp    = a[i];
                                a[i]    = a[i+1];
                                a[i+1] = temp;
                                Getauscht = 1;
                        }
                }
                m--;
        }
}
```

(*)

4

(*) Anm.: Ein indizierter Zeiger liefert den Wert, auf den der Zeiger zeigt!

ÜBUNG: Call-by-reference, Umgang mit Strings

Schreiben Sie ein Unterprogramm, welches zwei Strings lexikographisch vergleicht.

Ansatz: Die beiden Strings (s, t) werden Zeichen für Zeichen miteinander verglichen. Sobald ein Unterschied festgestellt wird, wird der Vorgang abgebrochen.

In diesem Fall wird die Differenz der ASCII-Werte der zuletzt betrachteten Zeichen ausgegeben.

Sind die beiden Strings gleich, wird 0 zurückgegeben.

Weiter ist das Hauptprogramm zu schreiben, welches das Unterprogramm aufruft.

```c
#include <stdio.h>
#include <stdlib.h>


/* Funktionsdeklaration */
void    strcmp(char *String1, char *String2);


/* Variablen */
char str1[] = "Autoreifen";
char str2[] = "Autoradio";


int main(){
        char cmp;

        cmp = strcmp(str1, str2);

        if(cmp>0)
                printf("str1 nach str2");
        else if(cmp == 0)
                printf("str1 gleich str2");
        else /* cmp<0 */
                printf("str1 vor str2");

        return EXIT_SUCCESS;
}
```

```c
/* Funktionsdefinition */
void strcmp(char *s, char *t){

        int     i=0;

        while(s[i]==t[i]){
                if(s[i++] == '\0') // Gleich und Stringende
                        return(0);
        }
        return (s[i] - t[i]); // Strings ungleich
}
```