

Computer Vision  
and Geometry Lab

# Informatik I for D-MAVT

## Exercise Session 7

# Organisatorisches

- Teaching Assistant
  - Alexander Schwing ([aschwing@student.ethz.ch](mailto:aschwing@student.ethz.ch))
  - CAB G 89
  - Website: <http://www.alexander-schwing.de/Info1DMaVt/>
- Übungsabgabe
  - Auf Papier, keine Emails
    - Später wenn Programme umfangreicher werden, ev. auch Abgabe per Email möglich
- Testatbedingungen
  - 75% aller Übungen
    - Voraussichtlich 12 Serien (d.h. 9 gelöste Serien)

# Probleme der letzten Übung

- Nicht C/C++ konforme Syntax für arrays

```
int n;                                int n;
std::cin >> n;                          std::cin >> n;
float f_arr[n];                        float *f_arr = new float[n];
...                                     ...
delete[] f_arr;
```

- Kein goto (Spaghetti-Code)

# Repetition

## ■ Arrays und Pointer

### ■ Statisch alloziert:

```
double var[2][3];
```

Linear im Speicher:



Aber: Lineare Indizierung `var[3]` ist nicht gültig!

Warum?

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

int main() {
    double var1[2][3];
    double cnt = 1.0;
    for(int i=0;i<2;++i)
        for(int j=0;j<3;++j)
            var1[i][j] = cnt++;
}
```

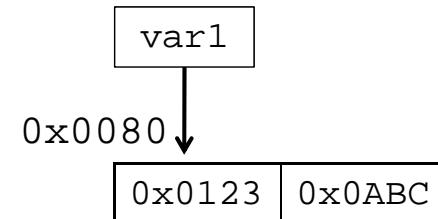
```
    double* pvar = var1[0];           //oder &var[0][0]
    for(int i=0;i<2*3;++i)
        cout << pvar[i] << endl;

    return 0;
}
```

# Repetition

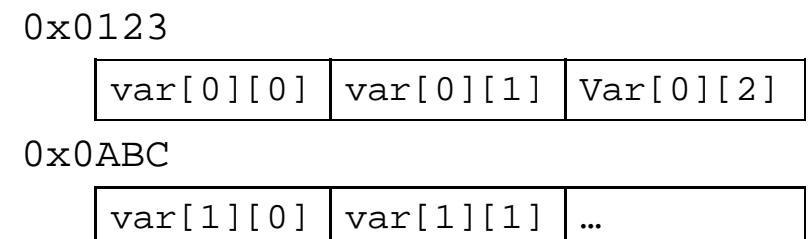
- Arrays und Pointer
  - Dynamisch alloziert (flexibler):

```
double** var1 = new double*[2];
var1[0] = new double[3];
var1[1] = new double[5];
```



- Kombination:

```
double* var1[2];
var1[0] = new double[3];
var1[1] = new double[5];
```



# Repetition

## ■ Verkettete Listen

```
int main() {
    Mannschaft Bundesliga[4] =
{{ "FCB",18,9,4,65,28,NULL,NULL},
 {"S04",18,7,6,52,29,NULL,NULL},
 {"Bremen",15,9,7,67,39,NULL,NULL},
 {"Bayer04",14,12,5,60,36,NULL,NULL}};

    Mannschaft tmp = Bundesliga[3];
    Bundesliga[3] = Bundesliga[2];
    Bundesliga[2] = tmp;
    return 0;
}
```

```
struct Mannschaft {
    char* Name;
    unsigned int g,u,v;
    unsigned int ToreG, ToreB;
    ...
    Mannschaft* next;
    Mannschaft* prev;
};
```

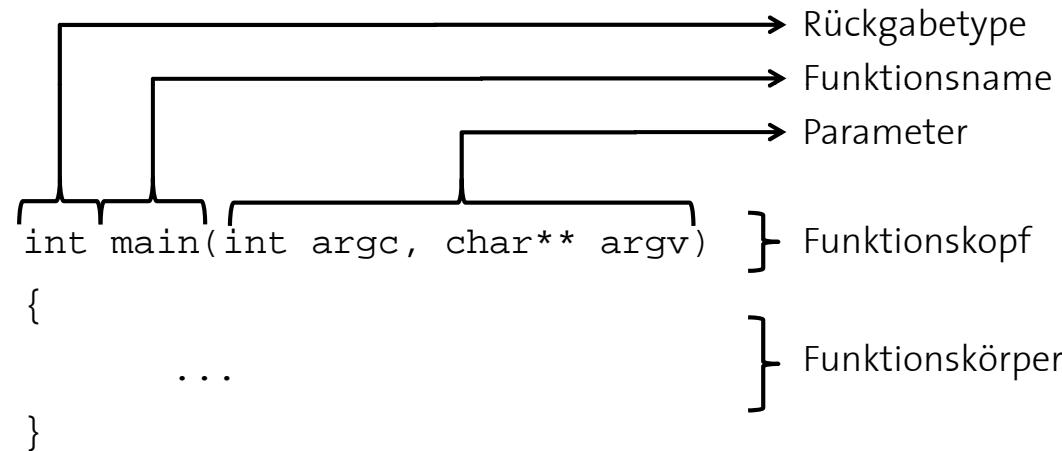
```
int main() {
    Mannschaft *Bundesliga = new Mannschaft;
    Bundesliga->Name = "FCB";
    Bundesliga->prev = NULL;
    Bundesliga->next = new Mannschaft;
    Bundesliga->next->Name = "S04";
    Bundesliga->next->prev = Bundesliga;
    Bundesliga->next->next = new Mannschaft;
    Bundesliga->next->next->Name = "Bremen";
    Bundesliga->next->next->prev = Bundesliga->next;
    Bundesliga->next->next->next = NULL;

    Mannschaft* ptr = Bundesliga->next;
    Bundesliga->next = Bundesliga->next->next;
    Bundesliga->next->prev = Bundesliga;
    ptr->next = NULL;
    ptr->prev = Bundesliga->next;
    Bundesliga->next->next = ptr;

    return 0;
}
```

# Funktionen

- Auch: Subroutine, Subprogramm
- Aufbau:



- Warum: Modularisierung, Zusammenfassen von Code-Snippets (z.B. Norm eines Vektors)

# Funktionen

## ■ Beispiele

```
#include <iostream>

using namespace std;

void begruessung() {
    cout << "Hoi zaeme!" << endl;
}

int main() {
    begruessung();
    return 0;
}
```

```
#include <iostream>
#include <cmath>

using namespace std;

double norm2(double* a, int len) {
    double retVal = 0.0;
    for(int i=0;i<len;++i)
        retVal += *a**a++;
    return retVal;
}

int main() {
    double var1[] = {3.0, 4.0};
    cout << norm2(var1, 2);
    return 0;
}
```

# Funktionen

- Definition (Prototyp) und Deklaration
  - Definition
    - Funktionskopf
    - Keine Parameternamen
    - Strichpunkt
    - Bsp: double norm2(double\*, int);
  - Deklaration (wie üblich)

# Funktionen

## ■ Beispiele

```
#include <iostream>

using namespace std;

void begruessung() {
    cout << "Hoi zaeme!" << endl;
}

int main() {
    begruessung();
    return 0;
}
```

```
#include <iostream>

using namespace std;

void begruessung();

int main() {
    begruessung();
    return 0;
}

void begruessung() {
    cout << "Hoi zaeme!" << endl;
}
```

# Funktionen

- Call-by-Value
  - Vorteil: sehr einfach
  - Nachteil: große Speicherbereiche werden unter Umständen (auf den Stack) kopiert
- Lösung:
  - Übergabe von Pointern (Effektiv Call-by-Value mit Pointer)
  - Call-by-Reference

# Referenzen

- Mehrere Namen für eine Variable (Synonyme)

- Beispiel:

```
int n, m = 5;  
int& nr = n;  
int& nr2;      //error: references need to point to an object  
nr = m;        //m = 5
```

- Müssen initialisiert werden und können ihren Wert (Stelle auf die sie zeigen) nicht mehr ändern
- (Referenzen sind ähnlich zu const pointern)

# Referenzen

## ■ Beispiel (Pointer vs. Reference)

```
void swap(int *v1, int *v2) {  
    int temp = *v1;  
    *v1 = *v2;  
    *v2 = temp;  
}  
  
int a = 4;  
swap(&a, NULL)      //error
```

```
void swap(int &v1, int &v2) {  
    int temp = v1;  
    v1 = v2;  
    v2 = temp;  
}  
  
//references always point to objects  
//a little safer?
```

- Ein const pointer kann auf NULL zeigen!
- ABER: ungültige Referenzen können auch erzeugt werden, z.B.:

```
int* p;  
p = NULL;  
int& r = *p;
```

# Referenzen

## ■ Beispiele

```
void incr(int& y) {  
    y++;  
}
```

```
void incr(int* y) {  
    (*y)++;  
}
```

```
void incr(int y) {  
    y++;  
}
```

```
int incr(int y) {  
    return y++;  
}
```

```
int incr(int y) {  
    return ++y;  
}
```

```
incr(5);  
//Kompilerfehler
```

```
int i = 5;  
incr(&i++);  
//Kompilerfehler
```

```
int i = 5;  
incr(i);  
// i ist 5
```

```
int i = 5;  
i = incr(i);  
// i ist 5
```

```
int i = 5;  
i = incr(i);  
// i ist 6
```

Prefix: T1& operator ++(T1& a); //++a

Postfix: T1 operator ++(T1& a, int); //a++

```
int i = 5;  
incr(i++);  
//Kompilerfehler
```

```
int i = 5;  
incr(&++i);  
// i ist 7
```

int i = 5;  
incr(i++);  
//Kompilerfehler

int i = 5;  
incr(&++i);  
// i ist 7

# Referenzen

## ■ Beispiele

```
void foo(int* &a) {  
    a = new int[2];  
}
```

## Mit Pointern?

```
void foo(int** a) {  
    *a = new int[2];  
}
```

# Funktionen

- Übergabe von Arrays durch Pointer
- Vorbelegen von Parametern

```
int summe(int* arr, int offset = 0, int len=5) {  
    ...  
}  
  
/*double summe2(double* arr, int offset = 0, int len) {      //Kompilererror!!!  
    ...  
}*/  
  
int a[5] = {1, 2, 3, 4, 5};  
int b1 = summe(a);  
int b2 = summe(a, 0);  
int b3 = summe(a, 0, 5);
```

# Funktionen

## ■ Überladen

```
int summe(int* a, int offset = 0, int len = 5);  
double summe(double *a, double offset = 0, int len = 5);  
double summe(int* a, int offset, int len);//Kompilererror!
```

## ■ Inline

```
double inline summe(...);  
//oder: inline double summe(...);
```

# Übungen

## ■ Aufgabe 1: Funktionsaufrufe I

## ■ Aufgabe 2: Funktionsaufrufe II

```
void f(int a, int* b, int& c) {  
    ...  
}  
  
int k, l, m;  
f(1, 2, 3);           //Fehler  
f(k, &l, m);         //Ok  
f(k, &l, m+1);       //Fehler  
f(k, &l, l*m);       //Fehler  
f(k, &l, m++);       //Fehler  
f(k, &l, ++m);       //Ok  
f(k, &l, m+k);       //Fehler
```

# Übungen

- Aufgabe 3: Programmierübung
- Aufgabe 4: Raytracer  
typedef wird benutzt um einen alternativen Namen für einen type zu definieren, z.B.:

```
typedef  unsigned int uint;

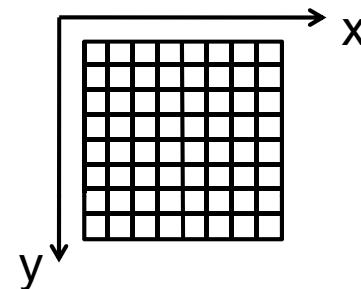
int main() {
    uint a, b, c = 1;
    ...
}
```

# Übungen

## Aufgabe 4: Raytracer (a)

```
int main() {  
    Vector3f pos(-4,0,1);  
    Vector3f lookat(0,0,1);  
    Vector3f up(0,0,1);  
    camera_model camera;  
    camera = init_camera(pos, lookat, up,1);  
  
    color_rgb rot(1,0,0);  
    color_rgb gruen(0,1,0);  
    color_rgb blau(0,0,1);  
    color_rgb gelb(1,1,0);  
    color_rgb schwarz(0,0,0);  
    color_rgb weiss(1,1,1);  
  
    return 0;  
}
```

Bild:



```
{  
    color_rgb *bild = new color_rgb[res_y*res_x];  
    for(int y=0;y<res_y;++y) {  
        for(int x=0;x<res_x;++x) {  
            bild[y*res_x+x] = blau;  
        }  
    }  
    write_picture(bild);  
    delete bild;
```

# Übungen

## ■ Aufgabe 4: Raytracer (b)

$$\vec{n} \cdot \vec{x} - s = 0$$

$$\vec{x} = \vec{p} + t\vec{d}$$

$$\rightarrow t = \frac{s - \vec{n} \cdot \vec{p}}{\vec{n} \cdot \vec{d}}$$

```
float int_plane(ray strahl, Vector3f &normale) {  
    Vector3f plane_n(0,0,1);  
    float s=0;  
    normale = plane_n;  
    return (s - ...) / ...  
}
```

Memberfunktion der Klasse Vector3f: dot  
Membervariablen der struct ray: p und d

```
double np = plane_n.dot(strahl.p);           //für dot-Product
```

# Übungen

## ■ Aufgabe 4: Raytracer (c)

```
ray strahl;  
strahl = generate_ray(x, ...);  
color_rgb farbe = schwarz;  
t_closest = 1e30;  
t = int_plane(strahl, normale)  
if(t>0 && t<t_closest) {  
    farbe = gelb;  
    tclosest = t;  
}
```

## ■ Aufgabe 4: Raytracer (d, e)

