A practical guide to C++

Janis Fehr
fehr@informatik.uni-freiburg.de

SommerCampus2004

Kursübersicht:

- **Kursmaterial:** http://www.informatik.uni-freiburg.de/~fehr/cpp/cpp.html
  - Folien
  - Source Code
  - Resourcen
- **Literatur:**
  - Bjarne Stroustrup: C++
  - Dirk Louis: C/C++
- **Gruppenarbeit:**
  - 2-3er Teams
- **Kursablauf:**
  - Vortragseinheiten
  - Übungen
  - Projektarbeit
- **Projekt:**
  - Steganographie

Projektübersicht:

Steganographie: Verstecke Informationen in anderen Medien

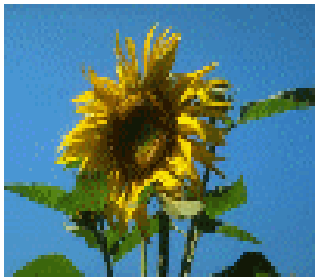
Funkspruch:

"Apparently neutral s protest is thoroughly discounted and ignored. Isman hard hit. Blockade issue affects pretext for embargo on by-products, ejecting suets and vegetables oils."

"Apparently neutral s protest is thoroughly discounted and ignored. Isman hard hit. Blockade issue affects pretext for embargo on by-products, ejecting suets and vegetables oils."

Pershing sails from NY June 1.

Mit Bildern:

Text →  → Text

Agenda:

- **Teil 1**
  - Geschichte von C++
  - Einführung
  - Speichermanagement
  - Zeiger
  - Streams
- **Teil 2**
  - Debugging
  - OOP
- **Teil 3**
  - C++ Standard Template Library
  - Templates
  - Exeptions
- **Teil 4**
  - Qt
  - Projekt

A, B, C ... die Geschichte

- Vor C kam B, vor B kam ... BCPL (Martin Richard)
- 1970: Ken Tomson arbeitet an der Entwicklung von Unix -> B
- 1972: Dennis Ritchie entwickelt B weiter zur ersten Version von C
- 1985: Bjarne Stroustrup erweitert C um OOP -> C++
- 1989: Ansi C
- 1998: Ansi C++

"Hello World!"

```
//HelloWorld, Janis Fehr
#include<stdlib.h>
#include<iostream>

int main()
{
  std::cout<<''Hello World!\n'';
  return 0;
}
```

Iostream Bibliothek einbinden

main Methode

Ausgabe Stream der Standard Biblithek

Kompelieren: g++ hello_world.cc -o hello_world

Ausführen: ./hello_word

Parameterübergabe

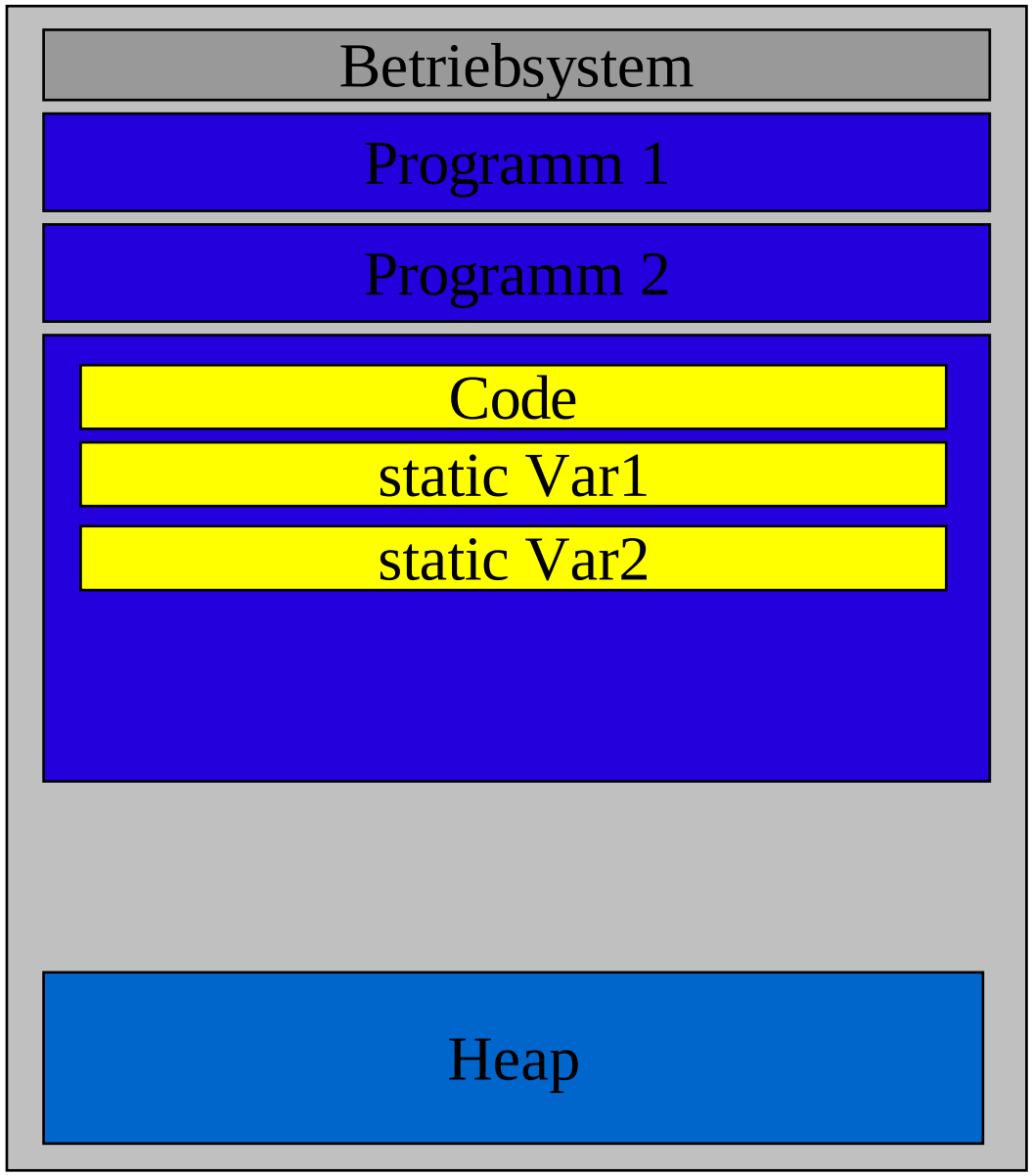
```
/*HelloWorld2, Janis Fehr */
#include<stdlib.h>
#include<iostream>

int main(int argc, char* argv[])
{
  for(int i=0;i<argc;i++)
  {
     std::cout<<argv[i]<<''\n'';
  }
  return 0;
}
```

Übung: schreibt einen kleinen Taschenrechner:

```
./rechner + 3 6
```

Tip: verwendet **atoi()** oder **atof()**

Speicher Management



Speicher Management 2

Kontrolle über den Speicher bedeutet Macht !

- Nutze Sie um Gutes zu tun
- Oder um grandios zu scheitern :-)

C/C++ gibt dem Programmierer sehr viel Macht

- Man sollte aber wissen was man tut !

z.B.: es gibt keinen GarbageCollector in C++ !!!

Gültigkeitsbereiche von Variablen

Gültigkeitsbereiche von Variablen

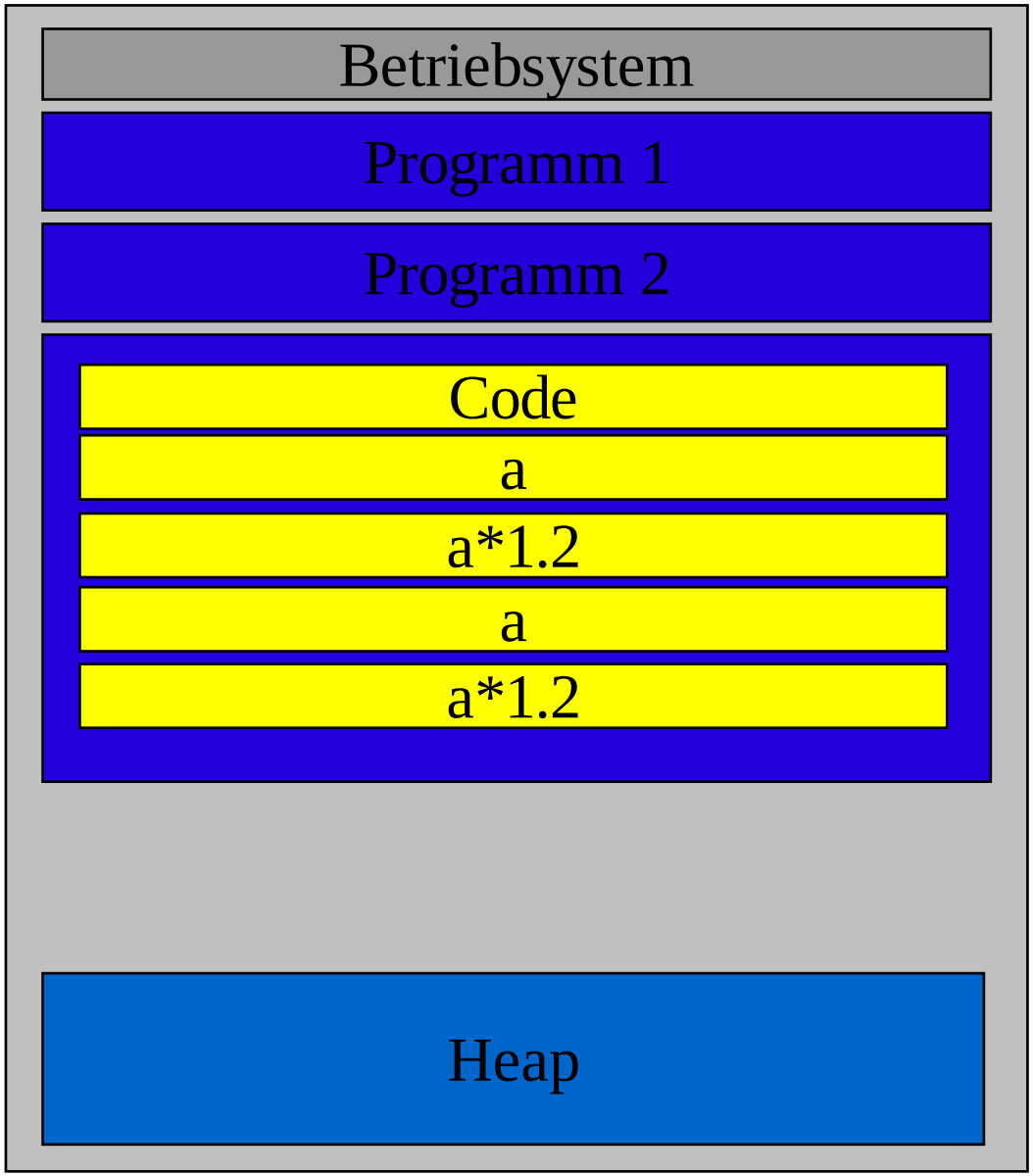
```
int foo(int a, double b)
{
   int B;
   int A = a;
   B=A;
   return B;
}

int main()
{
   int a = 5;
   int c = foo(a,a*1.2);
   return c;
}
```

Bei "statischen" Variablen ist der Gültigkeitsbereich intuitiv

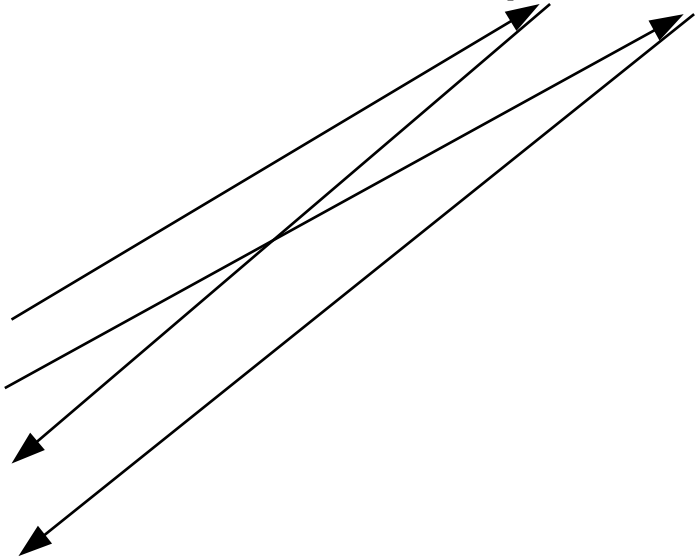
Globale Variablen: `global int myvar=5;`

Dynamische Variablen werden vom Programmierer kontrolliert !

Funktionsaufrufe



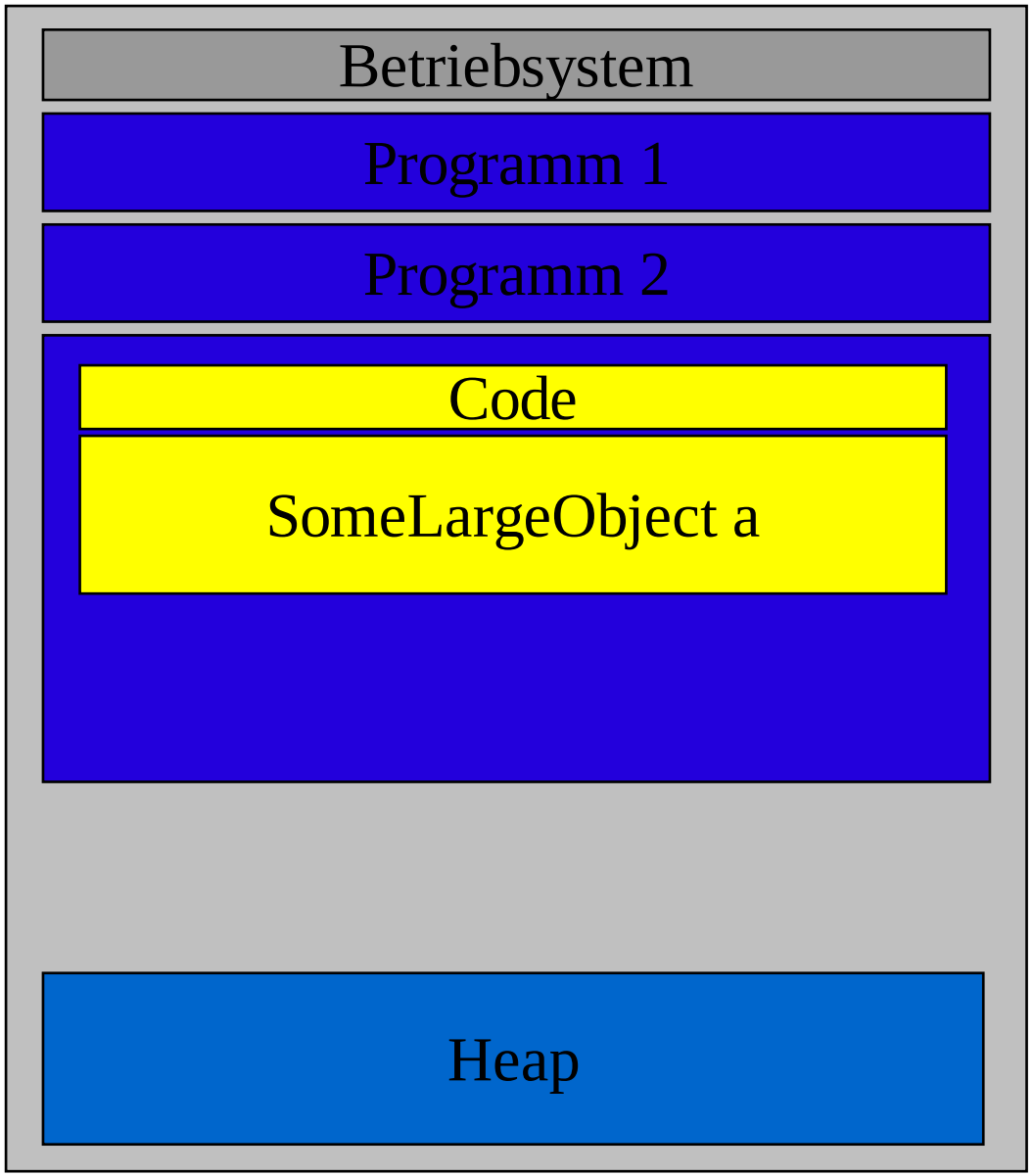

```
int a = 5;
int c = foo(a,a*1.2);
```



```
SomeLargeObject a;
foo(SomeLargeObject b);
```

Speicherverschwendung !!!

Referenzen:




```
SomeLargeObject a;
foo(SomeLargeObject &b);
```

- Keine Kopie
- Nur ein Objekt
- Veränderungen bleiben bestehen!

Siehe **foo1.cc**

const

CONST hat unterschiedliche, aber sehr wirkungsvolle Funktionen, je nachdem in welchem Kontext es auftritt:

- `const int a =5;` — Konstante
- `int foo(double b) const;` — Funktion die private Variablen nicht Verändern darf
- `int foo(const double &b);` — bei Referenzen: verhindert Überschreiben.

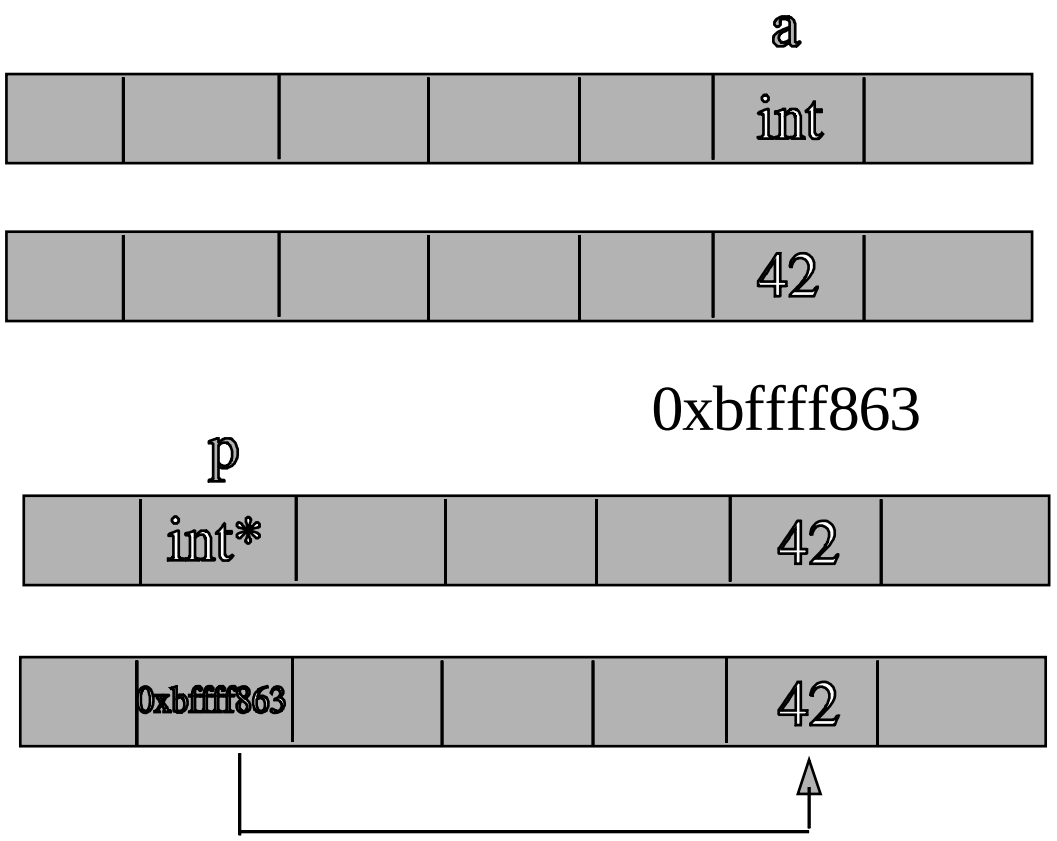
Dynamisch allokierter Speicher

- Große Objekte sollen nicht auf den Stack
- Man will die Größe der Objekte zur Laufzeit ändern (z.B. Arrays)
- Volle Kontrolle über den Speicher (Optimierung)
- Direktes Ansprechen von Speicheradressen
- Direkte Kommunikation mit der Hardware

---> ZEIGER

Zeiger (pointer)

```
int a;

a = 42;

std::cout<<&a;

int* p;

p = &a;
```




```
std::cout<<*p;
```

42

```
std::cout<<''an der Adresse''<<p<<''steht der Wert''<<*p;
```

Was ist das Ergebniss ?

```
int a = 42;
int* p = &a;
*p = 27;
std::cout<<a<<std::endl;
```

```
int a = 42;
int b = 137;
int* c = &a;
int* d = &b;
c=d;
std::cout<<a<<std::endl;
std::cout<<*c<<std::endl;
```

```
int a = 42;
int b = 137;
int* c = &a;
int** d = &c;
std::cout<<*d<<std::endl;
```

Dynamischer Zugriff auf den Heap

Objekte werden in C++ mit **new** auf dem Heap erzeugt.

```
int* a = new int;
*a = 5;

std::cout<<*a;
```

Der Speicher der Objekte wird mit **delete** wieder freigegeben.

```
delete a;
```

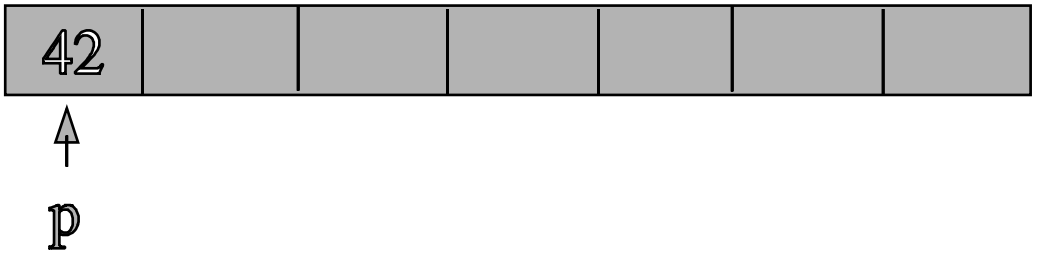
Achtung: "Heap-Leichen" können das System in die Knie zwingen.

Zeiger und Felder

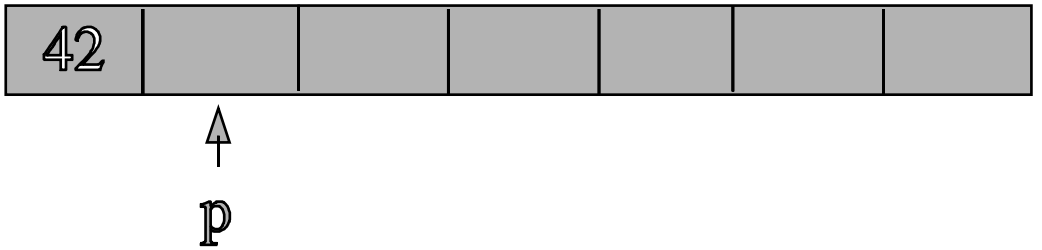
```
int* feld = new int[5];
```

```
feld[0] = 42;
```

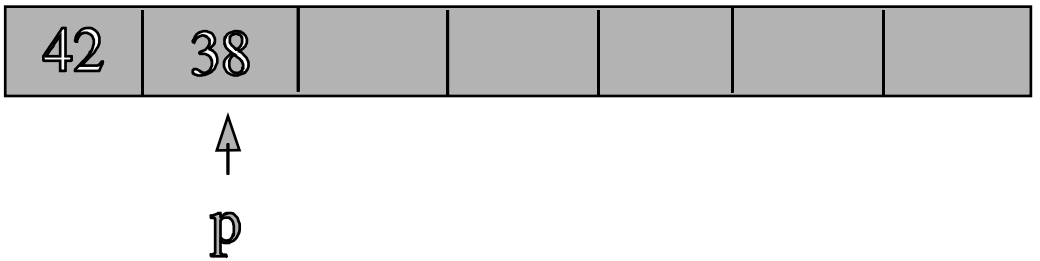

```
int* p = feld;
```




```
p++;
```




```
*p = 38;
```



Zeiger und Felder

- Ein Feld wird nur durch einen Zeiger auf das erste Ellement dargestellt
  - Optimale Performance
  - Optimaler Speicherverbrauch
  - Der Programmierer muss sich selbst die Größe merken !!!

Segmentation Fault

Übung: entwickelt ein 5x5 Feld, und setzt (4,3) auf den Wert 42.

Streams

Der übliche Weg der Ein- und Ausgabe in C++ ist über Streams. Wir haben schon den Stream auf die Standardausgabe kennen gelernt:

```
std::cout<<''Hallo''<< someVar <<''\n'';
```

Aber Streams werden auch zum Lesen und Schreiben von Files genutzt, oder für beliebige andere I/O Ereignisse.

Durch "überladen" des << Operators kann man sich beliebige Streams erstellen

File I/O

Lesen einer ASCII Datei ist sehr einfach:

```
#include<stdlib.h>
#include<iostream>
#include<fstream>   //Lib mit den Filestreams
#include<string>    //stings sind kein typ

int main()
{
   std::ifstream file(''data.txt'');//öffne data.txt zum Lesen
   std::string word;

   while(!file.eof()) //file ende?
   {
      file >> word; //stream Zeichen bis zun nächsten Leerzeichen
      std::cout<<word; //Bildschirmausgabe
   }
   return 0
}
```

Mit `std::ofstream` lassen sich analog Daten in Files streamen.

Überladen von Operatoren:

Beispiel: Klasse der komplexen Zahlen : definiere die Adition +=

```
Class MyComplex
{
    public:
        MyComplex(float real,float im);
        ...
        MyComplex operator+=(const MyComplex &v)
        {
            real += v.real;
            im += v.im;
        }
    private:
        float real;
        float im;
};
```

Aufgabe:

Schreibt ein Programm, welches das Bild **tiger.ppm** einliest, und die Pixelwerte in drei 2D Feldern, jedes Feld für einen Farbkanal, abspeichert.

Hinweise:

Das PPM-Format ist sehr simpel. Nach 4 zeilen Header folgt das Bild zeilenweise in ASCII. Dabei ist jeder Pixel durch 3 Werte (rot,grün,blau) dargestellt, die jeweils durch eine Leerstelle getrennt sind.

Weitere Infos zu PPM online unter Resourcen.