

TECHNISCHE UNIVERSITÄT DORTMUND

Wintersemester 2006/07

**Einführung in die Informatik für
Naturwissenschaftler und Ingenieure**
(alias Einführung in die Programmierung)
(Vorlesung)

Prof. Dr. Günter Rudolph
Fachbereich Informatik
Lehrstuhl für Algorithm Engineering

Kapitel 5: Funktionen

Inhalt

- Funktionen
 - mit / ohne Parameter
 - mit / ohne Rückgabewerte
- Übergabemechanismen
 - Übergabe eines Wertes
 - Übergabe einer Referenz
 - Übergabe eines Zeigers
- Programmieren mit Funktionen → heute
 - + static / inline / MAKROS

Rudolph: EINI (WS 2007/08) • Kap. 5: Funktionen 2

Kapitel 5: Funktionen

Statische Funktionen (in dieser Form: Relikt aus C)
sind Funktionen, die nur für Funktionen in derselben Datei sichtbar (aufrufbar) sind!

Funktionsdeklaration:
`static` Datentyp Funktionsname(Datentyp Bezeichner);

```
#include <iostream>
using namespace std;
static void funktion1() {
  cout << "F1" << endl;
}
void funktion2() {
  funktion1();
  cout << "F2" << endl;
}
Datei Funktionen.cpp
```

```
void funktion1();
void funktion2();
int main() {
  funktion1();
  funktion2();
  return 0;
}
Datei Haupt.cpp
```

Fehler! funktion1 nicht sichtbar!

wenn entfernt, dann gelingt Compilierung:
g++ *.cpp -o test

Rudolph: EINI (WS 2007/08) • Kap. 5: Funktionen 3

Kapitel 5: Funktionen

Inline Funktionen
sind Funktionen, deren Anweisungsteile an der Stelle des Aufrufes eingesetzt werden

Funktionsdeklaration:
`inline` Datentyp Funktionsname(Datentyp Bezeichner);

```
#include <iostream>
using namespace std;
inline void funktion() {
  cout << "inline" << endl;
}
int main() {
  cout << "main" << endl;
  funktion();
  return 0;
}
```

→ wird zur Übersetzungszeit ersetzt zu:

```
#include <iostream>
using namespace std;
int main() {
  cout << "main" << endl;
  cout << "inline" << endl;
  return 0;
}
```

Rudolph: EINI (WS 2007/08) • Kap. 5: Funktionen 4

Kapitel 5: Funktionen

Inline Funktionen

Vorteile:

1. Man behält alle positiven Effekte von Funktionen:
 - Bessere Lesbarkeit / Verständnis des Codes.
 - Verwendung von Funktionen sichert einheitliches Verhalten.
 - Änderungen müssen einmal nur im Funktionsrumpf durchgeführt werden.
 - Funktionen können in anderen Anwendungen wieder verwendet werden.
2. Zusätzlich bekommt man schnelleren Code!
(keine Sprünge im Programm, keine Kopien bei Parameterübergaben)

Nachteil:

Das übersetzte Programm wird größer (benötigt mehr Hauptspeicher)

Deshalb: vorangestelltes `inline` ist nur eine Anfrage an den Compiler! Keine Pflicht!

Kapitel 5: Funktionen

„Inline-Funktionsartiges“ mit Makros

Da müssen wir etwas ausholen ...

```
#include <iostream>
int main() {
    int x = 1;
    std::cout << x*x;
    return 0;
}
```



ersetzt Makros (beginnen mit #):
z.B. lädt Text aus Datei `iostream.h`

```
#define Makroname Ersetzung
```

Bsp:

```
#define MAX_SIZE 100
#define ASPECT_RATIO 1.653
```

Makronamen im Programmtext werden vom Prä-Compiler durch ihre Ersetzung ersetzt

Kapitel 5: Funktionen

```
#define MAX_SIZE 100
void LeseSatz(char *Puffer) {
    char c = 0;
    int i = 0;
    while (i < MAX_SIZE && c != '\.') {
        cin >> c;
        *Puffer++ = c;
    }
}
```

```
void LeseSatz(char *Puffer) {
    char c = 0;
    int i = 0;
    while (i < 100 && c != '\.') {
        cin >> c;
        *Puffer++ = c;
    }
}
```

Makros ...
dieser Art sind Relikt aus C!

Nach Durchlauf durch den Prä-Compiler

Tip: NICHT VERWENDEN!
stattdessen:
`int const max_size = 100;`

Kapitel 5: Funktionen

„Inline-Funktionsartiges“ mit Makros

```
#define SQUARE(x) x*x    Vorsicht: SQUARE(x+3) ergibt: x+3*x+3
```

besser:

```
#define SQUARE(x) (x)*(x) → SQUARE(x+3) ergibt: (x+3)*(x+3)
```

noch besser:

```
#define SQUARE(x) ((x)*(x)) → SQUARE(x+3) ergibt: ((x+3)*(x+3))
```

auch mehrere Parameter möglich:

```
#define MAX(x, y) ((x)>(y)?(x):(y))
```

```
int a = 5;
int z = MAX(a+4, a+a);    ergibt: int a = 5;
                           int z = ((a+4)>(a+a)?(a+4):(a+a));
```

Nachteil:
ein Ausdruck wird **2x** ausgewertet!



„Inline-Funktionsartiges“ mit Makros (Relikt aus C)

Beliebiger Unsinn möglich ...

```
// rufe Funktion fkt() mit maximalem Argument auf  
#define AUFRUF_MIT_MAX(x,y) fkt(MAX(x,y))
```

„Makros wie diese haben so viele Nachteile,
dass schon das Nachdenken über sie nicht zu ertragen ist.“

Scott Meyers: Effektiv C++ programmieren, S. 32, 3. Aufl., 2006.

```
int a = 5, b = 0;  
AUFRUF_MIT_MAX(++a, b); // a wird 2x inkrementiert  
AUFRUF_MIT_MAX(++a, b+10); // a wird 1x inkrementiert
```

Tip: statt funktionsartigen Makros besser richtige inline-Funktionen verwenden!