

F Funktionen und Programmstruktur

- Funktionen
- Gültigkeitsbereiche und Lebensdauer von Variablen
- Getrennt Übersetzung von Programmteilen
- C-Preprozessor
 - ◆ Include-Dateien
 - ◆ Makros

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.1

Reproduktion jeder Art oder Vervielfachung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.1 Funktionen (2)

- ↳ Funktionen dienen der Abstraktion
- Name und Parameter abstrahieren
 - vom tatsächlichen Programmstück
 - von der Darstellung und Verwendung von Daten
- Verwendung
 - ◆ mehrmals benötigte Programmstücke können durch Angabe des Funktionsnamens aufgerufen werden
 - ◆ Schrittweise Abstraktion (**Top-Down-** und **Bottom-Up-**Entwurf)

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.3

Reproduktion jeder Art oder Vervielfachung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.1 Funktionen

- **Funktion =**
Programmstück (Block), das mit einem **Namen** versehen ist und dem zum Ablauf **Parameter** übergeben werden können
- Funktionen sind die elementaren Bausteine für Programme
 - ↳ gliedern umfangreiche, schwer überblickbare Aufgaben in kleine Komponenten
 - ↳ erlauben die Wiederverwendung von Programmkomponenten
 - ↳ verbergen Implementierungsdetails vor anderen Programmteilen (**Black-Box-Prinzip**)

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.2

Reproduktion jeder Art oder Vervielfachung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.1 Funktionen (3) — Beispiel Sinusberechnung

```
#include <stdio.h>
#include <math.h>

double sinus (double x)
{
    double summe;
    double x_quadrat;
    double rest;
    int k;

    k = 0;
    summe = 0.0;
    rest = x;
    x_quadrat = x*x;

    while ( fabs(rest) > 1e-9 ) {
        summe += rest;
        k += 2;
        rest *= -x_quadrat/(k*(k+1));
    }
    return(summe);
}
```

```
main()
{
    double wert;

    printf("Berechnung des Sinus von ");
    scanf("%lf", &x);
    printf("sin(%lf) = %lf\n",
           wert, sinus(wert));
}
```

- beliebige Verwendung von `sinus` in Ausdrücken:

```
y = exp(tau*t) * sinus(f*t);
```

C-Ing

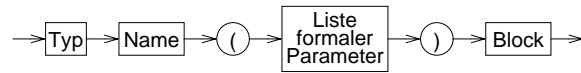
Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.4

Reproduktion jeder Art oder Vervielfachung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.2 Funktionsdefinition



■ Typ

- ◆ Typ des Werts, der am Ende der Funktion als Wert zurückgegeben wird
- ◆ beliebiger Typ
- ◆ `void` = kein Rückgabewert

■ Name

- ◆ beliebiger Bezeichner, kein Schlüsselwort

C-Ing

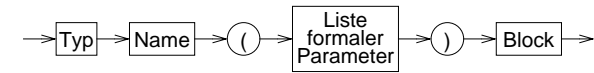
Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.5

Reproduktion jeder Art oder Vervielfachung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.2 Funktionsdefinition (3)



■ Block

- ◆ beliebiger Block
- ◆ zusätzliche Anweisung

```
return ( Ausdruck );
```

oder

```
return;
```

bei `void`-Funktionen

- Rückkehr aus der Funktion: das Programm wird nach dem Funktionsaufruf fortgesetzt
- der Typ des Ausdrucks muß mit dem Typ der Funktion übereinstimmen
- die Klammern können auch weggelassen werden

C-Ing

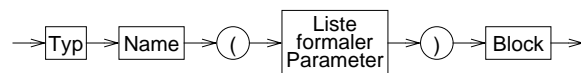
Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.7

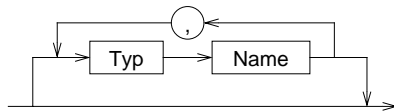
Reproduktion jeder Art oder Vervielfachung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.2 Funktionsdefinition (2)



■ Liste formaler Parameter

- ◆ **Typ:** beliebiger Typ
- ◆ **Name:** beliebiger Bezeichner
- ◆ die formalen Parameter stehen für die Werte, die beim Aufruf an die Funktion übergeben wurden (= **aktuelle Parameter**)
- ◆ die formalen Parameter verhalten sich wie Variablen, die im **Funktionsrumpf** definiert sind und mit den aktuellen Parametern vorbelegt werden



C-Ing

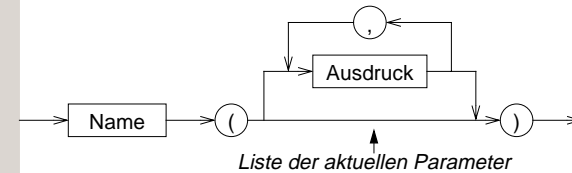
Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.6

Reproduktion jeder Art oder Vervielfachung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.3 Funktionsaufruf



- Jeder Funktionsaufruf ist ein Ausdruck
- `void`-Funktionen können keine Teilausdrücke sein
- ◆ wie Prozeduren in anderen Sprachen (z. B. Pascal)

C-Ing

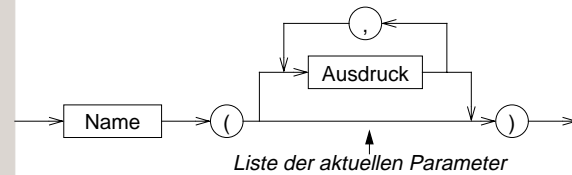
Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.8

Reproduktion jeder Art oder Vervielfachung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.3 Funktionsaufruf (2)



- Die Ausdrücke in der Parameterliste werden ausgewertet, **bevor** in die Funktion gesprungen wird
 ↳ **aktuelle Parameter**
- Anzahl und Typen der Ausdrücke in der Liste der aktuellen Parameter müssen mit denen der formalen Parameter in der Funktionsdefinition übereinstimmen
- Die Auswertungsreihenfolge der Parameterausdrücke ist **nicht** festgelegt

F.5 Funktionen — Regeln

- Funktionen werden global definiert
 ↳ keine lokalen Funktionen/Prozeduren wie z. B. in Pascal
- `main()` ist eine normale Funktion, die aber automatisch als erste beim Programmstart aufgerufen wird
- rekursive Funktionsaufrufe sind zulässig
 ↳ eine Funktion darf sich selbst aufrufen (z. B. zur Fakultätsberechnung)

```
fakultaet(int n)
{
    if ( n == 1 )
        return(1);
    else
        return( n * fakultaet(n-1) );
}
```

F.4 Funktionen — Beispiel

```
float power (float b, int e)
{
    float prod = 1.0;
    int i;

    for (i=1; i <= e; i++)
        prod *= b;
    return(prod);
}
```

```
float x, y;
y = power(2+x,4)+3;
```

≡

```
float x, y, power;
{
    float b = 2+x;
    int e = 4;
    float prod = 1.0;
    int i;

    for (i=1; i <= e; i++)
        prod *= b;
    power = prod;
}
y=power+3;
```

F.5 Funktionen — Regeln (2)

- Funktionen müssen **deklariert** sein, bevor sie aufgerufen werden
 = Rückgabtyp und Parametertypen müssen bekannt sein
 ♦ durch eine Funktionsdefinition ist die Funktion automatisch auch deklariert
- wurde eine verwendete Funktion vor ihrer Verwendung nicht deklariert, wird automatisch angenommen
 ▶ Funktionswert vom Typ `int`
 ▶ 1 Parameter vom Typ `int`
 ↳ **schlechter Programmierstil → fehleranfällig**

F.5 Funktionen — Regeln (2)

- Funktionsdeklaration
 - ◆ soll eine Funktion vor ihrer Definition verwendet werden, kann sie durch eine **Deklaration** bekannt gemacht werden
 - ◆ Syntax:

```
Typ Name ( Liste formaler Parameter );
```

 - ▶ Parameternamen können weggelassen werden, die Parametertypen müssen aber angegeben werden!
 - ◆ Beispiel:

```
double sinus(double);
```

F.7 Parameterübergabe an Funktionen

- allgemein in Programmiersprachen vor allem zwei Varianten:
 - ▶ call by value
 - ▶ call by reference

1 call by value

- Normalfall in C
- Es wird eine Kopie des aktuellen Parameters an die Funktion übergeben
 - ↳ die Funktion kann den Übergabeparameter durch Zugriff auf den formalen Parameter lesen
 - ↳ die Funktion kann den Wert des formalen Parameters (also die Kopie!) ändern, ohne daß dies Auswirkungen auf den Wert des aktuellen Parameters beim Aufrufer hat
 - ↳ die Funktion kann über einen Parameter dem Aufrufer keine Ergebnisse mitteilen

F.6 Funktionsdeklarationen — Beispiel

```
#include <stdio.h>
#include <math.h>

double sinus(double);
/* oder: double sinus(double x); */

main()
{
    double wert;

    printf("Berechnung des Sinus von ");
    scanf("%lf", &wert);
    printf("sin(%lf) = %lf\n",
        wert, sinus(wert));
}

double sinus (double x)
{
    double summe;
    double x_quadrat;
    double rest;
    int k;

    k = 0;
    summe = 0.0;
    rest = x;
    x_quadrat = x*x;

    while ( fabs(rest) > 1e-9 ) {
        summe += rest;
        k += 2;
        rest *= -x_quadrat/(k*(k+1));
    }
    return(summe);
}
```

2 call by reference

[F.7 Parameterübergabe an Funktionen](#)

- In C nur indirekt mit Hilfe von Zeigern realisierbar
- Der Übergabeparameter ist eine Variable und die aufgerufene Funktion erhält die Speicheradresse dieser Variablen
 - ↳ die Funktion kann den Übergabeparameter durch Zugriff auf den formalen Parameter lesen
 - ↳ wenn die Funktion den Wert des formalen Parameters verändert, ändert sie den Inhalt der Speicherzelle des aktuellen Parameters
 - ↳ auch der Wert der Variablen (aktueller Parameter) beim Aufrufer der Funktion ändert sich dadurch

F.8 C-Preprozessor

1 Kurzüberblick

- bevor eine C-Quelle dem C-Compiler übergeben wird, wird sie durch einen Makro-Preprozessor bearbeitet
- Anweisungen an den Preprozessor werden durch ein #-Zeichen am Anfang der Zeile gekennzeichnet
- die Syntax von Preprozessoranweisungen ist unabhängig vom Rest der Sprache
- Preprozessoranweisungen werden nicht durch ; abgeschlossen!
- wichtigste Funktionen:
`#define` Definition von Makros
`#include` Einfügen von anderen Dateien

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.17

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.8 C-Preprozessor

2 Einfügen von Dateien

- `#include` fügt den Inhalt einer anderen Datei in eine C-Quelldatei ein
- Syntax:

```
#include <Dateiname >  
oder  
#include "Dateiname "
```
- mit `#include` werden *Header-Dateien* mit Daten, die für mehrere Quelldateien benötigt werden einkopiert
 - Deklaration von Funktionen, Strukturen, externen Variablen
 - Definition von Makros
- wird `Dateiname` durch `< >` geklammert, wird eine **Standard-Header-Datei** einkopiert
- wird `Dateiname` durch `" "` geklammert, wird eine Header-Datei des Benutzers einkopiert (vereinfacht dargestellt!)

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.19

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Makrodefinitionen

F.8 C-Preprozessor

- Makros ermöglichen einfache textuelle Ersetzungen (parametrierbare Makros werden später behandelt)
- ein Makro wird durch die `#define`-Anweisung definiert
- Syntax:

```
#define Makroname Ersatztext
```
- eine Makrodefinition bewirkt, daß der Preprozessor im nachfolgenden Text der C-Quelle alle Vorkommen von ***Makroname*** durch ***Ersatztext*** ersetzt
- Beispiel:

```
#define EOF -1
```

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.18

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.9 Programmstruktur & Module

1 Softwaredesign

- Grundsätzliche Überlegungen über die Struktur eines Programms **vor** Beginn der Programmierung
- Verschiedene Design-Methoden
 - ◆ Top-down Entwurf / Prozedurale Programmierung
 - traditionelle Methode
 - bis Mitte der 80er Jahre fast ausschließlich verwendet
 - an Programmiersprachen wie Fortran, Cobol, Pascal oder C orientiert
 - ◆ Objekt-orientierter Entwurf
 - moderne, sehr aktuelle Methode
 - Ziel: Bewältigung sehr komplexer Probleme
 - auf Programmiersprachen wie C++, Smalltalk oder Java ausgerichtet

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.20

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Top-down Entwurf

F.9 Programmstruktur & Module

Zentrale Fragestellung

- ◆ was ist zu tun?
- ◆ in welche Teilaufgaben lässt sich die Aufgabe untergliedern?
 - Beispiel: Rechnung für Kunden ausgeben
 - Rechnungspositionen zusammenstellen
 - Lieferungspositionen einlesen
 - Preis für Produkt ermitteln
 - Mehrwertsteuer ermitteln
 - Rechnungspositionen addieren
 - Positionen formatiert ausdrucken

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.21

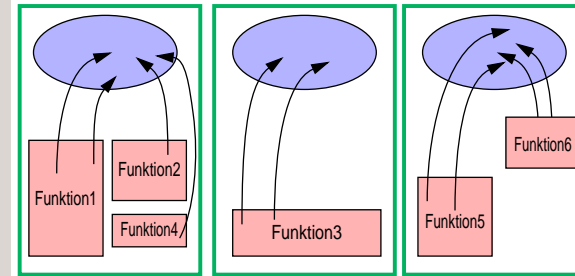
Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Top-down Entwurf (3) Modul-Bildung

F.9 Programmstruktur & Module

- Lösung:
Gliederung von Datenbeständen zusammen mit Funktionen, die darauf operieren

➔ Modul



C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

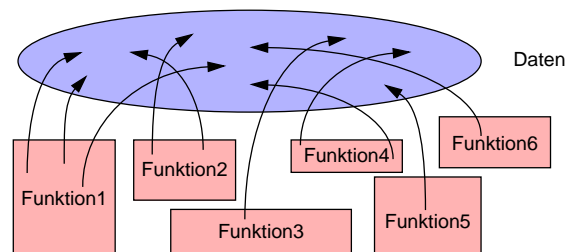
F.23

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Top-down Entwurf (2)

F.9 Programmstruktur & Module

- Problem:
Gliederung betrifft nur die Aktivitäten, nicht die Struktur der Daten
- Gefahr:
Sehr viele Funktionen arbeiten "wild" auf einer Unmenge schlecht strukturierter Daten



C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.22

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Module in C

F.9 Programmstruktur & Module

- Teile eines C-Programms können auf mehrere .c-Dateien (C-Quelldateien) verteilt werden
- Logisch zusammengehörende Daten und die darauf operierenden Funktionen sollten jeweils zusammengefasst werden
- ➔ Modul
- Jede C-Quelldatei kann separat übersetzt werden (Option -c)
 - Zwischenergebnis der Übersetzung wird in einer .o-Datei abgelegt

```
% cc -Aa -c main.c          (erzeugt Datei main.o)
% cc -Aa -c f1.c            (erzeugt Datei f1.o)
% cc -Aa -c f2.c f3.c      (erzeugt f2.o und f3.o)
```

- Das Kommando cc kann mehrere .c-Dateien übersetzen und das Ergebnis — zusammen mit .o-Dateien — binden:

```
% cc -Aa -o prog main.o f1.o f2.o f3.o f4.c f5.c
```

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.24

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

3 Module in C

F.9 Programmstruktur & Module

!!! .c-Quelldateien auf keinen Fall mit Hilfe der #include-Anweisung in andere Quelldateien einkopieren

- Bevor eine Funktion aus einem anderen Modul aufgerufen werden kann, muß sie **deklariert** werden
 - Parameter und Rückgabewerte müssen bekannt gemacht werden
- Makrodefinitionen und Deklarationen, die in mehreren Quelldateien eines Programms benötigt werden, werden zu **Header-Dateien** zusammengefaßt
 - ◆ Header-Dateien werden mit der #include-Anweisung des Preprozessors in C-Quelldateien einkopiert
 - ◆ der Name einer Header-Datei endet immer auf `.h`

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

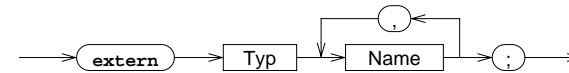
F.25

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Gültigkeit im gesamten Programm

F.10 Gültigkeit von Namen

- Globale Variablen werden außerhalb von Funktionen definiert
- Globale Variablen sind ab der Definition in der gesamten Datei zugreifbar
- Globale Variablen, die in anderen Modulen **definiert** wurden, müssen vor dem ersten Zugriff bekanntgemacht werden (**extern-Deklaration** = Typ und Name bekanntmachen)



■ Beispiele:

```
extern int a, b;  
extern char c;
```

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.27

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.10 Gültigkeit von Namen

- Gültigkeitsregeln legen fest, welche Namen (Variablen und Funktionen) wo im Programm bekannt sind
- Mehrere Stufen
 1. Global im gesamten Programm (über Modul- und Funktionsgrenzen hinweg)
 2. Global in einem Modul (auch über Funktionsgrenzen hinweg)
 3. Lokal innerhalb einer Funktion
 4. Lokal innerhalb eines Blocks
- Überdeckung bei Namensgleichheit
 - eine lokale Variable innerhalb einer Funktion überdeckt gleichnamige globale Variablen
 - eine lokale Variable innerhalb eines Blocks überdeckt gleichnamige globale Variablen und gleichnamige lokale Variablen in umgebenden Blöcken

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.26

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Gültigkeit im gesamten Programm

F.10 Gültigkeit von Namen

■ Probleme mit globalen Variablen

- ◆ Zusammenhang zwischen Daten und darauf operierendem Programmcode geht verloren
- ◆ Funktionen können Variablen ändern, ohne daß der Aufrufer dies erwartet (Seiteneffekte)
- ◆ Programme sind schwer zu pflegen, weil bei Änderungen der Variablen erst alle Programmteile, die sie nutzen gesucht werden müssen

➔ **globale Variablen möglichst vermeiden!!!**

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.28

Reproduktion jeder Art oder Vervielfältigung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 Gültigkeit im gesamten Programm F.10 Gültigkeit von Namen

Globale Funktionen

- Funktionen sind generell global (es sei denn, die Erreichbarkeit wird explizit auf das Modul begrenzt)
- Funktionen aus anderen Modulen müssen ebenfalls vor dem ersten Aufruf **deklariert** werden (= Typ, Name und Parametertypen bekanntmachen)
- Das Schlüsselwort **extern** ist bei einer Funktionsdeklaration nicht notwendig
- Beispiele:


```
double sinus(double);
float power(float, int);
```
- Globale Funktionen (und soweit vorhanden die globalen Daten) bilden die äußere Schnittstelle eines Moduls
 - "vertragliche" Zusicherung an den Benutzer des Moduls

3 Lokale Variablen F.10 Gültigkeit von Namen

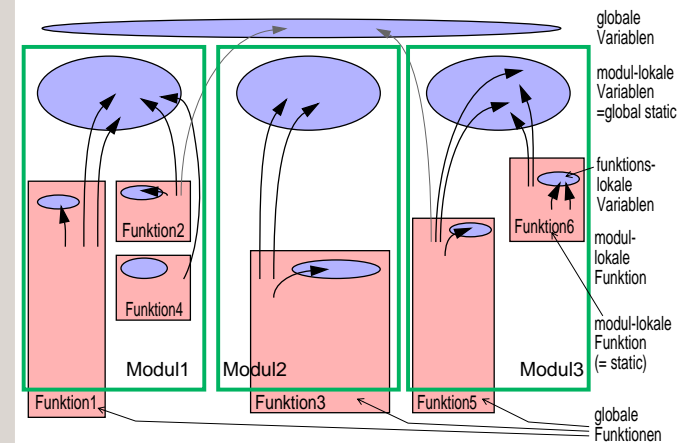
- Variablen, die innerhalb einer Funktion oder eines Blocks definiert werden, sind lokale Variablen
- bei Namensgleichheit zu globalen Variablen oder lokalen Variablen eines umgebenden Blocks gilt die jeweils letzte Definition
- lokale Variablen sind außerhalb des Blocks, in dem sie definiert wurden, nicht zugreifbar und haben dort keinen Einfluß auf die Zugreifbarkeit von Variablen

2 Gültigkeit nur im Modul F.10 Gültigkeit von Namen

- Zugriff auf eine globale Variable oder Funktion kann auf das Modul (= die Datei) beschränkt werden, in der sie definiert wurde
 - Schlüsselwort **static** vor die Definition setzen
-
- **extern**-Deklarationen in anderen Modulen sind nicht möglich
 - Die **static**-Variablen bilden zusammen den Zustand eines Moduls, die Funktionen des Moduls operieren auf diesem Zustand
 - Hilfsfunktionen innerhalb eines Moduls, die nur von den Modulfunktionen benötigt werden, sollten immer **static** definiert werden
 - sie werden dadurch nicht Bestandteil der Modulschnittstelle (= des "Vertrags" mit den Modulbenutzern)

!!! das Schlüsselwort **static** gibt es auch bei lokalen Variablen (mit anderer Bedeutung!)

4 Übersicht F.10 Gültigkeit von Namen



F.11 Lebensdauer von Variablen

- Die Lebensdauer einer Variablen bestimmt, wie lange der Speicherplatz für die Variable aufgehoben wird
- Zwei Arten
 - ◆ Speicherplatz bleibt für die gesamte Programmausführungszeit reserviert
 - statische (`static`) Variablen
 - ◆ Speicherplatz wird bei Betreten eines Blocks reserviert und danach wieder freigegeben
 - dynamische (`automatic`) Variablen

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.33

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 static-Variablen

F.11 Lebensdauer von Variablen

- Der Speicher für alle globalen Variablen ist generell von Programmstart bis Programmende reserviert
- Lokale Variablen erhalten bei Definition mit dem Schlüsselwort `static` eine **Lebensdauer über die gesamte Programmausführung** hinweg
 - der Inhalt bleibt bei Verlassen des Blocks erhalten und ist bei einem erneuten Eintreten in den Block noch verfügbar
- !!! Das Schlüsselwort `static` hat bei globalen Variablen eine völlig andere Bedeutung (Einschränkung des Zugriffs auf das Modul)
- Static-Variablen können durch beliebige konstante Ausdrücke initialisiert werden
 - die Initialisierung wird nur einmal beim Programmstart vorgenommen (auch bei lokalen Variablen!)
 - wird eine static-Variable nicht explizit initialisiert, ist sie automatisch mit 0 vorbelegt

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.35

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

1 auto-Variablen

F.11 Lebensdauer von Variablen

- Alle lokalen Variablen sind automatic-Variablen
 - der Speicher wird bei Betreten des Blocks / der Funktion reserviert und bei Verlassen wieder freigegeben
 - der Wert einer lokalen Variablen ist beim nächsten Betreten des Blocks nicht mehr sicher verfügbar!
- Lokale auto-Variablen können durch beliebige Ausdrücke initialisiert werden
 - die Initialisierung wird bei jedem Eintritt in den Block wiederholt
 - !!! wird eine auto-Variable nicht initialisiert, ist ihr Wert vor der ersten Zuweisung undefiniert (= irgendwas)

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.34

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.12 Wertaustausch zwischen Funktionen

Mechanismus	Aufrufer → Funktion	Funktion → Aufrufer
Parameter	ja	mit Hilfe von Zeigern
Funktionswert	nein	ja
globale Variablen	ja	ja

- Verwendung globaler Variablen?
 - ◆ Variablen, die von vielen Funktionen verwendet werden und/oder oft als Parameter übergeben werden müßten
 - Menge der Funktionen muß überschaubar bleiben
 - Zugriff auf Modul begrenzen (globale static-Variablen)
 - **sonst sehr schlechter Programmierstil**
 - ◆ Variablen, die keiner Funktion als Variable oder Parameter fest zugeordnet werden können
 - Modul suchen, dem die Variable zugeordnet werden kann!!!
 - ◆ Variablen, deren Lebensdauer nicht beschränkt sein darf, die aber nicht in `main()` deklariert werden sollen
 - in zugehöriger Funktion lokal-static definieren

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.36

Reproduktion jeder Art oder Verwendung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.13 Getrennte Übersetzung von Programmteilen — Beispiel

■ Hauptprogramm (Datei `fplot.c`)

```
#include "trig.h"
#define INTERVALL 0.01

/* Funktionswerte ausgeben
 */
int main(void)
{
    char c;
    double i;

    printf("Funktion (Sin, Cos, Tan, cOt)? ");
    scanf("%x", &c);

    switch (c) {
        ...
        case 'T':
            for (i=-PI/2; i < PI/2; i+=INTERVALL)
                printf("%lf %lf\n", i, tan(i));
            break;
        ...
    }
}
```

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.37

Reproduktion jeder Art oder Verwechslung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.13 ...

F.13 Getrennte Übersetzung von Programmteilen — Beispiel

■ Trigonometrische Funktionen (Datei `trigfunc.c`)

```
#include "trig.h"

double tan(double x) {
    return(sin(x)/cos(x));
}

double cot(double x) {
    return(cos(x)/sin(x));
}

double cos(double x) {
    return(sin(PI/2-x));
}

...

```

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.39

Reproduktion jeder Art oder Verwechslung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.13 ...

F.13 Getrennte Übersetzung von Programmteilen — Beispiel

■ Header-Datei (Datei `trig.h`)

```
#include <stdio.h>
#define PI 3.1415926535897932
double tan(double), cot(double);
double cos(double), sin(double);
```

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.38

Reproduktion jeder Art oder Verwechslung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

F.13 ...

F.13 Getrennte Übersetzung von Programmteilen — Beispiel

■ Trigonometrische Funktionen — Fortsetzung (Datei `trigfunc.c`)

```
...

double sin (double x)
{
    double summe;
    double x_quadrat;
    double rest;
    int k;

    k = 0;
    summe = 0.0;
    rest = x;
    x_quadrat = x*x;

    while ( fabs(rest) > 1e-9 ) {
        summe += rest;
        k += 2;
        rest *= -x_quadrat/(k*(k+1));
    }
    return(summe);
}
```

C-Ing

Einführung in die Programmierung für Ingenieure — C
© Jürgen Kleinöder, Universität Erlangen-Nürnberg, IMMD IV, 1998

F-Funktionen.doc 1998-06-10 11.31

F.40

Reproduktion jeder Art oder Verwechslung dieser Unterlagen, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.