C Kurzeinführung in die Programmiersprache C

- Literatur zur C-Programmierung:
  - Darnell, Margolis. *C: A Software Engineering Approach*. Springer 1991
  - Kernighan, Ritchie. *The C Programming Language.* Prentice-Hall 1988

C-1 Überblick

- Struktur eines C-Programms
- Datentypen und Variablen
- Anweisungen
- Funktionen
- C-Preprozessor
- Programmstruktur und Module
- Zeiger(-Variablen)
- sizeof-Operator
- Explizite Typumwandlung — Cast-Operator
- Speicherverwaltung
- Felder
- Strukturen
- Ein- /Ausgabe
- Fehlerbehandlung

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

C-2 Struktur eines C-Programms

globale Variablendefinitionen

Funktionen

```
int main(int argc, char *argv[]) {
    Variablendefinitionen
    Anweisungen
}
```

- Beispiel

```
int main(int argc, char *argv[]) {
    printf("Hello World!");
}
```

- Übersetzen mit dem C-Compiler:
  `cc -o hello hello.c`
- Ausführen durch Aufruf von **`hello`**

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

C-3 Datentypen und Variablen

- Datentypen legen fest:
  - Repräsentation der Werte im Rechner
  - Größe des Speicherplatzes für Variablen
  - erlaubte Operationen

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

1 Standardtypen in C

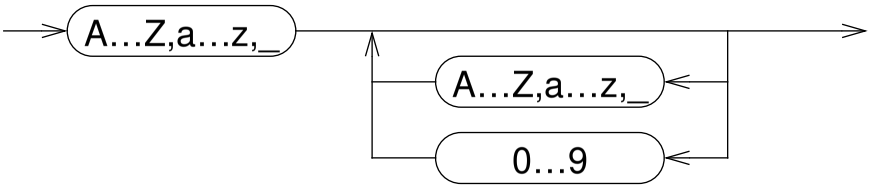
- Eine Reihe häufig benötigter Datentypen ist in C vordefiniert

`char`	Zeichen (im ASCII-Code dargestellt, 8 Bit)
`int`	ganze Zahl (16 oder 32 Bit)
`float`	Gleitkommazahl (32 Bit) etwa auf 6 Stellen genau
`double`	doppelt genaue Gleitkommazahl (64 Bit) etwa auf 12 Stellen genau
`void`	ohne Wert

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

2 Variablen

- Variablen besitzen
  - **Namen** (Bezeichner)
  - Typ
  - zugeordneten Speicherbereich für einen Wert des Typs
    Inhalt des Speichers (= **aktueller Wert** der Variablen) ist veränderbar!
  - **Lebensdauer**
- Variablenname:

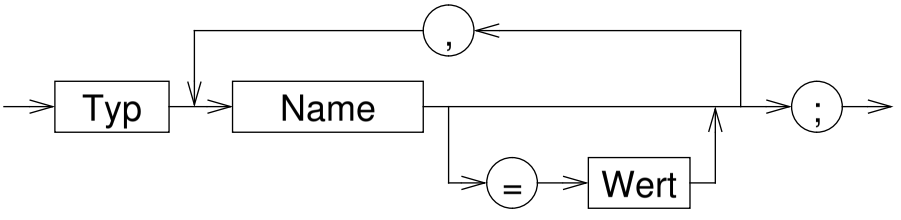


(Buchstabe oder _ ,
evtl. gefolgt von beliebig vielen Buchstaben, Ziffern oder _)

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

2 Variablen (2)

- eine **Variablen-Definition** deklariert eine Variable
  und reserviert den benötigten Speicherbereich



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

2 Variablen (3)

- Variablen-Definition: Beispiele

```
int a1;
float a, b, c, dis;
int anzahl_zeilen=5;
char trennzeichen;
```

- Position von Variablendefinitionen im Programm:
  - nach jeder "{"
  - außerhalb von Funktionen
- Wert kann bei der Definition initialisiert werden
- Wert ist durch Wertzuweisung und spezielle Operatoren veränderbar

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

3 Strukturierte Datentypen (structs)

- Zusammenfassen mehrerer Daten zu einer Einheit

```
struct person {
     char *name;
     int alter;
};
```

- Variablen-Definition

```
struct person p1;
```

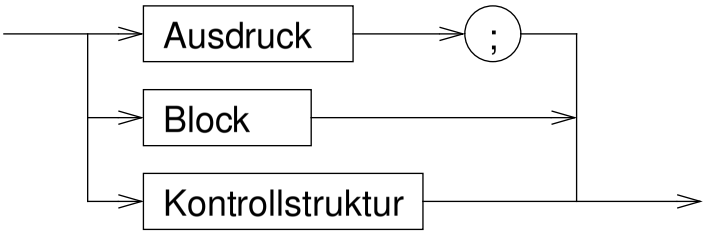
- Zugriff auf Elemente der Struktur

```
p1.name = "Hans";
```


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

C-4 Anweisungen

Anweisung:



1 Ausdrücke - Beispiele

- `a = b + c;`
- `{ a = b + c; x = 5; }`
- `if (x == 5) a = 3;`

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

2 Blöcke

- Zusammenfassung mehrerer Anweisungen
- Lokale Variablendefinitionen → Hilfsvariablen
- Schaffung neuer Sichtbarkeitsbereiche (***Scopes***) für Variablen

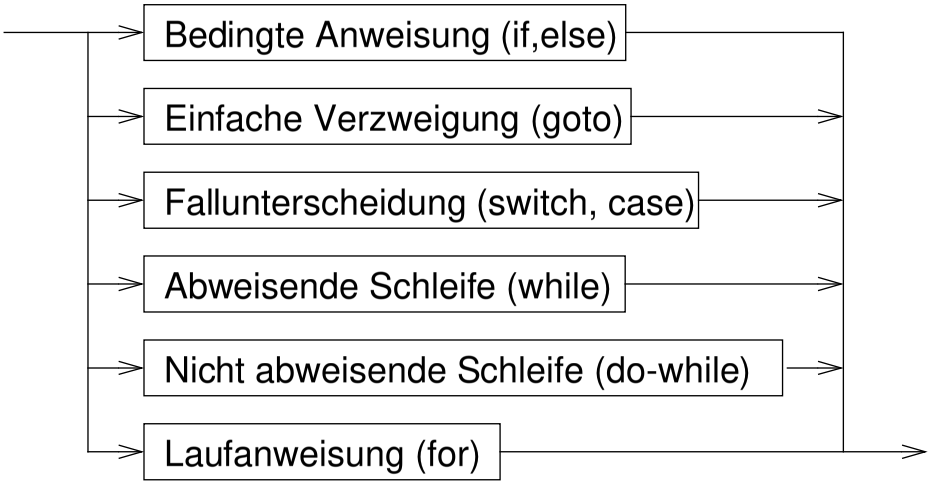
```
main()
{
    int x, y, z;
    x = 1;
    {
        int a, b, c;
        a = x+1;
        {
            int a, x;
            x = 2;
            a = 3;
        }
        /* a: 2, x: 1 */
    }
}
```


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

3 Kontrollstrukturen

- Kontrolle des Programmablaufs in Abhängigkeit vom Ergebnis von Ausdrücken

Kontrollstruktur:



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

4 Kontrollstrukturen — Schleifensteuerung

- `break`
  - bricht die umgebende Schleife bzw. **`switch`**-Anweisung ab

```
int c;

do {
    if ( (c = getchar()) == EOF ) break;
    putchar(c);
}
while ( c != '\n' );
```

- `continue`
  - bricht den aktuellen **Schleifendurchlauf** ab
  - setzt das Programm mit der Ausführung des Schleifenkopfes fort

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

C-5 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
  - ➥ verringern die Komplexität durch Zerteilen umfangreicher, schwer überblickbarer Aufgaben in kleine Komponenten
  - ➥ erlauben die Wiederverwendung von Programmkomponenten
  - ➥ verbergen Implementierungsdetails vor anderen Programmteilen (**Black-Box**-Prinzip)

1 Funktionsdefinition

- Schnittstelle (Ergebnistyp, Name, Parameter)
- \+ Implementierung

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

2 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);
}
```

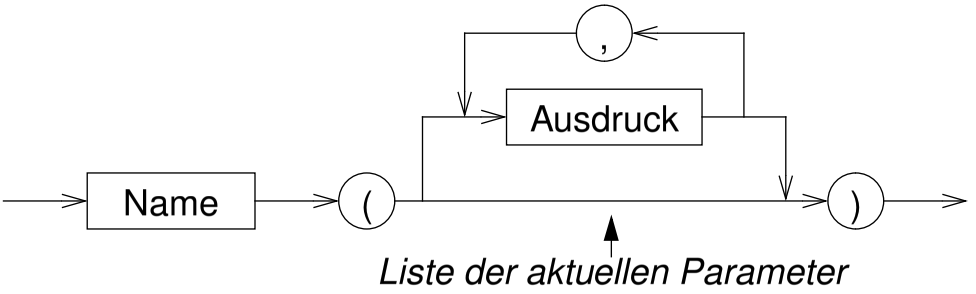
```
int main(int argc, char *argv[])
{
     double wert;

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

- beliebige Verwendung von **`sinus`** in Ausdrücken:

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


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

3 Funktionsaufruf



- 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

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

4 Regeln

- Funktionen werden global definiert
- `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)

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


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

4 Regeln (2)

- Funktionen müssen **deklariert** sein, bevor sie aufgerufen werden
  - = Rückgabetyp 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**

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

5 Funktionsdeklaration

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

    Typ Name (Liste formaler Parameter);

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

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


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

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);
}
```


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

7 Parameterübergabe an Funktionen

- allgemein in Programmiersprachen vor allem zwei Varianten:
  - ➤ call by value (wird in C verwendet)
  - ➤ call by reference (wird in C nicht verwendet)
- call-by-value: 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

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

C-6 C-Preprozessor

- 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

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

1 Makrodefinitionen

- 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
```


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

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!)

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

C-7 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

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

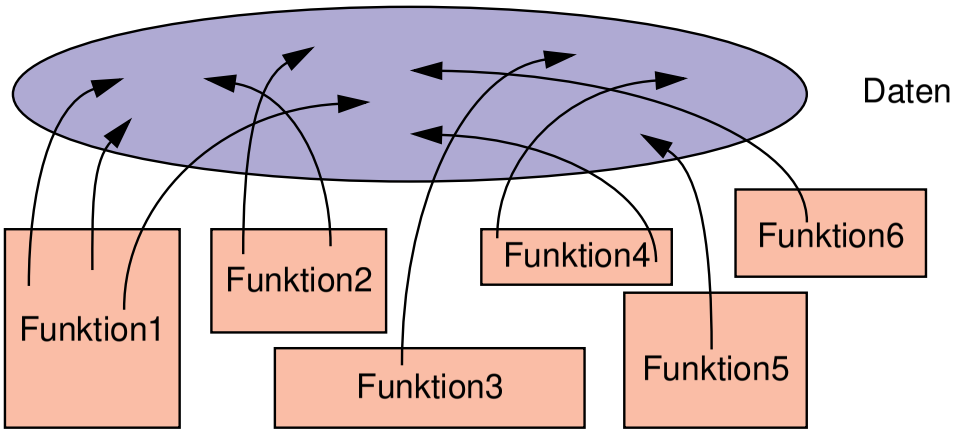
2 Top-down Entwurf

- Zentrale Fragestellung
  - ◆ was ist zu tun?
  - ◆ in welche Teilaufgaben läßt sich die Aufgabe untergliedern?
    - ➤ Beispiel: Rechnung für Kunden ausgeben
      - – Rechnungspositionen zusammenstellen
        - – Lieferungsposten einlesen
        - – Preis für Produkt ermitteln
        - – Mehrwertsteuer ermitteln
      - – Rechnungspositionen addieren
      - – Positionen formatiert ausdrucken

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

2 Top-down Entwurf (2)

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

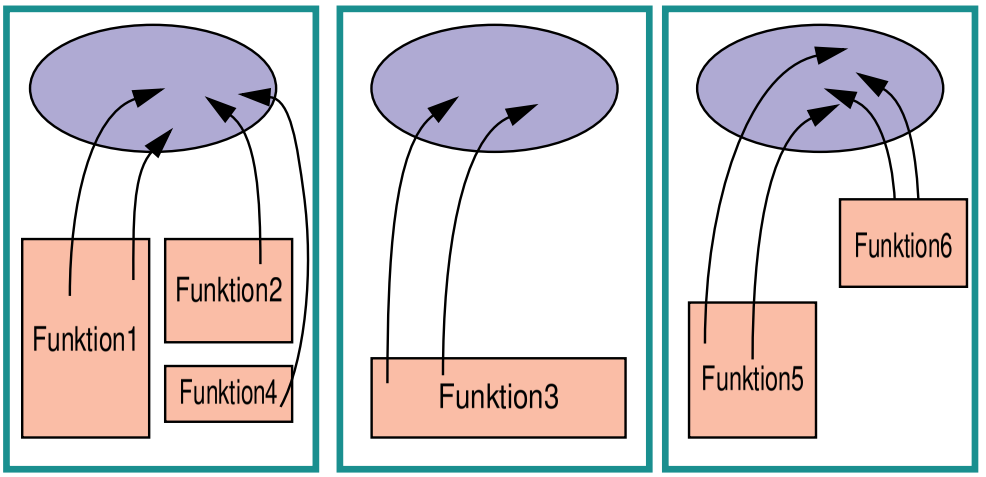


Reproduktion jeder Art oder Verwendung dieser Unterlage, außer zu Lehrzwecken an der Universität Erlangen-Nürnberg, bedarf der Zustimmung des Autors.

2 Top-down Entwurf (3) Modul-Bildung

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

  ➥ **Modul**



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

3 Module in C

- 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 -c main.c          (erzeugt Datei main.o)
% cc -c f1.c            (erzeugt Datei f1.o)
% cc -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 -o prog main.o f1.o f2.o f3.o f4.c f5.c
```


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