Christian-Albrechts-Universität zu Kiel

Rechenzentrum

Programmieren in C

Dr. Simone Knief, Rechenzentrum der CAU

knief@rz.uni-kiel.de

01.03.-03.03.2022

C A U
Christian-Albrechts-Universität zu Kiel
Rechenzentrum

Gliederung

- Einleitung
- Komponenten eines C-Programms
- Daten speichern: Variablen und Konstanten
- Anweisungen und Ausdrücke
- Operatoren
- Kontrollstrukturen
- Felder und Strings
- Funktionen, globale und lokale Variablen
- Zeiger
- Arbeiten mit Dateien
- Strukturen

Literatur

- Online verfügbares Buch: C von A bis Z
  - openbook.rheinwerk-verlag.de/c_von_a_bis_z/
- Hefte aus dem HERDT-Verlag
  - ANSI C 3.0 : Grundlagen der Programmierung
    - https://herdt-campus.com/product/CANSI3
- Kursunterlagen:
  - www.rz.uni-kiel.de/de/kurse/kursunterlagen

Geschichte von C

- 1972 entwickelt von Brian W. Kernighan und Dennis Ritchie
- 1978: Buch: *The C Programming Language: "*K&R Standard"
- 1983: American National Standard Institute (ANSI) bildet eine Komitee, um eine einheitliche Definition von C zu erreichen
- 1989: ANSI Standard-C (C89-Standard)
- 1990: Standard wird mit kleinen Änderungen von der International Standard Organization (ISO) übernommen.
- 1999: Überarbeitung und Ergänzung zum C99-Standard
- 2011: C11-Standard (z.B. mit Multithreading)
- 2018: C18-Standard (nur kleine Änderungen gegenüber C11; Fehlerkorrekturen)

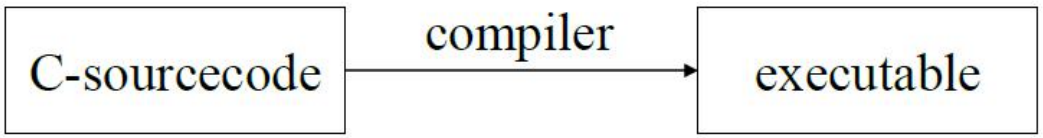
Details zu Änderungen in den verschiedenden Versionen
https://de.wikipedia.org/wiki/Varianten_der_Programmiersprache_C

Warum C ?

- C ist relativ einfach aufgebaut
- C ist dennoch eine leistungsfähige und flexible Sprache
- breite Palette von Compilern und Tools verfügbar
- C ist portabel
- C kommt mit wenigen Schlüsselwörtern aus
- C ist modular
- C ermöglicht hardwarenahe Programmierung
- C-Programme sind ressourcensparend
- C ist eine Teilmenge von C++

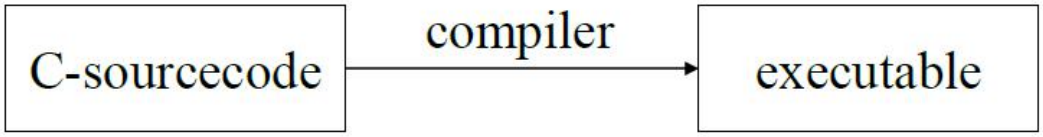
Vom Problem zum Programm

- Problembeschreibung
- Modellbildung: Entwurf von Datenstruktur und Algorithmen
- Erzeugung eines ausführbaren Programms:
  - Erstellung des Programmcodes mit einem Editor
  - Compilieren in eine Objektdatei
  - Linken der Objektdateien zu einem ausführbaren Programm
- Testen des Programms
- Produktionsbetrieb



Compiler

- Übersetzt Programmcode in die Maschinensprache
- besteht eigentlich aus 3 Programmen
  - Compiler
  - Präprozessor
  - Linker
- Präprozessor
  - fügt vor der Programmübersetzung zusätzlichen Text in den Code ein
  - entfernt Codezeilen aus dem Code (z.B. Kommentare)
- Linker
  - auch als Binder bezeichnet
  - verbindet die einzelnen Programmmodule zu einer ausführbaren Datei



Beispiel

- Problem
  - Programm soll auf dem Bildschirm ausgeben:
    "Viel Spass mit C !"
- Programmcode mit einem Editor eingeben:

```
#include <stdio.h>
int main ()
{
  printf ("Viel Spass mit C !\n");
  return 0;
}
```

- Code abspeichern unter dem Namen hello.c
- Programmcode compilieren:
  - gcc -o hello hello.c
  - kein Fehler → Datei hello wird erzeugt
- Programm starten mit ./hello

- ohne Option –o: executable hat den Namen a.out

Freie Compiler

- Linux:
  - Gnu-Compiler: gcc.gnu.org
- Windows:
  - Code::Blocks: http://codeblocks.org/
    - Datei codeblocks-13.12mingw-setup.exe auswählen
  - Dev-Compiler: www.bloodshed.net/devcpp.html
  - Pelles-C-Compiler: www.pellesc.de
  - MinGW: www.mingw.org
- umfangreiche Liste verfügbarer Compiler:
  - www.thefreecountry.com/compilers/cpp.shtml

Überblick: Komponenten eines C-Programms

```
/* Programm zur Bildung einer Summe zweier Zahlen */
#include <stdio.h>
int main (void)
{
    int x,y,summe;
    printf("Bitte geben Sie zwei ganze Zahlen ein! \n");
    scanf(" %d %d", &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

Programmkommentare

```
/* Programm zur Bildung einer Summe zweier Zahlen */
#include <stdio.h>
int main (void)
{
     int x,y,summe;
    printf(“Bitte geben Sie zwei ganze Zahlen ein! \n“);
    scanf(“ %d %d“, &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

Programmkommentare

- /* Kommentartext */
- z.B.: /* Programm zur Bildung einer Summe zweier Zahlen */
- kann über mehrere Zeilen gehen; darf nicht verschachtelt werden
- int xwert; // Kommentartext
- enthalten Erklärungen und Kommentare zum Code
- Compiler ignoriert Zeichen innerhalb des Kommentarblocks bzw. nach //
  - Präprozessor entfernt Kommentarzeilen bei der Übersetzung
- beeinflussen die Performance eines Programms nicht
- sollen helfen das Programm auch ohne fremde Hilfe zu verstehen
- auskommentieren von Programmteilen, die evtl. nicht benötigt werden (z.B. zur Fehleranalyse)
- keine ünnötigen Kommentare für offensichtliche Anweisungen:
  - *z.B.: /* die folgende Programmzeile gibt die Anweisung: hello world aus */*
    printf(“hello world\n”);

Komponenten eines C-Programms

```
/* Programm zur Bildung einer Summe zweier Zahlen */
#include <stdio.h>
int main (void)
{
     int x,y,summe;
    printf(“Bitte geben Sie zwei ganze Zahlen ein! \n“);
    scanf(“ %d %d“, &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

#include-Direktive

- #include <stdio.h>
- include-Dateien oder Header-Dateien
- gehören zu den Präprozessoranweisungen
- sind Anweisungen an den Präprozessor
- stehen immer am Anfang eines Programms
- beginnen mit einem #
- Inhalt wird beim Compilieren in den Programmcode eingefügt
- Header-Datei <stdio.h> enthält Informationen über Ein- und Ausgabefunktionen (z.B. printf und scanf)
- am Zeilenende darf kein Semikolon stehen (<-> Anweisungen)

Komponenten eines C-Programms


```
/* Programm zur Bildung einer Summe zweier Zahlen */
#include <stdio.h>
int main (void)
{
     int x,y,summe;
    printf(“Bitte geben Sie zwei ganze Zahlen ein! \n“);
    scanf(“ %d %d“, &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

Funktion main

- ist in jedem C-Programm vorhanden
- ```
  int main (void)
  {
      Programmanweisungen
  return 0;
  }
  ```
- erste Funktion, die in einem C-Programm aufgerufen wird → Einstiegspunkt
- geschweifte Klammern schließen Programmteile ein, die eine C-Funktion bilden
  → Anweisungsblock
- return 0;
  - Hauptprogramm beenden und Fehlercode 0 (d.h. kein Fehler) zurückgeben
  ```

Komponenten eines C-Programms

```
/* Programm zur Bildung einer Summe zweier Zahlen */
#include <stdio.h>
int main (void)
{
   int x,y,summe;
   printf(“Bitte geben Sie zwei ganze Zahlen ein! \n“);
   scanf(“ %d %d“, &x, &y);
   summe=x+y;
   printf("Die Summe von %d und %d ist %d \n",x,y,summe);
   return 0;
}
```

Variablendefinition

- int x,y,summe;
- Variablen sind Speicherstellen für verschiedene Arten von Daten
  - Zeichen, Text, ganze oder reelle Zahlen
- Variablen müssen vor der ersten Verwendung zunächst definiert werden
- informiert den Compiler über Variablennamen und Datentyp
- reserviert einen festen Speicherbereich im Hauptspeicher

Komponenten eines C-Programms

```
/* Programm zur Bildung einer Summe zweier Zahlen */
#include <stdio.h>
int main(void)
{
     int x,y,summe;
    printf(“Bitte geben Sie zwei ganze Zahlen ein! \n“);
    scanf(“ %d %d“, &x, &y);
    summe=x+y;
    printf("Die Summe von %d und %d ist %d \n",x,y,summe);
    return 0;
}
```

Programmanweisungen

- Anweisungen realisieren alle Operationen, die ein Programm ausführen soll
- typische C-Anweisungen:
  - Informationen auf Bildschirm ausgeben :
    printf(“Bitte geben Sie zwei ganze Zahlen ein! \n“);
  - Daten von der Tastatur einlesen:
    scanf(“ %d %d", &x, &y);
  - mathematische Operationen ausführen:
    summe=x+y;
  - Funktionen aufrufen
- Quellcode enthält gewöhnlich eine Anweisung pro Codezeile
- Anweisungen immer mit einem Semikolon (;) abschließen

```
#include <stdio.h>
int main(void)
{
    printf ("Hallo \n");
    return 0;
}
```

```
#include <stdio.h>
int main(void)
{
    printf ("Hallo \n");
    return 0;
}
```

```
#include<stdio.h>
int main(void){printf("Hallo \n");
    return 0;}
```

Programmierstil

- Anordnung des Programmcodes
- nur eine Anweisung pro Zeile
- Anweisungsblöcke einrücken
- Programme großzügig kommentieren

Zusammenfassung I

- Entwicklungszyklus eines Programms vorgestellt
- Hauptkomponenten eines C-Programms eingeführt

- bisher behandelte C-Komponenten:
  - Programmkommentare
    /* Kommentartext */ bzw. //
  - include-Direktive für Ein- und Ausgabefunktionen
    #include <stdio.h>
  - in jedem C-Programm gibt es eine Funktion main

```
int main (void)
{
   Programmanweisungen
   return 0;
}
```

Daten speichern: Variablen und Konstanten

- Was ist eine Variable ?
- Wie definiere und verwende ich Variablen ?
- Welche Datentypen gibt es in C ?
- Wie definiere und verwende ich Konstanten ?

Was ist eine Variable ?

- benannte Speicherstelle für Daten (Zahlen oder Zeichen) im Hauptspeicher
- Wert kann sich während der Programmausführung verändern
- zu einer Variablen gehören folgende Informationsbestandteile:
  - Variablenname:
    - stellt die Speicherstelle des Wertes symbolisch dar
  - Variablenwert:
    - Zahlenwert, der an dieser Speicherstelle gespeichert wird
  - Datentyp:
    - legt das Format für den zu speichernden Wert fest
  - Adresse:
    - genaue Position im Hauptspeicher des Computers
  - Beispiel: `int xwert = 5;`

Variablenname

- kann Zeichen, Ziffern und den Unterstrich enthalten.
- erste Zeichen muss immer ein Buchstabe (oder Unterstrich) sein
- C unterscheidet zwischen Groß- und Kleinschreibung
  - *d.h. Summe und summe sind unterschiedliche Variablen*
- C-Schlüsselwörter (z.B. float, double) sind als Variablenname nicht zulässig
- Variablennamen müssen eindeutig in einer Funktion sein
- Namen sollten immer aussagekräftig sein
  - sollten in Kleinbuchstaben geschrieben werden (<-> Konstanten)
- Beispiele für gültige Variablennamen:
  - Prozent, summe, gewinn_pro_jahr, umsatz, y2x5
- Beispiele für nicht zulässige Variablennamen:
  - sparkassen#konto, double, 9winter, Zähler

C Schlüsselwörter: Übersicht

- auto, break, case, char, const, continue, default, do, double
- else, enum, extern, float, for, goto, if, int, long, register, return
- short, signed, sizeof, static, struct, switch, typedef, union, unsigned
- void, volotaile, while

- _Bool, _Complex, _Imaginary, inline, restrict *(seit C99)*

- _Alignas, _Alignof, _Atomic, _Generic, _Noreturn, _Static_assert, _Thread_local *(seit C11)*

Elementare Datentypen

- sind die Baupläne für die Darstellung einer Variablen
- verschiedene numerische Werte haben unterschiedlichen Speicherbedarf und ausführbare mathematische Operationen sind nicht für alle Datentypen gleich
- 3 Kategorien von Datentypen
  - Textzeichen: Darstellung eines einzelnen Zeichens
  - Integer-Zahlen: Darstellung ganzer Zahlen
    - vorzeichenbehaftet oder vorzeichenlos
  - Gleitkommazahlen: Zahlen mit Nachkommastellen
    - float: Dezimalzahlen mit einfacher Genauigkeit
    - double: Dezimalzahlen mit doppelter Genauigkeit
    - long double: Dezimalzahlen mit zusätzlicher Genauigkeit
    - Darstellung: Dezimalpunkt anstelle Dezimalkomma
      - z.B. 5.25

Speicherbedarf einer Variablen ermitteln

- sizeof-Operator ermittelt Größe eines Datentyps (in Byte) auf einem System
- Beispielprogramm:

```
#include <stdio.h>
int main(void)
{
    printf("Ein char belegt %d Byte.\n",sizeof(char));
    printf("Ein int belegt %d Bytes.\n",sizeof(int));
    printf("Ein short belegt %d Bytes.\n",sizeof(short));
    printf("Ein long Integer belegt %d Bytes. \n", sizeof(long));
    printf("Ein float belegt %d Bytes.\n",sizeof(float));
    printf("Ein double belegt %d Bytes.\n",sizeof(double));
    printf("Ein long double belegt %d Bytes.\n",sizeof(long double));
    return 0;
}
```

Speicherbedarf ermitteln

- Programm übersetzen:
  - gcc –o size sizeof.c
- Programm ausführen:
  - ./size
- Programmausgabe:
  Ein char belegt 1 Byte.
  Ein int belegt 4 Bytes.
  Ein short belegt 2 Bytes.
  Ein long belegt 8 Bytes.
  Ein float belegt 4 Bytes.
  Ein double belegt 8 Bytes.
  Ein long double belegt 16 Bytes.

Übersicht: Datentypen

Variablentyp	Schlüsselwort	Bytes	Wertebereich
Zeichen	char	1	-128 bis 127
Ganzzahl	int	4	-2147483648 bis 2147483648
Kurze Ganzzahl	short	2	-32768 bis 32767
Lange Ganzzahl	long	8	-+9223372036854775808
Zeichen ohne VZ	unsigned char	1	0 bis 255
Ganzzahl ohne VZ	unsigned int	4	0 bis 4294967296
Kurze Ganzzahl ohne VZ	unsigned short	4	0 bis 65535
Lange Ganzzahl ohne VZ	unsigned long	4	0 bis 1.8E+19
Gleitkommzahl einfache Genauigkeit	float	4	1.2E-38 bis 3.4E+38 Genauigkeit 6 Dezimalstellen
Gleitkommazahl doppelte Genauigkeit	double	8	2.3E-308 bis 1.7E+308 Genauigkeit 15 Dezimalstellen

Wertebereich für Ganzzahlen

- Headerdatei limits.h enthält Wertebereich
- Beispielprogramm:

```
#include <stdio.h>
#include <limits.h>
int main(void)
{
    printf("min. short-Wert:  %d \n",SHRT_MIN);
    printf("max.  short-Wert: %d \n",SHRT_MAX);
    printf("min.  int-Wert: %d \n",INT_MIN);
    printf("max. int-Wert: %d \n",INT_MAX);
    printf("min. long-Wert %ld \n",LONG_MIN);
    printf("max. long-Wert: %ld \n",LONG_MAX);
    return 0;
}
```

Wertebereich für Ganzzahlen

- Programm übersetzen:
  - gcc –o limits limits_ganz.c
- Programm ausführen:
  - ./limits

- Programmausgabe:

```
min. short Wert: -32768
max. short Wert: 32767
min. int Wert: -2147483648
max. int Wert: 2147483647
min. long Wert: -9223372036854775808
max. long Wert: 9223372036854775807
```

Limits für Gleitkommazahlen

- Headerdatei float.h enthält Limits für Gleitkommazahlen
- Beispielprogramm:

```
#include <stdio.h>
#include <float.h>
int main(void)
{
    printf("float Genauigkeit:  %d \n",FLT_DIG);
    printf("double Genauigkeit: %d \n",DBL_DIG);
    printf("long double Genauigkeit: %d \n",LDBL_DIG);
    return 0;
}
```

Limits für Gleitkommazahlen

- Programm übersetzen:
  - gcc –o floats limits_float.c
- Programm ausführen:
  - ./floats
- Programmausgabe:

  float Genauigkeit: 6
  double Genauigkeit: 15
  long double Genauigkeit: 18

Datentypen

- in C keinen “Boolschen” Datentyp für Wahrheitswerte True oder False
  - Wert von Null → False
  - Wert ungleich Null (meist 1) → True

- Datentyp void
  - ist null Bytes groß
  - z.B. verwendet wenn Funktionen keinen Rückgabewert haben
- komplexe Datentypen
  - durch Aneinanderreihung von elementaren Datentypen → Felder
  - durch Neudefinition → Strukturen

Verwendung von Variablen

- 3 Schritte für Variablenverwendung erforderlich:
  - Variablendeklaration
  - Variableninitialisierung
  - Verwendung

Variablendeklaration und -definition

- jede Variable muss vor der Verwendung erst deklariert und definiert werden
- **Deklaration**: teilt dem Compiler Namen und Datentyp mit
- mit der **Definition** wird auch zusätzlich Speicherplatz reserviert
- allg. Form: Datentyp Variablenname;
- auf einer Zeile können mehrere Variablen desselben Typs deklariert werden
- Variablenname kann frei gewählt werden
- Variablen immer am Anfang eines Blocks deklariereren
  - C99-Standard: ausreichend Variablen vor erster Verwendung zu deklarieren
- Variablen sind im allgemeinen nur innerhalb ihres Blockes {...} definiert
- Beispiele:
  - int zaehler, start;
  - float prozent, total;
  - char zeichen;

Variablen deklarieren

```
#include <stdio.h>
int main(void)
{
    z=100+x;
      return 0;
}
```

→ Fehler bei Programmübersetzung:

→ z nicht deklariert (erste Benutzung in dieser Funktion)

Variablen deklarieren

```
#include <stdio.h>
int main(void)
{
    z=100+x;
    return 0;
}
```

→ Fehler bei Programmübersetzung:

```
#include <stdio.h>
int main(void)
{
    int z,x;
    z=100+x;
    return 0;
}
```

Variablen initialisieren

- nach der Deklaration haben Variablen oft einen unbestimmten Wert
- Variablen sollten vor der ersten Verwendung zusätzlich immer initialisiert werden
- Initialisierung legt einen Anfangswert für die Variable fest
- Initialisierung kann bei der Deklaration oder in einer extra Zuweisung erfolgen
- Variablen dürfen nicht mit Werten außerhalb der Gültigkeit initialisiert werden
- Beispiele:
  - int zaehler=0;
  - int zaehler;
    zaehler=0;
  - int breite=5;
  - float prozent=0.25;
  - char zeichen='X';
- Initialisierung z.B. wichtig bei: summe=summe+1;

Variablen verwenden (Beispiel)

```
/* Flaeche und Umfang eines Rechtecks berechnen */
#include<stdio.h>
int main(void)
{
    int umfang=0, flaeche=0;
     int breite=5;
    int hoehe=3;
    umfang=2*(breite+hoehe);
    flaeche=(breite*hoehe);
    printf("Der Umfang betraegt %d. \n",umfang);
    printf("Die Flaeche betraegt %d. \n",flaeche);
     return 0;
}
```

Variablen verwenden (Beispiel)

```
/* Flaeche und Umfang eines Rechtecks berechnen */
#include<stdio.h>
int main(void)
{
    int umfang, flaeche;
     int breite=5;
    int hoehe=3;
    umfang=2*(breite+hoehe);
    flaeche=(breite*hoehe);
    printf("Der Umfang betraegt %d. \n",umfang);
    printf("Die Flaeche betraegt %d. \n",flaeche);
     return 0;
}
```

- Programm übersetzen:
  - gcc –o rechteck rechteck.c
- Programm ausführen:
  - ./rechteck
- Programmausgabe:
  Der Umfang betraegt 16.
  Die Flaeche betraegt 15.

Konstanten

- Speicherbereich für Daten mit dem ein Programm arbeitet
- Wert kann während der Programmausführung nicht verändert werden
- Konstanten sind Variablen mit einem festen Wert
- wird mit dem Schlüsselwort const definiert:

  const datentyp NAME = wert;
- Konstanten werden gleichzeitig deklariert und initialisiert
- für Konstantennamen gelten die gleiche Regeln wie bei Variablennamen
- Beispiele:
  - const int RADIUS=4;
  - const float BREITE=10.25;
  - const int SCHULDEN=10000, float ZINSSATZ=4.5;

Konstanten (Beispiel)

```
/* Kreisberechnung: Umfang und Flaeche */
#include <stdio.h>
int main(void)
{
    int radius=0;
    const float PI=3.14159;
    float umfang=0.0, flaeche=0.0;
    printf("Bitte den Radius eingeben.\n");
    scanf("%d",&radius);
    flaeche=2*PI*radius*radius;
    umfang=2*PI*radius;
    printf("Der Kreisumfang betraegt %f \n",umfang);
    printf("Die Kreisflaeche betraegt %f \n",flaeche);
    return 0;
}
```

Zusammenfassung: Variablen und Konstanten

- Speicherstellen für Daten (Zahlen und Zeichen)
- Datentypen in C:
  - char: Zeichen
  - short, long und int: Ganzzahlen
  - float und double: Gleitkommazahlen
- Variablen müssen vor der ersten Verwendung deklariert und sollten initialisiert werden
  - datentyp name = wert;
- Konstanten werden mit dem Schlüsselwort const deklariert:
  - const datentyp NAME = wert;

Anweisungen und Ausdrücke

- Was ist eine Anweisung ?
- Was ist ein Ausdruck ?

Anweisungen

- Anweisungen
  - vollständige Vorschrift an den Compiler eine bestimmte Aufgabe auszuführen
  - C-Anweisungen sind immer mit einem Semikolon abzuschließen
    - Ausnahme: Präprozessoranweisungen z.B. #include <stdio.h>
  - Verbundanweisung (Block):
    - Gruppe von C-Anweisungen, die in geschweiften Klammern stehen
  - Anweisungen werden in C von rechts nach links ausgewertet
  - Beispiel:
    - summe = x + y;
    - printf("Hallo C-Programmierer.\n");

Ausdrücke

- besteht aus einem oder mehreren Operanden, die miteinander durch Operatoren verknüpft sind
- Ausdrücke haben als Ergebnis immer einen numerischen Wert
- sind eine Verarbeitungsvorschrift
- liefern ein Ergebnis in Abhängigkeit der Verarbeitung und Operanden
- bestehen aus Operanden und Operatoren, die nach festen Regeln zusammen gesetzt sind
- Beispiel: (x+y)*z

Ein- und Ausgabe von Informationen

- Grundlagen der Ein- und Ausgabeanweisungen in C
- Informationen auf dem Bildschirm ausgeben mit der Funktion printf
- Informationen von der Tastatur einlesen mit der Funktion scanf

Informationen am Bildschirm anzeigen

- wird realisiert durch die Bibliotheksfunktion printf
- allgemeine Syntax:

```
#include <stdio.h>
printf(formatstring);
```

- Ausgabe einer Textmeldung:

```
printf("In Ihrem Programm ist ein Fehler aufgetreten ! \n");
```

Ausgabe: *In Ihrem Programm ist ein Fehler aufgetreten !*

- Ausgabe von Text und Variablen:

```
printf("Die Werte von x  und y sind  %d und  %d \n“,xwert , ywert);
```

Ausgabe: *Die Werte von x und y sind 9 und 4*

Formatstrings der Funktion printf

- gibt an, wie die Ausgabe formatiert werden soll
- allgemein:
  ```
  #include <stdio.h>
  printf(Formatstring);
  ```
- Bsp: printf("der Wert von x ist %d \n",xwert);
- enthält folgende Informationen:
  - Literalen Text
  - Escape-Sequenzen: z.B: \n für neue Zeile
  - Formatbezeichner (Umwandlungszeichen)
    - gibt Format der auszugebenden Variablen an
    - %d : integer-Zahl (int)
    - %f: Gleitkommazahl (float)
    - %lf: Gleitkommazahl (double)
    - %s: Zeichenstring (char-Feld)
    - %c: einfaches Zeichen (char)
    - %e bzw. %E: double im Format d.dde+dd bzw. d.ddE+dd
- für jede Variable muss ein Formatbezeichner vorhanden sein :
  printf("Die Summe von %d und %d ist %d \n",x,y,z);
  Programmausgabe: *Die Summe von 5 und 4 ist 9.*

printf-Funktion (Beispiel)

```
/* Beispiel fuer printf-Funktion */
#include <stdio.h>
int main(void)
{
    int x=10;
    float f=12.25;
    char ergebnis='j';
    printf("Nur Text wird ausgegeben. \n");
    printf("Eine Integer-Variable hat den Wert %d. \n",x);
    printf("Eine float-Variable hat den Wert %f. \n",f);
    printf("Textausgabe mit einem Zeichen: %c. \n",ergebnis);
    printf("Prozentzeichen ausgeben: %%.\n");
    return 0;
}
```

- häufige Fehlerquelle:
  - Datentypen vom Formatbezeichner und Variablen stimmen nicht überein

printf-Funktion (Beispiel)

```
#include <stdio.h>
int main(void)
{
int x=10;
float f=12.25;
char ergebnis='j';
printf("Nur Text wird ausgegeben \n");
printf("Eine Integer-Variable hat den Wert %d
      \n",x);
printf("Eine float-Variable hat den Wert %f \n",f);
printf("Textausgabe mit einem Zeichen:
      %c\n",ergebnis);
printf("Prozentzeichen ausgeben %%\n");
return 0;

}
```

Programmübersetzung:

gcc -o p2 printf.c

Programm starten:

./p2

Programmausgabe:

Ein Text wird ausgegeben.
Eine Integer-Variable hat den Wert 10.
Eine float-Variable hat den Wert 12.25
Textausgabe mit einem Zeichen j.
Prozentzeichen ausgeben: %

printf-Funktion (Beispiel mit Fehler)

```
#include <stdio.h>
int main(void)
{
    int x=10;
    float f=12.25;

    printf("Ein Integer hat den Wert %f. \n",x);
    printf("Eine float-Variable hat den Wert %d. \n",f);

    return 0;
}
```

- häufige Fehlerquelle:
  - Datentypen vom Formatbezeichner und Variablen stimmen nicht überein

printf-Funktion (Beispiel mit Fehler)

```
#include <stdio.h>
int main(void)
{
int x=10;
float f=12.25;
printf("Ein Integer hat den Wert %f \n",x);
printf("Eine float-Variable hat den Wert
    %fd\n",f);
return 0;

}
```

Programmausgabe:

Ein Integer hat den Wert 0.00.

Eine float-Variable hat den Wert -154690

→ **falsche Formatbezeichner**

→ Übersetzung: keine Fehlermeldung (aber Warnung)

printf-Funktion: Aufruf

- Werte, die an printf-Funktion übergeben werden, nennt man Argumente
- als erstes Argument von printf → nur Strings erlaubt (“ “)
- printf(55)
  - Compilerwarnung und Fehler bei Ausführung (Speicherzugriffsfehler)
- printf(“55“) → korrekt: Ausgabe Zahl 55

Escape-Sequenzen

- nicht druckbare Steuerzeichen
- Zeichen, die nicht direkt auf Bildschirm sichtbar werden, sondern eine bestimmte Aufgabe erfüllen
- beginnen mit einem Backslash (\)
- Beispiele:
  - \n bewegt den Cursor auf die Anfangsposition der nächsten Zeile
  - \" gibt das Zeichen " aus
  - \' gibt das Zeichen ' aus
  - \? gibt das ? aus
  - \\ gibt den Backslash (\) aus
  - \t horizontaler Tabulator
  - \v vertikaler Tabulator

Daten einlesen

- erfolgt mit der Bibliotheksfunktion scanf
- allemeine Syntax:
  ```
  #include<stdio.h>
  scanf(Formatstring);
  ```
- Beispiele:
  ```
  scanf("%d", &xwert);   ---> Integer-Zahl einlesen
  scanf("%f", &ywert);  ---> Gleitkommazahl einlesen
  scanf("%d %f", &xwert, &ywert);  ---> 2 Variablen einlesen
  ```
- Adressoperator &: gibt die genaue Adresse im Speicher an
- beim scanf-Aufruf nur Formatbezeichner innerhalb “ “; kein zusätzlicher Text
- keine Leerzeichen zwischen Formatbezeicher und “
  - nicht: scanf(“%d “); → nur scanf(“%d”);
  - →kein Fehler beim Compilieren, wird versucht weitere Variable einzulesen

scanf-Funktion (Beispiel)

```
/* Daten einlesen mit scanf */
#include <stdio.h>
int main(void)
{
    int xwert;
    float ywert;
    printf("Bitte geben Sie eine Integer- und eine Gleitkommazahl ein.\n");
    scanf("%d %f", &xwert,&ywert);
    printf("Ihre Eingabe lautet: %d und %f. \n",xwert,ywert);
    return 0;
}
```

scanf-Funktion (Beispiel)

```
/* Daten einlesen mit scanf */
#include <stdio.h>
int main(void)
{
int xwert;
float ywert;
printf("Bitte geben Sie eine Integer- und eine
      Gleitkommazahl ein.\n");
scanf("%d %f", &xwert,&ywert);
printf("Ihre Eingabe lautet: %d und %f.
      \n",xwert,ywert);
return 0;
}
```

- Programm übersetzen:
  - gcc –o scanf scanf.c
- Programm starten:
  - ./scanf
- Programmausgabe

Bitte geben Sie eine Integer- und eine Gleitkommazahl ein.

2 10.25

Ihre Eingabe lautet: 2 und 10.250000.

getchar: Einlesen einzelner Zeichen

- getchar(); Funktion
- nur zum Einlesen einzelner Zeichen geeignet
- kein Formatbezeichner erforderlich

- Beispiel:

```
char c;
printf("Mit welchem Buchstaben beginnt ihr Vorname? \n ");
c = getchar();
printf("Ich weiss jetzt, dass Ihr Vorname mit '%c' beginnt.\n", c);
```

Mit welchem Buchstaben beginnt ihr Vorname?
K
Ich weiss jetzt, dass Ihr Vorname mit 'K' beginnt.

Unformatierte Ausgabe

```
#include <stdio.h>
int main(void)
{
    int x=10,y=12345;z=555;
    float f1=12.25,f2=3.333,f3=0.56789;
    printf (int-Werte: \n”);
    printf(“%d \n”,x);
    printf(“%d \n”,y);
    printf(“%d \n”,z);
    printf(“float-Werte: \n”);
    printf(“%f \n”,f1);
    printf(“%f \n”,f2);
    printf(“%f \n”,f3);
    return 0;
}
```

Unformatierte Ausgabe

```
#include <stdio.h>
int main(void)
{
int x=10,y=12345;z=555;
float f1=12.25,f2=3.333,f3=0.56789;
printf (int-Werte: \n”);
printf(“%d \n”,x);
printf(“%d \n”,y);
printf(“%d \n”,z);
printf(“float-Werte: \n”);
printf(“%f \n”,f1);
printf(“%f \n”,f2);
printf(“%f \n”,f3);
return 0;
}
```

- Programmausgabe:

  int-Werte:
  10
  12345
  555
  float-Werte:
  12.250000
  3.333000
  0.567890

Formatierte Ausgabe

```
#include <stdio.h>
int main(void)
{
    int x=10,y=12345;z=555;
    float f1=12.25,f2=3.333,f3=0.56789;
    printf (int-Werte: \n”);
    printf(“%10d \n”,x);
    printf(“%10d \n”,y);
    printf(“%10d \n”,z);
    printf(“float-Werte: \n”);
    printf(“%10.4f \n”,f1);
    printf(“%10.4f \n”,f2);
    printf(“%f10.4 \n”,f3);
    return 0;
}
```

Formatierte Ausgabe

```
#include <stdio.h>
int main(void)
{
int x=10,y=12345;z=555;
float f1=12.25,f2=3.333,f3=0.56789;
printf (int-Werte: \n”);
printf(“%10d \n”,x);
printf(“%10d \n”,y);
printf(“%10d \n”,z);
printf(“float-Werte: \n”);
printf(“%10.4f \n”,f1);
printf(“%10.4f \n”,f2);
printf(“%10.4f \n”,f3);
return 0;
}
```

- Programmausgabe:

```
int-Werte:
        10
     12345
       555
float-Werte:
   12.2500
    3.3330
    0.5679
```

Formatierte Ausgabe

```
#include <stdio.h>
int main(void)
{
int x=10,y=12345;z=555;
printf (int-Werte mit Leerzeichen: \n”);
printf(“%5d \n”,x);
printf(“%5d \n”,y);
printf(“%5d \n”,z);
printf(“int-Werte mit Nullen: \n”);
printf(“%05d\n”,x);
printf(“%05d\n”,y);
printf(“%05d \n”,z);
return 0;
}
```

- Programmausgabe:

```
int-Werte mit Leerzeichen:
   10
12345
  555
int-Werte mit Nullen:
00010
12345
00555
```

Zusammenfassung: Ein- und Ausgabe

- Bildschirmausgabe mit der printf-Funktion:
  - Textausgabe
    - printf("Dies ist eine reine Textausgabe.\n");
  - Text und Variablenwerte ausgeben:
    - printf("Der Wert von x ist %d \n",xwert);

- Einlesen von der Tastatur
  - z.B. scanf("%d",&xwert);

- Formatbezeichner:
  - %d: Integer-Zahl
  - %f: Gleitkommazahl
  - %s: Zeichenstring
  - %c: Zeichen
  - *%%: Prozentzeichen (%) ausgeben*

- Bibliotheksfunktion stdio.h einfügen
  - #include <stdio.h>

Operatoren

- sind Symbole zum Verknüpfen von Zahlen, Variablen und Ausdrücken
  - z.B.: +, -, *, /
- können einen, zwei oder drei Operanden miteinander verbinden
- Ein Operator ist ein Symbol, mit dem man in C eine Operation oder Aktion auf einen oder mehreren Operanden vorschreibt.
- Ein Operand ist ein Element, das der Operator verarbeitet:
  - Beispiel: (3+2)*8
- C-Operatoren lassen sich in mehrere Kategorien einteilen:
  - Zuweisungsoperator
  - arithmetische Operatoren
  - relationale Operatoren
  - logische Operatoren

Zuweisungsoperator

- der Zuweisungsoperator ist das Gleichheitszeichen (=)
- allgemeine Form der Wertzuweisung:

  var = ausdruck;

  z.B. summe = x + y;
- Auswertung erfolgt von rechts nach links
  - nach der Auswertung des Ausdrucks auf der rechten Seite wird sein Wert der Variablen auf der linken Seite zugewiesen
- rechte Seite kann beliebiger Ausdruck sein
- linke Seite muss ein Variablenname sein
- Zuweisungsoperator darf nicht mit dem Gleichheitszeichen verwechselt werden

Arithmetische Operatoren

- 8 mathematische Operatoren
  - fünf binäre Operatoren
  - zwei unäre Operatoren
  - ein ternärer Operator

Binäre arithmetische Operatoren

- wirken immer auf zwei Operanden
- 5 binäre Operatoren:
  - Addition: +
  - Subtraktion: -
  - Multiplikation: *
  - Division: /
  - Modulo-Operator: %

Modulo-Operator

- Modulo-Operator %
- liefert den Rest einer ganzzahligen Division
- Beispiele:
  100 % 9 ---> 1
  10 % 5 ---> 0
  40 % 6 ---> 4
- nur auf Integer-Zahlen anwendbar und nicht auf Gleitkommazahlen

Operatoren (Beispiel)

```
/* Beispiel für Modulo-Operator */
#include <stdio.h>
int main(void)
{
    int x=5, y=2;
    int d1, d2;
    d1=x/y;
    d2=x%y;
    printf("Division: %d / %d = %d \n",x,y,d1);
    printf("Modulo: %d %% %d = %d \n",x,y,d2);
    return 0;
}
```

Operatoren (Beispiel)

```
/* Beispiel für Modulo-Operator */
#include <stdio.h>
int main(void)
{
int x=5, y=2;
int d1, d2;
d1=x/y;
d2=x%y;
printf("Division: %d / %d = %d \n",x,y,d1);
printf"(Modulo: %d %% %d = %d
    \n",x,y,d2);
return 0;
}
```

- Programm übersetzen:
  gcc -o mod mod1.c
- Programm ausführen:
  ./mod
- Programmausgabe
  Division 5 /2 = 2
  Modulo 5 %2 = 1

Division: Beispiel


```
#include <stdio.h>
int main ()
{
    int var1=100;
    int var2=0;
    int var3=0;
    var3=var1/var2;
    return 0;
}
```

Division: Beispiel

```
#include <stdio.h>
int main ()
{
    int var1=100;
    int var2=0;
    int var3=0;
    int var4=0;
    var3=var1/var2;
    var4=var1%var2;
    return 0;
}
```

→ Laufzeitfehler:

→ Gleitkomma-Ausnahme (Speicherabzug geschrieben)

Rechenarten in C

- verfügbar:
  - Addition, Multiplikation, Division und Modulo
- andere Rechenoperation (z.B. Wurzel- und Winkelfunktionen)
  - werden in C über Bibliotheken bereit gestellt
- kein Operator für Potenzierung (z.B. ^)
  - auch über Bibliotheksfunktion

Mathematische Bibliotheksfunktionen

- Laufzeitbibliothek math.h enthält alle wichtigen mathematischen Funktionen
  - Beispiele:
    - Quadratwurzel sqrt
    - Potenzfunktion pow
    - Sinus, Kosinus und Tangens
- #include <math.h> in Code einfügen
- Variablen sind stets vom Datentyp double
  - Formatbezeichner %lf bei Ausgabe

- für Programmübersetzung mit gcc ist -lm Compilerflag erforderlich
  - gcc -o prog prog.c -lm

  - ohne –lm: bei Übersetzung Fehlermeldung: undefined reference to

Programmbeispiel: Quadratwurzel

```
/* Quadratwurzel bilden */
#include <stdio.h>
#include <math.h>
int main(void)
{
   double var2, var1;
   var1=169;
   var2=sqrt(var1);
   printf("Die Quadratwurzel von %lf ist %lf. \n", var1,var2);
   return 0;
}
```

Programmbeispiel: Quadratwurzel

```
/* Quadratwurzel bilden */
#include <stdio.h>
#include <math.h>
int main(void)
{
   double var2, var1;
   var1=169;
   var2=sqrt(var1);
   printf("Die Quadratwurzel von %lf ist %lf. \n",
     var1,var2);
return 0;
}
```

- Programm übersetzen
  gcc -o sqrt sqrt.c -lm
- Programm ausführen:
  ./sqrt
- Programmausgabe:
  Die Quadratwurzel von 169 ist 13.0000

Zusammenfassung: Operatoren I

- Zuweisungsoperator:
  - var = ausdruck1;

- binäre mathematische Operatoren
  - Addition: +
  - Subtraktion: -
  - Multiplikation *
  - Division: /
  - Modulo-Operator: % → Divisionsrest

- mathematische Bibliotheksfunktionen
  - Headerdatei einfügen : #include <math.h>
  - Variablen vom Datentyp double (Formatbezeichner %lf)
  - Programmübersetzung mit -lm Compilerflag

Datentypumwandlungen

- C ist eine sehr typenstrenge Programmiersprache
- Datentypen müssen konform sein
- Beispiel:

```
#include <stdio.h>
int main(void)
{
   int wert_int=0;
   float  wert_float=10.1234;
   wert_int=wert_float;
   printf("wert_int ist %d ist \n",wert_int);
   return 0;
}
```

Datentypumwandlungen

- C ist eine sehr typenstrenge Programmiersprache
- Datentypen müssen konform sein
- Beispiel:

```
#include <stdio.h>
int main(void)
{
   int wert_int=0;
   float  wert_float=10.1234;
   wert_int=wert_float;
   printf("wert_int ist %d ist \n",wert_int);
   return 0;
}
```

→ Ausgabe: wert_int ist 10

Datentypumwandlungen

- implizite Datentypumwandlung
  - automatische Konvertierung durch den Compiler
- explizite Datentypumwandlung
  - Programmierer führt Typumwandlung manuell aus

implizite Datentypumwandlung

- Ausdruck enthält Variablen mit verschiedenen Datentypen
  - Compiler wandelt Datentypen einzelner Operanden selbständig um
- Typumwandlungen bei Zuweisungen:
  - Wert auf der rechten Seite wird an den Wert auf der linken Seite angepasst
- Typumwandlung bei arithmetischen Ausdrücken
  - arithmetische Umwandlung
  - geringere Genauigkeit wird an genaueren Datentyp angepasst:
    - Beispiele:
      - y= 3.0 + 2.0 → y = 5.0
      - y= 3.0 +2 → y = 5.0
      - y = 22/11 → y=2
      - y= 22/11.0 → y = 2.0

explizite Datentypumwandlung

```
#include <stdio.h>
int main(void)
{
    int int1=3,int2=4;
    float ergebnis=0.00;
    ergebnis=int1/int2;
    printf("Das Ergebnis ist %f \n",ergebnis);
    return 0;
}
```

→ Ausgabe: Das Ergebnis ist 0.000 → Fehler

explizite Datentypumwandlung

- erfolgt mit dem cast-Operator
- (Zieldatentyp) ausdruck;
  - Zieldatentyp ist der Datentyp, zu dem ausdruck konvertiert wird
- Beispiel:

```
int x,y;
ergebnis=(float)x/y;
```

  - 1. Schritt: Umwandlung int x → float x (gilt nur für diese Anweisung)
  - 2. Schritt: Division durchführen → ergebnis ist float Zahl

- nicht auf Strings (Zeichenketten) anwendbar

explizite Datentypumwandlung

```
#include <stdio.h>
int main(void)
{
    int1=3,int2=4;
    float ergebnis=0.00;
    ergebnis=int1/int2;
    printf("Das Ergebnis ist %f
    \n",ergebnis);
    return 0;
}
```

→ Ausgabe: Das Ergebnis ist 0.000
→ Fehler

```
#include <stdio.h>
int main(void)
{
    int1=3,int2=4;
    float ergebnis=0.00;
    ergebnis=(float) int1/int2;
    printf("Das Ergebnis ist %f
    \n",ergebnis);
    return 0;
}
```

→Ausgabe: Das Ergebnis ist 0.75000

Explizite Datentypumwandlung: Beispiel 2

```
#include <stdio.h>
Int main(void)
{
   float wert1, wert2;
   wert1=(float) 3/2;
   wert2= (float) (3/2);
   printf(“wert1 ist %f \n“,wert1);
   printf (“wert2 ist %f \n“,wert2):
   return 0;
}
```

Explizite Datentypumwandlung: Beispiel 2

```
#include <stdio.h>
int main(void)
{
   float wert1, wert2;
   wert1=(float) 3/2;
   wert2= (float) (3/2);
   printf("wert1 ist %f \n",wert1);
   printf ("wert2 ist %f \n",wert2):
   return 0;
}
```

Ausgabe:

```
wert1 ist 1.5000
wert2 ist 1.0000
```

Arithmetische Operatoren

- 7 mathematische Operatoren
  - zwei unäre Operatoren
  - fünf binäre Operatoren

Unäre mathematische Operatoren

- unäre Operatoren wirken nur auf einen Operanden
- sind nur auf Variablen und nicht auf Konstanten anwendbar
- in C gibt es zwei unäre Operatoren:
  - Inkrementoperator: ++
    - erhöht einen Operanden um 1
    - variable = variable +1;
  - Dekrementoperator: --
    - erniedrigt einen Operanden um 1
    - variable = variable – 1;
- x++ hat eine andere Bedeutung als ++x
- unäre Operatoren können sowohl vor dem Operanden (Präfix-Modus) oder nach dem Operanden (Postfix-Modus) stehen.
- können nur auf einfache Variablen angewendet werden, nicht dagegen auf Ausdrücke wie etwa (i+j)++;
- werden vorwiegend bei Schleifen verwendet

Präfix- und Postfix-Modus

- Präfixmodus (++x)
  - Inkrement-/Dekrementoperator wirkt auf den Operanden, bevor er verwendet wird
  - Variable wird erst dekrementiert/inkrementiert und anschließend einer Variablen zugewiesen
  - ++x ist äquivalent zu x=x+1;
  - --y ist äquivalent zu y=y-1;

- Postfix-Modus (x++)
  - Inkrement-/Dekrementoperator wirkt auf den Operanden, nachdem er verwendet wurde
  - z.B.: y=x++
  - Wert der Variablen rechts vom Zuweisungsoperator wird der Variablen links zugewiesen. Anschließend wird die Variable dekrementiert / inkrementiert

Präfix- und Postfix-Modus (Beispiel)

- Inkrementierung
  - Präfix: x=10 und y=++x; ---> x=11; und y=11;
  - Postfix: x=10 und y=x++; ----> x=11; und y=10;

- Dekrementierung
  - Präfix: x=10 und y=--x ; ---> x=9; und y=9;
  - Postfix: x=10 und y=x--; --> x=9; und y=10;

- Beispiel:
  int itemp=5, icount=3, isum=0
  isum = itemp-- * icount
  isum = ? itemp=?

Präfix- und Postfix-Modus (Beispiel)

- Inkrementierung
  - Präfix: x=10; und y= ++x; ---> x=11; und y=11;
  - Postfix: x=10 und y=x++; ----> x=11; und y=10;

- Dekrementierung
  - Präfix: x=10; und y=--x ; ---> x=9; und y=9;
  - Postfix: x=10 und y=x--; --> x=9; und y=10;

- Beispiel:
  int itemp=5, icount=3, isum=0
  isum = itemp-- * icount
  isum = 15 und itemp=4

Operatoren (Beispiel)

```
/* Beispiel für In-/Dekrementoperationen */
#include <stdio.h>
int main(void)
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

Operatoren (Beispiel)

```
/* Beispiel für In-/Dekrementoperationen
   */
#include <stdio.h>
int main(void)
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

- Programmausgabe:

```
Wert von x: 0
```

Operatoren (Beispiel)

```
/* Beispiel für In-/Dekrementoperationen
   */
#include <stdio.h>
int main(void)
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

- Programmausgabe:

```
Wert von x: 0
Wert von x: 1
```

Operatoren (Beispiel)

```
/* Beispiel für In-/Dekrementoperationen
   */
#include <stdio.h>
int main(void)
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

- Programmausgabe:

```
Wert von x: 0
Wert von x: 1
Wert von x: 1
```

Operatoren (Beispiel)

```
/* Beispiel für In-/Dekrementoperationen
   */
#include <stdio.h>
int main(void)
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

- Programmausgabe:

```
Wert von x: 0
Wert von x: 1
Wert von x: 1
Wert von x: 2
```

Operatoren (Beispiel)

```
/* Beispiel für In-/Dekrementoperationen
   */
#include <stdio.h>
int main(void)
{
    int x=0;
    printf("Wert von x: %d\n",x);
    printf("Wert von x: %d\n",++x);
    printf("Wert von x: %d\n", x++);
    printf("Wert von x: %d\n", x);
    printf("Wert von x: %d\n",x--);
    printf("Wert von x: %d\n", --x);
    return 0;
}
```

- Programmausgabe:

```
Wert von x: 0
Wert von x: 1
Wert von x: 1
Wert von x: 2
Wert von x: 2
```

Operatoren (Beispiel)

```
/* Beispiel für In-/Dekrementoperationen
   */
#include <stdio.h>
int main(void)
{
int x=0;
printf("Wert von x: %d\n",x);
printf("Wert von x: %d\n",++x);
printf("Wert von x: %d\n", x++);
printf("Wert von x: %d\n", x);
printf("Wert von x: %d\n",x--);
printf("Wert von x: %d\n", --x);
return 0;
}
```

- Programmausgabe:

```
Wert von x: 0
Wert von x: 1
Wert von x: 1
Wert von x: 2
Wert von x: 2
Wert von x: 0
```

Rangfolge der Operatoren

- In C hat jeder Operator eine bestimmte Priorität
- Wert von x in diesem Beispiel ist ?
  - x=4+5*3;
- Priorität nimmt in der folgenden Reihenfolge ab:
  - Inkrement-/Dekrementoperationen
  - Multiplikation, Division und Modulo-Operator
  - Addition und Subtraktion
- Operatoren mit höherer Priorität werden zuerst ausgewertet
- bei gleicher Priorität erfolgt die Auswertung von links nach rechts
- Verwenden Sie Klammern, um die Auswertungsreihenfolge eindeutig festzulegen

Relationale Operatoren

- Vergleichsoperatoren (dienen zum Vergleich von Ausdrücken)
- Ergebnis ist entweder wahr oder falsch
- Ergebnis falsch entspricht Zahlenwert von Null
- Ergebnis wahr entspricht Zahlenwert von ungleich Null meist 1
- werden häufig für Bedingungen und Schleifen verwendet
- 6 Vergleichsoperatoren
  - größer als (>): Bsp.: 9>5 --> wahr
  - größer gleich (>=) : Bsp.: 9 >=5 --> wahr
  - kleiner (<): Bsp.: 9 <5 --> falsch
  - kleiner gleich (<=): Bsp.: 9 <=5 --> falsch
  - gleich (==): Bsp.: 9==5 --> falsch
  - ungleich (!=): Bsp.: 0!=5 --> wahr
- Operatoren == und != haben geringere Priorität als die anderen Operatoren
  x == y> z entspricht x == (y>z)

Vergleichsoperatoren (Beispiel)

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main(void)
{
    int a=5, b=12, c=7;
    printf("Ergebnis ist %d\n",a<b);
    printf("Ergebnis ist %d\n", b<c);
    printf("Ergebnis ist %d\n",a+c<=b);
    printf("Ergebnis ist %d\n",b-a>=c);
    printf("Ergebnis ist %d\n",a==b);
    printf("Ergebnis ist %d\n",a+c==b);
    printf("Ergebnis ist %d\n",a!=b);
    printf("Ergebnis ist %d\n",a<b<c);
    return 0;
}
```

Vergleichsoperatoren (Beispiel)


```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main(void)
{
int a=5, b=12, c=7;

printf("Ergebnis ist %d\n",a<b);  ---> Ergebnis ist 1
printf("Ergebnis ist %d\n", b<c);
printf("Ergebnis ist %d\n",a+c<=b);
printf("Ergebnis ist %d\n",b-a>=c);
printf("Ergebnis ist %d\n",a==b);
printf("Ergebnis ist %d\n",a+c==b);
printf("Ergebnis ist %d\n",a!=b);
printf("Ergebnis ist %d\n",a<b<c);
return 0;
}
```

Vergleichsoperatoren (Beispiel)

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main(void)
{
int a=5, b=12, c=7;

printf("Ergebnis ist %d\n",a<b);  ---> Ergebnis ist 1
printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
printf("Ergebnis ist %d\n",a+c<=b);
printf("Ergebnis ist %d\n",b-a>=c);
printf("Ergebnis ist %d\n",a==b);
printf("Ergebnis ist %d\n",a+c==b);
printf("Ergebnis ist %d\n",a!=b);
printf("Ergebnis ist %d\n",a<b<c);
return 0;
}
```

Vergleichsoperatoren (Beispiel)

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main(void)
{
int a=5, b=12, c=7;

printf("Ergebnis ist %d\n",a<b);  ---> Ergebnis ist 1
printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
printf("Ergebnis ist %d\n",b-a>=c);
printf("Ergebnis ist %d\n",a==b);
printf("Ergebnis ist %d\n",a+c==b);
printf("Ergebnis ist %d\n",a!=b);
printf("Ergebnis ist %d\n",a<b<c);
return 0;
}
```

Vergleichsoperatoren (Beispiel)

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main(void)
{
int a=5, b=12, c=7;

printf("Ergebnis ist %d\n",a<b);  ---> Ergebnis ist 1
printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
printf("Ergebnis ist %d\n",b-a>=c); ---> Ergebnis ist 1
printf("Ergebnis ist %d\n",a==b);
printf("Ergebnis ist %d\n",a+c==b);
printf("Ergebnis ist %d\n",a!=b);
printf("Ergebnis ist %d\n",a<b<c);
return 0;
}
```

Vergleichsoperatoren (Beispiel)

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main(void)
{
int a=5, b=12, c=7;

printf("Ergebnis ist %d\n",a<b);  ---> Ergebnis ist 1
printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
printf("Ergebnis ist %d\n",b-a>=c); ---> Ergebnis ist 1
printf("Ergebnis ist %d\n",a==b); ---> Ergebnis ist 0
printf("Ergebnis ist %d\n",a+c==b);
printf("Ergebnis ist %d\n",a!=b);
printf("Ergebnis ist %d\n",a<b<c);
return 0;
}
```

Vergleichsoperatoren (Beispiel)

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main(void)
{
int a=5, b=12, c=7;

printf("Ergebnis ist %d\n",a<b);  ---> Ergebnis ist 1
printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
printf("Ergebnis ist %d\n",b-a>=c); ---> Ergebnis ist 1
printf("Ergebnis ist %d\n",a==b); ---> Ergebnis ist 0
printf("Ergebnis ist %d\n",a+c==b); ----> Ergebnis ist 1
printf("Ergebnis ist %d\n",a!=b);
printf("Ergebnis ist %d\n",a<b<c);
return 0;
}
```

Vergleichsoperatoren (Beispiel)

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main(void)
{
int a=5, b=12, c=7;

printf("Ergebnis ist %d\n",a<b);  ---> Ergebnis ist 1
printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
printf("Ergebnis ist %d\n",b-a>=c); ---> Ergebnis ist 1
printf("Ergebnis ist %d\n",a==b); ---> Ergebnis ist 0
printf("Ergebnis ist %d\n",a+c==b); ----> Ergebnis ist 1
printf("Ergebnis ist %d\n",a!=b); ---> Ergebnis ist 1
printf("Ergebnis ist %d\n",a<b<c);
return 0;
}
```

Vergleichsoperatoren (Beispiel)

```
/* Beispiel fuer Vergleichsoperatoren */
#include <stdio.h>
int main(void)
{
    int a=5, b=12, c=7;

    printf("Ergebnis ist %d\n",a<b);  ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n", b<c); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c<=b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",b-a>=c); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a==b); ---> Ergebnis ist 0
    printf("Ergebnis ist %d\n",a+c==b); ----> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a!=b); ---> Ergebnis ist 1
    printf("Ergebnis ist %d\n",a<b<c); ---> Ergebnis ist 1
    return 0;
}
```

Hinweis: Vergleichsoperator = =

- Verwechseln Sie Zuweisungsoperator = nicht mit Vergleichsoperator = =
  - Unterschiedliche Bedeutung
    - = weist einer Variablen einen Wert zu
    - == vergleicht zwei Werte miteinander
- Compiler gibt bei Verwechselung keine Warnung oder Fehlermeldung aus, evtl. erhält man jedoch ein falsches Ergebnis

- Tipp:
  - bei Vergleichen Variablennamen auf der rechten Seite schreiben, also 5==a
    - wird ein = vergessen → Compiler liefert Fehlermeldung

Logische Operatoren

- verwendet, wenn mehrere Vergleiche auf einmal ausgeführt werden sollen
- mehrere relationale Ausdrücke werden zu einem Ausdruck zusammen gefasst
- 3 logische Operatoren:
  - AND-Operator (&&)
    - ausdruck1 && ausdruck2
    - nur wahr, wenn ausdruck1 und ausdruck2 wahr sind
    - Bsp.: (5==5) && (6!=2)
  - OR-Operator (||)
    - ausdruck1 || ausdruck2
    - wahr sobald mind. eine Bedingung wahr ist
    - Bsp.: ((5>1)|| (6<1)
  - NOT-Operator (!)
    - !(ausdruck)
    - kehrt Wahrheitswert des Ausdrucks um
      - falsch, wenn ausdruck wahr ist
    - Bsp.: !(5==4)

Zusammengesetzte Zuweisungsoperatoren

- Möglichkeit eine binäre mathematische Operation mit einer Zuweisung zu kombinieren
- z.B.: x=x+5; identisch mit x += 5;
- allg. Form: ausd1 op= ausd2
- Operator += heißt verkürzter Zuweisungsoperator
- anwendbar auf: +, -, *, /, %
- Beispiele:
  - x += y; ---> x = x +y;
  - y -= z+1 --> y = y -z +1;
  - a /=b; --> a = a / b;
  - x += y/8; -> x=x+y/8;
  - y %= 3; ---> y = y %3;
- reduzieren den Schreibaufwand

Operatoren: Rangfolge

- Operatoren mit höherer Priorität haben Vorrang

- Priorität nimmt in folgender Reihenfolge ab:
  1. unäre Inkrement- und Dekrementoperationen
  2. Multiplikation, Division und Modulo-Operator
  3. Addition und Subtraktion
  4. Vergleichsoperatoren: <, >, <= und >=
  5. Vergleichsoperatoren: = = und !=
  6. Logische Operatoren: &&, || und !

Zusammenfassung: Operatoren

- Zuweisungsoperator
  - =
- unäre mathematische Operatoren
  - x++, ++x, x-- und --x
- binäre mathematische Operatoren:
  - +, -, *, / und %
- Vergleichsoperatoren:
  - <, >, <=, <=, == und !=
- logische Operatoren
  - &&, || und !
- Rangfolge der Operatoren beachten

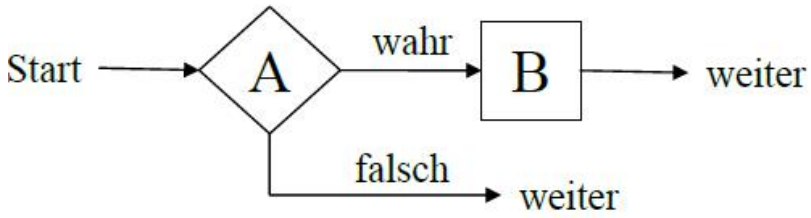
Kontrollstrukturen

- beeinflussen den Programmablauf
  - ermöglichen Abweichungen vom streng linearen Verlauf
  - z.B. kann auf Benutzereingaben unterschiedlich reagiert werden
  - in Abhängigkeit vom Wert einer/mehrerer Variablen wird Programmablauf gesteuert
- in C gibt es folgende Kontrollstrukturen
  - Auswahlanweisungen:
    - bedingte Anweisungen: if-else-Anweisung
    - Fallunterscheidungen: switch-Anweisung
  - *Schleifen (Iterationsanweisungen):*
    - *kopfgesteuerte Schleife: while-Schleife*
    - *fußgesteuerte Schleife: do-while-Schleife*
    - *Zählschleife: for-Schleifen*

if-Anweisung

- Code wird nur unter bestimmten Bedingungen ausgeführt
- allgemeine Form

```
if (Bedingung A)
{
    Anweisung1;
    Anweisung2;
}
    Anweisung3;
```



- Anweisungen 1 und 2 werden nur ausgeführt, wenn die Bedingung A wahr ist
- wenn Bedingung A nicht wahr ist, wird gleich Anweisung 3 ausgeführt
- nur eine Anweisung in if-Konstrukt → { } kann entfallen

```
if (Bedingung)
    Anweisung1;
Anweisung3;
```

if-Anweisung (Beispiel)

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main(void)
{
    int x,y;
    printf("Bitte geben Sie zwei Integer-Zahlen ein.\n");
    scanf("%d %d", &x, &y);
    if (x == y)
      {printf("x ist gleich y \n"); }
    return 0;
}
```

if-Anweisung (fehlerhaftes Beispiel)

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main(void)
{
int x,y;
printf("Bitte geben Sie zwei Integer-Zahlen ein.\n");
scanf("%d %d",&x, &y);
if (x == y) ;
   {printf("x ist gleich y \n"); }
return 0;
}
```

if-Anweisung (fehlerhaftes Beispiel)

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main(void)
{
int x,y;
printf("Bitte geben Sie zwei Integer-Zahlen ein,\n");
scanf("%d %d",&x, &y);
if (x == y) ;
   {printf("x ist gleich y \n");} --> Zeile wird immer ausgegeben
return 0;
}
```

if-Anweisung (fehlerhaftes Beispiel)

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main(void)
{
int x,y;
printf("Bitte geben Sie zwei Integer-Zahlen ein,\n");
scanf("%d %d",&x, &y);
if (x == y) ;
   {printf("x ist gleich y \n");} --> Zeile wird immer ausgegeben
return 0;
}
```

--> if-Anweisungszeile nie mit einem Semikolon abschließen !!

if-Anweisung (erweitertes Beispiel)

```
/* Beispiel fuer if-Bedingung */
#include <stdio.h>
int main(void)
{
int x,y;
printf("Bitte geben Sie zwei Integer-Zahlen ein,\n");
scanf("%d %d",&x, &y);
if (x == y)
   printf("x ist gleich y \n");
if(x>y)
   printf("x ist groesser als y \n ");
if (x<y)
   printf("x ist kleiner als y \n");
return 0;
}
```

if-else-Anweisung

- wird verwendet, wenn mehrere Auswahlmöglichkeiten vorliegen
- allgemeine Struktur

```
if (Bedingung1)
        { Anweisung1; }
else if (Bedingung2)
        { Anweisung2; }
else
        { Anweisung3; }
 naechste-Anweisung;
```

- else-Klausel ist optional

if-else-Anweisung

- Vorteil einer else-if gegenüber zweier if-Bedingungen
  - Bedingung muss nur einmal überprüft werden
  - schnellere Programmausführung
- mehrere else-if Anweisungen möglich, aber nur eine else-Anweisung

- allgemeine Struktur

```
if (Bedingung1)
        { Anweisung1; }
else if (Bedingung2)
        { Anweisung2; }
else
        { Anweisung3; }
```

if-else-Anweisung (Beispiel)

```
#include <stdio.h>
int main(void)
{
    int x,y;
    printf("Bitte geben Sie zwei Integer-Zahlen ein,\n");
    scanf("%d %d",&x, &y);
      if (x == y)
            printf("x ist gleich y \n");
      else if (x>y)
            printf("x ist groesser als y \n");
      else
            printf("x ist kleiner als y \n");
       return 0;
}
```

switch-Anweisung

- dient wie if-Bedingungen zur Formulierung von Mehrwegeentscheidungen
- wird verwendet wenn eine unter vielen Bedingungen ausgewählt werden soll
- Wert eines Ausdrucks wird mit einer Liste von Konstanten vom Datentyp char, int oder long verglichen
- jeder Vergleich wird mit case (Fall) eingeleitet
- es ist nur ein Vergleich auf Gleichheit möglich

switch-Anweisung

- allgemeine Form:

```
switch (Ausdruck)
 {
      case konstante1:
            anweisung1;
            break;
      case konstante2:
            anweisung2;
            break;
      ....
      case konstanten:
            anweisung;
            break;
      default:
             anweisung;
}
naechste-Anweisung;
```

switch-Anweisung (Beispiel)

```
#include <stdio.h>
int main(void)
{
   char eingabe;
   printf(“Fahren Sie mit dem Bus zur Uni (j/n)? \n");
   scanf("%c", &eingabe);
   switch (eingabe)
   {
        case ‘j' :
           printf(“Anfahrt erfolgt mit dem Bus\n");
           break;
       case ‘n':
           printf(“keine Nutzung des Busses\n");
           break;
       default:
           printf("falsche Eingabe \n") ;
   }
return 0;}
```

switch-Anweisung (fehlerhaftes Beispiel)

```
#include <stdio.h>
int main(void)
{
   char eingabe;
   printf(“Fahren Sie mit dem Bus zur Uni (j/n)? \n");
   scanf("%c", &eingabe);
   switch (eingabe)
   {
        case ‘j' :
           printf(“Anfahrt erfolgt mit dem Bus\n");
      case ‘n':
           printf(“keine Nutzung des Busses\n");
      default:
           printf("falsche Eingabe \n") ;
   }
return 0;}
```

switch-Anweisung (fehlerhaftes Beispiel)

```
#include <stdio.h>
int main(void)
{
   char eingabe;
   printf(“Fahren Sie mit dem Bus zur Uni (j/n)?
      \n");
   scanf("%c", &eingabe);
   switch (eingabe)
   {
       case ‘j' :
          printf(“Anfahrt erfolgt mit dem Bus\n");
      case ‘n':
          printf(“keine Nutzung des Busses\n");
      default:
          printf("falsche Eingabe \n") ;
   }
return 0;}
```

- Programmausgabe

  gcc swtich.c –o switchbsp

  ./switchbsp

  Fahren Sie mit dem Bus zur Uni (j/n)?

  j
  Anfahrt erfolgte mit dem Bus
  keine Nutzung des Busses
  falsche Eingabe

  n
  Anfahrt erfolgte mit dem Bus
  keine Nutzung des Busses
  falsche Eingabe

  → ohne break werden alle Fälle ausgeführt

switch-Anweisung: Beispiel 2

```
int tag;
printf("Der wievielte Tag der Woche ist heute ?");
scanf ("%d",&tag);
switch (tag)
{case 1:
case 3:
case 5:
    printf("Heute ist Montag, Mittwoch oder Freitag. \n");
    break;
case 7:
    printf("Heute ist Sonntag \n");
    break;
…
```

→ break zwingend erforderlich wenn alle nachfolgenden case-Fälle übersprungen werden sollen

Bedingungsoperator

- einzige ternäre Operator in C
- allgemeine Form:
  Bedingung ? Ausdruck1 : Ausdruck2
- Kurzform einer if- mit einer alternativen else-Anweisung
- liefert abhängig von einer Bedingung einen von zwei möglichen Ergebniswerten:
  - Bedingung ist wahr → Ergebnis ist Ausdruck1
  - Bedingung ist falsch → Ergebnis ist Ausdruck 2
- Beispiel:

```
z = (x>y) ? x : y ;

if (x>y)
    z = x;
else
    z = y;
```

Bedingungsoperator

- Bedingung ? Ausdruck1 : Ausdruck2
- Unterschied Bedingungsoperator und if-else-Anweisung:
  - Bedingungsoperator hat immer einen Ergebniswert (<-> if-else)

  →kann in Formeln und Funktionsaufrufen verwendet werden

- nur mit Bedingungsoperator möglich:

  printf("Der größte Wert ist %d",((x > y) ? x : y));

- Bedingungsoperator kann verschachtelt werden
  - Ausdruck wird u.U. leicht unübersichtlich
  - Ergebnis des folgenden Ausdrucks ist?

    ergebnis=(a>b?(a>c?a:c):(b>c?b:c))

Kontrollstrukturen I: Zusammenfassung

- beeinflussen den Programmablauf
- in C gibt es folgende Kontrollstrukturen
  - Auswahlanweisungen:
    - bedingte Anweisungen: if-else-Anweisung
    - Fallunterscheidungen: switch-Anweisung

Kontrollstrukturen I: Zusammenfassung

- beeinflussen den Programmablauf
- in C gibt es folgende Kontrollstrukturen
  - Auswahlanweisungen:
    - bedingte Anweisungen: if-else-Anweisung
    - Fallunterscheidungen: switch-Anweisung
  - Schleifen (Iterationsanweisungen):
    - kopfgesteuerte Schleifen: while-Schleife
    - fußgesteuerte Schleifen: do-while-Schleife
    - Zählschleife: for-Schleifen

Kontrollstrukturen II

- Iterationsanweisungen (Schleifen)
- dienen zur wiederholten Ausführung von Anweisungen
- in C unterscheidet man drei unterschiedliche Schleifenarten
  - for-Schleife
  - while-Anweisung
  - do-while-Anweisung

Aufgabe:

- Code zu schreiben, der z.B. 50 mal eine Zeile berechnet bzw. ausgibt

- Möglichkeit:

```
printf("Textausgabe  \n");
printf("Textausgabe \n");
printf("Textausgabe \n");
printf("Textausgabe \n");
….
```

- Alternative:
  - Schleifenkonstrukte einsetzen

Schleifen: allgemeiner Aufbau

Aufgabe:

- Code zu schreiben, der z.B. 50 mal eine Zeile berechnet bzw. ausgibt

```
printf("Textausgabe  \n");
printf("Textausgabe \n");
printf("Textausgabe \n");
....
```

- Bedingung:
  - Code nur ausführen wenn eine Bedingung erfüllt ist
- Zählvariable (Schleifenvariable, Schleifenzähler)
  - Initialisierung: Startwert festlegen
  - Inkrementierung: Erhöhung nach jeder Ausführung

for-Schleife

- allgemeine Struktur:

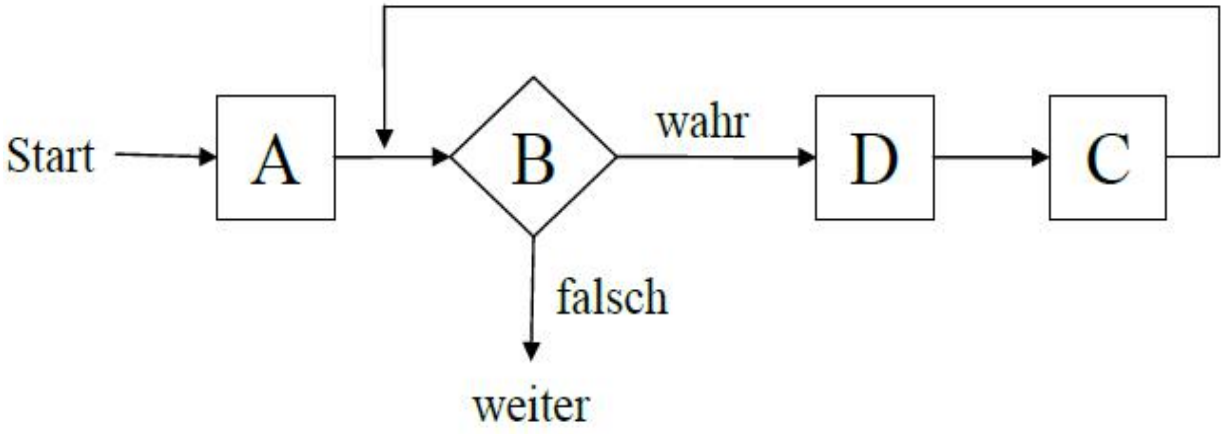
```
for (Initialisierung; Bedingung; Inkrement)
    {
      Anweisungen
     }
naechste-Anweisung;
```

- Initialisierung: legt Anfangswert für Zählvariable fest
  - Ausführung nur bei erstem Durchlauf
- Bedingung: Schleife bricht ab, wenn Bedingung nicht mehr erfüllt ist
- Inkrement: legt fest, wie die Zählvariable nach jedem Durchlauf verändert wird
  - meist Addition oder Subtrakion, aber auch andere Rechenoperationen erlaubt
- wird verwendet, wenn die Anzahl der max. Schleifendurchläufe vorher bekannt ist
- for-Schleife ist eine Zählschleife
- Schleifen dürfen verschachtelt werden

for-Schleif: Flußdiagramm

- allgemeine Struktur:

```
for (Initialisierung A; Bedingung B; Inkrement C)
    {
     Anweisungen D
    }
naechste-Anweisung;
```



for-Schleife (Beispiel)

```
/* Beispiel für eine for-Schleife */
#include <stdio.h>
int main(void)
{
      int x=0;
      for (x=0; x<5;x++)
      {
          printf("Der Wert von x ist: %d \n",x);
      }
      return 0;
}
```

for-Schleife (Beispiel)

```
/* Beispiel für eine for-Schleife */
#include <stdio.h>
int main(void)
{
        int x=0;
        for (x=0; x<5;x++)
        {
         printf("Der Wert von x ist: %d. \n",x);
        }
        return 0;
}
```

- Programm übersetzen:
  - gcc -o for for.c
- Programm ausführen:
  - ./for
- Programmausgabe

  Der Wert von x ist 0.
  Der Wert von x ist 1.
  Der Wert von x ist 2.
  Der Wert von x ist 3.
  Der Wert von x ist 4.

while-Schleife

- allgemeine Form

```
while (Bedingung)
        {Anweisung(en);}
naechste-Anweisung;
```

- Anweisung wird solange ausgeführt solange die Bedingung wahr ist
- enthält keinen Initialisierungsschritt
- Initialisierung muss vor der Schleife erfolgen
- Inkrementierungsschritt muss innerhalb der Schleife erfolgen
- wird z.B. verwendet, wenn
  - Anzahl der Schleifendurchläufe vorher nicht bekann ist
  - Schleife bei einem bestimmten Ereignis abgebrochen werden soll
- Endlosschleife entsteht, wenn Schleifenvariable nicht richtig geändert wird
- wird auch als abweisende Schleife bezeichnet

while-Schleife (Beispiel)

```
/* Beispiel für eine while-Schleife */
#include <stdio.h>
int main(void)
{
   int eingabe;
   int zaehler=1;
   printf("Bitte Anzahl der Schleifendurchlaeufe eingeben. \n");
   scanf ("%d", &eingabe);
   while (zaehler<=eingabe)
   {
      printf("Schleifendurchlauf  %d  \n",zaehler);
      zaehler=zaehler+1;
    }
    printf("Programm beendet \n");
    return 0;
}
```

while-Schleife (Beispiel)

```
/* Beispiel für eine while-Schleife */
#include <stdio.h>
int main(void)
{
   int eingabe;
   int zaehler=1;
   printf("Bitte Anzahl der Schleifendurchlaeufe
      eingeben. \n");
   scanf ("%d", &eingabe);
   while (zaehler<=eingabe)
   {
       printf("Schleifendurchlauf  %d  \n",zaehler);
         zaehler=zaehler+1;
    }
    printf("Programm beendet \n");
    return 0;
}
```

- Programm übersetzen
  - gcc -o while while.c
- Programm aufführen:
  - ./while
- Programmausgabe:

```
Bitte Anzahl der Schleifendurchlaeufe
    eingeben.
4
Schleifendurchlauf 1
Schleifendurchlauf 2
Schleifendurchlauf 3
Schleifendurchlauf 4
Programm beendet
```

Vergleich: for- und while Schleife

```c
/* Beispiel für eine for-Schleife */
#include <stdio.h>
int main(void)
{
   int x;
   for (x=0; x<5;x++)
  {
      printf("Der Wert von x ist: %d. \n",x);
   }
      printf("Programm beendet \n");
      return 0;
}
```

```c
/* Beispiel fuer eine while-Schleife */
#include <stdio.h>
int main(void)
{
   int x=0;
   while(x<5)
   {
      printf("Der Wert von x ist %d. \n",x);
      x=x+1;
    }
   printf("Programm beendet \n");
   return 0;

}
```

do-while-Schleife

- allgemeine Form

```
do
   {Anweisung(en)}
while (Bedingung);
naechste-Anweisung;
```

- fußgesteuerte Schleife, da Bedingung erst am Ende abgefragt wird
- Schleife wird mindestens einmal durchlaufen
- wird auch als nichtabweisende Schleife bezeichnet
- wird relativ selten verwendet
- bis die Bedingung erfüllt ist, führe die Anweisung aus
- do-while-Schleife muss mit einem Semikolon abgeschlossen werden

do-while-Schleife (Beispiel)

```
/* 1. Beispiel für eine do-while Schleife */
#include <stdio.h>
int main(void)
{
     int count, ergebnis;
     count=1;
     ergebnis=0;
     do
    {
       ergebnis=ergebnis+count;
       printf("Das ergebnis ist: %d \n",ergebnis);
       count++;
      }
      while (count <=10);
      return 0;
}
```

do-while-Schleife (Beispiel)

```
/* 1. Beispiel für eine do-while Schleife */
#include <stdio.h>
int main(void)
{
    int count, ergebnis;
    count=1;
    ergebnis=0;
    do
   {
      ergebnis=ergebnis+count;
      printf("Das Ergebnis ist: %d \n",ergebnis);
      count++;
     }
     while (count <=10);
     printf("Programm beendet \n");
     return 0;
}
```

- Programmausgabe

  Das Ergebnis ist: 1
  Das Ergebnis ist: 3
  Das Ergebnis ist: 6
  Das Ergebnis ist: 10
  Das Ergebnis ist: 15
  Das Ergebnis ist: 21
  Das Ergebnis ist: 28
  Das Ergebnis ist: 36
  Das Ergebnis ist: 45
  Das Ergebnis ist: 55
  Programm beendet

do-while-Schleife (Beispiel)

```
/* 2. Beispiel für eine do-while Schleife */
#include <stdio.h>
int main(void)
{
    int summe=0;
    int eingabe;
    do
   {
      printf("Bitte Zahl zum Addieren eingeben (0 fuer Ende).\n");
      scanf("%d",&eingabe);
      summe=summe+eingabe;
     }
      while (eingabe !=0);
      printf("Die Summe ist %d.\n",summe);
     return 0;
}
```

do-while Schleife (Beispiel 2)

- Programm übersetzen:
  - gcc -o while2 dowhile2.c
- Programm ausführen
  - ./while2
- Programmausgabe:

```
Bitte Zahl zum Addieren eingeben (0 fuer Ende).
2
Bitte Zahl zum Addieren eingeben (0 fuer Ende).
4
Bitte Zahl zum Addieren eingeben (0 fuer Ende).
5
Bitte Zahl zum Addieren eingeben (0 fuer Ende).
0
Die Summe ist 11.
```

Schleifen abbrechen

- 2 Möglichkeiten Schleifen vorzeitig abzubrechen
  - **continue-Anweisung**
    - beendet nur den aktuellen Schleifendurchlauf
    - setzt Schleife fort, wenn Bedingung noch weiter erfüllt ist
  - **break-Anweisung**
    - beendet die gesamte Schleife
    - Programm wird nach der Schleife fortgesetzt

continue-Anweisung (Beispiel)

```
/* Schleifenabbruch mit continue */
#include <stdio.h>
int main(void)
{
   int i;
   for(i=-3; i<=3; i++)
   {
       if(i==0)
        continue;
       printf("Der Wert ist  %d\n",i);
    }
    printf("Program beendet\n");
    return 0;
}
```

continue-Anweisung (Beispiel)

```
/* Schleifenabbruch mit continue */
#include <stdio.h>
int main(void)
{
   int i;
   for(i=-3, i<=3, i++)
   {
      if(i==0)
       continue;
      printf("Der Wert ist:  %d\n",i);
    }
    printf("Program beendet\n");
    return 0;
}
```

- Programmausgabe:

```
Der Wert ist: -3
Der Wert ist: -2
Der Wert ist: -1
Der Wert ist: 1
Der Wert ist: 2
Der Wert ist: 3

Programm beendet
```

break-Anweisung (Beispiel)

```
/* Beispiel fuer eine break-Anweisung */
#include<stdio.h>
int main(void)
{
    int eingabe;
    int i=1;
    for(i=1;i<10;i++)
    {
      printf("Bitte geben Sie eine Integer-Zahl ein. (Abbruch mit 999)\n");
      scanf("%d",&eingabe);
      if(eingabe == 999) break;
      printf("Sie haben die Zahl %d eingegeben.\n",eingabe);
    }
    printf("Programm beendet.\n");
    return 0;
}
```

Kontrollstrukturen: Zusammenfassung

- beeinflussen den Programmablauf
- in C gibt es folgende Kontrollstrukturen
  - Auswahlanweisungen:
    - bedingte Anweisungen: if-else-Anweisung
    - Fallunterscheidungen: switch-Anweisung
  - Schleifen (Iterationsanweisungen):
    - kopfgesteuerte Schleifen: while-Schleife
    - fußgesteuerte Schleifen: do-while-Schleife
    - Zählschleife: for-Schleifen
- Schleifenabbruch:
  - continue-Anweisung: aktueller Schleifendurchlauf wird abgebrochen
  - break-Anweisung: gesamte Schleife wird beendet