

# **Vorschau**

**Einführung in die Programmierung**

**Michael Felderer**

**Institut für Informatik, Universität Innsbruck**

# Disclaimer

- Dieser Foliensatz ist eine Kurzfassung von Themen die in späteren Vorlesungen ausführlich behandelt werden
- Das Ziel dieser Vorlesungseinheit ist einen kurzen Überblick über die kommenden Themen zu geben und rasch die Fähigkeit zu erlangen etwas komplexeres Verhalten zu programmieren
- Verwenden Sie daher für die Klausurvorbereitung die Unterlagen der späteren Vorlesungseinheiten

# Das erste C-Programm (C99-Stil)

```
/* Das erste C-Programm */

#include <stdio.h>      // für Funktion printf
#include <stdlib.h>     // für EXIT_SUCCESS

/* Hauptprogramm */
int main(void) {
    printf("Hello World!\n");
    return EXIT_SUCCESS;
}
```

**Linux Kommandozeile (Programm hat den Namen test.c):**

```
[...]$ gcc -Wall -Werror -std=c11 test.c -o test
[...]$ ./test
Hello World!
```

# Integrierte Entwicklungsumgebung (IDE)

- IDE = *Integrated Development Environment*
- Kombiniert Texteditor, Code-Vervollständigung, Compiler, Kommandozeile, Debugger und weitere Hilfswerkzeuge in konfigurierbarer Form
- Beispiele
  - Eclipse CDT
  - JetBrains CLion
  - Qt Creator
  - Visual Studio Code
  - ...

- Bisher:
  - Programm gibt nur einen bestimmten Text aus
  - Text steht beim Kompilieren fest
- Ziel:
  - Probleme lösen, etwa
    - alle geraden Zahlen ausgeben
    - Fibonacci-Folge berechnen
    - ...
  - dazu müssen wir
    - Rechnen
    - Werte speichern
    - Je nach Ergebnis unterschiedlichen Code ausführen
    - Code mehrfach ausführen

# Variablen

- Variable
  - Ein Programm verarbeitet **Daten**, die in sogenannten **Variablen** abgelegt werden
  - Ein Programm legt die **Ergebnisse** wieder in solchen **Variablen** ab
  - Eine Variable ist eine **benannte Speicherstelle**
- Kennzeichen (für eine Variable)
  - **Name**
  - **Datentyp**
  - **Wert**
  - Adresse
  - Gültigkeitszeitraum
  - Sichtbarkeitsbereich

# Variablen (vereinfacht)

- Datentyp
  - Integer (Ganzzahl)
    - Wird für ganzzahlige Werte mit Vorzeichen verwendet
- Name (Bezeichner) einer Variable
  - Ist eine Folge aus Buchstaben, Ziffern und Unterstrich
  - Der Name muss aber mit Buchstabe oder Unterstrich beginnen
- Wert schreiben
  - Der Wert einer Variable kann sich ändern
  - Wert kann durch eine Zuweisung gesetzt/geändert werden
- Wert lesen
  - Kann z.B. in arithmetischen Ausdrücken verwendet werden
- Wert ausgeben

# Variablen (2)

- Deklaration/Definition

- „neue Variable anlegen“

- `int x;`

Datentyp      Bezeichner

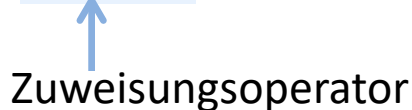


- Initialisierung/Zuweisung

- „zuweisen eines Wertes“

- `x = 1;`

Zuweisungsoperator



- Ausgabe des aktuellen Werts

- `printf("%d\n", x);`

Variable





# Variablen (3)

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

int main(void) {
    int a;
    int b;
    a = 1;
    b = a;

    printf("%d\n", a); // 1
    printf("%d\n", b); // 1

    // Zuweisung eines neuen Werts
    b = 2;

    printf("%d\n", a); // 1
    printf("%d\n", b); // 2
    return EXIT_SUCCESS;
}
```

# Operatoren (1)

- Arithmetische Operatoren
  - $*$ ,  $/$ ,  $+$ ,  $-$
  - Modulo Operator  $\%$ 
    - Rest bei der Division mit Rest
      - $11:2 = 5 \quad 1 \text{ Rest}$
      - $11 \% 2 = 1$
- Logische Operatoren
  - $a \ \&\& \ b$  = logisches Und
  - $a \ || \ b$  = logisches Oder
  - $!a$  = logische Negation

# Operatoren (2)

- Vergleichsoperatoren

|                        |  |
|------------------------|--|
| <code>a == b</code>    | ist der Wert von a <b>gleich</b> dem Wert von b                  |
| <code>a != b</code>    | ist der Wert von a <b>ungleich</b> dem Wert von b                |
| <code>a &gt; b</code>  | ist der Wert von a <b>größer als</b> der Wert von b              |
| <code>a &lt; b</code>  | ist der Wert von a <b>kleiner als</b> der Wert von b             |
| <code>a &gt;= b</code> | ist der Wert von a <b>größer oder gleich</b> wie der Wert von b  |
| <code>a &lt;= b</code> | ist der Wert von a <b>kleiner oder gleich</b> wie der Wert von b |

- Ergebnis ist entweder WAHR oder FALSCH
  - ANSI-C (C89) hat keinen Datentyp zur Darstellung boolescher Werte
  - Es wird der Wert 0 als **false** interpretiert und Werte ungleich 0 als **true**
  - Vergleichende Operatoren haben den Wert 0, wenn die entsprechende Aussage falsch ist, andernfalls haben sie den Wert 1

# Operatoren (3)

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

int main(void) {
    int a = 1;
    int b = 2;
    int c;

    c = a + 2;
    printf("%d\n", c); // 3

    printf("%d\n", b == 2); // 1 (WAHR)

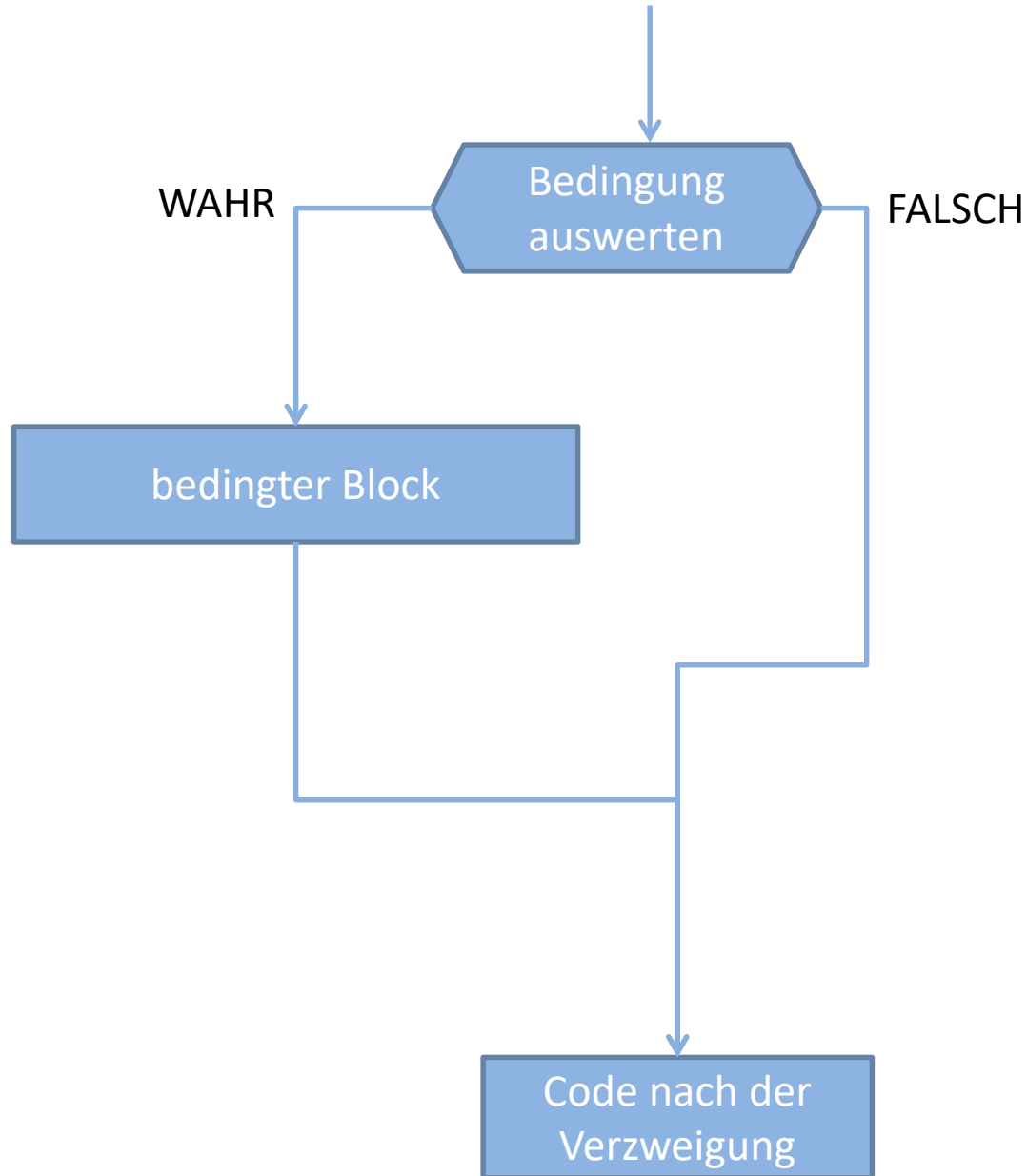
    c = (c + 1) * b;
    printf("%d\n", c); // 8

    return EXIT_SUCCESS;
}
```

# Verzweigung (1)

- Mit Verzweigungen kann man den Ablauf des Programms beeinflussen, in dem man logische Bedingungen definiert und damit entscheidet, an welcher Stelle das Programm fortgesetzt werden soll.
- `if( Bedingung ) {  
 // bedingter Block  
}`
- Alles zwischen { und } wird nur ausgeführt wenn die Bedingung WAHR ist
- Ist die Bedingung FALSCH, wird der Code nach } ausgeführt

# Verzweigung (2)



# Verzweigung (3)

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

int main(void) {
    int a = 1;

    if(a % 2 == 1) {
        printf("ungerade\n");
    }

    if(a % 2 == 0) {
        printf("gerade\n");
    }

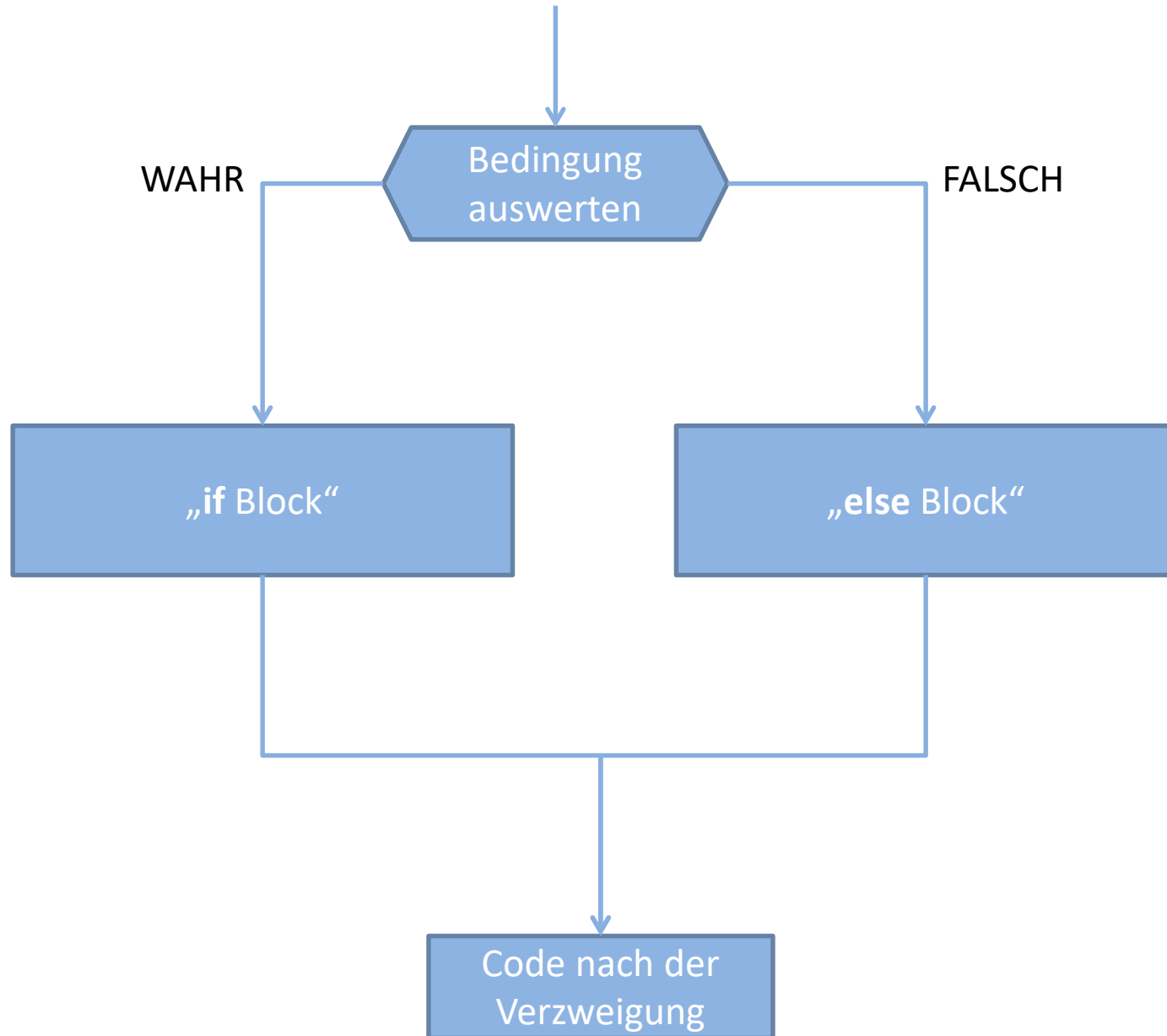
    printf("Ende\n");
    return EXIT_SUCCESS;
}
```

# Verzweigung - else (1)

- der **else**-Block wird immer ausgeführt wenn die Bedingung FALSCH ergibt
- **if**( Bedingung ) {  
    // Bedingung == WAHR  
}  
**else** {  
    // Bedingung == FALSCH  
}



# Verzweigung - else (2)



# Verzweigung - else (3)

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

int main(void) {
    int a = 1;

    if(a % 2 == 1) {
        printf("ungerade\n");
    }
    else {
        printf("gerade\n");
    }

    printf("Ende\n");
    return EXIT_SUCCESS;
}
```

# Schleifen (1)

- Problem
  - Ausgabe der Zahlen von 1 bis 5
  - Ausgabe der Zahlen von 1 bis 10
  - Ausgabe der Zahlen von 1 bis 1.000.000.000 ?
- Lösung: Schleifen
  - Mit Schleifen können bestimmte Anweisungen mehrfach ausgeführt werden

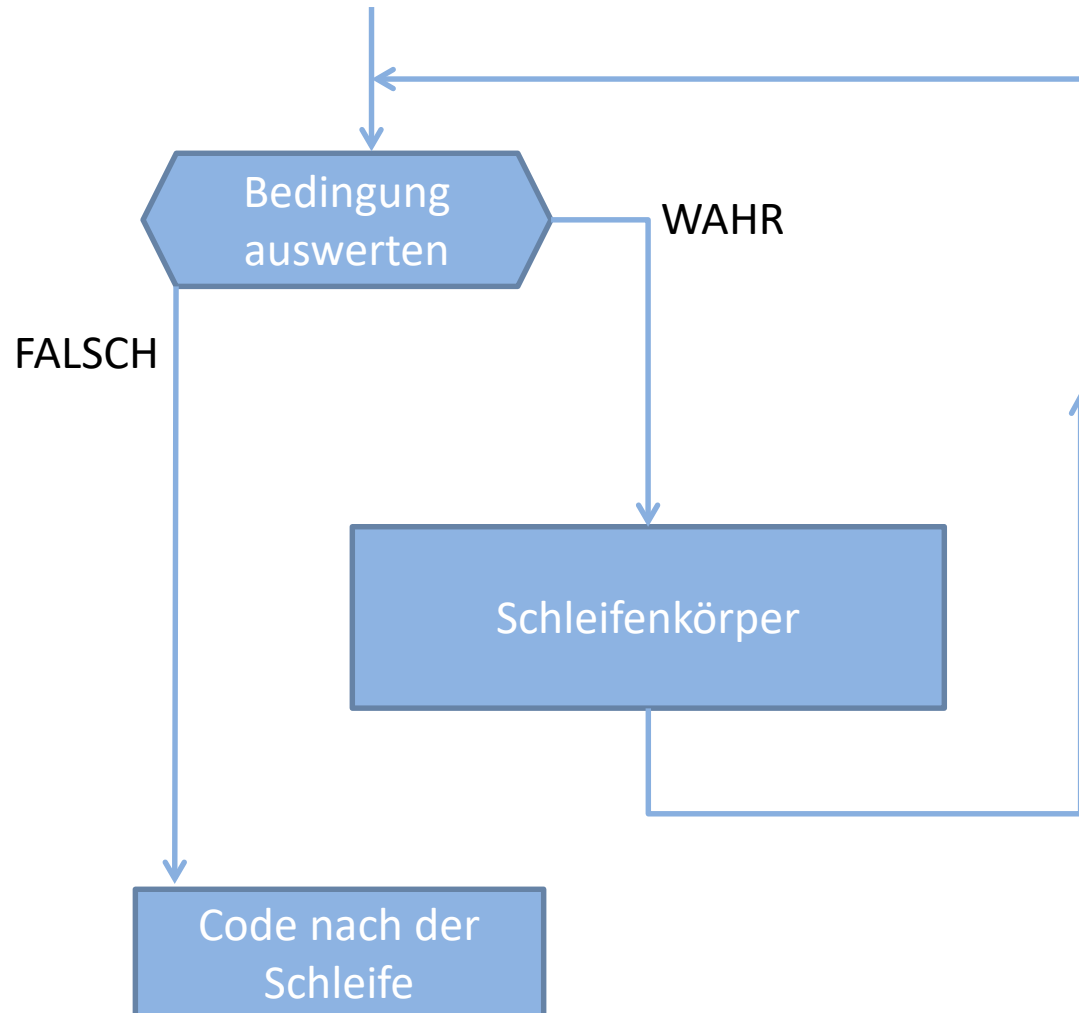
## Schleifen (2)

- Die while-Schleife führt einen Block solange aus wie die Bedingung WAHR ergibt

```
while( Bedingung ) {  
    // Schleifenkörper  
    // ...  
}
```

# Schleifen (3)

- Bedingung wird bei jedem Durchlauf überprüft



# Schleifen (4)

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

int main(void) {
    int i = 1;

    while( i < 100 ) {
        printf("%d\n", i);

        i = i + 1;
    }

    printf("Ende\n");
    return EXIT_SUCCESS;
}
```

# Debugging (interaktiv)

- Debugger
  - Werkzeug zur Programmdiagnose und zum Auffinden von Fehlerstellen
    - Steuerung des Programmablaufs, insbesondere durch Haltepunkte
    - Inspizieren von Daten
    - Modifizieren von Daten
- Beispiel: Debugging des folgenden fehlerhaften Programms zur Ausgabe der durch 3 teilbaren Zahlen kleiner 10

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

int main(void) {
    int i = 0;

    while( i < 9 ) {
        if( i % 3 = 0 ) {
            printf("%d\n", i);
        }
        i = i+1;
    }
    return EXIT_SUCCESS;
}
```

# Debugging (interaktiv) - Beispiel

- Interaktives Debugging in Visual Studio Code

