

# Einführung in die Programmierung

## **Grundbegriffe**

Prof. Dr. Peter Jüttner

# Inhalt

- **Grundbegriffe**
  - Grundstruktur eines C Programms
  - Variable
  - Konstante
  - Datentypen
  - Ausgabe

# Grundstruktur eines C-Programms

## Die main() Funktion

```
main()
{
    printf("Hochschule Deggendorf");
};
```

- ist Bestandteil jedes C Programms
- ist der Startpunkt (Hauptprogramm) und Endpunkt eines C-Programms, d.h. main wird immer als erstes ausgeführt bzw. beendet das Hauptprogramm.
- '{' und '}' schließen die auszuführenden Befehle des Hauptprogramms ein.
- noch kein komplettes Programm...

# Grundstruktur eines C-Programms

## Die main() Funktion

- jetzt komplett

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

```
void main(void)  
{
```

```
    printf("Hochschule Deggendorf\n");
```

```
    /* Gebe einen Text aus */
```

```
    system("PAUSE");
```

```
};
```

`#include <stdio.h>` macht Funktionen aus Datei `stdio.h` bekannt (u.a. `printf`)

Ausgabefunktion `printf` gibt Text auf Bildschirm aus

`\n` erzeugt einen Zeilenwechsel am Ende der Ausgabe

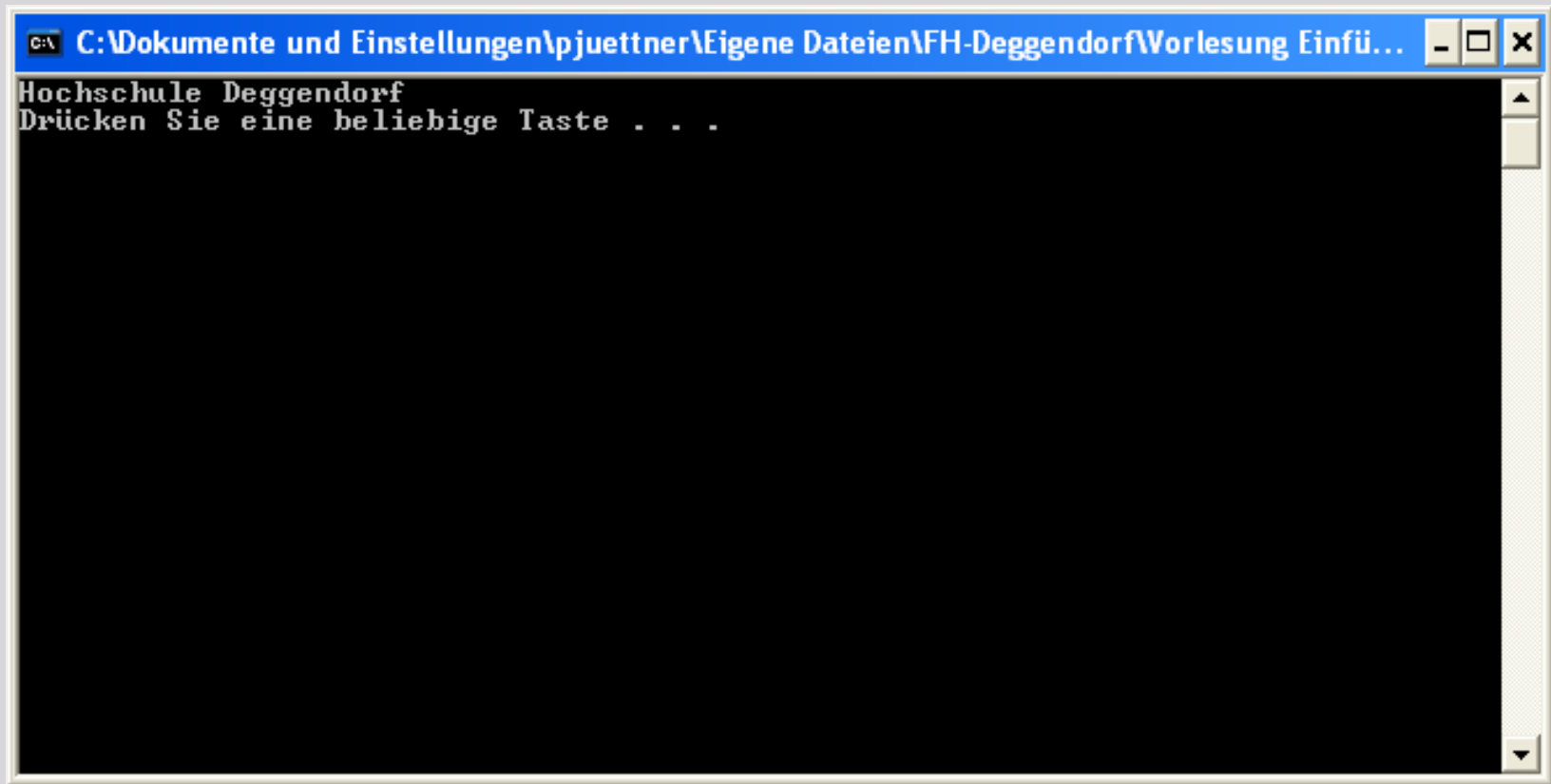
`/* ... */` kommentiert das Programm. Kommentar hat keinen Einfluß auf Ablauf

`system("PAUSE")` wartet auf Betätigung einer Taste

# Grundstruktur eines C-Programms

## Die main() Funktion

- Ergebnis der Ausführung



The screenshot shows a Windows command prompt window with a blue title bar. The title bar text is "C:\Dokumente und Einstellungen\pjuettnetner\Eigene Dateien\FH-Deggendorf\Vorlesung Einfü...". The window content is black with white text. The first line of text is "Hochschule Deggendorf". The second line of text is "Drücken Sie eine beliebige Taste . . .". The window has standard Windows window controls (minimize, maximize, close) in the top right corner.

```
C:\Dokumente und Einstellungen\pjuettnetner\Eigene Dateien\FH-Deggendorf\Vorlesung Einfü...  
Hochschule Deggendorf  
Drücken Sie eine beliebige Taste . . .
```

# Grundstruktur eines C-Programms

## Die main() Funktion

- ein bisschen Syntax...

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

#include Anweisung für  
Präprozessor

```
void main(void)
```

```
{
```

```
printf("Hochschule Deggendorf\n");
```

Anweisung eine Funktion  
auszuführen

```
/* Gebe einen Text aus */
```

Konstanter Text (Konstante),  
durch " " begrenzt

```
system("PAUSE");
```

durch /\* und \*/ werden  
Kommentare begrenzt

```
};
```

durch '{' und '}' werden zusammen-  
gehörende Anweisungen geklammert

durch ';' werden Anweisungen  
beendet

# Grundstruktur eines C-Programms

## Sprachelemente (Syntaktische Elemente)

- Anweisungen definieren in Einzelschritten, was das Programm zu tun hat (Daumenregel: eine Anweisung pro Zeile!)
- Anweisungen werden durch einen Strichpunkt (Semikolon) abgeschlossen
- Zusammengehörende Anweisungen (im Beispiel die des Hauptprogramms main) werden durch { ... } zusammengefasst
- Textkonstante werden durch “...” begrenzt

# Grundstruktur eines C-Programms

## Sprachelemente (Syntaktische Elemente)

- Kommentare werden durch `/* ... */` begrenzt. Kommentare dienen dazu das Programm zu erklären. Kommentare haben keine Wirkung auf den Ablauf des Programms  
(alternative C++-Form `// ... Kommentar bis Zeilenende`)
- Mittels `printf(...)` wird eine Ausgabe auf dem Bildschirm erzeugt
- Mittels `system("PAUSE")` wird auf Betätigung einer Taste gewartet (notwendig bei manchen Compilern, sonst wird Programm sofort beendet)



# Grundstruktur eines C-Programms

## Sonstiges...

- Präprozessoranweisungen werden durch '#' eingeleitet. Durch `#include <Datei>` wird Datei textuell an Stelle der `#include` Anweisung einkopiert. (s. Kap. Präprozessor). Präprozessoranweisungen gehören im engeren Sinn nicht zur Sprache C.  
`#include` wird verwendet u.a. um Standardbibliotheken einzubinden.
- Standardbibliotheken enthalten nützliche Funktionen wie z.B. Ein-/Ausgabe, Verarbeitung von Zeichenketten, Zufallszahlengenerator. Standardbibliotheken sind zum großen Teil vereinheitlicht, gehören aber ebenfalls im engeren Sinn nicht zur Sprache C

## Grundstruktur eines C-Programms

**Zum Schluss dieses Abschnitts ...**

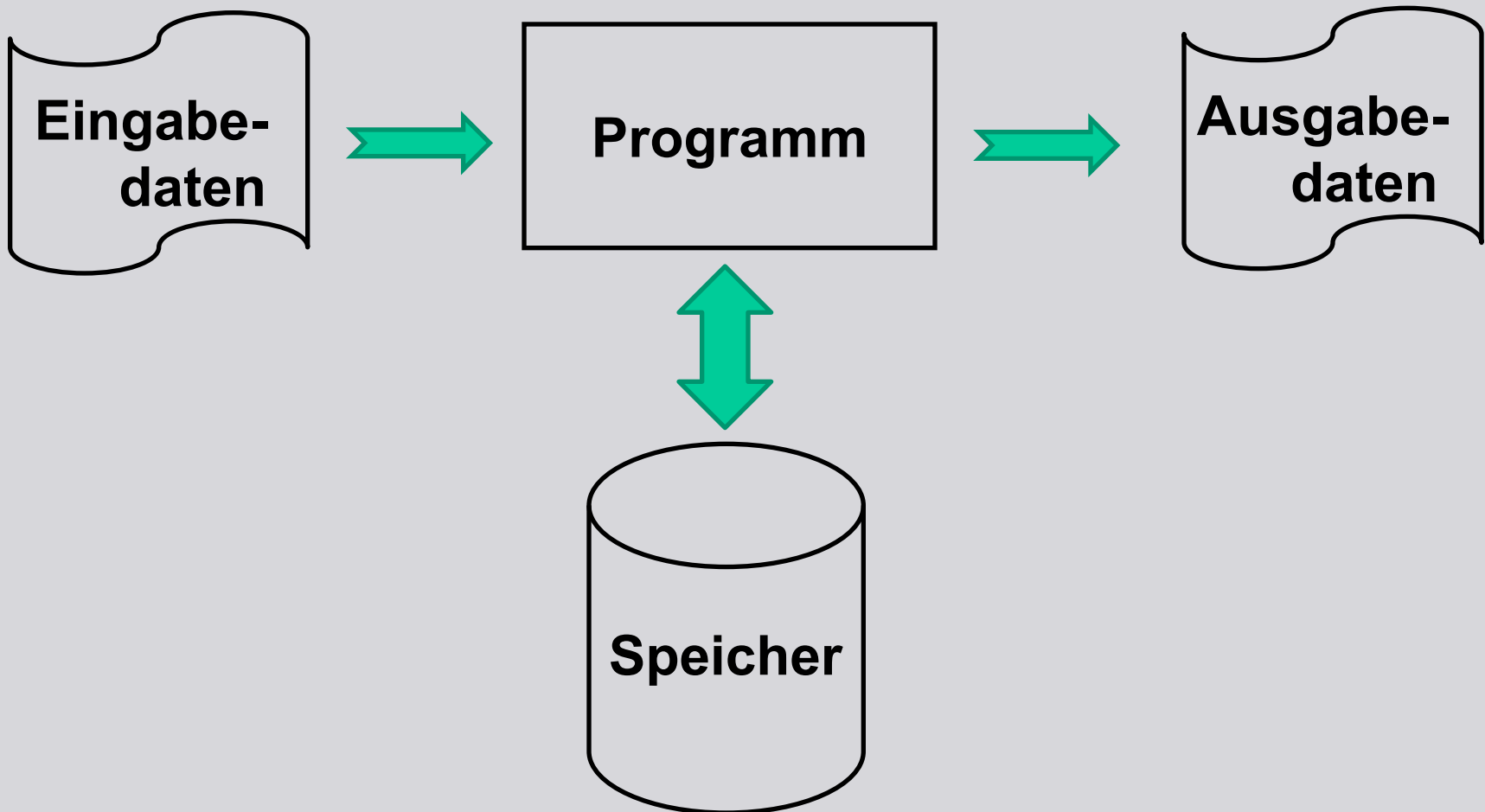
**Noch Fragen ??**

# Inhalt

- **Grundbegriffe**
  - Grundstruktur eines C Programms
  - **Variable**
  - Konstante
  - Datentypen
  - Ausgabe

# Variable

## Variable



# Variable

## Variable

- Ein Programm verarbeitet in der Regel Eingabedaten zu Ausgabedaten.
- Bei der Verarbeitung werden Daten (Ein-/Ausgabe und Zwischenergebnisse) im Speicher abgelegt.
- Der Speicherbereich, der ein Datum (oder auch mehrere Daten) enthält, wird über einen definierten Namen angesprochen und als **Variable** bezeichnet.
- Der Name einer Variablen ist im Programm (mit Einschränkung) frei wählbar

# Variable

## Variable

Namensregeln für Variable (und auch andere frei wählbare Namen)

- Der Name darf Buchstaben, Ziffern und \_ (Unterstrich) enthalten.
- Das erste Zeichen muss ein Buchstabe sein.
- Groß- und Kleinschreibung wird unterschieden.
- Vordefinierte C-Schlüsselwörter dürfen nicht als Variablennamen verwendet werden.
- Leerzeichen, Bindestriche, Punkte und andere Sonderzeichen sind nicht erlaubt.

# Variable

## Variable

Namensempfehlungen für Variable (und auch andere frei wählbare Namen)

- selbsterklärend (z.B. mitarbeiter, ergebnis, inhalt, element)
- nicht zu kurz (z.B. i, j, a, b, c)
- nicht zu lang (z.B. dieservariablennameistwirklichnichtmehreinfachzulesen)
- Sprache beachten (z.B. englische Namen in internationalen „Umgebungen“)

# Variable

## Variable

vordefinierte Namen in C (Schlüsselwörter / Keywords\*)

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	volatile	while

\*) Schlüsselwörter werden benötigt, um ein Programm für den Compiler verständlich zu machen



# Variable

## Variable

- Variable besitzen einen Typ, der angibt, wie viel Speicher eine Variable dieses Typs benötigt.
- Der Typ einer Variablen legt zusätzlich fest, welche Operationen mit einer Variablen durchgeführt werden dürfen.
- Einige Typen sind in C vordefiniert, weitere Typen können im Programm definiert werden.

# Variable

## Variablendeklaration

- Beispiele einer Variablendeklaration

`int zahl1;`

Deklaration einer Variablen  
namens Zahl1 vom Typ int

`int zahl2 = 10;`

Deklaration einer Variablen  
namens Zahl2 vom Typ int,  
Initialisierung mit dem Wert 10

`int zahl 3 = zahl2;`

Deklaration einer Variablen  
namens Zahl3 vom Typ int,  
Initialisierung mit dem Wert der  
Variablen zahl 2

`int zahl 4 = zahl2 + zahl 3;`

Deklaration einer Variablen namens Zahl4 vom  
Typ int, Initialisierung mit der Summe der Werte  
von Zahl2 und Zahl 3.

# Variable

## Allgemeines ...

- Variable dürfen in C fast überall deklariert werden, z.B. am Anfang des Programms, nach einer öffnenden geschweiften Klammer ({)
- Deklaration legt Variable im Speicher an und macht sie im Programm bekannt
- Nach der Deklaration ist eine Variable innerhalb des Programms bekannt
- Variable sollten / müssen immer initialisiert werden ( = ... ;)

# Variable

## Allgemeines ...

- Variable können (wie der Name schon sagt) während des Ablaufs des Programms durch eine Zuweisung (=) ihren Wert ändern.
- Auf Variable  $v$  kann im Programm beliebig oft lesend ( $\dots = v$ ) oder schreibend ( $v = \dots$ ) zugegriffen werden.
- Der schreibende Zugriff erfolgt durch eine **Zuweisung** der Form  
Variable = Ausdruck
- Der lesende Zugriff erfolgt in einem Ausdruck, z.B.  $\text{zahl1} = \text{zahl2} + 10$ ;

# Variable

## Allgemeines ...

- Variable dürfen in Ausdrücken miteinander verknüpft werden, z.B.  
`zahl1 = (zahl2 + zahl3) * zahl4;`

# Variable

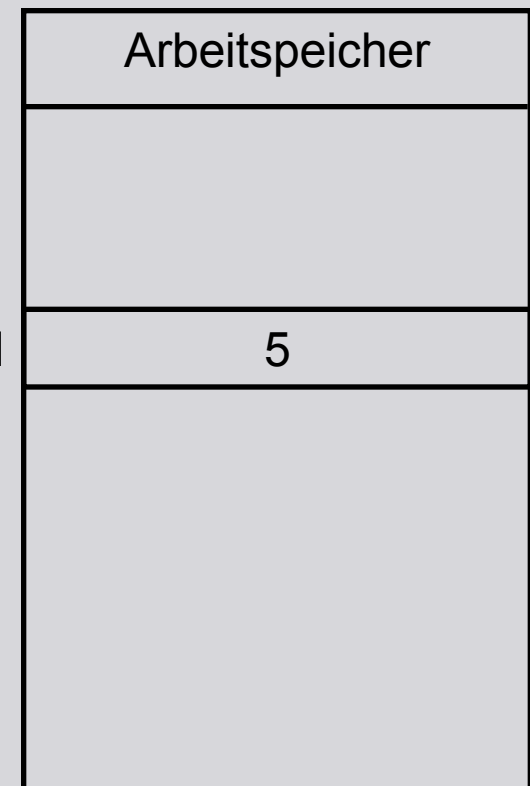
## Variable im Speicher

- Realisierung im Speicher (was macht der Compiler aus einer Variablen

- `int zahl = 5;`

Speicher an der Adresse xyz wird unter dem Namen zahl reserviert, Größe des reservierten Bereichs entspricht dem Typ int. Der Speicher wird mit dem Wert 5 beschrieben

Adr. xyz / zahl



# Variable

## Variable im Programm

- Realisierung im Speicher

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

```
int zahl1 = 5;
```

```
int zahl2;
```

```
void main(void)  
{
```

```
    zahl1 = 10;
```

```
    zahl2 = zahl1 + 5;
```

```
    printf("%d", zahl2);  
};
```

Deklaration einer Variablen namens Zahl1 vom Typ int mit Initialisierung (Wert 5)

Deklaration einer Variablen namens Zahl1 vom Typ int ohne Initialisierung

Schreibender Zugriff auf Variable zahl1, neuer Wert 10

Lesender Zugriff auf Variable zahl1, Wert ist 10

Ausgabe des Werts von zahl2 auf Bildschirm

Schreibender Zugriff auf Variable zahl2, Wert ist Ergebnis des Ausdrucks zahl1 + 5

# Variable

## Variable im Programm

- Realisierung im Speicher

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

```
int zahl1 = 5;
```

```
int zahl2;
```

```
void main(void)
```

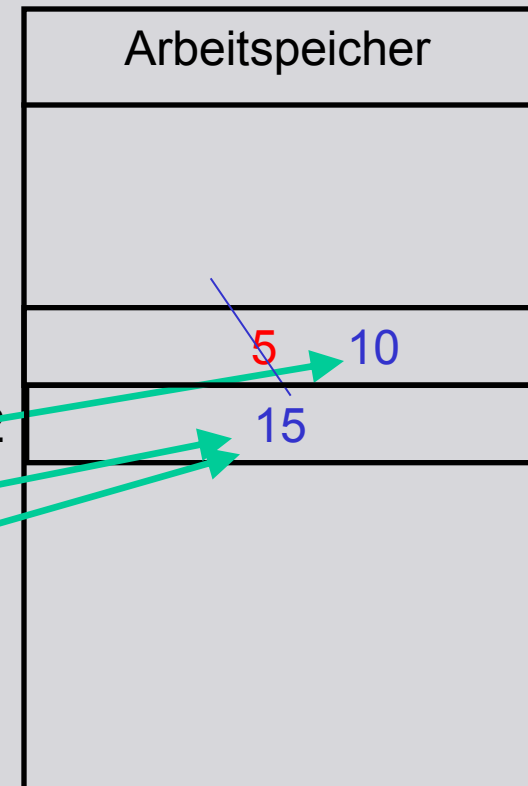
```
{
```

```
    zahl1 = 10;
```

```
    zahl2 = zahl1 + 5;
```

```
    printf("%d", zahl2);
```

```
};
```



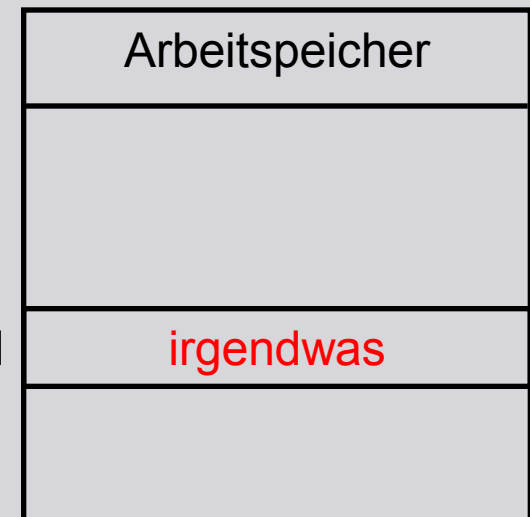
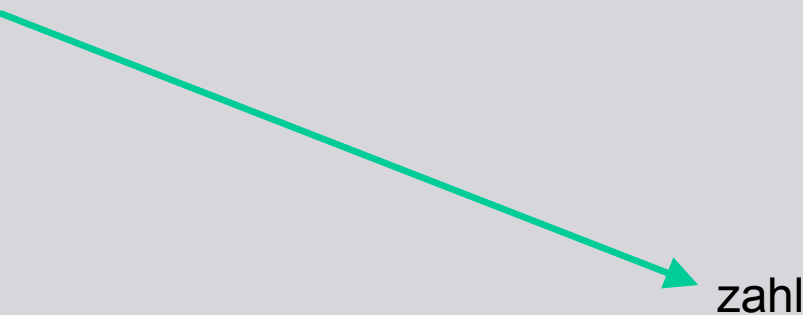


# Variable

## Variable im Programm

- **Achtung:** Variable, die nicht initialisiert werden haben auch einen „Wert“, nämlich den, der gerade im Speicher an der Stelle, wo sich die Variable befindet, steht! Dieser Wert ist in der Regel nicht vorhersehbar!

```
int zahl;
```



## Variable

**Zum Schluss dieses Abschnitts ...**

**Noch Fragen ??**

# Inhalt

- **Grundbegriffe**
  - Grundstruktur eines C Programms
  - Variable
  - **Konstante**
  - Datentypen
  - Ausgabe

# Konstante

## Literalkonstante haben wir z.T. schon kennengelernt...

- als Angabe eines konkreten Werts
- “Hochschule Deggendorf” als konstante Zeichenkette (konstanter String)
- 5 als Initialisierungswert einer Variablen
- haben auch immer einen (impliziten) Typ

# Konstante

## Literalkonstante in unterschiedlichen Formaten

- dezimale Zahl, z.B. 1, 3, 5, 100, -100
- hexadezimale Zahl, definiert durch Präfix **0x**, z.B. 0x0, 0x10, 0x100
- oktale Zahl, eingeleitet durch 0, z.B. 010
- Gleitkomma, z.B. 10.5, 3.14

# Konstante

## Symbolische Konstante

- definiert durch die Präprozessoranweisung **#define**
- Format: **#define KONSTANTENNAME Ausdruck**
- danach wird im Quellcode überall da, wo **KONSTANTENNAME** steht, **Ausdruck** textuell einkopiert (danach erst wird der Quellcode übersetzt)

# Konstante

## Symbolische Konstante

- Beispiel

```
#define PI 3.14
```

ab hier wird das Symbol  
PI durch 3.14 ersetzt

```
float radius = 5;
```

```
float kreisumfang = 2*radius*PI;
```

```
float kreisflaeche = radius*radius*PI;
```

wird textuell umgesetzt in  
 $2 * \text{radius} * 3.14$

wird textuell umgesetzt in  
 $\text{radius} * \text{radius} * 3.14$

# Konstante

## Symbolische Konstante

```
#include <stdio.h>

int main(void)
{
    #define PI 3.14

    float radius = 5;
    float kreisumfang = 2*radius*PI;
    float kreisflaeche = radius*radius*PI;

    printf("Kreisumfang bei Radius 5 : %f.2\n", kreisumfang);
    printf("Kreisflaeche bei Radius 5 : %f.2\n", kreisflaeche);

};
```



# Konstante

## konstante „Variable“

- Variable können durch das Schlüsselwort **const** zu Konstanten definiert werden
- `const` Variable müssen initialisiert werden (`const int cv = ... ;`)
- `const` Variable dürfen nicht überschrieben werden (`cv = ... ;` führt zu Übersetzungsfehler)
- Im Gegensatz zu Symbolischen Konstanten werden konstante Variable im Speicher abgelegt.

## Konstante

**Zum Schluss dieses Abschnitts ...**

**Noch Fragen ??**

# Inhalt

- **Grundbegriffe**
  - Grundstruktur eines C Programms
  - Variable
  - Konstante
  - **Datentypen**
  - Ausgabe

# Datentypen

## oder kurz Typ einer Variablen definiert ...

- die Größe der Variablen im Speicher (Anzahl der Bytes)
- die logische Struktur einer Variablen im Speicher
- die Operationen, die mit einer Variablen oder Konstanten eines bestimmten Typs möglich sind

# Datentypen

## vordefinierte Datentypen in C

Datentyp	Schlüsselwort in C	Größe in Bytes	Wertebereich
Zeichen (ganze Zahl, sehr kurz)	char	1	-128 bis 127
ganze Zahl (kurz)	short (short int)	2	-32768 bis 32767
ganze Zahl	int	4 (meist)	-2147483648 bis 2147483647
ganze Zahl (lang)	long (long int)	4	-2147483648 bis 2147483647
Zeichen (ganze Zahl, sehr kurz) ohne Vorzeichen	unsigned char	1	0 bis 255
ganze Zahl (kurz) ohne Vorzeichen	unsigned short (unsigned short int)	2	0 bis 65535
ganze Zahl ohne Vorzeichen	unsigned int	4 (meist)	0 bis 4294967295
ganze Zahl (lang) ohne Vorzeichen	unsigned long (unsigned long int)	4	0 bis 4294967295
Gleitkommazahl (einfache Genauigkeit)	float	4	Genauigkeit 7 Dezimalen
Gleitkommazahl (doppelte Genauigkeit)	double	8	Genauigkeit 19 Dezimalen

# Datentypen

## Standardisierung

- legt Größe einiger Datentypen nicht fest!
- z.B. ANSI Standard
  - Die Größe eines char beträgt ein Byte
  - Die Größe eines short ist kleiner oder gleich der Größe eines int.
  - Die Größe eines int ist kleiner oder gleich der Größe eines long.
  - Die Größe eines unsigned int ist gleich der Größe eines int.
  - Die Größe eines float ist kleiner oder gleich der Größe eines double.

# Datentypen

## Standardisierung

- ➔ genaue Größe prüfen (Compilermanual)
- ➔ für Portierbarkeit Datentypen umdefinieren (s. Kap. XXXX)

# Datentypen

## Verwendung von Typen

- Datentyp von Variablen

```
int ganze_Zahl;  
float gleitkommazahl;  
double lange_gleitkommazahl;  
char zeichen;
```

- Datentyp von Konstanten

```
ganze_Zahl = 5;  
gleitkommazahl = 3.;  
lange_gleitkommazahl = 7.5  
zeichen = 'x';
```

Ganzzahlkonstante

Gleitkommakon-  
stante, ausgedrückt  
durch Punkt ('.')

Zeichenkonstante, ausgedrückt durch  
Hochkommata (' ')



# Datentypen

## Verwendung von Typen

- Speicherlayout

```
int ganze_Zahl;
```

```
float gleitkommazahl;
```

```
double lange_gleitkommazahl;
```

```
char zeichen;
```

ganze\_Zahl

gleitkommazahl

lange gleitkommazahl

zeichen

Arbeitspeicher	
ganze_Zahl	4 Bytes
gleitkommazahl	4 Bytes
lange gleitkommazahl	8 Bytes
zeichen	1 Byte

# Datentypen

## Typkompatibilität

- Der Typ char hat eine „Doppelrolle“. Zum einen ist char ein Zeichen ('a', 'B', '!', '/' usw.), zum anderen ist char ein Zahlentyp
- in C sind die Standardtypen zueinander **kompatibel**, d.h. Variablen und Konstanten verschiedener Typen dürfen in Ausdrücken, z.B. in Zuweisungen und mathematischen Operationen miteinander verknüpft werden.  
**Achtung! Solche Verknüpfungen können unter Umständen unerwartete Nebeneffekte haben!**

# Datentypen

## Typkompatibilität (vorzeichenbehaftete Zahlen)

- Beispiele  
`int g_zahl;`  
`char zeichen;`  
`float gk_zahl;`

`g_zahl = 3.5;`

Zuweisung einer Gleitkommazahl an eine Ganzzahl, Nachkommateil wird abgeschnitten, `g_zahl` hat den Wert 3

`zeichen = 5.4;`

Zuweisung einer Gleitkommazahl an eine kurze Ganzzahl, Nachkommateil wird abgeschnitten, `zeichen` hat den Wert 5

`gk_zahl = 5;`

Zuweisung einer Ganzzahl an eine Gleitkommazahl, `gk_zahl` hat den Wert 5.0

`g_zahl = 4.5/2;`

Zuweisung einer Gleitkommazahl (4.5/2) an eine Ganzzahl, `g_zahl` hat den Wert 2

# Datentypen

## Typkompatibilität (vorzeichenbehaftete Zahlen)

```
#include <stdio.h>

int main(void)
{ int g_zahl;
  char zeichen;
  float gk_zahl;

  g_zahl = 3.5;
  printf("Wert von g_zahl nach g_zahl = 3.5;    : %d\n", g_zahl);




  zeichen = 5.4;
  printf("Wert von zeichen nach zeichen = 5.4;   : %d\n", zeichen);

  gk_zahl = 5;
  printf("Wert von gk_zahl nach gk_zahl = 5;     : %f\n", gk_zahl);

  g_zahl = 4.5/2;
  printf("Wert von g_zahl nach g_zahl = 4.5/2;    : %d\n", g_zahl);
};
```

# Datentypen

## Typkompatibilität (vorzeichenbehaftete Zahlen)

- `gk_zahl = 5/2;`  Zuweisung einer Ganzzahl an eine Gleitkommazahl, `gk_zahl` hat den Wert 2.0  
(5/2 ist eine Ganzzahldivision!)
- `gk_zahl = 5.0 / 2;`  Zuweisung einer Gleitkommazahl an eine Gleitkommazahl, `gk_zahl` hat den Wert 2.5  
(5.0 / 2 ist eine Gleitkommadivision!)
- `zeichen = 1000;`  Zuweisung einer „zu großen“ Ganzzahl an eine kurze Ganzzahl, `zeichen` hat den Wert ?

# Datentypen

## Typkompatibilität (vorzeichenbehaftete Zahlen)

```
#include <stdio.h>

int main(void)
{

    char zeichen;
    float gk_zahl;

    gk_zahl = 5/2;
    printf("Wert von gk_zahl nach gk_zahl = 5/2; : %f\n", gk_zahl);

    gk_zahl = 5.0 / 2;
    printf("Wert von gk_zahl nach gk_zahl = 5.0/2; : %f\n", gk_zahl);

    zeichen = 1000;
    printf("Wert von zeichen nach zeichen = 1000; : %d\n", zeichen);

};
```

# Datentypen

## Typkompatibilität (vorzeichenbehaftete Zahlen)

- `g_zahl = 1000;`  
`zeichen = g_zahl;`

Zuweisung einer int Ganzzahl mit Wert 1000 an ein char, zeichen hat den Wert -24  
(1000 „passt“ nicht in 1 Byte, 3 Bytes werden „vorne“ abgeschnitten)

`g_zahl = -129;`  
`zeichen = g_zahl;`

Zuweisung einer int Ganzzahl mit Wert -129 an ein char, zeichen hat den Wert 127  
(-129 „passt“ nicht in 1 Byte, 3 Bytes werden „vorne“ abgeschnitten)

# Datentypen

## Typkompatibilität (vorzeichenbehaftete Zahlen)

```
#include <stdio.h>

int main(void)
{

    char zeichen;
    float gk_zahl;

    g_zahl = 1000;
    zeichen = g_zahl;

    printf("Wert von zeichen nach zeichen = g_zahl mit Wert 1000 : %d\n", zeichen);

    g_zahl = -129;
    zeichen = g_zahl;
    printf("Wert von zeichen nach zeichen = g_zahl mit Wert -128 : %d\n", zeichen);

};
```



# Datentypen

## Typkompatibilität (vorzeichenlose Zahlen)

- Beispiele

```
unsigned short u_g_zahl;  
short g_zahl;  
unsigned char u_zeichen;  
char zeichen;
```

```
u_g_zahl = -100;
```

```
g_zahl = -1000;  
u_zeichen = g_zahl;
```

```
zeichen = 'x';  
u_zeichen = zeichen;
```

Zuweisung einer negativen Zahl an eine Variable mit vorzeichenlosem Typ  
(negative Zahl wird vorzeichenlos interpretiert)

Zuweisung einer short Ganzzahl mit Wert -1000 an ein unsigned char, u\_zeichen hat den Wert  
(-1000 „passt“ nicht in 1 Byte, 1 Byte wird „vorne“ abgeschnitten)

Zuweisung problemlos, da Buchstaben vorzeichenlos

# Datentypen

## Typkompatibilität (vorzeichenbehaftete Zahlen)

```
#include <stdio.h>

int main(void)
{
    u_g_zahl = -100;
    printf("Wert von u_g_zahl nach u_g_zahl = -100;    : %d\n", u_g_zahl);

    g_zahl = -1000;
    u_zeichen = g_zahl;
    printf("Wert von u_zeichen nach zeichen = g_zahl    : %d\n", u_zeichen);
};
```

# Datentypen

## Typkompatibilität (Zusammenfassung)

- Bei Zuweisungen von Konstanten und Variablen an Variable anderer Typen wird vom Compiler eine implizite Typumwandlung an den Zieltyp durchgeführt (Impliziter Cast). Diese Typumwandlungen passen den zugewiesenen Wert an den Typ der Zielvariable an. Dabei kann es zu Datenverlust/Datenänderung kommen bei
  - Zuweisungen von Gleitkommagrößen an Ganzzahlige Variable („Abschneiden des Nachkommateils“)
  - Zuweisungen von „großen“ Werten an „kleine“ Variable (Abschneiden der höherwertigen Bytes)
  - Zuweisungen von vorzeichenbehafteten Größen an vorzeichenlose Variable (Verlust des Vorzeichens, Änderung des Werts)