Datentypen

Datentypen von Speicherstellen

- Rückblick auf den grundsätzlichen Aufbau eines C-Programmes:
  - Jeder Block kann mit einer Menge an Deklarationen zu benutzender Speicherstellen beginnen
- Deklarationen beinhalten
  - Den Datentyp, den die Speicherstelle annehmen soll (dies beinhaltet auch die Größe in Bytes)
  - Den Namen, unter der die Speicherstelle im Programm angesprochen werden soll
  - Ggf. einen Initialwert
- Beispiel:
  - Datentyp meinName = Wert;
- Derartige Speicherstellen heißen **Variablen**

Elementare Datentypen in C

```c
char c;
/* Ein einzelnes Zeichen mit oder ohne
Vorzeichen je nach Implementierung */
int i;
/* Ganzzahlen mit Vorzeichen */
float x;
 /* Gleitpunktzahl mit 7 Stellen */
double y;
 /* Gleitpunktzahl mit 15 Stellen */
```

Bedeutung der Datentypen

- Der Datentyp **int** dient für ganze Zahlen in der Länge eines Maschinenwortes der CPU.
- **char** soll ein Zeichen aufnehmen können, sollte also mindestens ein Byte groß sein.
- **float** bezeichnet Gleitkommazahlen in der Länge eines Maschinenwortes und
- **double** Gleitkommazahlen gleicher oder höherer Genauigkeit (abhängig vom Compiler)

Modifikation der Wortbreite

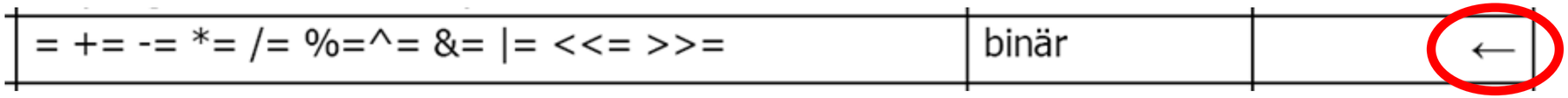
- Man kann versuchen, die Größe mit Modifizierern **short** und **long** zu verändern.
- Als Modifizierer von **int** dürfen sie auch alleine stehen.
- Der Standard schreibt aber lediglich die Gültigkeit folgender Aussage vor:
  - sizeof(short) <= sizeof(int) <= sizeof(long)
  - Dabei liefert sizeof(…) die Größe des angegebenen Datentyps in Bytes
- => Abhängig vom Compiler

Modifikation des Vorzeichens

- Die Modifizierer **unsigned** und **signed** kennzeichnen, ob eine Zahl ohne bzw. mit Vorzeichen interpretiert werden soll
- Beispiel:
  - unsigned int myInt; <- nur positive Werte erlaubt
  - signed int myInt; <- auch negative Werte erlaubt
- unsigned schaltet die Zweierkomplement-Interpretation aus, d.h. der Wertebereich der positiven Zahlen steigt im Vergleich zum signed Datentyp

Wertzuweisungen

- Schreibweise:
  - Variable = Ausdruck;
- Funktion:
  - Der Wert des Ausdrucks (rechte Seite) wird zuerst berechnet und in den Speicherplatz (linke Seite) geschrieben.



- Der Typ des Ausdrucks muss mit dem Typ der Variablen verträglich sein:
  - `int i;`
  - `i = 99; /* richtig */`
  - `i = 3.7; /* falsch */`
  - `i = 2 + 3 * 4; /* richtig */`

Ausdrücke

- Für Gleitkommazahlen und ganze Zahlen gibt es die Rechenarten + -* /
- Für ganze Zahlen gibt es zusätzlich die Rechenart % (Modulo = Rest einer ganzzahligen Division)
- Für ganze Zahlen ist Vorsicht bei der Division geboten: Was soll 3 / 2 als ganze Zahl liefern?
  - 7 / 2 liefert 3, 9 / 4 liefert 2
  - 7 % 2 liefert 1
- Bei ganzen Zahlen gilt stets:
  - i = (i / j)* j + (i % j)
  - Dividend = Quotient * Divisor + Rest

Ablauf von i = i + 1

- Anweisungen:
  - int i = 99;
  - i = i + 1;
- Die Wertzuweisung = ist nicht als Gleichheit zu sehen, sondern als Zuweisungsoperator:
  - i <- i + 1 (Speichere +(i,1) in i)
- Ablauf:
  - Der Wert von i wird gelesen. Ergebnis: 99
  - Dieser Wert wird erhöht.
  - Dieser neue Wert wird zurück geschrieben
  - i hat nun den Wert 100

Bitweise Operationen

- Ganze Zahlen (char, short, int bzw. long) sind Bitketten der Länge 8, 16, 32 bzw. 64 (je nach Plattform und Compiler!)
- Solche Bitketten können bitweise verarbeitet werden:
  - & Bitweises AND (0xff00 & 0xf0f0 liefert 0xf000)
  - | Bitweises OR (0xff00 |0xf0f0 liefert 0xfff0)
  - ^ Bitweises exklusives OR (0xff00 ^ 0xf0f0 liefert 0x0ff0)
  - ~ Bitweise Negation (~0xff00 liefert 0x00ff)

Bitweise Operationen

- Ganze Zahlen (char, short, int bzw. long) sind Bitketten der Länge 8, 16, 32 bzw. 64 (je nach Plattform und Compiler!)
- Bitketten können geschoben (shift) werden. Dies erfolgt im Sinne dualer Zahlen (binäre Werte)
  - Die Variable a wird um b Stellen nach links geschoben a << b
    - 1 << 2 → 4
  - Die Variable a wird um b Stellen nach rechts geschoben a >> b
    - 8 >> 2 → 2
  - Vorsicht bei negativen Zahlen!

Umwandlung der Datentypen

- In C können Datentypen ineinander umgewandelt werden, Beispiel:
  - `int Mehrwertsteuer = 10;`
  - `float Nettopreis = 4.00;`
  - `float Bruttopreis;`
  - `Bruttopreis = Nettopreis * (1 + Mehrwertsteuer / 100);`
- Was ist das Ergebnis?
- Richtig wäre:
  - `Bruttopreis = Nettopreis * (1 + (float)Mehrwertsteuer / 100);`
- Warum?

Umwandlung der Datentypen

- C kennt zwei Arten der Typumwandlung:
- 1.Explizite Typumwandlung: Der Programmierer gibt dies an
  - **(Datentyp) Ausdruck**
  - Semantik: Wandle den Wert des Ausdruckes in einen Wert vom Typ Datentyp

Umwandlung der Datentypen

- C kennt zwei Arten der Typumwandlung:
- 2.Implizite Typumwandlung
  - Für alle Operanden einer Operation wird der Datentyp untersucht
  - Alle Operanden werden in den reichhaltigsten Typen konvertiert
  - Beispiel: (float)a / 10 -> (float)a ist float, 10 ist int
  - -> (float)a / (float)10
- VORSICHT bei impliziter Typumwandlung: Besser explizite benutzen, da es nicht Compilerabhängig ist

Ausgabe mit printf

- Neben Texten kann auch der Inhalt von Variablen (formatiert) ausgegeben werden
  - int x;
  - printf(„Die Variable X hat den Wert %d\n“,x);
- %d wird Format-String genannt und kennzeichnet, wie die Daten auszugeben sind (welcher Datentyp). Es gibt:
  - %d oder %i für ganze Zahlen, Ausgabe als Dezimalzahl (int, short, long)
  - %x für ganze Zahlen, Ausgabe als Hexadezimalzahl (int, short, long)
  - %c für Zeichen (char)
  - %f für Gleitkommazahlen (float, double)
  - %s für Zeichenketten
- Für jede auszugebende Variable ist ein Format-String anzugeben
- Jede auszugebende Variable ist als Parameter im Anschluss an die Ausgabe-Zeichenkette zu übergeben. Achtung: Reihenfolge!

Ausgabe mit printf

- Beispiel für eine Ausgabe:
  - int i = 2;
  - int j = 67;
  - char c = ‘A‘;
  - printf(„i = %d, j = %d und c=%c\n“,i,c,j);
- Dies erzeugt die folgende Ausgabe:
  - i = 2, j = 65 und c=C
- Warum?

Eingabe mit scanf

- **scanf** dient der formatierten Eingabe von Daten.
- Die Eingabe und Umsetzung derselben wird (wie auch bei printf) über einen "Format-String" gesteuert.
- Beispiel: eine ganze Zahl einlesen:
  - int i;
  - printf("Bitte eine Zahl eingeben: ");
  - scanf("%d", &i);
- Beispiel: zwei Zahlen (eine ganze Zahl und eine Gleitkommazahl) einlesen:
  - int i;
  - float j;
  - printf("Bitte zwei Zahlen eingeben: a b ");
  - scanf("%d %f", &i, &j);