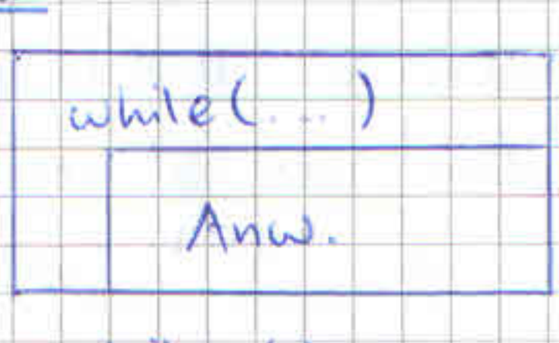
Nassi - Schneidermann

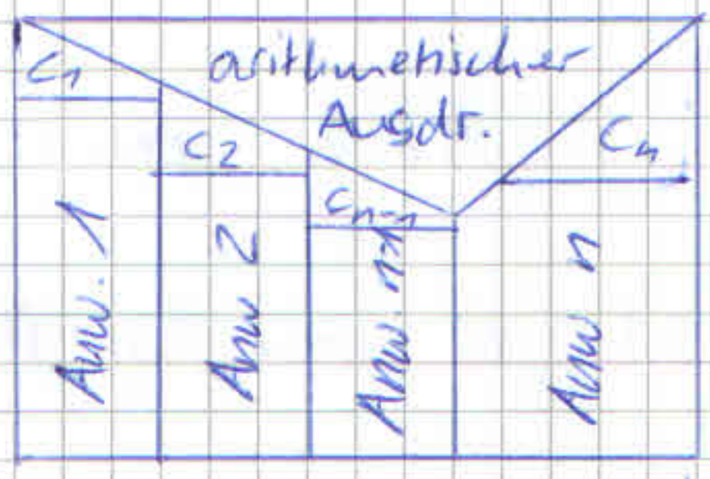


if (Selektion)





while / for ← Iteration → do while



switch (Selektion)



Lexikalische Konventionen

→ Zeichenvorrat in C S. 76

- einzelne Zeichen mit Hochkommas → 'a'
- Zeichenketten mit Doppelhochkommas → "abc"
- der Backslash wird für Ersatzdarstellungen benötigt
- Zeichen werden als 1 Byte dargestellt
- Lexikalische Einheiten:
  - Namen
  - res. Wörter
  - literale Konstanten
  - konst. Zeichenketten
  - Operatoren
  - Satzzeichen
- C ist "case sensitiv"!

→ Liste mit reservierten Wörtern S. 81

- Konstanten
  - symbolische Konstanten:

    # define PI 3.1415

    symbolische Konstanten haben einen Namen (benannte Konst.)
  - literale Konstanten:

    ganzzahlige Konstanten

    14 int - Konst. in Dez.

    014 int - Konst. in Okt.

    0x14 int - Konst. in Hex.

    14L Long - Konst.

    14U unsigned - Konst.

Zeichenkonstanten:
- ein Zeichen, eingeschlossen in Hochkommas
- um ' oder \ darstellen zu können, müssen Ersatzdarstellung her: für ' → '\'' oder für " → '\"'
- das erste Zeichen bei Ersatzdarstellung muss immer \ sein
→ Tabelle mit Ersatzdarstellungen: S. 86

Gleitpunktkonstanten:
- Beispiele: 30.0 3e2 3.e2 .3e2 usw
- e = 10^{Exponent}
- 10.3 ist vom Typ double. 10.3f ist vom Typ float

Aufzählungstypen (enums):
- enum boolean {False, true};
  False bekommt den Wert 0, true den Wert 1
- wird nichts anderes vereinbart, beginnt das Heraufzählen bei 0. Ansonsten wird einfach immer 1 dazugezählt

Konstante Zeichenketten (Strings):
- z.B.: "Ritchie" wäre der String: 'R' 'i' 't' 'c' 'h' 'i' 'e' '\0'

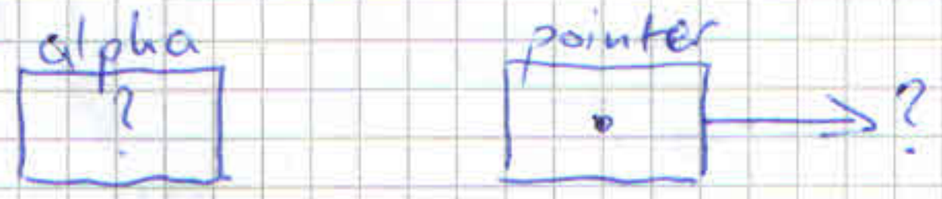
Datentypen & Variablen

- in C können beispielsweise eine int-Zahl einer float-Zahl zugewiesen werden (Nachkomma-Teil wird jedoch abgeschn.)
- der Datentyp legt Speicherbedarf, Wertebereich und Genauigkeit fest
→ Tabelle mit den Grössen der Datentypen S. 99
- es gibt interne und externe (globale) Variablen
- globale Variablen können grundsätzlich allen Funktionen zur Verfügung stehen
- Sichtbarkeit einer Variablen meint, dass man von einer Programmstelle aus über den Namen der Variablen auf sie zugreifen kann
- Lebensdauer ist die Zeit, während der der Variablen ein Platz im Speicher zur Verfügung steht
- externe Variablen leben so lange wie das Programm!

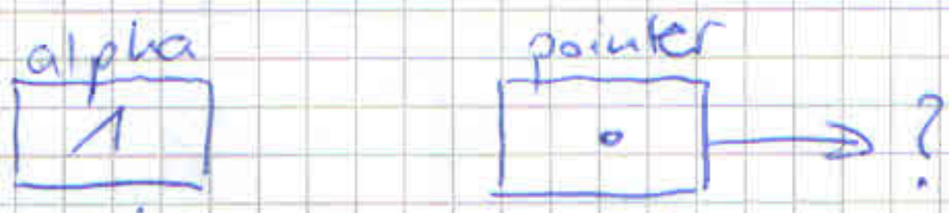
Pointer & Arrays

- Pointer können auf Datenobjekte, Pointervariablen und Funktionen zeigen
- Pointer und Speicher objekt sind vom Typ her gekoppelt (ein Pointer auf ein int muss den Typ int* haben)
- Definition eines Pointers: Typname* Pointername (int* p1)
- um einen Pointer auf ein Objekt zeigen zu lassen
  → p1 = &a (a ist Variable)
- um einen Pointer auf einen anderen Pointer zeigen zu lassen: int *b; int **c; (c ist ein Pointer auf ein Pointer auf int) → c = &b;

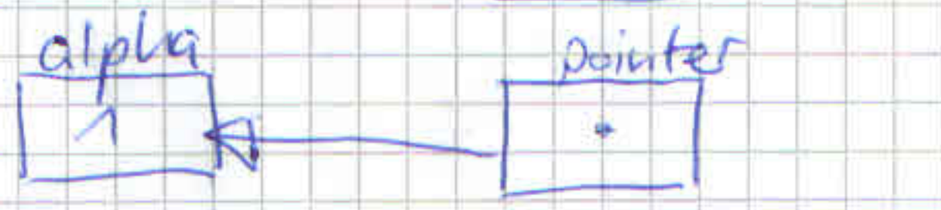
```
int alpha;
int * pointer;
```



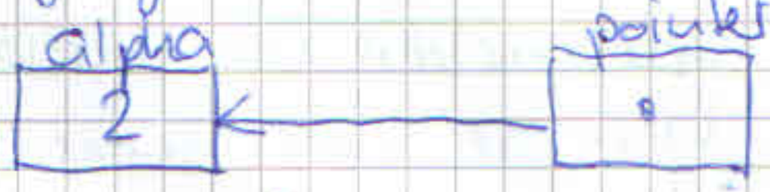

```
int alpha = 1;
int * pointer;
```




```
pointer = &alpha;
```



- Dereferenzierung mit dem Inhaltsoperator *
  *pointer = 2; → es wird dem Objekt, auf das pointer zeigt, 2 zugewiesen:



→ Grafik zu Referenzieren / Dereferenzieren S. 121

- int array[5]; array ist ein Array mit 5 int-Werten wobei der 1. Wert den Index 0 hat!
- entweder muss beim initialisieren eine Grösse in den Klammern angegeben werden oder die Elemente müssen bekannt sein! int arr[3]; oder int a[] = {1,2,3};
- Eingabe einer Zeichenkette von der Tastatur in einen String mit der Funktion gets()
  → char arr[40];
  gets(arr);

Anweisungen, Ausdrücke, Operatoren

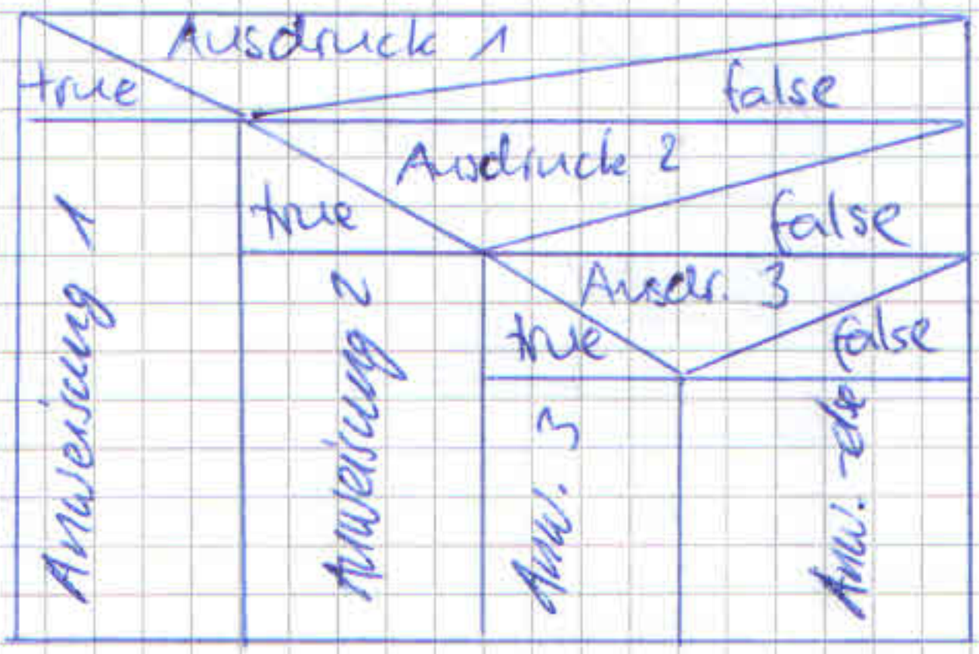
→ Prioritätenliste der Operationen S. 165

- = und == sind nicht dasselbe!
  = : Zuweisungsoperator
  == : Gleichheitsoperator (Vergleichsoperator)
- ≤ wird als <= und ≥ als >= dargestellt
- Links des Vergleichsoperators muss immer ein lvalue stehen → 3 == a geht z. B. nicht!

Kontrollstrukturen

→ Syntax der Verschiedenen Anweisungen S. 183 - 200

else - if




```
if (Ausdruck 1)
    Anweisung 1
else if (Ausdruck 2)
    Anweisung 2
else if (Ausdruck 3)
    Anweisung 3
else
```

- do while - Schleife mit ;! do { ... } while (Ausdr.);

Blöcke & Funktionen

→ Grafik zur Sichtbarkeit von Variablen S. 212

- globale Variablen können innerhalb von Blöcken durch lokale Variablen verdeckt werden!
- es können auf Funktionen mit denselben Namen wie Variablen verdeckt werden!
- Funktionen müssen vor ihrem Aufruf zumindest deklariert werden (Funktionsprototyp!) → Funktionsprototypen mit ; abschliessen!
- Gültigkeitsbereiche (scopes) sind: Datei, Funktion, Block und Funktionsprototyp

Fortgeschrittene Programmierung mit Pointern

- ist int a[5] ein Array mit 5 int-Elementen, dann zeigt ein pointer mit pointer = &a[i-1] auf das i-te Element im Array!
- der Name des Arrays kann als Pointer auf das 1. Element des Arrays verwendet werden! (a[0] = *a) und somit ist auch a[1] = *(a+1)
- mehrdimensionale Arrays z.B. int a[3][4]; entspricht einem Array aus 3 Zeilen und 4 Spalten

Strukturen & Unionen

- Ein Strukturtyp ist ein selbst definierter zusammengesetzter Datentyp (es muss jedes Komponente dessen Namen und Typ angegeben werden)
- es wird das Schlüsselwort struct dazu verwendet
- auf die Komponenten kann auf 2 versdiedene Arten zugegriffen werden

Beispiel:
```
struct koordinate
{
    float x;
    float y;
} punkt;
```
mit Pointer: `struct koordinate * p1;`
`p1 = &punkt;`
`p1 -> y = 4.00;` (mit Pfeiloperator)
oder
`punkt.y = 4.00;` (mit Punktoperator)
`(*p1).y = 4.00;` (dasselbe mit dereferenziertem P.)

- Unionen sind ähnlich wie Structures aber es ist jeweils nur ein Feld aktiv (abhängig vom Typ)
- die Grösse der Union ist so gross wie das grösste Feld der Union

Komplexere Datentypen, eigene Typnamen und Namensräume

→ ein paar gute Beispiele S. 333

- weitere Beispiele:

```
int* alpha[8];      alpha ist ein Array von 8 int-Pointern
int (*alpha)[8];    alpha ist ein Pointer auf ein Array mit 8 int-Werten
int*(*alph)[8];     "                                mit 8 int-Pointern
int* foo(..);       foo ist eine Funktion die einen int-Pointer
                    zurückgibt
int (*pFunc)(..);   pFunc ist ein Pointer auf eine
                    Funktion, die ein int zurückgibt
char* (*delta(..))[10]; delta ist eine Funktion,
                    die einen Pointer auf einen Array von 10 char-Pointern
                    zurück gibt
```

- typedef: Syntax →

```
typedef struct
{ :
} Artikel;
```

```
Artikel Messer;   ← Def. einer Variablen Messer des Typs Artikel.
Messer.preis = 1.5;   Initialisierung
```

- in C gibt es folgende Namensräume:
  Marken; Namen von Strukturen, Unions und enums;
  jede Struktur und Union für ihre Feldnamen;
  Bezeichner von Variablen, Funktionen, typedef-Namen
  und enum-Konstanten
- Namen, die zu verschiedenen Namensräumen gehören, dürfen auch innerhalb desselben Gültigkeitsbereichs gleich sein!