Programmieren

Sammlung gegliedert nach Modul

Fabian Suter, 19. Dezember 2023

https://github.com/FabianSuter/Programmieren.git

1 ProgC

1.1 Wichtige Kurzbefehle

`cd "Path"`	Pfad anwählen
`cd ..`	um eine Ebene nach oben (zurück)
`mkdir "Ordnername"`	Ordner erstellen
`rmkdir "Ordnername"`	Ordner löschen
`rm -rf *`	Alles innerhalb vom aktuellen Ordner löschen
`rm "Datei"`	Datei löschen
`mv "Name alt" "Name neu"`	Datei umbenennen
`cp "Datei alt" "Datei neu"`	Datei kopieren und benennen
`clang -Wall -o "Outputname" "Inputdatei"`	clang-Compiler mit Warnungen
`clang -Wall -o "Outputname" "Inputdatei" -lm`	-lm für Mathebibliothek
`ls`	Listet alle Files im akt. Verzeichnis auf
`ls -l`	Inkl. Informationen wie Grösse u.a.
`ls -a`	Inkl. versteckten Dateien
`ls -al`	Beide Varianten

1.2 Zahlensysteme

$2^0 = 1$	$2^1 = 2$	$2^2 = 4$	$2^3 = 8$	$2^4 = 16$	$2^5 = 32$	$2^6 = 64$	$2^7 = 128$

Grösse	**Abk.**	**Genauer Wert**	**Näherung**
Kilobyte	kB	2^{10} = 1024 Bytes	10^3 Bytes
Megabyte	MB	2^{20} = 1 048 576 Bytes	10^6 Bytes
Gigabyte	GB	2^{30} = 1 073 741 824 Bytes	10^9 Bytes
Terabyte	TB	2^{40} = 1 099 511 627 776 Bytes	10^12 Bytes

Oktal	3 Bits	X_8	X_O	X_q	X_oct	$0X$
Hex	4 Bits	X_16	X_h	XH	X_hex	0xX

Hexadezimal															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

ASCII (7-Bit) Ordnet gängigen Schriftzeichen einen Zahlenwert zu, um diese in einem Digitalrechner präsentieren zu können. Die Tabelle ist wichtig, um für geg. Schriftzeichen den repräsentierten Zahlenwert zu ermitteln (und umgekehrt).
Nachfolger: Unicode (8-, 16-, 32-Bit)

1.3 Datentypen

1.3.1 Datentypen

Ganzzahltypen		
char	Buchstaben, Zahlen	8 Bit (1 Byte)
short	kleine, ganzz. Werte	abh. (min 16bit)
int	eff. Grösse des Prozessors	abh. (min 16bit)
long	grosse ganzz. Werte	abh. (min 32bit)
long long	sehr grosse ganzz. Werte	abh. (min 64bit)
Gleitpunkttypen		
float	Gleitpunkt, single precision	abh.
double	Gleitpunkt, high precision, Standard	abh.
long double	Gleitpunkt, higher precision	abh.

Platzhalter		
char	`%c`	l, ll > Typ länger als ein `int`
Zeichenkette	`%s`	h, hh > Typ kürzer als ein `int`
Signed Ganzzahl dez.	`%d`	
Unsigned Ganzzahl dez.	`%u`	
Unsigned Ganzzahl okt.	`%o`	
Unsigned Ganzzahl hex.	`%x`	

Ganzzahlen können überlaufen!
Gleitpunktzahlen haben meist Rundungsfehler. Nie auf Gleichheit prüfen!

1.3.2 Typumwandlung

```
float f = 41.7;
```
Implizit: Eine Kommazahl ohne `f` am Ende hat den Typ `double`

```
int x = (int) f;
```
Explizit: x hat den Wert 41, Nachkommastellen werden abgeschnitten

1.3.3 Namen

1.3.4 Wertebereich

unsigned	$0...(2^n - 1)$	n=8 : 0...255
signed	$-2^{n-1}... + (2^{n-1} - 1)$	n=8 : -128...+127

1.4 Variablen

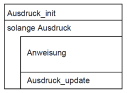
	Lokale Variable	**Globale Variable**
Sichtbarkeit	Zwischen Definition und Ende des aktuellen Blocks	Zwischen Definition und Ende der aktuellen Compile-Unit; über Deklaration `extern` auch in anderen Compile-Units importierbar
Lebensdauer	Laufzeit des zugehörigen Funktionsaufrufs	Laufzeit des Programms
Automatische Initialisierung	*keine*	automatische Initialisierung mit Wert 0

1.5 Schleifen

- `for`-Schleife: Für Zählschleifen, bzw. wenn die Anzahl Durchläufe bekannt ist
- `do...while`-Schleife: Keine Zählschleife, min. 1 Durchlauf
- `while`-Schleife: In allen anderen Fällen

1.5.1 For-Schleife

```
for (Ausdruck_init; solange Ausdruck; Ausdruck_update)
  Anweisung
```

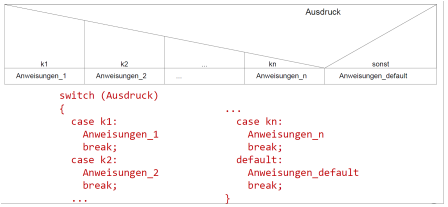


entspricht

```
Ausdruck_init;
while (solange Ausdruck)
{
  Anweisung
  Ausdruck_update
}
```

For-Schleife

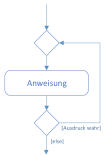
1.5.2 Switch-Schleife




```
switch (Ausdruck)
{
  case k1:
    Anweisungen_1
    break;
  case k2:
    Anweisungen_2
    break;
  ...
...
  case kn:
    Anweisungen_n
    break;
  default:
    Anweisungen_default
    break;
}
```

Switch-Schleife

1.5.3 Do-While-Schleife

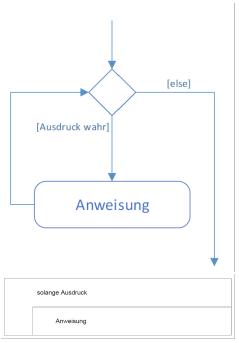



```
do
  Anweisung
while (Ausdruck);
```

hier braucht es einen Strichpunkt

Do-While-Schleife

1.5.4 While-Schleife



While-Schleife

1.5.5 Sprunganweisungen

- `break`: Schleifen abbrechen, zurückhaltend einsetzen!
- `continue`: nächsten Schleifendurchgang starten, sehr zurückhaltend einsetzen!
- `return`: aus Funktion zum Aufruf springen
- `goto`: zu einer Marke springen, VERMEIDEN!

1.6 Code-Snippets

1.6.1 Array und Pointer

```c
#include <stdio.h>

int main(){
        enum{array_size = 6};
        int test[array_size] = {1,2,3,4,5,6};
        for(int i =0; i<array_size; ++i)
                printf("Element %u: %i\n", i, test[i]);

        printf("Groesster: %d", *findAbsMax(test, array_size));
        return 0;
}
```

Main-Funktion zum Finden eines **betragsmässig** grössten Wertes innerhalb eines Arrays.

```c
int* findAbsMax(int* arr, size_t size){
        int* max_ptr = &arr[0];
        for(size_t i = 0; i < size; ++i){
                if((arr[i] >=0 && *max_ptr >=0 && arr[i] > *max_ptr)
                || (arr[i] <=0 && *max_ptr <=0 && arr[i] < *max_ptr)
                || (arr[i] >=0 && *max_ptr <=0 && arr[i] > *max_ptr * -1)
                || (arr[i] <=0 && *max_ptr >=0 && arr[i] * -1 > *max_ptr))
                        max_ptr = &arr[i];
        }
        return max_ptr;
}
```