

Programmieren

Sammlung gegliedert nach Modul

Fabian Suter, 30. 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	X _H	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

Typ	Anz. Bytes	Bereich	scanf	Spezielles
Ganze Zahlen				
byte	1	0 ... +255		
short	2	$-2^{15} \dots +2^{15} - 1$	%d; %i	Hex: %x; %X
int	4	$-2^{31} \dots +2^{31} - 1$	%d	Hex: %x; %X
long	8	$-2^{63} \dots +2^{63} - 1$	%ld; %li	Hex: %x; %X
Dezimalzahlen			(Expon.: %e)	
float	4	$1.2E - 38 \dots 3.4E + 38$	%f	6 Dez.stellen
double	8	$2.3E - 308 \dots 1.7E + 308$	%lf	15 Dez.stellen
Spezial				
char	1	Einzelne Buchstaben	%c	
boolean	1	True / False		
string		Zeichenkette; Text	%s	

Vorzeichen, Versch.				
unsigned char	1	0 ... +255	%c	
signed char	1	-128 ... +127	%c	
unsigned int	4	$0 \dots +2^{32} - 1$	%u	
short int	2	$-2^{15} \dots +2^{15} - 1$	%hd	
unsigned short int	2	$0 \dots +2^{16} - 1$	%hu	
long int	4	$-2^{31} \dots +2^{31} - 1$	%ld	
unsigned long int	4	$0 \dots +2^{32} - 1$	%lu	
long long int	8	$-2^{63} \dots +2^{63} - 1$	%lld	
unsigned long long int	8	$0 \dots +2^{64} - 1$	%llu	
long double	16	$3.3E - 4932 \dots 1.1E + 4932$	%Lf	18 Dez.stellen

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	$0 \dots (2^n - 1)$	n=8 : 0...255
signed	$-2^{n-1} \dots + (2^{n-1} - 1)$	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

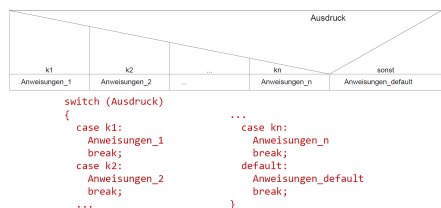
1.5.1 For-Schleife

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



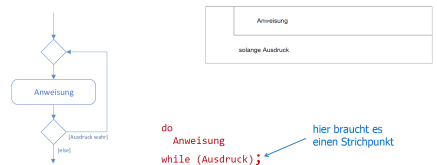
For-Schleife

1.5.2 Switch-Schleife



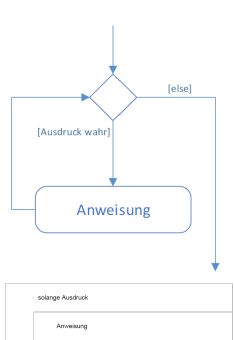
Switch-Schleife

1.5.3 Do-While-Schleife



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

```
#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.

```
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;
}
```