APPLIED COMPUTER SCIENCES

FH | JOANNEUM
University of Applied Sciences

Systemnahe Programmierung
SS 2024

Unit 2

Helmut Lindner

Variable und Datentypen

Variablendeklaration

Variable müssen vor der Nutzung deklariert werden

```
int a;

int a,b,c;              -> Mehrfachdeklaration
```

Initialisierung

```
int a = 2;

int a, b,c = 1, d=3;
```

Variable und Datentypen

Regeln für Variablennamen

Gültige Zeichen: Buchstaben, Zahlen, '_'

- Dürfen nicht mit einer Zahl beginnen
- Keine Schlüsselwörter
- Sind case-sensitiv

Gültig:

```
int _Zahl
float s_Weight3
```

Ungültig:

```
char 1_variable -> Zahl darf nicht erstes Zeichen sein
double my$money; -> $ ist nicht erlaubtes Zeichen
```

Variable und Datentypen

Speicherklassen auto und register

Variablen, die ohne zusätzliche Speicherklasse definiert werden, haben die Speicherklasse `auto`.

```
auto int i;  = int i;
```

Die Speicherklasse `register` dient dazu, dem Compiler mitzuteilen, dass die Variable wenn möglich in einem Register gehalten werden soll (z.B.: Indexvaraible)

```
register int i;
```

Die Speicherklasse `register` sollte im Normalfall nicht verwendet werden, da mittlerweile die Compiler so gut optimierten Code erzeugen, dass eine manuelle Registerzuordnung die Optimierung eher schwächt.

Variable und Datentypen

Datentyp	**Untertypen**	**Größe** (bytes)
char	signed char (-128 - 127) unsigned char (0-255)	1
short	signed short unsigned short	2
int	signed int unsigned int	4
long	signed long unsigned long	8
long long	signed long long unsigned long long	8..
float	-	4
double	-	8
long double	-	16

Wenn der Untertyp nicht angeführt wird, dann ist der Type signed (`int = signed int`).
Die Größe kann je nach System unterschiedlich sein, fix ist nur `sizeof(char)` = 1 Byte
Wenn Typen mit fest definierter Bitlänge benötigt werden, dann sollten die Typen aus `stdint.h` verwendet werden. Bsp: `uint16_t`

Variable und Datentypen

Übung

Schreiben sie ein Programm, das den benötigten Speicherplatz für die Datentypen `char, short, int, float` und `double` ausgibt.

Definieren sie jeweils eine Variable vom entsprechenden Typ und berechnen sie deren Speicherbedarf.

Hinweis: verwenden sie den `sizeof()` Operator und die `printf()` Funktion.

Variable und Datentypen

Kein boolean Datentyp?

Alle C-Ausdrücke `expr` evaluieren zu

`false` ... wenn `expr == 0`
`true` ... wenn `expr <> 0`

```
char c=0;

if (c) ... false

int i=-99;

if (i) ... true
```

Seit C99 ist ein Boolean Datentyp mit `<stdbool.h>` verfügbar

Datentyp `_Bool`

oder Macro `bool`

```
bool toBeOrNotToBe = true;
```

Literale

Numerische Literale

Untertyp angeben

```
u ... unsigned unsigned int a = 30u;
l ... long     long a = 30l;
               unsigned long a = 30ul;
```

Zahlensystem

```
dezimal        30
oktal          036
hexadezimal    0x1E
```

Fliesskommazahlen

```
f ... float            float f = 0.7f;
l ... long double      long double ld = 0.7l;
```

Entweder mit Komma oder Exponentialschreibweise

```
float f = 0.007;
float f = 7E-3;
```

Literale

Zeichen und Strings

Zeichen (Typ char)

```
‘A‘
‘\x41‘      ... Hexadezimalcode
‘\0101‘     ... Oktalcode
```

Strings

```
“Hello World“  “Hello““World“
```

Escape Sequenzen in Strings

```
\  ... Escape Character
\\ ... \
\‘ ... ‘
\“ ... “
\b ... Backspace
\t ... Tab
\r ... Carriage Return
\n ... Linefeed
...
```

Konstante & Enums

Konstante deklarieren

```
const int = 1;
const char msg[] =“Hello world“;
```

Enums

Enums bilden eine konstante Menge, deren Elementen ein Zahlenwert zugewiesen wird.

```
enum BOOL {no, yes } …  no=0, yes=1
enum COLOR { red=1, blue, green, yellow=10 } … blue=2, green=3
```

Operatoren

Artithmetische Operatoren

```
+, -, *, /, %
```

Relationale Operatoren

```
>, >=, <, <=, ==, !=
```

Logische Operatoren

```
&&, ||, !
```

Wenn der Wert eines Ausdrucks feststeht, dann wir der Rest nicht mehr ausgewertet!

Beispiel:

```
(1==1) || ((c=getchar())==‘y‘) -> der zweite Teil wird nicht ausgeführt
(0) && ((x=x+1)>0)             -> der zweite Teil wird nicht ausgeführt
```

Sonstige Operatoren

```
sizeof(<expr>) -> berechnet den tatsächlichen Speicherverbrauch von expr
```

Operatoren

Increment/decrement Operatoren

```
++, --
```

Postfix

```
x++     y=x++ -> y=x; x=x+1;   -> x wird zuerst zugewiesen, dann erhöht.
x--     y=x-- -> y=x; x=x-1;
```

Prefix

```
++x     y=++x -> x=x+1;y=x;   -> x wird zuerst erhöht, dann zugewiesen.
--y     y=--x -> x=x-1;y=x;
```

Zuweisungs-Operatoren

Zuweisungsoperator: `=`

Shorthand Varianten

```
+=, -=, *=, /=, %=
    i += 8; -> i = i + 8;
```

Operatoren

Übung

Schreiben sie ein Programm, das folgende Aussagen prüft:
Sie haben 3 Variablen

```
int x = 10;
int y = 1;
int z;
```

Wie groß sind `x,y,z` nach folgendem Ausdruck?

```
z = (y++ + x);
```

und nach folgendem?

```
z = (--y + x);
```

Operatoren

Bitweise Operatoren

```
&  ... AND
|  ... OR
^  ... XOR
~  ... NOT          (Einerkomplement)
<< ... shift left
>> ... shift right
```

Beispiel

```
0x01<<4 = 0x10
0x10>>4 = 0x01
```

Ein Leftshift um n Stellen entspricht einer Multiplikation mit 2^n

Shorthand Varianten

`<<=,  >>=,  &=, |=, ^=`

Operatoren

Häufige Bit Operationen

Bitoperationen werden häufig dazu verwendet einzelne Bits zu manipulieren.

```
char c;
```

Bit n setzen (= 1)

```
c = c | (1<<n)
```

Bit n löschen (= 0)

```
c = c & ~(1<<n)
```

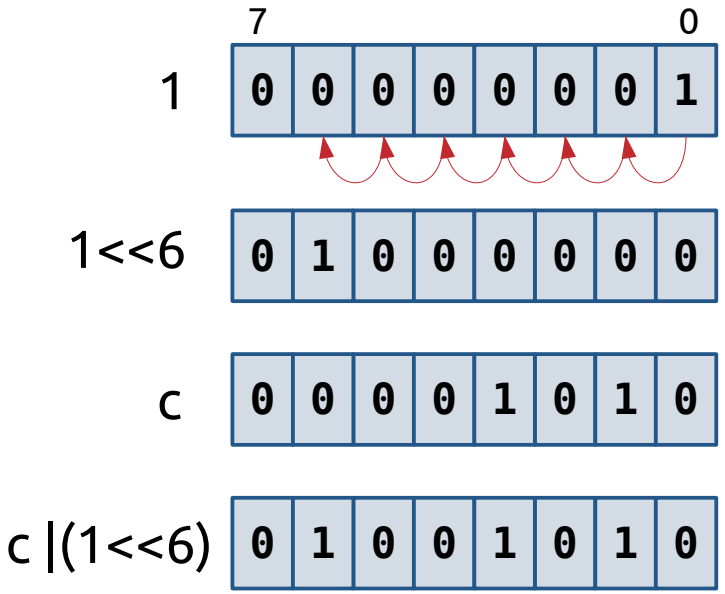
Bit n invertieren

```
c = c ^ (1<<n)
```

Abfragen, ob Bit n gesetzt ist

```
c & (1<<n)
```

Beispiel: Bit 6 setzen in `char c=10;`



Operatoren

Rangfolge Operatoren

Typ	Prio	Operatoren	Assoziativität
Einstellig	15 14	[] . -> () ! ~ ++ -- & * (typecast) sizeof + -	links -> rechts **rechts -> links**
Arithm.	13 12	* / % + -	links -> rechts links -> rechts
Bitweises Schieben	11	<< >>	links -> rechts
Vergleich	10 9	< <= > >= == !=	links -> rechts links -> rechts
Bitweise boolesche Ops.	8 7 6	& ^ |	links -> rechts links -> rechts links -> rechts
Logisch	5 4	&& ||	links -> rechts links -> rechts
Tern.Op	3	? :	**rechts -> links**
Zuweisung	2	= += -= *= /= <<= >>= &= ^= |=	**rechts -> links**
Sequenz	1	,	links -> rechts

Operatoren

Übung

Definieren sie eine `char` Variable und initialisieren sie diese mit dem Kleinbuchstaben `'q'`.

Wandeln sie den Inhalt der Variable unter Benutzung arithmetischer Operationen in einen Großbuchstaben um.

Das Programm sollte folgende Ausgabe erzeugen:

```
Kleinbuchstabe: q Großbuchstabe: Q
```

Hinweis
Die Buchstaben A-Z und a-z sind im ASCII Code sequenziell abgelegt.

- A-Z entspricht dezimal 65-90
- a-z entspricht dezimal 97-122

Einfache Ein/Ausgabe

stdin, stdout, stderr

Die drei Streams `stdin`, `stdout` und `sterr` sind per default in jedem Programm definiert und zum Schreiben bzw. Lesen bereit.
`stdio.h` bietet einige Funktionen, mit denen einfach auf diese Streams zugegriffen werden kann.

`EOF` ... Rückgabewert wenn End-Of-File erreicht wurde (Stream geschlossen)

```
int getchar(void);
```
Ein Zeichen von `stdin` lesen, Rückgabewert ist das gelesene Zeichen oder `EOF` (deswegen `int`). Abschluss mit `<Enter>` Taste notwendig.

```
int putchar(int ch);
```
Ein Zeichen auf `stdout` ausgeben, Rückgabewert ist das Zeichen das ausgegeben wurde oder `EOF`.

Einfache Ein/Ausgabe

scanf und printf

```
int scanf(const char * restrict format,...);
```

Liest die Parameter formatiert von `stdin` ein. Rückgabewert ist die Anzahl der gelesenen Objekte.

```
int zahl1, zahl2;
int gelesen = scanf("%d %d",&zahl1, &zahl2);
```

```
int printf(const char * restrict format,...);
```

Gibt die Parameter formatiert auf `stdout` aus

```
char c; int d;
printf("char: %c int: %d\n",c,d);
```

printf Formate

Aufbau: `%[flags][width][.precision][modifier]<type>`
type:

type	Bedeutung	Beispiel
d,i	integer	`printf("%d",10);` -> 10
x,X	integer (Hex)	`printf("%x",10);` -> 0xA
u	unsigned integer	`printf("%u",10);` -> 10
c	character	`printf("%c",'A');` -> A
s	string	`printf("%s","hello");` -> hello
f	float	`printf("%f",2.3);` -> 2.3
d	double	`printf("%lf",2.3);` -> 2.3
e, E	float Exponentialdarstellung	`printf("%e",220.33);` -> 2.0033e+02
a,A	Hexadezimale float Darstellung	`printf("%a",10);` -> 0x1.4p+3
p	Zeiger Hexadezimal	`printf("%p",&i);` 0x0012ae2ff
%	%-Zeichen	`printf("%d %%",10);` -> 10%

printf Formate

Aufbau: `%[flags][width][.precision][modifier]<type>`

flags:

`-`	...	linksbündig ausrichten (default rechtsbündig)
`+`	...	Vorzeichen immer ausgeben
`#`	...	Oktal/Hexadezimalzahlen mit 0, 0x ausgeben
`0, Leerzeichen`	...	Links mit 0 oder Leerzeichen auffüllen

width:

Mindestanzahl an Zeichen die ausgegeben werden (kein Abschneiden)

Beispiele:

```
printf("%3d",2); -> __2   ( __ = 2 Leerzeichen)
printf("%7s","hello"); -> __hello

printf("%-7s","hello"); -> hello__
printf("%+3d",21); -> +21
printf("%#8x",21); -> 0x15
printf("%04d",21); -> 0021
```

printf Formate

Aufbau: `%[flags][width][.precision][modifier]<type>`

precision:

- Anzahl der Zeichen für %d, %i, %o, %u und %x
- Anzahl der Nachkommastellen für %a, %e, %f

```
printf("%.6d",314); -> ___314
printf("%.2f",3.14159); -> 3.14
printf("%05.2f",3.14159); -> 03.14
```

modifier:

Datensubtyp anpassen

```
h ... als short interpretieren (mit i,d,u,x)
l .. als long interpretieren (mit i,d,u,x)
L .. als double interpretieren (mit e,f)

printf("%Lf",314159.0l); -> 314159.000000
```

printf Formate

Verwendung

Der Formatstring von `printf` kann folgende Elemente enthalten:

- Formatausdrücke
- Text
- Escape Sequenzen

```
printf (“Spalte 1 %d\tSpalte 2 %.2f\t Spalte 3 %s\n“,wert1,wert2,wert3);
```

Bei `scanf` dürfen nur Formatausdrücke verwendet werden. Soll vor einer Eingabe ein Text stehen, so muss dieser vorher mit `printf` oder einer anderen Funktion ausgegeben werden.

Beispiel:

```
printf (“Bitte geben Sie 2 Zahlen ein:“);
anzahl = scanf(“%d %d“, &zahl1, &zahl2);
```

Formate

Übung

Lesen sie mit der `getchar()` Funktion eine `int` Variable ein und geben sie deren ASCII-Wert als Dezimalzahl und als Hexadezimalzahl aus.

Beispiel:

```
w
Der ASCII Wert von w ist 119/0x77
```

Typumwandlung

Automatische Typkonvertierungen

`char` wird bei Ausdrücken immer in `int` umgewandelt.

```
char c;  (c++) -> int
```

`float` wird bei Ausdrücken immer in `double` umgewandelt.

In Ausdrücken mit gleichem Grundtyp wird das Ergebnis immer in den größten vorkommenden Datentyp konvertiert.

Ganzzahlige Typen:

```
char < short < int < long < long long
```

Beispiel: `int * long -> long`

Gleitkomma-Typen:

```
float < double < long double
```

Beispiel: `double / long double -> long double`

Die Konvertierung von längeren Datentypen in kürzere kann zu Datenverlust führen.

Typumwandlung

Gemischte Ausdrücke

Bei der Auswertung von Ausdrücken wird solange wie möglich nach `int` konvertiert.
Sobald ein Gleitkommatyp im Ausdruck vorkommt, wird der gesamte Ausdruck in Gleitkomma umgewandelt.

Übung

Wie lautet das korrekte Ergebnis der Berechnung `(c+i)*f` ?

```
int i=2;
char c=‘a‘;
float f = 3.14159;

printf (?);
```

Typumwandlung

Explizite Typkonvertierungen - Cast

Syntax: `(Typ) <Ausdruck>`

```
c = (char)65; // 'A'
double pi_i = 3.14159;

float some_num = (float) c * pi_i; // 3.0
```

Blöcke

Statement

```
i=0;
printf(“Out: %d\n“, i);
```

Compound Expressions

Durch Komma getrennte Ausdrücke:

```
i=0, j=0, puts(“Test“);
```

Block

Alles zwischen `{` und `}`

Beispiel

```
int main (int arvc, char *argv) { ... }
```

Anweisungen, denen kein Anweisungsblock folgt, werden mit einem Semikolon abgeschlossen.

Conditionals

If-else

```
if (expr)       -> expr == 0 -> false, sonst true!
    Block
else if (expr)
    Block
else
    Block
```

Verschachtelung

```
    if (expr)
        if (expr)...
    else            -> zuerst wird immer das innerste if abgeschlossen
        ...

    if (expr){
        if (expr) ...
    }
    else            -> so gehört das else zum äusseren if
        ...
```

Conditionals

Ternärer Operator

```
expr ’?’ expr1 ’:’ expr2
```

Entspricht:

```
if (expr)
    expr1;
else
    expr2;
```

Beispiel:

```
x>100 ? “alt“ : “jung“;
```

Conditionals

Übung

Schreiben sie ein Programm, das prüft, ob ein Jahr ein Schaltjahr ist. Das Jahr soll als `int` mit `scanf()` eingelesen werden.

Folgende Regeln gelten für Schaltjahre:

- Ist die Jahreszahl durch 400 teilbar, ist es ein Schaltjahr
- Ist die Jahreszahl durch 100 teilbar aber nicht durch 400, ist es ein Schaltjahr.
- Ist die Jahreszahl durch 4 teilbar aber nicht durch 100, dann ist es ein Schaltjahr.
- In allen anderen Fällen ist das Jahr kein Schaltjahr.

Beispiel:

```
Jahr eingeben: 2011
2011 ist kein Schaltjahr.
```

Conditionals (3)

switch

```
switch (expr) { -> expr muss zu einer ganzen Zahl auswertbar sein! (int oder char)
    case 1 :
        /* block or statement */
    break;
    case 2 :
        /* block or statement */
    break;
    default:
        /* block or statement */
    break;
}
```

Conditionals (4)

switch

Switch ist wie eine „Sprungtabelle“
Der Code wird vom ersten „case“ Match ausgeführt bis ein „break“ auftritt.

```
switch (num) {
    case 1 :
    case 2 :
        /* do something if num == 1 or 2 */
    break;
    case 3 :
        /* do something else if num == 3 */
    break;
    case 4 :
        /* if num == 4: do this ... */
    case 5 :
        /* and do that if num == 4 or 5 */
    break;
    default:
        /* block or statement */
    break;
}
```

Schleifen

while

Der Body wird solange ausgeführt, solange die Auswertung von `(expr) true` ergibt.

```
while (expr)
    /* body */
```

Wenn `expr` beim ersten Aufruf `false` liefert, wird der Body nie durchlaufen.

Beispiel: `while ( i< 10 ) i++;` oder kürzer `while(++i<10);`

do while

Der Body wird solange ausgeführt, solange die Auswertung von `(expr) true` ergibt.

```
do /* body */
while (expr);
```

Der Body wird mindestens einmal durchlaufen.

Beispiel: `do i++; while ( i< 10 );`

Schleifen

for

```
for (initialization; condition; increment)
    /* body */
```

Beispiel:

```
for (i=0; i<1000; i++)
    printf(“%d“,i);
```

Die einzelnen Ausdrücke können auch leer sein, „increment“ wird dann als true angenommen.

Schleifen

break

Mit `break` kann jede Schleife vorzeitig verlassen werden.

```
while ( i < 15 ){
    if ( i%2 == 0 )
        break;
    /* do something ...*/
}
```

continue

Mit `continue` wird der Rest der Schleife übersprungen und mit der Loop-Condition fortgesetzt.

```
for (i=0, j=1000; i<1000; i++, j--) {
    if (j == i) // skip if i==j
        continue;
    printf("%d",i);
}
```

Schleifen

Übung a

Schreiben sie ein Programm, das eine Zahl zwischen 1 und 7 einliest und den entsprechenden Wochentag als Text ausgibt.
Das Programm soll solange Zahlen einlesen bis die Zahl `0` eingegeben wurde.
Bei Eingabe einer ungültigen Zahl soll eine entsprechende Meldung ausgegeben werden.

Verwenden sie eine `while` Schleife, ein `switch` Statement und `scanf()` dazu.

Übung b

Schreiben sie ein Programm, das eine ganze Zahl einliest und deren Fakultät `n!` Berechnet und auf `stdout` ausgibt.

Wobei: $n! = 1 * 2 * 3 * \ldots n$ oder $n! = \prod_{k=1}^{n} k$

Verwenden sie eine `for` Schleife und `scanf()` dazu.

Funktionen

Definition

Syntax:

```
<returntype> function_name(args) { statements; }
```

Beispiel:

```
float flaecheEllipse(float a, float b) {
    float pi=3.14159; -> lokale Variable

    return a*b*pi;
}
```

Returntype = `void ->` Es wird kein Wert zurückgegeben.
Wenn kein Returntype angegeben wird, dann wird `int` angenommen.
Ein Funktion muss vor ihrer Verwendung deklariert oder definiert werden.

Parameter

Die Parameterübergabe findet immer By-Value statt.

Funktionen

Unterschied Definition - Deklaration

Deklaration

Die Deklaration beschreibt den Namen, Rückgabewert und Parameter einer Funktion = Signatur der Funktion.
Sie wird benötigt, wenn die Funktion in anderen Compilation-Units (.c Dateien) verwendet wird oder wenn sie vor ihrer Definition verwendet wird.

```
int add(int a, int b);
```

Die Deklaration sollte immer in einer Headerdatei (.h) angegeben werden.

Definition

Die Definition legt das Verhalten der Funktion fest.

```
int add(int a, int b) {
    return a+b;
}
```

C-Standard Bibliothek

Mathematische Funktionen

```
#include <math.h>

double sqrt(double x);
double pow(double x, double y)
double exp(double x);
double log(double x);
double sin(double x);
double cos(double x);
…
```

Die Mathematikfunktionen befinden sich in einer eigenen Bibliothek. Diese muss zum Executable dazu gelinkt werden.
In diesem Fall kann mit dem Flag

```
-l<Bibliotheksname>
```

die gewünschte Bibliothek angegeben werden. Die Mathematikbibliothek hat den Namen `m`. (Finden mit: `locate libm.a`)

Also:

```
cc ex1.c -o luftballon -lm
```

C-Standard Bibliothek

Funktionen für Zeichen

```
#include <ctype.h>

int isalnum(int c) testen auf alphanumerisches Zeichen (a-z, A-Z, 0-9)
int isalpha(int c) testen auf Buchstabe (a-z, A-Z)
int iscntrl(int c) testen auf Steuerzeichen (\f, \n, \t ...)
int isdigit(int c) testen auf Dezimalziffer (0-9)
int isgraph(int c) testen auf druckbare Zeichen ohne Leerzeichen
int islower(int c) testen auf Kleinbuchstaben (a-z)
int isprint(int c) testen auf druckbare Zeichen mit Leerzeichen
int ispunct(int c) testen auf druckbare Interpunktionszeichen
int isspace(int c) testen auf Whitespace
int isupper(int c) testen auf Großbuchstaben (A-Z)
int isxdigit(int c) testen auf hexadezimale Ziffern (0-9, a-f, A-F)
```

Returnwert <> 0 : Bedingung erfüllt

```
int tolower(int c) wandelt Groß- in Kleinbuchstaben um
int toupper(int c) wandelt Klein- in Großbuchstaben um
```

Funktionen

Übung

Schreiben sie eine Funktion `flaecheDreieck(...)`, die den Flächeninhalt eines Dreiecks nach der Formel von Heron berechnet und zurückgibt.

$$A=\sqrt{s(s-a)(s-b)(s-c)}$$

$$s=\frac{a+b+c}{2}$$

Übergeben sie der Funktion die Werte von `a,b` und `c` als Variable vom Typ `double`.

Lesen sie die Werte von `a,b,c` mit `scanf()` ein.

Funktionen

Übung

Schreiben sie eine Funktion `printBinary(unsigned char c)` die eine `unsigned char` Variable entgegen nimmt und diese in binärer Form ausgibt.
Testen sie die Funktion mit den Zahlen 10, 128 und 91 (in dieser Reihenfolge).

Beispiel:

```
00001010
...
```

Hinweis:
Benutzen sie die Bitoperatoren um ein einzelnes Bit darzustellen und schieben sie die Zahl nach rechts.
Benutzen sie Zeichenarithmetik um das darstellbare ASCII-Zeichen (0 oder 1) zu berechnen.
Die Anzahl der Stellen des char Datentyps (`CHAR_BIT`) können sie aus `limits.h` erfahren.

Scope

Scope von Variablen

scope.c

```
#include<stdio.h>

float x = 3.1;
float y = 4.8;        -> Globale Variable

void func() {
        printf("func x: %.2f y: %.2f\n", x,y);
}

int main(int argc, char *argv[]) {
        int x=5; // -> lokal zu main-Block

        printf("main Funktion x: %d y: %.2f\n",x,y);
        func();
        {
                float y = 2.2; // -> lokal zu innerem Block
                printf("Innerer Block x: %d y: %.2f\n",x,y);

        }
        return 0;
}
```

Speicherklasse static

Die Speicherklasse `static` hat im Kontext einer globalen und einer lokalen Variable eine unterschiedliche Bedeutung.

Lokale static Variable

Eine lokal zu einer Funktion mit „`static`“ definierte Variable bleibt für die gesamte Ausführung des Programms erhalten.

```
int add(int a) {
    static int x=0;   -> der Wert von x bleibt über jeden Aufruf von add() erhalten.

    x += a;
    return x;
}
```

Beispiel:

```
add(3);     -> 3
add(5);     -> 8
add(87);    -> 95
```

Speicherklasse static

static - globale Variable und Funktionen

Ist eine globale Variable oder eine Funktion mit „`static`“ deklariert, dann ist ihre Sichtbarkeit auf die beinhaltende Datei beschränkt.

scope.c

```
#include<stdio.h>

float x = 3.1;
float y = 4.8;          -> Globale Variable, im ganzen Programm sichtbar

static int zahl=0;      -> Globale Variable, nur in scope.c sichtbar
```

```
static void func() {
        printf("Funktion: %.2f %.2f\n", x,y);
}

  ...
```