13.9 Gültigkeitsbereiche und Speicherklassen (Fortsetzung)

Es gibt vier Speicherklassen:

auto Lokale Variablen einer Funktion werden erzeugt wenn die Funktion aufgerufen wird und gelöscht (*), wenn die Funktion verlassen wird. Solche Variablen heißen "automatische Variablen".

static Innerhalb einer Funktion deklariert: Lokale Variablen, die ihren Wert zwischen den Funktionsaufrufen behalten.
Außerhalb der Funktionen deklariert: Modul-globale Variablen, d.h. innerhalb des Moduls global, außerhalb des Moduls unbekannt.

extern Außerhalb der Funktionen deklarierte Variablen stehen allen Funktionen des Moduls zur Verfügung (= globale Variablen).
Durch Deklaration mit "extern" können diese Variablen sogar über Modulgrenzen hinweg *importiert* werden.

register Variablen, die möglichst in den Prozessorregistern gehalten werden sollten (Geschwindigkeitsvorteil).

(*) gelöscht werden die Variablen/Verwaltung, nicht die Daten im Speicher selbst

ÜBUNG: (Gültigkeitsbereiche)

```c
/* --- Modul_1.c --- */
#include "Modul_2.h"

int calc_i();     /* Fkt.-Deklaration  */

static int i=0;   /* Modul-global      */

int main(){

    int i=1, m;           /* lokal */
    printf("i=%d \n", i);
    printf("i=%d \n", calc_i());
    printf("i=%d \n", calc_i_ext());
    printf("k=%d \n", k);

    return 0;
}
```

```c
int calc_i(){     /* Fkt.-Definition */
    return i;
}
```

Was wird ausgedruckt ?
Ist etwas falsch ?

```c
/* -- Modul_2.c -- */


int k=15; /* global */
static int i=2;

int calc_i_ext(){
    return i;
}
```

```c
/* -- Modul_2.h --*/

extern int k;
int calc_i_ext();
```

```
/* --- Modul_1.c --- */

  /* -- Modul_2.h -- */
  extern int k;
  int calc_i_ext();

int calc_i();     /* Fkt.-Deklaration  */

static int i=0;   /* Modul-global      */

int main(){

    int i=1, m;           /* lokal */
    printf("i=%d \n", i);
    printf("i=%d \n", calc_i());
    printf("i=%d \n", calc_i_ext());
    printf("k=%d \n", k);

    return 0;
}
```

```
int calc_i(){
    return i;
}
```

```
/* -- Modul_2.c -- */


int k=15; /* global */
static int i=2;

int calc_i_ext(){
    return i;
}
```

```
/* --- Modul_1.c --- */
```

```
/* -- Modul_2.h -- */
extern int k;
int calc_i_ext();
```

```
int calc_i();     /* Fkt.-Deklaration  */
static int i=0;   /* Modul-global      */
```

```
int main(){

    int i=1, m;           /* lokal */
    printf("i=%d \n", i);
    printf("i=%d \n", calc_i());
    printf("i=%d \n", calc_i_ext());
    printf("k=%d \n", k);

    return 0;
}
```

```
int calc_i(){
    return i;
}
```

```
/* -- Modul_2.c -- */


int k=15; /* global */
static int i=2;
```

```
int calc_i_ext(){
    return i;
}
```

```
/* --- Modul_1.c --- */

  /* -- Modul_2.h -- */
  extern int k;
  int calc_i_ext();

int calc_i();     /* Fkt.-Deklaration  */

static int i=0;   /* Modul-global      */

int main(){

    int i=1, m;           /* lokal */
    printf("i=%d \n", i);
    printf("i=%d \n", calc_i());
    printf("i=%d \n", calc_i_ext());
    printf("k=%d \n", k);

    return 0;
}

int calc_i(){
    return i;
}
```

```
/* -- Modul_2.c -- */


int k=15; /* global */
static int i=2;

int calc_i_ext(){
    return i;
}
```

```
/* --- Modul_1.c --- */

   /* -- Modul_2.h -- */
   extern int k;
   int calc_i_ext();

int calc_i();     /* Fkt.-Deklaration  */

static int i=0;   /* Modul-global      */

int main(){

    int i=1, m;           /* lokal */
    printf("i=%d \n", i);
    printf("i=%d \n", calc_i());
    printf("i=%d \n", calc_i_ext());
    printf("k=%d \n", k);

    return 0;
}

int calc_i(){
    return i;
}
```

(Verw. auf) ***Globale Variable !***
Möglichst nicht verwenden !

```
/* -- Modul_2.c -- */


int k=15; /* global */
static int i=2;

int calc_i_ext(){
    return i;
}
```

ENDE vorgezogenes Kapitel 13.9

13.6 Anweisungen

Sequenzen

- Verbund-Anweisung
- Ausdrücke als Anweisungen
- Funktionsaufrufe

– **Verzweigungen**

- if-else - Anweisung
- switch - Anweisung

– **Schleifen**

- while - Anweisung
- do-while - Anweisung
- for - Anweisung

13.6.1 Verbund-Anweisung

Die Verbundanweisung (*compound-Statement*) – auch Block genannt - eine mit '{' und '}' geklammerte Folge von einzelnen Deklarationen und Statements.

Syntax:

compound statement = '{' { *declaration* } { *statement* } '}' .

Der Verbund dient dazu einzelne Anweisungen zu gruppieren,
um sie syntaktisch wiederum als eine Anweisung betrachten zu können.

Der Verbund ist auch ein Namensraum/Gültigkeitsbereich für die in ihm deklarierten Datenobjekte.

Deklarationen (Vereinbarungen) können an jeder Stelle getroffen werden. Es ist sinnvoll sie zu Beginn des Blockes vorzunehmen.

Der Kontrollfluss in der Verbundanweisung ist der einer Sequenz.

```
{
    int a, b, c;    a=1;    b=2;    c=a+b;
}
```

13.6.2 IF-ELSE-Anweisung

Syntax:

'**if**' '(' *expression* ')' *statement* ['**else**' *statement*]

Beispiele:

```
if ( a < b )  b = a;    else a = b;  // ohne Einrückung
                                     // möglich aber unschön

if ( a < b )             // ohne Verbundanweisung
   b = a;                // schöner aber riskant
else
   a = b;


if ( a >= b) {           // mit Verbund-Klammern (am besten)
         h = sin(b);
         a = b*b;
}
```

Anmerkung: In C gilt jeder Wert ungleich 0 als "Wahr" !

13.6.2 IF-ELSE-Anweisung (Fortsetzung)

Ein häufig angewendeter Konstrukt ist die sog. *Else-If-Kette* :

'if' '('*expression*')'
    statement
'else' 'if' '('*expression*')'
    statement
'else' 'if' '('*expression*')'
    statement
...

...

'else'
    statement

Beispiel:

```
if(a<0){
        V=-1;
        printf("negative! \n");
}
else if(a==0){
        V=0;
        printf("zero! \n");
}
else{
        V=1;
        printf("positive! \n");
}
```

Wichtig : Ein *else* gehört immer zum (rückwärtsgehend) letzten *if*, welches noch kein *else* hat (Verbundanweisungen überspringen).

ÜBUNGEN: if-else-Anweisung

Welcher Wert wird ausgegeben? (Achtung: C-Puzzles, ganz schlechter Code)

Aufg. 1

```
int i=0, a=0, b=2;
…
if (i==0)
    a=7;
else
    b=15;
    a=b+1;

printf("%d",a);
```

Aufg. 2

```
int i=0,b=10,a;
…
if (i=0)
    a=10;
else if (i=2){
    b=15;
    a=b+1;
}
else a=0;

printf("%d",a);
```

Aufg. 3

```
int i=1,a=5,b=10;
…
if (i==0);
{
    a=10;
    b=a+1;
}
printf("%d",a);
```

ÜBUNG: if-else-Anweisung - Wahl der richtigen Bedingung

(Achtung: C-Puzzles, ganz schlechter Code)

1. Vereinfachen Sie:

```
if (a){
   if(b)
      if(c) D;
}else
   if(b)
      if(c) E;
      else  F;
   else;
```

2. Was wird gedruckt:

```
x = -1;
y = -1;

if (x>0)
    if(y>0)
        print("1");
else
    print("2");
```

Anm.: a,b,c sind beliebige expressions
D,E,F sind beliebige statements

13.6.3 Switch-Anweisung

Diese Kontrollstruktur ist als Mehrfachverzweigung oder Verteiler gedacht.

'switch' '(' *expression* ')'
{ [**'case'** constant-value ':' [*statement*]... [**'break'**]] ...
[**'default'** : [*statement*] ...]
}

Beispiel:

```
switch ( character )
{
   case 'a' :
   case 'A' : alpha = 'a';  break ;
        ....
   case 'z' :
   case 'Z' : alpha = 'z';  break ;

   default : alpha = '0';
}
```

Achtung: Das *break* ist notwendig, damit nicht die folgenden Fälle abgearbeitet werden!

ÜBUNGEN: Switch-Anweisung

1. Was wird ausgegeben?

```
int in=2;
...
switch(in){
        case 1:
        case 2:
        case 4:
        case 6:
                printf("A");
        case 9:
                printf("u");
                printf("t");
        case 10:
                printf("o");
                break;
         case 11:
                printf("ma");
        default:
                printf("t \n");
}
```

2. Geben Sie zu folgendem Ausdruck eine äquivalente switch-Anweisung an:

```
if ((num<=8) && (num>=5))
        printf("schlecht \n");

else if ((num==3) || (num==4))
        printf("mittel \n");

else if ((num<=2) && (num>0))
        printf("gut \n");

else
        printf("unmoeglich \n");
```

13.6.4 WHILE-Anweisung (kopfgesteuerte Schleife)

Syntx:

'**while**' '(' *expression* ')' *statement*

Die Anweisung wird ausgeführt solange der Prüfausdruck wahr ist (d.h. **!= 0**).

Beispiele:

```
while( a < b )
      a = a + 2;

while( y<=10 ){
      x=x+1;
      y=y+1;
}
```

13.6.5 DO-WHILE-Anweisung (fußgesteuerte Schleife)

Syntax:

'do' *statement* **'while'** '(' *expression* ')' ;

Die Anweisung wird ausgeführt solange der Prüfausdruck wahr ist (!= 0).

Beispiel:

```
do{
      summe = summe +1;
      i = i + 1;
}while (i <= 100);
```

13.6.6 FOR-Anweisung

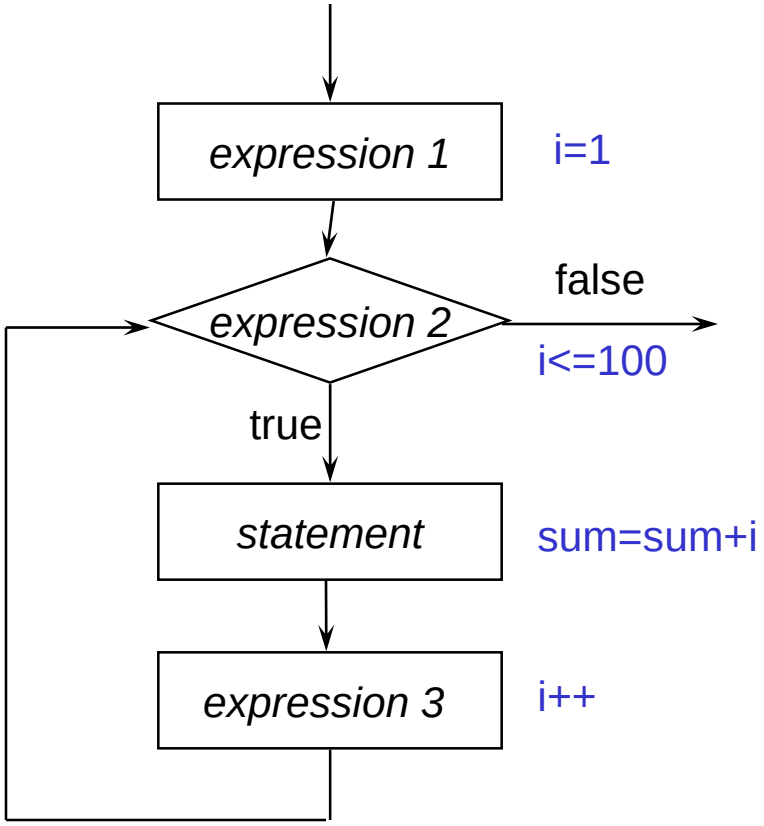
'for' '(' [*expression 1*]; [*expression 2*]; [*expression* 3] ')' *statement*

Üblicherweise:

expression 1: Initialisierung

expression 2: Bedingung

expression 3: Incrementierung / Decrementierung



Beispiel:

```
sum=0;
for (i=1; i<=100; i++)
   {
     sum = sum + i;
   }
```

ÜBUNGEN: for-Schleife

1. Schreiben Sie ein Programm, welches alle Zahlen des "2-aus-5-Code" tabellarisch ausgibt, also:

```
00011
00101
00110
...
11000
```

2. Was wird ausgegeben?

```
char ch2[]="Irgendein Text.";
char ch1[]="Irgendein Text.";
int  Len,i,k;

Len=strlen(ch1); /* liefert die Stringlaenge */
for(i=0, k=Len-1;  i<Len;  i++,k--){
        ch2[k] = ch1[i] ;
}
printf("%s \n", ch2);
```