Aufgabe 23: Speicherklassen

Inhalt

23.1.1 Ausgabe des Programms block.c

Was gibt das folgende Programm *block.c* aus?

```c
#include  <stdio.h>

int i=0;

int  main(void)
{
    int i=1;

    printf("i=%d\n", i);
    {
       int i=2;
       printf("i=%d\n", i);
       {
          i++;
          printf("i=%d\n", i);
       }
       printf("i=%d\n", i);
    }
    printf("i=%d\n", i);
    return(0);
}
```

Antwort:

i =1

Es gibt eine globale Variable i, die mit 0 initialisiert wird. Innerhalb der main-Funktion gibt es eine lokale Variable i, die mit 1 initialisiert wird. Diese lokale Variable i überschreibt die globale Variable i innerhalb des Gültigkeitsbereichs der main-Funktion.

23.2.1 Ausgabe des Programms speikla1.c

Was gibt das folgende Programm *speikla1.c* aus?

```
#include  <stdio.h>

int  i=0;

int setze1(int x);
int setze2(int x);

int  main(void)
{
  auto int  i=5;
   setze1(i/2);     printf("i = %d\n", i);
   setze1(i=i/2);   printf("i = %d\n", i);
   i = setze1(i/2); printf("i = %d\n", i);
   setze2(i);       printf("i = %d\n", i);
   return(0);
}

int setze1(int i)
{
   i = i<=2 ? 5 : 0;
   return(i);
}

int setze2(int i)
{
   i = i%i * (i*i/(2*i)+4);    printf("i = %d\n", i);
   return(i);
}
```

Antwort:

i = 5
i = 2
i = 5
i = 0
i = 5

1. setze1(i/2); printf("i = %d\n", i);
i/2 berechnet sich zu 5/2 = 2 (ganzzahlige Division) → setze1(2) wird aufgerufen, setze1 prüft, ob i <= 2 ist (was wahr ist, da 2 <= 2) → Daher wird i auf 5 gesetzt und von setze1 zurückgegeben → Die Ausgabe zeigt den aktuellen Wert von i, der immer noch 5 ist.

2. setze1(i=i/2); printf("i = %d\n", i);
i=i/2 weist i den Wert von 2 zu (nach der Berechnung 5/2) → setze1(2) wird erneut aufgerufen, setze1 setzt i wieder auf 5 → Die Ausgabe zeigt den aktuellen Wert von i, der jetzt 2 ist.

3. i = setze1(i/2); printf("i = %d\n", i);
i/2 berechnet sich zu 2/2 = 1 → setze1(1) wird aufgerufen, setze1 setzt i wieder auf 5 → Die Zuweisung i = setze1(i/2); setzt i auf den Rückgabewert von setze1(1), also 5 → Die Ausgabe zeigt den aktuellen Wert von i, der wieder 5 ist.

4. setze2(i); printf("i = %d\n", i);
setze5(5) wird aufgerufen → Die operation i = i%i * (i*i/(2*i)+4); berechnet sich zu 0, da % eine hohe Priorität hat als *, d.h. 5 % 5 = 0 → printf in setze2 gibt 0 als Ausgabe → i hat immer den Wert5 in main, d.h. printf in main gibt wieder 5 als Ausgabe.

23.2.2 Ausgabe des Programms speikla2.c

Was gibt das folgende Programm *speikla2.c* aus?

```
#include  <stdio.h>

int  i=1;

int setze(void);
int naechst(int x);
int letzt(int x);
int neu(int x);

int  main(void)
{
  auto int  i, j;
   i = setze();
   for (j=1; j<3; j++) {
      printf("i=%d, j=%d\n", i, j);
      printf("   naechst(i)=%d\n", naechst(i));
      printf("   letzt(i)=%d\n", letzt(i));
      printf("   neu(i+j)=%d\n", neu(i+j));
   }
   return(0);
}

int setze(void)
{
   return(i);
}

int naechst(int j)
{
   return(j-i++);
}

int letzt(int j)
{
   static int i=10;
   return(j-i--);
}

int neu(int i)
{
   auto int j=10;
   return(i-j+-i);
}
```

Antwort:

i=1, j=1
 naechst(i)=1
 letzt(i)=10
 neu(i+j)=12
i=1, j=2
 naechst(i)=2
 letzt(i)=9
 neu(i+j)=13

In main:

i = setze() → setzt die lokale Variable i auf den Wert der globalen Variable i, also 1.
Eine Schleife for (j=1; j<3; j++) wird gestartet, wobei j zuerst 1 und dann 2 ist.

Innerhalb der Schleife für jede Iteration:

- printf("i=%d, j=%d\n", i, j) → gibt die aktuellen Werte von i (global) und j aus. i=1, j=1

- printf(" naechst(i)=%d\n", naechst(i)) → Ruft naechst(i) auf, wo i als Argument übergeben wird → In naechst(int j): j = i++; weist j den aktuellen Wert von i zu und inkrementiert dann i → Die Funktion gibt j zurück, also den vorherigen Wert von i (bevor es inkrementiert wurde) → naechst(i)=1

- printf(" letzt(i)=%d\n", letzt(i)) → Ruft letzt(i) auf, wo i als Argument übergeben wird → In letzt(int j): j = i--; weist j den aktuellen Wert von i zu und dekrementiert dann i → Da i statisch ist und initialisiert wurde, behält es seinen Wert zwischen den Funktionsaufrufen bei → Die Funktion gibt j zurück, also den vorherigen Wert von i (bevor es dekrementiert wurde) → letzt(i)=10

- printf(" neu(i+j)=%d\n", neu(i+j)) → Ruft neu(i+j) auf, wobei i+j als Argument übergeben wird → In neu(int i): j ist eine lokale Variable, die mit 10 initialisiert wird → j += i; addiert i zu j und weist das Ergebnis zurück zu j → Die Funktion gibt j zurück, also das Ergebnis der Addition von i und j → neu(i+j)=12

Die Schleife wird zweimal durchlaufen, daher werden die oben genannten Ausgaben zweimal für j=1 und j=2 wiederholt.

23.2.3 Ausgabe des Programms speikla3.c

Was gibt das folgende Programm aus, das sich aus den drei Modulen *speikla3.c*, *modulb.c* und *modulc.c* zusammensetzt?

Modul *speikla3.c*:

```
#include  <stdio.h>

int  i=1;

extern int setze(void);
extern int naechst(void);
extern int letzt(void);
extern int neu(int x);

int  main(void)
{
   auto int  i, j;

   i = setze();
   for (j=1; j<3; j++) {
      printf("i=%d, j=%d\n", i, j);
      printf("   naechst()=%d\n", naechst());
      printf("   letzt()=%d\n", letzt());
      printf("   neu(i+j)=%d\n", neu(i+j));
   }
   return(0);
}
```

Modul *modulb.c*:

```
static int i=10;

int naechst(void) { return(i+=1); }
int letzt(void)   { return(i--1); }

int neu(int i)
{
   static int j=5;
   return(i-j+=i);
}
```

Modul *modulc.c*:

```
extern int i;

int setze(void)
{
   return(i);
}
```

Antwort:

i=1, j=1
 naechst()=11
 letzt()=10
 neu(i+j)=7
i=1, j=2
 naechst()=11
 letzt()=10
 neu(i+j)=10

In main:

i = setze(); setzt die lokale Variable i auf den Wert der globalen Variable i aus modulc.c, also 1.
Es wird eine Schleife für j von 1 bis 2 durchlaufen.

Innerhalb der Schleife für jede Iteration:

- printf("i=%d, j=%d\n", i, j); gibt die aktuellen Werte von i und j aus. (i=1, j=1)
- printf(" naechst()=%d\n", naechst()); ruft naechst() aus modulb.c auf, erhöht i um 1 (von 10 auf 11) und gibt den neuen Wert 11 zurück.
- printf(" letzt()=%d\n", letzt()); ruft letzt() aus modulb.c auf, verringert i um 1 (von 11 auf 10) und gibt den neuen Wert 10 zurück.
- printf(" neu(i+j)=%d\n", neu(i+j)); ruft neu() aus modulb.c auf, erhöht j um i+j (hier 1+1=2 für die erste Iteration und 1+2=3 für die zweite Iteration) und gibt den neuen Wert zurück (2 +5 = 7).

Die Schleife wird zweimal durchlaufen, daher werden die oben genannten Ausgaben zweimal für j=1 und j=2 wiederholt.

23.3.1 Konstante Zeiger und Zeiger auf Konstanten

Welche Anweisungen im folgenden Programm *constzei.c* sind erlaubt und welche nicht?

```
int  main(void) {
    const double pi=3.14;
    double *dz;
    int var=100;
    int *const cz=&var;
    const int* zc=&var;
    const int* const czc=&var;

    pi = 6.2;
    dz = &pi;
    *dz = 6.2;

    *cz = 50;
    cz = zc;
    *zc = 500;
    zc = cz;
    *czc = 5000;
    czc = zc;
   return 0;
}
```

Antwort:

Nicht erlaubte Anweisungen:

- pi = 6.2; ist nicht erlaubt, da pi als const double deklariert ist und nicht verändert werden kann.
- dz = π und *dz = 6.2; nicht erlaubt, da dz ein Zeiger auf double ist, aber pi ist konstant und kann nicht durch dz geändert werden.

- *zc = 500; ist nicht erlaubt, da zc ein Zeiger auf eine konstante int ist und nicht verwendet werden kann, um den Wert von var zu ändern.
- cz = zc; ist nicht erlaubt, da cz ein konstanter Zeiger auf int ist und nicht auf ein anderes Ziel zeigen kann.
- *czc = 5000; ist nicht erlaubt, da czc ein konstanter Zeiger auf eine konstante int ist und sowohl die Variable var als auch der Zeiger czc nicht geändert werden können.
- czc = zc; ist nicht erlaubt, da czc ein konstanter Zeiger auf eine konstante int ist und nicht auf einen anderen Zeiger zeigen kann.

23.3.2 const-Parameter bei Funktionsdefinitionen

Welche Anweisung im folgenden Programm *consfunk.c* ist nicht erlaubt und was würde dieses Programm ausgeben, wenn diese Anweisung entfernt wird?

```
double  wurz2(const double *arg);
double  wurz3(      double *arg);

int  main(void) {
         double  z1=10.0, z2;
   const double  z3=8.0;

   z2 = wurz2(&z3); printf("%f\n",z2);
   z2 = wurz2(&z1); printf("%f\n",z2);
   z2 = wurz3(&z3); printf("%f\n",z2);
   z2 = wurz3(&z1); printf("%f\n\n",z2);
   printf("%f\n%f\n", z1, z3);
   return(0);
}
double wurz2(const double *arg) {
   *arg = 27.0;
   return(pow(*arg,0.5));
}
double wurz3(double *arg) {
   *arg = 32.0;
   return(pow(*arg,1/3.0));
}
```

Antwort:

1. Nicht erlaubte Anweisungen:

z2 = wurz2(&z3); ist nicht erlaubt, da wurz2 erwartet, dass arg ein const double* ist, während z3 als const double deklariert ist. Das Ändern von *arg ist daher nicht erlaubt.

2. Erlaubte Anweisungen:

z2 = wurz2(&z1); ist erlaubt, da z1 eine normale double-Variable ist und somit &z1 vom Typ double* ist, was mit der Signatur der Funktion wurz2(const double *arg) übereinstimmt. Die Funktion wurz2 kann arg lesen, aber nicht ändern.

z2 = wurz3(&z3); und z2 = wurz3(&z1); sind beide erlaubt, da wurz3 erwartet, dass arg ein double* ist, was sowohl mit &z3 als auch &z1 kompatibel ist. In beiden Fällen kann arg geändert werden, da arg als double* deklariert ist.

Ausgabe:

3.162278
3.174802
3.174802

32.000000
32.000000
