4 Funktionen

4.1 Funktionen und ihre Argumente

Funktionen:

***typ fname*(*typ1 var1, ..., typn varn*)**{
    statements;
} es wird eine Funktion *fname* mit den Argumenten *var1, ..., varn* vom jeweiligen Typ *typ1, ... typn* definiert. Der von der Funktion zurückgegebene Wert ist vom Typ *typ*. In den Programmen, die *fname* aufrufen, muß *fname* in der Form *typ fname (typ1, ..., typn)* deklariert werden.

In diesem Sinne ist auch `main` eine Funktion. Variablen, die innerhalb einer Funktion vereinbart wurden (interne Variablen), sind nur lokal bekannt. Diese Variablen haben undefinierte (sinnlose) Werte. Vereinbarungen außerhalb einer Funktion sind global (extern) und können von allen Funktionen benutzt und verändert werden. Externe Variablen haben zu Beginn den Wert 0.

Rücksprung:

return *value*; Funktionsende, es wird *value* zurückgegeben. Rücksprünge können an verschiedenen Stellen erfolgen.

Funktionen, die keinen Wert zurückgeben, sind als `void` zu erklären. Sie benötigen demnach auch kein `return`.

Listing 13: Noreturn (noreturn.c)

```
#include <stdio.h>

int main(void) {
        void noreturn(void);
        noreturn();
        return 0; }

void noreturn(void)
{
        printf("Function of no return\n");
}
```

Die lokalen Variablen in einer Funktion werden bei jedem Aufruf neu initialisiert. Man kann dies verhindern, indem die Variablen als `static` vereinbart werden.

Listing 14: Zähler (counter.c)

```
#include <stdio.h>

int main(void)
{
        int i;
        void counter(void);
        void falsecounter(void);
        for (i = 0; i <= 10; i++) {
                counter();
                falsecounter();
        }
        return 0;
}

void counter(void)
{
        /*  c wird nur beim ersten
         *      Aufruf auf 0 gesetzt */
        static int c = 0;
        printf("counter: %d \t", ++c);
}

void falsecounter(void)
{
        /* c wird bei jedem Aufruf auf 0 gesetzt */
        int c = 0;
        printf(" falsecounter: %d\n", ++c);
}
```

Die beiden Typenerklärungen für die Funktionen *counter* und *fcounter* in `main` können fortgelassen werden, wenn man das Hauptprogramm hinter die beiden Funktionen schreibt.

4.2 Rekursive Funktionen

Funktionen können sich selbst aufrufen. Bei jedem rekursiven Aufruf werden alle lokalen Variablen durch neue ersetzt. Nicht jedes rekursive Programm ist sinnvoll.

Listing 15: Fibonacci-Zahlen (fibo.c)

```
#include <stdio.h>

/* fuer n >= 33 stellt man Unterschiede
 * in der Laufzeit fest! */

int main(void) {
        int n, fibonacci(int), fib(int);
        printf("Berechnung einer Fibonnacci-Zahl\n");
        printf("Eingabe n (0 < n < 44): ");
        (void) scanf("%i", &n);
        printf(" f( %i ) = %i\n", n, fibonacci(n));
        printf(" f( %i ) = %i\n", n, fib(n));
        return 0; }

int fibonacci(int n) {
        int i, f0 = 0, f1 = 1, f2=0;
        for (i = 1; i <= n; i++) {
                f2 = f1 + f0;
                f0 = f1;
                f1 = f2;
        }
        return f2; }

int fib(int n)
{
        if (n == 0 || n == 1) {
                return 1;
        } else {
                return (fib(n - 1) + fib(n - 2));
        }
}
```

4.3 Adressen und Zeiger

Variablen, die eine Speicherplatzadresse aufnehmen sollen, nennt man Zeiger oder pointer.

Zeiger (Pointer) ist eine Variable, die eine Adresse enthält. Die Variable muß denselben Typ wie die Variable haben, auf die sie verweist.

typ *var vereinbart einen Zeiger *var*, der auf eine Variable vom Typ *typ* verweist.

Adreßoperator:

&var Bestimmt die Adresse des ersten Speicherplatzes von *var*.

Verweisoperator:

****var*** Bestimmt den Inhalt der in *var* angegebenen Adresse.

Listing 16: Adressen und Verweise (adressen.c)

```
#include <stdio.h>

int main(void)
{
        int var = 12345, *pvar, var1;
        pvar = &var;
        var1 = *pvar;
        printf("Wert: %d, Adresse: %x\n",
                        var, (int) pvar);
        printf("Inhalt: %d von Adresse %x\n",
                        var1, (int) &var1);
        return 0;
}
```

Adresse	Wert
pvar= &var	*var*
ffbfed08	12345
&var1	*var1* = **pvar*
ffbfed04	12345

4.4 Call by value

In C werden die Argumente einer Funktion als Wert übergeben, deshalb bleiben die Argumente unverändert. Wenn Zeiger übergeben werden, können die durch sie bestimmten Adressen neu belegt werden.

Listing 17: Swapping (swapping.c)

```
#include <stdio.h>

static void falseswap(int,int), swap(int*,int*);

void falseswap(int x, int y)
{ /* call by value */
        int tmp;
        tmp = x;
        x = y;
        y = tmp;
}

void swap(int *px, int *py)
{ /* call by reference */
        int tmp;
        tmp = *px;
        *px = *py;
        *py = tmp;
}

int main(void)
{
        int x = 1949, y = 1951;
        printf("x = %i, y = %i \n", x, y);
        falseswap(x, y);
        printf("nach falseswap\n x = %i, y = %i \n",
                                x, y);
        swap(&x, &y);
        printf("nach swap\n x = %i, y = %i \n", x, y);
        return 0;
}
```