

Die Programmiersprache C

von Michael Wellner

1. Grundlagen

1.1. Der allgemeine Aufbau eines C-Programms

Jedes C-Programm hat eine Hauptfunktion, die Funktion "main". Hinter dem Namen einer Funktion bzw. Prozedur müssen immer zwei Klammern ("()") stehen. In den Klammern werden gegebenenfalls Variablen/ Werte übergeben. Danach kommt der Anweisungsblock der Prozedur, dieser ist von 2 geschweiften Klammern umgeben.

Da man natürlich Programme schreiben kann, die keinen Anweisungsblock enthalten ist das folgende Beispiel das kleinstmögliche C-Programm.

```
main () {}
```

Um die Übersichtlichkeit zu gewährleisten wird meist folgende Schreibweise verwendet:

```
main()
{
    // Anweisungsblock
}
```

Wenn man eines dieser ersten beiden Beispiele kompilieren und ausführen würde, würde man kaum etwas von dem Programm bemerken, denn sobald man es gestartet hat ist es beendet. Um dieser Tatsache ein Ende zu bereiten schreiben wir folgendes kleines Programm:

```
main()
{
    getch ();
}
```

Die Funktion "getch ()" (get character) wartet auf eine Eingabe des Users, danach wird das Programm normal weiter ausgeführt.

Zu beachten ist, dass in C jede Anweisung mit einem Semikolon endet!

1.2 Ein- und Ausgabefunktionen zum Ersten!

Ein- und Ausgabefunktionen sind Funktionen, die den Dialog zwischen dem Computer und dessen User herstellen.

Hierzu zuerst ein kleines Beispiel:

```
main ()
{
    printf ("Eine Taste druecken...!\n\n");
    getch ();
}
```

Beim Ablauf des Programms wird "Eine Taste druecken...!" auf dem Bildschirm ausgegeben. Der Cursor sitzt zwei Zeilen weiter unten.

Wir stellen also fest, dass die Funktion "printf" Text auf dem Bildschirm ausgibt. Aber was hat es mit dem "n" hinter dem Backslash auf sich? - Zeichen die hinter dem Backslash stehen sind Steuerzeichen. Es gibt folgende

\n	nächste Zeile
\t	Tabulator
\a	Piepston
\b	ein Zeichen zurück

Wir können mit printf also Daten ausgeben. Um Daten auszugeben müssen wir uns erst etwas um die Initialisierung von Variablen kümmern.

1.3 Initialisierung von Variablen

Um eine Eingabe vom Benutzer heran zu holen muss vorher vereinbart werden, um welchen Datentyp es sich bei der Eingabe handelt.

Hier sind erst einmal die wichtigsten Datentypen aufgelistet. Wozu wir die Steuersequenz brauchen werden wir im nächsten Beispiel sehen.

<u>Initialisierung</u>	<u>Steuersequenz</u>	<u>Bedeutung</u>
char	%s	String (Zeichenkette)
char	%c	Character (einzelnes Zeichen)
int	%d	Integer (ganze Zahlen)
float	%f	Floating Rates (Dezimalzahlen)

1.4 Ein- und Ausgabefunktionen zum Zweiten!

In einem kleinen Beispiel soll die Anwendung der Steuersequenz gezeigt werden, dabei werden wir auch eine C-Standard-Eingabefunktion kennen lernen.

```
main ()
{
    char Text;
    printf ("Text eingeben:\n");
    scanf ("%s", Text);
    getch();
}
```

Da der einzugebende Text eine Zeichenkette ist, wird das Steuerzeichen "%s" eingegeben und am Anfang des Anweisungsblocks wird "Text" als character mit "char" initialisiert.

2. Operatoren

Ich will an dieser Stelle nur grundlegendes zu Operatoren sagen bzw. schreiben. Operatoren sind eigentlich sehr komplex und man könnte viel zu ihnen schreiben. Da Operatoren aber viel mit Theorie zu tun haben und oftmals auch kompliziert sind, gehe ich nur auf einfache und grundlegende Dinge ein.

2.1 Das logische UND und das logische ODER

Es kommt in Algorithmen oft vor, dass einige Operationen nur ausgeführt werden sollen, wenn zwei oder eine von 2 bestimmten Bedingungen erfüllt werden.

Wenn man will, dass zwei Bedingungen erfüllt sein müssen muss man die Bedingungen mit dem logischen UND verknüpfen, das sieht in C dann so aus:

```
if (Bedingung1 && Bedingung2)
{
    /* Anweisungen;
}
```

Soll ein Anweisungsblock abgearbeitet werden, wenn mindestens eine von zwei (oder auch mehr) Bedingungen erfüllt sind, verwendet man den logischen ODER-Operator.

```
if (Bedingung1 || Bedingung2)
{
    /* Anweisungen
}
```

2.2 Die Operatoren der Grundrechenarten

Um einer Variable einen Wert zuzuweisen braucht man meist Rechnungen, da das recht einfach ist will ich das nur an ein paar Beispielen zeigen:

```
a = a + 12.2    /* Addieren
a = a - b      /* Subtrahieren
a = 2 * 4      /* Multiplizieren
a = 12 / 3     /* Dividieren
```

3. Schleifen

3.1. If und If...else

If (engl. = falls) führt einen Anweisungsblock aus wenn eine Bedingung erfüllt wird. Der Syntax sieht folgendermaßen aus:

```
if (Bedingung) {
/* Anweisungsblock
}
```

Wenn ein anderer Anweisungsblock ausgeführt werden soll, falls die if-Bedingung nicht erfüllt wurde, wendet man dazu else an:

```
if (Bedingung) {
/* Anweisungsblock
}
if (Bedingung) {
/* Anweisungsblock
}
else {
/* Anweisungsblock
}
```

Wie man in diesem Beispiel sehen kann, kann man auch mehrere if-Bedingungen in eine If Schleife schreiben.