

C-Zusammenfassung

Kommentar

Ein Kommentar wird beim Übersetzen nicht beachtet. Ein Kommentar beginnt `/*` und endet mit `*/`. Er kann sich über mehrere Zeilen erstrecken, darf aber nicht selbst wieder einen Kommentar enthalten.

<code>#include <Dateiname></code>	Präprozessordirektive zur Einbindung einer Informationsdatei, die es ermöglicht, bestimmte Funktionen dieser Datei im Programm zu verwenden, z. B. printf() und scanf () .
<code>main()</code>	Beginn des Hauptprogramms.
<code>return <Wert></code>	Gibt einen Wert zurück (ANSI-C; ist für die Funktion der Datentyp void vorgesehen, so kann auf return verzichtet werden).
<code>{ und }</code>	Begrenzungen eines Programmblocks.
<code>;</code>	Abschließendes Zeichen für Deklarations-, Definitions- und Anweisungszeilen.

printf – Sonderzeichen

Funktion zur Ausgabe von Text und Variablen (standardmäßig am Bildschirm).

<code>\n</code>	Zeilenvorschub , entspricht RETURN bei der Tastatureingabe
<code>\t</code>	Tabulator , auf dem Bildschirm wird ein Tabulator [TAB] - Sprung gemacht
<code>\r</code>	Carriage Return , unter UNIX irrelevant. Unter DOS besteht ein Zeilenende aus <code>\n\r</code>
<code>\f</code>	Irrelevant für Bildschirm und Tastatur, bewirkt unter einer Druckerausgabe einen Seitenvorschub (formfeed)
<code>\0</code>	NULL , wird intern für eine String-Endmarkierung verwendet. Wird manchmal auch bei Druckersequenzen benutzt.
<code>\\</code>	Backlash , erzeugt ein <code>\</code> auf dem Bildschirm
<code>\'</code>	Erzeugt ein Hochkomma auf dem Bildschirm
<code>\"</code>	Erzeugt Anführungszeichen, die ja sonst schwerlich darstellbar sind

scanf Funktion zur Eingabe von Variablen (standardmäßig über Tastatur)

Formatelemente in C (in scanf() und printf() verwendbar):

Element:	Bedeutung:
<code>%d</code>	vorzeichenbehaftete Ganzzahlen (dezimal)
<code>%i</code>	vorzeichenbehaftete Ganzzahlen (dezimal)
<code>%u</code>	vorzeichenlose Ganzzahlen (dezimal)
<code>%o</code>	vorzeichenlose Ganzzahlen (oktal)
<code>%x</code>	vorzeichenlose Ganzzahlen (hexadezimal)
<code>%f</code>	Fließkommazahl %e Fließkomazahl (exponentiell)
<code>%c</code>	ein Zeichen
<code>%s</code>	eine Zeichenkette

Grundlegende Variablentypen

int	[integer] Ganzzahlen (Zahlen ohne Nachkommaanteil) z.B. 2 / -3 / 5 / u.s.w.
char	[character] ein Zeichen z.B. a / k / z , aber auch die RETURN-Taste
float	[floating point] Fließkommazahlen (Zahlen mit Nachkommaanteil) z.B. 3.23 / 4.32 / 3.01
double	Fließkommazahlen mit doppelt so großem Wertebereich wie der Float-Typ.

Hier nun die Wertebereiche der Variablentypen:

int	- 32 768 ... 32 767 (16 Bit Maschinen) - 2 147 483 648 ... 2 147 483 647 (32 Bit Maschinen)
unsigned int	0 ... 65 535 (16 Bit Maschinen) 0 ... 4 294 967 295 (32 Bit Maschinen)
short int	- 32 768 ... 32 767
long int	- 2 147 483 648 ... 2 147 483 647
unsigned short int	0 ... 65 535
unsigned long int	0... 4 294 967 295
char	alle ASCII-Zeichen , das sind alle Zeichen, die man auf dem Bildschirm darstellen kann
unsigned char	alle ASCII-Zeichen
float	float -10^{38} ... 10^{38}
double	-10^{308} ... 10^{308}

Größe der C-Standardtypen

Datentyp	Anzahl in Bytes
char	1
short	2
int	2
long	4
float	4
double	8

Bedingungen / Vergleiche

Möchte man feststellen, ob eine Variable einer bestimmten Bedingung genügt, so muss man einen Vergleich anstellen. In der folgenden Tabelle wird auch angegeben, welche Bedingung erfüllt sein muss, damit ein Ausdruck als wahr interpretiert wird.

==	gleich beide Ausdrücke sind gleich
!=	Ungleich/ nicht gleich beide Ausdrücke sind nicht gleich
>	größer der erste Ausdruck ist größer als der zweite

>=	größer oder gleich der erste Ausdruck ist größer oder gleich dem zweiten
<	kleiner der erste Ausdruck ist kleiner als der zweite
<=	kleiner oder gleich der erste Ausdruck ist kleiner oder gleich dem zweiten
&&	logisches und beide Ausdrücke müssen wahr sein
	logisches oder einer der Ausdrücke muss wahr sein
!	Negation der folgende Ausdruck muss falsch sein

Rangfolge von Operatoren

< , <= , > , >=	hat Vorrang vor	== , !=	hat Vorrang vor	&& ,
-----------------	-----------------	---------	-----------------	------

If Bedingung (Anweisung)

Trifft die Bedingung zu, so wird die Anweisung ausgeführt. Es darf im Ja.Zweig nur eine Aussage stehen. Sollen dort mehrere ausgeführt werden, müssen sie zu einer Verbundanweisung zusammengefasst werden

```
if (Bedingung)
{
Anweisung;
}
```

Zweifach- Auswahl

Ist die Bedingung erfüllt, wird Anweisung 1 ausgeführt, sonst Anweisung 2.

```
if (Bedingung)
{
Anweisung 1;
}
else
{
Anweisung 2;
}
```

Fallunterscheidung

```
switch
{
case Fall1 : Anweisung1;
break;
case Fall2 : Anweisung2;
break;
case Fall3 : Anweisung3;
break;
default: Anweisung;
```

```
}
```

default nur wenn kein case zutrifft

break:

Der Befehl dient zum Verlassen des Schleifenkörpers, in dem break ausgeführt wird, bzw. der switch-Anweisung

while-Schleife (Kopfgesteuerte Schleife)

Hinter der Bedingung kann nur eine Anweisung formuliert werden. Diese Anweisung wird solange wiederholt, wie die Bedingung zutrifft. Die Anweisung nach der Bedingung kann allerdings eine Verbandanweisung sein. Mehrere Anweisungen können mit Hilfe der geschweiften Klammer zu einem Block zusammen gefasst werden (Verbandanweisung)

Die Bedingung sollte sich innerhalb des Anweisungsblocks auf der while-Schleife verändern, d. h. die Schleife muß die Möglichkeit einer Beendigung haben.

```
while (Bedingung)
{
  Anweisung1;
  Anweisung2;
}
```

do-while Schleife (fußgesteuerte Schleife)

Die Bedingung sollte sich innerhalb des Anweisungsblocks auf der while-Schleife verändern, d. h. die Schleife muß die Möglichkeit einer Beendigung haben.

Die do-while-Schleife wird mindestens einmal durchlaufen, hier wichtig das Semikolon
Der Schleifenrumpf wird wiederholt, wenn die Bedingung erfüllt ist.

```
do
{
  Anweisungsblock;
}
while (Bedingung);
```

for-Schleife (Zählschleife)

Bei einer for-Schleife wird zuerst die Initialisierung abgearbeitet. Solange die Bedingung wahr ist, wird der Schleifenkörper und anschließend die Anweisung ausgeführt. Danach wird wieder die Bedingung geprüft, und so weiter.

```
for (Variable = Anfangswert; Variablenzählgrenzbedingung; Schrittweite/Zählrichtung)
{
  Anweisungsblock;
}
```

Der Umwandlungsoperator

Bsp.:

```
int i;
```

```
float f;
```

```
i = (int) f;
```

Der Umwandlungsoperator (cast-operator) wandelt den Wert von f in ganzzahlig (int).

Nachkommastellen werden abgeschnitten, f selbst bleibt float.

Typedef

Typedef dient zur Vereinfachung. Es wird kein neuer Datentyp geschaffen, sondern nur ein Synonym für einen bestehenden Datentyp.

Schlüsselwort	Typbeschreibung	Name des Types
typedef	unsigned short	USHORT
typedef	unsigned char	BYTE
typedef	unsigned int	WORD
typedef	int	BOOL
typedef	unsigned long	DWORD

Syntax:

```
typedef unsigned short USHORT;
```

Strukturen, Datensätze, Records: benutzerdefinierte Datentypen

Datenoperationen (der folgende Teil stammt von einem Kollegen)

Datenkanal öffnen

```
int.variable = open("name", option, rechte);
```

int.variable	Integervariable die als Filediskriptor verwendet wird
name	Name der Datei auf die der Datenkanal verweisen soll
option	Was wird auf dem Datenkanal gemacht O_RDONLY oder 0 nur lesen O_WRONLY oder 1 nur schreiben O_RDWR oder 2 lesen und schreiben O_CREAT oder 100 erstellen O_AP anhängen O_CREAT+RDWR eine nicht vorhandene Datei wird erstellt Beispiel: lfd = open ("datei", O_RDWR, 0666); oder lfd = open ("datei", 2, 0666);
rechte	0xxx die Null wird als FLAG für das Oktalformat benutzt. xxx vergibt die Rechte für die Datei Bei Lesevorgängen kann die Angabe der Rechte entfallen.

Daten auf den Datenkanal schreiben

int.rückgabevariable = write (Filediskriptor, variable, anzahl der Bytes);

int.rückgabevariable	integer Variable in die der Rückgabewert der Routine geliefert wird. Entweder die Zahl der geschriebenen Bytes oder -1 für Fehler
Filediskriptor	siehe Daten Kanal öffnen
Variable	Variable, deren Inhalt gespeichert werden soll
Anzahl der Bytes	Wie viele Bytes sollen geschrieben werden Der Datenzeiger bestimmt den Anfangspunkt, ab dem geschrieben wird (siehe Datenzeiger setzen)

Daten auf dem Datenkanal lesen

int.rückgabevariable = read (Filediskriptor, variable, anzahl der Bytes);

int.rückgabevariable	integer Variable in die der Rückgabewert der Routine geliefert wird. Entweder die Zahl der gelesenen Bytes oder -1 für Fehler
Filediskriptor	siehe Daten Kanal öffnen
Variable	Variable, deren Inhalt durch die gelesenen Bytes ersetzt werden soll
Anzahl der Bytes	Wie viele Bytes sollen gelesen werden Der Datenzeiger bestimmt den Anfangspunkt, ab dem geschrieben wird (siehe Datenzeiger setzen)

Datenzeiger setzen

Iseek (filediskriptor, ipos, Startpunkt);

Filediskriptor	siehe Daten Kanal öffnen
ipos	integer Wert, der mit dem Startwert zusammen die Position des Datenzeigers ergibt
Startpunkt	0 Anfang der Datei 1 Aktuelle Position des Datenzeigers 2 Ende der Datei

Quelle:
InBit, Paderborn