LANDESBERUFSSCHULE 4 SALZBURG

Informatik

Version 2

LBS 4

Dieses Skript dient als zusätzliche Lernunterlage für Informatik

Inhalt

Escapesequenzen:

Diese dienen zum Formatieren von Text sowie zum Anzeigen von reservierten Sonderzeichen.

\a	BEL(bell) akustisches Warnsignal
\b	BS(backspace) setzt Cursor um eine Position nach links
\f	FF(formfeed) ein Seitenvorschub wird ausgelöst. Wird hauptsächlich bei Programmen verwendet mit denen man etwas ausdrucken kann.
\n	NL(newline) Cursor geht zum Anfang der nächsten Zeile
\r	CR(carriage return) Cursor springt zum Anfang der aktuellen Zeile
\t	HT(horizontal tab) Zeilenvorschub zur nächsten horizontalen Tabulatorposition
\v	VT(vertical tab) Cursor springt zur nächsten vertikalen Tabulatorposition
\"	" wird ausgegeben
\'	' wird ausgegeben
\?	? wird ausgegeben
\\	\ wird ausgegeben
\0	NULL (ist die Endmarkierung eines Strings)
\0nn	Ausgabe eines Oktalwertes.(z.B. \033 = ESCAPE-Zeichen)
\xhh	Ausgabe eines Hexdezimalwertes. (z.B. \xff = 255)

Kommentare:

Kommentare werden vom Compiler ignoriert. Guter Programmierstil zeigt sich u.a. in der aussagekräftigen Kommentierung von Programmen.

Beispiele:

```
//Kommentar.c
#include <stdio.h>

int main (){
        int i = 10;             //Variable int mit dem Namen i und Wert 10
        printf("%d",i);         //Gibt die Zahl 10 aus
        printf("\n");           //springt eine Zeile weiter
        printf("10");           //Gibt ebenfalls 10 aus
return 0;
/*Hier sehen sie noch eine 2. Möglichkeit Kommentare einzufügen
Diese Kommentare werden mittels '/'und'*' geöffnet und mit '*'
und '/'
```

Datentypen:

In diesem Kapitel wollen wir die Möglichkeiten kennen lernen, welche Arten von Daten (Nummerische, nichtnummerische, ...) wir verarbeiten können.

Variable und Konstante:

Variable sind Platzhalter für Informationen. Diese können alle möglichen Datentypen beschreiben. Variable können den zugewiesenen Wert verändern.

Konstante sind ebenfalls Platzhalter für Informationen. Werte die in Konstante gespeichert sind können während der Laufzeit des Programmes nicht mehr geändert werden.

Gültige Zeichen für Variable und Konstante sind:

- Unterstrich (_)
- Großbuchstaben (A – Z)
- Kleinbuchstaben (a- z)
- Nummern (0 – 9)

Ungültige Zeichen

- keine Leerzeichen und Komma (, ;)
- keine Sonderzeichen außer Unterstrich
- keine Reservierten Namen wie (int, char, while, for, …..)

Der Name der Variable darf nur mit einem Unterstrich **(_)** oder einem **Buchstaben** beginnen.

Beispiele:

- int _abc // gültiger Name
- char b123 //gültiger Name
- int 1abc //ungültiger Name
- long abc*a //ungültiger Name
- const int num //Konstante

Konstanten und Variablen werden zu Beginn des Programmes, Funktion deklariert.

Datentypen

Es stehen verschiedene Datentypen zur Verfügung wie z.B.:

Ganzzahlen, Fließkommazahlen, Zeichen, aber auch komplexe Datentypen wie

Geburtsdatum, Speiseplan oder Kontostand.

Die wichtigsten Datentypen in C

Name	Größe	Wertebereich	Patzh.
char	1 Byte == 8 Bit	-128...127	%c
unsigned char	1 Byte == 8 Bit	0...255	%c
short	2 Bytes == 16 Bit	-32768...+32767	%d od. %i
unsigned short	2 Bytes == 16 Bit	0...65535	%d od. %i
int	4 Bytes == 32 Bit	-2147483648...+2147483647	%d, %i
unsigned int	4 Bytes == 32 Bit	0...4294967295	%d od. %i
long (32 Bit Syst)	4 Bytes == 32 Bit	-2147483648...+2147483647	%ld, %li
long (64 Bit Syst)	8 Bytes == 64 Bit	+–9.223.372.036.854.755.807	%ld, %li

unsigned long	4 Bytes == 32 Bit	0...4294967295	%ld, %li
float	4 Bytes == 32 Bit	3.4*10-38...3.4*1038	%f
double	8 Bytes == 64 Bit	1.7*10-308...1.7*10308	%lf
long double	10 Bytes == 80 Bit	3.4*10-4932...3.4*104932	%lf
void unbekannter Datentyp wird später besprochen			
enum		Aufzählung	
typedef		Typvereinbarungen	

Beispiele:

```
char ch1= 'A';
char ch2= '+';
char ch3= '\n';                    //Eine Zeilenschaltung: siehe Escapezeichen
char ch4= ' ';                     //ein space
char a = A;                        //falsch, Variablenzuweisung
char b = 65;                       //Schlechter Stil
int a=5;
int b=3, c= a * b;
int zahl1= 27;                     //Wert 27 dezimal
int zahl2= 0x1b;                   //Hex-Zahl
int zahl3= 033;                    //Oktal-Zahl
float kommazahl= -2.345e-001;      //gleichwertig = -0.2345;
long value= 34567L;
const float PI =3.14159;
enum ampel {rot, gelb, gruen};
ampel amp_fussgaenger= rot, amp_strasse= gruen;
typedef int KONTONR;
```

Anmerkung zu dem Datentyp int:

Bei 16 Bit Systemen werden 2 Byte (= 1 CPU WORD) verwendet, um eine Ganzzahl zu repräsentieren.

Bei 32 Bit Systemen werden 4 Byte (= 1 CPU WORD) verwendet, um eine Ganzzahl zu repräsentieren.

Den Wertebereich aller Datentypen finden sie in der Headerdatei <limits.h>

Folgendes Beispiel zeigt welchen Wertebereich der Datentyp **int** bei Ihrem System besitzt.

```
//Size_int.c
#include <stdio.h>
#include <limits.h>
int main()
{
        printf("int hat auf Ihrem System: %li Bytes\n", sizeof(int));
        printf("Der Wertebereich von %d bis %d\n",INT_MIN, INT_MAX);
        return 0;
}
```

Anmerkung zu dem Datentyp char:

Die ASCII - Code - Tabelle ist eine Tabelle an die sich alle Programmierer der Welt halten müssen. Sie enthält alle Zeichen die der Computer darstellen kann. Wobei die Zeichen 0-31 Steuerzeichen sind, die nicht auf dem Bildschirm darstellbar sind.

Der ASCII (American Standard Code for Information Interchange) wurde von US-Ingenieuren entwickelt. Zur damaligen Zeit benutzte man als achtes Bit das Paritätsbit und hatte somit nur noch sieben Bits zur Verfügung. Also Platz für 128 Zeichen und Sonderzeichen.

Jetzt fehlte der Platz für westeuropäische (äöüßÄÖÜ)und slawische Zeichen (von der japanischen Schriftart mit über 40.000 Zeichen ganz zu schweigen und der kyrillischen Schriftart).

Nun war die ISO (International Organisation for Standards) gefragt. Der ASCII – Zeichensatz wurde somit auf 8 Bits erweitert und unter der Bezeichnung ISO-8859-1, ISO-8859-2 etabliert. Der Westeuropäische Standard stellt die ISO-8859-1 da. Damit lassen sich folgende Zeichen darstellen.....

```
//char2.c
#include <stdio.h>
int main(){
  int i;
        for (i=0; i<254; i++){
         printf(" | %d : %c | ",i,i);
       }
return 0;
}
```

Ein und Ausgabe (printf, scanf)

In C verwendet man für die Ein- und Ausgabe bereitgestellte Funktionen. Bevor diese verwendet werden können, muss man allerdings die Datei <stdio.h> inkludieren, denn dort sind die sog. Prototypen (wird später erklärt) der E/A- Funktionen vereinbart:

printf

Zur Ausgabe kann man u.a. die Funktion printf() verwenden.

Der Prototyp ist folgendermaßen definiert:

int printf(const char *format,...);

```
printf("Formatsting %d" , variable );
```

```
//printf1.c
#include <stdio.h>
 int main(){
 int zeichen;
 zeichen=printf("Hallo Welt!"
 printf(" enthaelt %d Zeichen \n",zeichen);
 return 0;
}
```

scanf

Zur Eingabe kann man u.a. die Funktion scanf() verwenden. Siehe Syntax dieser Funktion:

#include <stdio.h>

int scanf(const char *format, ...);

```
//scanf1.c
#include <stdio.h>
int main (){
 int i;
  printf("Bitte geben Sie eine Zahl ein: ");
  scanf("%d",&i);      /*Wartet auf Eingabe durch den Benutzer*/
  printf("Die Zahl die sie eingegeben haben war %d\n",i);
return 0;
}
```

Wie auch bei **printf()** werden hier zwei Klammern und zwei Hochkommata verwendet. Also formatiert eingelesen. Das Formatzeichen **%d** steht für die formatierte Eingabe einer dezimalen Zahl. Was aber bedeutet hier das Zeichen '**&**'.

Eine Variable kann man in vier einzelne Teile aufteilen....

Typ	Name	Adresse	Wert

In unserem Programmbeispiel oben heißt das jetzt konkret, der Typ ist ***'int'***, der Name ist '**i**', die Adresse wird während der Laufzeit zugewiesen, darauf haben sie keinen Einfluss. Wir nehmen hier z.B. $0001. Der Wert ist der den sie mit ***'scanf'*** noch eingeben müssen. Wenn sie jetzt z.B. 5 eingeben heißt das in unserem Fall...

Typ	Name	Adresse	Wert
int	i	$0001	5

Das '**&**'- Zeichen ist in diesem Fall nichts anderes als der **Adressoperator**, das heißt der Variablen '**i**' vom Typ ***'int'*** mit der Adresse '0001' wird der Wert 5 zugewiesen.

```
Hinweis:
scanf("%d",zahl); //FALSCH da Adressoperator & fehlt
```

Das nachstehende Beispiel zeigt ein Problem beim Einlesen von mehreren Variablen. Dabei wird das Enter-Zeichen gepuffert und scanf() liest es automatisch ein.

```
//scanf2.c
#include <stdio.h>
int main(){
 char a,b,c,tmp;
 printf("1. Buchstabe : ");
 scanf("%c",&a);
 printf("2. Buchstabe : ");
 scanf("%c",&b);
 printf("3. Buchstabe : ");
 scanf("%c",&c);
 printf("Sie gaben ein : %c %c %c ",a,b,c);
return 0;
}
```

Eine weitere Möglichkeit ist die Verwendung von fflush().

```
//scanf4.c
#include <stdio.h>
int main(){
 char a,b,c;
 printf("1. Buchstabe : ");
 scanf("%c",&a);
 fflush(stdin);
 printf("2. Buchstabe : ");
 scanf("%c",&b);
 fflush(stdin);
 printf("3. Buchstabe : "
 scanf("%c",&c);
 printf("Sie gaben ein : %c %c %c ", a,b,c);
return 0;
}
```

Die letzte Möglichkeit ist, **scanf()** gar nicht zu verwenden.

Zeichenweise Ein- und Ausgabe

putchar

Mit der Funktion getchar() können einzelne Zeichen von der Tastatur eingelesen werden. Die Eingabe ist gepuffert, d.h. die Eingabe muss mit Enter abgeschlossen werden. Mit putchar() kann man einzelne Zeichen ausgeben

Beispiel:

```
//putchar.c eingabe und ausgabe
#include <stdio.h>
 int main(){
 int ch;
```

```
 while ( (ch=getcha  r()) != EOF)
 putchar(ch);
return 0;
}
```