Inhaltsverzeichnis

Wichtige Befehle der Programmiersprache C **2**
Klein- und Großschreibung 2
Kommentare 2
Zahlentypen in C 2
Wahrheitswerte 2
Variable 2
Konstante 3
Reihen (arrays) 3
Einfache Rechnungen 3
Vergleichsoperatoren 4
Einfache Verzweigung 4
Mehrfache Verzweigung 5
For-Schleife 5
While-Schleife 6
Do-While-Schleife 6
Logikoperatoren 7
Funktionen 7
Typenumwandlung 9
Bot-Befehle **10**
Warten 10
Motoren 10
LEDs 10
Textausgabe 10
Rad-Encoder 11
Abstandsensoren 11
Abgrundsensoren 12
Lichtsensoren 12
Kompass-Sensor 12
Tastaturbefehle 12

Wichtige Befehle der Programmiersprache C

Klein- und Großschreibung

C unterscheidet zwischen Groß- und Kleinschreibung.
Beispiele:
A und a sind unterschiedliche Variablen.
Alle von C reservierten Befehle, z.B. void, return, switch, for, while, do, ... müssen kleingeschrieben werden.

Kommentare

Kommentare werden vom Compiler nicht berücksichtigt. Kommentare können vom Programmierer zur Dokumentation eingesetzt werden, damit andere Programmierer sein Programm besser verstehen können.
Kommentare können auch eingesetzt werden, um Befehle kurzzeitig zu deaktivieren
// Ich bin ein Kommentar

Zahlentypen in C

1. Ganzzahlige Datentypen (Auswahl):
int (-32768 bis +32767)
int8_t (-128 bis 127)
int16_t (-32768 bis +32767)
int32_t (-2147483648 bis 2147483647)
uint8_t (0 bis 255)
uint16_t (0 bis 65535)

2. Gleitkommazahlen (Auswahl):
float (±1. 175e-38 bis ±3. 402e38)
double (±5.0E-324 bis ±1.7E308)

Wahrheitswerte

Der Datentyp bool (Wahrheitswerte) umfasst nur die zwei Werte true und false.

Variable

Bevor eine Variable verwendet werden kann, muss zunächst ihr Datentyp festgelegt werden. Dies wird auch Deklaration der Variablen genannt
Beispiel für die Deklaration der Variablen x und y (Kommazahlen), i (Ganze Zahl) und w (Wahrheitswert):

```
float x,y;
int i;
bool w;
```

Nach der Deklaration besitzen Variable irgendeinen (zufälligen) Wert.
Ein Wert lässt sich einer Variablen folgendermaßen zuweisen:

```
x=5.01;
int i=7;
bool w=true;
```

Man unterscheidet zwischen lokal und global definierten Variablen.
Lokale Variablen werden innerhalb einer Funktion deklariert und sind auch nur dort gültig. Die Deklaration einer lokalen Variablen muss innerhalb der geschweiften Klammern der zugehörigen

Funktion erfolgen. Bevor einer Variablen ein Wert zugewiesen wird, oder mit ihr gerechnet wird, muss sie deklariert werden:

```
Rückgabedatentyp Funktionsname(Parameter)
{
  float x;
  int i,j;
  bool w;
   ...
   ...
}
```

Globale Variablen gelten im Gegensatz zu lokalen Variablen im gesamten Programm, sie werden im Kopf des Programms, deklariert:

```
float x,y,z;
int i1,i2,i3;
bool w;
```

Falls möglich, sollten nur lokale Variablen verwendet werden.
Globale Variablen haben ein größeres Fehlerpotenzial, da sie von mehreren Funktionen geändert werden können.

Konstante

Im Gegensatz zu einer Variablen, kann der einmal festgelegte Wert einer Konstanten nicht mehr geändert werden
Beispiel: In der Konstanten mit dem Namen Pi wird ein Näherungswert für die Zahl Pi gespeichert:

```
const  float Pi=3.141593
```

Reihen (arrays)

Benötigt man eine Vielzahl von Variablen mit dem gleichen Datentyp, die man durchnummerieren möchte, dann bietet sich ein array (Reihe) an:
Beispiel: Eine Reihe von 7 Variablen für ganze Zahlen:

```
int a[7];
```

Damit erhält man die Variablen a[0], a[1], a[2], a[3], a[4], a[5], a[6]
Diesen Variablen kann man Werte zuweisen und mit ihnen rechnen, wie mit allen anderen Variablen auch.
Beispiel:

```
a[0]=25;
a[1]=2*a[0]-6;
a[2]=-17;
a[3]=i*29;
a[4]=2;
a[5]=113;
a[6]=5;
```

Achtung! Der Index fängt immer bei Null an.

Einfache Rechnungen

Zum Durchführen von Berechnungen mit Zahlen stehen die einfachen Grundrechenarten zur Verfügung:
+ (Pluszeichen)
- (Minuszeichen)
* (Malzeichen)

/ (Geteiltzeichen)
((Klammer auf)
) (Klammer zu)
Beispiel:
Das Ergbenis der Rechnung wird der Variable y zugewiesen:
float y;
y=7*(x-23)+80*x/13;

Vergleichsoperatoren

Zwei Zahlen lassen sich mit den folgenden Vergleichsoperatoren miteinander vergleichen:
Vergleichsoperatoren
== (gleich)
<= (kleiner gleich)
>= (größer gleich)
< (kleiner)
> (größer)
!= (ungleich)
Das Ergebnis eines solchen Vergleichs ist vom Datentyp bool (true oder false)
Beispiel
Das Ergebnis von x==7 ist false, wenn in x ein Wert ungleich 7 gespeichert ist. Ansonsten ist das Ergebnis true
Man könnte das Ergebnis zum Beispiel einer Variablen vom Typ bool zuweisen:
bool w=(x==7);
Das erste Gleichheitszeichen hinter w steht für eine Zuweisung des Wertes (true oder false) an die Variable w. Die zwei Gleichheitszeichen zwischen 5 und 7 bilden den Vergleichsoperator, der Überprüft, ob links und rechts wirklich der gleiche Wert gespeichert ist.

Einfache Verzweigung

Bei der If-Verzweigung wird zwischen zwei Wegen entschieden:

```
if (wahrer Ausdruck) ... ;
else ... ;
```

Falls mehrere Befehle ausgeführt werden sollen, so fasst man diese Blöcken zusammen:

```
if (wahrer Ausdruck)
{
  ...;
  ...;
  ...;
}
else
{
  ...;
  ...;
  ...;
}
```

Achtung! Vor { darf hier kein Semikolon stehen!
Beispiel: Falls x>5 soll der Wert von x um 25 erhöht werden. Ansonsten soll x um 1 erhöht werden:

```
if (x>5) x=x+25;
else x=x+1;
```

Mehrfache Verzweigung

Im Gegensatz zur If-Verzweigung kann hier zwischen mehr als 2 Wegen entschieden werden:

```
switch(Ausdruck)
{
case Ausdruckswert1: ...;
break;
case Ausdruckswert2: ...;
break;
.
.
case Ausdruckswertn: ...;
break;
default: ...;
}
```

Bemerkung: Ohne „break“ werden die nachfolgenden Fälle ebenfalls ausgeführt.
Beispiel: Hier ist i eine Variable vom Datentyp int.
Falls i den Wert 1 hat, soll die Zuweisung y=2.0 ausgeführt werden.
Falls i den Wert 2 hat, soll die Zuweisung x=3.5*x ausgeführt werden.
Falls i den Wert 13 hat, soll die Zuweisung x=5.7*i ausgeführt werden.
Falls i einen anderen Wert hat, soll die Zuweisung x=i ausgeführt werden.
Für diese Fälle sieht die switch-Verzweigung folgendermaßen aus:

```
switch(i)
{
case 1: y=2.0;
break;
case 2: x=3.5*x;
break;
case 13: x=5.7*i;
break;
default: x=i;
}
```

For-Schleife

Falls ein oder mehrere Befehle öfters wiederholt werden sollen und die Anzahl der Wiederholungen bekannt ist, dann ist eine For-Schleife die richtige Wahl.

- Falls nur ein Befehl wiederholt werden soll:
  for (Initialisierung; Laufbedingung; Schrittweite) ... ;
- Falls mehrere Befehle nacheinander wiederholt werden sollen:

```
for (Initialisierung; Laufbedingung; Schrittweite)
{
 ...;
...;
...;
}
```

Achtung! Vor { darf hier kein Semikolon stehen!

Beispiel: Es sollen alle Ganzen Zahlen zwischen 1 und 207 addiert werden. Zusätzlich sollen alle Vielfache von 3 bis einschließlich 621 addiert werden.

```
int i;
int sum1=0;
int sum2=0;
for (i=1;i<=207;i++)
{
  sum1=sum1+i;
  sum2=sum2+3*i;
}
```

(Bemerkung: i++ ist eine Abkürzung für i=i+1)

While-Schleife

Falls ein oder mehrere Befehle öfters wiederholt werden sollen und die Anzahl der Wiederholungen nicht ohne Weiteres bekannt ist, dann ist eine While-Schleife die richtige Wahl.
Falls nur ein Befehl wiederholt werden soll:

```
while (Schleifenbedingung) ... ;
```

Falls mehrere Befehle nacheinander wiederholt werden sollen:

```
while (Schleifenbedingung)
{
  ...;
  ...;
  ...;
}
```

Achtung! Vor { darf hier kein Semikolon stehen!
Die Schleife wird solange wiederholt, bis die Schleifenbedingung nicht mehr wahr (true) ist.
Beispiel: Es soll die kleinste Potenz von 3 gefunden werden, die größer als 8567 ist

```
int prod=1;
while (prod<=8567) prod=prod*3;
```

Do-While-Schleife

Die Do-While-Schleife ist eine Variante der While-Schleife. Bei der While-Schleife wird bereits vor Eintritt in die Schleife überprüft, ob die Schleifenbedingung wahr ist. Falls nicht, wird die Schleife nicht ausgeführt. Bei der Do-While-Schleife wird die Schleifenbedingung erst nach dem ersten Durchlauf überprüft.
Falls nur ein Befehl wiederholt werden soll:

```
do ... while (Schleifenbedingung);
```

Falls mehrere Befehle nacheinander wiederholt werden sollen:

```
do
{
  ...;
```

```
  ...;
  ...;
}
while (Schleifenbedingung);
```

Achtung! Vor { darf hier kein Semikolon stehen!
Die Schleife wird wiederholt, solange die Schleifenbedingung wahr (true) ist.
Beispiel: Es soll die kleinste Potenz von 3 gefunden werden, die größer als 8567 ist.

```
int prod=1;
do  prod=prod*3 while(prod<=8567);
```

Logikoperatoren

Auf Wahrheitswerte (bool: true oder false) können folgende Operatoren angewandt werden:

&& (und)
|| (oder)
! (nicht)

Das Ergebnis ist wieder ein Wahrheitswert (bool: true oder false)
Beispiel:
Eine Schleife soll nur dann fortgesetzt werden, wenn x ungleich 7 und gleichzeitig y>20:

```
bool weiter;
weiter=((x!=7) && (y>20))
```

Funktionen

Um das Hauptprogramm übersichtlich zu gestalten, ist es sinnvoll, bestimmte Teilaufgaben als Funktionen, auch Unterprogramme genannt, auszulagern. An Funktionen in C kann man wie bei mathematischen Funktionen Werte übergeben (in C Parameter genannt) und Werte zurückerhalten (in C werden Werte mit return(...) zurückgegeben). Die Definition einer Funktion findet unterhalb des Hauptprogramms statt. Oberhalb des Hauptprogramms werden die Funktionen deklariert (d.h., dem Compiler bekannt gemacht) Achtung: Im Gegensatz zur Definition einer Funktion muss die Deklaration einer Funktion mit einem Semikolon abgeschlossen werden! Im Hauptprogramm selbst wird dann die Funktion mit den gewünschten Parametern aufgerufen, so dass sie beim Programmablauf an dieser Stelle ausgeführt wird. Man kann eine Funktion auch an mehreren Stellen im Hauptprogramm oder in anderen Funktionen aufrufen. Hierbei können auch unterschiedliche Parameterwerte an die Funktion übergeben werden. Wird eine Funktion nicht im Hauptprogramm oder einer anderen Funktion aufgerufen, dann wird sie während dem Programmablauf auch nicht ausgeführt. Nur die im Hauptprogramm aufgerufenen Funktionen werden ausgeführt.

Aufbau (Definition) einer Funktion in C:

```
Rückgabedatentyp Funktionsname(Parameter)
{
  ...
  return(...)
}
```

Bei return(...) wird die Funktion verlassen.
Beispiel 1:

Es soll eine Funktion mit dem Namen "x_hoch_2" definiert werden, die das Quadrat einer Kommazahl als Ergebnis zurückgibt:

```
float x_hoch_2(float x)
{
  return(x*x);
}
```

Beispiel 2:
Es soll eine Funktion mit dem Namen "Maximum" definiert werden, die zwei Kommazahlen vergleicht und die größere der beiden Zahlen als Ergebnis zurückgibt.

```
float maximum(float x, float y)
{
  if (x>y) return(x);
  else return(y);
}
```

Funktionen kann man nun in der Definition von anderen Funktionen oder im Hauptprogramm aufrufen.
Beispiele für den Aufruf der oben definierten Funktionen:
Das Ergebnis soll jeweils der Variable y zugewiesen werden.

```
float y;
y=x_hoch_2(x);
y=x_hoch_2(9.56);
y=Maximum(x,y);
y=Maximum(x,13.1);
y=Maximum(-5,18);
```

In C kann man auch Funktionen definieren, die keine Ergebnisse zurückliefern:

```
void Funktionsname1(Parameter)
{
  ...
}
```

Bemerkung: void = das Nichts

In C kann man auch Funktionen definieren, die keine Übergabeparameter besitzen:

```
Rückgabedatentyp Funktionsname2(void)
{
  ...
  return(...)
}
```

In C kann man auch Funktionen definieren, die keine Ergebnisse zurückliefern und keine Übergabeparameter besitzen:

```
void Funktionsname3(void)
{
```

```
  ...
}
```

Beim Aufruf einer Funktion werden alle "void" weggelassen. Diese stehen nur in der Definition und Deklaration einer Funktion.
Aufruf einer solchen Funktion:

```
Funktionsname3();
```

Typenumwandlung

In C ist oftmals eine Umwandlung von int nach float notwendig, wenn in einer Rechnung keine Float-Werte verwendet werden:
Beispiel für Fehler ohne Umwandlung:

```
int n,m;
float x;
n=5;
m=2;
x=n/m;
```

Hier wird x der Wert 2,0 zugewiesen, obwohl 5/2=2,5 gilt.
Wie kommt es zu diesem Verhalten?
Da n und m ganze Zahlen sind, wird die Rechnung zuerst im Bereich der Ganzen Zahlen durchgeführt. Hierbei werden die Nachkommastellen abgeschnitten. Anschließend wird dann das Ergebnis 2,0 der Variablen x zugewiesen.
Wie kann dieses Problem umgangen werden?
Lösung: Man wandelt für die Rechnung eine Int-Zahl in eine Float-Zahl um:

```
x = (float)(n)/m;
```

Damit wird x das richtige Ergebnis 2,5 zugewiesen.
Erklärung: Innerhalb einer Rechnung wandelt der Compiler die Int-Werte von selbst in Float-Werte um, wenn innerhalb der Rechnung mindestens ein Float-Wert vorhanden ist.

Bot-Befehle

Aktoren:

Warten

Der Warten-Befehl liefert eine einstellbare Verzögerung:
void warten(uint32_t ms)
Hierbei wird für den Parameter "ms" die Warte-Zeit in Millisekunden eingesetzt.
Beispielanwendung:
Die 1. LED soll für 0,5 Sekunden leuchten:

```
LED_ein(1);
warten(500);
LED_aus(1);
```

Motoren

Der Motoren-Befehl steuert die Drehzahl des linken und des rechten Rades:
void Motoren(int links, int rechts)
Hierbei können für die Parameter "links" und "rechts" Werte zwischen - 100 und 100 eingesetzt werden. Der Wert für "links" steuert die Drehzahl des linken Rades, der Wert für "rechts" steuert die Drehzahl des rechten Rades. Ein negativer Wert bedeutet, dass sich das Rad nach hinten dreht. Ein positiver Wert bedeutet, dass sich das Rad nach vorne dreht. Beim Wert 0 steht das Rad still. Beim Wert 100 dreht sich das Rad mit maximaler Drehzahl nach vorne. Beim Wert -100 dreht sich das Rad mit maximaler Drehzahl nach hinten.
Beispielanwendung:
Der Bot soll eine leichte Linkskurve fahren:
Motoren(40,80);

LEDs

Die LED-Befehle steuern das Ein- und Ausschalten der 8 LEDs:
void LED_ein(int LED_Nummer)
void LED_aus(int LED_Nummer)
Die LED-Nummern (LED_Nummer) gehen von 1 bis 8. Wird der Befehl LED_ein mit der gewünschten LED-Nummer aufgerufen, dann leuchtet diese LED solange, bis der Befehl LED_aus mit dieser Nummer aufgerufen wird.
Beispielanwendung:
Die 1. LED soll für 0,5 Sekunden leuchten:

```
LED_ein(1);
warten(500);
LED_aus(1);
```

Textausgabe

Mit den Textausgabe-Befehlen können im Display des Bots Texte mit maximal 50 Zeichen ausgegeben werden.

void Textausgabe(const char * text)
void Displayloeschen(void)
void Intausgabe(int32_t k)
void Text_Intausgabe(const char * text, int32_t k)
void Doubleausgabe(double x)
void Text_Doubleausgabe(const char * text, double x)

Beispielanwendungen:
Der Text "Ich fahre." soll ausgegeben werden:
Textausgabe("Ich fahre.");
Der Wert der ganzzahligen Variable i soll ausgegeben werden:
Intausgabe(i);
Der Text "i=" und der Wert der ganzzahligen Variable i sollen ausgegeben werden:
Text_Intausgabe("i=",i);
Der Wert der Kommazahl-Variablen y soll ausgegeben werden:
Doubleausgabe(y);
Der Text "y=" und der Wert der Kommazahl-Variablen y sollen ausgegeben werden:
Text_Doubleausgabe("y=",y);

Sensoren:

Rad-Encoder

An jedem Rad ist ein sogenannter Rad-Encoder-Sensor befestigt. Mit diesen Radencodern kann man messen, um welchen Winkel sich ein Rad gedreht hat. Der Wert des Radencoders wird jeweils um 1 erhöht, wenn sich das Rad um ca. 3,6 Grad weiter noch vorne gedreht hat. Der Wert des Radencoders wird jeweils um 1 erniedrigt, wenn sich das Rad um ca. 3,6 Grad weiter noch hinten gedreht hat.
Der Radencoder-Befehl für das linke Rad:
int32_t SensorEncoderL(void);
Der Radencoder-Befehl für das rechte Rad:
int32_t SensorEncoderR(void);
Aufruf der Befehle:
Um den aktuellen Wert des rechten Radencoders einzulesen, muss folgender Befehl aufgerufen werden:
SensorEncoderR();

Abstandsensoren

Auf der linken und der rechten Seite des Bots sind jeweils ein Ultraschall-Abstandsensor befestigt. Diese Sensoren sind nach vorne ausgerichtet. Der Wert eines Abstandsensors gibt den Abstand des Sensors vom Hindernis in der Einheit cm wieder. Es könne nur Abstände bis maximal 255 cm gemessen werden.
Der Befehl für den linken Sensor:
int SensorAbstandL(void)
Der Befehl für den rechten Sensor:
int SensorAbstandR(void)
Aufruf der Befehle:
Um den aktuellen Wert des rechten Abstand-Sensors einzulesen, muss folgender Befehl aufgerufen werden:
SensorAbstandR();

Abgrundsensoren

Der Bot besitzt insgesamt 4 Ultraschall-Abgrundsensoren. Zwei in der Mitte und zwei vorne (jeweils links und rechts). Diese Sensoren sind nach unten ausgerichtet. Der Wert eines Abstandsensors gibt den Abstand des Sensors vom Boden in der Einheit cm wieder. Es könne nur Abstände bis maximal 255 cm gemessen werden.
Der Befehl für den vorderen, linken Sensor:
int SensorAbgrundVL(void);
Der Befehl für den vorderen, rechten Sensor:
int SensorAbgrundVR(void);
Der Befehl für den mittleren, linken Sensor:
int SensorAbgrundML(void);
Der Befehl für den mittleren, rechten Sensor:
int SensorAbgrundMR(void);

Aufruf der Befehle:
Um den aktuellen Wert des vorderen, linken Abgrund-Sensors einzulesen, muss folgender Befehl aufgerufen werden:
SensorAbgrundVL();

Lichtsensoren

Auf der Vorderseite des Bots befinden sich links und rechts jeweils ein Lichtsensor (LDR). Diese Sensoren können Werte zwischen 0 und 4095 annehmen. Je kleiner der Wert, desto mehr Licht fällt auf den Sensor.
Der Befehl für den linken Sensor:
int SensorLichtL(void)
Der Befehl für den rechten Sensor:
int SensorLichtR(void)
Aufruf der Befehle:
Um den aktuellen Wert des rechten Licht-Sensors zu erfahren, muss folgender Befehl aufgerufen werden:
SensorLichtR();

Kompass-Sensor

In den Bot ist ein Kompass-Sensor eingebaut. Der Kompass liefert Werte zwischen 0 und 360 Grad.
Der Befehl für den Kompass-Sensor:
int SensorKompass(void)
Aufruf des Befehls:
SensorKompass()

Tastaturbefehle

Mit den folgenden Befehlen, kann im Programm überprüft werden, ob eine der Tasten
W, E, A, S ,D oder F betätigt wird. Dies kann z.B. genutzt werden, um in die Bewegung des Bots interaktiv einzugreifen, und so z.B. eine Fernsteuerung zu programmieren.

bool W_Taste(void)
bool E_Taste(void)
bool A_Taste(void)
bool S_Taste(void)

bool D_Taste(void)
bool F_Taste(void)
Wird z.B. die Taste W auf der Tastatur gedrückt, dann liefert die Funktion W_Taste() den Rückgabewert true, ansonsten false
Beispielaufruf:
Wenn die A-Taste betätigt wird, soll LED 5 leuchten, ansonsten nicht:

```
if (A_Taste()) LED_ein(5);
else LED_aus(5);
```