C - Standardbibliothek

C verfügt über keine eingebauten komplexen Funktionalitäten für z. B. Ein- und Ausgabeoperationen (im Gegensatz zu anderen Sprachen, wie z.B. FORTRAN). Die komplexeren Funktionen werden von einer C-Standardbibliothek zur Verfügung gestellt. Der Umfang der Standard-bibliothek ist standardisiert, d.h. man kann auf jeder Rechnerplattform von einem gewissen Satz von Funktionen ausgehen.

Es gibt auch noch spezialisierte Bibliotheken, z.B. für die Programmierung von grafischen Benutzeroberflächen (GUIs)

- Windows: Windows-API, MFC (Microsoft Foundation Classes, C++)
- Unix: X-Bibliothek, Motif, Qt, KDE

Bibliotheken für Datenstrukturen:

Arrays, Sortierverfahren u.ä. in C++ STL Bibliothek.

C-Standardbibliothek

Funktionen für:

- Zeit und Datum
- Zeichenkettenverarbeitung
- Dynamische Speicherverwaltung
- Ein- und Ausgabe
- Typumwandlungen
- Steuerungs- und Betriebssystemfunktionen
- Mathematikfunktionen

Konstanten:

Limit-Konstanten (z.B. maximale darstellbare Werte eines Typs)

C-Standardbibliothek

Um Funktionen aus der C-Standardbibliothek benutzen zu können, müssen die Funktionen vorher dem Compiler bekannt gemacht werden. Das passiert durch die Präprozessor-Anweisung

#include <…>

die eine Textdatei (s.g. Header-Datei) einbindet, welche die Deklaration ausgewählter Standardfunktionen, Konstanten und Makros enthält.

Beispiel:

#include <string.h>

wobei eine Zeile in *string.h* die Funktion *strcpy* deklariert:

char strcpy(char *dest, char *src);*

Bei der späteren Benutzung der Funktion *strcpy* ist diesen dem Compiler bekannt. Diese Funktion wird dann aus einer vorab übersetzten Bibliothek an das Programm angefügt, oder dynamisch aus einer DLL-aufgerufen.

C-Standardbibliothek

Die Funktionen der C-Standardbibliothek sind in unterschiedlichen Header-Dateien deklariert. Jede Header Datei gruppiert eine Reihe von Bibliotheksfunktionen, z.B.

math.h – Mathematische Funktionen, wie sin(x), pow(x,e)

string.h – Funktionen zur Zeichenkettenverarbeitung,
wie strcpy (steht für string copy), strcmp (string compare),

stdio.h – Funktionen zur Ein- und Ausgabe, wie printf, scanf

Ein Programm kann beispielsweise so beginnen:

```
#include <stdio.h>
#include <string.h>
#include <math.h>
…
```

Headerdateien der Standardbibliothek

Für C89/C90 werden folgende Header und Funktionsgruppen definiert:

assert.h	*Assertions*
ctype.h	*Tests auf bestimmte Zeichentypen*
errno.h	*Codes von Systemfehlern*
float.h	*Angaben zu den Wertbereichen von Gleitkommazahlen*
limits.h	*Angaben zu Beschränkungen des verwendeten Systems*
locale.h	*Einstellungen des Gebietsschemas*
math.h	*Mathematische Funktionen*
setjmp.h	*erweiterte Sprungfunktionen*
signal.h	*Signalbehandlung*
stdarg.h	*Argumentbehandlung für variadische Funktionen*
stddef.h	*zusätzliche Typdefinitionen*
stdio.h	*Ein- und Ausgabe*
stdlib.h	*vermischte Standardfunktionen, u.a. Speicherverwaltung*
string.h	*Zeichenkettenoperationen*
time.h	*Datum und Uhrzeit*

Ausgewählte Funktionen: stdio.h (1)

Ein- und Ausgabe: *<stdio.h>*

Ein Großteil der Bibliotheksfunktionen (ca. ein Drittel) sind die Ein- und Ausgabefunktionen, Typen und Makros, die in <stdio.h> vereinbart sind.

Formatierte Ein- und Ausgabe über Konsole:

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

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

Formatierte Ein- und Ausgabe von/aus Strings

*int sscanf(const char *s, const char *format, ...);*

sscanf(s, ...) ist äquivalent zu scanf(...), mit dem Unterschied, dass die Eingabezeichen aus der Zeichenkette s stammen.

*int sprintf(char *s, const char *format, ...)*

sprintf funktioniert wie printf, nur wird die Ausgabe in den Zeichenvektor s geschrieben und mit '\0' abgeschlossen.

Ausgewählte Funktionen: stdio.h (2)

Formatierte Ein- und Ausgabe über Konsole:

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

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

Der Formatstring *format* ist eine gewöhnliche Zeichenkette, die oft als Zeichenkettenkonstante angegeben wird, also direkt im Programmcode.

Darin können zwei Arten von Objekten vorkommen:

- gewöhnliche Zeichen: diese werden direkt in die Ausgabe kopiert
- Umwandlungsangaben, veranlassen eine Umwandlung vom Argument in die Ausgabe (printf) oder von der Eingabe auf das Argument

Eine Umwandlungsangabe beginnt mit % und endet mit einem Umwandlungszeichen.

Zwischen % und dem Umwandlungszeichen können noch zusätzliche optionale Angaben gemacht werden, wie z.B. die Anzahl der dargestellten Dezimalstellen vor und nach dem Dezimalpunkt einer Fließkommazahl.

Ausgewählte Funktionen: stdio.h (3)

Formatierte Ein- und Ausgabe über Konsole …

Beispiel:

*printf("Das Datum ist: **%02d**. **%02d**. **%04d** \n",tag,mon,jahr);*

Die Umwandlungsangaben sind %02d, %02d und %04d.

Am Platz von %02d, %02d, %04d werden die Werte von tag, mon, jahr als 2- bzw. 4-stellige Dezimalzahlen ausgebeben. Es werden führende Nullen vorangestellt (wegen 02d anstatt 2d).

Ausgabe:

Das Datum ist: 24. 02. 2011

Ausgewählte Funktionen: stdio.h (4)

scanf/scanf_s - Umwandlungszeichen

Umwandlungszeichen	Eingabeformat	Variable
%d	*dezimal, ganzzahlig*	*int **
%i	*ganzzahlig (oktal (mit 0 am Anfang) oder hexadezimal (mit 0x oder 0X am Anfang)*	*int **
%o	*oktal ganzzahlig (mit oder ohne 0 am Anfang)*	*int **
%u	*dezimal ohne Vorzeichen*	*unsigned int **
%x	*hexadezimal ganzzahlig (mit oder ohne 0x oder 0X am Anfang)*	*int **
%c	*ein einzelnes Zeichen*	*char **
%s	*Folge von Nicht-Zwischenraum-Zeichen (ohne Anführungszeichen)*	*char **
%e, %f, %g	*Fließkommazahl (float)*	*float **
%lf	*Fließkommazahl (double)*	*double**
%p	*Zeiger, wie ihn printf("%p") ausgibt*	*void **

Ausgewählte Funktionen: stdio.h (5)

printf-Umwandlungen

Umwandlungszeichen	Argument	Umwandlung in
%d, %i	*int*	*dezimal mit Vorzeichen*
%u	*Int*	*dezimal ohne Vorzeichen*
%o	*int*	*oktal ohne Vorzeichen (ohne führende Null)*
%x, %X	*int*	*hexadezimal ohne Vorzeichen (ohne führendes 0x oder 0X)*
%c	*int, char*	*einzelnes Zeichen,*
%s	*char **	*aus einer Zeichenkette werden Zeichen ausgegeben bis vor '\0'*
%f, %lf (bei double)	*float, double*	*dezimal als [-]mmm.ddd, wobei die Genauigkeit die Anzahl der d festlegt. Voreinstellung ist 6; bei 0 entfällt der Dezimalpunkt*
%e, %E	*float, double*	*dezimal als [-]m.dddddde±xx oder [-]m.ddddddE±xx, wobei die Genauigkeit die Anzahl der d festlegt. Voreinstellung ist 6; bei 0 entfällt der Dezimalpunkt*

Ausgewählte Funktionen: stdio.h (6)

printf-Umwandlungen (Fortsetzung)

Umwandlungszeichen	Argument	Umwandlung in
%g, %G	*float, double*	*%e oder %E wird verwendet, wenn der Exponent kleiner als -4 oder nicht kleiner als die Genauigkeit ist; sonst wird %f benutzt*
%p	*void **	*als Zeiger, die Darstellung kann je nach Implementierung variieren*

Ausgewählte Funktionen: stdio.h (7)

Formatierte printf-Ausgabe von Fließkommazahlen:

```
#define PI 3.1415926535
#define Z 3
float a[Z];
int i;
…
for (i=0;i<Z;i++)
{ a[i]= i*PI*5.0;
   printf("Wert %02d: %f \n",i,a[i]);
}
```

Ausgabe:

```
Wert 00: 0.000000
Wert 01: 15.707963
Wert 02: 31.415926
```

6 Stellen nach dem Komma sind Standard

```
…
for (i=0;i<Z;i++)
{ a[i]= i*PI*5.0;
   printf("Wert %02d: %6.2f \n",i,a[i]);
}
```

Ausgabe:

```
Wert 00:    0.00
Wert 01:   15.71
Wert 02:   31.42
```

Formatierung auf 6 Zeichen inkl. Dezimalpunkt und 2 Stellen nach Komma

Ausgewählte Funktionen: stdlib.h (1)

Standard-Lib(rary):

Dieser Teil deklariert einige Funktionen zur Umwandlung von Zahlendarstellungen, wie z.B.

*long atol(const char *s)* Umwandlung von ASCII Text nach long int

Funktionen zur Erzeugung von Zufallszahlen

int rand(void) liefert eine ganzzahlige Pseudo-Zufallszahl im Bereich von 0 bis RAND_MAX; RAND_MAX ≥ 32767. Jeder neue Programmlauf erzeugt gleiche Folge von Zufallszahlen (bei ausschließlicher Nutzung von *rand()*)

void srand(unsigned int seed)

benutzt seed als Ausgangswert für eine neue Folge von Pseudo-Zufallszahlen.

Soll wieder mit dem ersten Wert einer Zufallszahlenfolge begonnen werden, so ist *srand(1)* auszuführen. Der darauf folgende Aufruf rand() gibt dann den ersten Wert aus.

Ausgewählte Funktionen: stdlib.h (2)

void abort(void)

abort sorgt für eine anormale Beendigung des Programms.

void exit(int status)

exit beendet das Programm normal. Wie status an die Umgebung des Programms geliefert wird, hängt von der Implementierung ab, aber Null gilt als erfolgreiches Ende. Die Werte EXIT_SUCCESS und EXIT_FAILURE können ebenfalls angegeben werden.

*int system(const char *s)*

system liefert die Zeichenkette s an die Umgebung zur Ausführung. Hat s den Wert NULL, so liefert system einen von Null verschiedenen Wert, wenn es einen Kommandoprozessor gibt. Wenn s von NULL verschieden ist, dann ist der Resultatwert implementierungsabhängig.

Mathematische Funktionen: math.h (1)

In <math.h> sind mathematische Funktionen und Makros vereinbart.

Trigonometrische Funktionen:

double sin(double x)	Sinus von x
double cos(double x)	Kosinus von x
double tan(double x)	Tangens von x

Winkelwert x muss in „rad“ übergeben werden.
Dabei misst ein Vollkreis 2 π rad

Umkehrfunktionen für sin, cos, tan:

double asin(double x)	arcsin(x) im Bereich [-π/2, π/2], x in [-1, 1].
double acos(double x)	arccos(x) im Bereich [0, π], x in [-1, 1].
double atan(double x)	arctan(x) im Bereich [-π/2, π/2].
double atan2(double x, double y)	arctan(y/x) im Bereich [-π, π].

Mathematische Funktionen: math.h (2)

Fortsetzung Trigonometrische Funktionen:

double sinh(double x) Sinus Hyperbolicus von x

double cosh(double x) Cosinus Hyperbolicus von x

double tanh(double x) Tangens Hyperbolicus von x

Potenzen und Logarithmen:

double exp(double x) Exponentialfunktion e^x

double log(double x) natürlicher Logarithmus, ln(x), x > 0.

double log10(double x) Logarithmus zur Basis 10, log10(x), x > 0.

double pow(double x, double y) Potenzierung x^y

Mathematische Funktionen: math.h (3)

Fortsetzung:

double sqrt(double x)	Wurzel von x, x >= 0.
double ceil(double x)	kleinster ganzzahliger Wert, der nicht kleiner als x ist.
double floor (double x)	größter ganzzahliger Wert, der nicht größer als x ist.
double fabs(double x)	absoluter Wert | x |
double ldexp(double x,n)	x^{2n}

… und weitere Funktionen.

Mathematische Funktionen: math.h (4)

Beispiel 1: Abrunden, wenn nur vollständige Anteile gezählt werden sollen.

$$\text{AnzahlVolleFlaschen} = \left\lfloor \frac{\text{Volumen\_Fass}}{\text{Volumen\_Flasche}} \right\rfloor$$

```
#include <math.h>
int AnzahlVolleFlaschen;
double Volumen_Fass;
const double Volumen_Flasche = 0.75;
…
AnzahlVolleFlaschen = (int) floor ( Volumen_Fass / Volumen_Flasche );
…
```

Mathematische Funktionen: math.h (5)

Beispiel 2: Aufrunden, wenn auch unvollständige Anteile (voll) gezählt werden.

$$\text{AnzahlTouren} = \left\lceil \frac{\text{Gewicht\_Gesamtladung}}{\text{max\_Zuladegewicht}} \right\rceil$$

```
#include <math.h>
int AnzahlTouren;
double Gewicht_Gesamtladung;
double max_Zuladegewicht;
…
AnzahlTouren = (int) ceil ( Gewicht_Gesamtladung / max_Zuladegewicht );
…
```

Mathematische Funktionen: math.h (6)

Beispiel 3: Benutzung der trigonometrischen Funktionen

- Eine Funktion *winkel_zeiger_ms*, die einen Sekundenwert (oder Minutenwert) in eine Winkel eines Uhrenzeigers umrechnet.
- Eine weitere Funktion *zeiger_pos*, die mit dem Winkel eine Zeichenposition der Zeigerspitze in einem 2-dim Raster berechnet.

```
#include <math.h>
#define PI 3.1415926535

// Funktion berechnet den Winkel eines Zeigers in Grad
void winkel_zeiger_ms(int sek,  float *winkel)
{
  *winkel = (float)sek/60.0*360.0;
}
```

Mathematische Funktionen: math.h (7)

Beispiel 3 - Fortsetzung:

```
void zeiger_pos(int mp_x, int mp_y, int zeiger_len, float winkel,
                int *ep_x, int *ep_y)
{
  int dx, dy;
  float radiant_mass= winkel/360.0*2.0*PI;
  dx = (int)( sin(radiant_mass)*(float)zeiger_len );    // Argument als rad!
  dy = (int)( cos(radiant_mass)*(float)zeiger_len );   // …nicht als Winkel in Grad
  *ep_x = mp_x + dx;
  *ep_y = mp_y - dy;
 }
```

Benutzung:

```
int mp_x=300, mp_y=300, zeiger_len=200;
winkel_zeiger_ms(sek, &winkel);
zeiger_pos(mp_x, mp_y, zeiger_len, winkel, &ep_x, &ep_y);
// zeichne Zeiger-Linie von Punkt (mp_x, mp_y) nach Punkt (ep_x, ep_y)
```

Mathematische Funktionen & Fehlerbehandlung

Einige Konstanten und Makros zur Anzeige von Fehlern (insb. Verletzungen des Definitions- und Wertebereichs) bei Nutzung der math-Funktionen sind in ***errno.h*** definiert. Diese von Null verschiedenen ganzzahlige Konstanten, werden über eine Variable errno abgelesen:

- ***errno==0*** – kein Fehler
- ***errno == EDOM*** – Anzeige eines Fehlers im Argumentbereich einer Funktion. Ein solcher Fehler (domain error) liegt vor, wenn ein Argument nicht in dem Bereich liegt, für den eine Funktion definiert ist.
  Der Resultatwert hängt von der Implementierung ab.
- ***errno == ERANGE*** - Fehler im Resultatbereich der Funktionen. Dieser Fehler (range error) liegt vor, wenn das Resultat nicht als double dargestellt werden kann.
- ***HUGE_VAL*** - ist ein positiver double-Wert. Ist das Resultat absolut zu groß, liefert die Funktion HUGE_VAL mit dem korrekten Vorzeichen und errno == ERANGE. Ist das Resultat zu nahe bei Null, liefert die Funktion null; je nach Implementation kann errno den Wert ERANGE erhalten.

Wertebereiche: limits.h (1)

limits.h stellt Konstanten für den Wertebereich der ganzzahligen Typen bereit.

Konstante	(minimaler absoluter Wert)	Erklärung
CHAR_BIT	*8*	*Anzahl der Bits zur Darstellung eines char*
CHAR_MAX	*255*	*maximaler Wert für char*
CHAR_MIN	*0*	*minimaler Wert für char*
SCHAR_MAX	*127*	*maximaler Wert für signed char*
SCHAR_MIN	*-128*	*minimaler Wert für signed char*
UCHAR_MAX	*255*	*maximaler Wert für unsigned char*
INT_MAX	*+32.767*	*maximaler Wert für int*
INT_MIN	*-32.767*	*minimaler Wert für int*
UINT_MAX	*65.535*	*maximaler Wert für unsigned int*
LONG_MAX	*+2.147.483.647*	*maximaler Wert für long*
LONG_MIN	*-2.147.483.647*	*minimaler Wert für long*
ULONG_MAX	*4.294.967.295*	*maximaler Wert für unsigned long*
SHRT_MAX	*+32.767*	*maximaler Wert für short*
SHRT_MIN	*-32.768*	*minimaler Wert für short*
USHRT_MAX	*65.535*	*maximaler Wert für unsigned short*

Wertebereiche: limits.h (2)

Benutzung der Wertebereichskonstanten z.B. beim Finden eines Minimums und eines Maximums unter Int-Zahlen, die auch negativ sein können.

```
#include <limits.h>
#define NUM_ELEMENTE 1000
int max = INT_MIN;
int min = INT_MAX;
int e[NUM_ELEMENTE];
…
for (i=0;i<num_elemente;i++)
{   if (e[i] < min) min=e[i];
    if (e[i] > max) max=e[i];
}
```

Algorithmus funktioniert aber auch ohne diese Konstanten, wenn man anfangs max=e[0] und min=e[0] setzt; sofern ein e[0] existiert.

Wertebereiche: float.h

Auswahl einiger Konstanten bezogen auf Gleitpunktarithmetik

Konstante	(minimaler absoluter Wert)	Erklärung
FLT_DIG	*6*	*Genauigkeit in Dezimalziffern (für float)*
FLT_EPSILON	*lE-5*	*kleinster Wert x für den 1.0 + x ungleich 1.0 gilt*
FLT_MAX	*1E+37*	*maximaler Gleitpunktwert*
FLT_MIN	*1E-37*	*minimaler, normalisierter Gleitpunktwert*
DBL_DIG	*10*	*Genauigkeit in Dezimalziffern (für double)*
DBL_EPSILON	*lE-9*	*kleinster Wert x für den 1.0 + x ungleich 1.0 gilt*
DBL_MAX	*lE+37*	*maximaler Gleitpunktwert*
DBL_MIN	*lE-37*	*minimaler, normalisierter double Gleitpunktwert*