Die C-Stdlib

Matthias Bracht, Steffen Lang, Marcus Echter

17.05.2005

Die C-Stdlib

- C als „kleine“ Sprache
- insbesondere Ein-/Ausgabe nicht definiert
- keine Funktionen

„Ausnahmen“:

- **sizeof** *expression* bzw. **sizeof** *(type)*
- compiler-known functions (intrinsics)

Intrinsics: Beispiel

Saturated Addition in ANSI-C:

```
result = var1 + var2;
if (((var1 ^ var2) & 0x80000000) == 0)
    if ((result ^ var1) & 0x80000000)
      result = var1 < 0 ? 0x80000000 : 0x7fffffff;
```

Als compiler-known function:

```
result = _sadd(var1, var2);
```

Die C-Stdlib

- Standardbibliothek bietet standardisierte Funktionen, Typen, Makros
- 1989 vom American National Standards Institute (ANSI) als Teil von *ANSI C* genormt
- Aufgaben: Ein-/Ausgabe, Zeichenketten, Speicherverwaltung, mathematische Funktionen...
- Vorteil der Portabilität

Die C-Stdlib (Aufbau)

- Deklarierung in Standard-Header-Dateien
- Implementierung in Programmbibliothek ausgelagert: z.B. libc.so.6 (Linux), msvcrt.dll (Windows)
- Einbinden der jeweiligen Header-Dateien mit *#include <header>*

Die C-Stdlib: Header-Dateien

<assert.h>
<ctype.h>
<errno.h>
<float.h>
<limits.h>
<locale.h>
<math.h>
<setjmp.h>
<signal.h>
<stdarg.h>
<stddef.h>
<stdio.h>
<stdlib.h>
<string.h>
<time.h>

Die C-Stdlib – assert.h

- Fehlersuche mit Testpunkten

```
#undef assert

#ifdef NDEBUG
#define assert(p)   ((void)0)
#else
#define assert(p)   ((p) ? (void)0 : _assert(#p, \
    __FILE__, __LINE__))
#endif
```

Die C-Stdlib – ctype.h

- Tests für Zeichenklassen
- **isalpha(c), isdigit(c), islower(c), isupper(c),** …
- **tolower(c), toupper(c)**

- für den erweiterten Zeichensatz: wctype.h
- **iswalpha(c), towupper(c)** etc.

Die C-Stdlib: Header-Dateien (4)

- errno.h: Codes von Systemfehlern
  #define ENOSPC 28 /* No space left on device */

- float.h: Gleitkomma-Wertebereiche
  #define FLT_MAX_EXP +128

- limits.h: ganzzahlige Beschränkungen
  #define INT_MAX +32767

- locale.h: lokale Einstellungen
  #define LC_MONETARY 4

Die C-Stdlib: Header-Dateien (5)

- math.h: mathematische Funktionen
  #define HUGE_VAL _huge_dble

- setjmp.h: erweiterte Sprungfunktionen

- signal.h: Signalbehandlung
  void (*signal(int sig, void (*handler)(int)))(int)

Die C-Stdlib: stdarg.h

- stdarg.h: variable Argumentlisten

```
va_list ap;

va_start(va_list ap, lastarg); /* initialisiert ap; lastarg ist
letzter benannter Parameter */

type va_arg(va_list ap, type); /*jeder Aufruf liefert nächstes
unbenanntes Argument*/

va_end(va_list ap);
```

Die C-Stdlib: Beispiel myprintf


```
#include <stdio.h>
#include <stdarg.h>

extern char *itoa(int, char *, int);
void myprintf(const char *fmt, ...) {
        const char *p;
        va_list argp;
        int i;
        char *s;
        char fmtbuf[256];

        va_start(argp, fmt);
        for(p = fmt; *p != '\0'; p++) {
                if(*p != '%') {    putchar(*p); continue;     }
```

Die C-Stdlib: Beispiel myprintf(2)


```
            switch(*++p) {
                case 'c':       i = va_arg(argp, int);
                                putchar(i);
                                break;
                case 'd':       i = va_arg(argp, int);
                                s = itoa(i, fmtbuf, 10);
                                fputs(s, stdout);
                                break;
                case '%':       putchar('%');
                                break;
                /* hier noch weitere cases einfügen */
            }
        }
va_end(argp);
}
```

Die C-Stdlib: stdlib.h

- Konvertierungen (später)
- Zufallszahlen (**rand**)
- Speicherverwaltung (**malloc, free**)
- Environment-Funktionen (**exit, system**)
- Such- und Sortierfunktionen (**qsort**)
- Integer-Arithmetik (**abs**)

- stddef.h: Typdefinitionen, **NULL** und **errno**

- time.h: Zeit- und Datumsfunktionen

Die C-Stdlib – Erweiterungen (1)

- 1995: Normative Amendment 1 (NA1)

- iso646.h: alternative Operator-Schreibweisen
  z.B. **and** statt **&&, and_eq** statt **&=**
- wctype.h: ctype.h für den erweiterten Zeichensatz
  z.B. **iswalpha()** statt **isalpha()**
- wchar.h: stdio.h/string.h für den erw. Zeichensatz
  z.B. **fgetwc()** statt **fgetc()**

Die C-Stdlib – Erweiterungen (2)

- 1999: C99-Standard

- complex.h: komplexwertige Zahlen (definiert z.B. **const float complex I**)
- fenv.h: Kontrolle der Gleitpunktzahlen-Umgebung
- stdbool.h: **bool** statt **_Bool**; Bezeichner **true**, **false**
- inttypes.h: genauere Integertypen
- stdint.h: Integertypen vorgegebener Breite (z.B. **IntN_t**)
- tgmath.h: typengenerische Mathematikfunktionen

Die C-Stdlib – geht es auch ohne?

- Implementierungen der Standardbibliothek können Code unnötig groß und damit langsam machen
- prinzipiell kann man auch alles selbst schreiben
- Einsatz vor allem im Bereich Embedded Systems
- Möglichkeit der zielgerichteten Optimierung von Codegröße/Laufzeit

stdio - Inhalt

1. C-Dateikonzept
2. Funktionen für Dateioperationen
3. Funktionen für Ein- / Ausgabe

- Dateien als sequentielle Datenströme
- Müssen eröffnet werden

```
FILE *fp;
fp = fopen("dateiname", "w");
```

- FILE ist Struktur mit Informationen über Datenstrom
- stdin, stdout und stderr werden vom Betriebssystem vordefiniert und eröffnet

FILE *fopen(const char *filename, const char *mode)

eröffnet die angegebene Datei

int fclose(FILE *stream)

schließt den Datenstrom; liefert **EOF** bei Fehlern, sonst Null

int fflush(FILE *stream)

ungeschriebene, gepufferte Daten werden geschrieben; liefert **EOF** bei Fehlern, sonst Null

- Formatierte Ausgabe
- Formatierte Eingabe
- Eingabe von Zeichen
- Ausgabe von Zeichen

int fprintf(FILE *stream, const char *format, ...)

wandelt Ausgaben unter Kontrolle von **format** um und schreibt sie in **stream**; Resultat ist Anzahl der geschriebenen Zeichen

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

ist äquivalent zu **fprintf(stdout, format, …)**

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

Ausgabe wird in Zeichenvektor s geschrieben

```
int a=4; char b[]="Vier";
printf("%s = %d \n", b, a);
```

liefert

```
Vier = 4
```

int fscanf(FILE *stream, const char *format, ...)

liest von **stream** unter Kontrolle von **format**, legt Werte in nachfolgenden Argumenten ab, welche Zeiger sein müssen; Resultat ist Anzahl der gelesenen Zeichen

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

ist äquivalent zu **fscanf(stdin, format, …)**

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

Eingabe wird aus Zeichenvektor s gelesen

```
char a[] = "Vier = 4"; char b[5]; int c;
sscanf(a, "%s = %d", b, &c);
```

entspricht

```
b="Vier"; und c=4;
```

int fgetc(FILE *stream)

liefert nächstes Zeichen aus **stream** als **unsigned char** (umgewandelt in **int**), **EOF** bei Fehler oder Dateiende

char *fgets(char *s, int n, FILE *stream)

liest nächste Zeile in **s** ein, aber max. **n-1** Zeichen

int getc(FILE *stream)

ist äquivalent zu **fgetc**, kann aber ein Makro sein

int getchar(void)

ist äquivalent zu **getc(stdin)**

char *gets(char *s)

liest die nächste Zeile von **stdin** in **s** ein

int fputc(int c, FILE *stream)

schreibt **c** (umgewandelt in **unsigned char**) in **stream**; liefert **c** oder **EOF** bei Fehler

int fputs(const char *s, FILE *stream)

schreibt **s** in **stream**, liefert nicht-negativen Wert oder **EOF** bei Fehler

int putc(int c, FILE *stream)

ist äquivalent zu **fputc**, kann aber ein Makro sein

int putchar(int c)

ist äquivalent zu **putc(c, stdout)**

int puts(const char *s)

schreibt **s** und einen Zeilentrenner in **stdout**

1. Datentyp
2. Definition
3. Funktionen aus string.h
4. Stringfunktionen aus stdlib.h
5. Fehlerquellen

Strings in C - Datentyp

- Kein eigener Datentyp
- Array vom Typ char
- ' \0' als Terminierungszeichen

"Hallo Welt" entspricht

'H'	'a'	'l'	'l'	'o'	' '	'W'	'e'	'l'	't'	'\0'

- String aus n Zeichen benötigt n+1 Byte Speicher

- Verschiedene Möglichkeiten der Definition

```
char text1[] = "Hallo";

char text2[] = "Hal" "lo";

char text3[] = {'H', 'a', 'l', 'l', 'o','\0'};

char *text4 = "Hallo";

char text5[6] = "Hallo";

char *text6; text6 = (char *) malloc(6); text6 = "Hallo";
```

- Variablen s und t sind vom Typ char *
- Parameter cs und ct sind vom Typ const char *
- Parameter n hat Typ size_t
- c ist ein int-Wert, der in char umgewandelt wird

Manipulation

char *strcpy(s,ct)	Zeichenkette **ct** in Vektor **s** kopieren
char *strncpy(s,ct,n)	höchstens **n** Zeichen aus **ct** in **s** kopieren
char *strcat(s,ct)	Zeichenkette **ct** hinten an die Zeichenkette **s** anfügen;
char *strncat(s,ct,n)	höchstens **n** Zeichen von **ct** hinten an die Zeichenkette **s** anfügen

Vergleich

int strcmp(cs,ct)	liefert <0 wenn **cs<ct**, 0 wenn **cs==ct**,oder >0, wenn **cs>ct**
int strncmp(cs,ct,n)	höchstens **n** Zeichen vergleichen

Suche

char *strchr(cs,c)	liefert Zeiger auf das erste **c** in **cs**
char *strrchr(cs,c)	liefert Zeiger auf das letzte **c** in **cs**
size_t strspn(cs,ct)	Anzahl der Zeichen aus **ct** am Anfang von **cs**
size_t strcspn(cs,ct)	Anzahl der Zeichen *nicht* aus **ct** am Anfang von **cs**

```
strspn("Hallo", "Hoa"); liefert 2
strcspn("Hallo", "ost"); liefert 4
```

char *strpbrk(cs,ct) Position in **cs**, an der erstmals ein Zeichen aus **ct** vorkommt

char *strstr(cs,ct) Zeiger auf erste Kopie von **ct** in **cs**

Sonstige

size_t strlen(cs) Länge von **cs** (ohne **'\0'**)

```
strlen("Hallo"); liefert 5
```

char *strerror(n) Zeichenkette, die Fehler **n** beschreibt

```
strerror(2); liefert "No such file or directory"
```

char *strtok(s,ct) liefert Zeichenfolgen aus **s** die durch Zeichen aus **ct** getrennt sind, bei weiteren Aufrufen **NULL** statt **s** verwenden

```
char s[]="Hallo-Welt_Programm";
char ct[]="-_";

printf("%s\n",strtok(s,ct));
printf("%s\n", strtok(NULL,ct));
printf("%s\n", strtok(NULL,ct));
```

liefert

```
Hallo
Welt
Programm
```

Konvertierung

double strtod(const char *s, char **endp)

wandelt Zeichenkette **s** in **double** um, Zwischenraum am Anfang wird ignoriert; speichert Zeiger auf nicht umgewandelten Rest bei ***endp**

long strtol(const char *s, char **endp, int base)

wandelt in **long** um, **base** ist Basis der Eingabe

unsigned long strtoul(const char *s, char **endp; int base)

wandelt in **unsigned long** um

Strings in C – stdlib.h

double atof(const char *s)	äquivalent zu **strtod(s, (char**) NULL)**
int atoi(const char *s)	äquivalent zu **(int)strtol(s, (char**) NULL, 10)**
long atol(const char *s)	äquivalent zu **strtol(s, (char**) NULL, 10)**

```
char s[]="   2.4568abc";
char t[]="9876def";
double a; int b;

a=atof(s);
b=atoi(t);
```

entspricht

a=2.4568; und b=9876;

Strings in C - Fehlerquellen


```
char text1[] = "Hallo";
text1[strlen(text1)] = 'i';                    /* '\0' wird überschrieben */
printf("text1=%s" , text1);

char text2[] = "langer Text";
char text3[] = "kurz";
strcpy(text3 , text2);                         /* text3 ist zu kurz, fremder
                                                  Speicher wird überschrieben */

char text4[] = "langer Text";
char text5[] = "kurz";
strncpy(text5 , text4 , strlen(text5));        /* Länge von text5 wird berücksichtigt */
```

Inhalt

1. Was ist Posix?
2. Der POSIX 1003 Standard
3. Was gibt es noch?

Was ist Posix?

- POSIX = Portable Operating System Interface for UniX
- Standard, der die Schnittstelle zwischen einer Anwendung und dem Unix-Betriebssystem definiert
- 1990 (P1003.1) bzw. 1992 (P1003.2) von der IEEE entwickelt
- Aktuelle Version: IEEE Std 1003, 2004
- POSIX-konforme Programme müssen diese Schnittstelle einhalten

Was ist Posix?

- Eigentlich wird die Schnittstelle zur Bibliothek spezifiziert, nicht die der Systemaufrufe an sich
- Schnittmenge der Features aus System V und BSD
- Sowohl für Anwendungs- als auch für Betriebssystemprogrammierer gleichermaßen verständlich

Der POSIX 1003 Standard

- Im Prinzip sprachunabhängig
- In C bspw. durch eine spezielle POSIX-Bibliothek realisiert
- POSIX-Standardkomponente

P1003.1 Betriebssystemkern und C-Bibliotheken

P1003.2 Shell und Kommandos

P1003.3 POSIX Test Suite

P1003.4 Realzeiterweiterungen

P1003.5 Sprachanbindung an ADA

P1003.6 Systemsicherheit

P1003.x Systemadministration

P1003.1 Prozessmanagement

pid = fork()

Kindprozess als Abbild des Vaters erzeugen

pid = waitpid(pid, &statloc, opts)

Auf Beendigung des Kindes warten

s = execve(name, argv, envp)

Speicherwerte des Prozesses ersetzen

exit(status)

Prozess beenden und Status zurückliefern

s = kill(pid, sig)

Signal an Prozess schicken

- Bsp.: Eine stark vereinfachte Shell

```
while (TRUE) {
        type_prompt();
        read_command (command, params);

        pid = fork ();
        if (pid < 0) {
                printf(„Unable to fork“);
                continue;    }

        if (pid != 0) {
                waitpid (-1, &status, 0);
        } else {
                execve (command, params, 0);
        }               }
```

- POSIX-Thread-Aufrufe

pthread_create

Neuen Thread im Adressraum des Aufrufers erzeugen

pthread_exit

Aufrufenden Thread beenden

pthread_join

Auf die Beendigung eines Threads warten

fd = creat(name, mode)

erzeugt eine Datei

fd = open(le, how, ...)

Datei zum Lesen, Schreiben öffnen

s = close(fd)

Offene Datei schließen

n = read(fd, buffer, nbytes)

Daten einer Datei in Puffer lesen

n = write(fd, buffer, nbytes)

Daten von Puffer in datei speichern

position = lseek(fd, offset, whence)

Dateilesezeiger bewegen

s = stat(name, &buf)

Statusinformationen einer Datei holen

s = fstat(fd, &buf)

Statusinformationen einer Datei holen

s = pipe(&fd[0])

Pipe erzeugen

s = fcntl(fd, cmd, ...)

Dateisperrung und andere Optionen

Bsp.: Kopierprogramm


```
#include <sys/types.h>
#include <fcntl.h>
#include <stdlib.h>
#include <unistd.h>

int main (int argc, char *argv[])
{                       int in_fd, out_fd. rd_count, wt_count;
                        char buffer[BUF_SIZE];
                        if (argc != 3) exit (1);
                        in_fd = open (argv[1], O_RDONLY);
                        if (in_fd < 0) exit (2);
                        out_fd = creat (argv[2], OUTPUT_MODE);
                        if (out_fd < 0) exit (3);
                        while (TRUE) {
                                        rd_count = read (in_fd, buffer, BUF_SIZE);
                                        if (rd_count <= 0) break;
                                        wt_count = write (out_f, buffer, rd_count);
                                        if (wt_count <= 0) exit (4);                    }
                        close (in_fd);
                        close (out_fd);
                        if (rd_count == 0) exit (0);
                        else exit (5);
}
```

P1003.1 Verzeichnismanagement

s = mkdir(path, mode)

Neues Verzeichnis erzeugen

s = rmdir(path)

Verzeichnis entfernen

s = link(oldpath, newpath)

Link auf bestehende Datei erzeugen

s = unlink(path)

Link löschen

P1003.1 Verzeichnismanagement

s = chdir(path)

Aktuelles Verzeichnis wechseln

dir = opendir(path)

Verzeichnis zum Lesen öffnen

s = closedir(dir)

Verzeichnis schließen

dirent = readdir(dir)

Lesen eines Verzeichniseintrags

rewinddir(dir)

Zurückspulen und neu lesen

s = chmod(path, mode)
Schutzbits der Datei ändern

s = access(path, mode)
Aktuelles Verzeichnis wechseln

uid = getuid( )
Echte UID erfragen

uid = geteuid( )
Effektive UID erfragen

gid = getgid( )
Echte GID erfragen

gid = getegid( )
Effektive GID erfragen

s = chown(path, owner, group)
Eigentümer und Gruppe ändern

s = setuid(uid)
Setzen der UID

s = setgid(gid)
Setzen der GID

P1003.2 Unix-Hilfsprogramme

cat	Ausgabe von Dateien auf Standardausgabe
chmod	Schutzbits für Dateien setzen
cp	Ein oder mehrere Dateien kopieren
cut	Spalten einer Datei abschneiden
grep	Muster in einer Datei suchen
head	Erste Zeilen einer Datei ausgeben
ls	Verzeichnis anzeigen
make	Dateien übersetzen und ein Programm erzeugen
mkdir	Verzeichnis erstellen
od	Oktale Ausgabe einer Datei
paste	Spalten eines Textes in eine Datei schreiben
pr	Datei für den Druck vorbereiten
rm	Ein oder mehrere Dateien löschen
rmdir	Leeres Verzeichnis löschen
sort	Zeilen einer Datei alphabetisch sortieren
tail	Letzte Zeilen einer Datei ausgeben
tr	Übersetzung verschiedener Kodierungen

- *Beispiele*
- cp quelle ziel
- head -20 datei
- head 20 datei
- ls *.c
- ls x.c y.c z.c
- ls [ape]*
- sort <in >aus
- sort <in >temp; head -30 temp; rm temp
- sort <in | head -30
- grep ter *.t | sort | head -20 | tail -5 foo
- wc –l <a >b &
- sort <x | head &

Was gibt es noch?

- Berkeley Unix
- SVID = System V Interface Description

- XPG = The X/Open Portability Guide
- Portabilität für Systemaufrufe und Bibliotheken, Kommandos und Dienstprogramme, Sprachen und Data Management

Vielen Dank

für eure Aufmerksamkeit!