Einführung in C - Introduction to C

8. C Standard Library

Prof. Dr. Eckhard Kruse

DHBW Mannheim

Some standard functions

Example

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

scanf("%d",&i);
printf("%d", i);
…

…
int *ptr=malloc(100*sizeof(int));
free(ptr);
…
double value=sin(0.5);
…
FILE *file=fopen("test.mp3","rb");
…
strcpy(s->name,new_name);
…
```

Without standard library functions there is not much you can do in C.

C standard library

Definition

The **C standard library** is a standardized collection of header files and library routines providing common definition and operations, such as input/output, string handling and access to operating system functionality.

<assert.h>	Diagnostics
<ctype.h>	Character Class Tests
<errno.h>	Error Codes Reported by (Some) Library Functions
<float.h>	Implementation-defined Floating-Point Limits
<limits.h>	Implementation-defined Limits
<locale.h>	Locale-specific Information
<math.h>	Mathematical Functions
<setjmp.h>	Non-local Jumps
<signal.h>	Signals
<stdarg.h>	Variable Argument Lists
<stddef.h>	Definitions of General Use
<stdio.h>	Input and Output
<stdlib.h>	Utility functions
<string.h>	String functions
<time.h>	Time and Date functions

When you need something, which probably would be of use for many programmers, look for it in the standard library!

printf: formatting

optional:
minimum width:
if '*', value taken from next argument (must be int).
. separates width from precision
precision:
%s: maximum chars to be printed
%eEf: digits after decimal point
%gG: significant digits,
integer: minimum number of digits to be printed.
if '*', value taken from next argument (must be int).

optional length modifier:

h	short or unsigned short
l	long or unsigned long
L	long double

```
printf("%3.1f %+ld", f, i);
```

optional flag

-	left adjust
+	always sign
space	space if no sign
0	zero pad
#	Depends on conversion char: #o: first digit will be zero, #x #X: prefix 0x or 0X #eEfgG: always decimal point #gG: trailing zeros not removed

conversion character

d,i	int argument, printed in signed decimal notation
o	int argument, printed in unsigned octal notation
x,X	int argument, printed in unsigned hexadecimal notation
u	int argument, printed in unsigned decimal notation
c	int argument, printed as single character
s	char* argument
f	double argument, printed with format [-]mmm.ddd
e,E	double arg., printed with format [-]m.dddddd(e|E)(+|-)xx
g,G	double argument
p	void* argument, printed as pointer
n	int* argument : the number of characters written to this point is written into argument
%	no argument; prints %

sprintf, sscanf

sprintf and **sscanf** work similar as printf and scanf, but instead of using the console for input/output, data is printed into a string (sprintf), or read from a string (sscanf). Thus, these functions are convenient e.g. to transform number types into strings and vice versa.

*int sprintf (char * str, const char * format, ...);*
*int sscanf (const char * str, const char * format, ...);*

```
#include <stdio.h>

char str[20];
int i=1234, j;

sprintf(str,"%d",i);    // → str="1234"
sscanf(str,"%d",&j);    // j=1234
```

Other way to convert strings to int/float:
*int atoi(const char * str);*
*float atof (const char * str);*

<string.h> String handling functions

char* strcpy(char* s, const char* ct); Copies ct to s including terminating NUL and returns s.

char* strncpy(char* s, const char* ct, size_t n); Copies at most n characters of ct to s. Pads with NUL characters if ct is of length less than n. Note that this may leave s without NUL-termination. Return s.

char* strcat(char* s, const char* ct); Concatenate ct to s and return s.

char* strncat(char* s, const char* ct, size_t n); Concatenate at most n characters of ct to s. NUL-terminates s and return it.

int strcmp(const char* cs, const char* ct); Compares cs with ct, returning negative value if cs<ct, zero if cs==ct, positive value if cs>ct.

int strncmp(const char* cs, const char* ct, size_t n); Compares at most (the first) n characters of cs and ct, returning negative value if cs<ct, zero if cs==ct, positive value if cs>ct.

char* strchr(const char* cs, int c); Returns pointer to first occurrence of c in cs, or NULL if not found.

char* strrchr(const char* cs, int c); Returns pointer to last occurrence of c in cs, or NULL if not found.

char* strpbrk(const char* cs, const char* ct); Returns pointer to first occurrence in cs of any character of ct, or NULL if none is found.

char* strstr(const char* cs, const char* ct); Returns pointer to first occurrence of ct within cs, or NULL if none is found.

size_t strlen(const char* cs); Returns length of cs.

…and some more!

Reliability and security risks

In C, especially working with pointers, arrays and strings always bears the risk of accessing wrong memory areas. This is a security and reliability risk:

```
char str[100];
scanf("%s", str);        // This simple function is insecure and unreliable
scanf("%99s", str);      // This is better
```

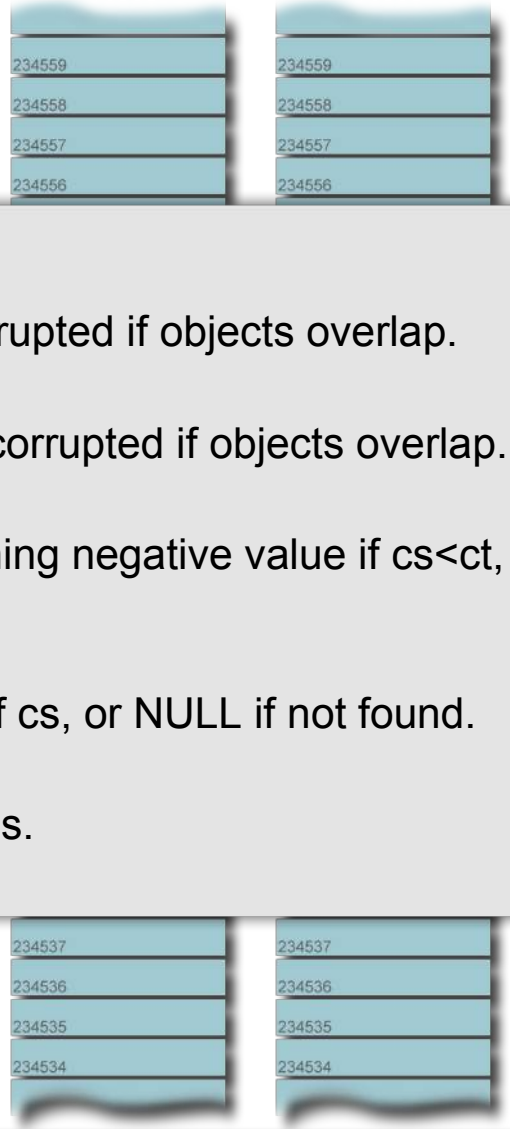
Many functions you find in two (or more) versions, e.g.:

- *char* strcpy(char* s, const char* ct);*
- *char* strncpy(char* s, const char* ct, size_t n);* ← copies max n chars (→ no 0-termination!), if less, fills with 0 char.

C11:

- *errno_t strncpy_s(char *restrict dest, rsize_t destsz,*
  *const char *restrict src, rsize_t count);*

Memory handling (string.h)



void* memcpy(void* s, const void* ct, size_t n);
Copies n characters from ct to s and returns s. s may be corrupted if objects overlap.

void* memmove(void* s, const void* ct, size_t n);
Copies n characters from ct to s and returns s. s will not be corrupted if objects overlap.

int memcmp(const void* cs, const void* ct, size_t n);
Compares at most (the first) n characters of cs and ct, returning negative value if cs<ct, zero if cs==ct, positive value if cs>ct.

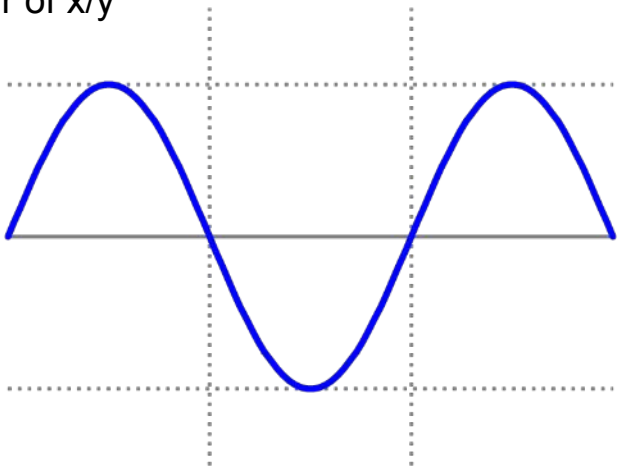
void* memchr(const void* cs, int c, size_t n);
Returns pointer to first occurrence of c in first n characters of cs, or NULL if not found.

void* memset(void* s, int c, size_t n);
Replaces each of the first n characters of s by c and returns s.

<math.h> Functions

HUGE_VAL	magnitude returned (with correct sign) on overflow error
double exp(double x);	exponential of x
double log(double x);	natural logarithm of x
double log10(double x);	base-10 logarithm of x
double pow(double x, double y);	x raised to power y
double sqrt(double x);	square root of x
double ceil(double x);	smallest integer not less than x
double floor(double x);	largest integer not greater than x
double fabs(double x);	absolute value of x
double ldexp(double x, int n);	x times 2 to the power n
double fmod(double x, double y);	floating-point remainder of x/y
double sin(double x);	sine of x
double cos(double x);	cosine of x
double tan(double x);	tangent of x
double asin(double x);	arc-sine of x
double acos(double x);	arc-cosine of x
double atan(double x);	arc-tangent of x
double atan2(double y, double x);	arc-tangent of y/x
double sinh(double x);	hyperbolic sine of x
double cosh(double x);	hyperbolic cosine of x
double tanh(double x);	hyperbolic tangent of x

....and some others



<stdio.h> Some file functions

- **FILE* fopen(const char* filename, const char* mode);** Opens file *filename* and returns a stream, or NULL on failure. *mode* may be one of the following for text files ("r"=text reading, "w"=text writing, "a" , "r+", "w+", "a+"... or one of those strings with "b" added after the first character, for binary files.
- **int fclose(FILE* stream);** Closes *stream* (after flushing, if output stream). Returns EOF on error, zero otherwise.
- **int remove(const char* filename);** Removes specified file. Returns non-zero on failure.
- **int rename(const char* oldname, const char* newname);** Change name of file oldname to newname.
- **size_t fread(void* ptr, size_t size, size_t nobj, FILE* stream);** Read (at most) nobj objects of size size from stream stream into ptr and returns number of objects read. (feof and ferror can be used to check status.)
- **size_t fwrite(const void* ptr, size_t size, size_t nobj, FILE* stream);**
  Writes to *stream*, *nobj* objects of size *size* from array ptr.
  Returns number of objects written.
- **int feof(FILE* stream);**
  Returns non-zero if end-of-file indicator is set for stream stream.

  … and some others

File Analyzer

Exercise

`file-analyzer.c`

801

C-program-generator

Exercise

C-program-generator.c

802

Other libraries

To get access to advanced functionality, e.g. graphical output, handling of windows etc. additional libraries are required. Most often these depend on the operating system and are not standardized, but there are also some cross-platform libraries.

Some examples of cross-platform libraries:

- **Operating system:**
  POSIX (portable operating system interface):
  https://standards.ieee.org/standard/1003_1-2017.html
- **UI/Multimedia:**
  OpenGL: http://www.opengl.org/
  SDL Simple Directmedia layer: http://www.libsdl.org/
  Xlib: http://en.wikipedia.org/wiki/Xlib
  Cairo: http://www.cairographics.org/
  GUI (C++): ...
- **more e.g.:** http://www.freebyte.com/programming/cross_platform/

and of course countless platform-specific libraries...

SDL example

Exercise

SDL-example.c

803