I2C

Inhalt

Strings

Functions from strings.h:

Common string functions

```
strlen(s1); // Returns length. Relies on null termination

strcpy(s1, s2); //Copy string from s2 to s1. Does not check if s1 is big enough
strncpy(s1, s2, n); //Copy n characters from s2 to s1

strcat(s1, s2); // Concatenate s2 with s1. Does not check if s1 is big enough
strncat(s1, s2, n); // Concatenate n chars of s2 with s1

strcmp(s1, s2); // Concatenate s2 with s1. Return value is like .compareTo from java (-1, 0, 1)
strncmp(s1, s2, n); // Concatenate n chars of s2 with s1

strchr(str, c); // search char c in string str
strstr(s1, s2); // search substring s2 string s1

strtok(s1, s2); // legit die unlogischti Funktion ever. Sött strings ufsplitte aber zouberet aube chli strings usem nüüt füre wende NULL übergisch.

atof() // returns double value from string
atoi() // returns integer value from string
atol() // returns long value from string
atoll() // returns long long value from string

/* self explaining */
islower();
isuppeer();
isalpha();
isalnum();
iscntrol();
isprint();
isgraph(); // isprintable but without space
isdigit();
isxdigit();
isblank();
isspace();
ispunct();

toupper()
tolower()
```

Pointers

Void pointers

- mostly used for function paramaters, function can accept any data type.
- must first be explicitly cast to a data type before use.

Dynamic Memory

If you want to allocate memory on an address without creating a variable, you can manually allocate it with some functions.
For example if you want a pointer to an address with some storage without first creating a variable, which allocates the needed storage.

File handling

Opening files

opening files ist possible with the `fopen()` function form `stdio.h`. The first argument is a char pointer for the file adress, the second one for the opening mode. The function returns FILE pointer to the first byte of the file or NULL if the file could not be read correctly.

file open example

```
char *path = "./example.txt";
FILE *f = fopen(path, "w");
```

The options for opening files are as follows:

option char	description
w	open for write operations. Existing content will be discarded
a	appends to a file
r	open for reading operations
w+	reading and writing. First truncating the file to zero length if it exists or creating the file if it doesn't exist.
a+	reading and writing but writing does append to file

option char	description
r+	Open a text file for update (reading and writing)

> **Warning**
>
> always close the file after usage with the `fclose()` function

Renaming files

renaming a file is as simple as follows:

```
rename("old_name", "new_name");
```

the function returns an error code. Zero for success non-zero for failure.

Deleting files

files can be deleted with

```
remove("file_name");
```

Reading from a file

First of all, all reading is done from the file pointer. When opening a file it points to the first byte of the file. But if you read some data the value changes resp. gets incremented. If you want to set it to the beginning again you can either use

```
rewind(fptr);
```

or

```
fseek(fptr, 0, SEEK_SET);
```

Reading data from a text file can be done with the `fgetc(FILE* fptr)` function, which takes a pointer to a file which is opened in reading mode and returns a char.

reading a character

```
char myChar = fgetc(myFile);
```

returns `EOF` if character could not be read correctly.

for reading a string the `fgets(char* str, int nchars, FILE* fptr)` function can be used, where the arguments are the string (char *) where it is written to in memory, the length of the string to read and the file. It reads either until a whitespace is detected or nchars are read. If read correctly it returns the pointer to the str otherwise NULL.

```
char* sptr = fgets(myBuffer, 50, myFile);
```

To sum it up:

```
#include <stdio.h>

int main(){
   FILE* fp;
   char c;
   char str[60];

   /* opening file for reading */
   fp = fopen("file.txt", "r");

   /* always check if fp is NULL */
   if(fp == NULL){
      /* writing to stderr */
      perror("Error opening file");
      return(-1);
   }

   /* Read 1 char */
   if((c = fgetc(fp)) != EOF){
      /* writing first character of file to stdout */
      printf("%c\n", c);
   }


   rewind(fp); // set cursor to beginning of file, because it isn't anymore after fgetc()

   /* Read 60 characters or until '\0' is detected */
   if((fgets(str, 60, fp)) != NULL){
      printf("%s\n", str);
   }

   fclose(fp);
   fp = NULL;

   return 0;
} // end main
```

Reading formatted content

Reading formatted context from a text file can be done with

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

The return value is the number of items successfully read. The format is similar to the syntax used in `scanf()`.

```c
#include <stdio.h>

int main(){
   FILE* fp = fopen("file.txt", "r");
   char str1[10], str2[10], str3[10], str4[10];

   if(fp == NULL){
      return -1;
   }

   /* If the file has 3 words delimited by a whitespace
   it will write them into str1, str2 and str3 */
   fscanf(fp, "%s %s %s %s", str1, str2, str3, str4);

   printf("String1: %s\n", str1);
   printf("String2: %s\n", str2);
   printf("String3: %s\n", str3);
   printf("String4: %s\n", str4);

   fclose(fp);

   return 0;
} // end of main
```

If the file content is:

```
Text Only

Hello, how are you?
```

the output would be:

```
Text Only

String1: Hello,
String2: how
String3: are
String4: you?
```

Writing to a file

If you want to write only one character:

```c
fputc(int c, FILE* pfile);
```

writing a string:

```
fputs(const char *str, FILE* pfile);
```

this will write the string given to the file

Write formatted output to file

```
fprintf(FILE *stream, const char* format, ...);
```

Positioning in file

for knowing where in the file you are currently you can use:

```
long ftell(FILE *pfile);  // returns a long integer which is the offset of the starting point of the file
long fgetpos(FILE *pfile, fpos_t *position);  // stores the position in the fpos_t pointer. Is meant to be used in fsetpos()
```

```
FILE* fp = fopen("file.txt", "w+");

/* do sth with the file here */

long pos = ftell(fp);

fpos_t here;
fgetpos(fp, &here);
```

For setting the position you can use `fseek(FILE *fp, long offset, int origin);` the second offset determines the offset in bytes based on the origin given in the 3rd argument. Valid origin values are:

- SEEK_SET - beginning of file
- SEEK_CUR - current position
- SEEK_END - end of file

```
FILE* fp = fopen("file.txt", "w+");

/* do sth with the file here */

seek(fp, 0, SEEK_SET); // set cursor to beginning of file
```

Or you can use `fsetpos(FILE* pfile, fpost_t *position)` where the struct is of the type you get from `fgetpos()`. `fsetpos()` can bring you only to a position you have been before. `fseek()` can bring you anywhere in the file.