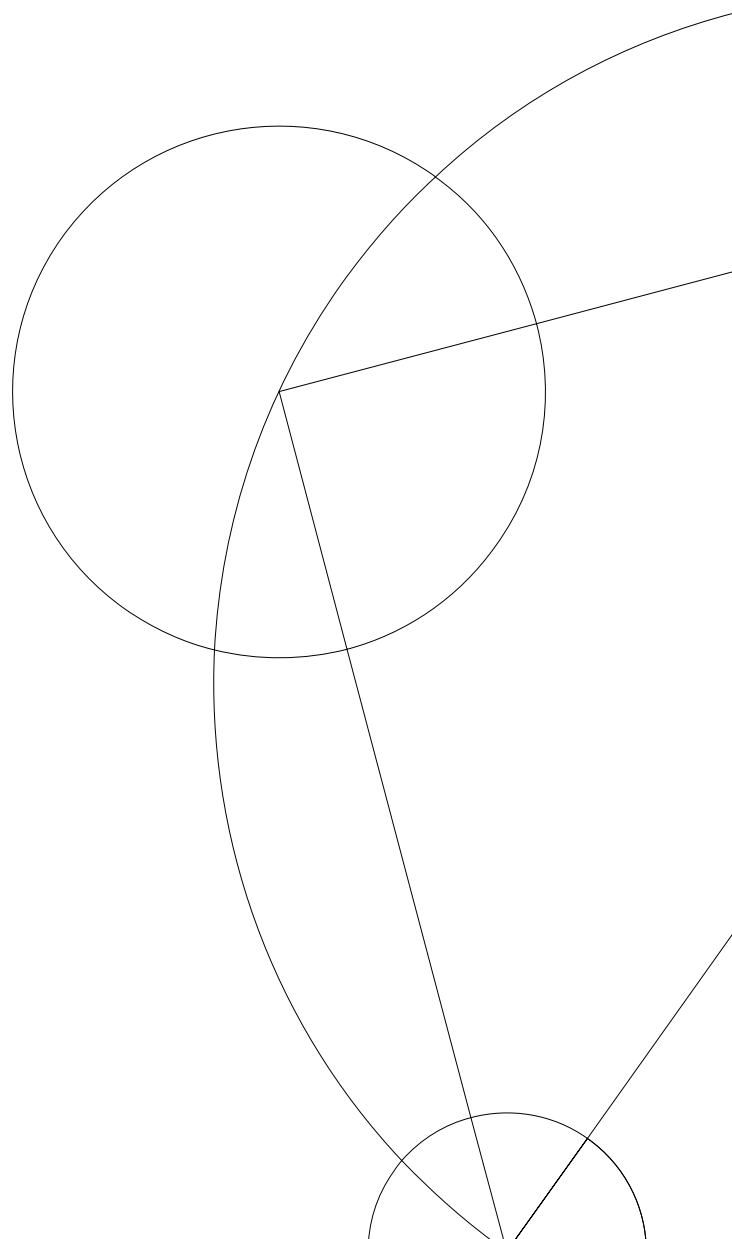




A0 - CompSys

Mads Kronborg - `xlq446`
Niklas Lohmann - `Jgs746`
Christian Baun - `rbp111`

September 9, 2018



1 Compiling and running the code

To compile the program you should be in the correct directory, which in our case will be `/src`.

When `src` is your current working directory, do *make* in this folder to compile an executable.

```
niklas@DESKTOP-UL37P5D:/mnt/c/Compsys/Uge 1$ ls
src
niklas@DESKTOP-UL37P5D:/mnt/c/Compsys/Uge 1$ cd src/
niklas@DESKTOP-UL37P5D:/mnt/c/Compsys/Uge 1/src$ make
gcc -std=c11 -Wall -Werror -Wextra -pedantic -g -o file file.c
niklas@DESKTOP-UL37P5D:/mnt/c/Compsys/Uge 1/src$
```

Figure 1: Making `src` the current working directory and running `make`

With the creation of an executable you can now use it to test files using `./file ascii`. Which in this case will return the filetypes of `ascii`, which is an `ascii` file.

```
niklas@DESKTOP-UL37P5D:/mnt/c/Compsys/Uge 1/src$ ./file ascii
ascii: ASCII text
```

Figure 2: Using `file` on `ascii`

In `/src` we also have the provided `test.sh` file which can be used for testing. You run it with `bash test.sh` which test the different cases we have given it.

```
niklas@DESKTOP-UL37P5D:/mnt/c/Compsys/Uge 1/src$ bash test.sh
make: 'file' is up to date.
Generating a test_files directory..
Generating test files..
Running the tests..
>>> Testing test_files/ascii1.input..
>>> Success :-)
>>> Testing test_files/ascii10.input..
>>> Success :-)
>>> Testing test_files/ascii11.input..
>>> Success :-)
>>> Testing test_files/ascii2.input..
>>> Success :-)
```

Figure 3: running `bash test.sh`

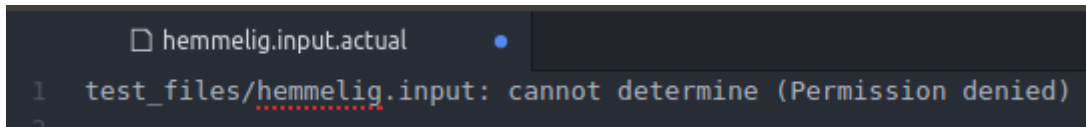
1.1 Testing

The test script `test.sh` was supplied with more than 30 tests.

The tests were created using an ASCII table (Web resource: <https://www.ascii-code.com/>) and keeping in mind the ASCII definition from the assignment text. The printable characters are taken from this web source: <https://codegolf.stackexchange.com/questions/85088/print-the-ascii-printable-character-set>. Both control characters and printable characters are tested, but none above char number 127, since these return ISO-8859 text with `file(1)`, thus fail when compared to the data file format in `file.c`.

The only test not included in the *test.sh*, is the test for a hidden/secret file, that we do not have permission to read. We haven't been able to automate this the same way as the other tests, but we can verify by the contents of the file, that the program returns the error we want it to.

I.e. running our program on a file which we do not have the permission to read, the resulting file contains the following:

A screenshot of a terminal window with a dark background. The title bar at the top reads 'hemmelig.input.actual'. The terminal content shows a command prompt followed by the command 'test_files/hemmelig.input: cannot determine (Permission denied)'. The command is highlighted in red, and the error message is in white. There are some red dots below the command.

Which is the API wished for in the assignment text.

2 Discuss the non-trivial parts of your implementation and your design decisions, if any.

Our implementation relies heavily on the "stdio.h" library. This library contains many tools and functions, but the ones that are relevant to our solution, **fopen**, **fseek**, **ftell** and **fread** come to mind.

fopen is pretty self-explanatory, since all it does is opening the file.

fseek is used in combination with **ftell** to tell us how big the file is, so we can check if the file is empty, and if not, how much space we will need to work with the file.

And **fread** is used to read the file. It does so by taking four arguments;

- ptr - a pointer to a block of memory with a minimum size of size*nmemb bytes.
- size - an int specifying the size in bytes of each element to be read.
- nmemb - another int, this time to specify the number of elements, each one with a size of the 2nd argument in bytes.
- stream - This is the pointer to a FILE object that specifies an input stream.

And once we have the file prepared and opened, we start checking every character in the file, so we can deduce whether the file is either, ascii, data or empty.

For design decisions, our implementation is pretty comparable to the assignment and the hints given in it. We use an enum class to specify our file types, and we make use of the "errno.h" library, to give out the correct error messages when the code is run.

3 Disambiguate any ambiguities you might have found in the assignment.