C-Lessons

Variables

Lecturers: Mirko Jantschke, Pascal Scholz

26. November 2018

Contents

Do not reinvent the wheel

Useful headers

Man page

Do not reinvent the wheel

The Hitchhiker's Guide to the standard library

Many of the problems you will be working on have already been solved.

These solution are provided in *libraries*. Prefer them over your own code!

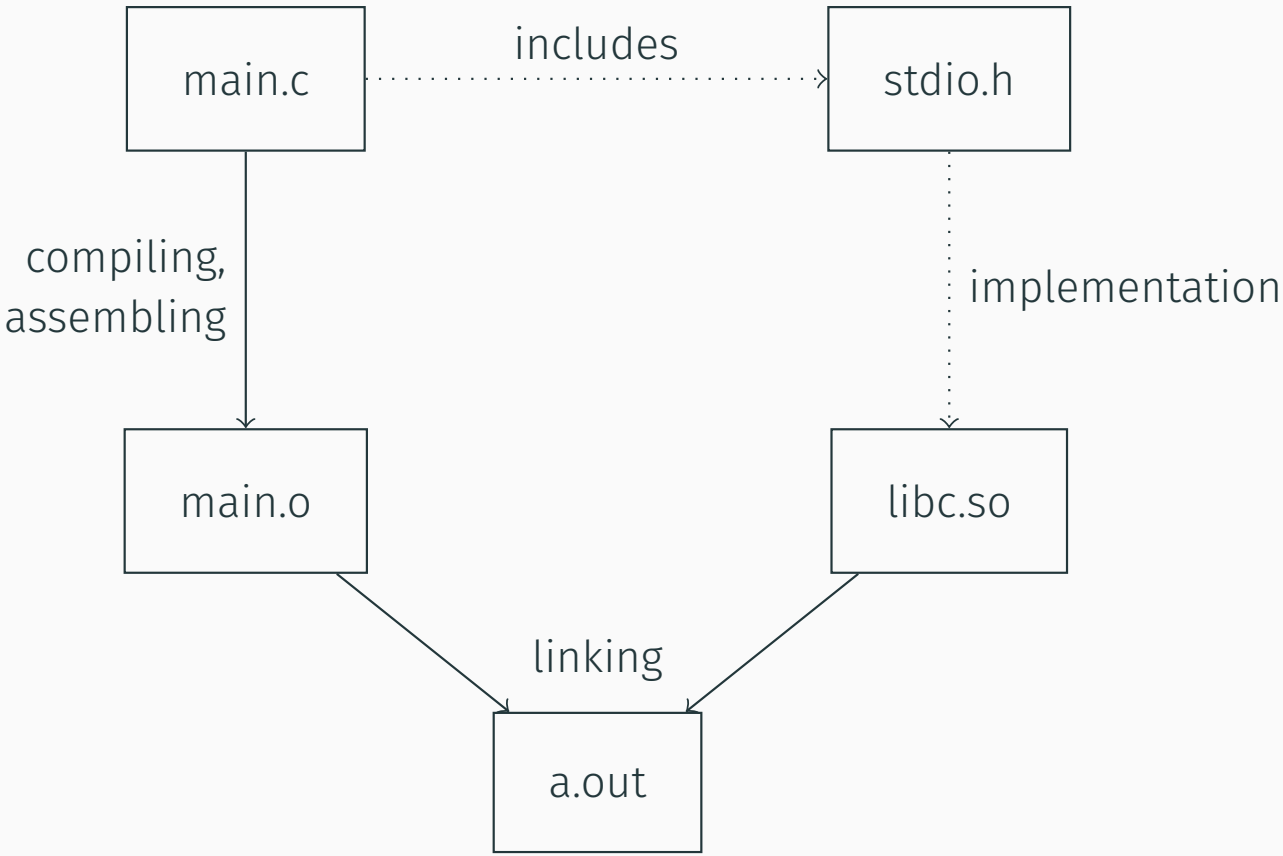
- they are safer
- they are more efficient
- you do not have to do everything by yourself

We have already used the **C standard library** for i/o:

```
#include <stdio.h>  /* Anyone remember this? */
```

This statement includes the *header* file **stdio.h**.
After compilation, *gcc* will link your program with the C library.

Linking happens



Useful headers

assert.h

- Contains the *assert()* macro, witch evaluates the truth value of an expression
- If it is true, nothing happens
- Else the program aborts and an error message is printed

→ useful to avoid undefined behaviour / worse errors at runtime

We can also use it if we just want to test things:

```
unsigned int input;
printf("Enter a one-digit decimal number:\n");
scanf("%d", &input);
assert(input < 10);
```

math.h

- Declares a lot of mathematical functions
- Finally you are able to calculate square roots, logarithms, etc.
- Most of those functions have *double* arguments and return values

If you use functions from *math.h*, add the *-lm* as the **last** option to *gcc* to avoid errors:

```
gcc main.c -lm
```

stdio.h

- Declares the basic functions to read and write data
- You know *printf()* and *scanf()*, but there is more:
- Characters, unprocessed and formatted strings
- Command line I/O and file access
- Many functions for high-level file management

As an example, *puts()* can be used instead of *printf()* if you have a basic string without placeholders - '\n' is added automatically:

```
puts("Hello World!");
/* Equivalent to printf("Hello World!\n") */
```

stdlib.h

This probably is the most powerful header providing various different functionalities. Here is just an excerpt:

- *EXIT_SUCCESS* and *EXIT_FAILURE* constants as an alternative to returning *0* or *1* at the end of *main()*
- Alternative ways to exit the program
- Generation of pseudo-random numbers
- Search and sorting function
- Dynamic memory management

...and more things you have not even heard of (yet)

string.h

string.h

Wait! Strings?

string.h

Wait! Strings?

Yes, there are strings in C.
They are just handled differently from what you would expect.

string.h is crucial if you want to work with C strings seriously.
We will use some of the functions declared there in later lessons.

time.h

- Data types to store different time formats
- Functions to get the calendar and cpu time
- Functions to format time values
- Functions to measure and calculate time differences

Handling time usually is quite complicated, but with the help of *time.h* it gets a lot easier.

Measure the execution time of your programs to see how efficient they are!

Man page

Documentation

Learning all the library functions is way less effective than knowing where to look them up quickly.
Man page is a Unix tool containing documentation of programs, system calls and libraries - such as the C standard library.

To access a certain man page, just type:

```
$ man page
```

Example for *printf()*:

```
$ man printf
```

However, this describes the shell command *printf*.

Effective use of *man*

Man has many sections, library functions are in #3.
Write the section number between *man* and the page:

```
$ man 3 printf
```

To get all pages *printf* occurs in, use the *-k* option:

```
$ man -k printf
```

If you need more information on *man* - it has its own man page:

```
$ man man
```