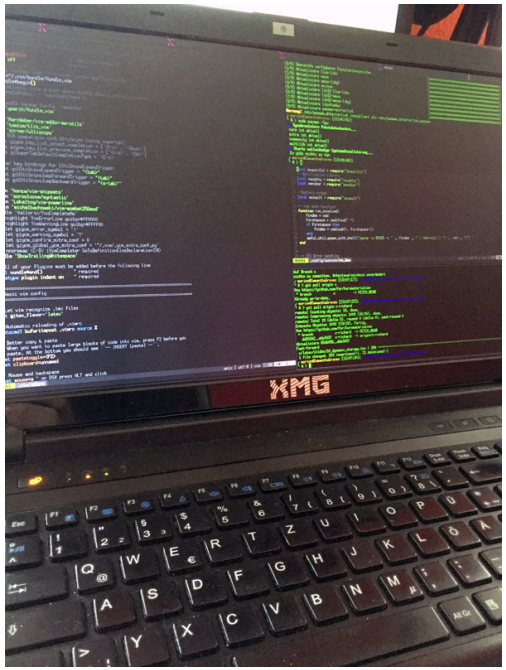
C introduction

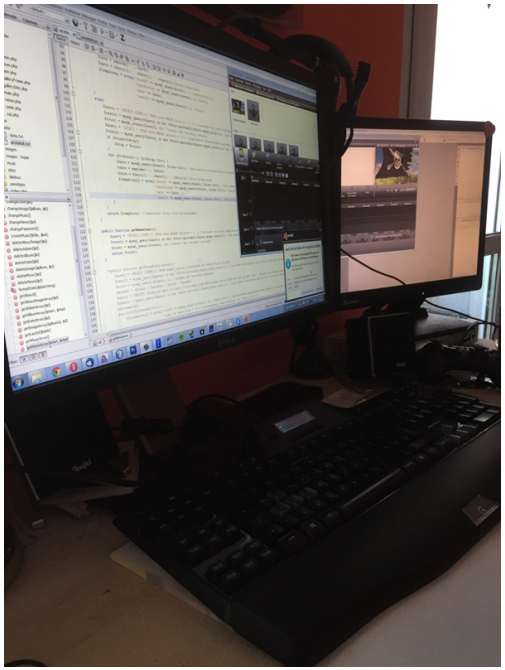
Basic program structure

Contents

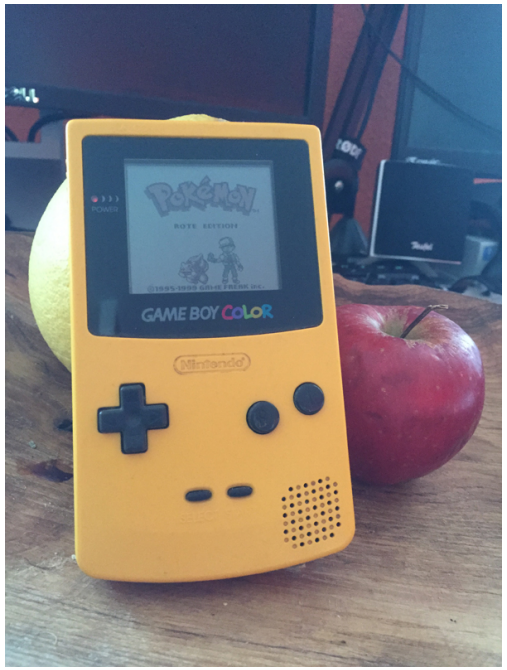
OS's you may use



Linux

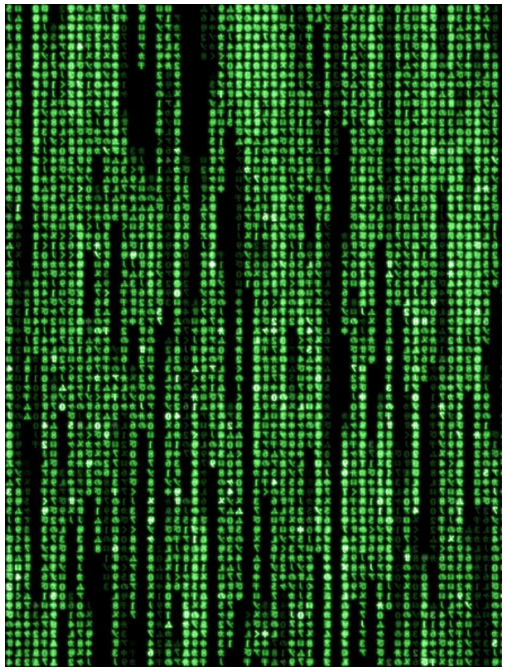


Windows



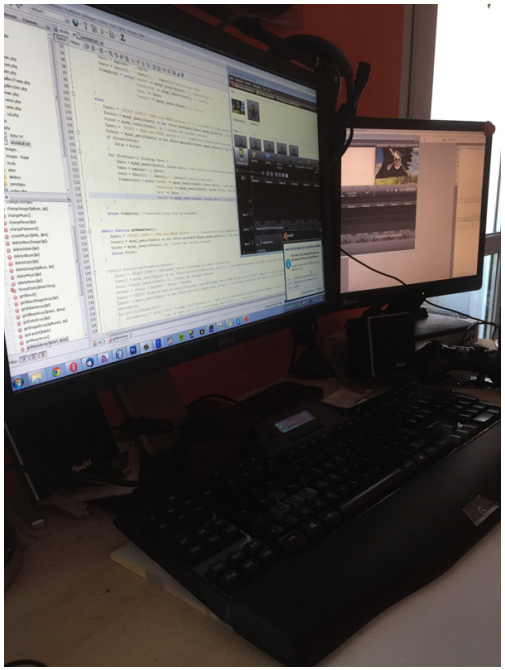
Mac OS

OS's you may use

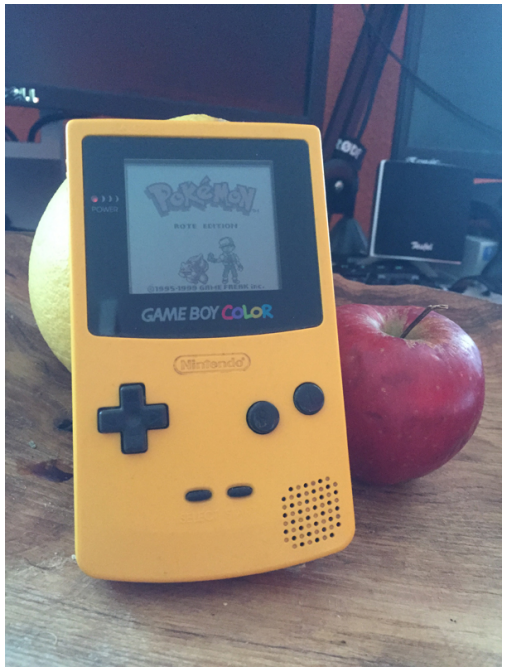


Linux

recommended

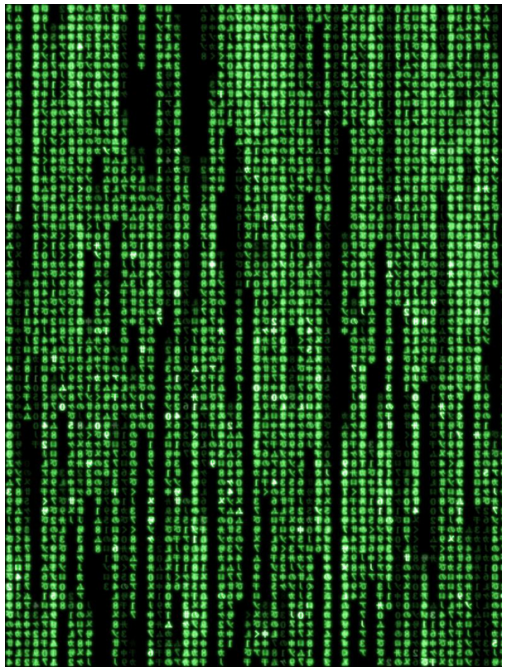


Windows



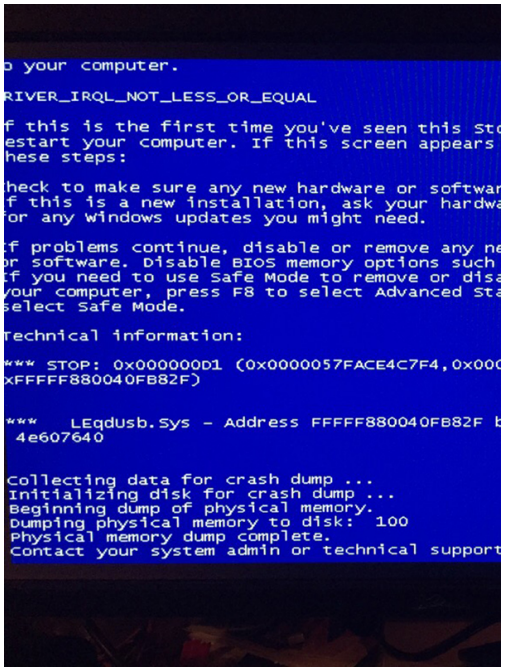
Mac OS

OS's you may use



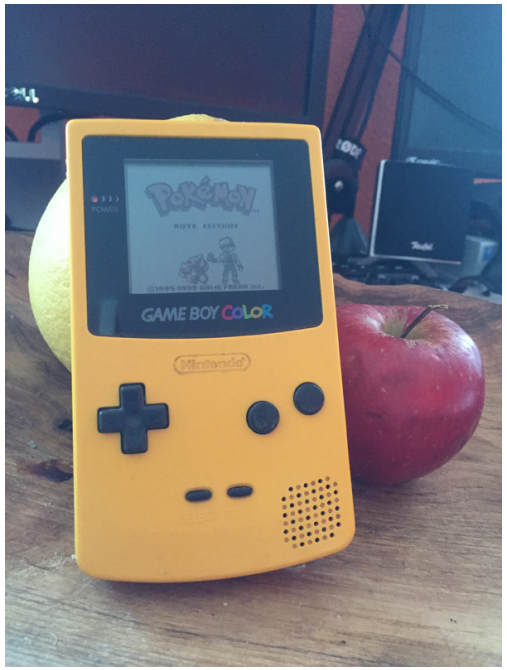
Linux

recommended



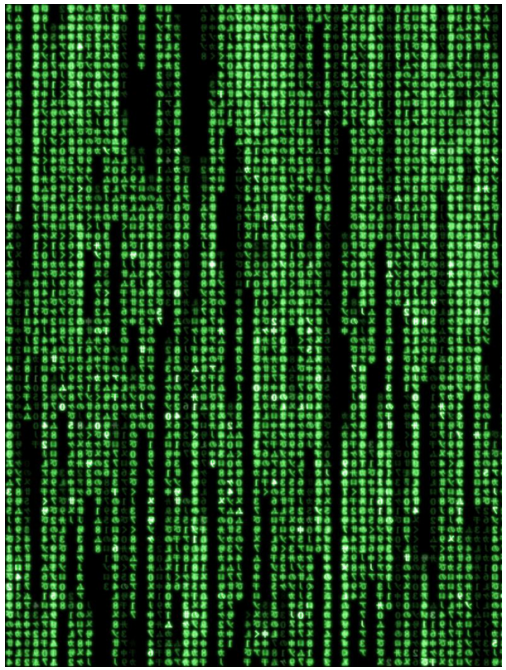
Windows

supported



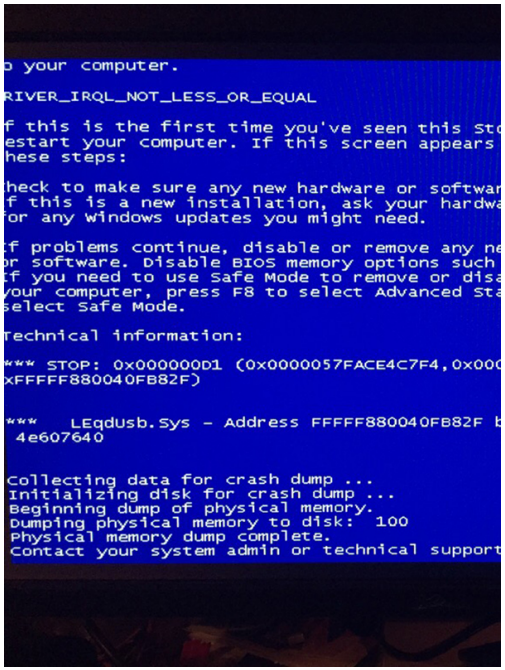
Mac OS

OS's you may use



Linux

recommended



Windows

supported

Mac OS

supported

Installing gcc on Linux

Ubuntu / Debian:

```
$ sudo apt-get install gcc
```

Arch:

```
$ sudo pacman -S gcc
```

... and you're done ;-)

cygwin

- Download installer from `https://cygwin.com/install.html`
- Run it
  - "Install from Internet"
  - Choose your installation path
  - Choose path for installation files
  - "Direct Connection"
  - Choose a mirror
  - Important software already is selected
  - **Optional**: powerful editor "vim" in *Editors*
  - **Recommended**: "GDB" in *Devel* and "libncurses-devel" in *libs* for the advanced course
  - Watching loading bars...
  - ???
  - Profit!
- Use cygwin-console like a linux terminal

Mac OS

If you want to use Mac OS in the course,

- Install *gcc* with *homebrew*, or
- Use *XCode*

Your tutor might not be able to help you with Mac OS specific problems.

The first program

- Create a new file named **main.c**.
- Open it in your text editor of choice.
- Fill it as follows:

```c
#include <stdio.h>

int main(void) {
    printf("Hello World!\n");
    /* Print "Hello World!" on the
       command line */
    return 0;
}
```

From source to bits

Source code

```
$ gcc main.c
```

(Preprocessing, compiling, assembling, linking)

Executable program

Linux (**a.out**)

```
$ ./a.out
```

Windows (**a.exe**)

```
$ ./a.exe
```

A basic program

```c
#include <stdio.h>

int main(void) {

    printf("Hello World!\n");
    /* Print "Hello World!" on the
       command line */

    return 0;
}
```

} Preprocessor statements

} Main function

Preprocessor statements

- Processed before compilation
- Have their own language, start with a #

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

- Includes the *input/output header* from the **C standard library**
- Needed to use *printf()*

Preprocessor statements hace way more use cases, but they are very different from the actual C programming language.
In this course, we will use them for inclusions only.

The main function

- Basic function of every program
- Exists **exactly once** per program
- Called on program start

```
3 int main(void) {
```

- As a function, *main()* can take parameters and return a value
- Get used to *void* and *int*. They will be explained later
- '{' marks the start of the main function scope

The main function scope

- Contains program statements
- They are processed from top to bottom

```
9       return 0;
10  }
```

- Last statement, ends main function (and thus the whole program)
- *0* tells the OS that everything went right
- '}' marks the end of the main function scope

Statements

- Instructions for the computer
- End with a ; (semicolon)

```
5       printf("Hello World!\n");
```

- Here is the empty statement:

```
        ;
```

- All statements are located in function blocks

Comments

- Information for the programmer, cut out before compilation

Single line comments:

```
6     // Prints "Hello World!" on the command line
```

Block comments (multi-line):

```
6     /* Prints "Hello World!"
7        on the command line */
```

Better style of block comments:

```
6     /*
7      * Prints "Hello World!"
8      * on the command line
9      */
```

A few words on style

- There can be multiple statements on one line
- Indentation is not necessary at all

A few words on style

- There can be multiple statements on one line
- Indentation is not necessary at all
- **But**...

```
#include <stdio.h>
int
main    (         void ){printf("Hello World!\n");
        // Prints
/*"Hello World!"              */
        return 0;}
```

Much more enjoyable

- Put each statement on a single line
- Indent every statement in the main function by one *tab* (you can also use *spaces*)
- Use /* ... */ rather than // ...
- Leave blank lines between different parts of the program
- Use *spaces* consistently to get clear code:

```
int main(void) {
    printf("Hello World!");
    /* Prints "Hello World!" */
    return 0;
}
```