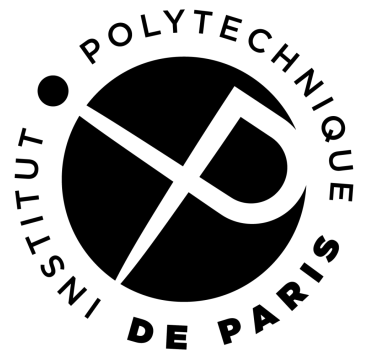


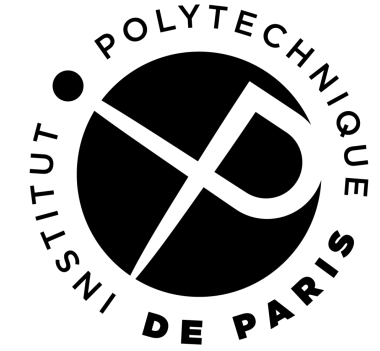
From the source to the execution

Bachelor of Science - École polytechnique

gael.thomas@inria.fr

Key concepts

- First language constructs
- Compilation and execution of a program



I. My first program

My first program

header of the program

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

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

the line with “main” indicates where the program starts

the instructions of the program goes between **braces**

My first program

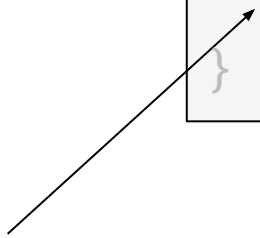
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

print `"Hello, world!"`
in the terminal

Note: `\n` adds a carriage return (next line)

My first program

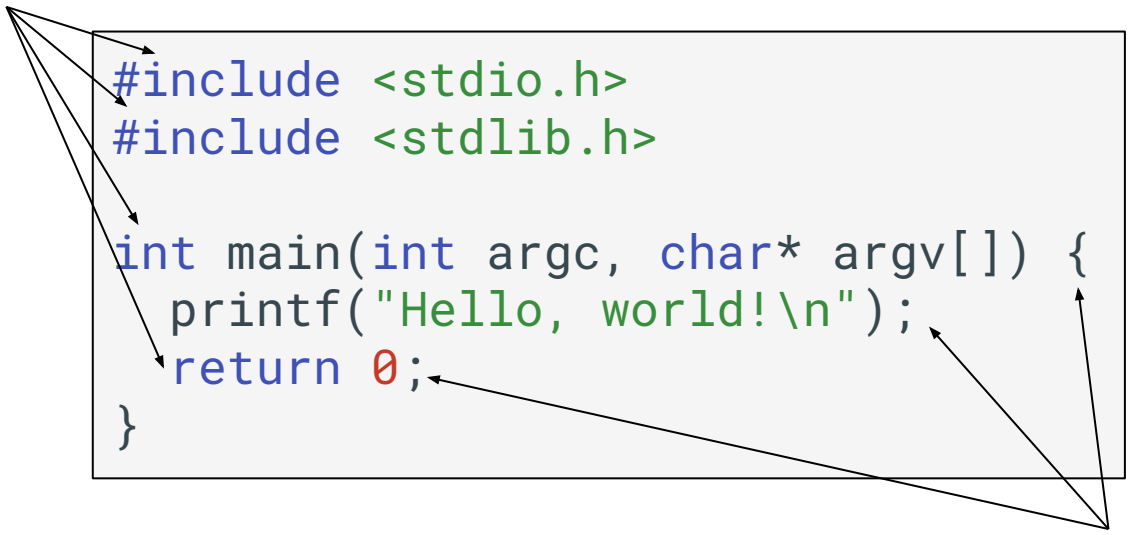
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```



Returns the value `0`
(`0` means “no error” when it’s the return code of a program)

Syntactic elements

In `blue`, the keywords of the language: #include, int, return, etc.



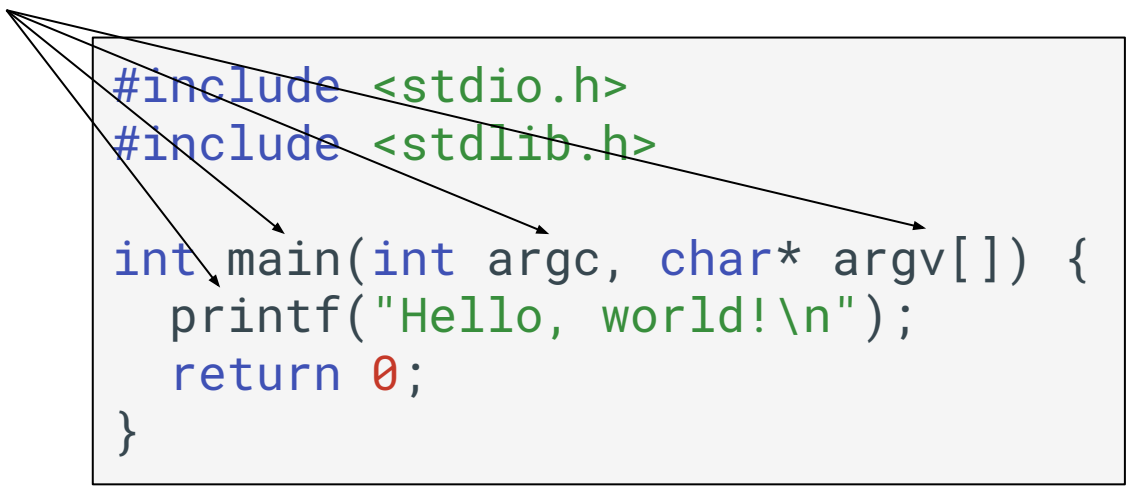
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

The words that are not letters are also keywords (e.g., {, (, *, etc.) (but they are not highlighted in blue)

A keyword is a word defined by the language

Syntactic elements

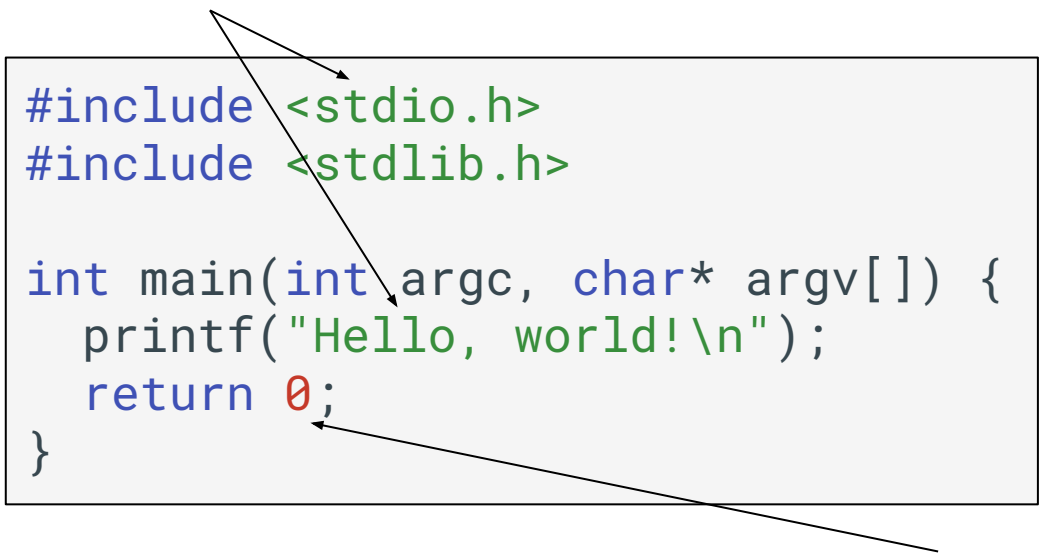
In black: the symbols



A symbol is an identifier defined by the developer

Syntactic elements

In green, a literal that is not a number:
a string when surrounded by double quotes
the name of a file for the #include keyword



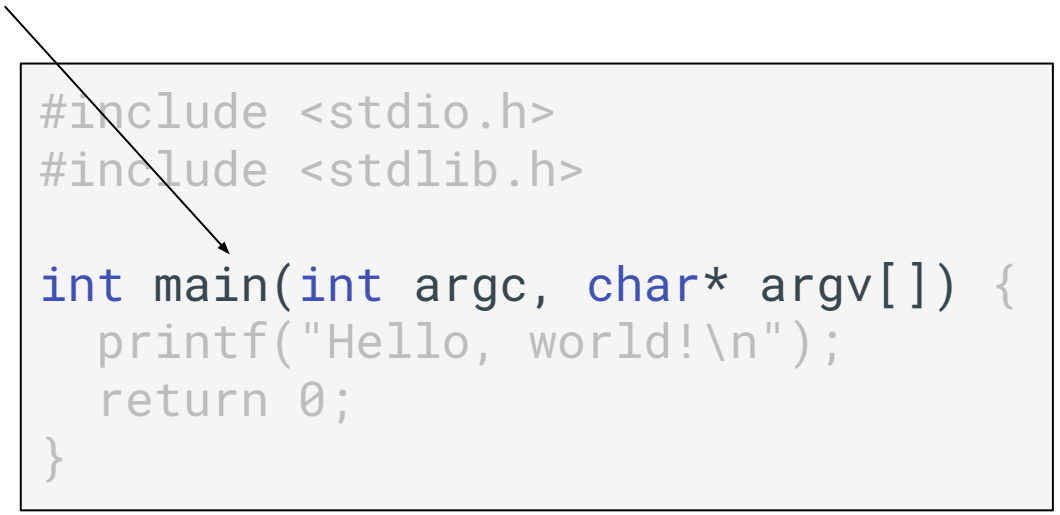
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

In red, a literal that is a number

A literal is a fixed value in the source code

Semantic elements

A function definition



```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

A function is a group of instructions that creates a macro-instruction

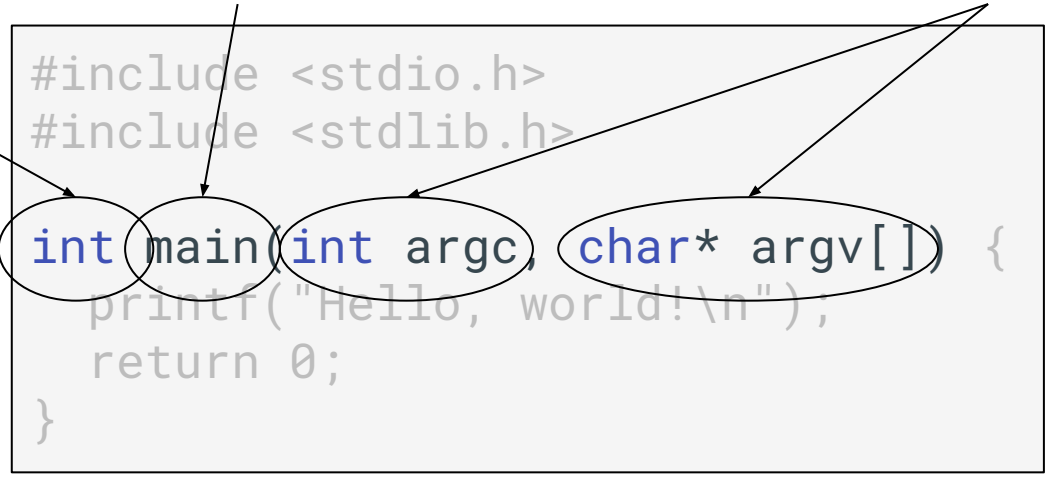
- Allows for code reuse
  (avoid writing the same code several times)
- Can take arguments and return a result

Semantic elements

result of the function: an integer

name of the function: **main**

Arguments of the function
- an integer parameter named argc
- an array of strings named argv



A function is a group of instructions that creates a macro-instruction

- Allows for code reuse
  (avoid writing the same code several times)
- Can take arguments and return a result

Semantic elements

A block (surrounded by { and })

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

A block groups together a set of instructions

Here, the block contains the instructions of the function main

Semantic elements

A function call:

```
name_of_the_function(param0,param1...);
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

We say that the instruction “calls” the function “printf”

Just like if we had inserted the code of “printf” here

Semantic elements

A functi...

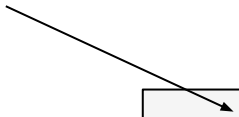
But, where is the definition of printf?

We say that the instruction “calls” the function “printf”

Just like if we had inserted the code of “printf” here

Semantic elements

Include directives



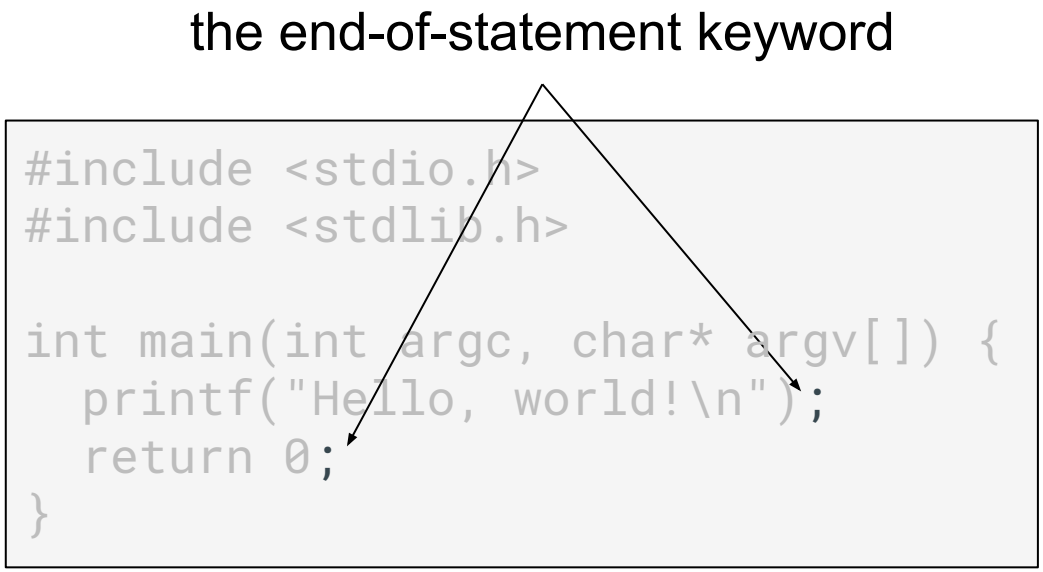
```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

`#include` means copy-paste the content of the source file given as argument

The file `stdio.h` contains the declaration of `printf`:

```c
int printf(const char* format, ...);
```

Semantic elements



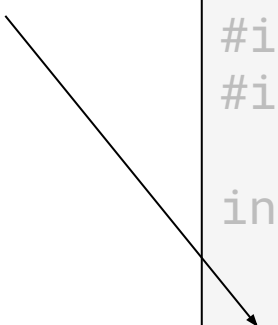
The semicolon indicates the end of a statement (instruction)

required because a statement can span multiple lines, e.g.:

```
printf(
  "Hello, world!\n"
);
```

Semantic elements

a return statement



```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

`return` ends a function and can return a value

The function returns the literal `0`
(because the function is supposed to return an `int`)

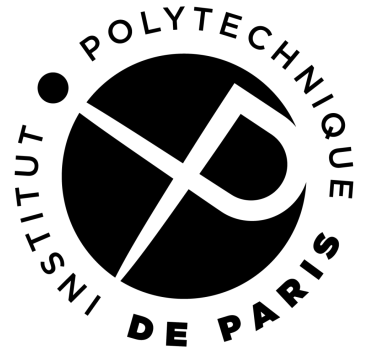
You can also comment your code

- A comment is a text that is not executed
  - Useful to explain what your code does

```c
#include <stdio.h>
#include <stdlib.h>

/*
 * A multi-line comment is enclosed between
 * a slash star and a star slash
 * (note that the other stars without slashes are
 * only here to make the comment prettier)
 */
int main(int argc, char* argv[]) {
  // a single line comment starts with slash slash
  printf("Hello, world!\n");
  /* a multi-line comment on a single line */
  return 0;
}
```

Congratulation!

You already understand 50% of the C language!



II. From the source to the execution

Writing a program in C

- You have to write your C code in a code editor

  We advise you to use:
  - vscode if you want an intuitive code editor
  - emacs or vim if you want a powerful but less intuitive code editor
  - we forbid the use of gedit, nano or notepad!

- And you have to store your source code in a file
  - A C source file usually ends with the “.c” suffix

Compiling a program written in C

- You cannot directly execute a file that contains C code
  - Before, you have to transform it into an executable that contains
    - The (global) data of the program
    - And the machine code corresponding to the source

- Machine code = the code directly executed by a processor
  - A processor basically executes a loop that
    - Fetches a machine instruction (a number) from memory
    - Activate the hardware circuit corresponding to the instruction
  - For example, the instruction 1 executes an addition, the instruction 2 loads a byte from memory etc…

Compiling a program written in C

- Transforming a source into machine code is called "compilation"
- In the course, we will use the compiler named "gcc"

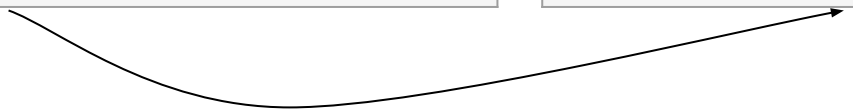
Compiling a program written in C

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

helloworld.c

```
0x7f 0x45 0x4c 0x46
0x02 0x01 0x01 0x00
0x00 0x00 0x00 0x00
0x00 0x00 0x00 0x00
0x03 0x00 0x3e 0x00
0x01 0x00 0x00 ...
```

helloworld



```
gcc -Wall -Werror helloworld.c -o helloworld
```

- gcc: the compiler
- -Wall: reports all the possible warnings (useful to avoid bugs)
- -Werror: considers any warning as an error (useful to avoid bugs)
- helloworld.c: the source file
- -o helloworld: output (-o) the executable in the file helloworld

Developing in C step by step

- In a terminal

```
$ ls
$
```

ls: command that shows the content of a directory (i.e.,folder)
=> initially, the directory is empty

Developing in C step by step

- In a terminal (in windows)

```
$ ls
$ code helloworld.c
$
```

```c
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char* argv[]) {
  printf("Hello, world!\n");
  return 0;
}
```

helloworld.c

Use the vscode editor to write the code in helloworld.c

(sometimes, the command is named vscode, sometimes code)

Developing in C step by step

- In a terminal

```
$ ls
$ vscode helloworld.c
$ ls
helloworld.c
$
```

Now the directory contains a single file: the source file helloworld.c

Developing in C step by step

- In a terminal

```
$ ls
$ vscode helloworld.c
$ ls
helloworld.c
$ gcc -Wall -Werror helloworld.c -o helloworld
$
```

Compile helloworld.c into helloworld

Developing in C step by step

- In a terminal

```
$ ls
$ vscode helloworld.c
$ ls
helloworld.c
$ gcc -Wall -Werror helloworld.c -o helloworld
$ ls
helloworld   helloworld.c
$
```

The directory contains now the source file helloworld.c and the executable helloworld

Developing in C step by step

- In a terminal

```
$ ls
$ vscode helloworld.c
$ ls
helloworld.c
$ gcc -Wall -Werror helloworld.c -o helloworld
$ ls
helloworld   helloworld.c
$ ./helloworld
Hello, world!
```

And we can finally execute our amazing application, yipeeh!

(note: the “./” at the beginning means “execute the helloworld application located in the current directory”)

Comparison with python

- Python is an interpreted language
  - It is executed by the application “python”
  - You write your code in a code editor
  - And the code is executed by the python interpreter with the command “python helloworld.py”

- C is a compiled language
  - It is executed directly by the processor
  - You write your code in a code editor
  - You compile your code into an executable with gcc
  - And the code is executed by the processor with the command “./helloworld”

Key concepts

- First language constructs
- Compilation and execution of a program