A gentle (re)introduction to C

with Olve Maudal

A one day workshop at EDC Software 2022, September 13

https://github.com/equinor/ReintroductionToC_Sep2022

- Introduction
- Getting started
- Organizing
- Optimizing
- Pointers, arrays and structures
- Tools and services
- Outroduction

Introduction

#d7f567 A gentle (re)introduction to C | 1 day course | Code | Olve Maudal

>> Title: A gentle (re)introduction to C
>> Scheduled: Tuesday 1000
>> Speaker(s): Olve Maudal
>> Length of session: 1 day
>> Room: Restaurant room (30?)
>> Max # of participants: 24
>> Type of session: workshop
>> Description: C is the mother of most programming languages and it is still one of the most popular languages (eg https://www.tiobe.com/tiobe-index/). This is a chance to (re)learn this extremely sharp and efficient programming language. Starting from scratch with "Hello, world!" we end up with a simple but useful webservice!
>> Level: intermediate
>> Speaker bio: Olve has been programming nearly every day for more than 40 years. My love for C goes deep and will last forever.
>> Extra info: This workshop assumes that you already know C or another programming language, and that you have access to a development environment supporting C99 or better.

What is your background with programming?
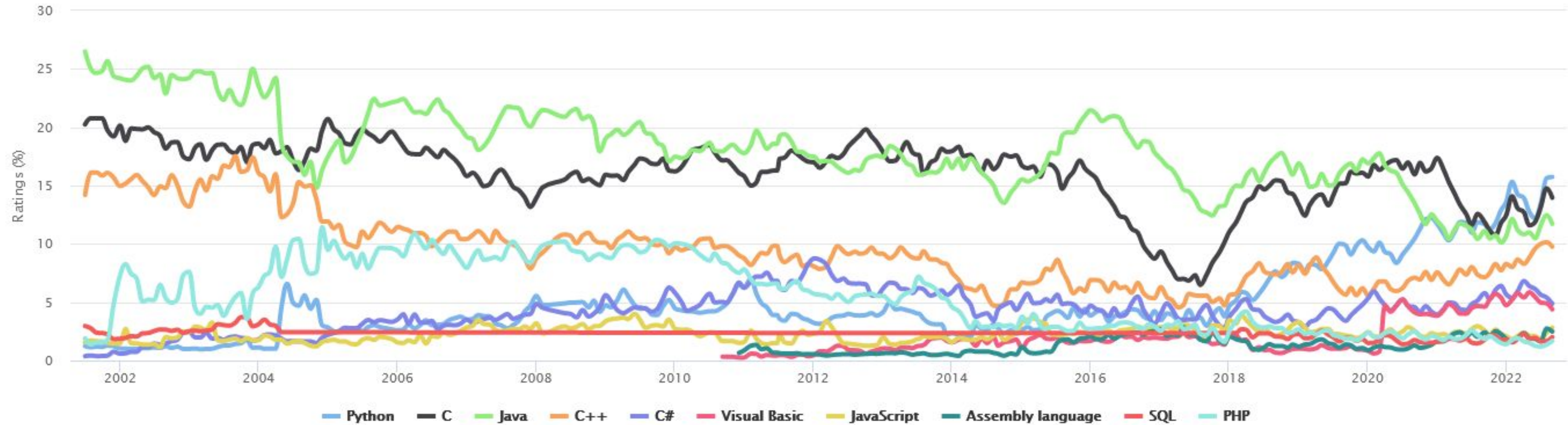
What is your experience with C?

What is your expectations for this course?

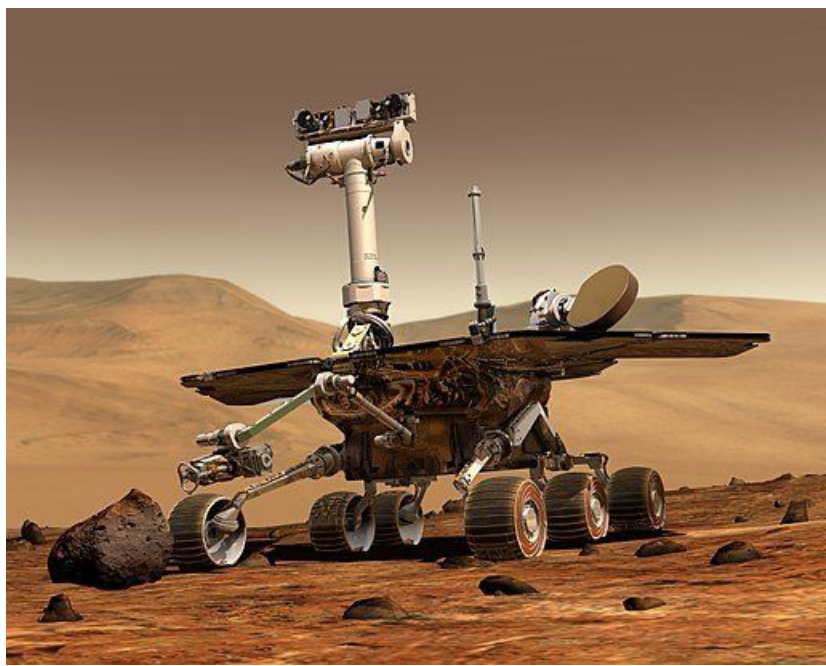Why would anyone want to learn about C (or C++) these days?

TIOBE Programming Community Index

Programming Language	2022	2017	2012	2007	2002	1997	1992	1987
Python	1	5	8	7	12	28	-	-
C	2	2	1	2	2	1	1	1
Java	3	1	2	1	1	16	-	-
C++	4	3	3	3	3	2	2	6
C#	5	4	4	8	14	-	-	-
Visual Basic	6	14	-	-	-	-	-	-
JavaScript	7	8	10	9	8	24	-	-

C is widely used for **systems programming in implementing operating systems and embedded system application**s. This is for several reasons:

- The code generated **after compilation doesn't demand many system features**, and can be invoked from some boot code in a straightforward manner – **it's simple to execute**.
- The C language statements and expressions typically **map well on to sequences of instructions for the target processor**, and consequently there is a low run-time demand on system resources – **it's fast to execute**.
- The language makes it **easy to overlay structures onto blocks of binary data**, allowing the data to be comprehended, navigated and modified – it can write data structures, even file systems.
- The language supports a rich set of operators, including bit manipulation, for integer arithmetic and logic, and perhaps different sizes of floating point numbers – **it can process appropriately-structured data effectively**.
- Platform hardware can be accessed with pointers and type punning, so system-specific features (e.g. Control/Status Registers, I/O registers) can be configured and used with code written in C – **it interacts well with the platform it's running on**.
- Depending on the linker and environment, C code can also call libraries written in assembly language, and may be called from assembly language – **it interoperates well with other code**.
- **C has a very mature and broad ecosystem**, including open source compilers, debuggers and utilities, and is the de facto standard. It's **likely the drivers already exist in C**, or that there is a similar CPU architecture as a back-end of a C compiler, so there is **reduced incentive to choose another language**.

A consequence of C's wide availability and efficiency is that compilers, libraries and interpreters of other programming languages are often implemented in C. **For example, the reference implementations of Python, Perl, Ruby, and PHP are written in C**.

C enables programmers to create efficient implementations of algorithms and data structures, because the layer of abstraction from hardware is thin, and its overhead is low, an important criterion for computationally intensive programs. **For example, the GNU Multiple Precision Arithmetic Library, the GNU Scientific Library, Mathematica, and MATLAB are completely or partially written in C**. Many languages support calling library functions in C, **for example, the Python-based framework NumPy uses C for the high-performance and hardware-interacting aspects**.

C is sometimes used as an **intermediate language by implementations of other languages**. This approach may be used for portability or convenience; by using C as an intermediate language, additional machine-specific code generators are not necessary. **C has some features, such as line-number preprocessor directives and optional superfluous commas at the end of initializer lists, that support compilation of generated code**. However, some of C's shortcomings have prompted the development of other C-based languages specifically designed for use as intermediate languages, such as C--.

	Energy			Time			Mb
(c) C	1.00		(c) C	1.00		(c) Pascal	1.00
(c) Rust	1.03		(c) Rust	1.04		(c) Go	1.05
(c) C++	1.34		(c) C++	1.56		(c) C	1.17
(c) Ada	1.70		(c) Ada	1.85		(c) Fortran	1.24
(v) Java	1.98		(v) Java	1.89		(c) C++	1.34
(c) Pascal	2.14		(c) Chapel	2.14		(c) Ada	1.47
(c) Chapel	2.18		(c) Go	2.83		(c) Rust	1.54
(v) Lisp	2.27		(c) Pascal	3.02		(v) Lisp	1.92
(c) Ocaml	2.40		(c) Ocaml	3.09		(c) Haskell	2.45
(c) Fortran	2.52		(v) C#	3.14		(i) PHP	2.57
(c) Swift	2.79		(v) Lisp	3.40		(c) Swift	2.71
(c) Haskell	3.10		(c) Haskell	3.55		(i) Python	2.80
(v) C#	3.14		(c) Swift	4.20		(c) Ocaml	2.82
(c) Go	3.23		(c) Fortran	4.20		(v) C#	2.85
(i) Dart	3.83		(v) F#	6.30		(i) Hack	3.34
(v) F#	4.13		(i) JavaScript	6.52		(v) Racket	3.52
(i) JavaScript	4.45		(i) Dart	6.67		(i) Ruby	3.97
(v) Racket	7.91		(v) Racket	11.27		(c) Chapel	4.00
(i) TypeScript	21.50		(i) Hack	26.99		(v) F#	4.25
(i) Hack	24.02		(i) PHP	27.64		(i) JavaScript	4.59
(i) PHP	29.30		(v) Erlang	36.71		(i) TypeScript	4.69
(v) Erlang	42.23		(i) Jruby	43.44		(v) Java	6.01
(i) Lua	45.98		(i) TypeScript	46.20		(i) Perl	6.62
(i) Jruby	46.54		(i) Ruby	59.34		(i) Lua	6.72
(i) Ruby	69.91		(i) Perl	65.79		(v) Erlang	7.20
(i) Python	75.88		(i) Python	71.90		(i) Dart	8.64
(i) Perl	79.58		(i) Lua	82.91		(i) Jruby	19.84

Getting started

These following slides are just unabridged and unfiltered notes that I use while doing an ad-hoc course. I do share the slides during and after the event, but I do not actually show them during the course - it is mostly live coding and discussions.

Beware, out of context and without explanation, the code snippets and examples are often just complete nonsense...

```c
#include <stdio.h>

int main(void)
{
    puts("Hello");
    return 0;
}
```

Notes:

walk through line by line
- include declarations for standard input/output
- main() is the start point for C programs (in hosted environments)
- main(void) means taking no arguments
- puts() is a standard library function
- string literal
- return status to the runtime environment

cc -o hello hello.c && ./hello
echo $?

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

// int puts(const char *);
// #define EOF (-1)
// #define EXIT_FAILURE 1
// #define EXIT_SUCCESS 0

int main(void)
{
    if (puts("Hello") == EOF)
        return EXIT_FAILURE;
    return EXIT_SUCCESS;
}
```

Notes:

Mention macros, but do not elaborate
Mention the preprocessor and linking, but do not elaborate
Do not discuss braces and compound statements here

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

int main(void)
{
    int the_answer = 42;
    if (printf("The answer is %d\n", the_answer) == EOF)
        return EXIT_FAILURE;
    return EXIT_SUCCESS;
}
```

Notes:

Promote cppreference
Promote man7
Discuss return values
Show != 17

```c
#include <stdio.h>

int main(void)
{
    int the_answer = 42;
    printf("The answer is %d!\n", the_answer);
    return 0;
}
```

Notes:

Discuss being too pedantic
Errors cause by error checking
Demonstrate implicit return

```c
#include <stdio.h>

int main(void)
{
    int fahr, celsius;
    int lower, upper, step;

    lower = 0;   // lower limit of temperature table
    upper = 300; // upper limit
    step = 20;   // step size

    fahr = lower;
    while (fahr <= upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%4d %4d\n", fahr, celsius);
        fahr = fahr + step;
    }

    return 0;
}
```

Exercise

f2c_table.c

```c
#include <stdio.h>

int main(void)
{
    int fahr, celsius;
    int lower, upper, step;

    lower = 0;    // lower limit of temperature table
    upper = 300; // upper limit
    step = 20;   // step size

    fahr = lower;
    while (fahr <= upper) {
        celsius = 5 * (fahr-32) / 9;
        printf("%4d %4d\n", fahr, celsius);
        fahr = fahr + step;
    }

    return 0;
}
```

```
$ cc f2c_table.c -o f2c_table
$ ./f2c_table
   0  -17
  20   -6
  40    4
  60   15
  80   26
 100   37
 120   48
 140   60
 160   71
 180   82
 200   93
 220  104
 240  115
 260  126
 280  137
 300  148
$
```

The lowest natural temperature ever directly recorded at ground level on Earth is −89.2 °C (−128.6 °F) at the then-Soviet Vostok Station in Antarctica on 21 July 1983.

The current official highest registered air temperature on Earth is 56.7 °C (134.1 °F), recorded on 10 July 1913 in Death Valley in the United States.

Use the code snippet on this slide as inspiration. Write a C program (c2f_table.c) that prints out a Celsius to Fahrenheit conversion table from -90 Celsius to 60 Celsius with a step interval of 10 degrees.

Extra:

- Experiment with different ways to write the program. Experiment with do-while loops, for loops, and gotos

- Be a teacher! Talk to other students. Explain things if needed.

source: Exercise inspired by K&R, see also:
https://en.wikipedia.org/wiki/Lowest_temperature_recorded_on_Earth
https://en.wikipedia.org/wiki/Highest_temperature_recorded_on_Earth

Sample solutions

```c
#include <stdio.h>

int main(void)
{
    int fahr, celsius;
    int lower, upper, step;

    lower = -90; // lower limit of temperature table
    upper = 60;  // upper limit
    step = 10;   // step size

    celsius = lower;
    while (celsius <= upper) {
        fahr = celsius * 9 / 5 + 32;
        printf("%4d %4d\n", celsius, fahr);
        celsius = celsius + step;
    }

    return 0;
}
```

```
$ cc c2f_table.c -o c2f_table
$ ./c2f_table
 -90 -130
 -80 -112
 -70  -94
 -60  -76
 -50  -58
 -40  -40
 -30  -22
 -20   -4
 -10   14
   0   32
  10   50
  20   68
  30   86
  40  104
  50  122
  60  140
$
```

```c
#include <stdio.h>

int main(void)
{
    int lower_limit_of_temp_table = -90;
    int upper_limit_of_temp_table = 60;
    int temp_step = 10;

    int celsius = lower_limit_of_temp_table;
    while (celsius <= upper_limit_of_temp_table) {
        int fahr = celsius * 9 / 5 + 32;
        printf("%4d %4d\n", celsius, fahr);
        celsius += temp_step;
    }

    return 0;
}
```

```c
#include <stdio.h>

int main(void)
{
    int lower_limit_of_temp_table = -90;
    int upper_limit_of_temp_table = 60;
    int temp_step = 10;

    for (int celsius = lower_limit_of_temp_table; celsius <= upper_limit_of_temp_table; celsius += temp_step) {
        int fahr = celsius * 9 / 5 + 32;
        printf("%4d %4d\n", celsius, fahr);
    }

    return 0;
}
```

```c
#include <stdio.h>

int main(void)
{
    int celsius = -90;

    do {
        int fahr = celsius * 9 / 5 + 32;
        printf("%4d %4d\n", celsius, fahr);
        celsius += 10;
    } while (celsius <= 60);

    return 0;
}
```

```c
#include <stdio.h>

int main(void)
{
    for (int c = -90; c <= 60; c += 10)
        printf("%4d %4d\n", c, c * 9 / 5 + 32);
}
```

```c
int printf(const char *, ...);
int main() {
        int c = -90, f;
again:  f = c * 9 / 5 + 32;
        printf("%4d %4d\n", c, f);
        if ((c += 10) <= 60) goto again;
}
```

Organizing

```c
#include <stdio.h>

int main(void)
{
    int lower = -90; // lower limit
    int upper = 60;  // upper limit
    int step = 10;   // temp step

    // print temp table
    int celsius = lower;
    while (celsius <= upper) {
        int fahr = celsius * 9 / 5 + 32;  // to fahrenheit from celsisus
        printf("%4d %4d\n", celsius, fahr);
        celsius += step;
    }

    return 0;
}
```

Notes:

start with sample answer
demonstrate transformation between for loops and while loops

```c
#include <stdio.h>

int main(void)
{
    int lower = -90; // lower limit
    int upper = 60;  // upper limit
    int step = 10;    // temp step

    // print temp table
    for (int celsius = lower; celsius <= upper; celsius += step) {
        int fahr = celsius * 9 / 5 + 32;  // to fahrenheit from celsisus
        printf("%4d %4d\n", celsius, fahr);
    }

    return 0;
}
```

Notes:

discuss comments in code

```c
#include <stdio.h>

int to_fahrenheit(int celsius)
{
    return celsius * 9 / 5 + 32;
}

int main(void)
{
    int lower = -90; // lower limit
    int upper = 60;  // upper limit
    int step = 10;   // temp step

    // print temp table
    for (int celsius = lower; celsius <= upper; celsius += step) {
        int fahr = to_fahrenheit(celsius);
        printf("%4d %4d\n", celsius, fahr);
    }

    return 0;
}
```

```c
#include <stdio.h>

int to_fahrenheit(int celsius)
{
    return celsius * 9 / 5 + 32;
}

void print_temp_table(int lower, int upper, int step)
{
    for (int celsius = lower; celsius <= upper; celsius += step) {
        int fahr = to_fahrenheit(celsius);
        printf("%4d %4d\n", celsius, fahr);
    }
}

int main(void)
{
    int lower = -90; // lower limit
    int upper = 60;  // upper limit
    int step = 10;   // temp step

    print_temp_table(lower, upper, step);

    return 0;
}
```

```c
#include <stdio.h>

extern int to_fahrenheit(int celsius)
{
    return celsius * 9 / 5 + 32;
}

static void print_temp_table(int lower, int upper, int step)
{
    for (int celsius = lower; celsius <= upper; celsius += step) {
        int fahr = to_fahrenheit(celsius);
        printf("%4d %4d\n", celsius, fahr);
    }
}

int main(void)
{
    int lower_limit = -90;
    int upper_limit = 60;
    int temp_step = 10;

    print_temp_table(lower_limit, upper_limit, temp_step);

    return 0;
}
```

Notes:

discuss symbol names
discuss external and internal linkage (extern vs static)
mention command query separation
discuss depth of calls
mention dependency injection, but do not demonstrate

c2f_table.c

```c
#include <stdio.h>

extern int to_fahrenheit(int celsius);

static void print_temp_table(int lower, int upper, int step)
{
    for (int celsius = lower; celsius <= upper; celsius += step) {
        int fahr = to_fahrenheit(celsius);
        printf("%4d %4d\n", celsius, fahr);
    }
}

int main(void)
{
    int lower_limit = -90;
    int upper_limit = 60;
    int temp_step = 10;

    print_temp_table(lower_limit, upper_limit, temp_step);

    return 0;
}
```

tempconv.c

```c
int to_fahrenheit(int celsius)
{
    return celsius * 9 / 5 + 32;
}

int to_celsius(int fahrenheit)
{
    return 5 * (fahrenheit - 32) / 9;
}
```

```
$ cc -c tempconv.c
$ cc -c c2f_table.c
$ nm tempconv.o
0000000000000031 T to_celsius
0000000000000000 T to_fahrenheit
$ nm c2f_table.o
                 U _GLOBAL_OFFSET_TABLE_
000000000000005a T main
0000000000000000 t print_temp_table
                 U printf
                 U to_fahrenheit
$ cc tempconv.o c2f_table-o -o c2f_table
$ ./c2f_table
```

```
-90 -130
-80 -112
-70  -94
-60  -76
-50  -58
-40  -40
-30  -22
-20   -4
-10   14
  0   32
 10   50
 20   68
 30   86
 40  104
 50  122
 60  140
```

c2f_table.c

```c
#include "tempconv.h"

#include <stdio.h>

static void print_temp_table(int lower, int upper, int step)
{
    for (int celsius = lower; celsius <= upper; celsius += step) {
        int fahr = to_fahrenheit(celsius);
        printf("%4d %4d\n", celsius, fahr);
    }
}

int main(void)
{
    int lower_limit = -90;
    int upper_limit = 60;
    int temp_step = 10;

    print_temp_table(lower_limit, upper_limit, temp_step);

    return 0;
}
```

tempconv.h

```c
#ifndef TEMPCONV_H_INCLUDED
#define TEMPCONV_H_INCLUDED

int to_fahrenheit(int celcisus);
int to_celsius(int fahrenheit);

#endif
```

tempconv.c

```c
#include "tempconv.h"

int to_fahrenheit(int celsius)
{
    return celsius * 9 / 5 + 32;
}

int to_celsius(int fahrenheit)
{
    return 5 * (fahrenheit - 32) / 9;
}
```

```
$ cc -c tempconv.c
$ cc -c c2f_table.c
$ cc tempconv.o c2f_table-o -o c2f_table
$ ./c2f_table
```

```
-90 -130
-80 -112
-70  -94
-60  -76
-50  -58
-40  -40
-30  -22
-20   -4
-10   14
  0   32
 10   50
 20   68
 30   86
 40  104
 50  122
 60  140
```

c2f_table.c

```c
#include "tempconv.h"

#include <stdio.h>

static void print_temp_table(int lower, int upper, int step)
{
    for (int celsius = lower; celsius <= upper; celsius += step) {
        int fahr = to_fahrenheit(celsius);
        printf("%4d %4d\n", celsius, fahr);
    }
}

int main(void)
{
    int lower_limit = -90;
    int upper_limit = 60;
    int temp_step = 10;

    print_temp_table(lower_limit, upper_limit, temp_step);

    return 0;
}
```

tempconv.h

```c
#ifndef TEMPCONV_H_INCLUDED
#define TEMPCONV_H_INCLUDED

int to_fahrenheit(int celcisus);
int to_celsius(int fahrenheit);

#endif
```

tempconv.c

```c
#include "tempconv.h"

int to_fahrenheit(int celsius)
{
    return celsius * 9 / 5 + 32;
}

int to_celsius(int fahrenheit)
{
    return 5 * (fahrenheit - 32) / 9;
}
```

Makefile

```makefile
all : c2f_table

test : c2f_table
	./c2f_table

c2f_table : c2f_table.o tempconv.o
	cc -o c2f_table -Wall -Wextra -Werror --pedantic -std=c17 c2f_table.o tempconv.o

c2f_table.o : c2f_table.c tempconv.h
	cc -Wall -Wextra -Werror --pedantic -std=c17 -c c2f_table.c

tempconv.o : tempconv.c tempconv.h
	cc -Wall -Wextra -Werror --pedantic -std=c17 -c tempconv.c

clean :
	rm *.o
	rm c2f_table
```

```
$ ls
Makefile  c2f_table.c  tempconv.c  tempconv.h
$ make
cc -Wall -Wextra -Werror --pedantic -std=c17 -c c2f_table.c
cc -Wall -Wextra -Werror --pedantic -std=c17 -c tempconv.c
cc -o c2f_table -Wall -Wextra -Werror --pedantic -std=c17 c2f_table.o tempconv.o
$ make test
./c2f_table
 -90 -130

  ...

  60  140
$ make clean
rm *.o
rm c2f_table
$
```

```c
#include <stdio.h>

extern int to_fahrenheit(int celsius)
{
    return celsius * 9 / 5 + 32;
}

static void print_temp_table(int lower, int upper, int step)
{
    for (int celsius = lower; celsius <= upper; celsius += step) {
        int fahr = to_fahrenheit(celsius);
        printf("%4d %4d\n", celsius, fahr);
    }
}

int main(void)
{
    int lower_limit = -90;
    int upper_limit = 60;
    int temp_step = 10;

    print_temp_table(lower_limit, upper_limit, temp_step);

    return 0;
}
```

Notes:

demonstrate to_othertempscaple injection

```c
#include <stdio.h>

extern int to_fahrenheit(int celsius)
{
    return celsius * 9 / 5 + 32;
}

static void print_temp_table(int lower, int upper, int step, int to_othertempscale(int))
{
    for (int temp = lower; temp <= upper; temp += step)
        printf("%4d %4d\n", temp, to_othertempscale(temp));
}

int main(void)
{
    int lower_limit = -90;
    int upper_limit = 60;
    int temp_step = 10;

    print_temp_table(lower_limit, upper_limit, temp_step, to_fahrenheit);

    return 0;
}
```

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

int main(void)
{
    int count = 10;
    while (count--) {
        int num = rand();
        printf("%d\n", num);
    }
    return 0;
}
```

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

int main(int argc, char * argv[])
{
    for (int i = 0; i < argc; ++i)
        printf("[%d] %s\n", i, argv[i]);
    return 0;
}
```

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

int main(int argc, char * argv[])
{
    if (argc != 2) {
        printf("usage: randnumbers NUM\n");
        return EXIT_FAILURE;
    }

    srand(time(NULL));

    int count = atoi(argv[1]);
    while (count--) {
        int num = rand();
        printf("%d\n", num);
    }

    return EXIT_SUCCESS;
}
```

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

static bool isprime(int number)
{
    if (number < 2)
        return false;
    if (number == 2)
        return true;
    for (int denominator = 2; denominator < number; ++denominator)
        if (number % denominator == 0)
            return false;
    return true;
}

int main(void)
{
    for (int num = 0; num < 20; ++num)
        if (isprime(num))
            printf("%d\n", num);
    return 0;
}
```

Exercise

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

static bool isprime(int number)
{
    if (number < 2)
        return false;
    if (number == 2)
        return true;
    for (int denominator = 2; denominator < number; ++denominator)
        if (number % denominator == 0)
            return false;
    return true;
}

int main(void)
{
    int count = 10;
    while (count) {
        int num = rand();
        if (isprime(num)) {
            printf("%d\n", num);
            --count;
        }
    }
    return 0;
}
```

```
$ cc randprime.c -o randprime
$ ./randprime
1350490027
2044897763
35005211
1369133069
135497281
1120048829
213975407
1504569917
1398295499
243268139
$
```

A prime is a natural number greater than 1 that is not a product of two smaller natural numbers. The first primes are: 2 3 5 7 11 13 17 19 …

To the left is a correct, but very slow and very naive, implementation of a program that prints out ten random primes. Do not change the algorithm for isprime(), this exercise is about **how to organize code**. (No worries, we will optimize the code soon!)

1) Type in this program and verify that you get 10 primes

2) Extract isprime() into primelib.h and primelib.c

3) Compile primelib.c and randprime.c separately. Then link them together to create an executable.

4) Estimate how long it would take to create 1000 random primes with this isprime function in your runtime environment. How much difference if you compile with "-O0" vs "-O2"?

Extra:

5) Create a Makefile that builds your program (see: https://www.gnu.org/software/make/manual/make.html#Simple-Makefile for inspiration)

6) Add support for some very simple command line arguments, eg "./randprime 20"

6) Be a teacher! Help and discuss the exercise with other students

Sample solution

primelib.c

```c
#include "primelib.h"

bool isprime(int number)
{
    if (number < 2)
        return false;
    if (number == 2)
        return true;
    for (int denominator = 2; denominator < number; ++denominator)
        if (number % denominator == 0)
            return false;
    return true;
}
```

primelib.h

```c
#ifndef PRIMELIB_H_INCLUDED
#define PRIMELIB_H_INCLUDED

#include <stdbool.h>

bool isprime(int number);

#endif
```

randprime.c

```c
#include "primelib.h"

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

int main(void)
{
    int count = 10;
    while (count) {
        int num = rand();
        if (isprime(num)) {
            printf("%d\n", num);
            --count;
        }
    }
    return 0;
}
```

Makefile

```makefile
all : randprime

test : randprime
	@date
	./randprime
	@date

randprime : randprime.c primelib.o
	cc -O0 -o randprime -Wall -Wextra -Werror --pedantic -std=c17 randprime.c primelib.o

primelib.o : primelib.c primelib.h
	cc -O0 -Wall -Wextra -Werror --pedantic -std=c17 -c primelib.c

clean :
	rm *.o
	rm randprime
```

```
$ make
cc -O0 -Wall -Wextra -Werror --pedantic -std=c17 -c primelib.c
cc -O0 -o randprime -Wall -Wextra -Werror --pedantic -std=c17 randprime.c
primelib.o
$ make test
Sat Aug 27 22:52:21 CEST 2022
./randprime
1350490027
2044897763
35005211
1369133069
135497281
1120048829
213975407
1504569917
1398295499
243268139
Sat Aug 27 22:52:51 CEST 2022
$
```

Optimizing

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

bool isprime(int number)
{
    if (number < 2)
        return false;
    for (int denominator = 2; denominator < number; ++denominator)
        if (number % denominator == 0)
            return false;
    return true;
}

int main(void)
{
    int minnum = 0;
    int maxnum = 1;
    while (maxnum < 1000000000) {
        maxnum *= 10;
        clock_t start = clock();
        int counter = 0;
        for (int i = minnum; i <= maxnum; i++)
            if (isprime(i))
                ++counter;
        clock_t stop = clock();
        double duration_in_ms = 1000*(stop-start)/(double)CLOCKS_PER_SEC;
        printf("There are %d primes between %d and %d inclusive (Timex: %.6f ms)\n",
            counter, minnum, maxnum, duration_in_ms);
    }
    return 0;
}
```

Notes:

demonstrate the exponential time
remember to discuss how to stop execution of a program

```cpp
bool isprime_cute(int number)
{
    int c = 0;
    for (int d = 1; d <= number; ++d)
        if (number % d == 0)
            c++;
    return c == 2;
}


bool isprime_naive(int number)
{
    if (number < 2)
        return false;
    for (int denominator = 2; denominator < number; ++denominator)
        if (number % denominator == 0)
            return false;
    return true;
}


bool isprime_improved(int number)
{
    if (number < 2)
        return false;
    if (number == 2)
        return true;
    if (number % 2 == 0)
        return false;
    for (int denominator = 3; denominator < number; denominator += 2)
        if (number % denominator == 0)
            return false;
    return true;
}
```

```cpp
bool isprime_smart(int number)
{
    if (number < 2)
        return false;
    if (number % 2 == 0)
        return number == 2;
    for (int denominator = 3; denominator * denominator <= number; denominator += 2)
        if (number % denominator == 0)
            return false;
    return true;
}


bool isprime_clever(int number)
{
    if (number == 2 || number == 3)
        return true;
    if (number <= 1 || number % 2 == 0 || number % 3 == 0)
        return false;
    for (int denominator = 5; denominator * denominator <= number; denominator += 6)
        if (number % denominator == 0 || number % (denominator + 2) == 0)
            return false;
    return true;
}
```

```c
#define IS_A_PRIME 0
#define IS_NOT_A_PRIME 1
#define PRIMECACHE_MAXNUM ((size_t)INT_MAX)

static bool primecache[PRIMECACHE_MAXNUM+1];
static size_t primecache_size = 0;

bool isprime_sieve(int number)
{
    if (number < 2)
        return false;
    if ((size_t)number > primecache_size) {
        primecache_size = (size_t)number;
        for (size_t i = 2; i * i <= primecache_size; ++i)
            if (primecache[i] == IS_A_PRIME)
                for (size_t j = i + i; j <= primecache_size; j += i)
                    primecache[j] = IS_NOT_A_PRIME;
    }
    return primecache[number] == IS_A_PRIME;
}
```

Notes:

discuss optimization on many parameters, speed, memory size, cache hits, correctness?

```c
#define CHECK_BIT(value,bitpos) (!!((value) & (1<<(bitpos))))
#define SET_BIT(value,bitpos) (value |= (1<<(bitpos)))
#define CLEAR_BIT(value,bitpos) (value &= ~(1<<(bitpos)))

#define SIEVE_MAX (INT_MAX)
typedef unsigned char sieve_type;
#define SIEVE_TYPE_BITS (sizeof(sieve_type) * CHAR_BIT)
static sieve_type compactcache[SIEVE_MAX/SIEVE_TYPE_BITS];
static size_t compactcache_size = 0;
#define SIEVE_SET_AS_NOT_A_PRIME(num_) (SET_BIT(compactcache[(num_-1)/SIEVE_TYPE_BITS], (num_-1)%SIEVE_TYPE_BITS))
#define SIEVE_IS_A_PRIME(num_) (CHECK_BIT(compactcache[(num_-1)/SIEVE_TYPE_BITS], (num_-1)%SIEVE_TYPE_BITS) == 0)

bool isprime_compactsieve(int number)
{
    if (number < 2)
        return false;
    if ((size_t)number > compactcache_size) {
        compactcache_size = (size_t)number;
        for (size_t i = 2; i * i <= compactcache_size; ++i)
            if (SIEVE_IS_A_PRIME(i))
                for (size_t j = i * i; j <= compactcache_size; j += i)
                    SIEVE_SET_AS_NOT_A_PRIME(j);
    }
    return SIEVE_IS_A_PRIME(number);
}
```

Pointers, arrays and structures

```
int main(void)
{
    putchar(0x48);
    putchar(0x65);
    putchar(0x6c);
    putchar(0x6c);
    putchar(0x6f);
    putchar(0x0a);
    return 0;
}
```

Notes:

```c
int main(void)
{
    int a[6] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a};
    size_t i = 0;
    putchar(a[i]);
    ++i;
    putchar(a[i]);
    ++i;
    putchar(a[i]);
    ++i;
    putchar(a[i]);
    ++i;
    putchar(a[i]);
    ++i;
    putchar(a[i]);
    ++i;
    return 0;
}
```

Notes:

```c
int main(void)
{
    int a[6] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a};
    size_t i = 0;
    while (i < 6) {
        putchar(a[i]);
        ++i;
    }
    return 0;
}

---

int main(void)
{
    int a[6] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a};
    for (size_t i = 0; i < 6; ++i)
        putchar(a[i]);
    return 0;
}
```

Notes:

```
int main(void)
{
    int a[] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a};
    size_t n = sizeof a / sizeof a[0];
    for (size_t i = 0; i < n; i++)
        putchar(a[i]);
    return 0;
}
```

Notes:

```c
int main(void)
{
    char a[] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a};
    size_t n = sizeof a / sizeof a[0];
    for (size_t i = 0; i < n; i++)
        putchar(a[i]);
    return 0;
}
```

Notes:

```c
static void myputchars(const char a[])
{
    size_t n = sizeof a / sizeof a[0];
    for (size_t i = 0; i < n; i++)
        putchar(a[i]);
}

int main(void)
{
    char a[] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a};
    myputchars(a);
    return 0;
}
```

Notes:

```c
static void myputchars(const char * a, size_t n)
{
    for (size_t i = 0; i < n; i++)
        putchar(a[i]);
}

int main(void)
{
    char a[] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a};
    size_t n = sizeof a / sizeof a[0];
    myputchars(a, n);
    return 0;
}
```

Notes:

```c
static void myputchars(const char * a, size_t n)
{
    const char * begin = a;
    const char * end = a + n;
    for (const char * it = begin; it != end; ++it)
        putchar(*it);
}

int main(void)
{
    char a[] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a};
    size_t n = sizeof a / sizeof a[0];
    myputchars(a, n);
    return 0;
}
```

Notes:

```
static void myputchars(const char * begin, const char * end)
{
    for (const char * it = begin; it != end; ++it)
        putchar(*it);
}

int main(void)
{
    char a[] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a};
    size_t n = sizeof a / sizeof a[0];
    myputchars(a, a + n);
    return 0;
}
```

Notes:

```c
static void myforeach(const char * begin, const char * end, int func(int))
{
    for (const char * it = begin; it != end; ++it)
        func(*it);
}

int main(void)
{
    char a[] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a};
    size_t n = sizeof a / sizeof a[0];
    myforeach(a, a + n, putchar);
    return 0;
}
```

Notes:

```
static void myputchars(const char * begin)
{
    for (const char * it = begin; *it; ++it)
        putchar(*it);
}

int main(void)
{
    char a[] = {0x48, 0x65, 0x6c, 0x6c, 0x6f, 0x0a, 0x00};
    myputchars(a);
    return 0;
}
```

```c
static void myputchars(const char * s)
{
    while (*s)
        putchar(*s++);
}

int main(void)
{
    char a[] = {'H', 'e', 'l', 'l', 'o', '\n', 0};
    myputchars(a);
    return 0;
}
```

Notes:

```c
static void myputchars(const char * s)
{
    while (*s)
        putchar(*s++);
}

int main(void)
{
    char a[] = "Hello\n";
    myputchars(a);
    return 0;
}
```

Notes:

```c
static void myputchars(const char * s)
{
    while (*s)
        putchar(*s++);
}

static void removedots(char * s)
{
    char * p = s;
    for (; *s; s++)
        if (*s != '.')
            *p++ = *s;
    *p = '\0';
}

int main(void)
{
    char a[] = "....H.e..l..l..o...\n";
    removedots(a);
    myputchars(a);
    return 0;
}
```

Notes:

```c
static void myputchars(const char * s)
{
    while (*s)
        putchar(*s++);
}

static const char * removedots(char * s)
{
    char * p = s, * q = s;
    while (*s)
        ((*s != '.') && (*p++ = *s), s++);
    return (*p = '\0', q);
}

int main(void)
{
    char a[] = "....H.e..l..l..o...\n";
    myputchars(removedots(a));
    return 0;
}
```

```c
static void myputchars(const char * s)
{
    while (*s)
        putchar(*s++);
}

static void myswap(char * a, char * b)
{
    char tmp = *a;
    *a = *b;
    *b = tmp;
}

int main(void)
{
    char a[] = "leHlo\n";
    myswap(&a[0], &a[2]);
    myputchars(a);
    return 0;
}
```

Notes:

```c
static void myputchars(const char * s)
{
    while (*s)
        putchar(*s++);
}

static int mycharcmp(const void * a, const void * b)
{
    return *((const char*)a) - *((const char*)b);
}

int main(void)
{
    char a[] = "leHlo\n";
    qsort(a, strlen(a) - 1, sizeof *a, mycharcmp);
    myputchars(a);
    return 0;
}
```

Notes:

```c
static void mydelayedputchar(char ch, long ms)
{
    thrd_sleep(&(struct timespec){.tv_nsec=ms*1000*1000}, NULL);
    putchar(ch), fflush(stdout);
}
static void mytypewriter(const char * s)
{
    while (*s)
        mydelayedputchar(*s++, (rand() % (1000 * 2)));
}
int main(void)
{
    char a[] = "Hello\n";
    mytypewriter(a);
    return 0;
}
```

Notes:

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

struct mystruct {
    int a;
    long b;
    char c;
};

int main(void)
{
    printf("%zu\n", sizeof(int));
    printf("%zu\n", sizeof(char));
    printf("%zu\n", sizeof(long));
    printf("%zu\n", sizeof(struct mystruct));
    printf("%zu\n", _Alignof(long));
    printf("%zu\n", _Alignof(struct mystruct));
    return 0;
}
```

Notes:

Tools & Services

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

int main(void)
{
    int ch;
    while ((ch = getc(stdin)) != EOF)
        putc(tolower(ch), stdout);
    return 0;
}
```

Notes:

downcase

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

int main(void)
{
    int ch;
    while ((ch = getc(stdin)) != EOF)
        putc(isspace(ch) ? '\n' : ch, stdout);
    return 0;
}
```

Notes:

wordsplit

```c
#include <stdio.h>

int main(int argc, char ** argv)
{
    for (int i = 0; i < argc; ++i)
        printf("argv[%d] = %s\n", i, argv[i]);
    return 0;
}
```

```c
#include <stdio.h>

int main(int argc, char ** argv)
{
    int i = 0;
    char ** p = argv;
    while (*p)
        printf("argv[%d] = %s\n", i++, *p++);
    return 0;
}
```

```c
#include <stdio.h>

int main(int argc, char ** argv, char ** envp)
{
    char ** p = envp;
    while (*p)
        printf("%s\n", *p++);
    return 0;
}
```

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

int main(int argc, char * argv[])
{
    bool print_nw = false;
    bool print_nc = false;
    const char * filename = NULL;

    while (--argc > 0 && (*++argv)[0] == '-') {
        int c;
        while (c = *++argv[0]) {
            switch (c) {
                case 'w':
                    print_nw = true;
                    break;
                case 'c':
                    print_nc = true;
                    break;
                default:
                    printf("wordcount: illegal option %c\n", c);
                    argc = 0;
                    return EXIT_FAILURE;
            }
        }
    }

    if (argc) {
        filename = argv[0];
        --argc;
    }

    if (argc != 0) {
        printf("usage: wordcount [-w] [FILE]\n");
        return EXIT_FAILURE;
    }

    FILE * fp = stdin;
    if (filename) {
        fp = fopen(filename, "r");
        if (!fp) {
            perror("Unable to open file\n");
            return EXIT_FAILURE;
        }
    }

    int ch = 0;
    int nl = 0;
    int nw = 0;
    int nc = 0;
    bool inside_a_word = false;

    while ((ch = getc(fp)) != EOF) {
        ++nc;
        if (ch == '\n')
            ++nl;
        if (ch == ' ' || ch == '\n' || ch == '\t')
            inside_a_word = false;
        else if (!inside_a_word) {
            inside_a_word = true;
            ++nw;
        }
    }

    if (print_nw)
        printf("%d\n", nw);

    if (print_nc)
        printf("%d\n", nc);

    if (!print_nc && !print_nw)
        printf("%d %d %d\n", nl, nw, nc);

    if (filename)
        fclose(fp);

    return 0;
}
```

```c
#include <ctype.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <threads.h>
#include <unistd.h>

int verboselevel = 0;

int main(int argc, char ** argv)
{
    int opt;
    int msec = 100;
    bool upcase = false;

    while ((opt = getopt(argc, argv, "uvt:")) != -1) {
        switch (opt) {
        case 'u':
            upcase = true;
            break;
        case 't':
            msec = atoi(optarg);
            break;
        case 'v':
            ++verboselevel;
            break;
        default:
            fprintf(stderr, "Usage: %s [-t msec] [-u] words...\n", argv[0]);
            exit(EXIT_FAILURE);
        }
    }

    if (verboselevel > 0) {
        size_t nwords = argc - optind;
        printf("upcase = %s, msec = %d, nwords = %zu\n", upcase ? "true" : "false", msec, nwords);
    }

    if (optind >= argc) {
        fprintf(stderr, "Expected some words to work with\n");
        exit(EXIT_FAILURE);
    }

    for (int i = optind; i < argc; ++i) {
        const char * s = argv[i];
        while (*s) {
            long delay_ms = rand() % (msec * 2);
            thrd_sleep(&(struct timespec){.tv_nsec=(delay_ms*1000*1000)}, NULL);
            int ch = *s++;
            if (upcase)
                ch = toupper(ch);
            putchar(ch);
            fflush(stdout);
        }
        putchar(' ');
        fflush(stdout);
    }
    putchar('\n');

    return EXIT_SUCCESS;
}
```

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

#include <arpa/inet.h>
#include <netinet/in.h>

#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>

#include <sys/sysinfo.h>
#include <unistd.h>


int main(void)
{
    printf("Uptime server version 0.1\n");

    struct addrinfo hints;
    memset(&hints, 0, sizeof hints);
    hints.ai_family = AF_INET;
    hints.ai_socktype = SOCK_STREAM;
    hints.ai_flags = AI_PASSIVE;

    struct addrinfo * res;
    int retval = getaddrinfo(NULL, "8081", &hints, &res);
    if (retval != 0) {
        fprintf(stderr, "getaddrinfo(): %s\n", gai_strerror(retval));
        exit(EXIT_FAILURE);
    }

    int sfd = socket(res->ai_family, res->ai_socktype, res->ai_protocol);
    if (sfd == -1) {
        freeaddrinfo(res);
        perror("socket()");
        exit(EXIT_FAILURE);
    }

    printf("sa_data: %s %d\n", res->ai_addr->sa_data, res->ai_addrlen);

    if (bind(sfd, res->ai_addr, res->ai_addrlen) != 0) {
        perror("bind()");
        close(sfd);
        freeaddrinfo(res);
        exit(EXIT_FAILURE);
    }

    printf("sa_data: %s\n", res->ai_addr->sa_data);

    freeaddrinfo(res);

    const int listen_backlog = 10;
    if (listen(sfd, listen_backlog) == -1) {
    perror("listen");
    exit(EXIT_FAILURE);
    }

    printf("Listening...\n");

    while (1) {
    struct sockaddr peer_addr;
    socklen_t peer_addr_size = sizeof peer_addr;
    int cfd = accept(sfd, &peer_addr, &peer_addr_size);
    if (cfd == -1) {
        perror("accept");
        exit(EXIT_FAILURE);
    }

    struct sysinfo info;
    sysinfo(&info);
    char buf[80];
    sprintf(buf, "%ld", info.uptime);
    send(cfd, buf, strlen(buf), 0);
    puts(buf);

    close(cfd);
    }
    close(sfd);

    return 0;
}

/*
code inspired by:
Hands-On Network Programming with C, Lewis Van Winkle, 2019
man7 documentation
The Linux Programming Interface, Michael Kerrisk, 2010
*/
```

Exercise

Use the uptime_server.c as a starting point. Try to come up with a potentially useful webservice returning information that can only be known by the device running the service. For example, return the CPU temperature and/or fanspeed. Also, you might want to experiment with reading data from the client, and/or returning a decorated (eg HTML) reply.

Outroduction