C Language and Security — Correction TD1

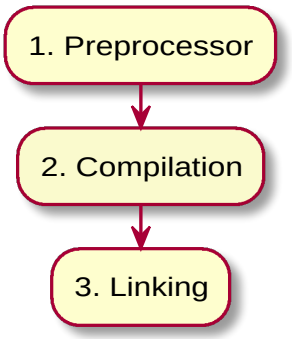
Drs. Mohamed EL BOUAZZATI, Camille Monière

September 2024

1 Exercise 1

1.1 Q1

The compilation *flow* is as follows (see the course):



The **compiler** (e.g., GCC) is the only tool! Indeed, it handles each step.

1.2 Q2

The tool for effectively scripting compilation, taking dependencies and access times into account, is (GNU) **Make**, with *makefiles*.

1.3 Q3

```
gcc -o main main.c
```

1.4 Q4

First, we represent the file tree:

```
dir
 |- main.c
 |- fonction_geo
       |- fonction_geo.h
       |- fonction_geo.c
```

We can then deduce the commands:

```
gcc -o main.o -c main.c
gcc -o fonction\_geo/fonction\_geo.o -c fonction\_geo/fonction\_geo.c
gcc -o main fonction\_geo/fonction\_geo.o main.o
```

2 Exercise 2

2.1 Q1

```
mkdir -p l3-snio/cetsec/td1/ex2
cd l3-snio/cetsec/td1/ex2
```

2.2 Q2

```
#include <stdio.h>

int main() {
    printf("Hello World\n");
    return 0;
}
```

2.3 Q3

```
  gcc -o hello main.c
  ./hello
  # Print "Hello World"
```

2.4 Q4

We add the function (version _adv as a bonus):

```

// Basic version
int is_leap_year(int year) {
  // First, check if divisible by 400, this is the strong rule
  if (year % 400 == 0) {
    return 1;
    // Then, check if divisible by 4 AND not divisible by 100
  } else if ((year % 4 == 0) && (year % 100 != 0)) {
    return 1;
  }
  // Otherwise, return zero.
  return 0;
}

int is_leap_year_adv(int year) {
  // More optimized version with Boolean algebra
  return !(year % 400) || (!(year % 4) && (year % 100));
```

To test, in the main function, we add:

```

int main() {
  // Test array
  const int yeartab[] = {2022, 1996, 1080, 3147};
  for (int i = 0; i < 4; i++) {
    // If lines 73 / 74 confuse you, look up 'C ternary operators'
    printf("%d leap year -> norm: %s | adv: %s.\n", yeartab[i],
```

2.5 Q5

We add:

```c
void affiche_date(int jour, int mois, int annee) {
  printf("%d-%d-%d\n", day, month, year);
}
```

and in `main()`, we add:

```c
  // clang-format off
  const int triplets[][3] = {
    {20, 10, 2023},
    {31, 4, 1998},
    {75, 100, -89},
    {29, 2, 1996}
  };

  for (int i = 0; i < 4; i++) {
    print_date(triplets[i][0], triplets[i][1], triplets[i][2]);
```

2.6 Q6

We add:

```c

int test_date(int day, int month, int year) {
  if (month < 1 || month > 12) {
    return 0;
  }

  if (year < 0) {
    return 0;
  }

  // Use an array of months to know the number of days
  const int monthtab[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
  const int leap_year = is_leap_year(year);
  if (month == 2) {
    if (day < 1 || day > (monthtab[1] + leap_year)) {
      // February case
      return 0;
    }
  } else if (day < 1 || day > monthtab[month - 1]) {
    return 0;
  }

  return 1;
```

and modify `affiche_date` as follows (no need to modify `main`, that's the benefit of procedural programming):

```c

void print_date(int day, int month, int year) {
  if (!test_date(day, month, year)) {
    printf("Error: Invalid date.\n");
  } else {
    printf("%d-%d-%d\n", day, month, year);
  }
```

2.7 Q7

We add:

```
52
53  void next_day(int day, int month, int year, int new_day[3]) {
54    // If the date is invalid, we don't waste time.
55    if (!test_date(day, month, year)) {
56      new_day[0] = -1;
57      new_day[1] = -1;
58      new_day[2] = -1;
59      return;
60    }
61
62    // Same technique as for testing...
63    const int monthtab[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
64    const int leap_year_add = (month == 2) * is_leap_year(year);
65
66    // Just math! Don't forget that `true` -> `1` in C
67    int next_day = day % (monthtab[month - 1] + leap_year_add) + 1;
68    int next_month = month % 12 + (next_day == 1);
69
70    new_day[2] = year + (next_month == 1);
71    new_day[1] = next_month;
72    new_day[0] = next_day;
```

and in `main`:

```
97
98    for (int i = 0; i < 4; i++) {
99      int nj[3] = {0, 0, 0};
100     next_day(triplets[i][0], triplets[i][1], triplets[i][2], nj);
101     print_date(nj[0], nj[1], nj[2]);
```

3 Exercise 3

3.1 Q1

```
1  mkdir -p l3-snio/cetsec/td1/ex3
2  cd l3-snio/cetsec/td1/ex3
```

3.2 Q2

We want the same result as in question 1.4. In a terminal, we execute:

```
1  mkdir fonctions_geo
2  touch main.c \
3        fonctions_geo/fonctions_geo.c \
4        fonctions_geo/fonctions_geo.h \
5        Makefile
```

3.3 Q3

The Makefile is as follows:

```
1  CC = gcc
2  CFLAGS := $(CFLAGS) -O2 -Wall -Werror -I.
3  LDFLAGS := $(LDFLAGS)
4
5  all: main
6
7  main: main.o fonctions_geo/fonctions_geo.o
8          $(CC) -o $@ $^ $(LDFLAGS)
9
10 main.o: main.c
11         $(CC) $(CFLAGS) -o $@ -c $^
12 fonctions_geo/fonctions_geo.o: fonctions_geo/fonctions_geo.c
13         $(CC) $(CFLAGS) -o $@ -c $^
14
15 .PHONY: clean clear
16 clean:
17         @rm -fv main.o fonctions_geo/fonctions_geo.o
18 clear: clean
19         @rm -fv main
```

Description of the lines:

- l1 Declaration of the variable `CC` (C Compiler);
- l2 Declaration of the variable `CFLAGS` (Compilation Flags, see the course “compilation options”);
- l3 Declaration of the variable `LDFLAGS` (Linker Flags);
- l5 Declaration of the default target, `all`, which depends on `main`;
- l7 Declaration of the target `main` (the executable), which depends on the object files (`main.o` and `fonctions_geo/fonctions_geo.o`);
- l8 Linking recipe, allowing to obtain `main` from `main.o` and `fonctions_geo/fonctions_geo.o`: “`$@`” gives the target, and “`$^`” returns the dependencies;
- l10 Declaration of the target `main.o` which depends on `main.c`;
- l11 Compilation recipe, allowing to obtain `main.o` from `main.c`;
- l12 Declaration of the target `fonctions_geo/fonctions_geo.o` which depends on `fonctions_geo/fonctions_geo.c`;
- l13 Compilation recipe, allowing to obtain `fonctions_geo/fonctions_geo.o` from `fonctions_geo/fonctions_`
- l15 Indication that the targets `clean` and `clear` must always be rebuilt;
- l16-19 Declaration and definition of the targets `clean` and `clear`;

3.4 Q4-6

See the reference manual and the files on blackboard.

4 Exercise 4

Coming soon!