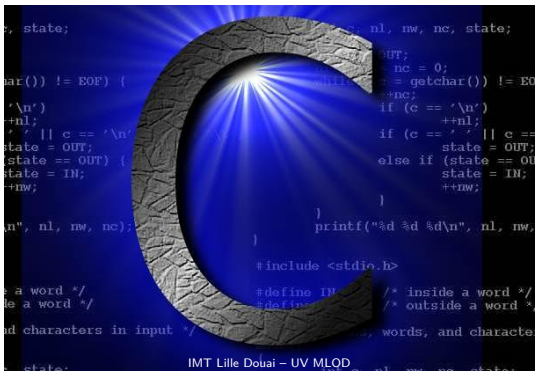
Algorithmique et langage C

— Outils: Make, GDB, ... —



IMT Lille Douai – UV MLQD

Luc Fabresse
luc.fabresse@imt-nord-europe.fr

Plan

Make

`http://www.gnu.org/software/make/`

Qu'est-ce que Make ?

Un programme qui permet d'automatiser la génération d'exécutables (ou d'autres fichiers) à partir de commandes de compilation, de fichiers sources et de règles de dépendances.

Un fichier Makefile contient :

- des règles
- des commandes
- la description des dépendances de compilation

La commande make

- lit le Makefile dans le répertoire courant
- exécute les commandes en fonction des règles

Exemple de Makefile

```
CFLAGS= -Wall -W --std=c99
CC=gcc

# Implicit rule
%.o: %.c %.h
  $(CC) $(CFLAGS) -o $@ -c $<

all: testDessin testLibBMP testMatrice testDessinMatrice testBlob mandelbrot

testMatrice: testMatrice.c matrice.o
  $(CC) $(CFLAGS) testMatrice.c matrice.o -o testMatrice

testLibBMP: testLibBMP.c libBMP.o
  $(CC) $(CFLAGS) testLibBMP.c libBMP.o -o testLibBMP

testDessin: testDessin.c libBMP.o dessin.o
  $(CC) $(CFLAGS) testDessin.c dessin.o libBMP.o -o testDessin

testDessinMatrice: testDessinMatrice.c dessinMatrice.o dessin.o libBMP.o matrice.o
  $(CC) $(CFLAGS) testDessinMatrice.c dessinMatrice.o dessin.o libBMP.o matrice.o -o testDessinMatrice

testBlob: testBlob.c dessin.o libBMP.o
  $(CC) $(CFLAGS) testBlob.c dessin.o libBMP.o -o testBlob

mandelbrot: mandelbrot.c libBMP.o complexe.o
  $(CC) $(CFLAGS) mandelbrot.c libBMP.o complexe.o -o mandelbrot

clean:
  rm -f *.o testMatrice testLibBMP testDessin testDessinMatrice smiley.bmp test.bmp dessinMatrice.bmp
```


La commande make

```
$ make
gcc -Wall -W --std=c99 -o libBMP.o -c libBMP.c
gcc -Wall -W --std=c99 -o dessin.o -c dessin.c
gcc -Wall -W --std=c99 testDessin.c dessin.o libBMP.o -o testDessin
gcc -Wall -W --std=c99 testLibBMP.c libBMP.o -o testLibBMP
gcc -Wall -W --std=c99 -o matrice.o -c matrice.c
gcc -Wall -W --std=c99 testMatrice.c matrice.o -o testMatrice
gcc -Wall -W --std=c99 -o dessinMatrice.o -c dessinMatrice.c
gcc -Wall -W --std=c99 testDessinMatrice.c dessinMatrice.o dessin.o libBMP.o matrice.o -o testDessinMatrice
gcc -Wall -W --std=c99 testBlob.c dessin.o libBMP.o -o testBlob
gcc -Wall -W --std=c99 -o complexe.o -c complexe.c
gcc -Wall -W --std=c99 mandelbrot.c libBMP.o complexe.o -o mandelbrot
```


Plan

CMake

outil pour générer un Makefile (ou un fichier pour un autre backend de compilation comme ninja)

Ecrire un CMakeList.txt

format plus descriptif et "simple" par rapport à un Makefile

Utiliser CMake

Exemple de CMakeList.txt

```
cmake_minimum_required(VERSION 3.10)
set(CMAKE_C_COMPILER "gcc")
set(CMAKE_C_FLAGS "-Wall -W -std=c99")
set(CMAKE_C_FLAGS_DEBUG "-O0 -g")

project(TP2)

add_executable(01-echangeContenu 01-echangeContenu.c)
add_executable(02-convexe-main 02-convexe-main.c 02-convexe.c)
add_executable(03-matrices-main 03-matrices.c 03-matrices-main.c)
add_executable(04-dates-main 04-dates-main.c 04-dates.c)
```

Utiliser la commande CMake

```
# create a build directory to isolate from source files
mkdir build
cd build

# generate a Makefile
# .. because CMakeList.txt is in the parent directory
cmake ..

# generate executables
make
```

Raylib

une bibliothèque C pour faire des jeux 2D/3D
`https://www.raylib.com/examples.html`

Exercice (vous devez le faire)

Re-compiler les exemples de Raylib en utilisant CMake.
cf. Readme : `https://github.com/raysan5/raylib/wiki/Working-on-GNU-Linux`

permet de mesurer le temps d'exécution d'un programme

Example

```
$ time a.out < data.txt > out.txt
0.15s user 0.00s system 98% cpu 0.155 total
```

real, user and sys (from man time)

- real : Elapsed real (wall clock) time used by the process, in seconds
- user : Total number of CPU-seconds that the process used directly (in user mode), in seconds
- sys : Total number of CPU-seconds used by the system on behalf of the process (in kernel mode), in seconds

« L'erreur est humaine, mais un vrai désastre nécessite un ordinateur. »
[William Henry Gates III] (*Bill Gates*)


```c
#include <stdio.h>

int main(void)
{
  int somme, n_premiers_entiers, indice ;

  printf("calcule la somme des n premiers entiers, entrez n : ");
  scanf("%d", n_premiers_entiers);

  indice = 0;
  somme = 0;
  while(indice <= n_premiers_entiers){
    somme += indice;
    indice++;
  }
  printf("la somme est %d\n", somme);

  return 0;
}
```



```
$ gcc sommeEntiers.c -o sommeEntiers
$ ./sommeEntiers
calcule la somme des n premiers entiers, entrez n : 3
1
3
Segmentation fault
```



```
$ gcc sommeEntiers.c -o sommeEntiers
$ ./sommeEntiers
calcule la somme des n premiers entiers, entrez n : 3
1
3
Segmentation fault
```

Que faire ?

3 solutions dont 2 mauvaises

Solution 1

“Môssieur, ça marche pôa !”

Solution 2

Insérer des printf partout

```c
#include <stdio.h>

int main(void)
{
  int somme, n_premiers_entiers, indice ;

  printf("calcule la somme des n premiers entiers, entrez n : ");
  scanf("%d", n_premiers_entiers);
printf("erreur ligne 9");
  indice = 0;
  somme = 0;
printf("erreur ligne 12");
  while(indice <= n_premiers_entiers){
printf("erreur ligne 14");
    somme += indice;
printf("erreur ligne 16");
    indice++;
printf("erreur ligne 18");
  }
  printf("la somme est %d\n", somme);

  return 0;
}
```


Solution 2

Insérer des printf partout

```c
#include <stdio.h>

int main(void)
{
  int somme, n_premiers_entiers, indice ;

  printf("calcule la somme des n premiers entiers, entrez n : ");
  scanf("%d", n_premiers_entiers);
printf("erreur ligne 9");
  indice = 0;
  somme = 0;
printf("erreur ligne 12");
  while(indice <= n_premiers_entiers){
printf("erreur ligne 14");
    somme += indice;
printf("erreur ligne 16");
    indice++;
printf("erreur ligne 18");
  }
  printf("la somme est %d\n", somme);

  return 0;
}
```

Impossible sur de “vrais” programmes !

Solution 3 : Utiliser un debugger

Avantages

- Ne modifie pas le code
- Extrêmement puissant

Inconvénient

- Apprentissage

GDB : The Gnu Debugger

Permet :

- de lancer un programme et contrôler son comportement
- de stopper l'exécution d'un programme suivant des conditions
- d'examiner ce qui s'est passé quand un programme s'est arrêté
- de changer votre programme afin d'expérimenter des corrections ou de découvrir le(s) bug(s)

Exemple en ligne de commande

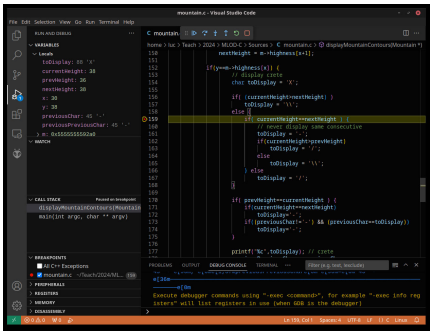
```
$ gcc -g sommeEntiers.c -o sommeEntiers
$ ./sommeEntiers
calcule la somme des n premiers entiers, entrez n : 3
1
3
Segmentation fault
$ gdb sommeEntiers
...
done

(gdb) run
Starting program: a.out
Reading symbols for shared libraries +. done
calcule la somme des n premiers entiers, entrez n : 4

Program received signal EXC_BAD_ACCESS, Could not access memory.
Reason: KERN_INVALID_ADDRESS at address: 0x000000005fc01052
0x00007fff852212b2 in __svfscanf_l ()
(gdb)
```


GUI pour GBD

- VSCode debugger (https://code.visualstudio.com/docs/cpp/cpp-debug)
- https://www.gdbgui.com/
- https://www.onlinegdb.com/online_c_compiler



Plan

valgrind

Pour détecter les fuites mémoires

```
valgrind --leak-check=yes --leak-check=full --show-leak-kinds=all --show-reachable=no \
./04-dates-main < ../04-dates.txt
```

```
==7975== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.
==7975== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==7975== Command: ./04-dates-main
==7975==
...
==7975==
==7975== HEAP SUMMARY:
==7975== in use at exit: 0 bytes in 0 blocks
==7975== total heap usage: 6 allocs, 6 frees, 5,168 bytes allocated
==7975==
==7975== All heap blocks were freed -- no leaks are possible
==7975==
==7975== For counts of detected and suppressed errors, rerun with: -v
==7975== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```


LeakSanitizer is a run-time memory leak detector. It can be combined with AddressSanitizer to get both memory error and leak detection, or used in a stand-alone mode.

https://clang.llvm.org/docs/LeakSanitizer.html

```
$ clang -fsanitize=address -g memory-leak.c ; ASAN_OPTIONS=detect_leaks=1 ./a.out
==23646==ERROR: LeakSanitizer: detected memory leaks
Direct leak of 7 byte(s) in 1 object(s) allocated from:

0 0x4af01b in __interceptor_malloc /projects/compiler-rt/lib/asan/asan_malloc_linux.cc:52:3

1 0x4da26a in main memory-leak.c:4:7

2 0x7f076fd9cec4 in __libc_start_main libc-start.c:287
SUMMARY: AddressSanitizer: 7 byte(s) leaked in 1 allocation(s).
```