

Pratique

Langage C

Aléatoire

Cheat Sheet C...

Trouvez la(es) votres

Geografi adalah ilmu yang
membahas tentang bumi dan manusia
dalam ruang. Dalam seluk
ruang ini, Geografi tidak
terpencil, ada sebuah
teori yang bisa disebut sebagai
teori plagiaris karena ini
adalah teori yang bisa disebut
sebagai plagiaris. Jadi apakah
teori ini bisa disebut
sebagai plagiaris atau tidak.
Sebenarnya, ini adalah teori yang benar.
Jadi, ini adalah teori yang benar.

Geografi

1) Geografi adalah ilmu yang...

Compilation/debugage et exécution

- Via IDE : (VS code, Codeblocks, ?)
- En ligne de commande : gcc/gdb
- L'outil make
- Compilateur fonction du SE :
 - Windows (like Linux) : Cygwin / MinGw-w64
 - Linux : gcc de base
 - MacOS : gcc (clang)

Pas très haut niveau... donc

- Manque de portabilité sur pas mal d'aspects
 - Codage de l'information
 - Aspects graphiques

Compiler

Avec l'IDE, ok, mais...

Important de savoir le faire en ligne de commande via gcc

Le compilateur GCC

Options	Description
<code>gcc -E</code>	Only invokes the preprocessor without compiling or using the linker.
<code>gcc -g</code>	Enables debug information, needed before gdb
<code>gcc -c</code>	Compiles source files to object files without linking to any other object files.
<code>gcc -I dir</code>	Includes the directories of header files
<code>gcc -llib</code>	link the code with the library files
<code>gcc file.c -o output_file</code>	Build the output generated to output file
<code>gcc -Wall -pedantic</code>	Enables all warning messages during the compilation.

Some interesting Gcc command line options <https://renenyffenegger.ch/notes/development/languages/C-C-plus-plus/GCC/options/index>
Option controlling the King of Output, GNU gcc onlinedocs, <https://gcc.gnu.org/onlinedocs/gcc/Overall-Options.html#Overall-Options>

Programme monolithique

- Option 1 : Tout est dans le même fichier on compile donc tout d'un coup

```
$gcc -Wall -pedantic testbaseentier.c
```

Si le fichier est gros
c'est peu pratique ! Et peu
modulaire...

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

void swap_int(int *v1, int *v2);

int main(int argc, char const *argv[]){
    int a = 10;
    int b = 15;
    printf("Avant appel à swap_int le contenu de a est %d et
    celui de b est %d\n", a, b);
    swap_int(&a,&b);
    assert(a==15 && b==10);
    printf("Après appel à swap_int le contenu de a est %d et
    celui de b est %d\n", a, b);
    return 0;
}

void swap_int(int *v1, int *v2){
    int temp = *v1;
    *v1 = *v2;
    *v2 = temp;
}
```

testbase.c



Compiler un programme C fait de plusieurs fichiers

Compiling C programs with multiple files, Jacob Sorber
https://www.youtube.com/watch?v=2YfM-HxQd_8

- Option : Tout compiler d'un coup

```
$gcc -Wall -pedantic testbase.c utilswap.c
```

S'il y a beaucoup de fichiers cela peut être long...

```
#ifndef __UTILSWAP_H__
#define __UTILSWAP_H__

void swap_int(int *v1, int *v2);

#endif
```

utilswap.h

```
#include "utilswap.h"

void swap_int(int *v1, int *v2){
    int temp = *v1;
    *v1 = *v2;
    *v2 = temp;
}
```

utilswap.c

```
#include "utilswap.h"
#include <stdio.h>
#include <assert.h>

int main(int argc, char const *argv[]){
    int a = 10;
    int b = 15;
    printf("Avant appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    swap_int(&a,&b);
    assert(a==15 && b==10);
    printf("Après appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    return 0;
}
```

testbase.c



Compiler un programme C fait de plusieurs fichiers

Compiling C programs with multiple files, Jacob Sorber
https://www.youtube.com/watch?v=2YfM-HxQd_8

- Option 2 : compiler les fichiers un à un (générer les .o) puis les linker à la fin

```
$gcc -Wall -pedantic -c utilswap.c
```

```
$gcc -Wall -pedantic -c testbase.c
```

```
$gcc -Wall -pedantic utilswap.c testbase.o -o testbase
```

On ne recompile que ce qui a été changé

```
#ifndef __UTILSWAP_H__
#define __UTILSWAP_H__

void swap_int(int *v1, int *v2);

#endif
```

utilswap.h

```
#include "utilswap.h"

void swap_int(int *v1, int *v2){
    int temp = *v1;
    *v1 = *v2;
    *v2 = temp;
}
```

utilswap.c

```
#include "utilswap.h"
#include <stdio.h>
#include <assert.h>
```

testbase.c

```
int main(int argc, char const *argv[]){
    int a = 10;
    int b = 15;
    printf("Avant appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    swap_int(&a,&b);
    assert(a==15 && b==10);
    printf("Après appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    return 0;
}
```

Compiler un programme C fait de plusieurs fichiers

- Option 3 : Utiliser un outil de build de code type Make

Compiling C programs with multiple files, Jacob Sorber
https://www.youtube.com/watch?v=2YfM-HxQd_8

Un outil de build de code : make

make et les makefiles,

Ou bien...

ant, maven, rake, ...

Learn Make in 60 secondes, Jacob Sorber,
<https://www.youtube.com/watch?v=a8mPKBxQ9No>
Makefile-Related Videos, Jacob Sorber,
<https://www.youtube.com/playlist?list=PL9IEIJKnBJiEPxenuhKU7J5smY4XjFnvg>



UNIVERSITÀ DI CORSICA
PASQUALE PAOLI

Le makefile un fichier nommé Makefile

- Il décrit les règles dictant la manière de faire le « build » de votre programme

```
#ifndef __UTILSWAP_H__
#define __UTILSWAP_H__

void swap_int(int *v1, int *v2);

#endif
```

utilswap.h

```
#include "utilswap.h"

void swap_int(int *v1, int *v2){
    int temp = *v1;
    *v1 = *v2;
    *v2 = temp;
}
```

utilswap.c

```
#include "utilswap.h"
#include <stdio.h>
#include <assert.h>

int main(int argc, char const *argv[]){
    int a = 10;
    int b = 15;
    printf("Avant appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    swap_int(&a, &b);
    assert(a==15 && b==10);
    printf("Après appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    return 0;
}
```

testbaseapp.c

```
$ make
gcc -c utilswap.c -o utilswap.o
gcc -Wall -pedantic -g -o testbaseapp testbaseapp.c utilswap.o
$ ./testbaseapp
Avant appel à swap_int le contenu de a est 10 et celui de b est 15
Après appel à swap_int le contenu de a est 15 et celui de b est 10
$ make clean
rm *.o testbaseapp
```

```
CC=gcc
CFLAGS=-Wall -pedantic -g
```

```
all: testbaseapp
```

```
utilswap.o: utilswap.h utilswap.c
$(CC) -c utilswap.c -o utilswap.o
```

```
testbaseapp: testbaseapp.c utilswap.o
$(CC) $(CFLAGS) -o testbaseapp testbaseapp.c utilswap.o
```

```
clean:
rm *.o testbaseapp
```

Makefile

-pedantic pour la
compatibilité norme ANSI

Make, variables automatiques (1)

- Utilisées pour faciliter la maintenance de vos Makefile

```
#ifndef __UTILSWAP_H__
#define __UTILSWAP_H__

void swap_int(int *v1, int *v2);

#endif
```

utilswap.h

```
#include "utilswap.h"

void swap_int(int *v1, int *v2){
    int temp = *v1;
    *v1 = *v2;
    *v2 = temp;
}
```

utilswap.c

```
#include "utilswap.h"
#include <stdio.h>
#include <assert.h>

int main(int argc, char const *argv[]){
    int a = 10;
    int b = 15;
    printf("Avant appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    swap_int(&a,&b);
    assert(a==15 && b==10);
    printf("Après appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    return 0;
}
```

testbaseapp.c

\$@ représente le nom de la cible
\$^ représente les dépendances

```
CC=gcc
CFLAGS=-Wall -pedantic -g

all: testbaseapp

utilswap.o: utilswap.h utilswap.c
$(CC) $(CFLAGS) -c $^

testbaseapp: testbaseapp.c utilswap.o
$(CC) $(CFLAGS) -o $@ $^

clean:
rm *.o testbaseapp
```

Makefile

Make, variables automatiques (2)

- Utilisées pour faciliter la maintenance de vos Makefile

```
#ifndef __UTILSWAP_H__
#define __UTILSWAP_H__

void swap_int(int *v1, int *v2);

#endif
```

utilswap.h

```
#include "utilswap.h"

void swap_int(int *v1, int *v2){
    int temp = *v1;
    *v1 = *v2;
    *v2 = temp;
}
```

utilswap.c

```
#include "utilswap.h"
#include <stdio.h>
#include <assert.h>

int main(int argc, char const *argv[]){
    int a = 10;
    int b = 15;
    printf("Avant appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    swap_int(&a, &b);
    assert(a==15 && b==10);
    printf("Après appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    return 0;
}
```

testbaseapp.c

\$@ représente le nom de la cible

^ représente les dépendances

%.o: %.h %.c généralise les traitements

```
CC=gcc
CFLAGS=-Wall -pedantic -g
```

```
all: testbaseapp
```

```
%.o: %.h %.c
$(CC) $(CFLAGS) -c $^
```

```
testbaseapp: testbaseapp.c utilswap.o
$(CC) $(CFLAGS) -o $@ $^
```

```
clean:
rm *.o testbaseapp
```

Makefile

Make, variables automatiques (3)

- Utilisées pour faciliter la maintenance de vos Makefile

```
#ifndef __UTILSWAP_H__
#define __UTILSWAP_H__

void swap_int(int *v1, int *v2);

#endif
```

utilswap.h

```
#include "utilswap.h"

void swap_int(int *v1, int *v2){
    int temp = *v1;
    *v1 = *v2;
    *v2 = temp;
}
```

utilswap.c

```
#include "utilswap.h"
#include <stdio.h>
#include <assert.h>

int main(int argc, char const *argv[]){
    int a = 10;
    int b = 15;
    printf("Avant appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    swap_int(&a, &b);
    assert(a==15 && b==10);
    printf("Après appel à swap_int le contenu de a est %d et celui de b est %d\n", a, b);
    return 0;
}
```

testbaseapp.c

LD_FLAGS = les librairies à utiliser -lncurses -lm etc...

OBJFILES = liste des fichiers objets constituant le programme final

TARGET = nom du programme final

```
CC=gcc
CFLAGS=-Wall -pedantic -g
LD_FLAGS =
OBJFILES = testbaseapp.o utilswap.o
TARGET = testbaseapp
```

```
all: $(TARGET)
```

```
%.o: %.h %.c
    $(CC) $(CFLAGS) -c $^
```

```
$(TARGET): $(OBJFILES)
    $(CC) $(CFLAGS) -o $(TARGET) $(OBJFILES) $(LD_FLAGS)
```

```
clean:
    rm $(OBJFILES) $(TARGET)
```

Makefile

« Débuguer »

Certes via votre IDE, mais aussi en ligne de commandes via
l'utilitaire gdb (The Gnu project Debugger), lldb

What Languages does GDB Support?

GDB supports the following languages (in alphabetical order):

- Ada
- Assembly
- C
- C++
- D
- Fortran
- Go
- Objective-C
- OpenCL
- Modula-2
- Pascal
- Rust

La page officielle : <https://www.sourceware.org/gdb/>

GDB en 60 secondes (Jacob Sorber) : <https://www.youtube.com/watch?v=mfmXcbiRs0E>

Un exemple simple

Cherchez l'erreur !
En débbugant
factorial.c

```
Enter the number : 3
The factorial number of 3 is 61831800
```

```
#include <stdio.h>

int factorial(int n){
    int result;
    for (int i=1;i<=n;i++){
        result *= i;
    }
    return result;
}

int main(int argc, char const *argv[])
{
    int num;
    printf("Enter the number : ");
    scanf("%d",&num);
    printf("The factorial number of %d is
    %d\n",num,factorial(num));
    return 0;
}
```



Les points importants pour débbuguer

- Il faut compiler en incluant les informations nécessaires au debug **-g**

```
>gcc -g -o factorial factorial.c
```

- Lancer le debugger (**gdb** ou **lldb**) sur l'exécutable

```
>gdb factorial
```

```
>lldb factorial
```

- Positionner un point d'arrêt dans le programme à la ligne voulue

```
(gdb) break 10
```

```
(lldb) b main
```

- Lancer le debugage

```
(gdb) run
```

```
(lldb) run
```

- Ecrire le contenu des variables

```
(gdb) print num
```

```
(lldb) p num
```

•c, continue : continue jusqu'au prochain point d'arrêt
•n, next : exécute la prochaine instruction sans rentrer dedans
•s, step : idem next mais entre dans la fonction et l'exécute ligne par ligne
•p, print
•ENTER réitère la commande précédente

Une gdb Cheat sheet complète :

<https://users.ece.utexas.edu/~adnan/gdb-refcard.pdf>



UNIVERSITÀ DI CORSICA
PASQUALE PAOLI

Explorer la mémoire pendant l'exécution via gdb

```
#include <stdio.h>
#include <stdint.h> //Pour utiliser des types entiers à taille fixée

typedef struct
{
    int8_t hours;
    uint32_t micros;
    uint16_t seconds;
} timestuff_t;

int main(int argc, char const *argv[])
{
    timestuff_t t= {.hours=6, .micros=0x12345678, .seconds = 0xDEAD};
    printf("%lu", sizeof(t));
    return 0;
}
```

Explorer la mémoire pendant l'exécution via gdb

```
$ gcc -g -o testmemory testmemorystructgdb.c
$ lldb testmemory
(lldb) target create "testmemory"
Current executable set to 'testmemory' (arm64).
(lldb) b main
Breakpoint 1: where = testmemory`main + 40 at testmemorystructgdb.c:13:17, address =
0x0000000100003f64
(lldb) run
Process 45769 launched: 'testmemory' (arm64)
Process 45769 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
    frame #0: 0x0000000100003f64 testmemory`main(argc=1, argv=0x000000016fdff428) at
testmemorystructgdb.c:13:17
    10
    11  int main(int argc, char const *argv[])
    12  {
-> 13      timestuff_t t= {.hours=6, .micros=0x12345678, .seconds = 0xDEAD};
    14      printf("%lu",sizeof(t));
    15      return 0;
    16  }
Target 0: (testmemory) stopped.
(lldb)
```

Combien d'octets devrait faire la structure t ?
Combien en fait elle réellement ?
Testez avec GDB en regardant le contenu de la mémoire...

How to examine memory in GDB ?, Jacob Sorber,
https://www.youtube.com/watch?v=A_pV61xFty8

x	[Nuf] expr	examine memory at address <i>expr</i> ; optional format spec follows slash
N		count of how many units to display
u		unit size; one of
		b individual bytes
		h halfwords (two bytes)
		w words (four bytes)
		g giant words (eight bytes)
f		printing format. Any print format, or
		s null-terminated string
		i machine instructions

Cheat Sheet GDB,
<https://users.ece.utexas.edu/~adnan/gdb-refcard.pdf>

```
//File testmemorystructgdb.c
#include <stdio.h>
#include <stdint.h> //Pour utiliser des entiers à taille fixée

typedef struct
{
    int8_t hours;
    uint32_t micros;
    uint16_t seconds;
} timestuff_t;

int main(int argc, char const *argv[])
{
    timestuff_t t= {.hours=6, .micros=0x12345678, .seconds =
0xDEAD};
    printf("%lu",sizeof(t));
    return 0;
}
```

Explorer la mémoire pendant l'exécution via gdb

```
$llldb testmemory
(llldb) target create "testmemory"
Current executable set to '/testmemory' (arm64).
(llldb) b 14
Breakpoint 1: where = testmemory`main + 64 at testmemorystructgdb.c:14:5, address =
0x0000000100003f7c
(llldb) run
Process 46008 launched: '/testmemory' (arm64)
Process 46008 stopped
* thread #1, queue = 'com.apple.main-thread', stop reason = breakpoint 1.1
  frame #0: 0x0000000100003f7c testmemory`main(argc=1, argv=0x000000010fdff428) at
testmemorystructgdb.c:14:5
    11  int main(int argc, char const *argv[])
    12  {
    13      timestuff_t t= {.hours=6, .micros=0x12345678, .seconds = 0xDEAD};
-> 14      printf("%lu",sizeof(t));
    15      return 0;
    16  }
    17

Target 0: (testmemory) stopped.
(llldb) p sizeof(t)
(unsigned long) $0 = 12
(llldb) p t
(timestuff_t) $1 = (hours = '\x06', micros = 305419896, seconds = 57005)
(llldb) p &t
(timestuff_t *) $2 = 0x000000010fdff290
(llldb) x &t
0x10fdff290: 06 00 00 00 78 56 34 12 ad de 00 00 01 80 60 29 ....xV4.....`
0x10fdff2a0: 28 f4 df 6f 01 00 00 00 01 00 00 00 00 00 00 00 (...o.....
(llldb) x/12 &t
0x10fdff290: 0x00000006 0x12345678 0x0000dead 0x29608001
0x10fdff2a0: 0x6fdff428 0x00000001 0x00000001 0x00000000
0x10fdff2b0: 0x6fdff400 0x00000001 0x0001108c 0x00000001
(llldb) x/12x &t
0x10fdff290: 0x00000006 0x12345678 0x0000dead 0x29608001
0x10fdff2a0: 0x6fdff428 0x00000001 0x00000001 0x00000000
0x10fdff2b0: 0x6fdff400 0x00000001 0x0001108c 0x00000001
(llldb) x/12xb &t
0x10fdff290: 0x06 0x00 0x00 0x00 0x78 0x56 0x34 0x12
0x10fdff298: 0xad 0xde 0x00 0x00
(llldb) x/3xw &t
0x10fdff290: 0x00000006 0x12345678 0x0000dead
(llldb) quit
```

x <i>[Nuf]</i> <i>expr</i>	examine memory at address <i>expr</i> ; optional format spec follows slash
N	count of how many units to display
u	unit size; one of
	b individual bytes
	h halfwords (two bytes)
	w words (four bytes)
	g giant words (eight bytes)
f	printing format. Any print format, or
	s null-terminated string
	i machine instructions

Cheat Sheet GDB,
<https://users.ece.utexas.edu/~adnan/gdb-refcard.pdf>

```
#include <stdio.h>
#include <stdint.h> //Pour utiliser des types entiers à taille fixée

typedef struct
{
    int8_t hours;
    uint32_t micros;
    uint16_t seconds;
} timestuff_t;

int main(int argc, char const *argv[])
{
    timestuff_t t= {.hours=6, .micros=0x12345678, .seconds = 0xDEAD};
    printf("%lu",sizeof(t));
    return 0;
}
```

La structure devrait faire 7 octets... Elle en fait 12...

How to examine memory in GDB ?, Jacob Sorber,
https://www.youtube.com/watch?v=A_pV61xFty8