

Mémento du langage C

Compiler (= produire un exécutable)

- Tout en un: `gcc -Wall -Wextra -g aze.c -o aze`
- (1) compilation: `gcc -Wall -Wextra -g -c aze.c`
- (2) édition de liens: `gcc aze.o toto.o -o prog`

Pré-processeur (= cherche/remplace automatique)

```
/* bloc de commentaires */
// Commentaire jusqu'à la fin de ligne
#include <libmodule.h>
#include "usermodule.h"
#define TAILLE 512
#define MAX(a,b) ((a)>(b)?(a):(b))
```

Blocs d'instructions (= morceaux de programme)

```
expression; // instruction
{ instructions... } // bloc
if (expression) {bloc}
if (expression) {bloc} else {bloc}
switch (expression) {
    case constante1: instructions... break;
    case constante2: instructions... break;
    default: instructions...
}
while (expression) {bloc}
for (init ; condition; increment) {bloc}
do {bloc} while (cond) // rarement utile
break; // termine le bloc (boucle ou switch)
continue; // prochaine itération
return expr; // la fonction retourne la valeur
```

Structure des programmes (= contenu des fichiers)

- Fichiers d'entête: que des déclarations
©, #include, #define, prototypes fctions globales
- Fichiers d'implém: déclarations et implémentations
©, #include, #define, prototypes fctions statiques, définition des fonctions
- Fichier principal: comme implem + fonction main
`int main(int argc, char** argv)`

Identificateurs (= noms de variables et fonctions)

- Lettre suivie par des lettres ou chiffres ou souligné
[a-zA-Z][a-zA-Z0-9_]*
- Mots réservés : `auto break case char const continue default do double else entry enum extern float for goto if int long register return short signed sizeof static struct switch typedef union unsigned void volatile while`

Déclarations, initialisations et prototypes

```
int i = 0; char c = '\n';
char* str = "hello"; char buf[BUFSIZ];
double x = 3.14; char* p = NULL;
double values[MAX] = {1, 2, 3};
typedef enum { FALSE, TRUE } bool_t;
typedef struct keyval {
    char* key;
    unsigned int val;
} keyval_t;
keyval_t klist[ ] = {{ "NSW", 0 }, { "Vic", 5 }};
void compute_ranking(int from);
```

Exemples de littéraux (= valeurs de variables)

```
123 0xAf0C 057 5L 3ul entiers (int)
3.1415 3f 1.29e-23 réels (double)
'x' '\t' '\033' caractères (char)
"hello" "abc\n" "" chaînes (char*)
```

Séquences d'échappement (dans les chaînes)

```
\t tabulation \n nouvelle ligne
\' caractère \' \" caractère "
\\ caractère \ \123 code ascii en octal
\0 caractère nul liste complète: man ascii
```

Taille des types (en fonction de l'architecture)

	x86	win64	amd64	
short	2	2	2	(en octets)
int	4	4	4	cf. sizeof()
long	4	4	8	char: tjs 1
long long	8	8	8	float: tjs 4
pointeurs	4	8	8	double: tjs 8

Portée et durée de vie des identificateurs

	Portée	Durée de vie
Globale	partout	infinie
Globale static	fichier seulement	infinie
Locale static	bloc seulement	infinie
Locale	bloc seulement	bloc

Les globales sont **extern** par défaut.

Bibliothèque standard: #include <stdlib.h>

```
atoi(s) atof(s) Chaîne vers int ou double
abs(n) val absolue rand() nb pseudo-aléatoire
malloc(n) calloc(1,n) Alloue n octets
realloc(p,n) Redimensionne p
free(p) Libère bloc exit(n) Termine prog (code n)
```

Opérateurs (priorité décroissante)

() [] . ->	parenthèses, tableau, structure, → structure pointée
++ -- ! * & ~ sizeof() (type)	Incr/décément, moins unaire, non déréférencemnt, adresse, complem.1 taille d'objet, transtypage ←
* / % + -	Opérateurs arithmétiques →
<< >>	Décalage binaire à gauche/droite →
< <= >= >	Opérateurs relationnels →
== != &	Tests d'(in)égalité; ET binaire →
^	OU exclusif (XOR); OU inclusif →
C ? V : F	Condition ternaire ←
= += -= *= &=	Affectation (avec modification) ←
,	Virgule (séquence d'expressions) →

```
c char p pointeur s chaîne char* d double
n int l long fh fichier FILE* b buffer char[]
```

I/O standard: #include <stdio.h>

```
stdin stdout stderr Flux de sortie (FILE*)
EOF NULL Des constantes utiles
fopen(s, "rwab+") Ouvre fichier, retourne un fh
fclose(fh) Ferme le fichier
fgetc(fh) getchar() Lit un cara (EOF si fini)
fputc(fh,c) putchar(c) Écrit un caractère
fread(b,1,n,fh) Lit un bloc de taille au plus n
fwrite(b,1,n,fh) Écrit un bloc de taille n
printf(fmt, list) Affichage formaté sur stdout
fprintf(fh,fmt, list) Affichage formaté sur fh
sprintf(b,fmt, list) Affichage formaté dans buffer
scanf(fmt, list) Lecture formaté depuis stdin
fscanf(fh, fmt, list) Lecture formaté depuis fh
sscanf(s, fmt, list) Lecture formaté depuis chaîne
Format de printf: %[largeur]/[precision]/type
```

```
d entier o octal x hexadécimal
f flottant g général e exponentiel (scientifique)
c caractère s chaîne p pointeur
```

Pour `scanf`, la liste contient des adresses

Utiliser des chaînes: #include <string.h>

```
strlen(s) longueur (sans le \0)
strcmp(s1,s2) compare. 1:<lex 0:==lex -1:>lex
strcpy(d,s) copie d←s strcat(d,s) ajout d←ds
strchr(s,c) cherche c strstr(s,s2) cherche s2
strtok(s,delim) Découpe s en tokens
```

Premier programme

```
1 #include <stdio.h>
2 int main(int argc, char** argv) {
3     int reponse;
4     for (int i=0; i<5; i++) {
5         printf("Quelle est la réponse?\n");
6         scanf("%d", &reponse);
7         if (reponse == 42) {
8             printf("Bravo, vous avez trouvé!\n");
9             exit(0); // termine le programme
10        } else {
11            printf("Non, ce n'est pas %d\n", reponse);
12        }
13    }
14    printf("Dommage, c'est perdu\n");
15    return 1;
16 }
```

Lire un entier ou une chaîne

```
1 int i;                char str[512];
2 scanf("%d", &i);      scanf("%s", str);
```

Lire un caractère et vider le buffer

```
1 char data, c;
2 scanf("%c", &data);
3 while ((c = getchar()) != '\n' && c != EOF) { }
```

Lecture d'un fichier ligne à ligne

```
1 char* line = NULL; size_t lgr = 0; ssize_t read;
2 while ((read = getline(&line, &lgr, fh) != -1){
3     /* ce que vous voulez */
4 }
5 free(line);
```

Pour savoir programmer en C, il faut programmer en C.

- Entraînez-vous, faites tous les TP et (mini-)projets.
- Maîtrisez vos outils: [c]make, valgrind, asan, gdb.
- Lisez les messages d'erreur pour les comprendre.
- Les logiciels libres comme terrain d'entraînement.
- Le C sous Windows, c'est encore plus dur.

Quelques bons livres sur le C:

- *Apprenez à programmer en C* sur OpenClassRoom.
- *Modern C*, livre libre par Jens Gustedt.
- fr.wikibooks.org/wiki/Programmation_C
- en.wikibooks.org/wiki/C_Programming

main.c

```
1 #include "point.h"
2
3 int main(int argc, char** argv) {
4     point_t* p1 = point_create(10, 20);
5     point_t* p2 = point_create(20, 40);
6     point_t* total = point_add(p1, p2);
7     point_display(total);
8     point_free(p1);
9     point_free(p2);
10    point_free(total);
11 }
```

point.h

```
1 #ifndef POINT_H
2 #define POINT_H
3
4 typedef struct point point_t; // implem cachée
5
6 point_t *point_create(double x, double y);
7 void point_free(point_t* p);
8
9 point_t* point_add(point_t* p1, point_t* p2);
10 void point_display(point_t* p);
11 #endif /* POINT_H */
```

Makefile

```
1 LDFLAGS=-g
2 CFLAGS= -Wall -Wextra -Wno-unused-parameter
3 CFLAGS:= $(CFLAGS) -Werror
4 prog: main.o point.o
5     gcc $(LDFLAGS) $^ -o $@
6 main.o: main.c point.h
7     gcc $(CFLAGS) -c $<
8 point.o: point.c point.h
9     gcc $(CFLAGS) -c $<
10 clean:
11     rm -f main.o point.o prog
12 .PHONY: clean # Cible à faire toujours
```

CMakeLists.txt (pour cmake)

```
1 project(MyProject)
2 set(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} -Wall")
3 add_executable(prog main.c point.c point.h)
```

point.c

```
1 #include "point.h"
2 #include <stdlib.h>
3 #include <stdio.h>
4
5 int maxRank = 0;
6 struct point {
7     int rank;
8     double x, y;
9 };
10 point_t* point_create(double x, double y) {
11     point_t* res = malloc(sizeof(point_t));
12     res->rank = maxRank++;
13     res->x = x;
14     res->y = y;
15     return res;
16 }
17 void point_free(point_t* p) {
18     free(p);
19 }
20 point_t* point_add(point_t* p1, point_t* p2) {
21     return
22         point_create(p1->x + p2->x, p1->y + p2->y);
23 }
24 void point_display(point_t* p) {
25     printf("%d: %g, %g\n", p->rank, p->x, p->y);
26 }
```

Exemple de session avec make

```
1 $ make
2 gcc -Wall -Wextra -Werror -c main.c
3 gcc -Wall -Wextra -Werror -c point.c
4 gcc -g main.o point.o -o prog
5 $ ./prog
6 2: 30, 60
7 $ make clean
8 rm -f main.o point.o prog
```

Exemple de session avec cmake

```
1 $ cmake . # /\ ceci écrase le Makefile
2 (tests automatique du système)
3 $ make
4 (compilation des différents fichiers)
5 $ ./prog
6 2: 30, 60
```