

Programmation avancée

Introduction et Rappel

Walter Rudametkin

Walter.Rudametkin@polytech-lille.fr
<https://rudametw.github.io/teaching/>

Bureau F011
Polytech'Lille

29 janvier 2016

Moi...

- ▶ Je suis étranger (hors UE)...
- ▶ J'ai un accent...
- ▶ Je me trompe beaucoup en français
 - ▶ (et en info, et en math, et ...)
 - ▶ N'hésitez pas à me corriger ou à me demander de répéter
- ▶ Je commence à enseigner
 - ▶ J'accepte des critiques (constructifs) et surtout des recommandations
 - ▶ N'hésitez pas à poser des questions

Remarque

Ce cours est très très très largement inspirés (i.e., copié) de ceux de Nathalie Devesa (MdC Polytech'Lille), qui à son tour s'est inspiré de Bernard Carré et de Laure Gonnord.

Volume horaire et évaluation

Volume horaire

- ▶ 22h CM
- ▶ 14h TD
- ▶ 22h TP
- ▶ 10h ET/Projet

Evaluation

- ▶ DS (2h) — 2 ECTS
- ▶ TP noté (2h) — 1.5 ECTS
- ▶ Projet — 1.25 ECTS
- ▶ Total: 4.75 ECTS

Cont. de Programmation Structurée

- ▶ Pr. Laurent Grisoni au S5
- ▶ Bases de l'algorithmique
 - ▶ Pseudo-code, décomposition de problèmes en sous-problèmes, complexité
- ▶ Bases de la programmation en C
 - ▶ Variables, types de données, boucles, fonctions, tableaux/matrices, tris, pointeurs, paramètres variables
- ▶ Outillage
 - ▶ Compilation, éditeur de texte, ligne de commande, Linux, redirections

Programmation Avancé

Objectifs

- ▶ Organiser les données pour pouvoir y accéder rapidement et efficacement
- ▶ Avoir une connaissance de l'utilisation et l'implémentation des structures de données
- ▶ Estimer les coûts (mémoire & temps)

Exemples de structures

- ▶ Listes contiguës, listes chaînées, piles, queues, queues de priorités, tas, arbres, arbres binaires, arbres bicolores, tables de hachage, graphes, filtres de bloom, ...

Rappel — Types de données

(Ces valeurs peuvent varier selon l'architecture et le compilateur)

Type	Min	Min form.	Max	Max formule
char	-128	-2^7	+127	$2^7 - 1$
unsigned char	0	0	+255	$2^8 - 1$
short	-32 768	-2^{15}	+32 767	$2^{15} - 1$
unsigned short	0	0	+65 535	$2^{16} - 1$
int (16 bit)	-32 768	-2^{15}	+32 767	$2^{15} - 1$
unsigned int	0	0	+65 535	$2^{16} - 1$
int (32 bit)	-2 147 483 648	-2^{31}	+2 147 483 647	$2^{31} - 1$
unsigned int	0	0	+4 294 967 295	$2^{32} - 1$
long (32 bit)	-2 147 483 648	-2^{31}	+2 147 483 647	$2^{31} - 1$
unsigned long	0	0	+4 294 967 295	$2^{32} - 1$
long (64 bit)	-9.22337×10^{18}	-2^{63}	$+9.22337 \times 10^{18}$	$2^{63} - 1$
unsig. long long	0	0	$+1.844674 \times 10^{19}$	$2^{64} - 1$
long long	-9.22337×10^{18}	-2^{63}	$+9.22337 \times 10^{18}$	$2^{63} - 1$
unsig. long long	0	0	$+1.844674 \times 10^{19}$	$2^{64} - 1$

Rappel — Taille des données

```
1  #include <stdio.h>
2
3  int main() {
4      printf("size data types\n");
5      printf("char:          %zu\n", sizeof(char));
6      printf("short:         %lu\n", sizeof(short));
7      printf("int:           %lu\n", sizeof(int));
8      printf("long int:      %lu\n", sizeof(long int));
9      printf("float:        %lu\n", sizeof(float));
10     printf("double:       %lu\n", sizeof(double));
11     printf("long double: %lu\n", sizeof(long double));
12     printf("void:         %lu\n", sizeof(void));
13
14     printf("\nsize pointers\n");
15     printf("char *:       %lu\n", sizeof(char *));
16     printf("short *:      %lu\n", sizeof(short *));
17     printf("int *:        %lu\n", sizeof(int *));
18     printf("long int *:   %lu\n", sizeof(long int *));
19     printf("float *:      %lu\n", sizeof(float *));
20     printf("double *:     %lu\n", sizeof(double *));
21     printf("long double *: %lu\n", sizeof(long double *));
22     printf("void *:       %lu\n", sizeof(void *));
23
24     return 0;
25 }
```

size_ofs.c

Rappel — Taille des données

```
1 size data types
2 char: 1
3 short: 2
4 int: 4
5 long int: 8
6 float: 4
7 double: 8
8 long double: 16
9 void: 1
10
11 size pointers
12 char *: 8
13 short *: 8
14 int *: 8
15 long int *: 8
16 float *: 8
17 double *: 8
18 long double *: 8
19 void *: 8
```

Output of size_ofs.c

Rappel — Pointeurs (source: TD Pr. Grisoni)

```
1  #include <stdio.h>
2
3  int main() {
4      int m,n,k;
5      int *p1,*p2,*p3;
6
7      m=22; n=33;
8      p1=&m; p2=&n;
9      printf("%d %d %d %d\n",*p1,*p2,m,n);
10
11     p3=p1; p1=p2; p2=p3;
12     printf("%d %d %d %d\n",*p1,*p2,m,n);
13
14     k=*p1; *p1=*p2; *p2=k;
15     printf("%d %d %d %d\n",*p1,*p2,m,n);
16
17     printf("\nPointer addresses\n");
18     printf("%p %p %p %p\n",p1,p2,&m,&n);
19     printf("%p %p %p %p\n",&p1,&p2,m,n);
20
21     return 0;
22 }
```

Rappel — Pointeurs (source: TD Pr. Grisoni)

```
1  #include <stdio.h>
2
3  int main() {
4      int m,n,k;
5      int *p1,*p2,*p3;
6
7      m=22; n=33;
8      p1=&m; p2=&n;
9      printf("%d %d %d %d\n",*p1,*p2,m,n);
10
11     p3=p1; p1=p2; p2=p3;
12     printf("%d %d %d %d\n",*p1,*p2,m,n);
13
14     k=*p1; *p1=*p2; *p2=k;
15     printf("%d %d %d %d\n",*p1,*p2,m,n);
16
17     printf("\nPointer addresses\n");
18     printf("%p %p %p %p\n",p1,p2,&m,&n);
19     printf("%p %p %p %p\n",&p1,&p2,m,n);
20
21     return 0;
22 }
```

```
1  22 33 22 33
2  33 22 22 33
3  22 33 33 22
```

```
5  Pointer addresses
```

```
6  0x7ffc1a828ce4 0x7ffc1a828ce8 0x7ffc1a828ce8 0x7ffc1a828ce4
7  0x7ffc1a828cd8 0x7ffc1a828cd0 0x21 0x16
```

Rappel — Pointeurs 2

```
1 void main() {
2     int*    x; // Allouer les pointeurs en mémoire
3     int*    y; // (mais pas les valeurs pointés)
4
5     x = malloc(sizeof(int));
6         // Allouer un entier (valeur pointé),
7         // et faites pointer x sur cette espace
8
9     *x = 42; // Donnez la valeur de 42 à l'espace pointé par x
10            // (Déreferencer x)
11
12     *y = 13; // ERREUR --- y n'a pas d'espace pointé en mémoire
13            //(SEGFAULT)
14
15     y = x; // Faites pointer y sur le même espace mémoire que x
16
17     *y = 13; // Dereferencez y et assignez 13
18            // (espace pointé par x et y)
19     free(x); // Libérer l'espace alloué
20 }
```