Programmation avancée

Introduction et Rappel

Walter Rudametkin

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

Bureau F011
Polytech'Lille

CM0

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

```c
#include <stdio.h>

int main() {
        printf("size data types\n");
        printf("char:        %zu\n",sizeof(char));
        printf("short:       %lu\n",sizeof(short));
        printf("int:         %lu\n",sizeof(int));
        printf("long int:    %lu\n",sizeof(long int));
        printf("float:       %lu\n",sizeof(float));
        printf("double:      %lu\n",sizeof(double));
        printf("long double: %lu\n",sizeof(long double));
        printf("void:        %lu\n",sizeof(void));

        printf("\nsize pointers\n");
        printf("char *:        %lu\n",sizeof(char *));
        printf("short *:       %lu\n",sizeof(short *));
        printf("int *:         %lu\n",sizeof(int *));
        printf("long int *:    %lu\n",sizeof(long int *));
        printf("float *:       %lu\n",sizeof(float *));
        printf("double *:      %lu\n",sizeof(double *));
        printf("long double *: %lu\n",sizeof(long double *));
        printf("void *:        %lu\n",sizeof(void *));

        return 0;
}
```

size_ofs.c

Rappel — Taille des données

```
size data types
char:          1
short:         2
int:           4
long int:      8
float:         4
double:        8
long double:  16
void:          1

size pointers
char *:        8
short *:       8
int *:         8
long int *:    8
float *:       8
double *:      8
long double *: 8
void *:        8
```

Output of `size_ofs.c`

Rappel — Pointeurs (source: TD Pr. Grisoni)

```c
#include <stdio.h>

int main() {
        int m,n,k;
        int *p1,*p2,*p3;

        m=22; n=33;
        p1=&m; p2=&n;
        printf("%d %d %d %d\n",*p1,*p2,m,n);

        p3=p1; p1=p2; p2=p3;
        printf("%d %d %d %d\n",*p1,*p2,m,n);

        k=*p1; *p1=*p2; *p2=k;
        printf("%d %d %d %d\n",*p1,*p2,m,n);

        printf("\nPointer addresses\n");
        printf("%p %p %p %p\n",p1,p2,&m,&n);
        printf("%p %p %p %p\n",&p1,&p2,m,n);

        return 0;
}
```

Rappel — Pointeurs (source: TD Pr. Grisoni)

```c
#include <stdio.h>

int main() {
        int m,n,k;
        int *p1,*p2,*p3;

        m=22; n=33;
        p1=&m; p2=&n;
        printf("%d %d %d %d\n",*p1,*p2,m,n);

        p3=p1; p1=p2; p2=p3;
        printf("%d %d %d %d\n",*p1,*p2,m,n);

        k=*p1; *p1=*p2; *p2=k;
        printf("%d %d %d %d\n",*p1,*p2,m,n);

        printf("\nPointer addresses\n");
        printf("%p %p %p %p\n",p1,p2,&m,&n);
        printf("%p %p %p %p\n",&p1,&p2,m,n);

        return 0;
}
```

```
22 33 22 33
33 22 22 33
22 33 33 22

Pointer addresses
0x7ffc1a828ce4 0x7ffc1a828ce8 0x7ffc1a828ce8 0x7ffc1a828ce4
0x7ffc1a828cd8 0x7ffc1a828cd0 0x21 0x16
```

Rappel — Pointeurs 2

```c
void main() {
  int*    x;  // Allouer les pointeurs en mémoire
  int*    y;  // (mais pas les valeurs pointés)

  x = malloc(sizeof(int));
      // Allouer un entier (valeur pointé),
      // et faites pointer x sur cette espace

  *x = 42; // Donnez la valeur de 42 à l'espace pointé par x
           // (Déreferencer x)

  *y = 13; // ERREUR --- y n'a pas d'espace pointé en mémoire
           //(SEGFAULT)

  y = x; // Faites pointer y sur le même espace mémoire que x

  *y = 13; // Dereferencez y et assignez 13
           // (espace pointé par x et y)
  free(x); // Liberer l'espace alloué
}
```