Algorithmique et structures de données

Loïc Demange
loic.demange@etud.univ-paris8.fr

18 septembre 2020

Préliminaires

- 10 cours de 3h, séparés en 1h30 cours - 1h30 TP (sauf premier cours)
- Deux évaluations, dont a minima une par projet

Préliminaires

- 18 septembre - Rappels algorithmique et langage C
- 25 septembre - Complexité
- 2 octobre - Tableaux et algorithmes de tri (1/2)
- 9 octobre - Tableaux et algorithmes de tri (2/2)
- 16 octobre - Programmation et projet (1/2)
- 23 octobre - Programmation et projet (2/2)
- 30 octobre - Pause
- 6 novembre - Listes chaînées
- 13 novembre - Files, piles
- 20 novembre - Arbres
- 27 novembre - Pause
- 4 décembre - Examen final

Base de l'algorithmique

- Qu'est-ce qu'un algorithme ?

Base de l'algorithmique

- Qu'est-ce qu'un algorithme ?

Un algorithme est une méthode permettant de réaliser une tâche particulière.

Exemple Se servir un verre d'eau : prendre un verre, mettre son verre en dessous du robinet, faire couler l'eau, fermer le robinet.

Base de l'algorithmique

C'est le principe d'un manuel d'utilisation ou d'une recette : suite d'instructions précises permettant d'accomplir un objectif.

Problème ?

Base de l'algorithmique

C'est le principe d'un manuel d'utilisation ou d'une recette : suite d'instructions précises permettant d'accomplir un objectif.

Problème ? ambiguïté.
Exemple "Une pincée de sel".

Solution ?

Base de l'algorithmique

C'est le principe d'un manuel d'utilisation ou d'une recette : suite d'instructions précises permettant d'accomplir un objectif.

Problème ? ambiguïté.
Exemple "Une pincée de sel".

Solution ? instructions précises.
Exemple "2g de sel".

C'est pour cela qu'on va utiliser des instructions définies et non ambiguës.

Base de l'algorithmique

- Qu'est-ce qu'une instruction ?

Base de l'algorithmique

- Qu'est-ce qu'une instruction ?

Une instruction est une action à réaliser.

Base de l'algorithmique

- Qu'est-ce qu'une variable ?

- Qu'est-ce qu'une variable ?

Une variable permet de stocker une information, de l'utiliser et de la modifier.

Remarque Il existe plusieurs types primitifs : entier, flottant, booléen, caractère, etc.

Base de l'algorithmique

- Qu'est-ce qu'une vérification conditionnelle **Si** ?

- Qu'est-ce qu'une vérification conditionnelle **Si** ?

La vérification conditionnelle **Si** permet de réaliser une suite d'instructions uniquement si une condition est remplie.

Exemple Si le verre n'est pas plein, le remplir.

Base de l'algorithmique

- Qu'est-ce qu'une boucle **Tant Que** ?

- Qu'est-ce qu'une boucle **Tant Que** ?

La boucle **Tant Que** permet de réaliser une suite d'instructions tant qu'une condition est remplie.

Exemple Tant que mon verre n'est pas plein, verser de l'eau.

Base de l'algorithmique

- Qu'est-ce qu'une boucle **Pour** ?

- Qu'est-ce qu'une boucle **Pour** ?

La boucle **Pour** permet de réaliser une suite d'instructions un nombre défini de fois.

Exemple Verser 3 fois de l'eau dans mon verre.

Base de l'algorithmique

- Qu'est-ce qu'une fonction ?

- Qu'est-ce qu'une fonction ?

Les fonctions permettent de factoriser une suite d'instructions en décomposant l'algorithme en sous-algorithmes.

Exemple Une fonction "verser_eau" qui permet d'y faire appel lorsqu'on en a besoin.

Base de l'algorithmique

L'automatisation a fait que les algorithmes ont ici pour vocation à être implémenté sur une cible (ordinateur, microcontrôleur, etc.).

Or, bien que compréhensible par les humains, ce langage n'est pas directement compréhensible par les machines.

C'est pour cela qu'il existe la compilation et l'interprétation.

Langage C

Le C est un langage compilé, qui se compile grâce à divers compilateurs (GCC, Clang, MSVC, etc.).

Ici, nous allons utiliser GCC.
Pour compiler dans un terminal :

```
gcc prog.c -o prog.o
```

Pour exécuter :

```
./prog.o
```

En C, chaque instruction se termine par un point-virgule.

Le premier code qui s'exécute est celui dans la fonction **main**.

```
int main()
{
    return 0;
}
```

Remarque La fonction **main** retourne un code erreur, de type **int**.

Langage C

En C, on déclare une variable en définissant le type et son nom.
On ne peut déclarer qu'une seule fois une variable d'un même nom dans un même bloc (accolades). Elle ne vit qu'au sein de ce bloc, de manière locale.

Par exemple,

```
int x;
```

définit une variable x de type entier.
On initialise cette variable à l'aide du signe =.

```
x = 2;
```

On peut aussi initialiser à la déclaration.

```
int x = 2;
```

Remarque Sans initialisation, la variable peut contenir n'importe quelle valeur.

On peut par la suite réutiliser la variable

```
int x = 2;
int y = x + x;
```

ou la modifier.

```
int x = 2;
x = 4;
```

La vérification conditionnelle **Si** repose sur le principe de conditions : on ne réalise l'action que si la condition est remplie.

```
if(cond)
{
    ...
}
```

Il existe aussi un **Sinon**, qui permet de réaliser une action si la condition n'est pas remplie.

```
if(cond)
{
    ...
}
else
{
    ...
}
```

cond étant une condition.
Pour écrire une condition d'égalité, on doit utiliser l'opérateur ==.

```
if(x == 2)...
```

Pour écrire une condition d'inégalité, on peut utiliser l'opérateur **!=**, <, >, <=, >=. On peut aussi chaîner les opérations ou les rendre négatives.

```
if(!cond1 || (cond2 && cond3))
{
    ...
}
```

se traduit par "Si **cond1** est faux (**!**) OU (||) si **cond2** ET (**&&**) **cond3** sont vraies, on réalise les opérations".

En C, les boucles **Tant Que** s'écrivent de cette manière.

```
while(cond)
{
    ...
}
```

La boucle **Pour** s'écrit de la manière suivante.

```
int i;
for(i = 0; i < 10; i++)
{
    ...
}
```

La première instruction est l'étape initiale, elle n'est réalisé qu'une seule fois.
La seconde est la condition Tant Que : "Tant que i est inférieur à 10, je réalise l'action".
La dernière est l'incrémentation de i. Ici, $i++ \equiv i = i + 1$.

```
int i;
for(i = 0; i < 10; i++)
{
    ...
}
```

Ce code est donc quasi-équivalent à celui-ci.

```
int i = 0;
while(i < 10)
{
    ...
    i++;
}
```

En C, les fonctions s'écrivent de cette manière, avant la fonction **main**.

```
int func(int x, int y)
{
    int z;
    ...
    return z;
}
```

x et **y** sont ici des paramètres, la fonction y a accès.
La fonction renvoie ici un entier, d'où le **return**. Il est possible de renvoyer un autre type ou de ne rien renvoyer, avec **void**.

Par la suite, on peut y faire appel dans n'importe quelle autre fonction, dont le **main**.

```
int main()
{
    int x = func(1, 2);
    ...
    return 0;
}
```

Il existe des bibliothèques, qui sont des collections de méthodes pré-compilées permettant de réaliser des opérations en un appel de fonction.

Pour cela, il faut préalablement inclure les bibliothèques qui nous intéressent dans notre programme grâce à la directive préprocesseur **#include**, et ce en tout début de programme.

Langage C

La bibliothèque **stdio.h** possède des fonctions permettant d'interagir avec les entrées/sorties de la cible.

Par exemple, **printf** permet d'écrire dans le terminal.

```
#include <stdio.h>
int main()
{
    int x = 10;
    printf("%d\n", x);
    ...
    return 0;
}
```

Ici, **%d** signifie qu'on va afficher le contenu d'une variable de type entier, qui sera précisé dans le paramètre suivant (ici **x**).
n signifie un retour à la ligne (et un vidage de la mémoire tampon).

TP

Objectifs du TP :

- Écrire du code, compiler, et exécuter le programme.
- Créer une fonction addition qui renvoie l'addition de deux entiers, et faire en sorte d'afficher le résultat.
- Faire en sorte de réaliser cette addition tant que l'entier est inférieur à 100.
- Faire en sorte de réaliser cette addition que 10 fois, et si l'entier est inférieur à 50 rajouter 75.
- [BONUS] Implémenter la suite de Fibonacci.