

# Module Algorithmie et programmation

## Introduction

---

Pierre Künzli

Adapté des cours de Paul Albuquerque, Guido Bologna et Orestis Malaspinas

# Organisation du module

---

- **Pierre Künzli**, enseignant vacataire
  - pierre.kunzli@hesge.ch, bureau A426
  - Présent à hepia les mardis après-midi/soir
- **Christophe Charpilloz**, enseignant vacataire
  - christophe.charpilloz@hesge.ch, pas de bureau à hepia
- **David Gonzalez**, assistant
  - david.dd.gonzalez@hesge.ch

- Etude de l'algorithmie et structures de données.
- Apprentissage de la programmation impérative.
- Effet de bord : apprentissage du C, apprentissage de la gestion manuelle de la mémoire.
- Voir fiches module  
<https://www.hesge.ch/hepia/bachelor/informatique-et-systemes-communication>.

- Deux semestres, deux cours par semestre.
  - Algorithmes et structures de données 1, 25%.
  - Programmation séquentielle 1, 25%.
  - Algorithmes et structures de données 2, 25%.
  - Programmation séquentielle 2, 25%.
- La note du module est la moyenne des notes des quatre cours.

- Algorithmes et structures de données :
  - Un ou deux tests théoriques par semestre.
- Programmation séquentielle :
  - ~ 3 TPs notés par semestre, peut-être un mini projet.

# Modalités

- Cours en présentiels. Streaming possible de la partie théorique en cas de besoin.
- Cours “intégrés”, différenciés au niveau de l'évaluation, Salle A534.
- Cours “Algorithmes et structures de données” :
  - Mardi soir ~ 18h30-20h (PK).
  - Cours théorique.
  - Exercices théoriques.
- Cours “Programmation séquentielle” :
  - Mardi soir ~ 20h-21h30 (PK) et jeudi soir 17h-18h30 (CC).
  - Séries d'exercices pratiques à **faire individuellement**.
  - En principe une série par semaine, sans corrigé.
  - Travaux pratiques notés.
  - Peut-être un mini projet.

(D'ici quelques semaines)

- Lundi : 12h-13h
- Mercredi : 12h-13h
- Jeudi : 12h-13h
- Vendredi : 17h-18h



# Environnement de travail

- Laptops personnels avec Linux (Ubuntu recommandé).
- Machine virtuelle ou **dual boot (recommandé)**.
- Utilisation de la ligne de commande.
- Compilateur gcc/make, gestion de version git.
- IDE recommandé : codium ou vscode.
- Cours accessible sur  
<https://githepia.hesge.ch/algoprogramsoir/algoprogram-etu-22-23>.
- Espace de cours  
<https://cyberlearn.hes-so.ch/course/view.php?id=7788> pointe sur le dépôt git.

# Petit sondage

- Quelle est votre expérience en programmation (langage, niveau) ?
- Avez-vous Linux installé sur votre machine ? Si non quel système d'exploitation utilisez-vous ?

# L'algorithmique

---

# Algorithmes, définition informelle (recette)

- des entrées (les ingrédients, le matériel utilisé) ;
- des instructions élémentaires simples (fire, flamber, etc.), dont les exécutions dans un ordre précis amènent au résultat voulu ;
- un résultat : le plat préparé.

- Existent depuis 4500 ans au moins (algorithme de division, crible d'Eratosthène).
- Le mot algorithme est dérivé du nom du mathématicien perse *Muḥammad ibn Musā al-Khwārizmī*, qui a été “latinisé” comme *Algoritmi*.

# Définition formelle

En partant d'un état initial et d'entrées (peut-être vides), une séquence finie d'instruction bien définies (ordonnées) implémentables sur un ordinateur permettant de résoudre une classe de problèmes ou effectuer un calcul.

# Formalisme de l'algorithme

L'algorithme devra être plus ou moins détaillé selon le niveau d'abstraction du langage utilisé ; autrement dit, une recette de cuisine doit être plus ou moins détaillée en fonction de l'expérience du cuisinier.

## Algorithme de vérification qu'un nombre est premier

---



# Algorithme de vérification qu'un nombre est premier

Nombre premier : nombre possédant deux diviseurs entiers distincts.

# Algorithme naïf (problème)

```
est_premier(nombre) {  
    si {  
        pour tout i, t.q.  $1 < i < \text{nombre}$  {  
            i ne divise pas nombre  
        }  
    } alors vrai  
    sinon faux  
}
```

# Algorithme naïf (problème)

```
est_premier(nombre) {  
    si {  
        pour tout i, t.q.  $1 < i < \text{nombre}$  {  
            i ne divise pas nombre  
        }  
    } alors vrai  
    sinon faux  
}
```

**Pas vraiment un algorithme : pas une séquence ordonnée et bien définie**

# Algorithme naïf (problème)

```
est_premier(nombre) {  
    si {  
        pour tout i, t.q.  $1 < i < \text{nombre}$  {  
            i ne divise pas nombre  
        }  
    } alors vrai  
    sinon faux  
}
```

**Pas vraiment un algorithme : pas une séquence ordonnée et bien définie**

**Problème : Comment écrire ça sous une forme algorithmique ?**

# Algorithme naïf (une solution)

```
est_premier(nombre) {  
  soit i := 2;  
  tant que i < nombre {  
    si nombre modulo i = 0 {  
      retourne faux  
    }  
    i := i + 1  
  }  
  retourne vrai
```

# Notions de base d'algorithmique

Pour construire un algorithme, on a besoin de différents ingrédients :

- **Variables,**
- **Opérateurs,**
- **Structures de contrôles,**
- **Boucles,**
- Fonctions.

# Les variables

- Une variable est une entité qui contient une information, elle possède :
  - un nom, on parle d'identifiant ;
  - une valeur ;
  - un type qui caractérise l'ensemble des valeurs que peut prendre la variable.
- L'ensemble des variables est stocké dans la mémoire de l'ordinateur.

## Principaux types de variable :

- Entier pour manipuler des entiers ;
- Réel pour manipuler des nombres réels ;
- Booléen pour manipuler des valeurs logiques ;
- Caractère pour manipuler des caractères alphabétiques et numériques ;
- Chaîne pour manipuler des chaînes de caractères.



# Les variables

- Une variable est l'association d'un nom avec un type, permettant de mémoriser une valeur de ce type.
- A un type donné, correspond un ensemble d'opérations définies pour ce type.

# Opérateur, opérande et expression

- **Un opérateur** est un symbole d'opération qui permet d'agir sur des variables ou de faire des “calculs”.
- **Une opérande** est une entité (variable, constante ou expression) utilisée par un opérateur.
- **Une expression** est une combinaison d'opérateur(s) et d'opérande(s), elle est évaluée durant l'exécution de l'algorithme, et possède une valeur (son interprétation) et un type.

# Opérateur, opérande et expression

Exemple dans  $a + b$  :

- $a$  est l'opérande gauche ;
- $+$  est l'opérateur ;
- $b$  est l'opérande droite ;
- $a + b$  est appelé une expression.

Si par exemple  $a$  vaut 2 et  $b$  3, l'expression  $a + b$  vaut 5.

Si par exemple  $a$  et  $b$  sont des entiers, l'expression  $a+b$  est un entier.

# Les opérateurs

- Un opérateur peut être unaire ou binaire :
  - Unaire s'il n'admet qu'une seule opérande, par exemple l'opérateur non.
  - Binaire s'il admet deux opérandes, par exemple l'opérateur +.
- Un opérateur est associé à un type de donnée et ne peut être utilisé qu'avec des variables, des constantes, ou des expressions de ce type.

# Manipulation de variables

On ne peut faire que deux choses avec une variable :

- Obtenir son contenu. Cela s'effectue simplement en nommant la variable.
- Affecter un (nouveau) contenu. Cela s'effectue en utilisant l'opérateur d'affectation, représenté par  $\leftarrow$ ,  $=$  ou  $:=$  suivant les formalismes.

# Manipulation de variable

Par exemple l'expression  $c = a + b$  se comprend de la façon suivante :

- On prend la valeur contenue dans la variable  $a$  ;
- On prend la valeur contenue dans la variable  $b$  ;
- On additionne ces deux valeurs ;
- On met ce résultat dans la variable  $c$ .

Si  $c$  avait eu auparavant une valeur, cette dernière aurait été perdue !

Un algorithme peut avoir des interactions avec l'utilisateur :

- Il peut afficher un résultat (le contenu d'une variable, du texte, une image, ...),
- Il peut attendre une entrée de l'utilisateur (afin de stocker l'information dans une variable ou changer son comportement en fonction de l'entrée).

# De l'algorithme au programme

Un algorithme est une représentation abstraite d'une suite d'instructions appliquant des transformations à des données. Un programme est la réalisation concrète d'un algorithme pouvant être exécuté sur un ordinateur. On parle d'implémentation.

Ce que dit l'académie français du terme implémenter :

<https://www.academie-francaise.fr/implémenter>.



# Le langage C

Pour implémenter nos programmes, on utilisera le langage C.

**Par exemple, un algorithme permettant de diviser une somme d'argent en billets et pieces**

```
somme = lire()  
billets100 = somme div 100  
reste = somme mod 100  
billets50 = reste div 50;  
reste = reste mod 50  
billets10 = reste div 10  
reste = reste mod 10  
pieces2 = reste div 2  
reste = reste mod 2  
pieces1 = reste  
ecrire(billets100, billets50, billets10, pieces2, pieces1)
```

# Le langage C

## Et son implémentation en C

```
#include <stdio.h>

void main(){
    int somme, reste, billets100, billets50, billets10;
    int pieces2, pieces1; // variables de type int
    somme = 2143; // affectation
    billets100 = somme / 100; // division entiere (arrondi vers 0)
    reste = somme % 100; // reste de la division entiere
    billets50 = reste / 50;
    reste = reste % 50;
    billets10 = reste / 10;
    reste = reste % 10;
    pieces2 = reste / 2;
    reste = reste % 2;
    pieces1 = reste;
    // affichage
    printf("%i billets de 100, %i billets de 50, %i billets de 10, %i pie
           billets100, billets50, billets10, pieces2, pieces1);
}
```

# Génération d'un exécutable

- Pour pouvoir être exécuté un code C doit être d'abord compilé (avec gcc ou clang).
- Pour un code prog.c la compilation "minimale" est

```
$ gcc prog.c  
$ ./a.out # exécutable par défaut
```

- Il existe une multitude d'options de compilation :

```
$ gcc -O1 -std=c11 -Wall -Wextra -g prog.c -o prog  
-fsanitize=address -fsanitize=leak -fsanitize=undefined
```

1. -std=c11 utilisation de C11.
2. -Wall et -Wextra activation des warnings.
3. -fsanitize=... contrôles d'erreurs à l'exécution (coût en performance).
4. -g symboles de débogages sont gardés.
5. -o définit le fichier exécutable à produire en sortie.
6. -O1, -O2, -O3 : activation de divers degrés d'optimisation

## Demo

---