Chapitre 4

L'algorithmie

Description et exemple de l'utilisation du langage algorithmique.

1. Introduction

L'algorithmie est utilisée pour décrire des **traitements** (programmes, sous-programmes, méthodes) dont la complexité nécessite une étape intermédiaire avant de passer à la phase de codage.

L'algorithme est donc un langage de très haut niveau puisqu'il n'est pas interprétable par un ordinateur. Ainsi sa syntaxe est à "laxiste" mais aussi précise pour lever certaines ambigüités.

La meilleure façon pour vérifier la validité d'un algorithme est, **de le faire lire** par un tiers. Par définition, il n'existe pas de moyens informatiques qui vérifient la validité fonctionnelle d'un algorithme.
La meilleure validation est bien sur son implémentation en un programme exécutable mais c'est ce que l'on veut éviter.

Un algorithme s'écrit en plusieurs niveaux dits **de profondeur** (niveau de détail).
On commence toujours par écrire un premier niveau assez général, loin des instructions du langage de programmation cible.

Ce premier niveau correspond à un découpage du problème en sous-problèmes.
Ensuite on décompose chacun des sous-problèmes sous la forme de nouveaux algorithmes.
Et ceci jusqu'à un certain niveau. Mais lequel ?

Réponse : le niveau qui permet de passer à la phase de programmation.

Ainsi cela dépend du niveau de connaissance et de l'expérience du programmeur qui va être capable de traduire l'algorithme en code, et aussi du niveau de liberté qu'on lui donne pour faire les choix de l'implémentation de l'algorithme.

Tout traitement se faisant sur des données, il faut préciser les **données en entrée** de l'algorithme et les **données en sortie** de l'algorithme. On peut aussi préciser les **données locales** utilisées par l'algorithme.

Tous les principes énoncés dans ce chapitre sont applicables à la programmation en générale.

1.1. Exemple : Faire un mille feuille (recette de cuisine)

Faire un mille-feuille

Données en entrée :
3 pâtes feuilletées achetées toutes faites
3 sachets de sucre vanillé
1 verre d'eau froide
1 four électrique standard
3 plaques de cuisson au four
1 pinceau de cuisine
1 sachets de sucre vanillé
6 œufs
200g de sucre

```
  100g de farine
  2 bâtonnets de vanille
  1 litre de lait demi-écrémé
  400g de sucre glace
  ½ jus de citron
  1 sachet de pépites de noisette caramélisées
  1 casserole de 1,5l
  2 spatules en bois
  1 batteur électrique
  1 saladier de 1,5 à 2 litres
  1 saladier de 0,5 litre
  2 cl de rhum
  2 cs d'arome vanille liquide
  1 frigidaire
  1 plat à gâteau

  Debut
     Faire cuire les 3 pates feuilletées;
     Les mettre de côté et les laisser refroidir;
     Faire 1 litre de crème pâtissière à la vanille;
     Mettre de côté en mettant dessus du papier de cuisson pour empêcher la
formation d'une peau;
     Faire le fondant et le garder au frigidaire;
     Une fois la crème froide, faire l'assemblage;
  Fin
  Données en sortie :
  Un gâteau mille-feuilles prêt à consommer.
```

Est-ce que ce niveau est suffisant pour que vous soyez capable de faire la recette ?

Non. Donc on va préciser les sous-problèmes (ou étapes ici) qui le nécessitent.

```
  Faire cuire les 3 pates feuilletées

  Données en entrée :
  3 pâtes feuilletées achetées toutes faites
  3 sachets de sucre vanillé
  1 verre d'eau froide
  1 four électrique standard
  3 plaques de cuisson au four
  1 pinceau de cuisine

  Debut
     Pour chaque pâte feuilletée faire
        Mettre la pâte sur une plaque avec son papier de cuisson;
        Piquer la pâte avec une fourchette; // Pour empêcher la pâte de
gonfler à la cuisson
        Badigeonner au pinceau la pâte avec de l'eau froide;
        Soupoudrer toute la surface de la pâte avec le contenu d'1 sachet
             de sucre vanillé;
     Finpour
     Mettre le mode de cuisson traditionnel avec chaleur tournante;
     Préchauffer le four à 160 degré;
     Mettre les 3 pâtes à cuire pendant 40 mn (surveiller la cuisson);
  Fin
```

```
Données en sortie :
3 pâtes feuilletées cuites
```

Faire 1 litre de crème pâtissière à la vanille

```
Données en entrée :
6 œufs
200g de sucre
100g de farine
2 bâtonnets de vanille
1 litre de lait demi-écrémé
1 casserole de 1,5l
2 spatules en bois
1 batteur électrique
1 spatule
1 saladier de 1,5 à 2 litres
2 cl de rhum
2 cs d'arome vanille liquide

Debut
   Faire bouillir le lait avec les gousses de vanilles fendues et
      une pincée de sel dans la casserole;
   Mettre 5 jaunes d'œufs dans le saladier et garder 2 blancs d'œuf
      de côté qui seront utilisés pour le fondant;
   Ajouter le sucre dans le saladier;
   Tantque le mélange ne blanchit pas faire
      battre au fouet électrique le mélange;
   FinTantque
   Ajouter 100 g de farine tamisée tout en fouettant le mélange;
   Puis ajouter l'oeuf entier et fouetter;
   Enlever les gousses de vanille;
   Verser le lait bouillant petit à petit sur le mélange tout
      en mélangeant;
   Remettre le mélange dans la casserole:
   Faire cuire à feux moyen tout en remuant avec une spatule en bois
      et en raclant bien le fond de la casserole jusqu'aux premiers
      gros bouillons;
   Incorporer le rhum à la préparation;
Fin

Données en sortie :
1 litre de crème pâtissière;
2 blancs d'œuf
```

Faire le fondant

```
Données en entrée :
400g de sucre glace
½ jus de citron
1 saladier de 0,5 litre
2 blancs d'œuf
1 spatule
1 frigidaire

Debut
   Mettre le sucre glace dans le saladier;
   Ajouter les blancs d'œuf;
```

```
   Mélanger à la spatule jusqu'à ce obtenir une pâte bien épaisse;
   Laisser reposer 15mn au frigidaire;
   Ajouter le jus de citron et bien mélanger;
   Mettre au frigidaire;
Fin

Données en sortie :
Un fondant
```

```
Faire l'assemblage

Données en entrée :
3 pâtes feuilletées
1 litre de crème pâtissière
un fondant
1 plat à gâteau

Debut
   Dans le plat mettre une première pâte feuilletée avec la partie
      plate en bas;
   Etaler dessus 45% de la crème pâtissière;
   Mettre une deuxième pâte feuilletée par-dessus avec la partie
      plate en bas;
   Etaler dessus 45% de la crème pâtissière;
   Mettre la troisième pâte feuilletée par-dessus avec la partie
      plate en haut;
   Etaler le fondant dessus;
   Mettre les 10% de crème pâtissière restante tout autour en
      bouchant les trous;
Fin

Données en sortie :
Un mille-feuille
```

2. Le corps de l'algorithme

```
Données en entrée :

Début

Fin
Données en sortie :
```

Définir des données en entrée et en sortie n'est pas obligatoire dans la mesure où ces données sont clairement identifier dans l'algorithme lui-même en utilisant un vocabulaire précis et homogène.

Souvent, il n'existe pas de données en entrée/sortie dans le cas où ces données sont lues et écrites sur des périphériques (ihm, saisie, fichier, base de donnée, …) et sont donc identifiées comme des données locales à l'algorithme.

Le corps d'un algorithme contient toujours une SEQUENCE

2.1. SEQUENCE : La séquence d'instruction

La séquence d'instruction est un bloc d'instruction dans lequel toutes les instructions s'exécutent séquentiellement de haut en bas.

Une **instruction** est une phrase d'algorithme qui correspond à un traitement. Chaque instruction se termine par un point-virgule (;).

Syntaxe :

Debut
Instruction 1;
Instruction 2;
…
Instruction N;
Fin

Exemple :

```
Debut
 Initialiser la bibliothèque;
 Afficher le menu;
 Choisir un des items du menu;
 En fonction de l'item choisi faire  l'action de l'item de menu;
Fin
```

Exemple :

```
Debut
 Créer la bibliothèque;
 Construire les éléments IHM en fonction de la bibliothèque;
 Afficher l'IHM;
Fin
```

2.2. CONDITION : Le bloc conditionnel

Le bloc conditionnel permet de réaliser un test et si le test est VRAI alors on exécute des instructions et si le test est FAUX alors on exécute d'autres instructions.

Syntaxe :

Si <*Test*> alors
<Instruction 1>;
<Instruction 2>;
…
<Instruction N>;
Sinon
<Instruction' 1>;
<Instruction' 2>;
…
<Instruction' N>;
Finsi

Si <*Test*> alors
<Instruction 1>;
<Instruction 2>;
…
<Instruction N>;
Finsi

2.3. CAS DE : Le choix multiple

Le choix multiple consiste en fonction d'une valeur d'exécuter des instructions différentes. Cela évite de faire des "si alors sinon" imbriqués.

Syntaxe :

Cas <variable> de
    valeur 1 :
        <Instructions>;
    valeur 2 :
        <Instructions>;
    …
    defaut :
        <Instructions>;
Fincas

Exemple :

```
//
j : ENTIER;
jour : CHAINE;
jabrev : CHAINE;
…

Cas j de
 1 :  jour = "LUNDI";
      jabrev = "L";
 2 :  jour = "MARDI";
      jabrev = "Ma";
 3 :  jour = "MERCREDI";
      jabrev = "Me";
 4 :  jour = "JEUDI";
      jabrev = "J";
 5 :  jour = "VENDREDI";
      jabrev = "V";
 6 :  jour = "SAMEDI";
      jabrev = "S";
 7 :  jour = "DIMANCHE";
      jabrev = "D";
Fincas
```

2.4. TANT QUE : La boucle conditionnelle

Le "tantque" permet de faire des instructions tant que le test est VRAI.

Syntaxe:

Tantque <*Test est vrai*> faire
    <Instruction 1>;
    <Instruction 2>;
    …
    <Instruction N>;
Fintantque

On trouve aussi la syntaxe suivante :

Tantque <*Test est vrai*> faire
Debut
    <Instruction 1>;
    <Instruction 2>;
    …
    <Instruction N>;

Fin

Exemple :

```
// Rechercher un élément dans un tableau
Debut
 i=0;
 fini = FAUX;
 Tantque non fini faire
   Si i>MAX alors
      fini = VRAI;
      trouvé = FAUX;
   Sinon
      Si x = tab[i] alors
        fini = VRAI;
        trouvé = VRAI;
      Finsi
      i = i + 1;
   Finsi
 Fintantque
Fin
```

2.5. FAIRE TANTQUE ou FAIRE JUSQU'A

Faire
<Instruction 1>;
…
<Instruction N>;
Tantque <Test est vrai>

Faire
<Instruction 1>;
…
<Instruction N>;
Jusqu'à <Test est vrai>

Exemples :

```
Faire
   Saisir votre mot de passe;
   Si le mot de passe n'est pas correct Alors
      Afficher un avertissement que le mot de passe est erroné;
   Finsi
Jusqu'à le mot de passe est correct;
```

ou

```
Faire
   Saisir votre mot de passe;
   Si le mot de passe n'est pas correct Alors
      Afficher un avertissement que le mot de passe est erroné;
   Finsi
Tantque le mot de passe est incorrect;
```

2.6. POUR : La boucle itérative

Deux variantes de la boucle "pour" existent.

Les boucles "Pour" permettent de réaliser une itération dont les bornes sont connues à l'avance et d'exécuter des instructions sur chaque itération.

1/ La boucle d'itération sur un intervalle :

Syntaxe :

Pour <i> = <min> à <max> faire
<Instruction 1>;
<Instruction 2>;
…
<Instruction N>;
Finpour

Exemple :

```
tab : TABLEAU DE REEL[MAX];
trouvé : booléen;
indice : ENTIER;
x : REEL;
Debut
 Saisir x;
 Saisir tab;
 trouvé = FAUX;
 Pour indice = 0 à MAX -1 faire
   Si x = tab[indice] alors
      trouvé = VRAI;
   Finsi
 Finpour
Fin
```

2/ La boucle d'itération sur une collection ou sur un tableau

Syntaxe :

Pour chaque <e> de <v> faire
<Instruction 1> en utilisant ou non <e>;
<Instruction 2> en utilisant ou non <e>;
…
<Instruction N> en utilisant ou non <e>;
Finpour

Exemple :

```
Biblio : Bibliothèque;
//Affiche tous les livres empruntés de la bibliothèque
Debut
 Pour chaque livre de Biblio.livres faire
   Si livre.emprunté alors
      Afficher le livre (livre.référence, livre.titre);
   Finsi
 Finpour
Fin
```

Exemple :

```
Tab : TABLEAU DE REEL[MAX];
```

```
Debut
 //Affiche tous les éléments d'un tableau
 Pour chaque e de Tab faire
   Afficher(e);
 Finpour
Fin
```

Il existe une instruction prédéfinie de la boucle "Pour" permettant de sortir de la boucle sans attendre la fin de son exécution : **sortir boucle**

Exemple :

```
//Rechercher un élément et s'arrêter dès qu'on l'a trouvé.

tab : TABLEAU DE Individu[MAX];
x : Individu;
trouvé : booléen;

Debut
 Saisir x.nom;
 Saisir x.prenom;
 trouvé = FAUX;
 Pour chaque e de tab faire
   Si e.nom = x.nom ET e.prénom = x.prénom alors
      trouvé = VRAI
      sortir boucle;
   Finsi
Finpour
 Si trouvé alors afficher "j'ai trouvé"
 Sinon afficher "je n'ai pas trouvé";
Fin
```

2.7. Les sous programmes

Pour factoriser des instructions utilisées à plusieurs endroits d'un algorithme, on écrit des sous-programmes (ou sous-algorithme) qui peut avoir des paramètres d'entrée et des paramètres de sortie.

2.7.1. La procédure

La syntaxe de déclaration d'une procédure est :

PROCEDURE <nom>(<paramètres>)
Debut

<Instruction 1>;
<Instruction 2>;
…
<Instruction N>;

Fin

Syntaxe d'appel :
<nom>(<paramètres>);

Exemple :

```
Biblio : Bibliothèque;
//Affiche tous les livres empruntés de la bibliothèque
Debut
```

```
nb=0;
Pour chaque livre de Biblio.Livres faire
  Si livre.emprunté alors
     Afficher le livre (livre.Référence, livre.Titre);
  Sinon
     t[nb] = livre;
     nb = nb +1;
  Finsi
Finpour
Afficher "Les livres disponibles sont:";
Pour chaque e de t faire
  Afficher le livre (e.Référence, e.Titre);
Finpour
Fin

PROCEDURE Afficher le livre (Ref:Référence; titre : CHAINE)
Debut
 Afficher Ref.id + Ref.année + ":" + titre
Fin
```

2.7.2. La fonction

La syntaxe d'une fonction est :

FONCTION <type> <nom>(<paramètres>)
Debut
<Instruction 1>;
<Instruction 2>;
...
<Instruction N>;
retourner <valeur>;
Fin

<type> est le type de la valeur retournée par la fonction.
<valeur> est la valeur retournée par la fonction.

Syntaxe d'appel :
x = **<nom>(<paramètres>)**;

Exemple :

```
Debut
 MaBiblio : bibliothèque;

 // Saisir le titre livre à rechercher dans la bibliothèque
 Saisir un titre;
 Si rechercherLivre(titre,MaBiblio) alors
   Emprunter le livre trouvé;
 Sinon
   // Saisir un autre titre
   Afficher "Veuillez saisir un autre titre ce livre est déjà emprunté";
   Si rechercherLivre(titre,MaBiblio) alors
      Emprunter le livre trouvé;
   Sinon
      Afficher "Pas trouvé";
   Finsi
 Finsi
Fin
```

```
FONCTION booléen rechercherLivre(titre : CHAINE; biblio:Bibliothèque)
Debut
 trouve : booléen;
 trouvé = FAUX;
 Pour chaque e de biblio.Livres faire
   Si e.titre = titre alors
      trouvé = VRAI;
      sortir boucle;
   Finsi
 Finpour
 retourner trouvé;
Fin
```

2.8. Un exemple

Un exemple qui correspond aux critères suivant :

- Le besoin est commun, connu de tous
- Suffisamment complexe (mais pas trop)
- Adapté à la programmation

Développement en séance.

3. Les structures de données

Dans un algorithme (comme cela est aussi le cas dans les langages évolués, on utilise des données.

Il est nécessaire de structurer ces données. On a les familles de données suivantes :

- les données primitives
- les structures
- les objets
- les tableaux
- la liste
- les fichiers (non développé ici)
- les bases de données (non développé ici).

Pour plus de précision dans l'écriture des algorithmes, on **déclare** les données qui consistent à préciser :

- le **nom** de la donnée
- le **type** de donnée

Exemple :

```
x : double;
compteur : entier (ou integer)
tab1 : tableau d'entier (ou array of integer);
tab2 : tableau de Livre (ou array of Livre);
mot : chaine de caractère (ou string)
agenda : fichier texte;
```

```
photo : fichier binaire ou fichier jpeg
mesContacts : liste de Contact
```

3.1. Les données primitives

Les données primitives représentent des données élémentaires non structurées directement issues des tous premiers langages de programmation : entier, double, chaîne, booléen, caractère.

Naturellement, on peut utiliser, sur ces données, des opérateurs qui permettent de construire des expressions arithmétiques, booléennes, relationnelles, …

3.2. Les structures

Les structures sont des regroupements de données primitives afin de définir un type de données fonctionnel.

Ces structures sont récursives.

La notation utilisée est la suivante :

```
Type <nom> struct
   <les champs de la structure>
Finstruct
```

Exemples :

```
Type Individu struct
   nom : chaine:
   prenom : chaine;
   age : entier;
Finstruct
```

On déclare une donnée de type Individu ainsi:

```
ind1 : Individu;
```

On accède aux champs de la structure avec la notation point (".") :

```
s1 : chaine;
s1 = ind1.nom;

a1 : entier;
a1 = ind1.age;
```

3.3. Les objets

Les objets sont équivalents à une structure mais comme on le verra plus loin un objet ne contient pas que des "champs".

Ces "champs" sont appelés des attributs.

La notation utilisée est la suivante :

```
class <nom du type de l'objet>
{
   <les attributs de la classe>
}
```

Exemples :

```
class Individu
{
   nom : chaine:
   prenom : chaine;
   age : entier;
}
```

On déclare **un objet de type** Individu ainsi:

```
ind1 : Individu;
```

On accède aux **attributs** de l'objet avec la notation point (".") :

```
s1 : chaine;
s1 = ind1.nom;

a1 : entire;
a1 = ind1.age;
```

La grosse différence avec la structure est qu'on peut déclarer des traitements dans une classe que l'on appelle des **méthodes**.
Ces traitements sont décrits sous forme d'un algorithme.

On appelle ces méthodes en utilisation la notation point.

Exemple de déclaration :

```
class Individu
{
   nom : chaine:
   prenom : chaine;
   age : entier;

   afficher() // Afficher un individu
   Debut
      s : chaine;
      s = nom + " " + prenom + " " + age;
      afficher s dans le terminal;
   Fin

   siMajeur() // Retourne vrai si l'individu est majeur
   Debut
      si age > 18 alors retourne vrai
      sinon retourne faux;
   Fin
}
```

Utilisation de cette classe :

```
ind1 : Individu;
ind1.nom = "LAFONT";
ind1.prenom = "Pierre";
ind1.age = 25;
```

```
  ind1.afficher();
```

```
  // Lecture d'un fichier qui contient des individus et pour lesquels on
veut afficher que les individus majeurs

  Pour chaque individu du fichier faire
     Si individu.majeur() alors individu.afficher();
  Finpour
```

3.4. Les tableaux

Un tableau est une structure qui permet de **ranger** des données de même type, les uns à côté des autres, et qui permet d'accéder aux éléments via un **indice**.

Un tableau est défini par le type de ses éléments et sa taille.

L'indice d'un tableau commence à 0.

Comme nous le verrons plus loin dans le cours, les tableaux sont à comparer avec les **collections**. Les premiers étant de taille fixe. Les deuxièmes étant de taille "infinie".

Syntaxe :

```
  tab[i]
```

Exemple de déclaration :

```
  tab : Entier[200];
  tab : Double[100];

  tabInd : Individu[300];

  Individu ind;
  ind = tabInd[10];
```

Exemple d'algorithme sur les tableaux : extraire d'un tableau de 100 entiers max contenant 30 entiers toutes les valeurs qui sont positives :

```
  Données en entrée :
  tab : Entier[100];  // les éléments
  nbtab : Entier;  // nombre d'éléments utiles
  Debut
     resultat : Entier[30];
     nb : Entier;
     nb=0;
     Pour i=0 à nbtab - 1 faire
        Si tab[i]>0 Alors
          resultat[nb] = tab[i];
          nb=nb+1;
        Finsi
  Fin
  Données en sortie :
  resultat : Entier[30];
  nb : Entier;  // Nbre d'élément trouvés
```

3.5. Les collections

Les collections peuvent être manipulées dans les algorithmes. On les préfère au tableau car elles sont d'un concept plus élevé.

On dit que l'on **ajoute** un élément, que l'on **parcourt** les éléments, que l'on **supprime** un des éléments.

Exemples :

```
liste1 : liste de Contact;
```

```
ajouter la saisie d'un Contact dans liste1;
```

ou

```
liste1.ajouter(saisir un Contact)
```

```
Pour chaque c de liste1 Faire
   Si c est un contact professionnel alors
      afficher c;
   Finsi
Finpour
```

ou

```
Pour chaque contact de la liste liste1 Faire
   si le contact est un contact professionnel alors l'afficher;
Finpour


Pour i = 0 à nbre d'élément de liste1 faire
   Si le i-ième élement de liste 1 est un contact professionnel alors
       afficher c;
  Finsi
Finpour
```

4. Les principes algorithmiques

4.1. Le raffinage algorithmique

Raffiner un algorithme consiste à gagner en précision tout en réécrivant l'algorithme plusieurs fois.

Exemple :

1er raffinage :

```
// On veut faire la gestion d'une bibliothèque
Debut
 Initialiser la bibliothèque;
 Afficher le menu;
 Choisir un des items du menu;
 En fonction de l'item choisi faire l'action de l'item de menu;
Fin
```

2ème raffinage :

```
// On veut faire la gestion d'une bibliothèque

Debut
 MaBiblio : Bibliothèque;

 //Initialiser la bibliothèque;
 Charger la bibliothèque avec le fichier "BIBLIO";
 Si le fichier n'existe pas alors sortir du programme;
```

```
   Finsi

   Afficher le menu;
   Choisir un des items du menu;
   En fonction de l'item choisi faire l'action de l'item de menu;
  Fin
```

Vous pouvez remarquer que l'on rappel en commentaire la partie d'algorithme qui a été raffiné.

3ème raffinage :

```
  // On veut faire la gestion d'une bibliothèque

  Debut
   MaBiblio : Bibliothèque;

   //Initialiser la bibliothèque;
   Charger la bibliothèque avec le fichier "BIBLIO";
   Si le fichier n'existe pas alors sortir du programme;
   Finsi

   //Afficher le menu;
   Afficher "1 -> Emprunter un livre"
   Afficher "2 -> Rendre un livre"
   Afficher "3 -> Afficher les livres non empruntés"
   Afficher "4 -> sauver la bibliothèque et sortir du programme"

   Choisir un des items du menu;
   En fonction de l'item choisi faire l'action de l'item de menu;
  Fin
```

4ème raffinage :

```
  // On veut faire la gestion d'une bibliothèque

  Debut
   MaBiblio : Bibliothèque;

   //Initialiser la bibliothèque;
   Charger la bibliothèque avec le fichier "BIBLIO";
   Si le fichier n'existe pas alors sortir du programme;
   Finsi

   //Afficher le menu;
   Afficher "1 -> Emprunter un livre"
   Afficher "2 -> Rendre un livre"
   Afficher "3 -> Afficher les livres non empruntés"
   Afficher "4 -> Sauver la bibliothèque et sortir du programme"

   Choisir un des items du menu;
   //En fonction de l'item choisi faire l'action de l'item de menu;
   Cas choix de
     1 : Emprunter un livre;
     2 : Rendre un livre;
     3 : Afficher les livres non empruntés;
     4 : Debut
           Sauver MaBiblio;
           Quitter le programme;
         Fin
   Fincas
  Fin
```

4.2. La décomposition algorithmique

La décomposition consiste à isoler une instruction d'un algorithme et écrire l'algorithme de cette instruction à part de l'algorithme initial.

```
// On veut faire la gestion d'une bibliothèque

Debut
 MaBiblio : Bibliothèque;

 //Initialiser la bibliothèque;
 Charger la bibliothèque avec le fichier "BIBLIO";
 Si le fichier n'existe pas alors sortir du programme;
 Finsi

 //Afficher le menu;
 Afficher "1 -> Emprunter un livre"
 Afficher "2 -> Rendre un livre"
 Afficher "3 -> Afficher les livres non empruntés"
 Afficher "4 -> Sauver la bibliothèque et sortir du programme"

 Choisir un des items du menu;
 //En fonction de l'item choisi faire l'action de l'item de menu;
 Cas choix de
   1 : Emprunter un livre;
   2 : Rendre un livre;
   3 : Afficher les livres non empruntés;
   4 : Debut
          Sauver MaBiblio;
          Quitter le programme;
       Fin
 Fincas
Fin

PROCEDURE Rendre un livre
Debut
 Saisir la référence du livre à rendre;
 Rechercher ce livre grâce à la référence;
 Si trouvé alors
   Marque le livre trouvé comme non emprunté;
 Sinon
   Afficher "Ce livre n'existe pas";
 Finsi
Fin

PROCEDURE Afficher les livres non empruntés
Debut
 Pour chaque livre de la bibliothèque faire
   Si le livre n'est pas emprunté alors
      Afficher la référence du livre, son titre, son auteur
   Finsi
Fin

PROCEDURE Emprunter un livre
Debut
 Saisir la référence du livre à emprunter;
 Rechercher ce livre grâce à la référence;
 Si trouvé alors
   Si le livre est déjà emprunté alors
```

```
      Afficher que le livre est déjà emprunté;
   Sinon
      Marquer ce livre comme étant emprunté;
   Finsi
 Sinon
   Afficher "Ce livre n'existe pas";
 Finsi
Fin
```

5. Exercices

5.1. Exercice 1

Ecrire l'algorithme qui teste si une chaîne de caractère est un palindrome.
La chaine est définie par le type de donnée :

```
Type CHAINE struct
   Nb : Entier;
   Tab : TABLEAU DE CARACTERE[MAX_CHAINE];
Finstruct
```

La méthode de saisie est :

```
CHAINE Saisir();
```

5.2. Exercice 2

Ecrire l'algorithme qui recherche une sous chaîne de caractère dans une plus grande chaîne. Les deux chaînes sont saisies.
Le programme affiche si la chaîne a été trouvée.

5.3. Exercice 3

Ecrire l'algorithme qui compte le nombre d'occurrence de chacun des mots contenu dans une phrase. Chaque mot est espacé par des caractères blancs.
Ecrire un premier algorithme général de premier niveau.
Ensuite raffiner et décomposer ce premier algorithme en utilisant le type COLLECTION pour stocker les occurrences de chacun des mots.

5.4. Exercice 4

Ecrire l'algorithme du programme qui permet de jouer au **jeu du pendu**.
Les mots proposés sont tirés aléatoirement d'un dictionnaire (fichier).
Les mots contenus dans le dictionnaire sont entre 4 et 15 lettres.
Chaque mot est caractérisé par une catégorie donnée (pays, métier, sport, partie du corps, …)
Pour jouer il faut choisir la longueur du mot à trouver (de 4 à 15) ou une catégorie.
Le joueur gagne s'il trouve en au plus 7 coups.
Le jeu affiche les lettres jouées.
http://www.pendu.learningtogether.net

5.5. Exercice 5

Ecrire l'algorithme du programme qui permet de gérer un agenda : afficher les rendez-vous d'une journée, ajouter un rendez-vous, modifier un rendez-vous, supprimer un rendez-vous..

5.6. Exercice 6

Ecrire l'algorithme du programme qui trie par ordre croissant un tableau d'entier.

Le nombre d'entier et leurs valeurs sont saisies à l'écran.

6. Corrections des exercices

6.1. Exercice 1

```
Solution 1 :

Debut
 Saisir la chaine à tester;
 ch2 = inverser la chaine ;
 Si la chaine égale à ch2 alors c'est un palindrome
 Sinon ce n'est pas un palindrome
Fin
```

Cette solution montre que la meilleure façon de résoudre un problème est de le décomposer en sous-problèmes moins complexes :
- inverser une chaîne
- comparer deux chaînes.

Par contre cela est souvent au détriment d'une certaine performance. L'algorithme se déroule avec deux boucles de parcours de chaîne.

La solution suivante est plus "humaine", plus difficile à écrire mais plus performante avec une seule boucle de parcours de chaîne.

```
Solution 2 :

// 1er raffinage
//
Debut
 Saisir la chaine à tester;
 Vérifier si la chaine est en miroir;
 Si cela est le cas alors
   c'est un palindrome
 Sinon
   ce n'est pas un palindrome
 Finsi
Fin
```

```
// 2ème raffinage
//
Debut
 Saisir la chaine à tester;

 //Vérifier si la chaine est en miroir;
 Pour chaque caractère de i = 0 à n/2 - 1 Faire
   Si le caractère est égale à celui se trouvant en n-1-i Alors
      on continue la boucle
   Sinon
      on arrête la boucle;
 Finpour

 Si on n'a pas arrêté la boucle alors
   c'est un palindrome
 Sinon
   ce n'est pas un palindrome
```

```
 Finsi
Fin
```

```
// 3ème raffinage
//
TYPE CHAINE EST STRUCTURE
 Nb : Entier;
 Tab : TABLEAU DE CARACTERE[MAX_CHAINE];
FIN STRUCTURE
Palindrome : Booléen;
i : entier;
Ch : Chaine;
Debut
 Ch = Saisir();
 Palindrome = vrai;
 Pour i = 0 à (Ch.Nb/2)-1 faire
   Si Ch[i] = Ch[Ch.Nb-1-i] alors
      // OK
   Sinon
      Palindrome = faux;
      sortir de la boucle;
   Finsi
 Finpour
 Si Palindrome alors
   Afficher "C'est un palindrome"
 Sinon
   Afficher "Ce n'est pas un palindrome";
 Finsi
Fin
```

6.2. Exercice 2

```
Debut
 Saisir la chaine;
 Saisir la sous-chaine;
 i = 0;
 lg = longueur de la sous-chaine;
 fini = faux;
 Tantque non fini faire
   Si i+lg > longueur de la chaine alors
      fini = vrai;
      trouvé = faux;
   Sinon
      Si la sous-chaine = chaine de i à i+lg-1 alors
         trouvé = vrai;
         fini = vrai;
      Sinon
          i = i +1;
   Finsi
 Fintantque
 Si trouvé alors
   Afficher "la sous-chaine a été trouvé en i"
 Sinon
   Afficher "pas trouvé"
 Finsi
Fin
```

6.3. Exercice 3

```
  Debut
   Saisir la phrase;
   Initialiser à vide la collection C;
   // La collection C est constituée de couple (Chaine,Entier).
   //   Ce couple contient un mot et son nombre d'occurrence
   //
   Transformer la phrase en une collection de mots: liste_mots;
   Pour chacun des mots m de liste_mots faire
     Si m est dans un des couples de C alors
        Incrémenter le nbre du couple trouvé;
     Sinon
        Ajouter le couple (m,1) dans C;
     Finsi
   Finpour
   Pour chaque couple de C faire
     Afficher le couple;
   Finpour
  Fin

  //Transformer une phrase en un tableau de mots
  //
  PROCEDURE COLLECTION de CHAINE SPLIT(Phrase:CHAINE)
  Debut
   liste_mots : COLLECTION de CHAINE;
   mot_courant : CHAINE;
   mot_courant = "";
   liste_mots = vide;
   Pour chaque caractère c de la phrase faire
     Si c = blanc et mot_courant ="" alors
        rien
     Sinon
        Si c = blanc alors
          ajouter mot_courant à liste_mots;
          mot_courant = "";
        Sinon
          mot_courant = mot_courant + c;
        Finsi
     Finsi
   Finpour
   Retourner liste_mots;
  Fin
```

6.4. Exercice 4

```
  // Jeu du pendu
  //
  Debut
   Collection Dico = charger le dictionnaire;

   Choisir le type de jeu (longueur ou catégorie);
   Si le type de jeu est la longueur alors
     Saisir la longueur (entre 4 et 15);
     Collection Mots = extraire les mots de Dico qui ont la longueur
choisie;
   Sinon
     Choisir la catégorie;
     Collection Mots = extraire les mots de Dico qui appartiennent à la
catégorie choisie;
   Finsi
```

```
 M = Tirer aléatoirement un mot parmi les mots de Mots;

 // On va pouvoir jouer
 //
 // On affiche les éléments suivants :
 //     - les lettres du mot (visibles et cachés)
 //     - le pendu (nombre de coup invalide)
 //     - le nombre de coup joué
 //     - les lettres jouées
 //
 Tantque non fini faire
   Si le nbre de coup invalide = 7 ou plus de lettres cachées alors
      fini = vrai;
      Afficher le résultat;
   Sinon
      Afficher les éléments;
      Saisir une lettre non encore jouées;
      Incrémenter le nombre de coup joué;
      Si la lettre appartient au mot alors
        maj des lettres jouées;
      Sinon
        Incrémenter le nombre de coup invalide;
        maj des lettres jouées;
      Finsi
   Finsi
 Fintantque
Fin
```

On décompose certaines instructions de l'algorithme :

```
//extraire les mots de Dico qui ont la longueur choisie
Debut
 Pour chaque mot du dictionnaire faire
   Si le mot a la bonne longueur alors le prendre
   Finsi
 Finpour
Fin
```

```
//extraire les mots de Dico qui appartiennent à la catégorie choisie
Debut
 Pour chaque mot du dictionnaire faire
   Si le mot appartient à la bonne catégorie alors le prendre
   Finsi
 Finpour
Fin
```

```
//Saisir une lettre non encore jouées
Debut
 Tantque non fini faire
   Saisir une lettre;
   Si cette lettre n'appartient pas aux lettres déjà jouées alors
      fini = vrai;
   Sinon
      afficher "La lettre a déjà été jouée. Veuillez en saisir une autre";
   Finsi
 Fintantque
Fin
```

```
//Afficher les éléments;
Debut
 Afficher une ligne;
 Afficher le nombre de coup joué;
 Saut de ligne;
 Afficher le pendu;
 Saut de ligne;
 Afficher les lettres jouées;
 Saut de ligne;
 Pour chaque lettre du mot à découvrir faire
   Si la lettre appartient aux lettres jouées alors
      Afficher la lettre;
   Sinon
      Afficher une étoile;
   Finsi
 Finpour
 Saut de ligne;
Fin
```

Maintenant on va définir les variables et types de données et ensuite re-écrire ces algorithmes en utilisant ces variables et ces types de données.

```
LeDico : COLLECTION DE MOT_DICO;

TYPE MOT_DICO EST STRUCTURE
 Catégorie : CHAINE;
 Mot       : CHAINE;

LesMots : COLLECTION DE CHAINE;

LesLettreDejaJouées : COLLECTION DE CARACTERE;

NbreCoupJoué : ENTIER;

NbreCoupInvalide : ENTIER;

MotADecouvrir : CHAINE:
```

```
//extraire les mots de Dico qui ont la longueur choisie
PROCEDURE ExtraireMotsLongueur(longueur : ENTIER)
Debut
 Pour chaque m de LeDico faire
   Si longueur(m.Mot)==longueur alors ajouter(LesMots,m.Mot)
   Finsi
 Finpour
Fin
```

```
//extraire les mots de Dico qui appartiennent à la catégorie choisie
PROCEDURE ExtraireMotsLongueur(catégorie : CHAINE)
Debut
 Pour chaque m de LeDico faire
   Si m.Categorie == catégorie  alors ajouter(LesMots,m.Mot)
   Finsi
 Finpour
Fin
```

```
//Saisir une lettre non encore jouées
Debut
 Tantque non fini faire
   Saisir une lettre(l);
   trouve=faux;
   Pour chaque x de LesLettreDejaJouées faire
      Si x == l alors  trouvé = vrai;
      Finsi
   Finpour
   Si non trouvé alors
      fini = vrai;
   Sinon
      afficher "La lettre a déjà été jouée. Veuillez en saisir une autre";
   Finsi
 Fintantque
Fin
```

```
//Afficher les éléments;
Debut
 Afficher une ligne;
 Afficher NbreCoupJoué ;
 Saut de ligne;
 Afficher NbreCoupInvalide ;
 Saut de ligne;
 //Afficher les lettres jouées;
 Pour chaque l de LesLettreDejaJouées  faire
   Afficher l;
 Finpour
 Saut de ligne;
 Pour chaque c de MotADecouvrir faire
   trouve=faux;
   Pour chaque x de LesLettreDejaJouées faire
      Si x == c alors  trouvé = vrai;
      Finsi
   Finpour
   Si trouve alors
      Afficher c;
   Sinon
      Afficher "*";
   Finsi
 Finpour
Fin
```

6.5. Exercice 5

Gestion d'un agenda

```
Debut
 Charger l'agenda;

 Afficher la dernière date courante de l'agenda;
 Tantque non fini faire
      Afficher la date courante;
      Afficher "  1 : Afficher les rdv de la date courante";
      Afficher "  2 : Changer la date courante";
      Afficher "  3 : Ajouter un nouveau rdv";
      Afficher "  4 : Modifier un rdv";
      Afficher "  5 : Supprimer un rdv";
```

```
      Afficher "  0 : quitter";
      Afficher "-------------------------------";
      Afficher "Choisir sa commande";
      Saisir le choix;
      Cas choix de
        1 : AfficherJour;
        2 : ChangerDate;
        3 : AjouterRdv;
        4 : ModifierRdv;
        5 : SupprimerRdv;
        0 :  Enregistrer l'agenda;
             fini = vrai;
      Fincas;
 Fintantque
Fin
```

Pour pouvoir écrire les algorithmes des traitements AfficherJour, AjouterRdv, ModifierRdv, SupprimerRdv il nous faut savoir comment est structuré, en termes de donnée, notre agenda.

1er cas de choix de structure de données

L'agenda est structuré comme une collection de RDV;

```
Chaque RDV est caractérisé par :
   une date
   une heure de début
   une heure de fin
   un texte

Les RDV sont classés par ordre chronologique : date puis heure de début.

On peut avoir des rdv qui se chevauchent.

On appelle AGENDA la collection qui contient tous les RDV.
On appelle DATE_COURANTE la date courante géré par le programme

AGENDA : COLLECTION DE RDV;

RDV est STRUCTURE de
 date       : DATE;
 heureDebut : HEURE;
 heureFin   : HEURE;
 texet      : CHAINE;
FIN STRUCTURE;
```

```
//AfficherJour
Debut
 Pour chaque rdv de AGENDA faire
   Si rdv.date = DATE_COURANTE alors
      Afficher rang de rdv + " :" + rdv.heureDebut + " " + rdv.heureFin +
" " + rdv.texte;
   Finsi
   Si rdv.date > DATE_COURANTE alors sortir boucle;
   Finsi
 Finpour
Fin
```

```
  //AjouterRdv
  Debut
   rdv.date        = Saisir la date;
   rdv.heureDebut  = Saisir l'heure de debut;
   rdv.heureFin    = Saisir l'heure de fin;
   rdv.texte       = Saisir le texte;
   rechercher le rang du rendez-vous qui est immédiatement supérieur à rdv;
   insérer rdv après ce rang dans AGENDA;
  Fin

  En le raffinant, on obtient :
  //AjouterRdv
  Debut
   rdv.date        = Saisir la date;
   rdv.heureDebut  = Saisir l'heure de debut;
   rdv.heureFin    = Saisir l'heure de fin;
   rdv.texte       = Saisir le texte;

   //rechercher le rang du rendez-vous qui est immédiatement supérieur à
rdv;
   trouve = faux;
   Pour chaque rdvcour de AGENDA faire
     Si (rdvcour.date > rdv.date) alors
        rang = rang de rdvcour;
        trouve = vrai;
     Sinon
        Si (rdvcour.date == rdv.date) alors
          Si (rdvcour.heureDebut > rdv.heureDebut) alors
             rang = rang de rdvcour;
             trouve = vrai;
          Finsi
        Finsi
     Finsi
   Finpour
   Si trouvé alors
     inserer(AGENDA,rdv,rang);
   Sinon
     ajouter(AGENDA,rdv);
   Finsi
  Fin
```

```
  // Modifier rdv
  Debut
   AfficherJour;
   Saisir le rang du rdv (parmi ceux proposés) à modifier (ou -1 pour
annuler);
   Si le rang saisie != -1 alors
     Saisir soit une date au format "JJ/MM/AA", soit une heure de début au
format "D"HH:MM, soit une heure de fin au format "F"HH:MM, soit un texte;
     En fonction du format changer soit la date, soit l'heure de début,
soit l'heure de fin, soit le texte, du rdv du rang saisi;
   Finsi
  Fin
```

```
  // Supprimer un rdv
  Debut
```

```
   AfficherJour;
   Saisir le rang du rdv (parmi ceux proposés) à modifier (ou -1 pour
annuler);
   Si le rang saisie != -1 alors
     supprimer(AGENDA,rang saisie);
   Finsi
  Fin
```

```
  //ChangerDate
  Debut
   Afficher "  1 : Passer au jour suivant";
   Afficher "  2 : Passer au jour précédent";
   Afficher "  3 : Nouvelle date";
   Saisir le choix;
   Cas choix de
          1 : DATE_COURANTE = DATE_COURANTE + 1;
          2 : DATE_COURANTE = DATE_COURANTE + 1;;
          3 : DATE_COURANTE = Saisir une date;
        Fincas;

  Fin
```

2ème cas de choix de structure de données

L'agenda est structuré en plusieurs tableaux imbriqués : un tableau d'années contentant un tableau de mois; Chaque mois contenant un tableau de Jour. Le tableau de jour contient une collection de rendez-vous.

```
Chaque RDV est caractérisé par :
   une date
   une heure de début
   une heure de fin
   un texte

Dans un jour, les RDV sont classés suivant l'heure de début.

On peut avoir des rdv qui se chevauchent.

On appelle AGENDA la collection qui contient tous les RDV.
On appelle DATE_COURANTE la date courante géré par le programme

AGENDA : TABLEAU[0..10] DE ANNEE_RDV;
// On considère que en 0 on a l'année 2010 en 1 l'année 2011, …

ANNEE_RDV est TABLEAU[0..11] de MOIS_RDV;

MOIS_RDV est TABLEAU[0..30] de JOUR_RDV;

JOUR_RDV est COLLECTION de RDV;

RDV est STRUCTURE de
 date        : DATE;
 heureDebut  : HEURE;
 heureFin    : HEURE;
 texet       : CHAINE;
FIN STRUCTURE;

DATE_COURANTE : STRUCTURE
 Jour : Entier;
 Mois : ENTIER;
 Annee : ENTIER;
FIN STRUCTURE;
```

Avec cette nouvelle définition des structures de données, les algorithmes précédents s'écrivent de la manière suivante :

```
//AfficherJour
Debut
 A = DATE_COURANTE.Annee;
 M = DATE_COURANTE.Mois;
 J = DATE_COURANTE.Jour;
 Pour chaque rdv de AGENDA[A-2010][M][J] faire
   Si rdv.date = DATE_COURANTE alors
      Afficher rang de rdv + " :" + rdv.heureDebut + " " + rdv.heureFin +
" " + rdv.texte;
   Finsi
 Finpour
Fin
```

```
  //AjouterRdv
  Debut
   rdv.date        = Saisir la date;
   rdv.heureDebut  = Saisir l'heure de debut;
   rdv.heureFin    = Saisir l'heure de fin;
   rdv.texte       = Saisir le texte;

   A = rdv.date.Annee;
   M = rdv.date.Mois;
   J = rdv.date.Jour;

   rechercher le rang du rendez-vous qui est immédiatement supérieur à rdv;
   insérer rdv après ce rang dans AGENDA;
  Fin

  En le raffinant, on obtient :
  //AjouterRdv
  Debut
   rdv.date        = Saisir la date;
   rdv.heureDebut  = Saisir l'heure de debut;
   rdv.heureFin    = Saisir l'heure de fin;
   rdv.texte       = Saisir le texte;

   A = rdv.date.Annee;
   M = rdv.date.Mois;
   J = rdv.date.Jour;

   //rechercher le rang du rendez-vous qui est le premier supérieur à rdv;
   trouve = faux;
   Pour chaque rdvcour de AGENDA[A-2010][M][J]faire
     Si (rdvcour.date > rdv.date) alors
        rang = rang de rdvcour;
        trouve = vrai;
     Sinon
        Si (rdvcour.date == rdv.date) alors
          Si (rdvcour.heureDebut > rdv.heureDebut) alors
             rang = rang de rdvcour;
             trouve = vrai;
          Finsi
        Finsi
     Finsi
   Finpour
   Si trouvé alors
     inserer(AGENDA[A-2010][M][J],rdv,rang);
   Sinon
     ajouter(AGENDA[A-2010][M][J],rdv);
   Finsi
  Fin
```

```
  // Modifier rdv
  Debut
   AfficherJour;
   Saisir le rang du rdv (parmi ceux proposés) à modifier (ou -1 pour
annuler);
   Si le rang saisie != -1 alors
     Saisir soit une date au format "JJ/MM/AA", soit une heure de début au
format "D"HH:MM, soit une heure de fin au format "F"HH:MM, soit un texte;
```

```
    En fonction du format changer soit la date, soit l'heure de début,
soit l'heure de fin, soit le texte, du rdv du rang saisi;
   Finsi
  Fin
```

```
  // Supprimer un rdv
  Debut
   AfficherJour;
   Saisir le rang du rdv (parmi ceux proposés) à modifier (ou -1 pour
annuler);
   Si le rang saisie != -1 alors
     A = DATE_COURANTE.Annee;
     M = DATE_COURANTE.Mois;
     J = DATE_COURANTE.Jour;
     supprimer(AGENDA[A-2010][M][J],rang saisie);
   Finsi
  Fin
```

```
  //ChangerDate
  Debut
   Afficher "  1 : Passer au jour suivant";
   Afficher "  2 : Passer au jour précédent";
   Afficher "  3 : Nouvelle date";
   Saisir le choix;
   Cas choix de
         1 : DATE_COURANTE = DATE_COURANTE + 1;
         2 : DATE_COURANTE = DATE_COURANTE + 1;;
         3 : DATE_COURANTE = Saisir une date;
       Fincas;

  Fin
```

6.6. Exercice 6

L'algorithme de tri par ordre croissant d'un tableau d'entier : l'algorithme du tri à bulle.

Les entiers sont rangés dans un tableau de 0 à N-1

```
  Debut

    Saisir N entiers et les ranger dans le tableau TAB;
    Si N=0 ou N=1 Alors
       Afficher TAB;
       Sortir du programme;
    Finsi;

    Faire
       Pour i=1 à N-2 faire
         Si TAB[i]>TAB[i+1] Alors
            échanger TAB[i] et TAB[i+1];
            il y a eu un échange;
         Finsi
       Finpour
    Tantque il y a eu un échange;
  Fin
```

On peut optimiser cet algorithme car lors de la 1ère boucle POUR l'élément le plus grand se retrouve forcément à la fin du tableau (d'où le nom de cet algo "à bulle"). On peut donc faire en sorte que la deuxième exécution de la boucle POUR ne teste pas cet élément qui a remonté comme une "bulle". Et ainsi de suite sur les autres "bulles".