Chapitre 8: Les Structures en langage C

8. Les Structures en langage C.

8.1. Introduction

- Différence entre une structure et un tableau
  - Un tableau permet de regrouper des éléments de **même type**.
  - Toutefois, il est généralement utile de pouvoir rassembler des éléments de **type différent** tels que des entiers et des chaînes de caractères.
  - Une structure est une suite finie d'objets de types différents.
- Les structures permettent de remédier à cette lacune des tableaux, en regroupant des objets (des variables) au sein d'une entité repérée par un seul nom de variable.
- Les objets contenus dans la structure sont appelés **champs de la structure**.

8. Les Structures en langage C.

8.2. Déclaration d'une structure

- Lors de la déclaration de la structure, on indique les champs de la structure, c'est-à-dire le type et le nom des variables qui la composent :

struct Nom_Structure { **type**_champ1 **Nom**_Champ1;
type_champ2 **Nom**_Champ2;
type_champ3 **Nom**_Champ3;
……
type_champN **Nom**_ChampN;}**;**

- Deux champs ne peuvent avoir le même nom.
- Les données peuvent être de n'importe quel type hormis le type de la structure dans laquelle elles se trouvent.

8. Les Structures en langage C.

8.2. Déclaration d'une structure: Exemples

- struct MaStructure { int Age;
  char Sexe;
  char Nom[12];
  float MoyenneScolaire;
  struct AutreStructure StructBis;
  /* en considérant que la structure *AutreStructure* est définie */ };

- struct MaStructure { int Age;
  char Age;
  struct MaStructure StructBis; };
- Il y a deux raisons à cela :
  - Le nom de variable *Age* n'est pas unique
  - Le type de donnée *struct MaStructure* n'est pas autorisé

8. Les Structures en langage C.

8.2. Déclaration d'une structure:

- La déclaration d'une structure ne fait que donner l'allure de la structure, c'est-à dire en quelque sorte une définition d'un type de variable complexe.
- La déclaration ne réserve donc pas d'espace mémoire pour une variable structurée (variable de type structure).
- Il faut donc définir une (ou plusieurs) variable(s) structurée(s) après avoir déclaré la structure...

8. Les Structures en langage C.

8.3. Définition d'une variable structurée

- La définition d'une variable structurée est une opération qui consiste à créer une variable ayant comme type celui d'une structure que l'on a précédemment déclarée, c'est-à-dire la nommer et lui réserver un emplacement en mémoire.
- La définition d'une variable structurée se fait comme suit :

struct Nom_Structure Nom_Variable_Structuree;

  - ***Nom_Structure*** représente le nom d'une structure que l'on aura préalablement déclarée.
  - ***Nom_Variable_Structuree*** est le nom que l'on donne à la variable structurée.

8. Les Structures en langage C.

8.3. Définition d'une variable structurée

- On peut définir plusieurs variables structurées en les séparant avec des virgules : **struct Nom_Structure Nom1, Nom2, Nom3, ...;**
- Exemple :
  - Soit la structure *Personne* est définie par la forme suivante

```
struct  Personne{ int Age;
                  char Sexe; };
```

  - On peut définir plusieurs variables structurées :

```
struct Personne Ahmed, Khalid, Hassan;
```

8. Les Structures en langage C.

8.4. Accès aux champs d'une variable structurée

- Chaque variable de type structure possède des champs repérés avec des noms uniques. Toutefois le nom des champs ne suffit pas pour y accéder étant donné qu'ils n'ont de contexte qu'au sein de la variable structurée...

- Pour accéder aux champs d'une structure on utilise l'opérateur de champ (un simple point) placé entre le nom de la variable structurée que l'on a défini et le nom du champ :

Nom_Variable.Nom_Champ;

Ainsi, pour affecter des valeurs à la variable *Pierre* (variable de type *struct Personne* définie précédemment), on pourra écrire :

```
Pierre.Age = 18;
Pierre.Sexe = 'M';
```

8. Les Structures en langage C.

8.4. Accès aux champs d'une variable structurée

- Particularités des structures en C
  - Possibilité de l'affectation globale

Dans le langage C (de la norme ANSI seulement) on peut utiliser l'affectation globale du genre : item 1 = item 2 .

  - Initialisation lors de la déclaration

En langage C on peut initialiser une variable structurée lors de sa déclaration, comme pour les tableaux avec des accolades et des virgules.

Par exemple : struct personne { char nom [30] ; char prenom [20] ; int age ; } ;

struct personne item = {"Dubois", "Paul", 30} ;

8. Les Structures en langage C.

8.4. Accès aux champs d'une variable structurée: Exemple

- Le programme suivant définit la structure complexe, composée de deux champs de type double ; il calcule la norme d'un nombre complexe.

```
struct complexe { double re;
                  double im; };
main() { struct complexe z;  z.re=5; z.im=6;
         double norme;
         ...
 norme = sqrt(z.re * z.re + z.im * z.im);
printf("norme de (%f + i %f) = %f \n",z.re,z.im, norme); }
```

8. Les Structures en langage C.

8.4. Accès aux champs d'une variable structurée: Exemple

- Les règles d'initialisation d'une structure lors de sa déclaration sont les mêmes que pour les tableaux. On écrit par exemple :

```
struct complexe z = {2. , 2.};
```

- En ANSI C, on peut appliquer l'opérateur d'affectation aux structures (à la différence des tableaux). Dans le contexte précédent, on peut écrire :

```
...
main() { struct complexe z1, z2;
...
z2 = z1; }
```

8. Les Structures en langage C.

8.5. Tableaux de structures

- Tant donné qu'une structure est composée d'éléments de taille fixe, il est possible de créer un tableau ne contenant que des éléments du type d'une structure donnée. Il suffit de créer un tableau dont le type est celui de la structure et de le repérer par un nom de variable :

struct Nom_Structure Nom_Tableau[Nb_Elements];

- Chaque élément du tableau représente alors une structure du type que l'on a défini...

- Le tableau suivant (nommé *Repertoire*) pourra par exemple contenir 8 variables structurées de type *struct Personne* :

struct Personne Repertoire[8];

8. Les Structures en langage C.

8.6. Exemple

```
#include <stdio.h>
#include <string.h>
struct Personne
    {char nom[30];
     char tel[20];};
struct Personne fichier[100];
int n; char name[30];
void initialiser() {
    n = 0;
    printf("Nom : "); gets(name);
    while (name[0] != '\0') {strcpy(fichier[n].nom, name);
    printf("Numero de telephone : "); gets(fichier[n].tel);
    n++;
    printf("Nom : ");gets(name);
    }
}
```

8. Les Structures en langage C.

8.6. Exemple

```
void afficher() {
   int i;
   for (i = 0; i < n; i++)
   printf("%30s : %20s\n", fichier[i].nom,  fichier[i].tel);
}

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

8. Les Structures en langage C.

8.7.Type pointeur de structure

- Voulant créer un répertoire on considère des structures comportant des champs pour le nom et le numéro de téléphone :

```
struct donnee { char nom [30] ; char tel [20] ; } ;
struct donnee *ptr, personne ;
ptr = &personne ;
 strcpy((*ptr).nom, "Paul") ;
 strcpy((*ptr).tel, "01 24 34 44 54") ;
```

- Opérateur flèche.- Les parenthèses dans l'accès au champ (*ptr).nom sont obligatoires. Pour ne pas alourdir les notations, on peut utiliser l'opérateur flèche équivalent qui donne : strcpy(ptr->nom, "Paul") ;

➔ la flèche étant tout simplement obtenue, au clavier, par la concaténation des symboles tiret et strictement plus grand.

8. Les Structures en langage C.

8.8. Structure comme paramètre de fonction

- Une structure peut servir de paramètre à une fonction, avec passage par valeur ou par adresse, et peut être le résultat d'une fonction, sans qu'il n'y ait rien de plus à en dire.
- Exercice:

Ecrire des fonctions affiche() et saisie() pour la structure précédente.

[Les déclarations seront :

void affiche(struct donnee personne) ;

void saisieE(struct donnee *personne) ;

les appels se faisant sous la forme :

affiche(personne);

saisie(&personne);]

Chapitre 9: La gestion des fichiers en langage C

9. La gestion des fichiers en C.

9.1. Introduction

- Un programme a en général besoin:
  - De lire des données(texte, nombres, images, sons, mesures,….;)
  - De sauvegarder des résultats(texte, nombres, images, sons,…)

➔ Le C offre la possibilité de lire et d'écrire des données dans un fichier

- Pour des raisons d'efficacité, les accès à un fichier se font par l'intermédiaire d'une mémoire tampon, ce qui permet de réduire le nombre d'accès aux périphériques.
- Pour manipuler un fichier, on utilise un pointeur sur une donnée spécifique dont le type est **FILE**(structure prédéfinie que nous n'avons pas besoin de connaitre précisément):

FILE *fichier

- La variable fichier contiendra l'adresse en mémoire du début du fichier

9. La gestion des fichiers en C.

9.2. Ouverture et fermeture d'un fichier

9.2.1. La fonction fopen (ouverture d'un fichier à l'aide de cette fonction)

```
Fichier = fopen("C:/Data/fichier1.txt","r");
```

Mode d'ouverture

- Cette fonction renvoie un pointeur sur le fichier ouvert.
- fopen est définie dans le fichier stdio.h par:

```
FILE *fopen (char *nom, char *mode)
```

- **nom** est une chaine de caractères contenant le nom du fichier ou bien un flot de données standard(stdin,…)
- **mode** désigne le type de traitement des données

9. La gestion des fichiers en C.

9.2. Ouverture et fermeture d'un fichier

9.2.1. La fonction fopen

- Les spécificateurs de mode d'accès diffèrent suivant le type de fichier considéré. On distingue:
  - ✓ **les *fichiers textes***, pour lesquels les caractères de contrôle seront interprétés en tant que tels lors de la lecture et de l'écriture;
  - ✓ **les *fichiers binaires***, pour lesquels les caractères de contrôle se sont pas interprétés.
- Le format binaire va occuper beaucoup moins de place en mémoire.
- Comme il s'agit d'une recopie directe de la mémoire, on n'a pas à savoir comment est codé chaque nombre.

9. La gestion des fichiers en C.

9.2. Ouverture et fermeture d'un fichier

9.2.1. La fonction fopen

Les différents modes d'accès sont les suivants :

"r"	ouverture d'un fichier texte en lecture
"w"	ouverture d'un fichier texte en écriture
"a"	ouverture d'un fichier texte en écriture à la fin
"rb"	ouverture d'un fichier binaire en lecture
"wb"	ouverture d'un fichier binaire en écriture
"ab"	ouverture d'un fichier binaire en écriture à la fin
"r+"	ouverture d'un fichier texte en lecture/écriture
"w+"	ouverture d'un fichier texte en lecture/écriture
"a+"	ouverture d'un fichier texte en lecture/écriture à la fin
"r+b"	ouverture d'un fichier binaire en lecture/écriture
"w+b"	ouverture d'un fichier binaire en lecture/écriture
"a+b"	ouverture d'un fichier binaire en lecture/écriture à la fin

9. La gestion des fichiers en C.

9.2. Ouverture et fermeture d'un fichier

9.2.1. La fonction fopen

- Ces modes d'accès ont pour particularités:

- Si le mode contient la lettre r, le fichier doit exister.
- Si le mode contient la lettre w, le fichier peut ne pas exister. Dans ce cas, il sera créé. Si le fichier existe déjà, son ancien contenu sera perdu.
- Si le mode contient la lettre a, le fichier peut ne pas exister. Dans ce cas, il sera créé. Si le fichier existe déjà, les nouvelles données seront ajoutées à la fin du fichier précédent.

9. La gestion des fichiers en C.

9.2. Ouverture et fermeture d'un fichier

9.2.2. La fonction fclose

➔ Il faut toujours fermer un fichier après l'avoir utilisé ➔ Afin de libérer la mémoire

- **La fonction fclose** permet de fermer le flot qui a été associé à un fichier par la fonction fopen. Sa syntaxe est :

fclose(fichier)

- La fonction fclose retourne un entier qui vaut zéro si l'opération s'est déroulée normalement (et une valeur non nulle en cas d'erreur).
- fclose est définie dans le fichier stdio.h:

```
int fclose(FILE *fichier);
```

9. La gestion des fichiers en C.

9.3. Les entrées-sorties formatées

9.3.1. La fonction d'écriture fprintf

La fonction fprintf, analogue à printf, permet d'écrire des données dans un fichier. Sa syntaxe est

fprintf(FILE * fichier, format, exp…)

Les spécifications de format utilisées pour la fonction fprintf sont les mêmes que pour printf.

Exemple:

```
int main(){
   FILE* fichier=NULL;
 fichier=fopen("test.txt". "w");
     if (fichier!=NULL){printf("Quel est votre nom? ");
     char nom[100];scanf("%s",nom);
     printf("Quel est votre age? ");
     long age=0;scanf("%ld",&age);
     fprintf(fichier,"Votre nom est %s et vous avez %ld ans", nom, age);
     fclose(fichier);}
return 0;  }
```

9. La gestion des fichiers en C.

9.3. Les entrées-sorties formatées

9.3.2. La fonction de saisie fscanf

La fonction fscanf, analogue à scanf, permet de lire des données dans un fichier.

Sa syntaxe est semblable à celle de scanf :

fscanf(FILE * fichier,"chaîne de contrôle",argument-1,...,argument-n)

Les spécifications de format sont ici les mêmes que celles de la fonction scanf.

Exemple: double a;

fscanf(fichier, "%lf", &a)

➔ presque même syntaxe que scanf: scanf("%lf",&a);

9. La gestion des fichiers en C.

9.3. Les entrées-sorties formatées

9.3.3. Exemple: écriture dans un fichier

```
include <stdio.h>
int main()
{ double a=1.5, b=2.5;
FILE *fichier;
//ouverture du fichier en écriture grâce à "w "
fichier= fopen ("essai.txt ",   "w");
// verifier que le fichier a bien été ouvert
 if(fichier !=NULL)
{   // Ecriture
 fprintf (fichier, "%lf\n%lf\n", a,b);
// fermeture du fichier
 fclose(fichier);
}}
```

9. La gestion des fichiers en C.

9.3. Les entrées-sorties formatées

9.3.3. Exemple: lecture à partir d'un fichier

```
#include <stdio.h>
int main() {
    int i;
    double T[2]
    FILE *fichier;
    //ouverture du fichier en écriture grâce à "r "
    fichier= fopen ("essai.txt ",   "r");
    // verifier que le fichier a bien été ouvert
     if(fichier !=NULL) {
       for(i=0;i<2;i++)  fscanf (fichier, "%lf\n", T+i);
       fclose(fichier);
     }
    for(i=0;i<2;i++)   printf("%lf\n", T[i]);
}
```

9. La gestion des fichiers en C.

9.3. Les entrées-sorties formatées

9.3.4. Ecriture/lecture de fichiers binaires

❑ **Ecriture d'un bloc de données binaire**

```
int fwrite(void *source, int taille_type, int nombre, FILE *flot)
```

fwrite: • Ecrit tout un bloc de données en un seul appel

• Retourne un entier=nombre d'éléments effectivement écrits

Exemple: `fwrite(tableau, sizeof(float), DIM, fichier);`

Pointeur sur les données — Pointeur sur le fichier

❑ **Lecture d'un bloc de données en binaire**

```
int fread(void *destination, int taille_type, int nombre, FILE *flot)
```

fread : • Lit un bloc de données en un seul appel

• Retourne un entier = nombre d'éléments effectivement lus

Exemple: `fread(tableau, sizeof(float), DIM, fichier);`

Pointeur sur les données — Pointeur sur le fichier

9. La gestion des fichiers en C.

9.3. Les entrées-sorties formatées

9.3.5. Exemple: Ecriture/lecture de fichiers binaires

```
# define DIM 10000
 int main(){ int i;
 double  T1[DIM], T2[DIM];
 FILE *fichier;
 for (i=0;i<DIM;i++)  T1[i]=i*atan(1);
 fichier =fopen("essai2.bin", "wb");
 if(fichier !=NULL) {
     fwrite(T1,sizeof(double),DIM,fichier);
     fclose(fichier);
 }
 fichier= fopen("essai2.bin", "rb");
  if(fichier !=NULL) {
        fread(T2,sizeof(double),DIM,fichier);
        fclose(fichier);
   }
}
```

	Fichiers textes	Fichiers binaires
Taille	Peu compacte	compacte
Lisible avec un programme courant	oui	non
Lisible avec un programme spécifique	oui	oui
Écriture/Lecture par blocs	**non**	**oui**
Fonctions	fopen(mode r,w ou a) fclose() fprintf() fscanf()	fopen(mode rb, wb ou ab) fclose() fwrite() fread()

9. La gestion des fichiers en C.

9.4. Positionnement dans un fichier

- Les différentes fonctions d'entrées-sorties permettent d'accéder à un fichier en mode séquentiel : les données du fichier sont lues ou écrites les unes à la suite des autres.

- Il est également possible d'accéder à un fichier en *mode direct*, c'est-à-dire que l'on peut se positionner à n'importe quel endroit du fichier.

- La fonction **fseek** permet de se positionner à un endroit précis ; elle a pour prototype :

```
int fseek(FILE *flot, long deplacement, int origine);
```

9. La gestion des fichiers en C.

9.4. Positionnement dans un fichier

- La variable ***deplacement*** détermine la nouvelle position dans le fichier. Il s'agit d'un déplacement relatif par rapport à l'origine ; il est compté en nombre d'octets.

- La variable ***origine*** peut prendre trois valeurs :
  - ✓ SEEK SET (égale à 0) : début du fichier ;
  - ✓ SEEK CUR (égale à 1) : position courante ;
  - ✓ SEEK END (égale à 2) : fin du fichier.

Annexes

Problème de recherche binaire(dichotomique)

- **le problème de la recherche dichotomique consiste à rechercher un élément dans un tableau de n éléments, procédant de la manière suivante :**

1. on partitionne le tableau en deux parties égales

2. si l'élément du milieu est celui qu'on cherche, on arrête la recherche

3. sinon, si l'élément du milieu est supérieur à celui qu'on cherche, alors, comme le tableau est déjà trié, ce dernier ne peut être que dans la partie gauche de l'élément du milieu

4. autrement l'élément du milieu est inférieur supérieur à celui qu'on cherche. Dans ce cas, ce dernier ne peut être que dans la partie droite de l'élément du milieu.

- **Le cas de base est celui où l'espace de recherche est nul.**

Diviser pour régner
Algorithme de recherche par dichotomie

- Recherche un nombre entier dans un tableau préalablement trié par ordre croissant (et sans doublons).

Principe

- savoir où se trouve le nombre 9 dans le tableau suivant :

indice	1	2	3	4	5	6	7	8	9
valeur	0	2	4	5	6	9	14	20	30

- Une méthode de bourrin consiste à passer tous les nombres du tableau en revue, de gauche à droite. Dans cet exemple, il faudra donc tester les 6 premiers nombres pour savoir que le 6ème nombre est un 9 ! Comme je vous l'ai dit, 6 tests, c'est une méthode de bourrin

Diviser pour régner

- On propose de le faire en 3 étapes seulement avec la méthode par dichotomie, signifie «couper en deux».

1. *le nombre du «milieu», c'est-à-dire le numéro 5 (opération : (1+9)/2=5*
2. *Comme 6 est plus petit que 9, on recommence mais seulement avec la partie supérieure du tableau car comme le tableau est trié, il ne peut y avoir de 9 avant !*

indice	6	7	8	9
valeur	9	14	20	30

3. *On prend à nouveau le nombre du «milieu». L'opération (6+9)/2 donne 7,5 ! Donc nous allons considérer que le nombre du «milieu» est le n°7 (je sais, ce n'est pas mathématiquement rigoureux, mais c'est bien pour cela que j'utilise des guillemets depuis le début).*
4. *Le nombre obtenu (15) est plus grand que 9, on recommence c partie inférieure du tableau.*

6	7
9	14

5. *Le nombre du «milieu» est le n°6 (logique) et il vaut 9. C'est gagné ! Notre résultat est que «le 9 est à la 6ème case».*

Question: *Quel est l'avantage de la recherche dichotomique?*

- Avec cet algorithme, je n'ai testé que la 5ème, la 7ème et la 6ème valeur du tableau, soit **seulement 3 tests** au lieu de 6 pour la méthode «bourrin» (on parle de force brute). Et cet écart peut très vite s'accroître notamment pour rechercher de grandes valeurs dans un grand tableau.

Question: *Quel est l'avantage de la recherche dichotomique?*

- Lors de la recherche dans un tableau de 1024 éléments:

  - le pire des cas pour la recherche séquentielle peut entraîner 1024 exécutions de la boucle.

  - le pire des cas pour la recherche dichotomique peut entraîner 10 exécutions de la boucle.

- recherche tableau de 1 048 576 éléments:

  - le pire des cas pour recherche séquentielle

    1 048 576 exécutions de la boucle.

  - le pire des cas : recherche dichotomique 20 exécutions de la boucle.

Méthodes de Tris

1. Tri par sélection
2. Tri par insertion
3. Tri Shell
4. Tri à bulles
5. Tri indirect
6. Tri rapide

Méthodes de Tris

- Créer un ordre dans les données
  - Ex. Ordre croissant, décroissant des valeurs numériques
  - Ordre croissant des mots, dossiers, personnes…
- Pourquoi trier?
  - Faciliter la recherche
  - Faciliter la gestion en générale

TRI PAR SELECTION

Principe du tri par sélection

Cette méthode est très simple. Supposons que l'on veuille trier les n éléments du tableau t.

Donnons-nous un tableau T de n nombres. Le principe du tri par sélection est le suivant. On **sélectionne le maximum** (le plus grand) de tous les éléments, et on le place en dernière position n-1par un échange. Il ne reste plus qu'à trier les n-1 premiers éléments, pour lesquels on itère le procédé.

Exemple

Soit le tableau

6	3	7	2	3	5

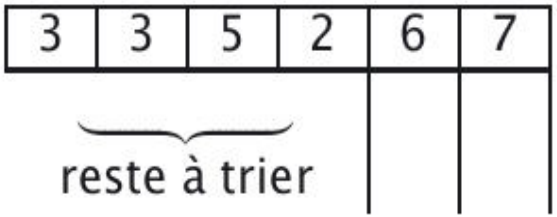
Le maximum de ce tableau est 7. Plaçons le 7 en dernière position par un échange.

6	3	5	2	3	7

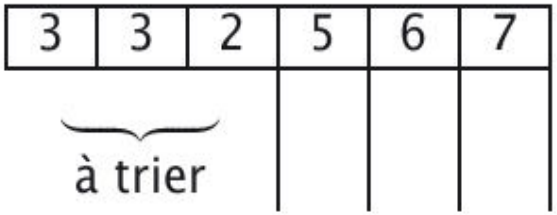
reste à trier

TRI PAR SELECTION

Le 7 est à sa position définitive ; il ne reste plus qu'à trier les cinq premiers éléments. Le maximum de ces cinq éléments est 6. Plaçons-le à la fin par un échange :



Le maximum parmi les nombres restant à trier est 5. Plaçons-le en dernière position par un échange :



Enfin le maximum parmi les nombres à trier est 3. On le place en dernier et le tableau est trié.

TRI PAR SELECTION

Algorithme

L'algorithme de tri par sélection (par exemple sur des nombres entiers) est le suivant :

```
PROCEDURE TriSelection(entier *T, entier n)
début
   entier k, i, imax, temp;
   pour k ← n-1 à 1 pas -1 faire
      /* recherche de l'indice du maximum :
      imax ← 0;
      pour i ← 1 à k pas 1 faire
         si T[imax] < T[i]  faire
            imax ← i;
         fin faire
      fin faire
      /* échange : */
      temp ← T[k];
      T[k] ← T[imax];
      T[imax] ← temp;
   fin faire
fin
```

Réservation de la case

Recherche de l'élément concerné

Permutation des deux valeurs

TRI PAR SELECTION

Complexité

Supposons que le tableau est noté T et sa taille N

```
PROCEDURE TriSelection(entier *T, entier n)
début
   entier k, i, imax, temp;
   pour k ← n-1 à 1 pas -1 faire
      /* recherche de l'indice du maximum : */
      imax ← 0;
      pour i ← 1 à k pas 1 faire
         si T[imax] < T[i]  faire
            imax ← i;
         fin faire
      fin faire
      /* échange : */
      temp ← T[k];
      T[k] ← T[imax];
      T[imax] ← temp;
   fin faire
fin
```

/* Boucle N-1 fois */

/* Boucle N-1 fois */

/* (N-1)+(N-2)+....+2+1=N(N-1)/2. Comparaisons */

/* N-1 échanges */

Tri par sélection :

- nombre de comparaison de l'ordre de $N^2/2$;
- nombre d'échanges de l'ordre de N.

Donc la méthode en $O(n^2)$.

Tri par insertion

- Principe
- Simulation
- Résolution de problèmes
- Complexité

Principe

- Le **tri par insertion** est le tri *le plus efficace* sur des listes de **petite taille**. C'est pourquoi il est utilisé par d'autres méthodes comme le tri rapide (ou *quicksort*). Il est d'autant plus rapide que les données sont déjà triées en partie dans le bon ordre.

- Le principe de ce tri est très simple: c'est le tri que toute personne utilise naturellement quand elle a des dossiers, des cartes (ou n'importe quoi d'autre) à classer. On prend un dossier et on le met à sa place parmi les dossiers déjà triés. Puis on recommence avec le dossier suivant.

TRI PAR INSERTION

Principe

Le tri par insertion insère, au fur et à mesure, l'élément frontière en position j dans la partie triée.

Etape j :

(a)

Arrêt des permutations quand $a_j < a_k$

(b)

La stratégie est identique à celle utilisée par les joueurs de cartes.

TRI PAR INSERTION

Exemple:
On trie d'abord les deux premiers éléments, puis on insère le 3ème à sa place pour faire une liste triée de 3 éléments, puis on insère le 4ème élément dans la liste triée. La liste triée grossit jusqu'à contenir les n éléments.

Soit le tableau

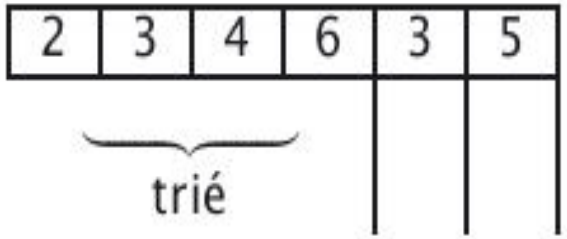
En triant les deux premiers éléments on obtient :

En insérant le troisième élément à sa place dans la liste triée :

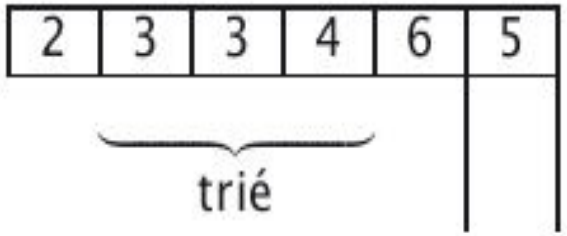
TRI PAR INSERTION

Exemple

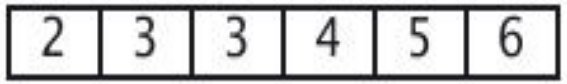
En insérant le quatrième élément à sa place dans la liste triée :



En insérant le cinquième élément à sa place dans la liste triée :



En insérant le sixième élément à sa place dans la liste triée :



Pour insérer un élément à sa place, on doit décaler les éléments déjà triés qui sont plus grands.

TRI PAR INSERTION

Algorithme

L'algorithme de tri par insertion est le suivant :

```
PROCEDURE TriInsertion(entier *T, entier n)
début
    entier k, i, v;
    pour k ← 1 à n-1 pas 1 faire
        v ← T[k];                          O(1)
        i ← k-1;                           O(1)
        /* On décale les éléments pour l'insertion */
        tant que i ≥ 0 et v < T[i] faire   O(1) et O(1)
              T[i+1] ← T[i];               O(1)
              i ← i-1;                     O(1)
         fin faire
         /* On fait l'insertion proprement dite */
         T[i+1] ← v;                       O(1)
    fin faire
fin
```

Les séquences d'échappement

\n	nouvelle ligne
\t	tabulation horizontale
\v	tabulation verticale (descendre d'une ligne)
\a	sonnerie
\b	curseur arrière
\r	retour au début de ligne
\f	saut de page
\\	trait oblique (back-slash)
\?	point d'interrogation
\'	apostrophe
\"	guillemets
\0	fin de chaîne

Fin du cours