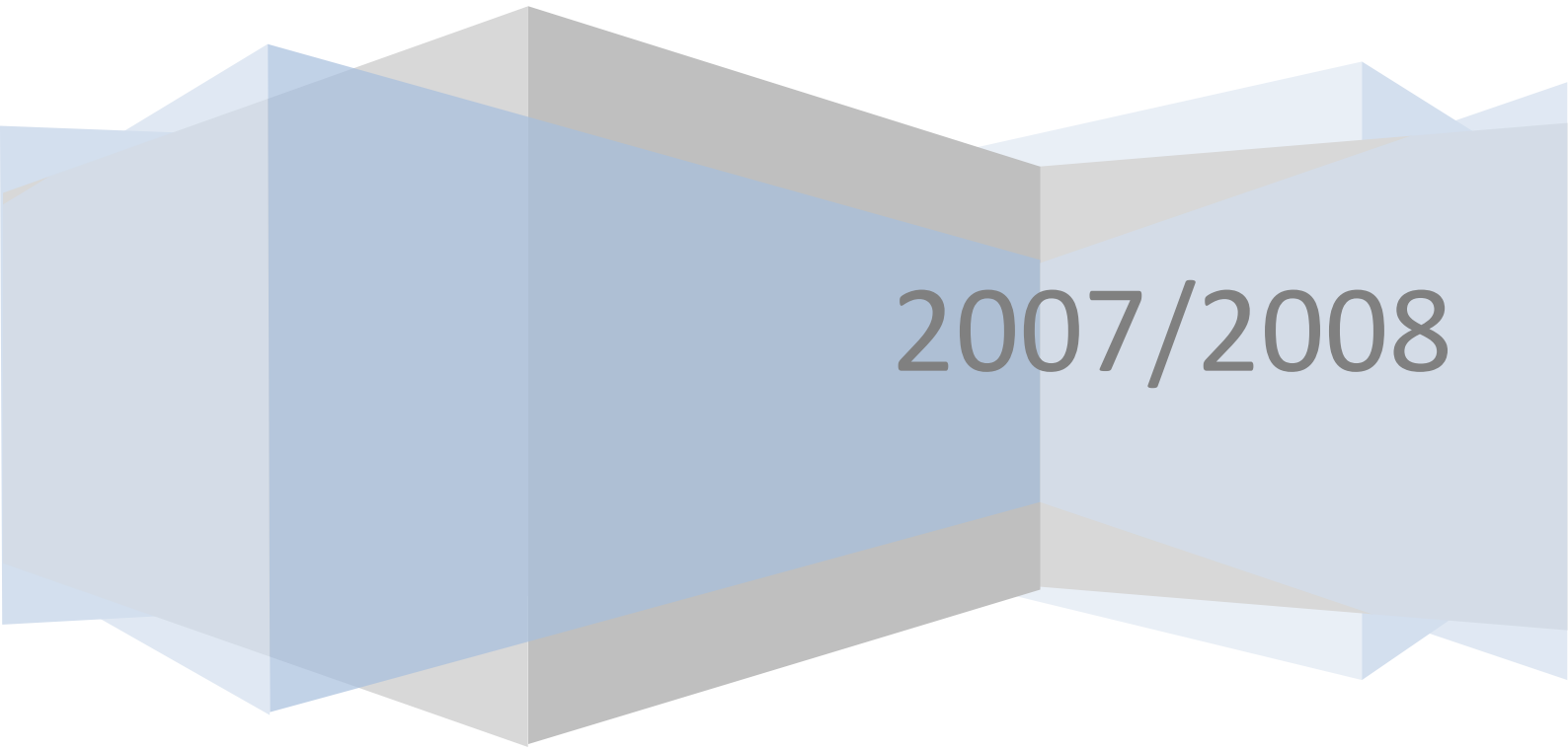
Langage C : Algorithmes + Programmation

Bastien Morier



2007/2008

Sommaire

I. Introduction

C'est un moyen pour présenter la résolution par calcul d'un problème à une autre physique ou virtuelle.

C'est un énoncé dans un langage bien défini d'une suite d'opération permettant de résoudre par calcul un problème.

Un algorithme énonce une résolution sous la forme d'une série d'opération à effectuer.

La mise en œuvre de l'algorithme consiste en l'écriture de ces opération dans un langage de programmation et concise alors la brique de base d'un programme informatique.

C'est un langage de programmation formel standard en français et traduit dans le langage informatique choisi.

II. Analyse

Objectif :

- modéliser l'expression des besoins du demandeur en vue du développement logiciel.
- pour les développements procéduraux : sart (Structured Analyse Real Time)
- pour les développements objets : UML (Unified Modeling Langage)

III. Variables

i. *Définition des variables*

Par définition un algorithme :

- traite une ou des variables entrantes : les données
- restitue une ou des variables sortantes : les résultats

Un algorithme qui ne satisfait pas à ces deux conditions n'a pas de justification et par conséquent n'existe pas.

Les données traitées par l'algorithme sont de deux types : variable ou constante.
Chaque variable et constante doit être déclarée avant d'être utilisé : nom + type.

Cette déclaration consiste en une réservation en mémoire de la place nécessaire et suffisante à l'hébergement de la valeur.

ii. *Types de variables*

- Entier : âge
- Réel : surface
- Caractère : monCar
- Chaîne de caractère : maChaine
- Booléen à deux états : vrai / faux
- Booléen : test

IV. Règles d'écriture des variables et des constantes

On fait particulièrement attention à choisir des noms explicites pour favoriser la compréhension de l'algorithme et plus tard la maintenance du code écrit.

Les noms sont composés exclusivement de lettres, de chiffres et des symboles « - » et « _ ».

Les constantes sont écrites en majuscules.

Il est indispensable d'initialiser les variables et les constantes déclarées au début de l'algorithme. Cela évite les effets de bord lors d'exécution du code.

Aucun accent n'est autorisé.

Les espaces sont interdits.

De préférence, on ne commence pas par un chiffre.

V. Opérateurs

iii. Opération entré – sortie

- Lire (variable)
- Écrire (« variable »)

iv. Opérateurs d'assignation

Addition de la valeur de gauche à la valeur de droite : + = (x ← x+2)
Soustraction de la valeur de gauche à la valeur de droite : - =
Multiplication de la valeur de gauche à la valeur de droite : * =
Division de la valeur de gauche à la valeur de droite : / =
Incrémentation : ++
Décrémentation : - -

v. Opérateur logiques

OU : | |
ET : &&
NON : !

vi. Opérateurs logique bit à bit

OU : |
ET : &
OU exclusif : ^

vii. Comparateurs de nombres

Egalité : = =
Infériorité stricte : <
Supériorité stricte : >
Infériorité : < =
Supériorité :> =
Différent : ! =

viii. Priorité des opérateurs

Plus prioritaire	**( )**	**[ ]**										
…	**- -**	**++**	**!**		**-**							
…	*****	**/**	**%**									

…	+	-										
…	<	< =	> =	>								
…	= =	! =										
…	^											
…	|											
…	&&	||										
…	?	:										
…	=	+ =	- =	* =	/ =	%	<<	>>	>>>	&	^ =	| =
Moins prioritaire	,				=	=	=	=	=	=		

Exercice :

*(5 * X) + 2 * ((3 * B) + 4) (5 * (X + 2) * 2) * (B + 4)*
*= 90 + 2 * 19 = (5 * 14 * 3) * 9*
= 98 = 1890

A == (B = 5) A += (X + 5)
FAUX A += 17
A = 37

*A != (C *= (-D)) A *= C + (X – D)*
*A != 20 A *= C + 10*
*FAUX A *= 0*
A = 0

VI. Structure d'un algorithme

```
/* Zone d'en-tête de l'algorithme (en commentaires)

----------------------------------------***[_ Titre _]***----------------------------------------
|                                                                                               |
| Auteur Date de création                                                                       |
|                                                                                               |
| Fonction (détails fonctionnels de l'algorithme)                                               |
|                                                                                               |
----------------------------------------*****************----------------------------------------
/* Constantes
Définition des constantes // Les constantes doivent être écrivent en majuscules

/* Variables
Définition des variables

/* Début
Initialisation des variables et des constantes
```

Bloc d'instructions de l'algorithme
...
/* Fin

Exercice :

Calculer l'intérêt et la valeur acquise par une somme placée pendant un an avec intérêt de 3,3%

```
/* Constantes
        TAUX-INTERET : réel
/* Variables
        ValeurAcquise, Somme : réel
/* Début
        TAUX-INTERET 3,3
        Écrire (« Entrez une somme: »)
        Lire (Somme)
        ValeurAcquise Somme * (TAUX_INTERET / 100)
        Écrire (« Valeur Acquise = », ValeurAcquise)
/* Fin
```

VII. Algorithme des structures

a) *Algorithme*

ix. *Structures de condition simple :*

```
Si (condition) Alors
        Bloc d'instruction si condition = vrai
Finsi
```

x. *Structure de condition complexe :*

```
Si (condition) Alors
        Bloc d'instruction si condition = vraie
SinonSi
        Bloc d'instruction si condition = fausse
Finsi
```

xi. *Structure de condition à choix multiples :*

```
SelonCas VariableDeChoix
        Cas VariableDeChoix : 'x'
                Bloc de traitement de cas
        Cas VariableDeChoix : 'y'
                Bloc de traitement de cas
                ...
        Autre cas
                Bloc de traitement des cas non prévus
FinSelonCas
```

xii. *Structure de boucle d'itération : Pour :*

```
Index : entier
Index 0 // initialisation de la valeur voulue

Pour Index de <ValeurInitiale> à <ValeurFinale> pas de <pas>
        Bloc d'instructions
FinPour
```

Δ Nombre de traitement de la boucle : (ValeurFinale – ValeurInitiale) / pas

Remarque : on utilise cette structure quand on connaît le nombre de passage dans la boucle

xiii. Structure de boucle TantQue :

```
TantQue (condition) Faire
        Bloc de traitement de la boucle
FinTantQue
```

xiv. Structure de boucle Jusqu'à :

```
Répéter
        Bloc d'instruction de la boucle
TantQue (condition)
```

Remarque : on utilise cette structure quand on ne connaît pas le nombre de passage dans la boucle mais que l'on va y passer au moins une fois.

b) Traduction en C

xv. Structure de condition simple :

```
if (condition)
{
        Bloc si condition vraie...
}
```

xvi. Structure de condition multiple :

```
if (condition)
{
        Bloc si condition vraie...
}
else
{
        Bloc si condition fausse...
}
```

xvii. Structure Pour :

```
int i;
for (i=1; i<=10; i++)
{
        Bloc d'instructions...
}
```

xviii. Structure TantQue :

```
while (condition)
{
        Bloc d'instructions...
}
```

xix. Structure Répéter :

```
do
{
        Bloc d'instructions...
```

}while (condition)

VIII. Fonctions et Procédures

Dans une fonction ou une procédure, les paramètres sont passés par valeur.

On ne peut donc pas modifier dans la fonction les paramètres effectifs du programme appelant.

Si on veut qu'une fonction modifie les paramètres effectifs dans le programme appelant, il faut passer non plus une valeur mais l'adresse du paramètre.

La fonction va pouvoir accéder directement au paramètre par son adresse et le manipuler (prendre sa valeur ou/et la changer).

On précise qu'un paramètre est passé par adresse en le soulignant.

fonction <type> <nomfonction> (argument 1 : type 1, argument 2 : type 2)

L'argument 1 est passé valeur :
- Il peut être utilisé dans la fonction
- Il ne peut pas être modifié dans la fonction pour le programme appelant

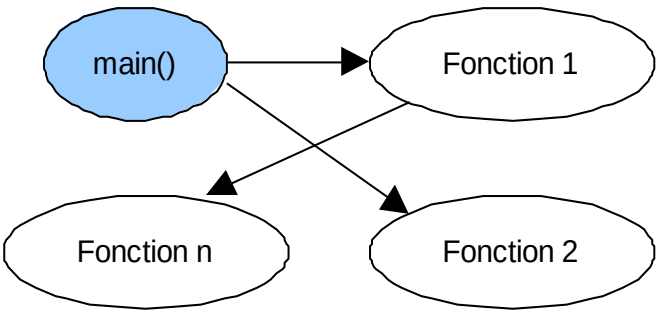
L'argument 2 est passé valeur :
- Il peut être utilisé dans la fonction
- Il peut être modifié dans la fonction pour le programme appelant

a) *Fonctions*

Un programme réalise des fonctions. L'analyse du cahier des charges est d'identifier et de présenter précisément des fonctions. On parle de cahier des charges fonctionnel.

Dans la phase d'analyse, l'algorithme consiste à traduire chacune de ces fonctionnalités en une ou plusieurs fonctions.

Un algorithme principal appelé *main()* aura pour mission de gérer les appels à ces fonctions pour réaliser le programme.



xx. *Déclaration des fonctions*

On déclare la fonction entre l'entête de l'algorithme et le main().

On précise le type de la fonction, c'est à dire le type de paramètre qui sera retourné lors de l'appel de la fonction.

On précise les arguments (ou paramètres) et leurs type qui seront passés à la fonction. Ces arguments seront disponibles (ou visibles) à l'intérieur de la fonction.

xxi. Déclaration : forme canonique

fonction <type> <nomfonction> (argument 1 : type 1, argument 2 : type 2, argument n : type n)

Exemple : retourne la somme de deux variables

fonction réel somme (x : réel, y : réel)

xxii. Définition de la fonction

On définie le corps de la fonction après la déclaration de la fonction.

Le corps de la fonction contient :

- La déclaration de l'initiation des variables et des constantes interne à la fonction

Visibilité réduite à la fonction

- Les instructions qui réalisent le traitement
- Le retour du paramètre si il existe

Exemple :
fonction réel somme (x : réel, y : réel)

```
//Début
        retourne (x + y)
//Fin
```

xxiii. Appel de la fonction

On peut appeler une fonction définie à tout endroit de l'algorithme de programmation.

Les arguments passés à la fonction sont définis et de même type que ceux qui sont déclarés.

Le paramètre retourne par la fonction est récupéré dans une variable du même type que celui qui est déclaré.

b) Procédures

Une fonction qui ne retourne pas de paramètre est appelé une procédure.

Une procédure retourne un résultat sous forme visible (écran, ...)

Par exemple : une procédure qui est chargé d'effacer l'écran

Une procédure ne retourne pas de paramètre en ne spécifié pas le type dans la déclaration.

IX. Les tableaux

xxiv. Problématique

Les variables simples présentées ne permettent pas d'effectuer efficacement des traitements itératifs.

Pour ces traitements, il faut déclarer autant de variable que de traitement.

Exemple : la saisie périodique toute les heures d'une température en vue de traitement statique

- température maximum sur une journée
- température minimum sur une journée

Dans ce cas, il faudrait déclarer 24 variables « heures » à remplir.

Une solution plus efficace : les tableaux de variables

xxv. Caractéristiques

Un tableau est un ensemble de données qui sont toutes de même type.
Le tableau a un nom.
Les données possèdent un identificateur unique : le nom du tableau.
La taille d'un tableau correspond au nombre de cases.
Les données se différentient par leur numéro d'indice.

xxvi. Déclaration d'un tableau à une dimension

Exemple : Age[20] : entier
<NomTableau>[ValeurMax] : type

Indices	
1	Chaque cases
2	contient une
3	variable
i	
n+1	

xxvii. Déclaration d'un tableau à deux dimension

Exemple : Age[20,30] : réel
<NomTableau>[N (nbr de lignes),M (nbr de colonnes)] : type

		indice colonne					
indice lignes	0	1	2	3	4	i	m-1
0							
1							
2							
I							
n-1							

xxviii. Traitement itératif d'un tableau de dimension deux :

Lecture des éléments du tableau
Tab[N,M]

```
Pour i de 1 à N
        Pour j de 1 à M
                Lire Tab[i,j] // accès à la variable d'indices i,j
        FinPour
FinPour
```

Boucles sur les lignes puis sur les colonnes.

X. Les pointeurs

A chaque déclaration d'une variable le compilateur réserve la place nécessaire dans une zone mémoire réservée.

La variable est donc placée à une adresse en mémoire.

Un pointeur est une variable qui contient l'adresse en mémoire d'une variable.

XI. Les chaines de caractères

a) *Algorithme*

Une chaine de caractère est une suite de caractère.

Leurs structure est comparable à celle des tableaux. Il s'agit pratiquement d'un tableau de caractères.

xxix. *Déclaration d'une chaine :*

UneChaine[4] : chaine

xxx. *Initialisation d'une chaine :*

UneChaine ← « toto »

xxxi. *Opération sur les chaines de caractères :*

- Concaténation de deux chaines : les chaines sont mises bout à bout pour fermer une nouvelle chaine. On utilise le signe +.
  Exemple : Chaine1 ← « papa » ; Chaine2 ← « maman »
  Chaine3 ← Chaine1 + Chaine2
  Chaine3 affiche « papamaman »

- Comparaison : ces comparaisons sont basées sur la comparaison de gauche à droite des caractères en valeur ASCII.
  Deux chaines de caractères sont égales si elles possèdent exactement la même suite de caractère.
  Une chaine X est inférieure à une chaine Y dès qu'une valeur ASCII d'un caractère est inférieure.
  On peut ainsi utiliser des chaines de caractères dans l'expression logique :
  - *Si les deux chaines X et Y sont identiques (X = Y) renvoie vrai.*
  - *Si X est inférieure à Y (X < Y) renvoie vrai.*

- Copier une chaine dans une autre chaine
  Exemple : Soit les chaines X et Y, Y peut être copié dans X en faisant X ← Y

- On peut connaitre la longueur d'une chaine, c'est-à-dire le nombre de caractères contenu dans la chaine.
  Exemple : Soit la chaine X ← « Etudiant »
  Longueur(X) = 8

b) *Traduction en C*

Une chaine est une suite de caractères ASCII.

Lorsqu'une chaine de caractère est codée en mémoire, elle se termine systématiquement par le caractère d'échappement « \0 » qui marque la fin de la chaine.

Une chaine de caractère contient donc un caractère de plus.

Lorsque vous définissez une constante de type chaine, le compilateur C lui affecte automatiquement le caractère NULL.

Lorsqu'un programme constitue une chaine (à partir d'une saisie au clavier par exemple), le programme doit placer lui-même le caractère NULL à la fin de la chaine pour la matérialiser.

xxxii. Déclaration d'une chaine :

Char <NomChaine>[taille + 1]
Exemple : char machaine[9] ;

xxxiii. Initialisation d'une chaine :

maChaine = « maChaine » ;

xxxiv. Initialisation d'une chaine sans taille :

maChaine[ ] = « maChaine » ;

xxxv. Manipulation de chaine avec pointeurs :

Déclaration : // pointe sur le premier élément de la chaine
Char *X ;
Char *Y = « Bonjour » ;

Affectation : X = « Bonjour » ;

Création d'un tableau qui pointe vers une chaine : char *jour[7] ;

Pour accéder à ce tableau : char **ptr_jour ;

xxxvi. Opération sur les chaines :

Lecture d'une chaine de caractère X : scanf(« %s », &X) ;
// Si X n'est pas déclaré comme pointeur

scanf(« %s », X) ;
// Si X est déclaré en pointeur

Ecrire une chaine de caractère X : printf(« %s \n», X) ;

On dispose aussi des fonctions gets(X) et puts(X) quelque soit le mode de déclaration.

Copie dans une chaine : strcopy(Y, X) ;
// Copie X dans Y

Connaitre la longueur d'une chaine : fontion srtlen( )

```
int longueur ;
char texte[ ] = « Bonjour » ;
longueur = strlen(texte) ;
// longueur = 7
```

XII. Enumération

a) Algorithme

Définition : une variable énuméré possède un nombre fixe de valeurs prédéfinies.

On définit un nouveau type de données correspondant à cette énumération.

On peut déclarer des variables de ce nouveau type qui ne pourront prendre que les valeurs prédéfinies.

xxxvii. Définition d'un variable énumérée de n élément de valeur élémentX :

type <NomEnumeration> = {élément1, élément2, élément3, élémentN}

Exemple : booléen

type booleen = {faux, vrai}

xxxviii. Utilisation :

UnBooleen ; booleen

xxxix. Déclaration d'une variable énumérée :

<NomEnumeration> : <UneEnumeration>

Exemple : JourSemaine

JourSemaine = {lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche}
LesJourSemaine : JourSemaine
LesJourSemaine ← lundi

b) Traduction en C

xl. Enumération ou type énumérés : énum

```
// Définition d'un type
        enum booleen {faux, vrai};
        enum booleen MonBool ;
        MonBool = vrai;
```

Ou

```
//Définition d'un type
        Enum booleen {faux, vrai} MonBool;
        MonBool = vrai;
```

xli. Définition d'une énumération (type énumération) : enum avec définition d'un type de variable

```
// Définition d'un type
        typedef enum lundi, mardi, mercredi, jeudi, vendredi, samedi, dimanche}jours ;        jours
LesJoursSemaines ;
        LesJoursSemaine = samedi ;
        printf(«Les variables enumerees sont des entiers ») ;
        printf(« Jours de la semaine : %d », LesJoursSemaine) ;
```

XIII. Les Structures

a) Algorithme

Elles permettent de regrouper ensemble des données de type différents correspondant à une même entité logique.

Remarques : en programmation objet les structures deviennent des classes.

Définition :

<Nom-Structure> : <Nom_variable1> ; type
<Nom_variable2> ; type
<Nom_variableN> ; type

Déclaration d'une variable de ce type :

<Nom_Variable_Structure>.<Nom_Variable_i>

Exemple :

StructAgenda : Nom : chaine[20]
NumeroTel : chaine[14]
UnAgenda : type StructAgenda
UnAgenda.Nom = « nom »
UnAgenda.NumeroTel = « xx.xx.xx.xx.xx »

b) *Traduction en C*

```
struct agenda
{
        char Nom[20] ;
        char NumTel[14] ;
} ;

struct agenda UnAgenda ;
strcopy(UnAgenda.Nom, « Nom ») ;
strcopy(UnAgenda.NumTel, « xx.xx.xx.xx.xx »);
printf("Nom : %s, Numero de telephone : %s\n", UnAgenda.Nom, UnAgenda.NumTel);
```

XIV. Les Fichiers

a) *Algorithme*

Les fichiers permettent de stocker des informations sur un disque ou d'autres supports de mémoire permanents.

Il existe deux grandes catégories de fichiers de données :

- Fichiers textes
- Fichiers binaires

Dans ce cours nous ne présenterons que l'utilisation des fichiers textes.

xlii. Fichiers textes

Les données sont stockés sous forme de caractères les uns à la suite des autres. Ils sont organisés en succession de lignes identiques : enregistrement.

Chaque ligne contient les mêmes informations, le fichier est de type séquentiel.

Un enregistrement peut contenir toutes sortes d'informations les unes à la suite des autres :
int, float, char*, etc.

Ces informations sont appelées des champs.

Exemple : carnet d'adresse qui contient dans chaque enregistrement : le nom, le prénom, l'adresse, etc. d'une personne

La séquence d'utilisation d'un fichier est la suivante :

- Déclaration
  fichier : Fichier
- Ouverture du fichier en mode lecture, écriture ou ajout d'un enregistrement
  En mode lecture : ouvrir(fichier) en lecture
  En mode écriture : ouvrir(fichier) en écriture
  En mode écriture ajout : ouvrir(fichier) en ajout
- Lecture ou écriture d'un enregistrement
  Lecture : lire(fichier, X)
  Ecriture : ecriture(fichier, X)
- Fermeture du fichier

  fermer(fichier)

Lors de l'écriture, il faut connaitre la structure d'un enregistrement : quantité et type des informations pour avoir par la suite une lecture correcte.

Lorsque la fin du fichier est atteinte, la variable eof = 1.

b) *Traduction en C*

Entre chaque enregistrement sont stockés les deux caractères CR (ASCII 13) et LF (ASCII 10).

Cela permet de signaler le passage à la ligne suivante.

Cela est géré automatiquement par le langage de programmation.

Les enregistrements peuvent être structurés de deux manières pour retrouver les données à l'intérieur d'un enregistrement :

- Par un délimiteur
  Un point virgule : DUPOND ;JEAN ;13 rue de mon cul
  Une tabulation : DUPOND JEAN 13 rue de mon cul
- Par des champs de tailles identiques : à la largeur fixe
  Avec une largeur fixe : DUPOND JEAN 13 rue de mon cul

Comparaison des deux méthodes :

- Délimiteur : moins de place en mémoire
  Lent
- A largeur fixe : plus de place en mémoire
  Rapide

Les fichiers binaires :

Les octets, quels qu'ils soient, sont écrits à la suite des autres

xliii. *Types d'accès : (pour fichiers texte)*

Accès séquentiel : On lit le fichier par ligne. On accède à une donnée seulement si on a la donnée précédente (lent).

Accès direct : On accède directement à l'enregistrement de son choix en précisant le numéro de l'enregistrement (rapide mais lourd).

Accès indexé : mélange des deux accès ci-dessus (plus compliqué, pour des gros fichiers)

Bibliothèque nécessaire : #include <conio.h>

xliv. Instructions pour accès séquentiel

- Création d'un fichier texte : FILE *f
- Ouverture :
  - En lecture : f = fopen(« C:\\nomFichier.txt », « r ») ;
  - En écriture : f = fopen(« C:\\nomFichier.txt », « w ») ;
  - En écriture ajout : f = fopen(« C:\\nomFichier.txt », « a ») ;

- Ecriture formaté :
  fprintf(fichier, « <fomat> », <informations>) ;
  fprintf(f, « %d\n », x) ;

- Lecture formaté :
  fscanf(fichier, « <fomat> », <&informations>) ;
  fscanf(f, « %d\n », &x) ;
- Lorsque le dernier enregistrement du fichier est atteint, une variable signale la fin du fichier :
  feof(f) = 1
- Fermeture d'un fichier :
  fclose(f) ;

xlv. Instructions pour accès direct

- Lecture d'un enregistrement :
  int fseek(FILE *stream, long offset, int whence) ;
  Paramètres whence :
  - SEEK_SET 0 Début de fichier
  - SEEK_CUR 1 Position actuel du pointeur
  - SEEK_END 2 Fin de fichier
- Lecture d'un enregistrement, principe :
  - On positionne le pointeur de flux sur l'indice (l'octet)
  - On lit l'enregistrement à partir de l'indice;
  - Option de lecture à partir du début du fichier : fseek(fichier, position, 0) ;
  - Utilisé particulièrement pour les enregistrements à taille fixe.

Exemple : *On positionne un enregistrement de 20 caractères (octets)*
On veut lire le 5eme enregistrement
La valeur de position est : (4 x 20) – 1 = 79
Donc : fseek(fichier, 79, 0) ;

0	*19*
20	*29*
30	*39*
40	*49*
50	*59*
60	*69*
…	