Travaux dirigés et pratiques n° 4
Fichiers

Partie TD (2 séances)

Exercice 4.1 (Ecriture de fichier)

Soit l'algorithme ci-dessous (déjà vu en cours).

1. Que fait cet algorithme ?
2. Modifier cet algorithme afin qu'il ne stocke que les lettres minuscules et majuscules non accentuées (et aucun autre caractère) dans un fichier nommé "lettres.txt". De plus, à la fin, l'algorithme affichera à l'écran le message stocké.
3. Simuler cet algorithme lorsque l'utilisateur saisit, dans cet ordre : 'a', '+', '4', 'b', '=', 'c' et '.'.

```
lexique
    flux_fichier flux
    chaine de caracteres nom
    caractere c
debut
1   ecrire("fichier ?")
2   lire(nom)
3   ouvrir(flux , nom) en ecriture
4   ecrire ("Donnez un caractere (. pour terminer)")
5   lire(c)
6   tant que non (c =  '.')  faire
7           ecrire(c) dans flux
8           ecrire ("Donnez un caractere (. pour terminer)")
9           lire(c)
10  fin tant que
11  fermer(flux)
fin
```

▽ **Correction**

Exercice pouvant être soumis pour l'évaluation ; ne pas donner l'algo initial dans le sujet. Si une simulation est demandée, le faire sur une séquence plus courte.

1. Il demande à l'utilisateur de saisir des caractères et les stocke dans un fichier jusqu'à ce que l'utilisateur saisisse un point.
2. Analyse :

   Il faut ajouter un traitement du caractère c saisi. Si c'est une lettre, il est concaténé à $message$ (chaîne, initialisée à "") et écrit dans le fichier $"lettres.txt"$

```
lexique
   caractere c
   flux_fichier flux
   chaine de caracteres message;
debut
1  message ← ""
2  ouvrir(flux, "lettres.txt") en ecriture
3  ecrire ("Donnez un caractere (. pour terminer)")
4  lire(c)
5  tant que non(c = '.') faire
6      si ('a' <= c) et (c <= 'z') alors
7          ecrire(c) dans flux
8          message ← message ~ c
       fin si
9      ecrire("Donnez un caractere (. pour terminer)")
10     lire(c)
11 fin tant que
12 fermer(flux)
13 ecrire(c)
fin
```

3. Simulation :

Instruction	c	flux	message	remarques
debut	?	?	?	
1	?	?	""	
2	?	1	""	
3	?	1	""	Aff : "Donnez un caractere (. pour terminer)"
4	'a'	1	""	
5	'a'	1	""	non(c = '.') ⇔non('a' = '.') ⇔VRAI
6	'a'	1	""	('a' ≤ c) (et c ≤ 'z') ⇔('a' ≤ 'a') (et 'a' ≤ 'z') ⇔VRAI
7	'a'	2	""	
8	'a'	2	"a"	
9	'a'	2	"a"	Aff : "Donnez un caractere (. pour terminer)"
10	'+'	2	"a"	
11->5	'+'	2	"a"	non(c = '.') ⇔non('+' = '.') ⇔VRAI
6	'+'	2	"a"	('a' ≤ c) (et c ≤ 'z') ⇔('+' ≤ 'a') (et '+' ≤ 'z') ⇔FAUX
9	'+'	2	"a"	Aff : "Donnez un caractere (. pour terminer)"
10	'b'	2	"a"	
11->5	'b'	2	"a"	non(c = '.') ⇔non('b' = '.') ⇔VRAI
6	'b'	2	"a"	('a' ≤ c) (et c ≤ 'z') ⇔('a' ≤ 'b') (et 'b' ≤ 'z') ⇔VRAI
7	'b'	3	"a"	
8	'b'	3	"ab"	
9	'b'	3	"ab"	Aff : "Donnez un caractere (. pour terminer)"
10	'='	3	"ab"	
11->5	'='	3	"ab"	non(c = '.') ⇔non('=' = '.') ⇔VRAI
6	'='	3	"ab"	('a' ≤ c) (et c ≤ 'z') ⇔('=' ≤ 'a') (et 'a' ≤ '=') ⇔FAUX
9	'='	3	"ab"	Aff : "Donnez un caractere (. pour terminer)"
10	'c'	3	"ab"	
11->5	'c'	3	"ab"	non(c = '.') ⇔non('c' = '.') ⇔VRAI
6	'c'	3	"ab"	('a' ≤ c) (et c ≤ 'z') ⇔('c' ≤ 'a') (et 'c' ≤ 'z') ⇔VRAI
7	'c'	4	"ab"	
8	'c'	4	"abc"	
9	'c'	4	"abc"	Aff : "Donnez un caractere (. pour terminer)"
10	'.'	4	"abc"	
11->5	'.'	4	"abc"	non(c = '.') ⇔non('.' = '.') ⇔FAUX
12	'.'	?	"abc"	
13	'.'	?	"abc"	Aff : "abc"
fin	'.'	?	"abc"	

Exercice 4.2 (Lecture de fichier)

Soit l'algorithme ci-dessous (déjà vu en cours).

1. Que fait cet algorithme ?
2. Modifier cet algorithme afin qu'il demande à l'utilisateur le nom d'un fichier d'entiers, affiche tous les entiers positifs contenus dans ce fichier puis indique le pourcentage d'entiers positifs dans le fichier.
3. Simuler le fonctionnement de cet algorithme sur le fichier "gains.txt" = <2 ; −10 ; 0 ; 48 ; −3>.

```
lexique
    flux_fichier flux
    chaine de caracteres nom
    caractere temp
debut
1   ecrire("fichier ?")
2   lire(nom)
3   ouvrir(flux , nom) en lecture
4   si (flux) alors
5         lire(temp) dans flux
6         tant que non (fini(flux))  faire
7               ecrire(temp)
8               lire(temp) dans flux
9       fin tant que
10    fermer(flux)
11  sinon
12     ecrire ("ouverture fichier ", nom, " impossible")
13  fin si
fin
```

▽ **Correction**

Exercice pouvant être soumis pour l'évaluation ; préciser l'algorithme à écrire (pas toutes les variantes) et limiter le fichier à 2 à 3 données si une simulation est demandée.

1. Analyse :
   - choix du nom du fichier => variable nom (chaîne de caractères)
   - ouverture du fichier en lecture=> variable ff (flux_fichier)
   - tester si le fichier existe
   - lecture d'une valeur => variable val (entier)
   - répétitive
     Arrêt : échec de lecture
     Traitement
       Si val positif
         mise à jour du nombre de valeurs positives => variable nb (entier, initialisé à 0)
         mise à jour du total des valeurs positives => variable tot (entier, initialisé à 0)
         lecture d'une valeur
   - affichage de la moyenne s'il y avait au moins une valeur
   - fermeture du fichier

```
lexique
   entier val, nb, tot
   flux_fichier ff
```

```
    chaine nom
debut
1   ecrire "Donnez le nom d'un fichier d'entiers :"
2   lire(nom)
3   ouvrir(ff, nom) en lecture
4   si (ff) alors
5      tot ← 0
6      nb ← 0
7      lire(val) dans ff
8      tant que non fini(ff) faire
9         si (val > 0) alors
10           ecrire(val)
11           nb ← nb + 1
12           tot ← tot + 1
           fin si
13        lire(val) dans ff
14     fin tant que
15     fermer(ff)
16     si (nb > 0) alors
17        ecrire("moyenne = ", 100.0*pos/nb)
       sinon
18        ecrire("fichier vide")
       fin si
   sinon
19     ecrire("pas de fichier de nom ", nom)
   fin si
fin
```

2. Simulation :

Instruction	val	nb	tot	ff	nom	remarques
debut	?	?	?	?	?	
1	?	?	?	?	?	Aff : "Donnez le nom d'un fichier d'entiers :"
2	?	?	?	?	"gains.txt"	
3	?	?	?	1	"gains.txt"	lecture de "gains.txt"
4	?	?	?	1	"gains.txt"	VRAI
5	?	?	0	1	"gains.txt"	
6	?	0	0	1	"gains.txt"	
7	2	0	0	2	"gains.txt"	
8	2	0	0	2	"gains.txt"	non fini(ff) ⇔VRAI
9	2	0	0	2	"gains.txt"	val > 0 ⇔2 > 0 ⇔VRAI
10	2	0	0	2	"gains.txt"	Aff : "2"
11	2	1	0	2	"gains.txt"	
12	2	1	2	2	"gains.txt"	
13	-10	1	2	3	"gains.txt"	
14->8	-10	1	2	3	"gains.txt"	non fini(ff) ⇔VRAI
9	-10	1	2	3	"gains.txt"	val > 0 ⇔-10 > 0 ⇔FAUX
13	0	1	2	4	"gains.txt"	
14->8	0	1	2	4	"gains.txt"	non fini(ff) ⇔VRAI
9	0	1	2	4	"gains.txt"	val > 0 ⇔0 > 0 ⇔FAUX
13	48	1	2	5	"gains.txt"	
14->8	48	1	2	5	"gains.txt"	non fini(ff) ⇔VRAI
9	48	1	2	5	"gains.txt"	val > 0 ⇔48 > 0 ⇔VRAI
10	48	1	2	5	"gains.txt"	Aff : "48"
11	48	2	2	5	"gains.txt"	
12	48	2	50	5	"gains.txt"	
13	-3	2	50	6	"gains.txt"	
14->8	-3	2	50	6	"gains.txt"	non fini(ff) ⇔FAUX
9	-3	2	50	6	"gains.txt"	val > 0 ⇔2 > 0 ⇔VRAI
13	-3	2	50	6	"gains.txt"	
14->8	-3	2	50	6	"gains.txt"	non fini(ff) ⇔FAUX
15	-3	2	50	?	"gains.txt"	
16	-3	2	50	?	"gains.txt"	nb > 0 ⇔2 > 0 ⇔VRAI
17	-3	2	50	?	"gains.txt"	aff : moyenne = 25
fin	-3	2	50	?	"gains.txt"	

Exercice 4.3 (Concaténation de fichiers)

Il s'agit d'écrire un algorithme qui demande à l'utilisateur le nom de deux fichiers contenant des données quelconques, puis les concatène dans un troisième fichier dont le nom est également saisi par l'utilisateur.

▽ **Correction**

Exercice pouvant être soumis pour l'évaluation ;

Analyse :

Remarquer que les données étant de types inconnus, il faut les lire comme des chaînes de caractères => variable $donnee$ (chaîne de caractères)

- choix du nom du premier fichier => variable $nom1$ (chaîne de caractères)
- choix du nom du second fichier => variable $nom2$ (chaîne de caractères)
- choix du nom du fichier de concaténation=> variable $nom3$ (chaîne de caractères)

Il faut ensuite recopier le premier fichier dans le fichier de concaténation, puis le second

- ouverture du fichier de concaténation en écriture => variable $flux_conc$ (flux_fichier)
- ouverture du premier fichier en lecture => variable $flux$ (flux_fichier)
- tester si le fichier existe
- répétitive : recopie des valeurs de flux dans $flux_conc$, penser à introduire un séparateur entre les données.
- fermeture de $flux$
- ouverture du second fichier en lecture => variable $flux$ (flux_fichier)
- tester si le fichier existe
- répétitive : recopie des valeurs de flux dans $flux_conc$, penser à introduire un séparateur entre les données.
- fermeture de $flux$
- fermeture de $flux_conc$

```
lexique
   chaine nom1, nom2, nom3, donnee
   flux_fichier flux, flux_conc
debut
   ecrire "Donner le nom d'un fichier de donnees : "
   lire nom1
   ecrire "Donner le nom d'un second fichier de donnees : "
   lire nom2
   ecrire "Donner le nom du fichier de concatenation : "
   lire nom2

   ouvrir(fluc_conc, nom3) en ecriture
   ouvrir(flux, nom1) en lecture
   si (flux) alors
      lire(donnee) dans flux
      tant que non fini(flux) faire
         ecrire(donnee, " ") dans flux_conc
         lire(donnee) dans flux
      fin tant que
      fermer flux
   fin si
   ouvrir(flux, nom2) en lecture
   si (flux) alors
      lire(donnee) dans flux
      tant que non fini(flux) faire
         ecrire(donnee, " ") dans flux_conc
```

```
            lire(donnee) dans flux
        fin tant que
        fermer flux
    fin si
    fermer flux_conc
fin
```

Exercice 4.4 (Calculs)

Écrire un algorithme qui calcule et affiche la moyenne, la valeur minimale et la valeur maximale d'une série d'entiers notés dans un fichier dont le nom est choisi par l'utilisateur.

▽ **Correction**

Exercice pouvant être soumis pour l'évaluation ;

Analyse :

- choix du nom du fichier => variable nom (chaîne de caractères)
- ouverture du fichier en lecture=> variable $flux$ (flux_fichier)
- tester si le fichier existe
- lecture d'une valeur => variable val (entier)
- initialisation de min (entier) à val
- initialisation de max (entier) à val
- initialisation de nb (entier) à 0
- initialisation de $somme$ (entier) à 0
- répétitive
  - Arrêt : échec de lecture
  - Traitement
    - mises à jour de min, max, nb, $somme$
    - lecture d'une valeur => variable val (entier)
- affichage de la moyenne, du min et du max s'il y avait au moins une valeur dans le fichier
- fermeture du fichier

```
lexique
    chaine nom
    flux_fichier flux
    entier somme, nb, min, max, val;
debut
    ecrire "Donner le nom du fichier de donnees : "
    lire nom

    ouvrir(flux, nom) en lecture
    si (flux) alors
        lire(val) dans flux
        min ← val
        max ← val
        somme ← 0
        nb ← 0
        tant que non fini(flux) faire
            si (val < min) alors
                min ← val
            fin si
            si (val > max) alors
                max ← val
            fin si
            somme ← somme + val
            nb ← nb + 1
            lire(val) dans flux
```

```
        fin tant que
        fermer flux
        si (nb > 0) alors
            ecrire("valeur minimale : ", min)
            ecrire("valeur maximale : ", max)
            ecrire("moyenne : ", somme / nb)
        sinon
            ecrire ("aucune valeur dans le fichier", nom)
        fin si
    sinon
        ecrire("Le fichier ", nom, " n'existe pas.")
    fin si
fin
```

Exercice 4.5 (Fusion de fichiers)

Deux fichiers nommés "liste1.txt" et "liste2.txt" contiennent chacun une séquence de mots non accentués triés par ordre alphabétique.

1. Écrire un algorithme qui affiche la liste triée par ordre alphabétique des mots issus de ces deux fichiers.
2. Simuler son fonctionnement pour les fichiers :
   "liste1.txt" = <"abeille" ; "chameau" "girafe" ; "panda">
   "liste2.txt" = <"chenille" ; "papillon" ; "saumon" ; "sauterelle">

▽ **Correction**

Cet exercice est assez difficile et il faudra peut-être mettre les étudiants sur la voie avec quelques explications, schémas et exemples.

```
variables
    flux_fichier flux1, flux2
    chaine nom1, nom2
debut
    ouvrir flux1, "liste1.txt" en lecture
    ouvrir flux2, "liste2.txt" en lecture
    si (flux1 et flux2) alors
        // initialisation des noms avec les
        // premieres donnees de chaque fichier
        lire nom1 dans flux1
        lire nom2 dans flux2
        // fusion des fichiers tant qu'ils sont non vides
        tant que (non fini(flux1) et non fini(flux2)) faire
            // affichage du nom le plus petit
            // et chargement de la donnee suivante
            si (nom1<nom2) alors
                ecrire nom1
                lire nom1 dans flux1
            sinon
                ecrire nom2
                lire nom2 dans flux2
            fin si
        fin tant que

        // une fois l'un des fichiers epuise, reste a vider l'autre
        // l'une des 2 repetitives sera effectuee 0 fois
        tant que non fini(flux1) faire
            lire nom1 dans flux1
```

```
            ecrire nom1
        fin tant que
        tant que non fini(flux2) faire
            lire nom2 dans flux2
            ecrire nom2
        fin tant que

        fermer flux1
        fermer flux2
      sinon
        si (flux1) alors
            fermer flux1
        sinon
            ecrire("Le fichier liste1.txt n'existe pas")
        fin si
        si (flux2) alors
            fermer flux2
        sinon
            ecrire("Le fichier liste2.txt n'existe pas")
        fin si
    fin si
fin
```

Exercice 4.6 (Histogramme)

Écrire un algorithme qui affiche (à l'écran ou dans un fichier) l'histogramme d'une série de notes entières comprises entre 0 et 10 stockées dans un fichier nommé "notes.txt". Chaque barre de l'histogramme est dessinée à l'aide d'étoiles.

1. L'histogramme est horizontal.
2. L'histogramme est vertical.

Exemple (avec des notes allant de 0 à 5) : "notes.txt" = <4 ; 0 ; 2 ; 3 ; 3 ; 4 ; 5 ; 3>

Histogramme horizontal

```
0  *
1
2  *
3  * * *
4  * *
5  *
```

Histogramme vertical

```
            *
        *   *
*       *   *   *
0   1   2   3   4   5
```

▽ Correction

Exercice difficile pour occuper les étudiants mordus.

```
constante
   N = 11
type
  tab_freq = tableau de N entier

//-------------------------------------------------------------------------
// calcule les frequences des notes issues du fichier nom
procedure calcul_histo(d chaine nom, m tab_freq table)
lexique
   flux_fichier flux
   entier  i, note
```

```
debut
   // comptage des frequences des valeurs
   ouvrir(flux, nom) en lecture
   pour i de 1 a N faire
      frequences[i] ← 0
   fin pour
   si (flux) alors
      lire(note) dans flux
      tant que non fini(flux) faire
         frequences[note] ← frequences[note] + 1
         lire(note) dans flux
     fin tant que
     fermer(flux)
  fin si
fin
//-------------------------------------------------------------------------
// affichage nb etoiles et retour a la ligne
procedure affiche_etoiles(d entier nb)
lexique
   entier i
debut
   pour i de 1 a nb faire
      ecrire("* ")
   fin pour
   ecrire("cr")  // retour a la ligne
fin
//-------------------------------------------------------------------------
// affichage histogramme horizontal
procedure affiche_histo_horizontal(d tab_freq  frequences)
lexique
   entier i
debut
   pour i de 1 a N faire
      ecrire(i, "  ")
      si (i < 10) alors
         ecrire ("   ") // 3 espaces
      sinon
         ecrire ("  ") // 2 espaces
      fin si
      affiche_etoiles(frequences[i])
   fin pour
fin
//-------------------------------------------------------------------------
// affichage histogramme vertical
procedure affiche_histo_vertical(d tab_freq  frequences)
lexique
   entier i, j,  freq_max
debut
   freq_max  ← frequences[0]
   pour i de 1 a N faire
      si (frequences[i] > freq_max) alors
          freq_max ← frequences[note]
      fin si
   fin pour

   pour j de  freq_max a 1 par pas de -1 faire
      pour i de 1 a N faire
         si (frequences[i] ≥ j) alors
            ecrire("*   ") // 2 espaces
```

```
            sinon
               ecrire("    ") // 3 espaces
            fin si
         fin pour
         ecrire("cr")  // retour a la ligne
      fin pour
      pour i de 0 a 9 faire
         ecrire("i  ") // 2 espaces
      fin pour
      pour i de 10 a N faire
         ecrire("i ") // 1 espace
      fin pour
fin
//-----------------------------------------------------------------------

lexique
   chaine nom
   tab_freq frequences
debut
   ecrire "Donner le nom du fichier de donnees : "
   lire nom
   calcul_histo(nom, frequences)
   affiche_histo_horizontal(frequences)
   affiche_histo_vertical(frequences)
fin
```

Partie TP (2 séances)

1 Téléchargement de deux programmes

Télécharger les programmes `lecture_fichier.cpp` et `ecriture_fichier.cpp` qui sont la transcription des algorithmes présentés en cours. Les étudier, tester leur fonctionnement.

Vous poueez réutiilser ces programmes pour vus aider à transcrire des algorithmes de la partie TD (question 3).

2 Transcription

Transcrire quelques algorithmes de la partie TD de cette feuille en programmes C++.

3 Lecture

Télécharger et étudier le document `fluxCpp.pdf` pour approfondir vos connaissances sur les fichiers.