Première partie du TP3 (C++) – INF 2170 gr 10

Équipe G 16

Martin Valiquette (VALM13037400) & Olivier Viera (VIEP12058605)

14 avril 2009

tp3.cpp

```
#include <iostream>
#include <fstream>
#include "tp3.h"

#define FILENAME "fichier.txt" // Fichier a utiliser

using namespace std;



int main() {
	char * mot;
	char motc;
	int fsize, i;
	PtrNoeud premier = new NoeudArbre;
	if (!verifierFichier((char *)FILENAME, fsize)) {
		cout << "Fichier invalide" << endl;
		fEntree.close();
		return 0;
	}
	if (fsize > LONGUEURTAMPON)
		cout << "Le fichier contient plus de caracteres que le maximum (" << LONGUEURTAMPON << ")" << endl;
	ProchainMot(mot);
	do {
		premier->mot = mot;
		premier->compte = 1;
		premier->droite = NULL;
		premier->gauche = NULL;
		motc = mot[0];
	}
	while (motc == 0 && ProchainMot(mot) != -1);

	while((i = ProchainMot(mot)) != 1) {
		motc = mot[0];
		if ((int)motc != 0)
			Inserer(mot, premier);
	}
	Afficher(premier);

	return 0;
}
```

```
// Cette methode verifie que le fichier existe, qu'il contienne des caractere valide et qu'il a un taille > 0
bool verifierFichier(char * fichier, int &size) {
	bool found = false;
	unsigned char carac;

	fEntree.open(fichier, ios::in);
	fEntree.seekg(0, ios_base::end);
	size = fEntree.tellg();
	fEntree.seekg(0, ios_base::beg);

	while (fEntree.tellg() < size && !found)  {
		carac = fEntree.get();
		bool valide = (carac >= 'A' && carac <= 'Z')||
			(carac >= 'a' && carac <= 'z')||carac =='-'||
			(carac >= '0' && carac <= '9');
		if (valide) { // Il y a minimum UN caractere valide
			found = true;
			fEntree.seekg(0, ios_base::beg);
		}
	}
	return found;


}
int Comparaison(char *Mot1, char *Mot2) // Comparer deux mots du tampon global.
{
	int i = 0;
	int j = 0;
	while(true){
		if(Mot1[i] != Mot2[j])		// non identiques
			if(Mot1[i] < Mot2[j])
				return -1;		// plus petit
			else
				return 1;		// plus grand
		else if(Mot1[i] == FINCHAINE)
			return 0;			// identiques
		i++; j++;
	}
} // Comparaison;

bool Chercher(char * Mot, PtrNoeud Arbre) // Chercher un élément dans l'arbre
{
	int compar;
	if(Arbre == NULL)
		return false;
	else{
		compar = Comparaison(Mot, Arbre->mot);
		if(compar == -1)
			return Chercher(Mot, Arbre->gauche);
		else if(compar == 1)
			return Chercher(Mot, Arbre->droite);
		else if(compar == 0)
```

```
                    return true;
      }
      return false;
}// Chercher



void Inserer(char * Mot, PtrNoeud & Arbre) // insérer un élément dans l'arbre
{
      if(Arbre == NULL){
            Arbre = new NoeudArbre;    // nouveau noeud rattaché à l'arbre
            Arbre->mot = Mot;          // adresse du mot
            Arbre->compte = 1;         // première occurrence
            Arbre->gauche = NULL;      // pas de descendant gauche
            Arbre->droite = NULL;      // pas de descendant droit
      }
      else
            if(Comparaison(Mot, Arbre->mot) == -1) // + petit, à gauche
                  Inserer(Mot, Arbre->gauche);
            else
                  if(Comparaison(Mot, Arbre->mot) == 1)// + grand, à droite
                        Inserer(Mot, Arbre->droite);
                  else
                        Arbre->compte++; // déjà là, mise à jour compteur
}// Inserer



int ProchainMot(char * &Mot)
{
        int fdf;
      bool nonEspace = false;
        Mot = &Tampon[IndexTamp];
        bool valide;
        unsigned char carac;
        do
        {
            carac = fEntree.get();

                if(!fEntree.eof()){
                        fdf = 0;
                        valide = (carac >= 'A' && carac <= 'Z')||
                                (carac >= 'a' && carac <= 'z')||carac =='-'||
                          (carac >= '0' && carac <= '9');

                        if(valide){
                                Tampon[IndexTamp] = carac;
                                IndexTamp++;
                        }else{
                                Tampon[IndexTamp] = 0;
                                IndexTamp++;
                        }
```

```
                }else{
                        Tampon[IndexTamp] = 0;
                        IndexTamp++;
                        fdf = 1;
                }
        }
        while(valide && fdf == 0);

        return fdf;
}



void Afficher(PtrNoeud Arbre) // traversée infixe
{
      if(Arbre != NULL){
            Afficher(Arbre->gauche);
            cout << Arbre->mot << " utilisé " << Arbre->compte << " fois." << endl;
            Afficher(Arbre->droite);
      }
}// Afficher
```

tp3.h

```
using namespace std;

const int LONGUEURTAMPON = 20000;
const char FINCHAINE = 0;
char Tampon[LONGUEURTAMPON]; // tampon de mots global
int IndexTamp = 0;           // index global du tampon global

ifstream fEntree;            // fichier d'entrée global

struct NoeudArbre;
typedef NoeudArbre *PtrNoeud;
struct NoeudArbre{char * mot;         // adresse du mot
                  int compte;         // nombre d'apparitions du mot
                  PtrNoeud gauche;    // pointeur gauche
                  PtrNoeud droite;}; // pointeur droit

int ProchainMot(char * &Mot);
void Inserer(char * Mot, PtrNoeud & Arbre);
bool Chercher(char * Mot, PtrNoeud Arbre);
int Comparaison(char *Mot1, char *Mot2);
void Afficher(PtrNoeud Arbre);
bool verifierFichier(char * fichier, int &size);
```

Première partie du TP3 (C++) – INF 2170 gr 10

Équipe G 16
Martin Valiquette & Olivier Viera
14 avril 2009

Stratégie de vérification

Validation de données :

Le ficher lu est d'un maximum de 20000 caractères. Le programme divise le texte en mots, ces mots peuvent comprendre des accents (pour la version Assembleur seulement).

Divers cas de donnée :

1. **Fichier au-delà de longueur maximale**
   Condition : Un fichier contenant plus de 20 000 caractères est analysé.
   Résultat : Le programme dira que le fichier contient plus que le maximum, mais mais il continue.
2. **Fichier vide**
   Condition : Un fichier de 0 caractères est analysé.
   Résultat : Une erreur est affichée et le programme s'arrête.
3. **Cas de routine**
   Condition : Un fichier de bonne longueur est analysé.
   Résultat : Les mots sont placés dans l'arbre correctement en ordre alphabétique et le nombre d'occurrences des mots est augmenté correctement.
4. **Bonne division des mots**
   Condition : Du texte avec des accents est analysé.
   Résultat : Les mots sont correctement extraits du texte **(Version Assembleur seulement)**
5. **Bonne division des mots**
   Condition : Des chiffres sont inclus dans le texte.
   Résultat : Les chiffres sont inclus dans l'arbre.

Preuves de tests

Cas 1: Exécution normale.

Contenu du fichier:
abc def ghi jkl mno pqr stu vwx yz
Affichage:
abc utilisé 1 fois.
def utilisé 1 fois.
ghi utilisé 1 fois.
jkl utilisé 1 fois.
mno utilisé 1 fois.
pqr utilisé 1 fois.
stu utilisé 1 fois.
vwx utilisé 1 fois.
yz utilisé 1 fois.

Cas 2: Exécution avec un **fichier vide**.

Contenu du fichier:
rien
Affichage:
Fichier invalide

Cas 3: Exécution avec un **fichier inexistant**.

Contenu du fichier:
N/D
Affichage:
Fichier invalide

Cas 4: Éxécution avec un fichier contenant **que** des numéros.

Contenu du fichier:
2394823984 2984 2398472 93084729 837498 59378935
Affichage:
2394823984 utilisé 1 fois.
2398472 utilisé 1 fois.
2984 utilisé 1 fois.
59378935 utilisé 1 fois.
837498 utilisé 1 fois.
93084729 utilisé 1 fois.

Cas 5: Éxécution avec un fichier contenant **des chiffres et des lettres**.

Contenu du fichier:
bonjour832 salut82 284bonjour2 43allo889 284bonjour823 9999999allo99 81
Affichage:
allo utilisé 2 fois.
bonjour utilisé 3 fois.
salut utilisé 1 fois.

Cas 6: Exécution avec un fichier excédant la longueur maximum.

Contenu du fichier:
Pour des raisons de propreté, on ne montrera pas le contenu du fichier.
Affichage:
Le fichier contient plus 20000 caractères.
(Contenu de l'arbre selon les 20 000 premiers caractères)