RAPPORT PROJET SDA 2021

1ER FEVRIER

AKHAYAR_Mahir_104
HADDIOUI_Sélim_104

Université de Paris

Table of Contents

I. Présentation de l'application 2
a. Rôle fonctionnel 2
b. Graphe de dépendance 3
c. Structure de données. 3
II. Organisation des tests 3
a. Exercice 1 3
b. Exercice 2 4
c. Exercice 3 5
d. Exercice 4 5
e. Exercice 5 6
f. Exercice 6 6
III. Bilan du projet 8
IV. Code source 9

I. Présentation de l'application

a. Rôle fonctionnel

Notre application a été conçu dans un but de divertissement. Celle-ci permet à des individus de jouer un tour de BOOGLE.

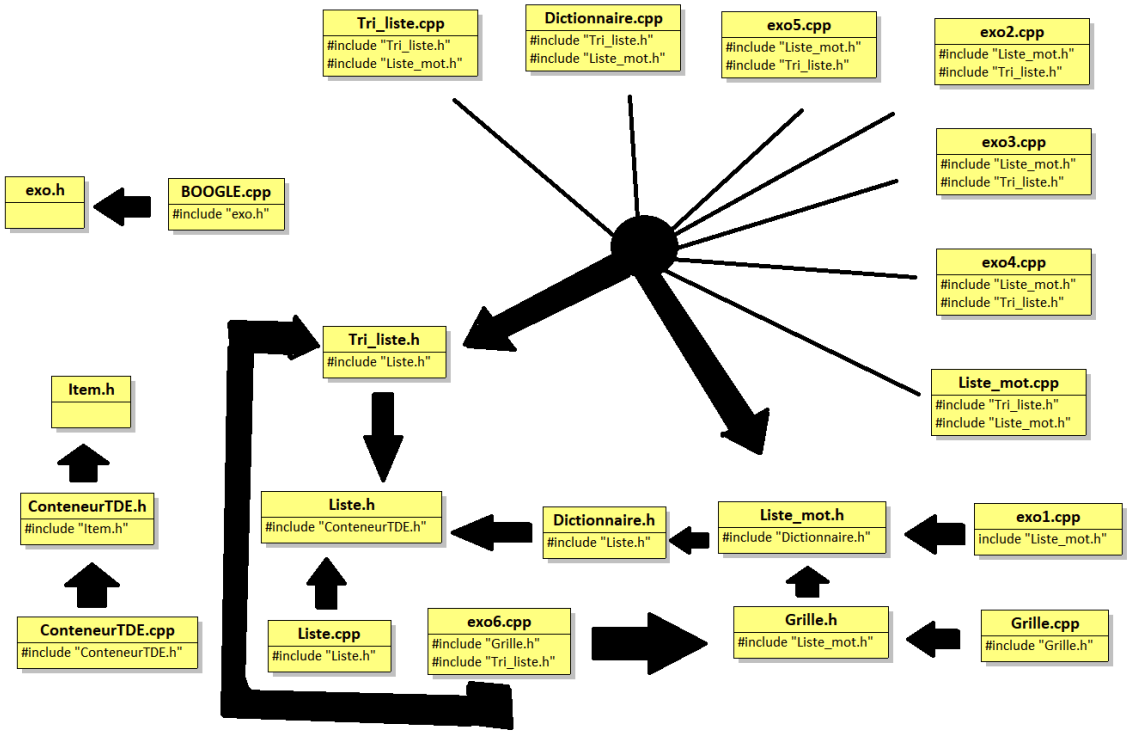
Pour cela il faut saisir le plateau, la liste des mots trouvés par chacun des joueurs ainsi que le dictionnaire utilisé pour valider les mots.

Le programme retournera le nombre de points de chaque joueur ainsi que le nombre maximum de points possible pour le tour.

Voici ci-dessous un tableau reprenant le barème du jeu.

Longueur du mot	Nombre de points
$\leq$ 2 lettres	0 point
3 ou 4 lettres	1 point
5 lettres	2 points
6 lettres	3 points
7 lettres	5 points
$\geq$ 8 lettres	11 points

b. Graphe de dépendance



c. Structure de données.

Comme structure de données nous avons opté pour la liste car nous voulions pouvoir modifier celles-ci de l'intérieur ce qui n'est pas possible avec une pile où une file qui ne nous permet de supprimer ou d'ajouter un item qu'au sommet pour la pile et respectivement à l'entrée et la sortie pour la file.

Celle-ci nous a permis de mieux gérer nos données et d'optimiser notre programme dans le sens où nous n'avions que très rarement à récrire entièrement une liste pour la modifier ce qui n'aurait pas été possible avec une file ou une pile.

II. Organisation des tests

L'organisation des tests de l'application a été fait en 6 exercices. Pour tous les tests la capture d'écran blanche correspond au résultat attendu et la capture d'écran noire correspond aux résultats obtenus.

a. Exercice 1

L'objectif du premier exercice est de saisir une liste et de récupérer la somme des points de chaque mot de celle-ci. Les tests que nous avons effectués est de saisir une liste de mots et de

l'afficher afin de vérifier si celle-ci a bien été enregistrée. Une fois ce test réussi nous avons pu passer à la consultation de la liste et le comptage des points. Ce dernier a été fait en deux étapes. La première était de vérifier que notre programme respectait bien la valeur de points de chaque mot. La seconde était de vérifier si la somme de chaque valeur était correcte. Pour vérifier l'ensemble nous avons saisi une liste de mots pour laquelle nous connaissions la valeur finale et si nous obtenions cette même valeur le test était donc réussi, celui-ci a marché, nous attendions 71 comme résultat et nous l'avons obtenu.

b. Exercice 2

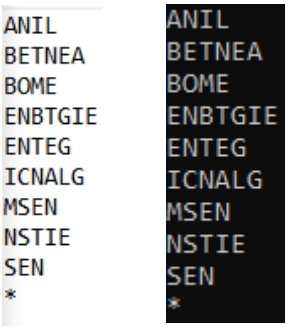
L'objectif du second exercice est de trier une liste saisit au préalable sous forme canonique. Pour tester l'exercice 2 nous avons saisi dans la liste, des mots tels que : « AAA », « AAA », « DDD », « BBB », L'objectif est que les répétitions soient supprimées et les mots triés par ordre alphabétique. Grâce à cette saisi et d'autres similaires un peu plus longues nous savions facilement quel résultat attendre et celui-ci à finit par être concluant. Le dernier test fut sur une longue liste de mots et celle-ci a bien fini trié en voici la sortie :

```
AMENAT
ANIL
BEIGNE
BETNEA
BOME
CLIGNAS
CLIGNES
EMBUE
ENBTGIE
ENTEG
GITANS
ICNALG
IL
ILE
LIENS
LINS
LUE
LUI
MSEN
NIE
NSTIE
OBEIT
SEN
TANGUE
*
```

```
AMENAT
ANIL
BEIGNE
BETNEA
BOME
CLIGNAS
CLIGNES
EMBUE
ENBTGIE
ENTEG
GITANS
ICNALG
IL
ILE
LIENS
LINS
LUE
LUI
MSEN
NIE
NSTIE
OBEIT
SEN
TANGUE
*
```

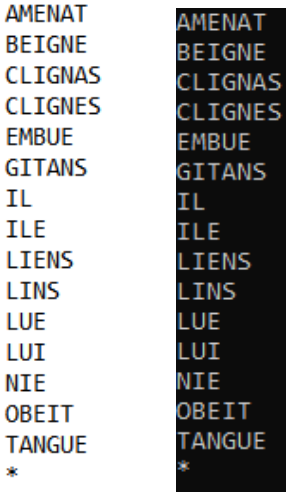
c. Exercice 3

L'objectif de l'exercice 3 est de saisir deux listes de mots, les trier et afficher à l'écran les mots de la seconde liste n'apparaissant pas dans la première. Nous avons effectué le même genre de test que pour l'exercice 2 à l'aide de mots tels que AAA ou CCC pour comprendre les erreurs de notre programme et les corriger. Le test final a été effectué sur deux longues listes de mots existants dont nous connaissions la sortie attendue et nous l'avons comparé avec la sortie obtenue. Ce test final a été concluant en voici la preuve :



d. Exercice 4

L'objectif de l'exercice 4 est de lire deux listes de mots saisies et d'afficher les mots de la seconde liste n'étant pas dans la première sous forme canonique. Nous avons procédé exactement tel pour l'exercice 3, en voici le résultat du test final :



e. Exercice 5

L'objectif de l'exercice 5 est de saisir deux listes de mots et d'obtenir les mots apparaissant dans au moins deux listes. Nous avons procédé exactement comme pour les exercices 3 et 4. Voici le résultat du test final :

```
BEIGNE      BEIGNE
EMBUE       EMBUE
ENBTGIE     ENBTGIE
ILE         ILE
LINS        LINS
MSEN        MSEN
NSTIE       NSTIE
OBEIT       OBEIT
SEN         SEN
TANGUE      TANGUE
*           *
```

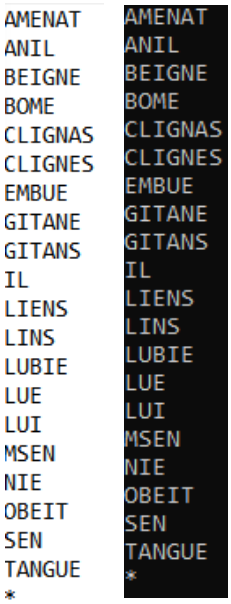
f. Exercice 6

L'objectif de l'exercice 6 est de saisir un plateau suivi de listes de mots et d'afficher les mots présents sur le plateau. Pour réaliser les tests nous avons utilisé un document WORD afin de schématiser notre plateau, le chemin du programme et l'affichage attendu. En voici un aperçu :

A	Z	Z	Z
Z	B	Z	Z
Z	E	I	O
Z	Z	Z	U

En saisissant le plateau AZZZ ZBZZ ZEIO ZZZU et le mot ABEIOU nous avons voulu tester les différentes possibilités de notre programme afin qu'il puisse chercher des lettres adjacentes tout

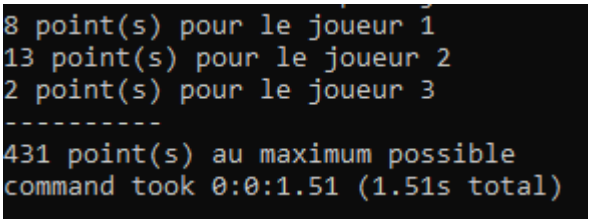
autour de lui, si une direction ne fonctionnait pas alors nous pouvions repérer le problème. Le test final à été de saisir une longue liste et de comparer l'affichage attendu à celui reçut en voici la preuve :



Test final

Pour le test final, nous avons simulé un tour de BOOGLE avec les résultats connus et espérant obtenir les mêmes, de plus nous avons testé la rapidité d'exécution de notre programme qui a été concluante car ne dépassant pas les 1,52 seconde, en voici le test :

```
8 point(s) pour le joueur 1
13 point(s) pour le joueur 2
2 point(s) pour le joueur 3
----------
431 point(s) au maximum possible
```



III. Bilan du projet

Durant ce projet, nous avons eu du mal à se lancer. En effet nous avons mis du temps car nous cherchions quel structure permettrait d'avoir une meilleure approche des 6 exercices.

Après avoir choisi la liste comme structure de données nous avons commencé l'exercice 1. Cet exercice ne nous a pas posé de soucis et nous l'avons réussi rapidement. À la suite de cela le deuxième exercice à été plus compliqué car nous avions eu du mal à établir une fonction qui trie par ordre alphabétique. Cependant en découvrant "qsort" nous avons remédié à ce soucis et avons réussi cette étape.

L'exercice 3 et 4 se ressemblait énormément ce qui nous a permis d'être plus rapide et efficace. De plus l'élaboration des fonctions n'était pas vraiment différente, ce qui nous a permis de le réussir simplement. L'exercice 5 nous a pris plus de temps, nous avons eu du mal à élaborer une liste de listes de mots, nous avons pensé au début à créer une autre liste contenant les dernières. Cependant cela a posé une multitude de problèmes.

C'est pour cela qu'après mure réflexion nous avons décidé de créer une structure dictionnaire. Cette structure nous a facilité la tâche et a conclu l'exercice 5. Puis pour l'exercice 6 la difficulté était le positionnement des lettres, c'est-à-dire comment situer une lettre par rapport à une autre.

Pour cela nous avons opté sur un style 2 dimensions qui permettait à la lettre de comprendre s'il y avait des voisins ou non en analysant l'abscisse et l'ordonnée de chacune. Enfin, l'ultime ne nous a pas posés de problèmes.

Ce projet, nous a appris à :

- utiliser des structures de données
- utiliser des fichiers d'en-tête
- s'améliorer sur l'allocation dynamique de tableau
- mieux manier le c et c++
- utiliser scrupuleusement les inclusions de bibliothèques
- mieux gérer la structuration de notre code

Ayant fait un projet similaire en IAP, nous avons appris de nos erreurs et nous sommes plutôt satisfait du travail que nous vous rendons. Au moment de vous le rendre nous ne trouvons aucun axe d'amélioration possible bien qu'il en existe sûrement.

IV. Code source

BOOGLE.cpp

```
/**
* @file BOOGLE.cpp
* @brief Jeu boogle
* @author Groupe 104 : AKHAYAR
Mahir et HADDIOUI Sélim
* @version 1.0 05/01/2020
*/

#include <iostream>

#include "exo.h"

using namespace std;


int main() {
	int num;
	cin >> num;
	switch (num) {
	case 1:
		exo1(); break;
	case 2:
		exo2(); break;
	case 3:
		exo3(); break;
	case 4:
		exo4(); break;
	case 5:
		exo5(); break;
	case 6:
		exo6(); break;
	}
}
```

exo.h

```
#pragma once
/**
* @file exo.h
* @brief Jeu boogle
* @author Groupe 104 : AKHAYAR Mahir
et HADDIOUI Sélim
*/

// @brief Compteur de points d'une
liste
void exo1();

// @brief Affichage sous forme
canonique d'une liste
void exo2();

// @brief Obtenir les mots dit
valides
void exo3();

// @brief Obtenir les mots étant
dans le dictionnaire
void exo4();

// @brief Obtenir les mots écrit par
au moins deux joueurs
void exo5();

// @brief Recherche des mots
apparaissant dans le plateau
void exo6();
```

exo1.cpp

```
/**
* @file exo1.cpp
* @brief Compteur de points d'une
liste
* @author Groupe 104 : AKHAYAR
Mahir et HADDIOUI Sélim
*/

#include "Liste_mot.h"

void exo1() {

	Liste l;
	saisir_liste(l);
	point(l);
	detruire(l);
}
```

exo2.cpp

```
/**
* @file exo2.cpp
* @brief Affichage sous forme
canonique d'une liste
* @author Groupe 104 : AKHAYAR
Mahir et HADDIOUI Sélim
*/

#include "Liste_mot.h"
#include "Tri_liste.h"

void exo2() {
	Liste l;
	saisir_liste_trie(l);
	affichage(l);
	detruire(l);
}
```

exo3.cpp

```
/**
* @file exo3.cpp
* @brief Obtenir les mots dit
valides
* @author Groupe 104 : AKHAYAR
Mahir et HADDIOUI Sélim
*/

#include "Liste_mot.h"
#include "Tri_liste.h"

void exo3() {
	Liste la;
	Liste lb;
	saisir_liste_trie(la);
	saisir_liste_trie(lb);
	suppr_rep_entre_listes(la,
lb);
	affichage(lb);
	detruire(la);
	detruire(lb);
}
```

exo4.cpp

```
/**
* @file exo4.cpp
* @brief Obtenir les mots étant
dans le dictionnaire
* @author Groupe 104 : AKHAYAR
Mahir et HADDIOUI Sélim
*/

#include "Liste_mot.h"
#include "Tri_liste.h"

void exo4() {
	Liste la;
	Liste lb;
	saisir_liste_trie(la);
	saisir_liste_trie(lb);
	rep_entre_listes(la, lb);
	affichage(lb);
	detruire(la);
	detruire(lb);
}
```

exo5.cpp

```
/**
* @file exo5.cpp
* @brief Obtenir les mots écrit par
au moins deux joueurs
* @author Groupe 104 : AKHAYAR
Mahir et HADDIOUI Sélim
*/

#include "Liste_mot.h"
#include "Tri_liste.h"

void exo5() {
	Dictionnaire dico;
	Liste l;
	remplir_dico(dico);
	enreg_mot_recurrent(dico,
l);
	affichage(l);
	detruire(l);
}
```

exo6.cpp

```
/**
* @file exo6.cpp
* @brief Recherche des mots
apparaissant dans le plateau
* @author Groupe 104 : AKHAYAR
Mahir et HADDIOUI Sélim
*/

#include "Grille.h"
#include "Tri_liste.h"

void exo6() {
	plateau plat;
	Liste l;
	saisir_grille(plat);
	saisir_liste_trie(l);
	afficher_mot_correct(plat,
l);
	detruire(l);
}
```

ConteneurTDE.h

```h
#pragma once

/**
 * @file ConteneurTDE.h
 * @brief Composant d'un conteneur d'items de capacité extensible.
 */

#include "Item.h"


 /** @brief Conteneur d'items alloués en mémoire dynamique
  *  de capacité extensible suivant un pas d'extension.
  */
struct ConteneurTDE {
	/// Capacité du conteneur (>0).
	unsigned int capacite;
	/// Pas d'extension du conteneur (>0).
	unsigned int pasExtension;
	/// Conteneur alloué en mémoire dynamique.
	Item* tab;
};

/**
 * @brief Initialise un conteneur d'items.
 * Allocation en mémoire dynamique du conteneur d'items
 * de capacité (capa) extensible par pas d'extension (p).
 * @see detruire, pour sa désallocation en fin d'utilisation.
 * @param[out] c Le conteneur d'items.
 * @param [in] capa Capacité du conteneur.
 * @param [in] p Pas d'extension de capacité.
 * @pre capa > 0 et p > 0.
 */
void initialiser(ConteneurTDE& c, unsigned int capa, unsigned int p);

/**
 * @brief Désalloue un conteneur d'items en mémoire dynamique.
 * @see initialiser.
 * @param[in,out] c Le conteneur d'items.
 */
void detruire(ConteneurTDE& c);

/**
 * @brief Lecture d'un item d'un conteneur d'items.
 * @param[in] c Le conteneur d'items.
 * @param[in] i La position de l'item dans le conteneur.
 * @return L'item à la position i.
 * @pre i < c.capacite
 */
Item lire(const ConteneurTDE& c, unsigned int i);

/**
 * @brief Ecrire un item dans un conteneur d'items.
 * @param[in,out] c Le conteneur d'items.
 * @param[in] i La position où modifier l'item.
 * @param[in] it L'item à écrire.
 */
void ecrire(ConteneurTDE& c, unsigned int i, const Item& it);
```

ConteneurTDE.cpp

```
/**
 * @file ConteneurTDE.cpp
 * @brief Composant de conteneur d'items de capacité extensible
 */

#include <cassert>

#include "ConteneurTDE.h"

void initialiser(ConteneurTDE& c, unsigned int capa, unsigned int p) {
	assert((capa > 0) && (p > 0));
	c.capacite = capa;
	c.pasExtension = p;
	c.tab = new Item[capa];
}

void detruire(ConteneurTDE& c) {
	delete[] c.tab;
	c.tab = NULL;
}

Item lire(const ConteneurTDE& c, unsigned int i) {
	assert(i < c.capacite);
	return c.tab[i];
}

void ecrire(ConteneurTDE& c, unsigned int i, const Item& it) {
	if (i >= c.capacite) {
		unsigned int newTaille = (i + 1) * c.pasExtension;
		Item* newT = new Item[newTaille];
		for (unsigned int i = 0; i < c.capacite; ++i)
			newT[i] = c.tab[i];
		delete[] c.tab;
		c.tab = newT;
		c.capacite = newTaille;
	}
	c.tab[i] = it;
}
```

Item.h

```
#pragma once

/// @brief Type devant être adapter en fonction des besoins de l'application.

typedef char* Item;

/// @brief chaine de caractère.
typedef char* Mot;
```

Liste.h

```
#pragma once

/**
 * @file Liste.h
 * @brief Composant de liste en mémoire dynamique et extensible.
 */

#include "ConteneurTDE.h"



struct Liste {
	/// Conteneur mémorisant les éléments de la liste.
	ConteneurTDE c;
	/// Nombre d'éléments stockés dans la liste.
	unsigned int nb;
	/// Tableau de pointeur de chaine de caracatére
};


/**
 * @brief Initialiser une liste vide, la liste est allouée en mémoire dynamique.
 * @see detruire, la liste est à désallouer en fin d'utilisation.
 * @param[out] l La liste à initialiser.
 * @param[in] capa Capacité de la liste.
 * @param[in] pas Pas d'extension de la liste.
 * @pre capa > 0 et pas > 0.
 */
void initialiser(Liste& l, const unsigned int capa, const unsigned int pas);

/**
 * @brief Désallouer une liste.
 * @see initialiser
 * @param[out] l La liste.
 */
void detruire(Liste& l);

/**
 * @brief Longueur de liste.
 * @param[in] l La liste.
 * @return La longueur de la liste.
 */
unsigned int longueur(const Liste& l);

/**
 * @brief Lire un élément de liste.
 * @param[in] l La liste.
 * @param[in] pos Position de l'élément à lire.
 * @return L'item lu en position pos.
 * @pre 0 <= pos < longueur(l).
 */
Item lire(const Liste& l, const unsigned int pos);
```

```
/**
 * @brief Ecrire un item dans la liste.
 * @param[in,out] l La liste.
 * @param[in] pos Position de l'élément à écrire.
 * @param[in] it L'item.
 * @pre 0 <= pos < longueur(l).
*/
void ecrire(Liste& l, const unsigned int pos, const Item& it);

/**
 * @brief Insérer un élément dans une liste.
 * @param[in,out] l La liste.
 * @param[in] pos La position à laquelle l'élément est inséré.
 * @param[in] it L'élément inséré.
 * @pre 0 <= pos <= longueur(l).
 */
void inserer(Liste& l, const unsigned int pos, const Item& it);

/**
 * @brief Supprimer un élément dans une liste.
 * @param[in,out] l La liste.
 * @param[in] pos La position de l'élément à supprimer.
 * @pre longueur(l) > 0 et 0 <= pos < longueur(l).
 */
void supprimer(Liste& l, const unsigned int pos);

/**
 * @brief Ajoute un item à la fin d'une liste
 * si celle-ci est complete, la rend extensible.
 * @param[in-out] l La liste où est rentrer l'Item.
 * @param[in] it L'item à saisir.
 */
void ajouter_fin(Liste& l, const Item it);
```

Liste.cpp

```cpp
/**
 * @file Liste.cpp
 * @brief Composant de liste en mémoire dynamique et extensible
 */

#include <cassert>
#include <cstring>

#pragma warning ( disable : 4996 )

#include "Liste.h"

void initialiser(Liste& l, const unsigned int capa, const unsigned int pas) {
	assert((capa > 0) && (pas > 0));
	initialiser(l.c, capa, pas);
	l.nb = 0;
}

void detruire(Liste& l) {
	for (unsigned int i = 0; i < l.nb; i++) {
		delete[] l.c.tab[i];
	}
	detruire(l.c);
}

unsigned int longueur(const Liste& l) {
	return l.nb;
}

Item lire(const Liste& l, const unsigned int pos) {
	assert(pos < l.nb);
	return lire(l.c, pos);
}

void ecrire(Liste& l, const unsigned int pos, const Item& it) {
	assert(pos < l.c.capacite);
	ecrire(l.c, pos, it);
}

void inserer(Liste& l, const unsigned int pos, const Item& it) {
	assert(pos <= l.nb);
	for (unsigned int i = l.nb; i > pos; i--) {
		ecrire(l.c, i, lire(l.c, i - 1));
	}
	ecrire(l.c, pos, it);
	l.nb++;
}

void supprimer(Liste& l, const unsigned int pos) {
	assert((l.nb != 0) && (pos < l.nb));
	l.nb--;
	for (unsigned int i = pos; i < l.nb; ++i)
		ecrire(l.c, i, lire(l.c, i + 1));
}
```

```
void ajouter_fin(Liste& l, const Item it)
{
	l.c.tab[l.nb] = new char[30];
	strcpy(l.c.tab[l.nb], it);
	l.nb++;
	if (l.nb >= l.c.capacite) {
		unsigned int newTaille = (l.nb + 1) * l.c.pasExtension;
		Item* newT = new Item[newTaille];
		for (unsigned int i = 0; i < l.c.capacite; ++i)
			newT[i] = l.c.tab[i];
		delete[] l.c.tab;
		l.c.tab = newT;
		l.c.capacite = newTaille;
	}
}
```

Liste_mot.h

```
#pragma once

/**
 * @file Liste_mot.h
 * @brief Composant d'une liste de chaine de caractère de capacité extensible.
 * @author Groupe 104 : AKHAYAR Mahir et HADDIOUI Sélim
 */

#include "Dictionnaire.h"

/**
 * @brief Initialise et permet la saisit d'une liste.
 * @see detruire, pour sa désallocation en fin d'utilisation.
 * @param[in-out] l La liste.
 */
bool saisir_liste(Liste& l);

/**
 * @brief calcul la valeur des points d'une liste et l'affiche.
 * @param[in] l La liste concerné.
 */
void point(const Liste& l);

/**
 * @brief Affiche les mots d'une liste.
 * @param[in] l La liste à afficher.
 */
void affichage(const Liste& l);

/**
 * @brief Enregistre dans la liste les mots présent plus d'une fois dans le dictionnaire
 * et désalloue ce dernier.
 * @param[in-out] l La liste à remplir.
 * @param[in-out] dic Dictionnaire à consulter puis désallouer.
 */
void enreg_mot_recurrent(Dictionnaire& dic, Liste& l);
```

Liste_mot.cpp

```
/**
* @file Liste_mot.cpp
* @brief Gestion des listes de mots
* @author Groupe 104 : AKHAYAR Mahir et HADDIOUI Sélim
*/

#include <iostream>
#include "Tri_liste.h"
#include "Liste_mot.h"

using namespace std;

bool saisir_liste(Liste& l) {
	initialiser(l, 10, 5);                    // liste de 10 mots avec un pas de 5
	char mot[31];
	int i = 0;
	bool a;
	while (1) {
		cin >> mot;
		if (strcmp(mot, "*") == 0) {
			if (i == 0){                  // Si le premier mot de la liste est * on
retourne 1 et on stop la fonction
				a = 1;
				break;
			}
			a = 0;                        // Si ce n'est pas * on retourne 0 et on
stop la fonction
			break;
		}
		ajouter_fin(l, mot);
		i++;
	}
	return a;
}
```

```
void point(const Liste& l) {
	int car;
	int point = 0;
	for (unsigned int x = 0; x < l.nb; x++) {
		car = strlen(lire(l, x));
		switch (car)
		{
		case 0:
		case 1:
		case 2:
			break;
		case 3:
		case 4:
			point += 1;
			break;
		case 5:
			point += 2;
			break;
		case 6:
			point += 3;
			break;
		case 7:
			point += 5;
			break;
		default:
			point += 11;
			break;
		}
	}
	cout << point;
}


void affichage(const Liste& l) {
	for (unsigned int u = 0; u < l.nb; u++) {
		cout << lire(l, u) << endl;
	}
	cout << "*" << endl;
}

void enreg_mot_recurrent(Dictionnaire& dic, Liste& l) {
	initialiser(l, 10, 5);
	for (unsigned int i = 0; i < dic.nb; i++) {
		if (dic.tab[i].repetition > 1)
			ajouter_fin(l, dic.tab[i].mot);
	}
	tri(l);
	detruire_dico(dic);
}
```

Tri_liste.h

```
#pragma once

/**
 * @file Tri_liste.h
 * @brief Fonction de tri de liste.
 * @author Groupe 104 : AKHAYAR Mahir et HADDIOUI Sélim
 */

#include "Liste.h"

/**
 * @brief Initialiser une liste, la saisir et la trier.
 * @see detruire, la liste est à désallouer en fin d'utilisation.
 * @param[in-out] l La liste à initialiser, saisir et trier.
 * @return Un booléen permettant de savoir si la liste est vide
 */
bool saisir_liste_trie(Liste& l);

/**
 * @brief Permet de savoir quel est l'ordre alphabétique de deux mots l'un vis-à-vis de
l'autre.
 * @param[in] A pointeur const vers le mot 1.
 * @param[in] B pointeur const vers le mot 2.
 * @return Un entier indiquant lequel est au dessus de l'autre
 */
int tri_alphabetique(const void* A, const void* B);

/**
 * @brief Tri une liste dans l'ordre alphabétique.
 * @param[in-out] l la liste a trier.
 */
void tri(Liste& l);

/**
 * @brief Supprime les mots de la deuxième liste qui ne sont pas présents dans la
première.
 * @param[in] la liste à consulter.
 * @param[in-out] lb la liste à modifier.
 */
void rep_entre_listes(const Liste& la, Liste& lb);

/**
 * @brief Supprimer les mots redondant à l'intérieur d'une liste.
 * @param[in-out] l la liste a modifier.
 */
void suppr_repetition(Liste& l);

/**
 * @brief Supprime les mots de la deuxième liste qui sont présents dans la première.
 * @param[in] la liste à consulter.
 * @param[in-out] lb la liste à modifier.
 */
void suppr_rep_entre_listes(const Liste& la, Liste& lb);
```

Tri_liste.cpp

```
/**
* @file Tri_liste.cpp
* @brief Fonction de tri d'une liste
* @author Groupe 104 : AKHAYAR Mahir et HADDIOUI Sélim
* @version 1.0 05/01/2020
*/

#include <cstring>
#include <cstdlib>
#include "Tri_liste.h"
#include "Liste_mot.h"

bool saisir_liste_trie(Liste& l) {
	bool a = saisir_liste(l);
	tri(l);
	return a;
}

int tri_alphabetique(const void* A, const void* B)
{
	const Item* a = (const Item*)A;
	const Item* b = (const Item*)B;
	return strcmp(*a, *b);
}

void tri(Liste& l) {
	qsort(l.c.tab, l.nb, sizeof(Item), tri_alphabetique);
	suppr_repetition(l);
}

void rep_entre_listes(const Liste& la, Liste& lb) {
	int a = 1;
	for (unsigned int i = 0; i < lb.nb; i++) {
		for (unsigned int j = 0; j < la.nb; j++) {
			a = strcmp(lb.c.tab[i], la.c.tab[j]);
			if (a != 0) {
				if (j == la.nb - 1) {
					supprimer(lb, i);
					i--;
				}
			}
			else
				break;
		}
	}
}

void suppr_repetition(Liste& l) {
	for (unsigned int i = 0; i < l.nb; i++) {
		if (i + 1 == l.nb)
			break;
		while (strcmp(l.c.tab[i], l.c.tab[i + 1]) == 0) {
			supprimer(l, i + 1);
			if (i + 1 == l.nb)
				return;
		}
	}
}
```

```
void suppr_rep_entre_listes(const Liste& la, Liste& lb) {
	int a = 1;
	for (unsigned int i = 0; i < lb.nb; i++) {
		for (unsigned int j = 0; j < la.nb; j++) {
			a = strcmp(lb.c.tab[i], la.c.tab[j]);
			if (a == 0)
			{
				supprimer(lb, i);
			}

		}
	}
}
```

Grille.h

```
#pragma once

/**
 * @file Grille.h
 * @brief Composant du plateau de BOOGLE
 * @author Groupe 104 : AKHAYAR Mahir et HADDIOUI Sélim
 */

#include "Liste_mot.h"

// Nombre de lettre dans le plateau
#define CASE 16

// Nombre de ligne du plateau
#define Mot_Plateau 4

 /** @brief Position d'une lettre dans le plateau
  */
struct Position {
	unsigned int x;	// abscisse
	unsigned int y; // ordonnée
};

/** @brief Lettre du plateau
*/
struct Lettre {
	char lettre;
	Position pos;
	bool visite; // si vrai alors la case a déjà était consulté
};

/** @brief Plateau
*/
typedef Lettre plateau[CASE + 1];
```

```
/**
 * @brief Recherche un mot dans le plateau.
 * @param[in-out] plat Le plateau à consulter,
 * les lettres du plateau sont mis à jour au moment de la visite.
 * @param[in] mot Le mot à rechercher
 * @return Un booléen permettant de savoir si le mot est présent dans
 * le plateau.
 */
bool recherche(plateau& plat, const Mot mot);

/**
 * @brief Indique si un lettre est adjacente à une autre
 * @param[in] pos La position de la lettre 1.
 * @param[in] posTest La position de la lettre 2.
 * @return Un booléen permettant de savoir si les lettres sont adjacentes entre
 * elles.
 */
bool est_adjacent(const Position& pos, const Position& posTest);

/**
 * @brief Saisir le plateau et enregistré la position des lettres
 * de celui-ci.
 * @param[in-out] plat Le plateau à modifié.
 */
void saisir_grille(plateau& plat);

/**
 * @brief Indique la position en 2D d'une lettre
 * @param[in] pos La position en 1D de la lettre.
 * @return La position en 2D de la lettre.
 */
Position enreg_pos(const unsigned int pos);

/**
 * @brief Recherche si une lettre adjacente correspond à la lettre recherchée.
 * @param[in-out] plat Le plateau à consulter,
 * les lettres du plateau sont mis à jour au moment de la visite.
 * @param[in] mot Le mot à rechercher
 * @param[in] posi Position de la lettre dans le mot à rechercher
 * @param[in] coord Position de la lettre dans le mot à rechercher
 * @return Un booléen permettant de savoir si la lettre est présente autour
 * de celle consulté.
 */
bool sous_recherche(plateau& plat, const Mot mot, const unsigned int posi, const unsigned
int coord);

/**
 * @brief Affiche les mots saisit qui sont présents dans le plateau.
 * @param[in-out] plat Le plateau à consulter,
 * les lettres du plateau sont mis à jour au moment de la visite.
 * @param[in] l La liste de mot à consulter
 */
void afficher_mot_correct(plateau& plat, const Liste& l);
```

Grille.cpp

```cpp
/**
* @file Grille.cpp
* @brief Gestion du plateau
* @author Groupe 104 : AKHAYAR Mahir et HADDIOUI Sélim
*/

#include <iostream>
#include "Grille.h"

using namespace std;


bool recherche(plateau& plat, const Mot mot) {
	for (unsigned int i = 0; i < CASE; i++) {
		plat[i].visite = false;
	}
	for (unsigned int i = 0; i < CASE; i++) {
		if (sous_recherche(plat, mot, 0, i))
			return true;
	}
	return false;
}

bool est_adjacent(const Position& pos, const Position& posTest)
{
	if (pos.x == posTest.x && (pos.y + 1 == posTest.y || pos.y - 1 == posTest.y)) // Haut
/ Bas
		return true;
	if (pos.y == posTest.y && (pos.x == posTest.x + 1 || pos.x == posTest.x - 1)) //
Gauche / Droite
		return true;
	if (posTest.y == pos.y - 1 && posTest.x == pos.x - 1) // Haut gauche
		return true;
	if (posTest.y == pos.y + 1 && posTest.x == pos.x - 1) // Haut droite
		return true;
	if (posTest.y == pos.y + 1 && posTest.x == pos.x + 1) // Bas droite
		return true;
	if (posTest.y == pos.y - 1 && posTest.x == pos.x + 1) // Bas gauche
		return true;

	return false;
}

void saisir_grille(plateau& plat) {
	char mot[5];
	unsigned int j;
	for (j = 0; j < CASE;) {
		cin >> mot;
		for (unsigned int i = 0; i < Mot_Plateau; i++) {
			plat[j].lettre = mot[i];
			plat[j].pos = enreg_pos(j);
			j++;
		}
	}
	plat[CASE].lettre = '\0';
}
```

```
Position enreg_pos(const unsigned int pos)
{
	Position p;
	p.x = pos % 4;
	p.y = (pos - p.x) / 4;
	return p;
}

bool sous_recherche(plateau& plat, const Mot mot, const unsigned int posi, const unsigned
int coord)
{
	if (coord > 15)
		return false;

	if (posi >= strlen(mot))
		return true;

	if (plat[coord].lettre != mot[posi])
		return false;

	if (plat[coord].visite)
		return false;

	plat[coord].visite = true;
	for (unsigned int i = 0; i < 16; ++i)
		if (est_adjacent(plat[coord].pos, plat[i].pos))
			if (sous_recherche(plat, mot, posi + 1, i))
				return true;
	plat[coord].visite = false;
	return false;
}

void afficher_mot_correct(plateau& plat, const Liste& l) {
	for (unsigned int i = 0; i < l.nb; i++) {
		if (recherche(plat, l.c.tab[i]))
			cout << l.c.tab[i] << endl;
	}
	cout << "*" << endl;
}
```

Dictionnaire.h

```
#pragma once

/**
 * @file Dictionnaire.h
 * @brief Gestion du dictionnaire
 * @author Groupe 104 : AKHAYAR Mahir et HADDIOUI Sélim
 */

#include "Liste.h"
```

```
// Dico
struct Dico {

	Mot mot;
	int repetition;		// clé comptant les redondances du mot

};

// Dictionnaire
struct Dictionnaire {

	Dico* tab;			// Dictionnaire contenant les mot et leurs clés
	unsigned int nb;	// nb d'éléments contenus
	unsigned int capa;	// capacité du dictionnaire
	unsigned int pas;	// pas d'extension du dictionnaire
};

/**
 * @brief Initialise un dictionnaire.
 * Allocation en mémoire dynamique du dictionnaire
 * de capacité (capa) extensible par pas d'extension (p).
 * @see detruire, pour sa désallocation en fin d'utilisation.
 * @param[out] D Le Dictionnaire.
 * @param [in] capa Capacité du conteneur.
 * @param [in] p Pas d'extension de capacité.
 * @pre capa > 0 et p > 0.
 */
void initialiser_dico(Dictionnaire& d, unsigned int capa, unsigned int pas);

/**
 * @brief Désalloue le dictionnaire en mémoire dynamique.
 * @see initialiser.
 * @param[in,out] c Le conteneur d'items.
 */
void detruire_dico(Dictionnaire& d);

/**
 * @brief Lecture d'un mot du dictionnaire.
 * @param[in] d Le dictionnaire.
 * @param[in] i La position du mot dans le dictionnaire.
 * @return Le mot à la position i.
 * @pre i < c.capacite
 */
Mot lire_dico_it(const Dictionnaire& d, unsigned int i);

/**
 * @brief Lecture du nombre de répétition d'un mot du dictionnaire.
 * @param[in] d Le dictionnaire.
 * @param[in] i La position du mot dans le dictionnaire.
 * @return Le nombre de répétition du mot à la position i.
 * @pre i < c.capacite
 */
int lire_dico_rep(const Dictionnaire& d, unsigned int i);
```

```
/**
 * @brief Ecrire mot dans le dictionnaire.
 * @param[in,out] d Le dictionnaire.
 * @param[in] i La position où modifier le mot.
 * @param[in] it L'item à écrire.
 */
void ecrire_dico(Dictionnaire& c, unsigned int i, const Dico& it);

/**
 * @brief Ajoute un item Dico à la fin du Dictionnaire
 * si celui-ci est complet, le rend extensible.
 * @param[in-out] d La variable de type Dico où est rentrer l'Item.
 * @param[in] it Le mot à saisir
 */
void ajouter_fin_dico(Dictionnaire& dic, const Dico& it);

/**
 * @brief Affiche les mots et le nombre de répétition du mot d'un dictionnaire.
 * @param[in-out] dic Le dictionnaire à afficher.
 */
void affichage_dico(const Dictionnaire& dic);

/**
 * @brief Enregistre plusieurs liste dans un dictionnaire jusqu'à
 * la saisit d'une liste vide.
 * @param[in-out] dic Le dictionnaire où sont enregistrer les listes.
 * @param[in] l La liste à saisir.
 */
void recursiv_dico(Dictionnaire& dic, const Liste& l);

/**
 * @brief Change un Item en Dico et initialise sa répétition à 1.
 * @param[in-out] d La variable de type Dico où est rentrer l'Item.
 * @param[in] it Le mot à saisir
 */
void conversion_dico(Dico& d, const Item& it);

/**
 * @brief Ajouter des mots dans un dictionnaire où incrémenter son
 * nombre de répétition s'il y est déjà présent.
 * @param[in-out] dic Le dictionnaire à remplir.
 */
void remplir_dico(Dictionnaire& dic);
```

Dictionnaire.cpp

```
/**
* @file Dictionnaire.cpp
* @brief Gestion du dictionnaire
* @author Groupe 104 : AKHAYAR Mahir et HADDIOUI Sélim
*/

#include <iostream>
#include <cassert>

#include "Liste_mot.h"
#include "Tri_liste.h"

#pragma warning ( disable : 4996 )
using namespace std;
```

```
void initialiser_dico(Dictionnaire& d, unsigned int capa, unsigned int pas) {
	assert((capa > 0) && (pas > 0));
	d.capa = capa;
	d.pas = pas;
	d.nb = 0;
	d.tab = new Dico[capa];
}

void detruire_dico(Dictionnaire& d) {
	delete[] d.tab;
	d.tab = NULL;
}

Mot lire_dico_it(const Dictionnaire& d, unsigned int i) {
	assert(i < d.capa);
	return d.tab[i].mot;
}

int lire_dico_rep(const Dictionnaire& d, unsigned int i) {
	assert(i < d.capa);
	return d.tab[i].repetition;
}

void ecrire_dico(Dictionnaire& d, unsigned int i, const Dico& it) {
	if (i >= d.capa) {
		unsigned int newTaille = (i + 1) * d.pas;
		Dico* newT = new Dico[newTaille];
		for (unsigned int i = 0; i < d.capa; ++i)
			newT[i] = d.tab[i];
		delete[] d.tab;
		d.tab = newT;
		d.capa = newTaille;
	}
	d.tab[i] = it;
}

void ajouter_fin_dico(Dictionnaire& dic, const Dico& it) {
	dic.tab[dic.nb].mot = new char[30];
	strcpy(dic.tab[dic.nb].mot, it.mot);
	dic.tab[dic.nb].repetition = 1;
	dic.nb++;
	if (dic.nb >= dic.capa) {
		unsigned int newTaille = (dic.nb + 1) * dic.pas;
		Dico* newT = new Dico[newTaille];
		for (unsigned int i = 0; i < dic.capa; ++i)
			newT[i] = dic.tab[i];
		delete[] dic.tab;
		dic.tab = newT;
		dic.capa = newTaille;
	}
}

void affichage_dico(const Dictionnaire& dic) {
	for (unsigned int u = 0; u < dic.nb; u++) {
		cout << lire_dico_it(dic, u) << " " << lire_dico_rep(dic, u) << endl;
	}
	cout << "*" << endl;
}
```

```
void recursiv_dico(Dictionnaire& dic, const Liste& l) {
	Dico it;
	int comp;
	if (dic.nb == 0) {
		conversion_dico(it, l.c.tab[0]);
		ajouter_fin_dico(dic, it);
	}
	for (unsigned int i = 0; i < l.nb; i++) {
		for (unsigned int j = 0; j < dic.nb; j++) {
			comp = strcmp(l.c.tab[i], dic.tab[j].mot);
			if (comp == 0) {
				dic.tab[j].repetition++;
				break;
			}
			else
			{
				if (j == dic.nb - 1) {
					conversion_dico(it, l.c.tab[i]);
					ajouter_fin_dico(dic, it);
					break;
				}
			}
		}
	}
}

void conversion_dico(Dico& d, const Item& it) {
	d.mot = it;
	d.repetition = 1;
}

void remplir_dico(Dictionnaire& dic) {
	Liste l;
	initialiser_dico(dic, 10, 5);
	bool v = 0;
	while ( 1 ) {
		v = saisir_liste_trie(l);
		if (v == 1)
			break;
		recursiv_dico(dic, l);
	}
	dic.tab[0].repetition--;		// permet de résoudre un problème de
surincrémentation du premier mot
}
```