Structures de données et algorithmes fondamentaux

Mémoire de projet

Fabien RONDAN
Maxime WANG

Groupe 108

Tables des matières

Présentation du projet

Introduction :

Le Boggle est un jeu qui a pour but de trouver le maximum de mots de 3 lettres minimum dans une grille 4 par 4.

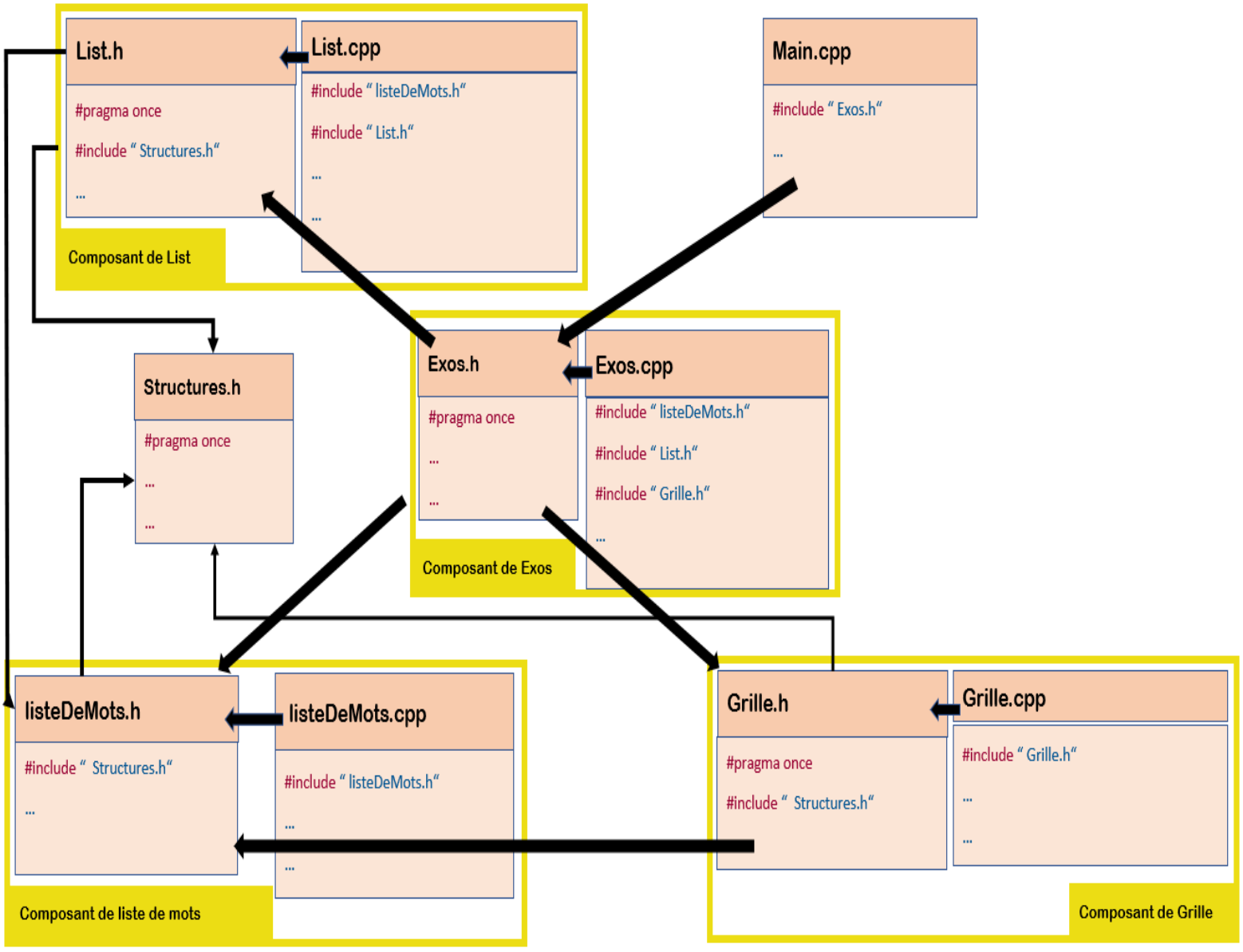
Problématique :

Programmer un jeu Boggle en multijoueur fonctionnel et jouable en C++.

Rôle fonctionnel du projet :

L'objectif de ce projet est de programmer un jeu Boggle fonctionnel sans utiliser les conteneurs de la bibliothèque standard du C++ tout en ayant une allocation en mémoire dynamique.

Graphe de dépendance



Bilan de validation des tests

Pour le bilan de la validation de tests, notre projet a validé les six exercices et le test complémentaire.

Le premier exo consiste donc à récupérer la liste de mots d'un joueur et le stocker dans un tableau dynamique puis calculer le score en fonction du nombres de lettres de chaque mots dans cette liste de mots. Pour ce faire nous avons dans un premier temps fait l'exercice sans allocation dynamique pour bien comprendre et mieux réussir les exercices. Puis nous avions commencé le projet assez tôt et nous n'avions donc pas encore vu l'allocation en mémoire dynamique. Nous avons choisi cette méthode de travail pour les cinq premiers exercices. Grâce à la fonction Share nous avons pu travailler en mode collaboratif sur tout le projet.

Ensuite, pour le deuxième exercice, nous devions trier cette liste par ordre alphabétique et supprimer les mots en doubles. Nous avons dû nous y reprendre plusieurs fois sur la fonction de tri tout d'abord nous avions oublié l'existence de la fonction strcmp ce qui nous a permis de mieux la comprendre. Puis avec les derniers cours, nous avons pu voir les tri les plus efficaces. Pour les doublons nous avons fait une fonction assez simple mais nous avions oublié de vérifier le mot à la même case lors de la suppression de doublons précédent, nous avons vite corrigé cette erreur et nous avons pu commencer l'exercice d'après.

Pour le troisième exercice nous nous sommes très vite aperçus qu'il était très similaire avec le quatrième. Dans l'un il fallait faire ressortir les mots de la deuxième liste qui n'apparaissait pas dans la première alors que dans l'autre exercice (°3) il fallait l'inverse et prendre les mots qui apparaissaient dans les deux listes, c'est pourquoi nous avons codé une fonction pour les deux exercices. Elle prend trois paramètres et le dernier qui est un booléen facultatif, si le paramètre trois n'est pas renseigné le booléen est par défaut 'true' (pour l'exo °4) et pour l'exercice trois, si on le met à 'false' le dernier paramètre.

Pour le cinq nous avons dû créer une nouvelle structure pour stocker une liste de liste. Nous avons eu quelques difficultés pour terminer la liste lorsque qu'il fallait rentrer deux étoiles (**) mais aussi lorsqu'il fallait rendre dynamique les listes de liste. En effet, nous avons eu beaucoup de problèmes pour l'allocation comme par exemple une mauvaise initialisation. Pour ensuite afficher les mots qui apparaissaient dans aux moins deux listes nous n'avons pas eu de problème.

Pour l'exercices six nous avons affiché les mots qui étaient sur le grille, pour lire le plateau il n'a pas eu de difficulté en revanche pour comprendre le script que le professeur a fourni il nous a fallu un peu plus de temps et de réflexions pour le comprendre mais une fois compris le programme était assez simple à coder, nous avons même pu faire l'exercice ultime-bis.bat et nous avons mis environ 1s pour le faire tourner sur nos ordinateurs.

Bilan de projet

Les difficultés rencontrées :

Durant ce projet on avait à faire un jeu Boggle multi-joueur, une contrainte nous étaient imposés, ils nous interdisaient d'utiliser les conteneurs de la bibliothèque standard du C++ (tels que vector, stack, etc.). Nous avons donc commencé le projet le plus tôt possible pour pouvoir prendre de l'avance mais aussi pour poser nos éventuelles questions à notre prof d'informatique M.Poitrenaud, si une difficulté survenait dans le code.

Après, nous avons aussi eu du mal au début du projet de codée en C++ en raison des habitudes qu'on avait acquise en C. Par exemple, on devait changer les pointeurs par les références.

On a également crée dans une de nos fonctions, sans avoir rien remarqué au départ, une fonction qui avait le même comportement que la fonction de la bibliothèque standard du C++ strcmp (). C'était seulement après avoir aider des gens d'autres groupes qu'on s'en est aperçu que l'on avait tout simplement pu avoir remplacer notre fonction par celle de la bibliothèque standard du C++, ce qui nous a donc fait perdre pas mal de temps. Cependant, cela reste une très belle recherche car elle nous a permis de comprendre comment la fonction compare() était codée.

De plus, une autre difficulté s'est manifestée dans le projet, en-faite il était difficile d'accéder aux valeurs se trouvant dans un tableau dynamique. En effet, les tableaux dynamiques n'affichent que les premiers éléments du tableau lorsque l'on marque un point d'arrêt à la fonction et donc nous avons pris un peu de temps lorsqu'il s'agissait de trouver un bug se trouvant dans une fonction qui utilise l'allocation en mémoire dynamique. Plus tard dans le projet, nous avons découvert un outil dans Visual qui permettait de résoudre ce problème qui s'appelle Espion Express grâce à notre prof M.Poitrenaud.

Ce qui peut-être amélioré :

Lors de ce projet, nous avons vu plusieurs points que l'on aurait pu améliorer. Tout d'abord, des fonctions qui pouvaient être séparées en deux ou être écrites plus simplement, nous avons pu en prendre conscience quand nous avons aidé nos camarades puisque nous avions la tête reposée et notre camarade voyait le problème différemment. Nous aurions peut-être pu éviter des lignes de code en nous reposant et en nous laissant la nuit pour y réfléchir. Ensuite pour la sauvegarde de nos fichiers nous avons chacuns stocké sur notre machine en envoyant sur le dossier sur Discord mais cela n'est pas très professionnel. Nous aurions pu utiliser par exemple Github qui est plus adapté car même si cela marchait à certains moments lorsqu'une personne touchait au code il fallait le renvoyer.

Ce qui est réussi :

Durant ce projet, nous avons aussi des choses dont nous sommes fières tout d'abord le travail d'équipe nous avons fait tout le projet ensemble ce qui nous a permis d'avancer plus rapidement car on réfléchissait à deux sur le problème. On pouvait aussi écrire en même temps et donc allait encore plus vite. Nous avons aussi eu une bonne gestion, en effet nous avons commencé le projet une à deux semaines après qu'il a été donné ce qui nous a permis de pas tout faire à la dernière minute. Il nous a aussi permis de commencer le projet avec les exercices 1 à 5 sans les allocations de mémoires, ce qui était plus facile. Ensuite une fois le cours vu, utiliser l'allocation dynamique ce qui était plus facile car nous avions déjà le code, donc nous n'avions plus qu'à le modifier et nous concentrer que sur l'allocation qui était pour nous une nouveauté.

Conclusion :

Tout d'abord ce projet collaboratif nous a appris à utiliser le langage C++. Ce projet nous a fait découvrir des moyens d'outils collaboratifs tels que Visual avec l'option Live Share, il nous a aussi permis de travailler à deux et de prendre conscience de l'importance du travail d'équipe ainsi qu'une bonne écriture des noms des variables et des commentaires afin que le code soit compris par l'ensemble des personnes qui utiliseront notre code. Le but premier de ce projet a été de comprendre et savoir utiliser l'allocation dynamique ainsi que de voir les subtilités du langage C++ tel que le passage par référence.

Annexe : le code complet

Fabien RONDAN
Maxime WANG

Groupe 108

Structures.h

```h
#pragma once

#define col 4
#define row 4

typedef char Mot[25 + 1];

/** @brief Liste de mots alloués en mémoire dynamique
 *  de capacité extensible suivant un pas d'extension
 */
struct listeDeMots {
    Mot* TabDeMots;            // tableau de mots alloué en mémoire dynamique
    unsigned int nombreDeMots; // nombre de mots (>0)
    unsigned int capacite;     // capacité du conteneur (>0)
    unsigned int pasExtension; // pas d'extension du conteneur (>0)
};


/**
 * @brief Liste de listes de mots alloués en mémoire dynamique
 *  de capacité extensible
*/
struct List {
    listeDeMots* listOfList;
    unsigned int nombreDeListes = 0;
    unsigned int capacite = 0;     // capacité du conteneur (>0)
};


/**
 * @brief La grille ou autrement dit le plateau de jeu Boggle
*/
struct Grille{
    char grille[col][row];
    bool grilleBool[col][row];
    unsigned int nombreDeLettres;
};

/**
 * @brief Les coordonnées utilisées pour retrouver la lettre dans la grille
*/
struct Coord {
    int x, y;
};
```

Main.cpp

```cpp
#pragma warning(disable:4996)

#include <iostream>

#include "Exos.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;
    }
}
```


```h
#pragma once

/**
 * @brief  Correspond au main du premier programme
*/
void exo1();

/**
 * @brief  Correspond au main du deuxième programme
*/
void exo2();

/**
 * @brief  Correspond au main du troisième programme
*/
void exo3();

/**
 * @brief  Correspond au main du quatrième programme
*/
void exo4();

/**
 * @brief  Correspond au main du cinquième programme
*/
void exo5();

/**
 * @brief  Correspond au main du sixième programme
*/
void exo6();
```

Exos.cpp

```cpp
#pragma warning(disable:4996)

#include "listeDeMots.h"
#include "List.h"
#include "Grille.h"


void exo1() {
    listeDeMots liste;

    initialiser(liste, 1, 5);
    motTrouver(liste);
    afficheScore(liste);
    detruire(liste);
}

void exo2() {
    listeDeMots liste;

    initialiser(liste, 1, 5);
    motTrouver(liste);

    tri(liste);
    supprimeDoublons(liste);
    afficher(liste);
    detruire(liste);
}
```

```
void exo3() {
    listeDeMots liste, liste2;

    initialiser(liste, 1, 5);
    initialiser(liste2, 1, 5);
    motTrouver(liste);
    motTrouver(liste2);

    tri(liste);
    supprimeDoublons(liste);
    tri(liste2);
    supprimeDoublons(liste2);

    showCommomWord(liste, liste2, false);

    detruire(liste);
    detruire(liste2);
}

void exo4() {
    listeDeMots liste, liste2;

    initialiser(liste, 1, 5);
    initialiser(liste2, 1, 5);
    motTrouver(liste);
    motTrouver(liste2);

    tri(liste2);
    supprimeDoublons(liste2);

    showCommomWord(liste, liste2);

    detruire(liste);
    detruire(liste2);
}
```

```
void exo5() {
    List listOfLists;
    listeDeMots finalWordList;
    initialiser(finalWordList, 1, 5);

    ajouterListe(listOfLists);
    showCommonWordList(listOfLists, finalWordList);
    tri(finalWordList);

    supprimeDoublons(finalWordList);
    afficher(finalWordList);

    detruire(finalWordList);

    for (int i = 0; i < listOfLists.nombreDeListes; i++) {

        detruire(listOfLists.listOfList[i]);

    }
}

void exo6() {
    listeDeMots liste;
    Grille grille;

    ecrireGrille(grille);

    initialiser(liste, 1, 5);
    motDansGrille(liste, grille);

    tri(liste);
    supprimeDoublons(liste);
    afficher(liste);

    detruire(liste);
}
```


```h
#pragma once
#include "Structures.h"


/**
 * @brief Initialise une liste de mots
 * Allocation en mémoire dynamique d'une liste de mots
 * de capacité (capa) extensible par pas d'extension (p).
 * @see detruire, pour sa désallocation en fin d'utilisation
 * @param [out] ref_liste : la liste de mots.
 * @param [in] capa : capacité de la liste.
 * @param [in] p : pas d'extension de capacité.
 * @pre capa>0 et p>0
 */
void initialiser(listeDeMots& ref_liste, unsigned int capa, unsigned int p);


 /**
  * @brief Permet d'afficher la liste de mots.
  * @param [in] ref_liste : La liste de mots auquel on va afficher toutes ces valeurs.
  * @pre ref_liste.nombreDeMots < ref_liste.capacite
 */
void afficher(const listeDeMots& ref_liste);


/**
 * @brief Désalloue la liste de mots en mémoire dynamique.
 * @see initialiser, la liste de mots a déjà été alloué
 * @param [out] ref_liste : la liste de mots.
 */
void detruire(listeDeMots& ref_liste);
```

```
/**
 * @brief Ecrit le mot en paramètre dans la liste de mots.
 * @param [out] ref_liste : La liste de mots auquel on va insérer notre mot
 * et augmente sa capacité si le nombreDeMots >= la capicité.
 * @param [in] mot : Le mot à insérer dans la liste de mots.
*/
void ecrire(listeDeMots& ref_liste, const Mot& mot);


/**
 * @brief Permet de saisir une liste de mots au clavier.
 * @param [in, out] ref_liste : La liste de mots auquel on insère le mot saisit en entrée et
 * on incrémente à chaque fois le nombre de mots d'un (+1).
*/
void motTrouver(listeDeMots& ref_liste);


/**
 * @brief Affiche à l'écran le nombre de points que la liste de mots représente.
 * @param [in] ref_liste : La liste de mots auquel on va lire la taille de chaque mots pour afficher
 * à l'écran le nombre de points que la liste de mots représente.
*/
void afficheScore(listeDeMots& ref_liste);


/**
 * @brief Permet de trier la liste de mots saisie au clavier.
 * @param [in, out] ref_liste : La liste de mots auquel on va trier ses mots.
*/
void tri(listeDeMots& ref_liste);
```

```
/**
 * @brief Permet de supprimer les mots de la liste retrouver en doublons et
 * réalloue à la fin un tableau de la bonne taille (nombre de mots y compris l'étoile).
 * @param [in, out] ref_liste : La liste de mots auquel on supprime les mots répétés.
*/
void supprimeDoublons(listeDeMots& ref_liste);


/**
 * @brief Renvoie "true" si l'un des mots de la seconde liste apparaissent dans la première,
 * renvoie "false" si les mots de la seconde liste n'apparaissent pas dans la première.
 * @param [in] ref_liste1 : La liste de mots auquel on va comparer avec le mot de la seconde liste.
 * @param [in] ref_liste2 : La seconde liste de mots auquel on va le comparer avec les mots
 * de la première liste.
 * @param [in] i : L'index position du mot de la seconde liste auquel on va comparer avec les autres.
 * mots de la première liste.
 * @return Un Boolean soit "true" soit "false".
*/
bool compare(listeDeMots& ref_liste1, listeDeMots& ref_liste2, const unsigned int i);
```

```
/**
 * @brief Le boolean modifie le comportement de la fonction, "true" est la valeur
 * par défaut, la fonction affiche à l'écran les mots de la seconde liste apparaissant
 * dans la première. Si on insère "false", la fonction affiche à l'écran les mots de la
 * seconde liste n'apparaissant pas dans la première.
 * @param [in] ref_liste1 : La première liste de mots auquel on va comparer avec
 * la deuxième liste de mots.
 * @param [in] ref_liste2 : La deuxième liste de mots auquel on va comparer avec
 * la première liste de mots.
 * @param [in] boolean : Un boolean définie par défaut "true".
*/
void showCommomWord(listeDeMots& ref_liste1, listeDeMots& ref_liste2, bool boolean = true);


/**
 * @brief Initialise une liste de mots de capacité augmenté
 * Allocation en mémoire dynamique d'une liste de mots
 * de capacité (capa) extensible par pas d'extension (p) et
 * on repasse la liste de mots de la nouvelle liste dans l'ancienne liste 'ref_liste'.
 * @see detruire, pour sa désallocation en fin d'utilisation
 * @param [in, out] ref_liste : La liste de mots auquel on va augmenter sa capacité
 * si le nombre de mots >= la capacité.
*/
void listeDynamique(listeDeMots& ref_liste);
```

listeDeMots.cpp

```cpp
#pragma warning(disable:4996)
#include <iostream>
#include <cassert>

#include "listeDeMots.h"

using namespace std;


void initialiser(listeDeMots& ref_liste, unsigned int capa, unsigned int p) {
    assert(capa > 0 || p > 0);

    ref_liste.capacite = capa;
    ref_liste.pasExtension = p;
    ref_liste.TabDeMots = new Mot[capa];
    ref_liste.nombreDeMots = 0;
}


void afficher(const listeDeMots& ref_liste) {
    assert(ref_liste.nombreDeMots <= ref_liste.capacite);

    for (unsigned int i = 0; i < ref_liste.nombreDeMots; i++)
    {
        cout << ref_liste.TabDeMots[i] << endl;
    }

}


void detruire(listeDeMots& ref_liste) {
    delete[] ref_liste.TabDeMots;
}
```

```
void ecrire(listeDeMots& ref_liste, const Mot& mot) {

        if (ref_liste.nombreDeMots >= ref_liste.capacite) {

            int nouvelleTaille = (ref_liste.nombreDeMots + 1) * ref_liste.pasExtension;

            Mot* nouvelleListe = new Mot[nouvelleTaille];

            for (unsigned int j = 0; j < ref_liste.capacite; j++)
            {
                strcpy(nouvelleListe[j], ref_liste.TabDeMots[j]);
            }
            delete[] ref_liste.TabDeMots;
            ref_liste.TabDeMots = nouvelleListe;
            ref_liste.capacite = nouvelleTaille ;

            // Changement du Pas d'Extention en fonction de la taille de la liste
            if (ref_liste.capacite <= 5000) { ref_liste.pasExtension = 4; }
            if (ref_liste.capacite <= 50000) { ref_liste.pasExtension = 3; }
            if (ref_liste.capacite <= 100000) { ref_liste.pasExtension = 2; }

        }

        strcpy(ref_liste.TabDeMots[ref_liste.nombreDeMots], mot);
        ref_liste.nombreDeMots++;

}
```

```
void motTrouver(listeDeMots& ref_liste) {
    bool isAlreadyInput = true;

    while (isAlreadyInput)
    {
        listeDynamique(ref_liste);

        cin >> ref_liste.TabDeMots[ref_liste.nombreDeMots];
        ref_liste.nombreDeMots++;

        if (strcmp(ref_liste.TabDeMots[ref_liste.nombreDeMots - 1], "*") == 0) {
            isAlreadyInput = false;
        }

    }
}


void afficheScore(listeDeMots& ref_liste) {
    unsigned int nbPoints = 0;

    for (unsigned int i = 0; i < ref_liste.nombreDeMots; i++) {
        unsigned int lenghtofString = strlen(ref_liste.TabDeMots[i]);
        if (lenghtofString >= 8) { nbPoints = nbPoints + 11; }
        if (lenghtofString == 7) { nbPoints = nbPoints + 5; }
        if (lenghtofString == 6) { nbPoints = nbPoints + 3; }
        if (lenghtofString == 5) { nbPoints = nbPoints + 2; }
        if (lenghtofString == 3 || lenghtofString == 4) { nbPoints = nbPoints + 1; }
    }
    cout << nbPoints;
}
```

```
void tri(listeDeMots& ref_liste) {

    for (unsigned int i = 0; i < ref_liste.nombreDeMots - 2; i++)
    {
        for (unsigned int y = 0; y < ref_liste.nombreDeMots - 2 - i; y++)
        {
            if (strcmp(ref_liste.TabDeMots[y], ref_liste.TabDeMots[y + 1]) > 0) {
                Mot permute;
                strcpy(permute, ref_liste.TabDeMots[y]);
                strcpy(ref_liste.TabDeMots[y], ref_liste.TabDeMots[y + 1]);
                strcpy(ref_liste.TabDeMots[y + 1], permute);
            }
        }
    }
}
```

```
void supprimeDoublons(listeDeMots& ref_liste) {

    for (unsigned int i = 0, j = 1; j < ref_liste.nombreDeMots - 1; i++, j++)
    {
        if (strcmp(ref_liste.TabDeMots[i], ref_liste.TabDeMots[j]) == 0) {
            for (unsigned int k = j + 1, l = j; k < ref_liste.nombreDeMots + 1; k++, l++)
            {
                strcpy(ref_liste.TabDeMots[l], ref_liste.TabDeMots[k]);
            }
            j = j - 1;
            i = i - 1;
            ref_liste.nombreDeMots -= 1;
        }
    }

    Mot* liste = new Mot[ref_liste.nombreDeMots];
    for (unsigned int i = 0; i < ref_liste.nombreDeMots; i++)
    {
        strcpy(liste[i], ref_liste.TabDeMots[i]);

    }
    delete[] ref_liste.TabDeMots;

    ref_liste.capacite = ref_liste.nombreDeMots;
    ref_liste.TabDeMots = liste;
}
```

```
bool compare(listeDeMots& ref_liste1, listeDeMots& ref_liste2, const unsigned int i) {
    bool isTheSameWord = false;
    for (unsigned int j = 0; j < ref_liste1.nombreDeMots - 1; j++)
    {
        if (strcmp(ref_liste1.TabDeMots[j], ref_liste2.TabDeMots[i]) == 0)
        {
            isTheSameWord = true;
            return isTheSameWord;
        }
    }
    return isTheSameWord;
}


void showCommomWord(listeDeMots& ref_liste1, listeDeMots& ref_liste2, bool boolean) {
    for (unsigned int i = 0; i < ref_liste2.nombreDeMots; i++)
    {
        if (compare(ref_liste1, ref_liste2, i) == boolean) {

            cout << ref_liste2.TabDeMots[i] << endl;
        }
    }
    if (boolean)
        cout << "*" << endl;
    cout << endl << endl;
}
```

```
void listeDynamique(listeDeMots& ref_liste) {

    if (ref_liste.nombreDeMots >= ref_liste.capacite) {

        int nouvelleTaille = (ref_liste.nombreDeMots + 1) * ref_liste.pasExtension;

        Mot* nouvelleListe = new Mot[nouvelleTaille];

        for (unsigned int j = 0; j < ref_liste.capacite; j++)
        {
            strcpy(nouvelleListe[j], ref_liste.TabDeMots[j]);
        }

        delete[] ref_liste.TabDeMots;
        ref_liste.TabDeMots = nouvelleListe;
        ref_liste.capacite = nouvelleTaille;

        // Changement du Pas d'Extention en fonction de la taille de la liste
        if (ref_liste.capacite <= 5) { ref_liste.pasExtension = 4; }
        if (ref_liste.capacite <= 20) { ref_liste.pasExtension = 3; }
        if (ref_liste.capacite <= 60) { ref_liste.pasExtension = 2; }

    }
}
```


```h
#pragma once
#include "Structures.h"


/**
 * @brief Ajoute la liste de mots saisie au clavier à la grande liste "listOfLists".
 * @param [in, out] ref_list : La liste des listes de mots auquel on va insérer des listes de mots
 * et incrémente de un (+1) le nombre de listes à chaque fois qu'une liste de mots se termine.
*/
void ajouterListe(List& ref_list);


/**
 * @brief Renvoie "true" si le mot de la première liste apparait dans la seconde,
 * renvoie "false" si le mot de la première liste n'apparait pas dans la seconde.
 * @param [in] ref_list : La grande liste des listes de mots auquel on va s'en servir
 * pour avoir le mot de la liste souhaité.
 * @param [in] idxList : L'indice de la liste courante.
 * @param [in] nextIdxList : L'indice de la liste suivante.
 * @param [in] i : L'indice du mot auquel on va le comparer avec les mots d'une liste.
 * @return Un Boolean soit "true" soit "false".
*/
bool compare2(List& ref_list, unsigned int& idxList, unsigned int& nextIdxList, const unsigned int i);
```

```
/**
 * @brief Permet d'insérer dans la liste "finalWordList" tous les mots apparaissant
 * dans au moins deux listes de mots.
 * @param [in] ref_list : La liste des listes de mots auquel on va vérifier les mots en communs
 * qui se retrouve dans au moins deux listes.
 * @param [in] idxList : L'indice de la liste comparé.
 * @param [in] nextIdxList : L'indice de la liste comparé.
 * @param [out] finalWordList : La liste auquel on va insérer tous les mots en communs dans
 * au moins deux listes.
*/
void saveCommonWordList(List& ref_list, unsigned int& idxList, unsigned int& nextIdxList,
    listeDeMots& finalWordList);


/**
 * @brief Affiche la liste de mots "finalWordList" auquel on a inséré les mots en communs des
 * différentes listes de mots.
 * @param [in] listOfWord : La liste des listes de mots auquel on va vérifier les mots en communs
 * qui se retrouve dans au moins deux listes.
 * @param [out] finalWordList : La liste de mots auquel on va insérer tous les mots en communs
 * dans au moins deux listes.
*/
void showCommonWordList(List& listOfWord, listeDeMots& finalWordList);
```

List.cpp

```cpp
#pragma warning(disable:4996)
#include <iostream>

#include "listeDeMots.h"
#include "List.h"

using namespace std;


void ajouterListe(List& ref_list) {
    bool saisiList = true;

    for (int i = 0; saisiList == true; i++)
    {
        if (ref_list.nombreDeListes >= ref_list.capacite)
        {
            listeDeMots* listeMot = new listeDeMots[ref_list.capacite + 2];

            for (int k = 0; k <= i; k++)
            {
                initialiser(listeMot[i], 1, 5);
            }

            if (ref_list.capacite != 0) {
                for (unsigned int j = 0; j < ref_list.nombreDeListes; j++)
                {
                    listeMot[j] = ref_list.listOfList[j];
                }
                delete[] ref_list.listOfList;
            }

            ref_list.listOfList = listeMot;
            ref_list.capacite = ref_list.capacite + 2;

        }

        initialiser(ref_list.listOfList[i], 1, 5);
        motTrouver(ref_list.listOfList[i]);

        ref_list.nombreDeListes++;

        if (strcmp(ref_list.listOfList[i].TabDeMots[0], "*") == 0) {
            saisiList = false;
            ref_list.nombreDeListes--;
        }

    }
}
```

```
bool compare2(List& ref_list, unsigned int& idxList, unsigned int& nextIdxList, const unsigned int i) {
    bool isTheSameWord = false;

    for (unsigned int j = 0; j < ref_list.listOfList[nextIdxList].nombreDeMots - 1; j++)
    {
        if (strcmp(ref_list.listOfList[idxList].TabDeMots[i], ref_list.listOfList[nextIdxList].TabDeMots[j]) == 0)
        {
            isTheSameWord = true;
            return isTheSameWord;
        }
    }

    return isTheSameWord;
}


void saveCommonWordList(List& ref_list, unsigned int& idxList, unsigned int& nextIdxList, listeDeMots& finalWordList) {

    for (unsigned int i = 0; i < ref_list.listOfList[idxList].nombreDeMots - 1; i++)
    {
        if (compare2(ref_list, idxList, nextIdxList, i) == true) {

            ecrire(finalWordList, ref_list.listOfList[idxList].TabDeMots[i]);

        }
    }

}


void showCommonWordList(List& listOfWord, listeDeMots& finalWordList) {

    for (unsigned int idxList = 0; idxList < listOfWord.nombreDeListes; idxList++)
    {
        for (unsigned int nextIdxList = idxList + 1; nextIdxList < listOfWord.nombreDeListes; nextIdxList++)
        {
            saveCommonWordList(listOfWord, idxList, nextIdxList, finalWordList);
        }
    }

    Mot etoile = "*";
    ecrire(finalWordList, etoile);

}
```

Grille.h

```h
#pragma once
#include "Structures.h"


/**
 * @brief Ecrire les lettres dans la grille de Boggle.
 * @param [out] ref_grille : La grille dans laquelle on écrit les lettres.
*/
void ecrireGrille(Grille& ref_grille);


/**
 * @brief Marquer toutes les lettres de la grille comme étant non visitées par un booleen "false".
 * @param [out] ref_grille : La grille auquel on marque toutes les cases comme étant non visitées.
*/
void resetToFalse(Grille& ref_grille);


/**
 * @brief Renvoie "true" si le mot se trouve dans la grille sinon renvoie "false".
 * @param [in] string : Le mot auquel on vérifie s'il existe dans la grille.
 * @param [in, out] ref_grille : La grille auquel on va vérifier si le mot existe dans cette grille.
 * @return Un Boolean soit "true" soit "false".
*/
bool recherche(Mot& string, Grille& ref_grille);
```

```
/**
 * @brief Renvoie "true" si la lettre du mot 'String' qui a pour coordonnée coord se trouve dans
 * la grille et implémente +1 à la position pos du mot sinon Renvoie "false" et décrémente -1 à la
 * position pos du mot.
 * @param [in, out] ref_grille : La grille qui permet de vérifier si le mot se trouve dans celle-ci et
 * si c'est le cas alors on marque la lettre de la grille comme étant visitée.
 * @param [in] string : Le mot auquel on vérifie s'il se trouve dans la grille Boggle.
 * @param [in, out] pos : La position de la lettre dans le mot auquel on incrémente de 1
 * si il se trouve dans la grille et correspond au mot string.
 * @param [in, out] coord : Elle désigne une case de la grille auquel on utilise pour retrouver la lettre
 * à cette coordonnée et on va changer ses coordonnées si la lettre est présente dans la grille.
 * @return Un Boolean soit "true" soit "false".
*/
bool SousRecherche(Grille& ref_grille, Mot& string, unsigned int pos, Coord& coord);


/**
 * @brief Permet de stocker les mots qui existent dans la grille dans la liste de mots.
 * @param [in, out] ref_liste : La liste de mots auquel on va insérer tous les mots qui existe dans la grille
 * dans la liste de mots "ref_liste" et alloue plus de capacité en mémoire si le nombre de mots >= la capacité.
 * @param [in] ref_grille : La grille Boggle auquel on va vérifier si le mot dans la liste de mot
 * existe dans la grille.
*/
void motDansGrille(listeDeMots& ref_liste, Grille& ref_grille);
```

Grille.cpp

```cpp
#pragma warning(disable:4996)
#include <iostream>

#include "listeDeMots.h"
#include "Grille.h"

using namespace std;

void ecrireGrille(Grille& ref_grille) {

    for (int i = 0; i < col; i++) {

        for (int j = 0; j < row; j++) {

            cin >> ref_grille.grille[i][j];
            ref_grille.nombreDeLettres++;
        }

    }

}


void motDansGrille(listeDeMots& ref_liste, Grille& ref_grille) {
    bool isAlreadyInput = true;

    while (isAlreadyInput)
    {
        listeDynamique(ref_liste);

        Mot tmpString;
        cin >> tmpString;

        if (recherche(tmpString, ref_grille)) {
            strcpy(ref_liste.TabDeMots[ref_liste.nombreDeMots], tmpString);
            ref_liste.nombreDeMots++;
        }

        if (strcmp(tmpString, "*") == 0) {
            strcpy(ref_liste.TabDeMots[ref_liste.nombreDeMots], tmpString);
            ref_liste.nombreDeMots++;
            isAlreadyInput = false;
        }

    }
}
```

```
bool recherche(Mot& string, Grille& ref_grille) {
    Coord coord;

    resetToFalse(ref_grille);

    for (int x = 0; x < col; x++)
    {
        for (int y = 0; y < row; y++)
        {
            coord.x = x;
            coord.y = y;
            if (SousRecherche(ref_grille, string, 0, coord))
                return true;
        }
    }

    return false;
}


void resetToFalse(Grille& ref_grille) {

    for (int i = 0; i < col; i++)
    {
        for (int y = 0; y < row; y++)
        {
            ref_grille.grilleBool[i][y] = false;
        }
    }

}
```

```
bool SousRecherche(Grille& ref_grille, Mot& string, unsigned int pos, Coord& coord) {

    if (pos >= strlen(string))
        return true;
    if (coord.x >= col || coord.y >= row)
        return false;
    if (ref_grille.grille[coord.x][coord.y] != string[pos])
        return false;
    if (ref_grille.grilleBool[coord.x][coord.y] == true)
        return false;

    ref_grille.grilleBool[coord.x][coord.y] = true;


    for (int ligne = max(0, coord.x - 1); ligne <= coord.x + 1; ligne++) {

        for (int colonne = max(0, coord.y - 1); colonne <= coord.y + 1; colonne++) {

            Coord newCoord;
            newCoord.x = ligne;
            newCoord.y = colonne;
            if (SousRecherche(ref_grille, string, pos + 1, newCoord))
                return true;

        }
    }

    ref_grille.grilleBool[coord.x][coord.y] = false;
    return false;
}
```

Annexe : le code complet

FIN

Merci de votre visite !!!