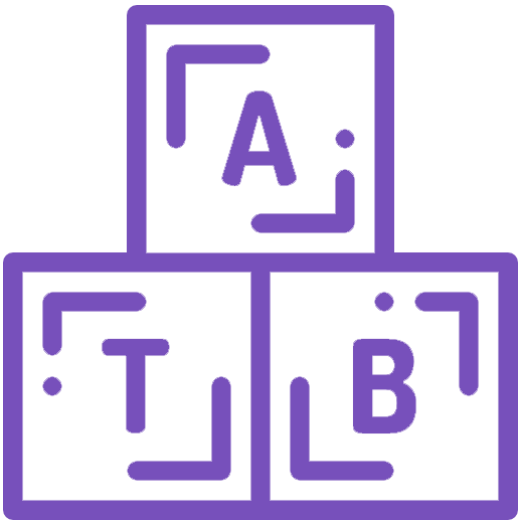


Rapport de projet

Structure des données et algorithmes fondamentaux

Boggle

Antoine BANHA
Groupe 102

Rayan ATROUNI
Groupe 112

DUT 1ère Année

Université de Paris

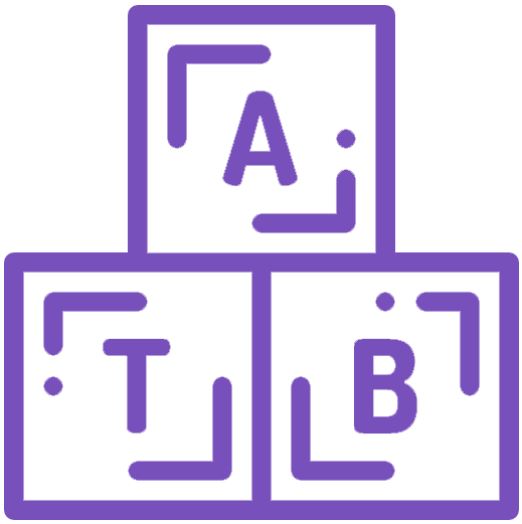


Table des matières

Antoine BANHA
Groupe 102

Rayan ATROUNI
Groupe 112

Université de Paris

Introduction du projet

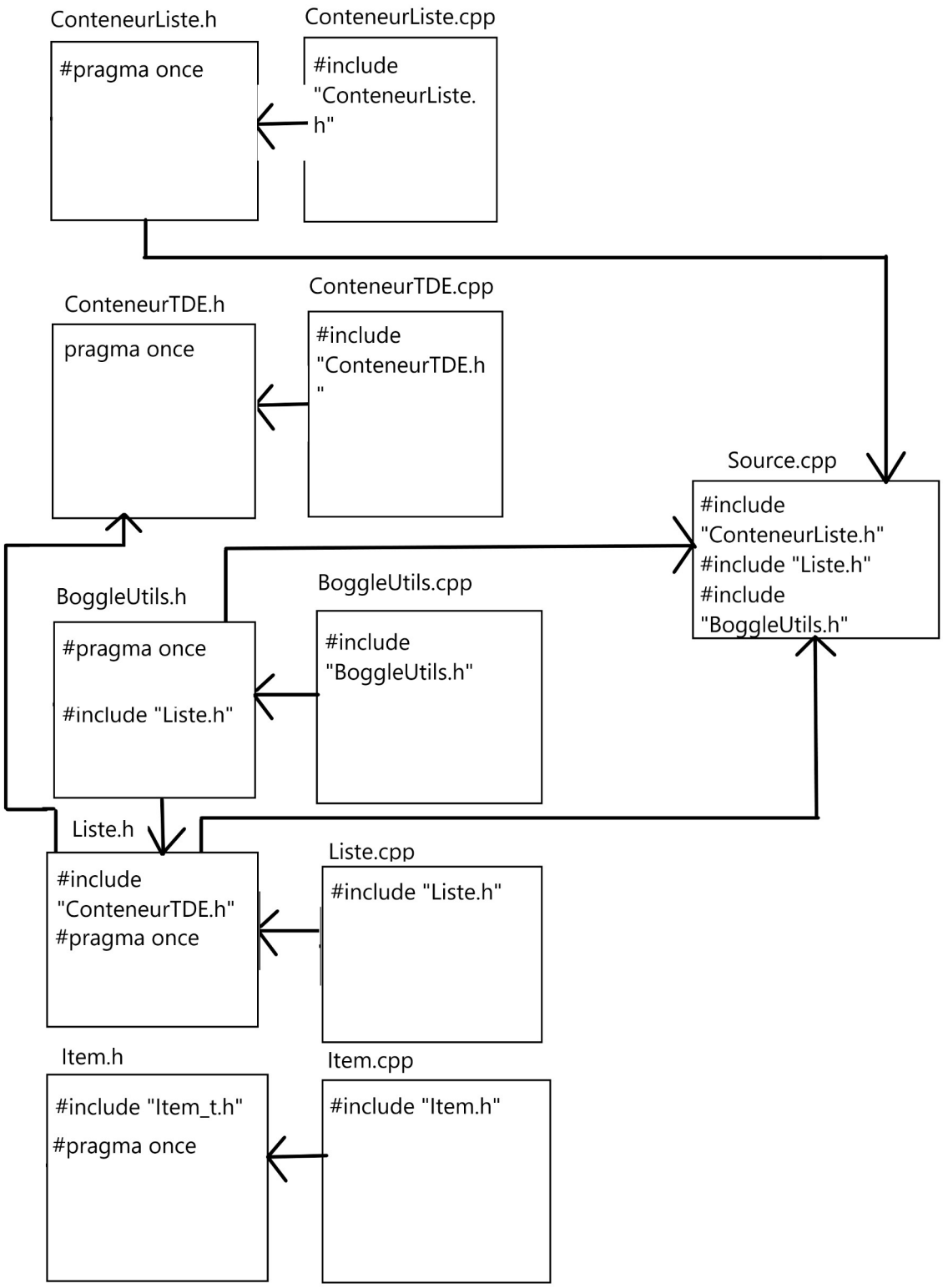
Ce projet consiste à créer un exécutable pour le jeu « Boggle », un jeu fonctionnant à partir d'une grille de lettre. Cette exécutable fonctionnerai en ligne de commande afin de laisser le test ultime interpréter lui-même ce que l'exécutable renvoie, il a dû falloir également normaliser nos affichages afin que les programmes ne se bloquent pas lors d'une saisie de liste par exemple.

Nous avons pu réaliser 6 exercices différents correspondant aux différents mécanisme du Boggle en C++ afin que le test ultime puisse les utiliser et calculer les points d'une partie. Nous avons également eu des contraintes présentées dans le sujet du projet qui nous ont nécessité de la réflexion sur nos méthodes, en effet, nous n'avions pas le droit d'utiliser les conteneurs de la bibliothèque standard du C++ et nous devions utiliser le maximum de code partagé entre les différents exercices. Pour régler le problème du partage de code, nous avions eu le droit à une architecture imposée qui nous a permis de ré-utiliser un maximum nos fichiers sources et d'entête mais également de faire attention aux changements que nous faisions car ils auraient alors des répercussions directes sur tous les autres exercices.

Nous avons également structuré notre code de façon logique et aérée en portant une attention particulière à la règle des 80 caractères par ligne.

Pour conclure, nous avons opté pour une solution de contrôle de version qui est Git afin de travailler en mode collaboration à distance. Nous avons également utilisé Visual Studio Live Share pour éditer le projet ensemble.

Graphe des dépendances



Antoine BANHA – Rayan ATROUNI

Listing des tests unitaires

Nous avons mené plusieurs tests pour assurer la fiabilité de nos composants, nous avons notamment testé le composant **Liste**, un composant extrêmement utilisé à travers notre programme.

Nous avons établi pour les premiers tests, une liste de 10 mots de façon automatique grâce à ce programme :

```
Liste l_1;
initialiser(l_1, 1, 2);

for (int i = 0; i < 10; ++i) {
	Item it;
	sprintf_s(it.mot, "test%d", i);
	inserer(l_1, longueur(l_1), it);
}
```

Nous avons commencé à tester les petites fonctions comme « longueur » :

```
assert(longueur(l_1) == 10);
```

Mais également la fonction lire :

```
assert(strcmp(lire(l_1, 3).mot, "test3") == 0);
```

Nous avons également testé une majorité de nos nouvelles fonctions :

```
Item it_test;
strcpy_s(it_test.mot, "test2");
assert(getByItem(l_1, it_test) == 2);


Liste l_2;

initialiser(l_2, 1, 2);

Item it;
strcpy_s(it.mot, "test");

for (int i = 0; i < 5; ++i) {
	inserer(l_2, longueur(l_2), it);
}
for (int i = 0; i < 4; ++i) {
	Item it_2;
	sprintf_s(it_2.mot, "test%d", i);
	inserer(l_2, longueur(l_2), it_2);
}

assert(occurences(l_2, it) == 5);
```

Listing des tests unitaires

```
auto filtre = [](Liste& l, Item& it) {
	int index = getByItem(l, it);
	return (index > -1);
};

assert(longueur(filtrer(l_1, l_2, filtre)) == 4);
```

Ces tests sont lancés en mode debug au début du programme afin de s'assurer que le programme n'aura pas de problème dans son exécution.

Bilan du projet

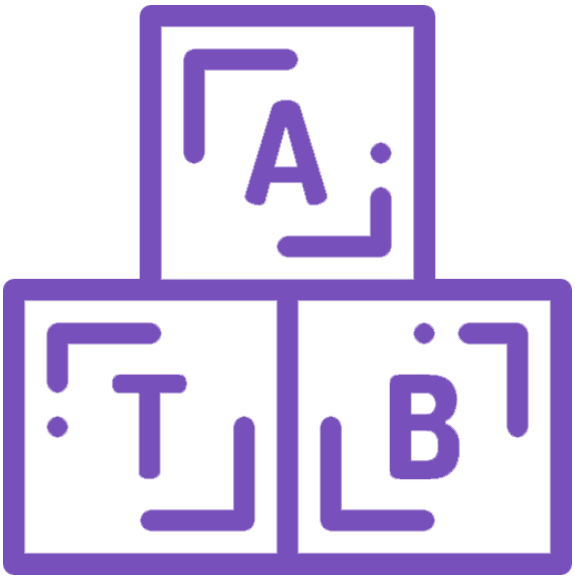
Ce projet a été assez enrichissant et assez unique par rapport au premier projet que nous avons mené qui était celui d'IAP.

Nous avons rencontré pendant ce projet plusieurs difficultés qui ont mené à plusieurs actions de notre part. La première action a été une réécriture entière de notre code source par la mise en place d'une structure imposée. Nos composants n'étaient pas assez bien structurés et nous n'étions pas vraiment satisfaits, donc nous avons décidé de réécrire le code pour ainsi partir sur de bonnes bases. Ce projet a bien évidemment demandé beaucoup de réflexion sur la structure de celui-ci, il nous a été imposé l'utilisation des « headers » pour y mettre nos prototypes, structures et types et cela n'a pas été très difficile.

Nous avons décidé de partir sur une structure assez claire utilisant pour la plupart des exercices, que des fonctions qui sont partagées avec les autres exercices et cela a été plutôt compliqué à mettre en œuvre, nous pouvons parler de la fonction de filtrage qui nécessite un pointeur de fonction mais qui s'est avéré très intéressant à utiliser pour passer de l'exercice 4 à 5 en un changement de ligne. Cette méthode est très utilisée dans le monde du JavaScript sous forme de « callbacks » et nous avons voulu essayer son implémentation en C++.

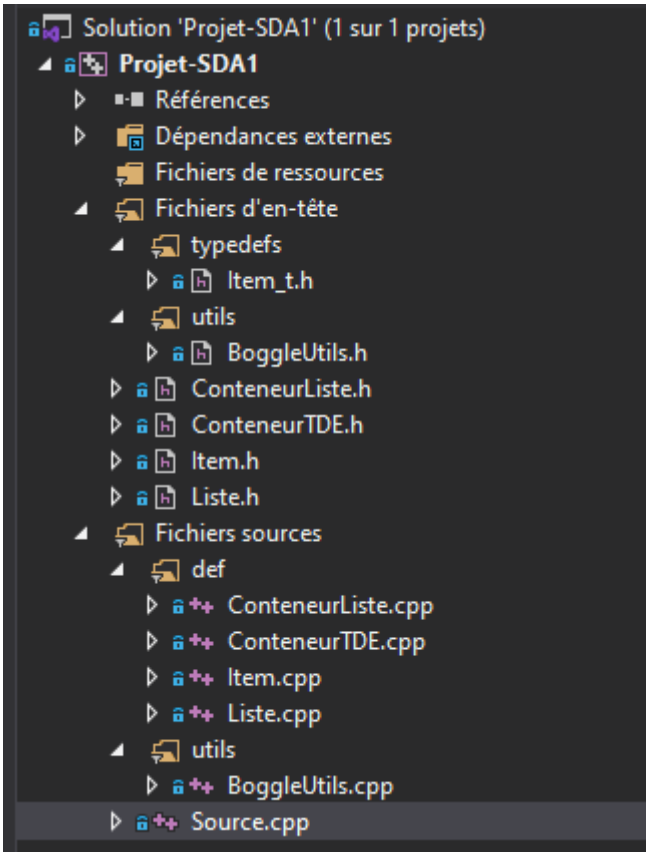
Nous sommes assez fiers du résultat final mais il y a quand même certains points que nous avons souhaité améliorer mais n'avons pas vraiment eu le temps (et le courage!) de le faire, notamment la structure **Mot**, qui présente un tableau statique pour le stockage d'une chaîne de caractères alors que nous pouvions utiliser le chaînage pour avoir une taille dynamique et cela aurait également demandé d'implémenter encore plus de fonctions pour permettre la lecture de ces chaînes.

La structure globale du projet a été pensée pour être assez aérée et facile à comprendre malgré son caractère non standard : nous avons décidé de séparer le plus possible les différents éléments au cas où nous avions besoin de tel et tel élément sans avoir à rapporter tous les prototypes de fonctions, même si cela s'avère inutile grâce au *#pragma once* du compilateur de Microsoft.



Annexes

Code source du projet



Structure du projet sur Visual Studio

```h
#pragma once

/**
 * @file BoggleUtils.h
 * @brief Utilitaires pour la mise en oeuvre du boggle.
 */

#include "Liste.h"

/**
  * @brief Définition du type structure pour une seule lettre de la grille.
  */
typedef struct LettreGrille {
	bool visite;
	char lettre;
} LettreGrille;

/**
  * @brief Structure permettant de déclarer une position dans la grille.
  */
struct PositionGrille {
	int x = 0;
	int y = 0;
};

/*
* @brief Structure présentant un tableau deux dimensions pour la grille.
*/
struct Grille {
	LettreGrille grille[4][4];
};

/**
 * @brief Calcul des points "boggle" d'un seul item.
 * @param[in] it L'item.
 * @return Le nombre de points.
 */
unsigned int calculPoint(const Item it);

/**
 * @brief Calcul des points "boggle" d'une liste entière.
 * @param[in] l La liste.
 * @return Le nombre de points.
 */
unsigned int calculPointGlobal(const Liste& l);

/**
 * @brief Création d'une grille avec une liste de départ (4x4) tapée au clavier.
 * @return Une grille (un tableau deux dimensions contenu dans la structure).
 */
Grille createGrille();

/**
 * @brief Vérifie si une position est bien dans les limites d'une grille boggle.
 * @param[in] coord La position d'une grille.
 * @return Un booléen: 1 indique que la position est dans les limites, sinon 0.
 */
bool checkLimites(const PositionGrille& coord);
```

```cpp
/**
 * @brief Recherche un Item (mot) dans une grille donnée.
 * @param[in,out] grille La grille de lettres.
 * @param[in] it L'item (mot) à rechercher dans la grille donnée.
 * @return Une grille (un tableau deux dimensions contenu dans la structure).
 */
bool rechercherMot(Grille& grille, Item& it);

/**
 * @brief Fonction de sous-recherche utilisée dans rechercheMot.
 * @param[in,out] grille La grille de lettres.
 * @param[in] it L'item (mot) à rechercher dans la grille donnée.
 * @return Une grille (un tableau deux dimensions contenu dans la structure).
 */
bool sousRecherche(Grille& grille, Item& it, int pos, PositionGrille& coord);
```

```cpp
/**
 * @file BoggleUtils.cpp
 * @brief Utilitaires pour la mise en oeuvre du boggle.
 */

#include <cstring>
#include <iostream>

#include "BoggleUtils.h"

unsigned int calculPoint(const Item it) {

	unsigned int motLength = (unsigned int)strlen(it.mot);

	if (motLength ⩽ 2) {
		return 0;
	}
	else if (motLength ⩽ 4) {
		return 1;
	}
	else if (motLength ⩽ 5) {
		return 2;
	}
	else if (motLength ⩽ 6) {
		return 3;
	}
	else if (motLength ⩽ 7) {
		return 5;
	}
	else if (motLength ⩾ 8) {
		return 11;
	}
	else {
		return 0;
	}
}

unsigned int calculPointGlobal(const Liste& l) {

	unsigned int total = 0;

	for (unsigned int i = 0; i < l.nb; ++i) {
		total += calculPoint(lire(l, i));
	}

	return total;
}

Grille createGrille() {

	Liste depart;
	Grille grille;
	PositionGrille pos;

	initialiser(depart, 1, 2);

	for (unsigned int i = 0; i < 4; ++i) {
		inserer(depart, depart.nb, saisie());
	}
```

```
	for (pos.x = 0; pos.x < 4; ++pos.x) {
		Item it = lire(depart, pos.x);
		for (pos.y = 0; pos.y < 4; ++pos.y) {
			LettreGrille m = { false, it.mot[pos.y] };
			grille.grille[pos.x][pos.y] = m;
		}
	}

	detruire(depart);

	return grille;

}

bool rechercherMot(Grille& grille, Item& it) {

	PositionGrille pos;
	for (pos.x = 0; pos.x < 4; ++pos.x) {
		for (pos.y = 0; pos.y < 4; ++pos.y) {
			grille.grille[pos.x][pos.y].visite = false;
		}
	}

	pos = { 0, 0 };

	for (pos.x = 0; pos.x < 4; ++pos.x) {
		for (pos.y = 0; pos.y < 4; ++pos.y) {
			if (sousRecherche(grille, it, 0, pos)) {
				return true;
			}
		}
	}

	return false;

}

bool checkLimites(const PositionGrille& coord) {
	return (coord.x > -1 && coord.y > -1 && coord.x < 4 && coord.y < 4);
}

bool sousRecherche(Grille& grille, Item& it, int pos, PositionGrille& coord) {

	if (pos ≥ strlen(it.mot)) {
		return true;
	}

	if (!checkLimites(coord)) {
		return false;
	}

	if (grille.grille[coord.x][coord.y].lettre ≠ it.mot[pos]) {
		return false;
	}

	if (grille.grille[coord.x][coord.y].visite == true) {
		return false;
	}
```

```
	grille.grille[coord.x][coord.y].visite = true;

	PositionGrille pCoord[8];
	PositionGrille tempCoord;
	unsigned int nbCoord = 0;

	tempCoord = { coord.x - 1, coord.y - 1 };
	if (checkLimites(tempCoord)) {
		pCoord[nbCoord++] = { tempCoord.x, tempCoord.y };
	}

	tempCoord = { coord.x - 1, coord.y };
	if (checkLimites(tempCoord)) {
		pCoord[nbCoord++] = { tempCoord.x, tempCoord.y };
	}

	tempCoord = { coord.x - 1, coord.y + 1 };
	if (checkLimites(tempCoord)) {
		pCoord[nbCoord++] = { tempCoord.x, tempCoord.y };
	}

	tempCoord = { coord.x, coord.y - 1 };
	if (checkLimites(tempCoord)) {
		pCoord[nbCoord++] = { tempCoord.x, tempCoord.y };
	}

	tempCoord = { coord.x, coord.y + 1 };
	if (checkLimites(tempCoord)) {
		pCoord[nbCoord++] = { tempCoord.x, tempCoord.y };
	}

	tempCoord = { coord.x + 1, coord.y - 1 };
	if (checkLimites(tempCoord)) {
		pCoord[nbCoord++] = { tempCoord.x, tempCoord.y };
	}

	tempCoord = { coord.x + 1, coord.y };
	if (checkLimites(tempCoord)) {
		pCoord[nbCoord++] = { tempCoord.x, tempCoord.y };
	}

	tempCoord = { coord.x + 1, coord.y + 1 };
	if (checkLimites(tempCoord)) {
		pCoord[nbCoord++] = { tempCoord.x, tempCoord.y };
	}

	for (unsigned int i = 0; i < nbCoord; ++i) {
		if (sousRecherche(grille, it, pos + 1, pCoord[i])) {
			return true;
		}
	}

	grille.grille[coord.x][coord.y].visite = false;
	return false;

}
```

```cpp
#pragma once

/**
 * @file ConteneurListe.h
 * @brief Composant d'un conteneur de listes à capacité extensible.
 */

#include "Liste.h"

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

 /**
  * @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 clistes_init(ConteneurListe& cl, 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 clistes_detruire(ConteneurListe& cl);

/**
 * @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
 */
Liste clistes_lire(const ConteneurListe& cl, 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 ajouter/modifier l'item.
 * @param[in] it L'item à écrire.
 */
void clistes_ecrire(ConteneurListe& cl, unsigned int i, const Liste& l);
```

```cpp
/**
 * @brief Rassemble tous les éléments de listes en une seule.
 * @param[in,out] c Le conteneur de listes.
 * @return Une liste contenant tous les items des listes du conteneur de listes.
 */
Liste clistes_join(const ConteneurListe& cl);
```

```cpp
/**
 * @file ConteneurListe.cpp
 * @brief Composant de conteneur de listes de capacité extensible.
 */

#include <iostream>
#include <cassert>
using namespace std;

#include "ConteneurListe.h"

void clistes_init(ConteneurListe& cl, unsigned int capa, unsigned int p) {
	assert((capa > 0) && (p > 0));
	cl.capacite = capa;
	cl.pasExtension = p;
	cl.tab = new Liste[capa];
}

void clistes_detruire(ConteneurListe& cl) {
	delete[] cl.tab;
	cl.tab = NULL;
}

Liste clistes_lire(const ConteneurListe& cl, unsigned int i) {
	assert(i < cl.capacite);
	return cl.tab[i];
}

void clistes_ecrire(ConteneurListe& cl, unsigned int i, const Liste& it) {
	if (i ≥ cl.capacite) {
		unsigned int newTaille = (i + 1) * cl.pasExtension;
		Liste* newT = new Liste[newTaille];
		for (unsigned int i = 0; i < cl.capacite; ++i)
			newT[i] = cl.tab[i];
		delete[] cl.tab;
		cl.tab = newT;
		cl.capacite = newTaille;
	}
	cl.tab[i] = it;
	cl.nbListes++;
}

Liste clistes_join(const ConteneurListe& cl) {

	Liste l;
	initialiser(l, 1, 2);

	for (unsigned int i = 0; i < cl.nbListes; ++i) {
		Liste tempL = clistes_lire(cl, i);
		for (unsigned int j = 0; j < tempL.nb; ++j) {
			inserer(l, l.nb, tempL.c.tab[j]);
		}
	}

	return l;

}
```

```cpp
#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 tde_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 tde_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 tde_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 ajouter/modifier l'item.
 * @param[in] it L'item à écrire.
 */
void tde_ecrire(ConteneurTDE& c, unsigned int i, const Item& it);
```

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

#include <cassert>
using namespace std;

#include "ConteneurTDE.h"

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

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

void tde_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;
}
```

```h
#pragma once

/**
 * @file Item.h
 * @brief Utilitaires pour la manipulation des items.
 */

#include "Item_t.h"

/**
 * @brief Saisie d'un ?l?ment du flux clavier ? un Item.
 * @return L'item saisie.
 */
Item saisie();
```

```h
#pragma once

/**
 * @file Item_t.h
 * @brief Définition du type Item.
 */

struct Mot {
	char mot[31];
};

typedef Mot Item;
```

```cpp
/**
 * @file Item.cpp
 * @brief Utilitaires de manipulation des items.
 */

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

Item saisie() {
	Item it;
	std::cin >> it.mot;
	return it;
}
```

```cpp
#pragma once

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

#include "ConteneurTDE.h"

 /**
  * @brief Structure Liste.
  */
struct Liste {
       /// Conteneur mémorisant les éléments de la liste.
       ConteneurTDE c;
       /// Nombre d'éléments stockés dans la liste.
       unsigned int nb;
};

/**
 * @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, unsigned int capa, 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, 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, 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, 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, unsigned int pos);

/**
 * @brief Ordonner la liste alphabétiquement.
 * @param[in,out] l La liste.
 */
void ordonner(Liste& l);

/**
 * @brief Insérer les mots jusqu'à la fin d'une liste (mentionnée par une *).
 * @param[in,out] l La liste.
 * @pre longueur(l) > 0.
 */
void entreeEtoile(Liste& l);

/**
 * @brief Affiche le mot à la position pos de la liste.
 * @param[in] l La liste.
 * @param[in] pos La position de l'élément.
 */
void afficher(const Liste& l, unsigned int pos);

/**
 * @brief Affiche une liste et mentionne sa fin avec une "*".
 * @param[in] l La liste.
 */
void afficherListe(const Liste& l);

/**
 * @brief Compare deux listes et retourne une liste en fonction du filtre.
 * @param[in,out] l_1 La liste.
 * @param[in,out] l_2 La liste.
 * @param[in] filtre Fonction permettant de filtrer.
 * @return Une liste ayant appliqué le filtre donné.
 */
Liste filtrer(Liste& l_1, Liste& l_2, bool (*filtre)(Liste& l, Item& it));

/**
 * @brief Récupérer l'indice d'un item donné dans une liste l.
 * @param[in] l La liste.
 * @param[in] it L'item à retrouver.
 * @return L'index de l'item it dans la liste l.
 * @pre l.nb > 0
 */
unsigned int getByItem(const Liste& l, const Item& it);
```

```cpp
/**
 * @brief Transforme une liste donnée en liste canonique, c'est-à-dire:
 *     les mots sont triés par ordre alphabétique croissant
 *     chaque mot n'apparaît d'une unique fois
 * @param[in] l La liste.
 * @return La liste transformée.
 * @pre l.nb > 0
 */
Liste formeCanonique(Liste& l);

/**
 * @brief Compte le nombre de fois qu'un élément apparait dans la liste.
 * @param[in] l La liste.
 * @param[in] it L'item permettant la comparaison.
 * @return Le nombre d'occurences de l'item it dans la liste l.
 * @pre 0 ⩽ pos ⩽ longueur(l).
 */
unsigned int occurences(const Liste& l, const Item& it);
```

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

#include <cassert>
#include <cstring>
#include <iostream>

#include "Liste.h"

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

void detruire(Liste& l) {
	tde_detruire(l.c);
}

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

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

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

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

void entreeEtoile(Liste& l) {
	Item it;
	bool t = false;
	while (true) {
		it = saisie();
		if (strcmp(it.mot, "*") == 0) {
			break;
		}
		if (t) {
			t = true;
			inserer(l, 0, it);
		}
		else {
			inserer(l, l.nb, it);
		}
	}
```

```
}

void supprimer(Liste& l, unsigned int pos) {
	assert((l.nb≠0) && (pos<l.nb));
	l.nb--;
	for (unsigned int i=pos; i<l.nb; ++i)
		tde_ecrire(l.c, i, tde_lire(l.c,i+1));
}

void afficher(const Liste& l, unsigned int pos) {
	std::cout << lire(l, pos).mot << std::endl;
}

void afficherListe(const Liste& l) {
	for (unsigned int i = 0; i < l.nb; ++i) {
		afficher(l, i);
	}
	std::cout << "*" << std::endl;
}

Liste filtrer(Liste& l_1, Liste& l_2, bool (*filtre)(Liste& l, Item& it)) {
	Liste l;
	initialiser(l, 1, 2);

	for (unsigned int i = 0, pos = 0; i < l_2.nb; ++i) {
		if (filtre(l_1, l_2.c.tab[i])) {
			inserer(l, pos, l_2.c.tab[i]);
		}
	}

	return l;
}

void ordonner(Liste& l) {
	for (unsigned int j = 0; j < l.nb; ++j) {
		for (unsigned int i = 0; i < (l.nb - 1); ++i) {
			if (strcmp(l.c.tab[i].mot, l.c.tab[i + 1].mot) > 0) {
				Item tmp;
				strcpy_s(tmp.mot, l.c.tab[i].mot);
				strcpy_s(l.c.tab[i].mot, l.c.tab[i + 1].mot);
				strcpy_s(l.c.tab[i + 1].mot, tmp.mot);
			}
		}
	}
}

unsigned int getByItem(const Liste& l, const Item& it) {
	assert(l.nb > 0);

	for (unsigned int i = 0; i < l.nb; ++i) {
		if (strcmp(l.c.tab[i].mot, it.mot) == 0) {
			return i;
		}
	}

	return -1; // fallback
}
```

```
Liste formeCanonique(Liste& l) {
	Liste newL;
	initialiser(newL, 1, 2);

	for (unsigned int i = 0, ins = 0; i < l.nb; ++i) {
		if (getByItem(newL, l.c.tab[i]) == -1) {
			inserer(newL, ins, l.c.tab[i]);
			ins++;
		}
	}

	ordonner(newL);

	return newL;
}

unsigned int occurences(const Liste& l, const Item& it) {
	unsigned int compteur = 0;

	for (unsigned int i = 0; i < l.nb; ++i) {
		if (strcmp(l.c.tab[i].mot, it.mot) == 0) {
			compteur++;
		}
	}

	return compteur;
}
```

```cpp
/**
* Projet de SDA
* @file Source.cpp
* @author Antoine Banha <antoine@jiveoff.fr>
* @author Rayan Atrouni <rayan.atrouni@etu.u-paris.fr>
*/

#include <iostream>
#include <cassert>
#include <clocale>
#include <cstring>

#include "ConteneurListe.h"

#include "Liste.h"
#include "BoggleUtils.h"

using namespace std;

/*
* Tests unitaires
*/

void tests() {

	Liste l_1;
	initialiser(l_1, 1, 2);

	for (int i = 0; i < 10; ++i) {
		Item it;
		sprintf_s(it.mot, "test%d", i);
		inserer(l_1, longueur(l_1), it);
	}

	assert(longueur(l_1) == 10);
	assert(strcmp(lire(l_1, 3).mot, "test3") == 0);

	Item it_test;
	strcpy_s(it_test.mot, "test2");
	assert(getByItem(l_1, it_test) == 2);

	Liste l_2;

	initialiser(l_2, 1, 2);

	Item it;
	strcpy_s(it.mot, "test");

	for (int i = 0; i < 5; ++i) {
		inserer(l_2, longueur(l_2), it);
	}
	for (int i = 0; i < 4; ++i) {
		Item it_2;
		sprintf_s(it_2.mot, "test%d", i);
		inserer(l_2, longueur(l_2), it_2);
	}

	assert(occurences(l_2, it) == 5);
```

```
	auto filtre = [](Liste& l, Item& it) {
		int index = getByItem(l, it);
		return (index > -1);
	};

	assert(longueur(filtrer(l_1, l_2, filtre)) == 4);

}

/*
* Structure imposée
*/

void exo1() {

	Liste l;
	initialiser(l, 1, 2);

	entreeEtoile(l);

	cout << calculPointGlobal(l);

	detruire(l);

}

void exo2() {

	Liste l;
	initialiser(l, 1, 2);

	entreeEtoile(l);

	Liste newL = formeCanonique(l);
	detruire(l);

	afficherListe(newL);

	detruire(newL);

}

void exo3() {

	Liste tempL_1;
	initialiser(tempL_1, 1, 2);

	entreeEtoile(tempL_1);

	Liste l_1 = formeCanonique(tempL_1);
	detruire(tempL_1);

	Liste tempL_2;
	initialiser(tempL_2, 1, 2);

	entreeEtoile(tempL_2);
```

```
	Liste l_2 = formeCanonique(tempL_2);
	detruire(tempL_2);

	auto filtre = [](Liste& l, Item& it) {
		return (getByItem(l, it) == -1);
	};

	Liste l_filtre = filtrer(l_1, l_2, filtre);

	detruire(l_1);
	detruire(l_2);

	Liste l_canonique = formeCanonique(l_filtre);

	detruire(l_filtre);

	afficherListe(l_canonique);

	detruire(l_canonique);

}

void exo4() {

	Liste tempL_1;
	initialiser(tempL_1, 100, 200);

	entreeEtoile(tempL_1);

	Liste tempL_2;
	initialiser(tempL_2, 1, 2);

	entreeEtoile(tempL_2);

	Liste l_2 = formeCanonique(tempL_2);
	detruire(tempL_2);

	auto filtre = [](Liste& l, Item& it) {
		int index = getByItem(l, it);
		return (index > -1);
	};

	Liste l_filtre = filtrer(tempL_1, l_2, filtre);

	detruire(tempL_1);
	detruire(l_2);

	Liste l_canonique = formeCanonique(l_filtre);

	detruire(l_filtre);

	afficherListe(l_canonique);

	detruire(l_canonique);
```

```
}

void exo5() {

	ConteneurListe cl;
	clistes_init(cl, 1, 2);

	while (true) {

		Liste tempL;
		initialiser(tempL, 1, 2);

		entreeEtoile(tempL);

		if (longueur(tempL) == 0) {
			detruire(tempL);
			break;
		}

		Liste lCanonique = formeCanonique(tempL);
		detruire(tempL);

		clistes_ecrire(cl, cl.nbListes, lCanonique);

	}

	Liste l = clistes_join(cl);

	Liste lFinal;
	initialiser(lFinal, 1, 2);

	for (unsigned int i = 0; i < longueur(l); ++i) {
		if (occurences(l, l.c.tab[i]) ⩾ 2) {
			inserer(lFinal, longueur(lFinal), l.c.tab[i]);
		}
	}

	detruire(l);

	Liste lCanonique = formeCanonique(lFinal);
	detruire(lFinal);

	afficherListe(lCanonique);

	detruire(lCanonique);

	clistes_detruire(cl);

}

void exo6() {

	Grille grille = createGrille();

	Liste l;
	initialiser(l, 100, 200);

	entreeEtoile(l);
```

```
        for (unsigned int i = 0; i < longueur(l); ++i) {
                if (rechercherMot(grille, l.c.tab[i])) {
                        afficher(l, i);
                }
        }

        std::cout << "*";

}

int main() {

        setlocale(LC_ALL, "");

        tests();

        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;
        }

        return 0;
}
```

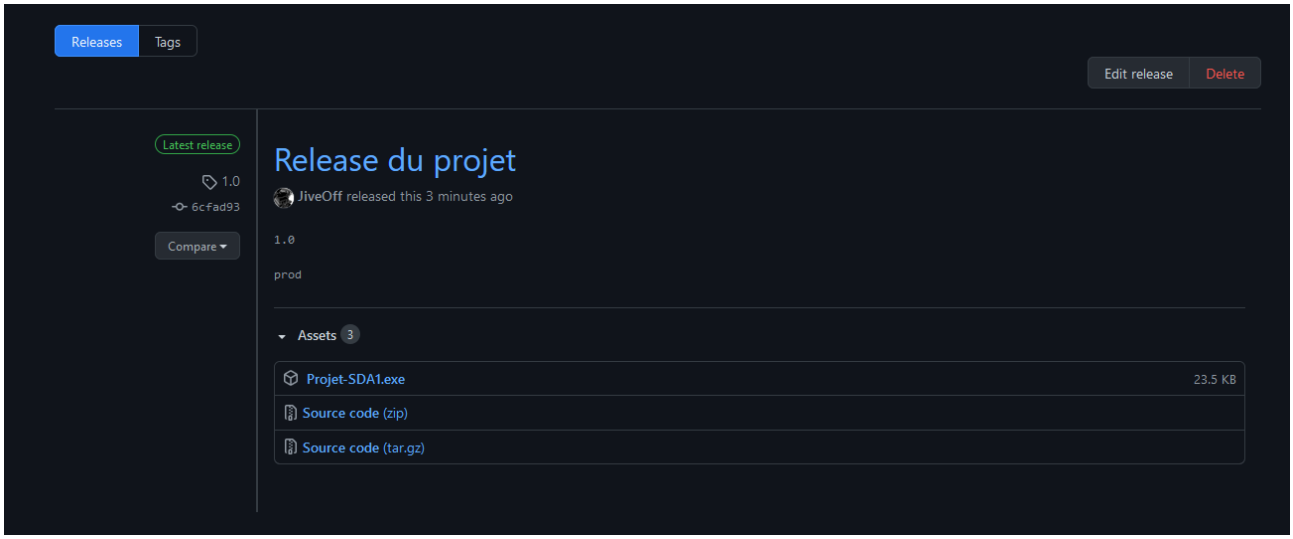
GitHub du projet

Lien vers notre GitHub de projet :

https://github.com/JiveOff/Projet-SDA1

(sera disponible le jour du rendu)

Le GitHub du projet contient également une release .exe du projet.



Fin du rapport, merci d'avoir lu !

Université de Paris