Rapport de programmation impérative

Projet - Le boss de fin

SAVOURÉ Florian / DELAHAYE Antoine 15/11/2021

Organisation du travail

Pour ce projet, nous avons décidé d'utiliser "Code with me", un outil permettant de se connecter sur l'IDE de son binôme et travailler en même temps sur la même base de code. La principale raison du choix de cet outil est que, accompagné de Discord, nous avons pu poser, répondre mutuellement à nos questions et prendre des décisions sur la structure du code. Nous avons organisé des moments dans la journée (surtout pendant les vacances) où nous nous retrouvions pour continuer le projet. Nous avons divisé les tâches autant que possible, par exemple, si l'un travaillait sur push() l'autre travaillait sur pop().

Exercice 1

Tout d'abord, pour la structure, nous avons décidé d'y inclure la taille du tableau, la taille du type, ainsi que 3 pointeurs de fonctions (afficher, aléatoire et détruire) qui nous servirons à appeler les fonctions correspondantes au type dans le tableau et bien sûr le tableau en lui-même.

La création de tableau est relativement simple, il nous faut allouer la structure en elle-même, puis le tableau générique et enfin assigner les différents paramètres nécessaires au remplissage de la structure, à savoir : la taille du type, le nombre d'éléments qu'il contient, ainsi que les 3 pointeurs de fonctions.

Afin d'afficher les valeurs après la création de la structure, il nous fallait définir une fonction qui va prendre en paramètre la structure tableau. Nous avons déterminé l'adresse du premier et dernier élément afin de pouvoir itérer avec un pas de taille_type. Pour calculer le décalage de n octets dans la mémoire afin de déterminer l'adresse du dernier élément, nous avons dû faire un pointeur de char temporaire qui est cast, à l'appel de la fonction affiché en, void *. À chaque tour de boucle, la fonction afficher pointée dans la structure est appelée afin d'afficher le type correctement.

Concernant la génération aléatoire, la fonction prend les mêmes paramètres que la fonction créer(). On va donc commencer par créer une structure tableau vide avec n places et assigner une valeur aléatoire à chaque place grâce à l'appel de fonction aléatoire correspondante au type. Pour calculer l'adresse de fin nous avons dû avoir recours à un pointeur temporaire afin de faire les opérations arithmétiques sur l'adresse de début tout comme l'affichage.

À la fin du programme, il faut libérer les structures précédemment allouées, on va donc les détruire, pour cela, la fonction détruire_tout s'occupe d'appeler la fonction pointée dans la structure correspondant à la destruction, ce qui aura pour effet de free() et de mettre à NULL le tableau générique, ensuite détruire_tout s'occupera de free() la structure en elle-même et la mettre, elle aussi, à NULL. Ainsi, en mettant les pointeurs à NULL, la mémoire est libérée correctement et proprement.

La fonction la plus simple à faire a été la fonction de tri, il nous a juste fallu utiliser qsort() et une fonction de comparaison.

Pour les fonctions dans le main_exercice1.c, nous avons toujours un pointeur générique en paramètre et pour chaque fonction, son fonctionnement propre à son type, par exemple pour aleatoire_struct() chaque élément (a, b et c) a une valeur attribué l'un après l'autre tandis que pour aleatoire_int(), on attribue juste un nombre entre 0 et INT_MAX (le nombre maximal pour un entier signé). Pour ce qui est de l'affichage, nous avons juste à cast le pointeur générique dans le type indiqué par le nom de la fonction et ensuite faire appel à printf() pour l'afficher. Les fonctions de destruction ne sont pas vraiment utiles, car on a juste à free() le pointeur générique et le mettre à NULL. On pourrait migrer ces fonctions dans detruire_tout() mais nous l'avons gardé pour les besoins du projet.

Exercice 2

Pour cet exercice, nous avons repris le code de l'exercice 1 et nous avons construit par-dessus. Concernant la structure, nous avons rajouté la taille maximale du tableau afin de pouvoir réallouer de la mémoire si besoin.

Pour la création de notre tableau variable, nous avons remplacé dans les paramètres de la fonction le nombre d'éléments dans le tableau par la taille maximale (car le tableau est par défaut vide). Nous avons aussi ajouté une fonction check_ptr() permettant de vérifier si un pointeur à bien été alloué ou réalloué, nous verrons plus par la suite son utilité pour push().

La fonction afficher_var() reste la même qu'afficher() dans l'exercice 1.

La génération aléatoire crée un tableau variable avec une taille max calculée en fonction de la taille du type. La taille max est un multiple de 1024 octets et si par exemple la taille du type est de 1030 octets, la taille max sera de 2048 octets, car on

détermine la place que prend le type et on alloue en conséquence la mémoire nécessaire. Bien sûr, dans le cas précédent, il faudra réallouer à chaque push dans le tableau, cependant pour la plupart des types 1024 sera suffisant pour ne pas réallouer constamment tout en restant un incrément relativement petit. Le reste de la logique reste ensuite la même que pour l'exercice précédent.

Pour ajouter un élément à la fin de notre tableau, nous utilisons push() qui prend en paramètre notre tableau et la valeur à ajouter. Lors de l'insertion, on regarde d'abord s'il y a de la place dans le tableau, si oui, alors on fait un memcpy() de notre valeur à l'adresse du dernier élément + la taille du type pour avoir l'adresse de destination. Si non, on réalloue 1024 octets de plus jusqu'à ce que le prochain élément rentre. Ensuite, si ce n'est pas le cas, on le memcpy() comme indiqué précédemment. À chaque réallocation, on prend aussi soin de regarder si l'adresse est valide, sinon le programme exit avec un message d'erreur.

Si on veut push à un indice précis, on peut utiliser push_indice(). Il va copier les valeurs après l'indice dans un endroit temporaire, insérer la valeur passée en paramètre et coller ce qui a été copié à la suite de la valeur insérée. La gestion mémoire reste la même que pour push().

Pour retirer et retourner le dernier élément, pop() va s'occuper de le faire. On va premièrement allouer l'adresse que l'on va renvoyer, copier le dernier élément dedans, mettre à 0 les bits du dernier élément dans le tableau pour le supprimer et ensuite décrémenter de 1 le nombre d'éléments. On regarde également si le tableau a été agrandi, et qu'il y a eu beaucoup d'élément de pop, alors on réduit la taille du tableau (si la taille max est de 2048 et que l'espace utilisé est de taille max - (1024 + 512) alors on realloc() de taille max - 1024, ce qui laisse donc 512 octets de libre encore dans le tableau, cela permet de ne pas realloc() constamment). Enfin on retourne l'adresse de la dernière valeur.

La fonction pop_indice() suit la même logique que pop() pour la gestion mémoire. Pour supprimer un élément, nous avons juste à copier les valeurs à l'adresse de l'indice, et les décaler de taille type à gauche, de ce fait, on va réécrire par-dessus l'élément que l'on veut supprimer. On remet à 0 les bits à la fin du tableau afin de supprimer le surplus et on retourne l'adresse du dernier élément.

Exercice 3

Pour connaître l'élément maximum entre 2 valeurs générique, nous pouvons utiliser memcmp() qui compare directement les bits entre eux, et retourne la différence. Avec cette fonction, on peut donc savoir quelle est la valeur la plus grande sans connaître son type. On va donc parcourir tout le tableau et comparer la valeur maximale avec la valeur actuelle. Finalement l'adresse de la valeur maximale est retournée.

La fonction slice() prend un paramètre 2 entiers n et m et la structure tableau, et calcule les adresses à l'indice n et m et copie cette zone mémoire dans un nouveau tableau variable. On applique ensuite le nombre d'éléments à n - m et on retourne l'adresse du tableau fraîchement créé.

Pour finir, filtrer() il prend un tableau variable en paramètre et un pointeur de fonction de prédicat. On commence par créer un nouveau tableau variable, on parcourt le tableau et on applique le prédicat sur la valeur actuelle. Nous avons codé isOdd() et isEven() qui permet de savoir si une valeur est respectivement impaire ou pair. Si le prédicat sur la valeur est vrai, alors elle est copiée à la suite dans le tableau générique. L'adresse du tableau filtré est ensuite retourné.