Push_swap

Parce que Swap_push ne semble pas aussi naturel

Résumé :
Cette activité vous demandera de trier des données empilées, à l'aide d'un nombre limité d'instructions et en effectuant le moins d'actions possible. Pour réussir, vous devrez manipuler différents types d'algorithmes et choisir la solution la plus appropriée (parmi plusieurs) pour un tri de données optimisé.
Il s'agit d'une activité de groupe à réaliser par exactement 2 apprenants.

Version : 1.0

Contenu

Chapitre I

Avant-propos

Il était une fois, dans les contrées mystérieuses de l'informatique, un certain Donald Knuth1 J.-C. a popularisé la notation Big-O pour nous aider à décrire la scalabilité des algorithmes. Big-O est une façon polie de dire : « Votre code s'exécutera soit rapidement indéfiniment… soit tellement lentement que vous aurez le temps de vous faire un café, de le boire et de vieillir en attendant. »

La légende raconte qu'à l'époque, les programmeurs ignoraient la notation Big-O. Ils exécutaient simplement leur code et, si cela prenait trop de temps, ils incriminaient le matériel. Puis Big-O est arrivé et a tout gâché en démontrant mathématiquement que parfois, un algorithme est tout simplement mauvais , même avec une puissance de calcul incroyable.

Pourquoi est-ce important pour push_swap ? Parce que vous vous retrouvez avec deux piles et un nombre limité de mouvements, essayant de trier des nombres plus rapidement qu'un tri par insertion en panne de sommeil. La notation Big O vous aidera à affronter la dure réalité : une stratégie brillante en O(n log n) sera toujours plus performante que votre stratégie maladroite en $O(n^2)$ lorsque la taille des données augmente… sauf, bien sûr, si vous choisissez d'ignorer les calculs et de laisser le nombre d'opérations exploser.

Alors, lorsque vous concevez vos algorithmes, souvenez-vous : la notation Big-O n'est pas là pour vous faire peur, mais pour vous éviter de devenir la personne qui écrit :

pb; pa; pb; pa; pb; pa; ...

10 000 fois de suite et s'étonne ensuite que le vérificateur les déteste. Triez intelligemment, pas lentement.

Big-O n'est pas une échelle de taille comparable à celle du pop-corn... mais si c'était le cas, O(n log n) resterait la valeur idéale.

1Oui, le type à la barbe magnifique et à la patience infinie.

Chapitre II

Instructions communes

• Votre projet doit être écrit en C.

Votre projet doit être conforme à la norme. Si vous utilisez des fichiers ou des fonctions supplémentaires, ils seront pris en compte lors de la vérification de la norme ; en cas d'erreur, vous obtiendrez la note de 0.

Vos fonctions ne doivent pas s'arrêter inopinément (erreur de segmentation, erreur de bus, double libération, etc.), sauf en cas de comportement indéfini. Dans le cas contraire, votre projet sera considéré comme non fonctionnel et recevra la note de 0 lors de l'évaluation.

• Toute la mémoire allouée sur le tas doit être correctement libérée lorsque cela est nécessaire. Fuites de mémoire ne sera pas toléré.

• Si le sujet l'exige, vous devez soumettre un Makefile qui compile vos fichiers sources en sortie requise avec les options -Wall, -Wextra et -Werror, en utilisant cc.
De plus, votre Makefile ne doit pas effectuer de reliaison inutile.

• Votre Makefile doit contenir au moins les règles $(NAME), all, clean, fclean et
concernant.

Pour soumettre des éléments bonus pour votre projet, vous devez inclure une règle bonus dans votre Makefile. Cette règle ajoutera tous les en-têtes, bibliothèques ou fonctions non autorisés dans la partie principale du projet. Les éléments bonus doivent être placés dans des fichiers _bonus.{c/h}, sauf indication contraire du sujet. L'évaluation des parties obligatoires et des éléments bonus est effectuée séparément.

Si votre projet vous permet d'utiliser libft, vous devez copier ses sources et son Makefile associé dans un dossier libft. Le Makefile de votre projet doit compiler la bibliothèque, puis compiler le projet.

Nous vous encourageons à créer des programmes de test pour votre projet, même si ce travail n'a pas à être soumis et ne sera pas noté. Cela vous permettra de tester facilement votre travail et celui de vos pairs. Ces tests vous seront particulièrement utiles lors de votre soutenance. D'ailleurs, lors de celle-ci, vous êtes libre d'utiliser vos tests et/ou ceux du pair que vous évaluez.

• Déposez votre travail dans le dépôt Git indiqué. Seul le travail présent dans le dépôt Git sera évalué. Si Deepthought est désigné pour évaluer votre travail, cela se produira.

Après vos évaluations par les pairs. Si une erreur survient dans une section quelconque de votre travail lors de la notation par Deepthought, l'évaluation sera interrompue.

Chapitre III

Instructions d'IA

• Context

Tout au long de votre apprentissage, l'IA peut vous assister dans de nombreuses tâches. Prenez le temps d'explorer les différentes fonctionnalités des outils d'IA et comment ils peuvent vous aider dans votre travail. Toutefois, abordez-les toujours avec prudence et évaluez les résultats de manière critique. Qu'il s'agisse de code, de documentation, d'idées ou d'explications techniques, vous ne pouvez jamais être absolument certain que votre question était bien formulée ni que le contenu généré est exact. Vos pairs constituent une ressource précieuse pour vous aider à éviter les erreurs et les angles morts.

• Message principal

Utilisez l'IA pour réduire les tâches répétitives ou fastidieuses.

Développez des compétences en matière d'incitation — avec ou sans codage — qui vous seront utiles.
carrière future.

Découvrez comment fonctionnent les systèmes d'IA pour mieux anticiper et éviter les risques, les biais et les problèmes éthiques courants.

Continuez à développer vos compétences techniques et relationnelles en travaillant avec vos pairs.

N'utilisez que du contenu généré par l'IA que vous comprenez parfaitement et dont vous pouvez assumer la responsabilité.
pour.

• Règles de l'apprenant :

- Vous devriez prendre le temps d'explorer les outils d'IA et de comprendre leur fonctionnement, afin de pouvoir les utiliser de manière éthique et réduire les biais potentiels.
- Vous devriez réfléchir à votre problème avant de demander de l'aide — cela vous aidera à écrire plus clairement.
  Des consignes plus détaillées et plus pertinentes, utilisant un vocabulaire précis.
- Vous devriez prendre l'habitude de vérifier, d'examiner, de questionner et de tester systématiquement tout ce qui est généré par l'IA.
- Vous devriez toujours solliciter une évaluation par les pairs — ne vous fiez pas uniquement à votre propre validation.

• Résultats de la phase :

• Développer des compétences d'incitation à la fois générales et spécifiques au domaine.

• Améliorez votre productivité grâce à une utilisation efficace des outils d'IA.

• Continuer à renforcer la pensée computationnelle, la résolution de problèmes, l'adaptabilité et collaboration.

• Commentaires et exemples :

Vous serez régulièrement confronté à des situations (examens, évaluations, etc.) où vous devrez démontrer une réelle compréhension. Préparez-vous et continuez à développer vos compétences techniques et relationnelles.

• Expliquer son raisonnement et débattre avec ses pairs révèle souvent des lacunes dans ses connaissances. Compréhension. Faites de l'apprentissage entre pairs une priorité.

Les outils d'IA manquent souvent de contexte spécifique et tendent à fournir des réponses génériques. Vos pairs, qui partagent votre environnement, peuvent vous apporter des informations plus pertinentes et précises.

• Là où l'IA tend à générer la réponse la plus probable, vos collègues peuvent apporter des perspectives alternatives et des nuances précieuses. Fiez-vous à eux pour un contrôle qualité.

Bonne pratique :

> Je demande à l'IA : « Comment tester une fonction de tri ? » Elle me suggère quelques pistes. Je les teste et j'en discute avec un collègue. Nous affinons ensuite la méthode ensemble.

Mauvaise pratique :

> Je demande à une IA d'écrire une fonction complète, puis je la copie-colle dans mon projet. Lors de l'évaluation par les pairs, je suis incapable d'expliquer son fonctionnement ni sa raison d'être. Je perds toute crédibilité et mon projet échoue.

Bonne pratique :

> J'utilise l'IA pour concevoir un analyseur syntaxique. Ensuite, je passe en revue la logique avec un collègue. Nous repérons deux bogues et réécrivons le code ensemble : il est meilleur, plus clair et parfaitement compréhensible.

Mauvaise pratique :

> J'ai laissé Copilot générer le code d'une partie essentielle de mon projet. La compilation réussit, mais je ne parviens pas à expliquer comment il gère les pipes. Lors de l'évaluation, je ne peux pas justifier mon erreur et mon projet échoue.

Chapitre IV

Introduction

Le projet Push Swap est un projet algorithmique très simple et direct : les données doivent être triées.

Vous disposez d'un ensemble de valeurs entières, de 2 piles et d'un ensemble d'opérations pour manipuler les deux piles.

Votre objectif ? Écrire un programme C appelé push_swap qui calcule et affiche sur la sortie standard le plus petit programme, composé d' opérations du langage push swap , qui trie les entiers reçus en arguments.

Facile?

On verra...

Chapitre V

Objectifs

L'objectif de ce projet est de vous faire découvrir la complexité algorithmique. de manière très concrète.

Trier des nombres est facile ; les trier rapidement avec seulement deux piles et un nombre limité de mouvements autorisés est une autre affaire. Trier une liste totalement aléatoire et trier une liste presque triée sont également deux choses extrêmement différentes.

Vous constaterez rapidement comment le choix de l'algorithme peut faire la différence entre une victoire rapide et un défilement interminable d'opérations.

Chapitre VI

Partie obligatoire

VI.1 Exigences du projet de groupe

- Ce projet doit être réalisé par exactement 2 apprenants travaillant ensemble.
- Les deux apprenants doivent contribuer de manière significative au projet et comprendre tous les aspects. algorithmes complétés.
- Le dépôt doit clairement indiquer les contributions des deux apprenants dans le fichier README.md. déposer.
- Lors de la soutenance, les deux apprenants doivent être présents et capables d'expliquer n'importe quelle partie du sujet. le code.
- Le dossier de projet doit inclure les identifiants de connexion des deux apprenants dans le dépôt.

VI.2 Les règles

- Vous avez 2 piles nommés a et b.
- Au début :
  - La pile a contient un nombre aléatoire de nombres négatifs et/ou positifs sans doublons. ◦ La pile b est vide.
- L'objectif est de trier les nombres par ordre croissant dans la pile a. Pour ce faire, vous disposez des opérations suivantes :

  sa (échanger a) : Échange les deux premiers éléments en haut de la pile a.
  Ne rien faire s'il n'y a qu'un seul élément ou aucun élément.

  sb (échange b) : Échange les deux premiers éléments en haut de la pile b.
  Ne rien faire s'il n'y a qu'un seul élément ou aucun élément.

  ss : sa et sb en même temps.

  pa (pousser a) : Prenez le premier élément en haut de b et placez-le en haut de a.
  Ne rien faire si b est vide.

pb (push b) : Prenez le premier élément en haut de a et placez-le en haut de b.
Ne rien faire si a est vide.

ra (rotation de a) : Décaler tous les éléments de la pile a d'une position vers le haut.
Le premier élément devient le dernier.

rb (rotation de b) : Décale tous les éléments de la pile b d'une position vers le haut.
Le premier élément devient le dernier.

rr : ra et rb en même temps.

rra (rotation inverse de a) : Décale tous les éléments de la pile a d'une position vers le bas.
Le dernier élément devient le premier.

rrb (rotation inverse de b) : Décale tous les éléments de la pile b d'une position vers le bas.
Le dernier élément devient le premier.

rrr : rra et rrb en même temps.

VI.3 Exigences relatives à l'algorithme

Pour bien comprendre la complexité algorithmique (temporelle et spatiale), vous devez implémenter quatre stratégies de tri distinctes et les intégrer à votre programme push_swap. Votre programme doit pouvoir sélectionner une stratégie à l'exécution en fonction de la configuration d'entrée.

VI.3.1 Modèle de complexité et contraintes de fonctionnement

Toutes les stratégies sont implémentées en C et doivent générer des séquences d'opérations Push_swap pour effectuer le tri. Cela signifie :

- Vos algorithmes C analysent l'entrée et génèrent la séquence appropriée de opérations : sa, sb, ss, pa, pb, ra, rb, rr, rra, rrb, rrr.
- Le résultat de votre stratégie est la séquence de ces opérations qui trie la pile.
- Lorsque vous indiquez une classe de complexité, celle-ci doit refléter le coût mesuré en nombre d'opérations Push_swap générées, et non la complexité théorique d'un algorithme classique basé sur un tableau.

VI.3.2 Métrique de désordre (obligatoire)

Dans ce sujet, le désordre est un nombre entre 0 et 1 qui indique à quel point votre pile initiale a est éloignée d'être triée.

Si les nombres sont déjà dans le bon ordre, le désordre est de 0. S'ils sont dans le pire ordre possible, le désordre est de 1. Toute valeur intermédiaire signifie que votre pile est partiellement triée, mais encore désordonnée.

Pour le calculer, on peut imaginer examiner toutes les paires de nombres possibles dans la pile.
Chaque fois qu'un nombre plus grand précède un nombre plus petit, cette paire est considérée comme une erreur.
Plus vous faites d'erreurs, plus le désordre se rapproche de 1.

```
fonction calculer_désordre(pile a) :
    erreurs = 0
    total_pairs = 0 pour i
    de 0 à taille(a)-1 :
        pour j de i+1 à taille(a)-1 : total_pairs += 1 si a[i]
            > a[j] : erreurs += 1 retourner
            erreurs / total_pairs
```

Vous devez mesurer le désordre avant d'entreprendre toute action.

VI.3.3 Stratégies requises

1. Algorithme simple (O(n ^{2})):
   Implémentez au moins un algorithme de base en O(n). ^{2}) classe. Exemples :
   - Adaptation au tri par insertion
   - Adaptation du tri par sélection
   - Adaptation du tri à bulles
   - Méthodes d'extraction min/max simples

2. Algorithme de complexité moyenne (O(n $\sqrt{n}$)) :
   Implémentez au moins un algorithme de complexité O(n $\sqrt{n}$). Exemples :
   - Tri par blocs (diviser en $\sqrt{n}$ blocs)
   - Méthodes de partitionnement par blocs
   - Adaptations du tri par compartiments avec $\sqrt{n}$ compartiments
   - Stratégies de tri basées sur l'étendue

3. Algorithme complexe (O(n log n)) :
   Implémentez au moins un algorithme de complexité O(n log n). Exemples :
   - Adaptation du tri Radix (LSD ou MSD)
   - Adaptation du tri fusion utilisant deux piles
   - Adaptation rapide du tri avec partitionnement de pile
   - Adaptation du tri par tas
   - Approches par arbres binaires indexés

4. Algorithme adaptatif personnalisé (conception par l'apprenant) : Concevez une stratégie adaptative qui sélectionne différentes méthodes internes en fonction du désordre mesuré. Vous n'êtes pas limité à un algorithme spécifique ; les techniques internes sont entièrement à votre discrétion. Cependant, votre conception doit respecter les objectifs de complexité suivants par régime (dans le modèle d'opération Push_swap) :

   Faible désordre : si le désordre < 0,2, la méthode choisie doit s'exécuter en temps O(n) .

   Désordre moyen : si 0,2 ≤ désordre < 0,5, la méthode choisie doit s'exécuter en O(n $\sqrt{n}$) temps.

   Niveau de désordre élevé : si le désordre est supérieur ou égal à 0,5, la méthode choisie doit s'exécuter en O(n log n). temps.

   Vous devez documenter dans votre dépôt (par exemple, README.md) la justification de vos seuils, les techniques internes utilisées dans chaque régime et un bref argument de complexité (limites supérieures) pour le temps et l'espace dans le modèle Push_swap.

VI.4 Exemple

Pour illustrer l'effet de certaines de ces opérations, trions une liste aléatoire d'entiers. Dans cet exemple, nous supposerons que les deux piles croissent à partir de la droite.

```
---------------------------------------------------------------------------------
Initialiser a et b :
2
1
3
6
5
8

--
ab
---------------------------------------------------------------------------------
Exécutif : 1

2
3
6
5
8

--
ab
---------------------------------------------------------------------------------
Exécutif pb pb pb : 6 3
5 2

8 1

--
ab
---------------------------------------------------------------------------------
Exec ra rb (équiv. à rr) : 5 2

8 1
6 3

--
ab
---------------------------------------------------------------------------------
Exécuter rra rrb (équivalent à rrr) :
6 3
5 2
8 1

--
ab
---------------------------------------------------------------------------------
Sonia directeur/directrice exécutif(ve) :
5 3
6 2
8 1

--
ab
---------------------------------------------------------------------------------
Exécuter l'étape étape :

1 2
3
5
6
8

--
ab
---------------------------------------------------------------------------------
```

Les entiers de la pile a sont triés en 12 opérations. Pouvez-vous faire mieux ?

VI.5 Le programme « push_swap »

Nom du programme : push_swap Fichiers	
à soumettre : Makefile, *.h, *.c NAME, all, clean, fclean,	
Makefile	re stack a : Une liste d'entiers
Arguments	
Fonction externe	• lire, écrire, allouer, libérer, sortie • ft_printf ou toute fonction équivalente VOUS avez codé
Libft autorisé	Oui
Description	Trier les piles

Votre projet doit respecter les règles suivantes :

• Vous devez soumettre un Makefile qui compilera vos fichiers sources. Il ne doit pas reconnecter.

• Les variables globales sont interdites.

• Vous devez écrire un programme nommé push_swap qui prend comme arguments :

◦ La pile est formatée comme une liste d'entiers (le premier argument est le sommet de la pile). empiler).

◦ Un sélecteur de stratégie optionnel :

--simple Force l'utilisation de votre algorithme en $O(n^2)$ algorithme. $O(n)$. --medium Force l'utilisation de votre algorithme en $O(n \sqrt{n})$. -- complex Force l'utilisation de votre algorithme en $O(n \log n)$. --adaptive Force l'utilisation de votre algorithme adaptatif basé sur le désordre. Il s'agit du comportement par défaut si aucun sélecteur n'est fourni.

• Le programme doit afficher la liste la plus courte possible des opérations Push_swap. Trier la pile a, le plus petit nombre étant en haut.

• Les opérations doivent être séparées par un \n et rien d'autre.

• La classe de complexité revendiquée pour chaque algorithme doit être valide dans ce modèle.

• La sélection de stratégie doit fonctionner pour toutes les entrées valides. Tout indicateur de sélection doit fonctionner quelle que soit la taille ou le désordre des entrées.

• Si aucun paramètre n'est spécifié, le programme ne doit rien afficher et donner le résultat suivant : réponse rapide.

- En cas d'erreur, il doit afficher « Erreur » suivi d'un \n sur la sortie d'erreur standard.
  Les erreurs peuvent inclure, par exemple : des arguments qui ne sont pas des entiers, des entiers en dehors de la plage valide ou des valeurs en double.

Votre programme binaire doit intégrer les quatre stratégies (Simple O(n), ^{2}), Moyen O(n $\sqrt{}$ n), Com-
Complex O(n log n) et Adaptative). Le nom et la classe de complexité de la stratégie sélectionnée doivent être disponibles en mode --bench.

- Le mode de test de performance optionnel (--bench) doit s'afficher après le tri :
  - Le désordre calculé (% avec deux décimales).
  - Le nom de la stratégie utilisée et sa classe de complexité théorique.
  - Le nombre total d'opérations.
  - Le nombre de chaque type d'opération (sa, sb, ss, pa, pb, ra, rb, rr, rra, rrb, rrr).

Les résultats du test de performance doivent être envoyés à la sortie d'erreur standard et n'apparaître que lorsque l'indicateur est présent.

VI.5.1 Exemples d'utilisation

Notation : les lignes préfixées par [bench] représentent les messages affichés par le mode de test d'évaluation optionnel (sur la sortie d'erreur standard). Le flux d'opérations reste sur la sortie standard.

```
$>./push_swap 2 1 3 6 5 8
jour
pb
rra
pb
pb
jour
pb
jour
pb
pb
au
revoir
au
revoir
au
revoir au revoir
```

Sélection par défaut (--adaptative) et nombre d'opérations :

```
$> ARG="4 67 3 87 23"; ./push_swap --adaptive $ARG | wc -l 13
```

Forcer la stratégie simple (O(nˆ2)) :

```
$> ./push_swap --simple 5 4 3 2 1
rra
pb
rra
pb
rra
pb
jour
pb
pb
au
revoir
au
revoir
au revoir
```

Forcer la stratégie complexe (O(n log n)) et vérifier avec le vérificateur :

```
$> ARG="4 67 3 87 23"; ./push_swap --complex $ARG | ./checker_linux $ARG
excores
```

push_swap avec une entrée de grande taille :

```
$> shuf -i 0-9999 -n 500 > args.txt ; ./push_swap $(cat args.txt) | wc -l 6784 $>
```

Exécuter avec l'analyse comparative activée ; masquer les opérations et afficher uniquement les métriques :

```
$>shuf -i 0-9999 -n 500 > args.txt ; ./push_swap --bench $(cat
args.txt) 2> bench.txt | ./checker_linux $(cat args.txt)
OK
$> cat bench.txt
[bench] disorder:  49.93%
[bench] strategy:  Adaptive / O(n√n)
[bench] total_ops:  7997
[bench] sa:  0  sb:  0  ss:  0  pa:  500  pb:  500
[bench] ra:  4840  rb:  1098  rr:  0  rra:  0  rrb:  1059  rrr:  0
```

Transmettre les opérations au vérificateur tout en enregistrant le benchmark dans un fichier :

```
$> ARG="4 67 3 87 23"; ./push_swap --bench --adaptive $ARG 2>
bench.txt | ./checker_linux $ARG
OK
$> cat bench.txt
[bench] disorder:  40.00%
[bench] strategy:  Adaptive / O(n√n)
[bench] total_ops:  13
[bench] sa:  0  sb:  0  ss:  0  pa:  5  pb:  5
[bench] ra:  2  rb:  1  rr:  0  rra:  0  rrb:  0  rrr:  0
```

Exemples de gestion des erreurs :

```
$> ./push_swap --adaptive 0 one 2 3 Erreur $> ./push_swap --
simple 3
2 3 Erreur
```

VI.6 Analyse comparative des performances

Pour valider ce projet, vous devez atteindre certains objectifs de performance avec un nombre minimal d'opérations :

- Pour générer 100 nombres aléatoires, votre programme doit utiliser :
  - Moins de 2000 opérations à réussir (exigence minimale)
  - Moins de 1500 opérations pour de bonnes performances
  - Moins de 700 opérations pour des performances excellentes
- Pour 500 nombres aléatoires, votre programme doit utiliser :
  - Moins de 12 000 opérations à réussir (exigence minimale)
  - Moins de 8 000 opérations pour de bonnes performances
  - Moins de 5500 opérations pour des performances excellentes

Tout ceci sera vérifié lors de votre évaluation à l'aide de l'outil de vérification fourni.

Chapitre VII

Exigences du fichier Lisez-moi

Un fichier README.md doit être fourni à la racine de votre dépôt Git. Son but est de permettre à toute personne ne connaissant pas le projet (collègues, personnel, recruteurs, etc.) de comprendre rapidement de quoi il s'agit, comment l'exécuter et où trouver plus d'informations à ce sujet.

Le fichier README.md doit inclure au moins :

- La toute première ligne doit être en italique et se lire : Ce projet a été créé dans le cadre du programme 42 par <login1>[, <login2>[, <login3>[...]]].
- Une section « Description » qui présente clairement le projet, y compris son objectif et un bref aperçu.
- Une section « Instructions » contenant toutes les informations pertinentes concernant la compilation, installation et/ou exécution.
- Une section « Ressources » répertoriant les références classiques relatives au sujet (documentation, articles, tutoriels, etc.), ainsi qu'une description de la manière dont l'IA a été utilisée — en précisant pour quelles tâches et quelles parties du projet.

Des sections supplémentaires peuvent être nécessaires selon le projet (par exemple, l'utilisation) . exemples, liste des fonctionnalités, choix techniques, etc.).

Les éléments supplémentaires requis seront explicitement indiqués ci-dessous.

- Une explication détaillée et une justification des algorithmes sélectionnés pour ce projet doit également être inclus.

Le choix de la langue est à votre discrétion. Il est recommandé d'écrire en anglais, mais ce n'est pas obligatoire.

Chapitre VIII

Partie bonus

De par sa simplicité, ce projet offre des possibilités limitées pour des fonctionnalités supplémentaires.
Mais pourquoi ne pas créer votre propre jeu de vérification ?

Grâce au programme de vérification, vous pourrez vérifier si le
La liste des opérations générées par le programme push_swap trie correctement la pile.

La partie bonus ne sera évaluée que si la partie obligatoire est parfaite. « Parfaite »
signifie que la partie obligatoire a été entièrement réalisée et fonctionne sans erreur. Dans ce
projet, cela implique :
Tous les critères de référence seront validés sans exception. Si vous n'avez pas
satisfait à TOUTES les exigences obligatoires, votre bonus ne sera pas attribué.
être évalués tout court.

VIII.1 Le programme « checker »

Vérificateur de nom de programme	
Fichiers à soumettre : *.h, *.c	
Makefile	prime
Arguments	pile a : Une liste d'entiers
Fonction externe	• lire, écrire, allouer, libérer, sortie • ft_printf ou toute fonction équivalente VOUS avez codé
Libft autorisé	Oui
Description	Exécuter les opérations de tri

Écrivez un programme nommé checker qui prend comme argument une pile formatée sous forme de liste d'entiers. Le premier élément de la pile doit être placé en haut (attention à l'ordre). Si aucun argument n'est fourni, le programme s'arrête et n'affiche rien.

Le programme attendra ensuite les instructions reçues en entrée standard et les lira, chaque instruction étant suivie d'un saut de ligne (\n). Une fois toutes les instructions lues, il devra les exécuter sur la pile reçue en argument.

- Si, après l'exécution de ces instructions, la pile a est effectivement triée et la pile b est vide, alors le programme doit afficher « OK » suivi d'un « \n » sur la sortie standard.

- Dans tous les autres cas, il doit afficher « KO » suivi d'un « \n » sur la sortie standard.

En cas d'erreur, vous devez afficher « Erreur » suivi d'un saut de ligne (\n) sur le terminal d'erreur standard. Les erreurs peuvent inclure, par exemple : certains arguments ne sont pas des entiers, certains arguments sont supérieurs à un entier, il y a des doublons, une instruction est inexistante et/ou est mal formatée.

```
$>./checker 3 2 1 0
rra
pb
sur
rra
pa
OK
$>./checker 3 2 1 0
sur
rra
pb
KO
$>./checker 3 2 un 0 Erreur $>./checker
""
                                1
Erreur
$>
```

Vous n'êtes pas tenu de reproduire exactement le même comportement que le binaire fourni. La gestion des erreurs est obligatoire, mais il vous appartient de choisir comment analyser les arguments.

Chapitre IX

Soumission et évaluation par les pairs

Déposez votre travail dans votre dépôt Git comme d'habitude. Seul le contenu de votre dépôt sera évalué lors de la soutenance. N'hésitez pas à revérifier les noms de vos fichiers pour vous assurer de leur exactitude.

Exigences de soumission des projets de groupe :

- Les deux apprenants doivent être répertoriés comme contributeurs dans le dépôt.
- Le fichier README.md doit clairement documenter les contributions des deux apprenants.
- Les deux apprenants doivent être présents lors de la soutenance par les pairs.
- Chaque apprenant doit être capable d'expliquer et de défendre n'importe quelle partie du code.

Au cours de l'évaluation, une brève modification du projet peut parfois être demandée. Il peut s'agir d'un changement mineur de comportement, de quelques lignes de code à écrire ou à réécrire, ou d'une fonctionnalité facile à ajouter.

Bien que cette étape ne soit pas applicable à tous les projets, vous devez vous y préparer si elle est mentionnée dans les directives d'évaluation.

Cette étape vise à vérifier votre compréhension réelle d'une partie spécifique du projet.
La modification peut être effectuée dans n'importe quel environnement de développement de votre choix (par exemple, votre configuration habituelle), et elle devrait être réalisable en quelques minutes, sauf si un délai spécifique est défini dans le cadre de l'évaluation.
Il peut vous être demandé, par exemple, d'effectuer une petite mise à jour d'une fonction ou d'un script, de modifier un affichage ou d'ajuster une structure de données pour stocker de nouvelles informations, etc.

Les détails (portée, objectif, etc.) seront précisés dans les directives d'évaluation et peuvent varier d'une évaluation à l'autre pour un même projet.