IG2I LE1 **TP IAP1 n°6** **2025-2026**

Enseignants: Anis Gargouri – Philippe Kubiak - Armand Toguyéni

Consignes : Vous devez insérer une cartouche au début de chacun de vos fichiers sources et y indiquer votre nom et prénom.

Partie 1

Exercice 1 : Gestion des préférences des enfants à la confiserie

La confiseuse a une chaîne de bonbons particulièrement fantaisiste dans sa boutique. Une chaîne de bonbons est une séquence de bonbons individuels, chaque bonbon ayant l'une des 26 saveurs différentes identifiées par les lettres minuscules de '**a**' à '**z**'.

Après l'école, les enfants viennent la voir et achètent des portions de la chaîne de bonbons. Chaque enfant a des préférences différentes, exprimées par des séquences de saveurs, et est prêt à payer un certain montant pour ses portions préférées.

La confiseuse peut extraire des portions de la chaîne de bonbons et les vendre aux enfants. Lorsqu'elle extrait une portion, elle rejoint les parties gauche et droite restantes de la Chaîne de Bonbons, et peut alors continuer à servir des portions supplémentaires, ou décider de s'arrêter. La même séquence de saveurs peut être vendue plusieurs fois au même enfant, tant que la confiseuse est capable de l'extraire plusieurs fois de sa chaîne de bonbons. Elle n'est pas obligée de servir tous les enfants, et elle n'est pas obligée de servir les enfants dans un ordre particulier.

Questions :

Implémenter sous la forme d'un module C, les fonctions correspondantes aux questions 1 à 5.

1/ Implémenter une fonction copierChaine qui prend une chaîne de caractères en entrée, crée une nouvelle chaîne et la remplit avec les mêmes caractères que la chaîne d'origine. Cette fonction pourrait être utilisée pour éviter de modifier la chaîne originale lors de l'extraction de portions.

2/ Implémenter une fonction validerPreferences qui prend une préférence d'enfant en entrée et vérifie si elle est composée uniquement de lettres minuscules de '**a**' à 'z'. Utilisez cette fonction pour valider les préférences avant d'effectuer des opérations sur la chaîne de bonbons.

3/ Implémenter une fonction rechercherSousChaine qui prend une chaîne de caractères et une sous-chaîne en entrée, et renvoie l'index de la première occurrence de la sous-chaîne dans la chaîne donnée.

4/ Implémenter une fonction retirerSousChaine qui prend une chaîne de caractères et une sous-chaîne en entrée, et retire la première occurrence de la sous-chaîne. Cette fonction pourrait être utilisée pour modifier la chaîne de bonbons après l'extraction d'une portion.

5/ Implémenter une fonction afficherPortionsVendues qui prend en entrée la chaîne de bonbons initiale, les préférences des enfants, les montants associés et affiche les portions vendues à chaque enfant, ainsi que le montant total gagné.

6/ Testez votre programme en écrivant la fonction main qui illustre un exemple de chaîne de bonbons et les préférences des enfants. Assurez-vous que le résultat obtenu est correct. Ce programme sera réalisé dans un fichier séparé du module des fonctions précédentes.

7/ Créer un makefile afin de permettre la compilation modulaire de l'application correspondant à cet exercice.

Contraintes :

- Vous pouvez supposer que la chaîne de bonbons initiale est suffisamment longue pour répondre à toutes les préférences des enfants.
- La confiseuse peut servir une même séquence de saveurs à un enfant plusieurs fois tant qu'elle peut l'extraire de la chaîne de bonbons.

Exemple :

Chaine de bonbons :

a	b	a	b	f	z	a	x	o	a	a	a	b	a	b	f	f

Préférences :

ababf	axoa	xyz
3	5	2
Enfant 1	Enfant 2	Enfant 3

Résultat :

Enfant 1 : Portion vendue "ababf" pour 3 euros.

Enfant 1 : Portion vendue "ababf" pour 3 euros.

Enfant 2 : Portion vendue "axoa" pour 5 euros.

Enfant 3 : Aucune portion vendue pour la préférence "xyz".

Montant total gagné : 11 euros.

Exercice 2 : Exercice à déboguer

Veuillez récupérer l'archive EXO1DEBUG.zip. Vous y trouverez 3 fichiers.

1. Déboguer le fichier source exo1Adebug.c
2. Créer un makefile afin de faire de la compilation modulaire avec le fichier hash.o
3. Exécuter l'exécutable obtenu et répondez aux questions qui vous sont posées. Réalisez une copie d'écran pour donner vos réponses dans un compte-rendu !!.

Partie 2 : Analyse et Manipulation de Séquences ADN

Vous allez développer des fonctions en langage C permettant d'analyser et de manipuler des séquences ADN. Une séquence ADN est une **chaîne de caractères composée uniquement des nucléotides** représentés par les lettres **A**, **T, C** et **G**.

- Les séquences d'ADN ne dépasseront pas MAXCAR caractères :
#define MAXCAR 100
- Le type Tchaine est défini par :
typedef char Tchaine[MAXCAR+1];

Contraintes Techniques :
- Le non-respect des prototypes donnés dans les exercices sera considéré comme erreur.
- Vous êtes libres d'utiliser des fonctions/macro-fonctions intermédiaires pour faciliter votre code, il faudrait dans ce cas les implémenter dans votre copie.
- L'usage des fonctions de string.h est autorisé, un appel incorrect à ces fonctions sera considéré comme une erreur.
- Nous supposerons que les entrées des fonctions exercices 2 et 3 sont obligatoirement des séquences ADN valides.
- Vous avez le droit d'utiliser une fonction demandait dans l'un des exercices même si elle n'est pas encore écrite.

Exercice 3 : Vérification si une composition est valide

L'objectif est de vérifier si une séquence ADN est composée uniquement des lettres A, T, C et G.

Prototype : ***int verifierComposition(const char seq[]);***

Exemple :

- Entrée : ATCGTTAGC → Résultat : Valide (1)

- Entrée : ATBXCG → Résultat : Invalide (0)

Exercice 4 : Fonctions sur les Nucléotides

Question 1 : Comptage des Nucléotides

Objectif : Comptez combien de fois chaque nucléotide (A, T, C, G) apparaît dans une séquence ADN.

Prototype : ***void compterNucleotides(const char seq[], int* countA, int* countT, int* countC, int* countG);***

Exemple :

- Entrée : ATGCTACG → Résultat : A = 2, T = 2, C = 2, G = 2

Question 2 : Identification des Nucléotides Majoritaires

Objectif : Identifiez le nucléotide le plus fréquent dans une séquence ADN.

Prototype : ***char nucleotideMajoritaire(const char seq[]);***

Exemple :

- Entrée : ATCGTTAGC → Résultat : Nucléotide majoritaire : T

Question 3 : Vérification d'Équilibre de Nucléotides

Objectif : Vérifiez si une séquence ADN est "équilibrée" (chaque type de nucléotide apparaît le même nombre de fois).

Prototype : ***int verifierEquilibre(const char seq[);***

Exemple :

- Entrée : ATGCATCG → Résultat : Équilibrée (1)
- Entrée : ATGATGA → Résultat : Non équilibrée (0)

Question 4 : Calcul du Complément d'une Séquence ADN

Objectif : Générez la séquence complémentaire, où chaque nucléotide est remplacé par son partenaire : A ↔ T, C ↔ G.

Prototype : ***void calculerComplement(char comp[], const char seq[]);***

Exemple :

- Entrée : ATGCTACG → Résultat : TACGATGC

Question 5 : Déterminer si une Séquence est un Palindrome ADN

Objectif : Vérifiez si une séquence ADN est un palindrome, c'est-à-dire que sa complémentation inversée est identique à la séquence initiale.

Prototype : ***int estPalindromeADN(const char seq[]);***

Exemple :

- Entrée : ATGCAT → Résultat : Oui (1)
- Entrée : ATGGTA → Résultat : Non (0)

Exercice 5 : Fonctions sur les Séquences

Question 6 : Trouver la Séquence la Plus Longue

Objectif : Identifiez la position de début de la plus longue sous-séquence d'une chaîne ADN ne contenant pas un nucléotide n donné en paramètres.

Prototype : ***int plusLongue(char result[], char n, const char seq[]);***

Exemple :

- Entrée : Séquence = ATGCGCTTAGCGT, Nucléotide= ‘A’ → Résultat : 1 (position de la sous-séquence : TGCGCTT)

Question 7 : Rotation de Séquence ADN

Objectif : Effectuez une rotation circulaire à gauche sur une séquence ADN d’un nombre donné de positions.

Prototype : ***void rotationSequence(char seq[], int pos);***

Exemple :

- Entrée : Séquence = ATGCATGC, Rotation = 3 → Séquence = CATGCATG

Question 8 : Fusion de Séquences avec Superposition Maximale

Objectif : Fusionnez deux séquences ADN en maximisant leur superposition.

Prototype : ***void fusionSuperposition(char result[], const char seq1[], const char seq2[]);***

Exemple :

- Entrée : Séquence 1 = ATGCGCG, Séquence 2 = GCGTA → Résultat : ATGCGCGTA