EIDGENÖSSISCHE TECHNISCHE HOCHSCHULE – LAUSANNE
POLITECNICO FEDERALE – LOSANNA
SWISS FEDERAL INSTITUTE OF TECHNOLOGY – LAUSANNE

Faculté Informatique et Communication
Cours Programmation I (SV)
Jamila Sam

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Série notée - Sujet 2

Instructions

— Max 60mn
— Max 45 points

1. Dans un navigateur, ouvrez la page
   se trouvant sous la rubrique **Séries d'exercices** de la page web du cours

   suivez scrupuleusement les instructions qui apparaissent.

2. Vous pouvez utiliser le navigateur une fois ces manipulations faites, pour consulter la page web du cours.

3. Pour **RENDRE** la série notée, en fin d'examen :
   (a) Allez à nouveau sur le lien donné précédemment et allez au point 3 des instructions. Le menu qui apparaît comporte déjà le nom du fichier à rendre.
   (b) Pour envoyer un fichier, cliquez sur le bouton `Browse` correspondant au nom du fichier et sélectionnez le fichier en question dans la liste qui vous est proposée. Cliquez ensuite sur le bouton `Envoyer`.
   Le fichier suivant est à envoyer, **même si non abordé** :

   protein.cc

   Pour terminer, n'oubliez pas de **VALIDER VOTRE ENVOI**.

Règles et recommandations :

1. La série est à réaliser **individuellement**. L'échange d'informations est strictement interdit (y compris l'échange de documents).

2. La série doit impérativement être réalisée sur les terminaux des salles CO020-023 et sur votre propre compte. Il ne doit y avoir **qu'un seul login sur votre compte durant le temps de la série notée**.

3. Vous pouvez amener votre propre clavier, mais il vous appartient de tester avant la série que ce matériel fonctionne correctement.

4. Il n'est pas permis : d'utiliser du matériel électronique, d'envoyer des emails, de vous connecter sur une autre machine ou d'imprimer pendant la durée de la série notée.

5. Documents autorisés: les documents du cours (transparents, séries, corrigés, matériel du MOOC). Il vous est possible de consulter cette documentation, et seulement celle-là, en ligne. 1 livre de référence sur le langage C++, au plus 100 feuilles de notes personnelles. En cas de doute, demandez l'avis de l'assistant responsable. **Les corrigés des anciens tests ne sont pas autorisés**.

6. Il est **impératif de mettre vos numéros SCIPER** en commentaire au début de chacun de vos fichiers.

7. Lisez attentivement chaque question de façon à ne faire que ce qui vous est demandé, et servez-vous de l'exemple de fonctionnement accompagnant les exercices pour vérifier votre solution. **Si l'énoncé ne vous paraît pas clair, ou si vous avez un doute, demandez des précisions à l'un des assistants.**

8. Si pour des raisons de langue, vous ne comprenez pas bien ce qui vous est demandé, n'hésitez pas à faire appel à un assistant pour obtenir des clarifications.

9. Sauf mention explicite, les affichages ne doivent pas obligatoirement correspondre à ceux donnés en exemple de fonctionnement; les mêmes informations doivent être affichées, mais le format peut être différent.

10. La série comporte un seul exercice.

11. Le fait que le code soit compilable ou exécutable compte dans la notation mais ne constitue pas le seul critère. La correction tiendra compte également des "solutions approchées".

Acides aminés (45 points)

Il s'agit dans cet exercice d'écrire un certain nombre de fonctions permettant de manipuler des protéines modélisées comme des séquences d'acides aminés.

Vous prendrez note des directives suivantes avant de commencer à coder :

1. Vous travaillerez dans le fichier `protein.cc` et **n'utiliserez pas d'autres fichiers**. Quelques éléments y sont déjà fournis (examinez les commentaires associés pour en comprendre l'utilité) :

   — un type utile défini au moyen de `typedef` ;

   — une constante globale utile `ACIDAM` ;

   — une fonction utilitaire `presente` (voir les commenraires d'explications fournis avec le code);

   — une fonction utilitaire `remplace` (voir les commenraires d'explications fournis avec le code);

   — deux fonctions `compte_sol` et `trouve_compacte_sol` à utiliser éventuellement en guise de dépannage (selon les directives de l'énoncé);

   — un programme principal auquel votre code devra strictement se conformer.
   Ce programme ainsi que les fonctions utilitaires fournies ne doivent pas être modifiés d'une autre façon que ce que l'énoncé vous suggère.

2. Vous compilerez sous la norme C++11. La commande dans le terminal est :

   ```
   g++ -std=c++11 protein.cc -o protein
   ```

3. Si vous le jugez utile, vous êtes libres d'ajouter d'autres méthodes à celles qui vous seront explicitement demandées.

4. **Lisez bien chaque descriptif de fonction jusqu'au bout avant de la coder**

5. Les fonctions que vous devrez coder peuvent dépendre les unes des autres. **Si vous n'avez pas réussi à coder une fonction codez quand même celles qui en dépendent**. Nous corrigerons chaque fonction comme si toutes les autres étaient correctes.

SUITE DE L'ÉNONCÉ au VERSO ⟶

1) Coefficient d'extinction

Les acides aminés constituant une protéine sont codifiés au moyen de lettres : `W` (Tryptophane), `Y`(Tyrosine) et `C`(Cystine). Nous nous contenterons de ces 3 acides pour cet exercice; une protéine sera donc modélisée par une séquence telle que `WWCCYW`.

Le type `Chaine` est fourni pour modéliser des séquences de caractères. Les protéines seront modélisées au moyen de ce type.

Le coefficient d'extinction permet de déterminer la nature d'une protéine. Il se calcule comme suit :

```
nW * 5500 + nY * 1490 + nC * 125
```

où `nW` représente le nombre de `W`, `nY` le nombre de `Y` et `nC` le nombre de `C`.

Il vous est demandé de programmer :

1. Une fonction `size_t compte(const Chaine& seq, char code);` permettant de compter le nombre d'occurences du caractère `code` dans la séquence `seq`.
2. Une fonction `int coeff_extinct (const Chaine& seq)` calculant le coefficient d'extinction de la séquence `seq`, selon la formule ci-dessus. Il n'est pas important que celle-ci contienne d'autres caractères que `W`, `Y` ou `C`. Vous utiliserez la fonction `compte`.
3. Une fonction `void affiche_extinct(const Chaine& seq)` affichant le coefficient d'extinction d'une séquence selon le modèle de l'exemple suivant :

   ```
   Coeff. extinction de W Y W W C : 18115
   ```

Pour tester le programme implémenté jusqu'ici, décommentez la partie du programme principal comprise entre `// TEST 1` et `// FIN TEST 1`.

Si vous n'avez pas réussi à programmer correctement `compte`, vous pouvez utiliser à la place la fonction fournie `compte_sol` dans toutes les fonctions devant appeler `compte`. Il faudra pour cela la décommenter au préalable.

La trace d'exécution pour cette partie devrait ressembler à ce qui suit :

```
Test 1
Analyse de la séquence : Y C W W C
Nombre de W : 2
Nombre de Y : 1
Nombre de C : 2
Coeff. extinction de Y C W W C : 12740
```

2) Analyse de séquences

Écrivez la fonction `Sequence max_extinct (const vector<Chaine>& sequences)` prenant en paramètre un ensemble de séquences et retournant celle avec le plus grand coefficient d'extinction. Si l'ensemble de séquences est vide, la séquence `"-"` sera retournée.

Pour tester le programme implémenté jusqu'ici, décommentez la partie du programme principal comprise entre `// TEST 2` et `// FIN TEST 2` ,

La trace d'exécution pour cette partie devrait ressembler à ce qui suit :

```
Test 2
Coeff. extinction de Y C W W C : 12740
Coeff. extinction de W W Y Y C : 14105
Coeff. extinction de Y Y Y C W : 10095
Chaine avec le plus grand coefficient : W W Y Y C
```

Si vous n'êtes pas parvenu à coder ces fonctions laisser simplement le code correspondant (code de test compris) en commentaire et passez à la suite.

3) Nucléotides

Les acides aminés peuvent aussi être codifiés de façon moins compacte au moyen de triplets de nucléotides :

— `W` peut être codifié au moyen du triplet `UUG`
— `C` peut être codifié au moyen du triplet `UGU`
— `Y` peut être codifié au moyen du triplet `UAU`

La constante fournie `ACIDAM` permet d'établir la correspondance entre le codage compact et le codage étendu avec des triplets de nucléotides.

Écrivez les fonctions :

1. `Chaine trouve_compacte(const Chaine& triplet)` cherchant dans `ACIDAM` la codification compacte associée à un triplet de nucléotides donné, et retournant cette dernière. Par exemple `trouve_compacte("UUG")` doit retourner `"W"`. Si `triplet` n'est pas de taille 3 ou si c'est un triplet qui n'existe pas dans la table `ACIDAM`, la chaîne `"-"` sera retournée.

2. `void affiche_compacte(const Chaine& seq)` affichant la représentation compacte associée à un triplet donné selon le format de l'exemple suivant :

   ```
   Codage compact de (UGU) : C
   ```

Pour tester cette partie, décommentez la partie du programme principal comprise entre `// TEST 3` et `// FIN TEST 3`.

La trace d'exécution pour cette partie devrait ressembler à ce qui suit :

```
Test 3
Codage compact de (AUGC) : -
Codage compact de (UGU) : C
Codage compact de (UAU) : Y
Codage compact de (UGC) : -
Codage compact de (UGG) : W
```

Si vous n'êtes pas parvenu à coder la fonction `trouve_compacte` vous pouvez utiliser à la place la fonction `trouve_compacte` partout où `trouve_compacte` est nécessaire. Il faudra pour cela la décommenter au préalable.

SUITE DE L'ÉNONCÉ au VERSO ⟶

4) Codage étendu à codage compact

En partant d'une codification étendue au moyen de triplets de nucléotides, on souhaite retrouver la codification compacte d'une protéine.
Programmez pour cela une fonction `compacte` prenant en paramètre une séquence `s` et un triplet de nucléotides donné et qui remplace toutes les occurences du triplet par sa forme compacte dans `s`. On ne se préoccupera pas ici de savoir si le triplet est vraiment un triplet (n'importe quelle séquence sera acceptée en guise de triplet).

Par exemple, si `s` vaut au départ `"UAU UGU UGG UGG UGU"`, alors `s` doit valoir `"UAU UGU W W UGU"` après l'appel `compacte(s, "UGG")`

Vous pourrez utiliser les fonctions fournies `presente` et `remplace` et vous devrez utiliser la fonction `trouve_compacte` de l'étape précédente.

Le prototype de `compacte` est délibérément omis. **Consultez le `main` fourni pour voir comment cette fonction est censée être utilisée entre `// TEST 4` et `// FIN TEST 4`.**

Pour tester cette partie, décommentez la partie du programme principal comprise entre `// TEST 4` et `// FIN TEST 4`.
La trace d'exécution pour cette partie devrait ressembler à ce qui suit :

```
Test 4
Séquence initiale : UAU UGU UGG UGG UGU
Remplacement de UGG : UAU UGU W W UGU
Remplacement de UAU :  Y UGU W W UGU
Remplacement de UGU : Y C W W C
```

5) Codage étendu à codage compact (Suite)

Programmez enfin une fonction `compacte` prenant en paramètre une séquence et y remplaçant tous les triplets présents dans la table `ACIDAM` par leur forme compacte. Si au terme de ces remplacements, il reste dans la séquence des caractères différents de `W`, `Y`, `C` ou espace , ces caractères seront supprimés. **L'utilisation de cette fonction `compacte` doit être conforme à l'utilisation qui en est faite dans le `main` entre `// TEST 5` et `// FIN TEST 5`**

Le prototype de `compacte` est délibérément omis.

Vous utiliserez la fonction `compacte` codée à l'étape précédente. Vous n'utiliserez pas de fonctions spécifiques aux `string`, mais de simples boucles pour réaliser les traitements.

Pour tester cette partie, décommentez la partie du programme principal comprise entre `// TEST 5` et `// FIN TEST 5`.
La trace d'exécution pour cette partie devrait ressembler à ce qui suit :

```
Test 5
Séquence initiale : UAU UGU UGG UGG UGU
Remplacement de tous les acides aminés :
Y C W W C
 La séquence d'origine avait le bon format

Séquence initiale : UAU UGU UGG UGG UGU UGZT
Remplacement de tous les acides aminés :
Y C W W C
 Attention la séquence d'origine n'avait pas le bon format
```