AP1 TP

11 Semaine 11 : Chaînes de caractères 1/1

Dans ce TP, nous allons manipuler les chaînes de caractères du standard C++, i.e., la classe `string`. Il faudra donc inclure la bibliothèque `string` au début de vos programmes.

Exercice 1 : Notion d'arguments en ligne de commande

Dans le répertoire `/net/Bibliotheque/AP1/TPSem11`, vous trouverez deux programmes, `rechercher1.cc` et `rechercher2.cc`, qui réalisent le même traitement : rechercher, au sein d'un mot, la première occurrence d'un autre mot, et afficher son indice d'apparition.

1. Consultez, compilez et exécutez ces 2 programmes.
2. Vous remarquerez que le 2ème attend des arguments en ligne de commande. C'est grâce aux paramètres déclarés pour la fonction main :
   `int main( int argc, char* argv[] ) {...}`
   Le paramêtre `argc` est le nombre d'arguments de la commande qui lance le programme (y compris le nom du programme lui-même), et le tableau `argv` contient tous les arguments donnés lors du lancement du programme (`argv[0]` est le nom du programme lui-même).
   Reportez vous au fichier `rechercher2.cc` et à son exécution pour plus de précisions à travers un exemple.
3. Quel est l'avantage de `rechercher2.cc` par rapport à `rechercher1.cc` ?

Exercice 2 : Recherche et substitution dans une chaîne de caractères

Le but est de rechercher un mot dans une chaîne de caractères, et de remplacer toutes les occurrences trouvées par un nouveau mot (exemple : "a" est remplacé par "on" dans "tata" qui devient "tonton").

1. Écrivez et testez une fonction, qui prend
   - ➤ en entrée/sortie une chaîne de caractères (un `mot` : sur lequel on effectue la recherche / remplacement)
   - ➤ en entrée un mot que l'on cherche à remplacer (`s1` une chaîne de caractères)
   - ➤ en entrée un autre mot qui se substituera au mot précédent (`s2`).

   Cette fonction a pour signature :
   `void substituer( string & mot, string s1, string s2 )`
   La fonction `substituer` sera appelée du programme principal (`main`) qui, comme pour l'exercice précédent, utilisera la notion d'arguments, et prendra donc en ligne de commande le mot "tata" et les 2 mots "a" et "on".
   Que se passe-il si l'on remplace la signature de `substituer` par :
   `void substituer( string mot, string s1, string s2 );`
   Que se passe t-il si on essaie de remplacer "a" par "aa" dans "tata" ? Pourquoi ?
2. Optimisez la fonction précédente en utilisant la syntaxe suivante pour `find` afin de relancer la recherche à partir de la dernière position trouvée :

```
string str = "abracadabra";
int n1 = str.find( "bra" );     // Syntaxe classique, retourne 1 et n1 vaut 1.
int n2 = str.find( "bra", 5 ); // Syntaxe etendue, recherche la
                                // sous-chaine "bra" à partir de la position 5
                                // dans str. Retourne donc 8 et n2 vaut 8.
```

Exercice 3 : Tableau de chaînes de caractères

On se propose maintenant de manipuler un tableau de chaînes de caractères.

1. **Définition d'un nouveau type.** On a tout à fait le droit de définir un tableau de chaînes de caractères de la même manière qu'on définit un tableau d'entiers :

```
// Declaration d'une constante pour la taille
const int MAX=200;
// Definition d'un nouveau type
typedef string TTabChaine[MAX];

//Declaration de variable
TTabChaine tbl_str;
//...
// affichage la 3eme chaine de ce tableau.
cout << tbl_str[2] << endl;
```

2. **Remplissage du tableau.** Pour éviter à l'utilisateur de taper les chaînes de caractères, nous allons utiliser un fichier texte qui contient les données à mettre dans le tableau. Pour cela, le lancement de l'exécutable est paramétré par un fichier texte (`/net/exemples/AP1/TPSem11/yellow.txt`). Si `tabChaine` est le nom de votre programme, vous taperez la commande :

   ```
   cat yellow.txt | tabChaine
   ```

   ou bien

   ```
   tabChaine < yellow.txt
   ```

   Écrivez une fonction de saisie des valeurs d'un tableau de chaînes de caractères :
   `void saisirTab( TTabChaine & tab, int & taille )`
   Cette fonction initialise la structure par des lectures successives (`cin>>`) tout en vérifiant que la taille du tableau n'est pas dépassée.
3. **Affichage du tableau.** Écrivez une fonction `void afficherTab( TTabChaine tab, int taille )`. Utilisez la pour vérifier que le remplissage du tableau est correct.
4. **Calcul du nombre d'occurrences pour chaque mot.** On veut faire des statistiques sur le texte lu. Pour cela, on va construire un tableau contenant des doublets associant à chaque mot du texte son nombre d'occurrences.
   (a) Définissez un nouveau type pour le tableau d'occurrences.
   (b) Écrivez une fonction permettant de remplir le tableau d'occurrences à partir du tableau contenant le texte. L'algorithme en langage naturel est le suivant :
   on parcourt le tableau contenant le texte ; à chaque mot rencontré on met à jour le tableau d'occurrences. Deux cas sont à distinguer :
      i. le mot est déjà présent dans le tableau : son nombre d'occurrences est augmenté de 1.
      ii. le mot n'est pas présent : il est inséré dans le tableau d'occurrences en respectant l'ordre lexicographique.

   (c) Écrivez une fonction qui permet d'afficher le tableau d'occurrences une fois qu'il est rempli.
   (d) Écrivez une fonction qui à partir d'un mot donne son nombre d'occurrences dans le texte.