IFT1166 – TRAVAIL PRATIQUE #1 – 23 Janvier 2K

Manipulation des chaînes de caractères

Mohamed Lokbani

LA DATE LIMITE EST LE 16 février 2K Minuit, avec une pénalité de 2 points par jours de retard.

À faire en équipe de deux.

1. BUT

Ce TP a pour but de faire la transition du C vers le C++. Vous allez pratiquer les matières suivantes :

Utilisation des outils de programmation Linux.
Programmation procédurale en C++.
Utilisation de la bibliothèque de manipulation des chaînes de caractères.
Compilation séparée.

N'utilisez pas de classes (vous aurez besoin de définir une classe canonique).

2. Description

Il existe aujourd'hui une quantité importante de données (informations) qu'il faudra exploiter de manière efficace afin d'en tirer profit. Par exemple, dans le domaine de la veille technologique, la quantité d'informations recueillies, nous permettra le repérage rapide des nouveautés techniques. Cependant, l'exploitation de cette quantité d'informations, est rendue difficile par la diversité de ses origines. L'opération d'extraction de texte d'un fichier au format word n'est pas la même que celle utilisée pour un fichier au format HTML. Des traitements parfois simples, parfois complexes, sont nécessaires pour sélectionner la partie du texte, jugée utile, et la mettre sous une forme permettant son exploitation (opérations de formatage, balisage et encodage).

Ce tp a pour but de vous faire réaliser un filtre, "à la UNIX", qui prend en entrée un texte balisé, et produit en sortie un texte mis en forme en tenant compte de ses balisages, mais débarrassé de ces derniers.

3. Balises

Pour ce TP il a été défini 6 balises `(u, l, n, g, i, t)` représentées chacune par un seul caractère, qui peut-être en majuscule ou en minuscule (ex: `u` ou `U`). La balise `<u>` marque le début du traitement, alors que la balise `</u>` marquera la fin du traitement. À chaque balise est associé un traitement donné:

`<u>` met le texte qui suit en majuscule. Cet effet durera jusqu'à la balise fermante `</u>`

`<l>` met le texte qui suit en minuscule. Cet effet durera jusqu'à la balise fermante `</l>`

`<n>` garde le texte qui suit comme il est. Cet effet durera jusqu'à la balise fermante `</n>`

`<g>` met les caractères minuscules du texte qui suit en majuscule, et les caractères en majuscule en minuscule. Cet effet durera jusqu'à la balise fermante `</g>`

`<i>` met le texte qui suit à l'envers. Cet effet durera au plus jusqu'à la balise fermante `</i>`

`<t>` trie la liste des mots du texte qui suit dans ordre croissant alphabétique. Cet effet durera jusqu'à la balise fermante `</t>`. La liste contient une série de mots, tous en minuscules, séparés par des blancs. Il peut y avoir un espace blanc avant le premier mot et après le dernier mot. Vous pouvez utiliser l'algorithme de tri de votre choix.

Exemple

Entrée	Sortie
`<u> en majuscule </U>`	`EN MAJUSCULE`
`<L> EN MINUSCULE </l>`	`en minuscule`
`<n> reste la même </n>`	`reste la même`
`<G> GrAnD pEtIt </G>`	`gRaNd PeTiT`
`<i> etour </i>`	`route`
`<t> travaux livre remise aventure </t>`	`aventure livre remise travaux`

4. Travail à réaliser

Vous avez à écrire un programme en C++ permettant de réaliser l'opération de filtrage des balises et la mise à jour du texte balisé. Pour ce faire certains outils vous sont fournis et certaines contraintes vous sont imposées.

4.1 Outils fournis

La lecture du texte balisé et l'écriture du résultat après filtrage, se feront par l'intermédiaire de fonctions définies dans une bibliothèque. De ce fait la saisie n'aura pas lieu du clavier et l'affichage ne se fera pas à l'écran. Cette approche permettra une correction automatisée de vos programmes. Cependant, durant la phase de développement de votre programme, vous pouvez vous servir du clavier pour saisir vos données de test, et l'écran pour afficher des étapes du déroulement de votre programme. Par la suite, avant la remise définitive, faites une remise à jour de votre programme afin d'inclure les fonctionnalités E/S décrites dans ce qui suit.

4.1.1. La lecture des données

Il vous est fourni une fonction permettant la lecture de manière transparente du texte à filtrer:

```
void load_data (char expression[]);
```

`load_data` est une fonction qui permet d'initialiser un tableau de caractères. Après appel de cette fonction le tableau va contenir un texte balisé, qui sera filtré par votre programme.

où

`expression` est un tableau de caractères.

Vous devez déclarer préalablement un tableau, par exemple, `tab`, de taille maximale `10000`. Vous faites par la suite l'appel de la fonction `load_data` avec comme argument `tab`. `load_data` remplit le tableau `tab` avec l'expression de test. Une expression n'excédant pas la taille maximale de `10000` caractères.

4.1.2. L'écriture du résultat

Afin d'accélérer la correction de votre programme, il vous est fourni une fonction permettant d'écrire de manière transparente, le résultat après filtrage dans un fichier texte. Ce fichier texte sera comparé de manière automatique à un fichier de référence. La fonction d'écriture est:

```
void write_data (char sortie[]);
```

Le tableau `sortie` contiendra le résultat de votre traitement. Le résultat sera écrit dans un fichier qui portera le nom de `resultat.tmp` vous ne devez pas vous préoccuper de l'existence de ce fichier. Sachez uniquement, qu'à chaque appel de la fonction `write_data`, s'il existe déjà dans votre répertoire un fichier qui porte le nom `resultat.tmp` ce dernier sera écrasé et remplacé par le nouveau. S'il en existe aucun, un nouveau fichier sera crée.

Voir aussi format de sortie (point suivant) et compilation des programmes (paragraphe 6.).

4.2 Contraintes sur le texte à filtrer

Il vous est fourni en entrée un texte balisé. Vous devez vérifier ce qui suit:

- le balisage de votre texte, est bien balancé: autant de balises fermantes que d'ouvrantes, sinon le programme s'arrête en affichant un message d'erreur. Exemple: `<u> test`. Il manque la balise fermante `</u>`.

- À chaque balise ouvrante doit suivre après le texte à mettre en forme, la balise fermante qui lui correspond. `<u> test </i>` génère une erreur. `<u> test </u>` ou `<i> test </i>` étaient quant à eux corrects.

- Considérez que vous avez à traiter uniquement les 6 balises préalablement définies au paragraphe 3.

- Entre deux paires de balises, il peut y avoir des commentaires. Ce texte ne sera affecté par aucun traitement, vu qu'il n'est pas balisé. Par ailleurs, lors de l'écriture du résultat, ce texte disparaîtra.

- Vous pouvez faire l'hypothèse, que dans le cas ou le texte est bien balancé, il ne peut pas contenir des imbrications de balises, i.e. votre texte ne contiendra pas une expression du style:

  `<u><i> test </i></u>`

  Le fait d'avoir interdit les imbrications, va vous permettre de réaliser ce tp sans utiliser une pile, une fonction récurrente est suffisante pour faire cette tâche

Un exemple complet :

```
<u> test </u> <i> test </i> <l> test </l> texte qui ne sert à rien <g>
test </g> <t> midi demain hier soir </t> un autre texte qui ne sert à
rien.
```

Donnera comme résultat après filtrage:

```
TEST tset test TEST demain hier midi soir
```

La règle générale: une succession de balises non imbriquées définies au paragraphe 3. Entre une paire de balises, il peut y avoir un commentaire qui sera ignoré lors de l'écriture du résultat final.

Le format de ce résultat est comme suit:

un espace blanc, puis le résultat obtenu en traitant chaque balise, en séparant chacun de ces résultats par un espace blanc, puis un espace blanc (final). Puisque ce résultat est écrit dans un tableau de caractères, il faudra inclure, après le dernier caractère écrit le caractère **`\0`** (anti-slash et le chiffre zéro) pour marquer la fin de la chaîne (voir l'exemple précédant).

5. Fichier à votre disposition

- data.h contient les entêtes des fonction load_data & write_data il doit être inclus dans vos programme comme un fichier d'en-tête (de la même façon que iostream.h etc.).

- data.o est le ficher objet compilé sous Linux (**ne fonctionnera que sur Linux**).

ces fichiers sont disponibles sur la page web du tp1@:

http://www.iro.umontreal.ca/~dift1166

sinon dans le répertoire:

/u/dift1166/Sujet/H2K/tp1

accessible via votre compte.

6. compilation des programmes

6.1. le contenu des fichiers

Vous devez remettre **deux** fichiers :

- `filtre.h` le fichier d'en-tête du programme.

- `filtre.cpp` contenant les fonctions et le programme principal.

Avant de compiler votre programme inclure les lignes suivantes :

- Dans votre fichier `filtre.h`

  Dans la section des "include" :

  ```
  #include "data.h"
  ```

- Dans votre fichier `filtre.cpp`

  Un commentaire au début du fichier indiquant : vos noms, prénoms & login.

- Dans la fonction `main`, faites un appel de la fonction:

  ```
  load_data (texte);
  ```

  afin de remplir le tableau avec les données préparées pour vos tests.

  puis après votre traitement

  ```
  write_data (resultat);
  ```

pour écrire le résultat.

Dans cet exemple, les noms `texte` et `resultat`, ont été choisis au hasard.

6.2. commande de compilation

Pour compiler, utilisez la commande:

g++ -Wall –o filtre filtre.cpp data.o

les fichiers `filtre.h` et `data.h` doivent se trouver dans le même répertoire que les fichiers `filtre.cpp` et `data.o`

7. Remise

Vous devez faire deux types de remise :

- Papier (à glisser sous la porte de mon bureau, Pavillon André-Aisenstadt, local 2249)
  bien identifier sur le listing le cours (IFT1166) et vos noms.

- Électronique : dans le répertoire où est votre programme, tapez:

remise ift1166 tp1 filtre.h filtre.cpp

LA DATE LIMITE EST LE 16 février 2K Minuit, avec une pénalité de 2 points par jours de retard.

8. Barème

Ce premier tp1 est noté sur 10 points. Nous vous suggérons la démarche suivante:

- écrire un simple programme qui utilise les fonctions `load_data` et `write_data`.

appel de `load_data` puis de `write_data`, ainsi vous aurez une idée du texte à traiter.

- écrire le code pour lire et vérifier que la chaîne retournée par la fonction `load_data` est bien balancée.

- au départ faire le traitement d'une seule balise (exemple le traitement de la balise `<u> </u>`).
si vous procédez ainsi, utilisez vos données de test à la place de ceux retournés par la fonction `load_data`, si vous préférez garder `load_data` à cette étape, prévoyez une sorte de traitement à vide des autres balises.

- généraliser pour les autres balises (garder la balise de tri pour la fin, elle est plus "compliquée").

Les points seront répartis comme suit :

Exactitude : 5 points

un programme qui ne compile pas 0/5

un programme avec des avertissements ("warnings") et qui donne les résultats escomptés 3.5

un programme avec des avertissements "warnings" mais qui fait des choses non prévues dans la spécification 0/5

Modularité : 2 points

structure de votre programme, décomposition entre .h et .cpp, décomposition en fonctions etc.

Efficience : 2 points

La manière avec laquelle vous allez coder votre programme. L'espace mémoire utilisé etc.

Style : 1 point

le code doit être lisible, bien indenté et commenté.

9. Fonctions utiles

Pour rappel, des fonctions ont été définies, dans `<string.h>`, pour manipuler les chaînes de caractères. Parmi ces fonctions, nous pouvons citer:

`char* strcpy (char* s1, char* s2)` copie la chaîne s2 dans le tableau de caractères s1.

`char* strcat (char* s1, char* s2)` ajoute la chaîne s2 à la chaîne s1. Si vous voulez inclure une chaîne vide entre s1 et s2. Mettre dans la chîne s2 un espace blanc.

`char* strtok(char* s1, const char* s2)` division de la chaîne s1 en jetons en fonction du contenu de la chaîne s2.

`int strcmp(const char* s1, const char* s2)` compare la chaîne s1 à la chaîne s2. Retourne 0 si les deux chaînes sont identiques.

`size_t strlen (const char* s)` détermine la longueur de la chaîne s1.

10. des questions à propos de ce TP?

l'adresse email: difft1166@iro.umontreal.ca

pour faciliter le traitement de votre requête, inclure dans le sujet de votre email, au moins la chaîne: [IFT1166]

les questions posées (ainsi que les réponses) durant la durée de ce tp seront archivées sur la page web du tp1. Vous pouvez les consulter on-line pour voir si votre question y figure déjà.

http://www.iro.umontreal.ca/~difft1166

11. mise à jour

25-01-2K correction au niveau du tableau exemple, paragraphe 3.
24-01-2K diffusion

bon courage!