Licence 1 - section B
TP 5 d'éléments d'informatique

Catherine RECANATI – Département d'Informatique – Institut Galilée

Semaine du 5 décembre au 11 décembre 2016

L'objectif de ce TP est encore de vous faire adopter de bonnes habitudes concernant l'organisation des fichiers sources de vos programmes. Vous devrez en général écrire

1. votre programme principal contenant la fonction `main` dans un fichier "`main.c`"
2. de même, la définition de vos fonctions personnelles devra être écrite dans des fichiers séparés, portant le nom de la fonction. Ces fichiers seront compilés séparemment avec l'option `-c` de `gcc`.
3. enfin, vous regrouperez dans un fichier nommé "`decla.h`" toutes les déclarations de vos fonctions personnelles au fur et à mesure que vous les écrirez. Cela vous permettra d'inclure ce fichier partout où vous utiliserez ces fonctions (dans `main.c` bien entendu, mais peut-être aussi dans un autre fichier de définition/déclaration de fonction).

Rappels : pour savoir comment créer de nouveaux répertoires et comment naviguer d'un répertoire à l'autre, reportez vous aux préambules des feuilles de TP précédents. Vous pouvez aussi consulter la commande `man` pour vous renseigner sur les commandes de manipulation de fichiers (cf. `cp`, `mv`, `cd` et `rm`).

Pour connaitre le mode d'emploi de la commande `man`, il suffit de taper `man man` dans la fenêtre d'un terminal. Vous y verrez à quoi correspondent les différentes rubriques du manuel. Ensuite pour connaître le manuel sur la commande `gcc` par exemple, tapez `man gcc`. Vous pouvez utiliser la commande `man` pour toute commande Unix, mais aussi pour certaines fonctions du système qui font partie des bibliothèques standards. Ces fonctions peuvent être appelées depuis un programme C (c'est le cas par exemple des fonctions `printf` et `scanf`).

1 Mots et caractères d'une phrase

On va reprendre le programme du TD10 sur le décompte des caractères et des mots d'une phrase entrée au clavier par l'utilisateur. La procédure de lecture de la phrase s'appellera `lirePhrase` et prendra en argument un tableau de caractères. On l'appellera depuis le programme principal avec la variable phrase de type tableau de la fonction `main`.

On part du fichier main.c suivant :

```
#include <stdio.h>
#include <stdlib.h>

/* inclure le fichier de declarations des fonctions utilisateur */
#include <decla.h>

#define TAILLE 500

int main ()  {
     char phrase[TAILLE] = "";

    /* initialisation de phrase */
     lirePhrase(phrase);

     return EXIT_SUCCESS;
}
```

et du fichier lirePhrase.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
void  lirePhrase(char tab[], unsigned long taille) {
        int nb = 0;

        printf("Entrez une phrase:\n");
        fgets(tab, taille, stdin);

     /* la phrase lue dans tab termine par un caractere '\n'  */
     /* suivi d'un caractere nul. On va supprimer ici */
     /* le caractere  '\n' qui precede le 'caractere nul   */

        nb = strlen(tab);    /* nb caracteres dans tab */
        tab[nb -1] = '\0' ;   /* on a supprime le caractere '\n'  du bout */

        printf("Vous avez entrez la phrase %s\n", tab);
}
```

Exercice 1.1 Afficher le nombre de caractères de la phrase

Modifier le programme précédent pour qu'il affiche le nombre de caractères de la phrase entrée au clavier.

Correction :

```
#include <stdio.h>
#include <stdlib.h>

/* inclure le fichier de declarations des fonctions utilisateur */
#include <decla.h>

#define TAILLE 500

int main ()  {
        char phrase[TAILLE] = "";
        int nbCar, nbMot;

      /* initialisation de phrase */
        lirePhrase(phrase);

        nbCar = strlen(phrase);
        printf("nb de caracteres de la phrase : %d\n", nbCar);

        return EXIT_SUCCESS;
}
```

Exercice 1.2 Compter les mots

Compléter le programme afin d'afficher le nombre de mots qui figurent dans la phrase. Pour cet exercice, on considèrera qu'un mot est une séquence de caractères compris entre 2 espaces.

Correction : reportez-vous au corrige de la feuille de TD10.

Exercice 1.3 Affichage d'nformations sur les caractères

On va modifier le programme pour qu'il affiche, pour chaque caractère de la table ASCII (il y en a 128) :

— son rang dans la table ASCII

— sa valeur comme char (si son rang > 31)

— son nombre d'apparitions dans la phrase qui a été entrée.

Pour imprimer le nombre d'apparitions, on va déclarer dans la fonction `main` un tableau d'entiers `frequence[]` de 128 éléments. Dans ce tableau, on souhaite trouver au rang i, le nombre d'apparitions dans la phrase du caractère ASCII de rang i. On va initialiser ce tableau en parcourant la phrase dans une procédure d'initialisation ayant pour arguments la phrase comme chaine de caractères.

1. Ecrire une fonction qui retourne le nombre d'apparitions d'un caractère ASCII de rang i dans une chaine de caractères passée en second argument. Compiler son fichier de définition pour obtenir un code objet, et ajouter sa déclaration au fichier `decla.h`.
2. Se servir de la fonction précédente pour initialiser le tableau `frequence[]` dans le programme principal.

3. Ecrire une fonction qui affiche les informations suivante sur un caractère et une chaine passée en argument :
   — son rang dans la table ASCII et sa valeur comme char (si son rang > 31)
   — son nombre d'apparitions dans la chaine passée en second paramètre.
   Compiler son fichier de définition pour obtenir un code objet, et ajouter sa déclaration au fichier `decla.h`.
4. Modifier maintenant le programme pour qu'il fasse ce qui a été annoncé. Lancer maintenant la compilation pour lier ensemble vos différents codes objet et créer un fichier exécutable.

Correction : reportez-vous au corrigé de la feuille de TD10.

2 Fonction strcmp de la bibliothèque <string.h>

La fonction `strcmp` (*string compare*), compare deux chaines (terminant par `'\0'`) passées en argument, et retourne 0 si les deux chaines ont les mêmes caractères. Sa valeur de retour est
— inférieure à 0 si s1[] est inférieure à s2[] selon lórdre lexicographique,
— égale à 0 si s1[]et s2[] ont les mêmes caractères,
— supérieure à 0 si s1[] est supérieure à s2[] selon lórdre lexicographique.

Attention : l'ordre lexicographique ne doit pas être confondu avec l'ordre sur la longueur des chaines. On peut avoir s1 < s2 selon l'ordre lexicographique avec strlen(s1) > strlen(s2). Ex : "abbb" < "abc" alors que strlen("abbb") = 4 > strlen("abc") = 3.

Exercice 2.1 Fonction `compare`

Ecrire une fonction `int compare(const char* s1, const char* s2)` analogue à `strcmp`, mais qui retourne 1 si s1 est supérieure à s2 dans l'ordre lexicographique, 0 si s1 et s2 ont les mêmes caractères, et -1 sinon.
Algorithme : on va parcourir, les chaines s1 et s2 dans une boucle avec deux pointeurs p et q. Il faudra sarrêter si les valeurs pointées par p et q sont différentes, ou si on est arrivé au bout des 2 chaines (dans ce cas égales).
En sortie de boucle :
— si *p < *q , alors s1 < s2
— si *p > *q , à línverse s1 > s2
— si *p == *q , on est au bout et s1=s2

```
#include <stdio.h>
#include <stdlib.h>

int compare (char *s1, char *s2)
     char *p = s1;
     char *q = s2;

     while (*p == *q) {
     // tant que les car. points sont les mmes
     // et que ce nst pas la fin
                if (*p == 0)
                      return 0; // mmes caractres
                p++; q++;
     }
     // (*p != *q)
     if (*p < *q)
             return -1; // *p - *q est ngatif
     else return +1;   // *p - *q est positif
}

int main() {
     char s1[80] = "";
     char s2[80] = "";

     do {
             printf("Entrez deux chaines sur deux lignes:\n");
             scanf("%s\n%s", s1, s2);
             printf("compare(s1, s2): %d\n", compare(s1,s2));
      }  while ( compare(s1,s2) != 0);
      return EXIT_SUCCESS;
}
```