Université Savoie Mont Blanc, UFR SceM
INFO505, L3 INFO

October 24, 2023

Problème 1, Analyse de textes

(Tableaux mono-dimensionnels, chaînes de caractères, fonctions, structures de données, pointeurs, passage par adresse, hachage)

On se donne un texte quelconque:

```
char texte[] = "Le petit poucet

Il était une fois un bûcheron et une bûcheronne qui avaient sept enfants, tous garçons; l'aîné n'
    avait que dix ans, et le plus jeune n'en avait que sept.

On s'étonnera que le bûcheron ait eu tant d'enfants en si peu de temps ; mais c'est que sa femme
    allait vite en besogne, et n'en avait pas moins de deux à la fois. ...";
```

On voudrait analyser le texte, par exemple le découper en mots, faire des statistiques sur la fréquence des lettres et des mots, savoir si un mot est utilisé, etc. Cela permet par exemple d'indexer le texte pour des recherches futures, ou de comparer deux textes pour savoir s'il y a eu plagiat.

Exemple : (Longueur d'une chaîne de caractères) On rappelle qu'une chaîne de caractères C est une suite de caractères consécutifs en mémoire. Le caractère de valeur 0 (ou '\0') termine la chaîne de caractères. Ainsi, la fonction calculant la longueur d'une chaîne peut s'écrire:

```
int strlen( char s[] )
{
  int n = 0;
  while ( s[ n ] != 0 ) n++;
  return n;
}
```

Question 1. Quelle est la taille du mot le plus long ? Quel est le nombre de mots ? (On suppose pour le moment que les mots sont séparés par des espaces.)

```
int size_longest_word( char t[] ); // t is the input string
int nb_words         ( char t[] ); // t is the input string
```

Question 2. Il y a beaucoup de séparateurs de mots (espaces, ponctuations, fin de ligne). Mêmes questions que précédemment avec des séparateurs.

```
int is_in_string     ( char c, char s[] ); // return true is c is a letter in s
int size_longest_word( char t[], char sep[] ); // sep contains the separators
int nb_words         ( char t[], char sep[] ); // sep contains the separators
```

Remarques : Si vous voulez utiliser le type `bool` plutôt que `int` pour désigner un booléen, il faut inclure `<stdbool.h>` (depuis C99).

Question 3. Savoir si un caractère est un séparateur prend un temps proportionnel au nombre de séparateurs. Pouvez-vous imaginer une structure de données et des fonctions qui permettent de déterminer plus rapidement si un caractère est un séparateur?

Question 4. Est-il facile de substituer un mot par un autre dans un texte (en C) ?

Question 5. Enlever une lettre dans une chaîne de caractères. Par exemple, on voudrait enlever tous les espaces ' ', les guillemets '"', etc.

```
int remove_letter( char c, char s[] ); // return the number of removed letters
                                       // the array stored at s is modified
```

Question 6. On veut savoir combien de fois chaque mot apparaît, connaître le mot le plus fréquent. Est-ce facile d'écrire une fonction qui réalise ces opérations ?

Question 7. Découper un texte mot à mot.

On se donnera une structure `Token` pour stocker le mot courant, puis on écrira cinq fonctions pour manipuler les `Token`. Elles auront les prototypes ci-dessous:

```
// Initialise le token sur le premier mot du texte
void Token_init(Token* t, char s[], char sep[] );
// Retourne le mot courant stocké dans le token
char* Token_valeur( Token* t);
// Retourne vrai si on est à la fin du texte.
bool Token_fini( Token* t);
// Passe au mot suivant du texte.
void Token_suivant( Token* t);
// Indique que l'on a fini de se servir du token.
void Token_termine( Token* t);
```

```
int main()
{
  char s   [] = "Nous irons tous, au  paradis";
  char sep[] = { ' ', '\n', ',', '.', ';', ':',
       '?', '!', '\'', '\"' };
  Token tok;
  Token_init( &tok, s, sep );
  for( ; ! Token_fini( &tok );
         Token_suivant( &tok ) )
    printf( "%s|", Token_valeur( &tok ) );
  Token_termine( &tok );
}
```

et le programme affichera : `Nous|irons|tous|au|paradis|`

Question 8. Ecrivez maintenant une fonction qui détermine si un mot appartient à un texte.

```
// retourne 0 si s1 == s2, négatif si s1 < s2, positif si s2 < s1
int strcmp  ( char s1[], char s2[] );
// retourne vrai si mot est un préfixe de texte
bool prefixe( char mot[], char texte[] );
// retourne vrai si mot est une sous-chaîne de texte
bool belongs( char mot[], char texte[] );
```

Question 9. On propose les fonctions suivantes (dites de hachage)

```
int hash1( char* str ) // hachage simple
{
  int v = 0;
  while ( *str != 0 ) { v += *str++; }
  return v;
}
```

```
int hash2( char* str ) // hachage plus évolué
{
  static int coef[] = { 7, 1251, 89437, 817 };
  int v = 0, i = 0;
  for ( int i = 0; str[ i ] != 0; ++i )
      v += str[ i ] * coef[ i % 4 ];
  return v;
}
```

Trouvez deux mots différents pour lesquels `hash1` donne la même valeur. Même question pour `hash2` ? On va s'en servir pour trouver les mots.

Question 10. Proposez une structure de données pour stocker les mots d'un texte, le nb de fois où il apparait, leurs hachés. On supposera que le nombre maximum de mots est borné par une constante `MAX_NB_MOTS`.

```
struct SWord { char word[ 20 ]; int length; int h; int nb; };
typedef struct SWord Word;
struct SDico {  ...  };
typedef struct SDico Dico;
void  dico_init  ( Dico* pDico );
void  dico_insert( Dico* pDico, char* w );
Word* dico_search( Dico* pDico, char* word ); // retourne null si non trouvé
```

L'utilisation du dictionnaire pourrait ressembler à ci-dessous:

```
int main( int argc, char* argv[] )
{
  char* fname = argc > 1 ? argv[ 1 ] : "le-petit-poucet.txt";
  char sep[]  = { ' ', ',', ';', '.', ':', '\"', '\'', '(', ')', '!', '?', '-', '`', 13, 10, 0 };
  /* Lecture du fichier */
  FILE* F     = fopen( fname, "r" );
  char* buffer= (char*) malloc( 1000000 );
  int   nread = fread( buffer, 1, 1000000, F );
  printf( "%s: %d bytes read.\n", fname, nread );
  buffer[ nread ] = 0;
  fclose( F );

  /* Lecture mot a mot des caracteres. */
  Token T;
  Dico  D;
  dico_init( &D );
  for ( Token_init( &T, buffer, sep ); ! Token_fini( &T ); Token_suivant( &T ) )
    {
      char* word = Token_valeur( &T );
      dico_insert( &D, word );
    }
  ...
```