CS-119(a) – ICC-C Examen Pb Ouvert

2024-06-24

On veut construire une liste chaînée qui contient la suite des mots d'un texte dans l'ordre de leur apparition. Vous pouvez utiliser la fonction `strlen()` pour obtenir la longueur d'une chaîne de caractères, mais vous ne pouvez pas utiliser d'autres fonctions prédéfinies, comme `strcpy`, `memcpy`, ou `strtok` par exemple.

a) (2 points)

Écrivez une fonction qui reçoit en paramètre une chaîne de caractères `string` et qui retourne le nombre de caractères qui correspondent au plus long mot qui commence à `string[0]`. Un mot est une suite contiguë de caractères alphanumériques. Vous pouvez utiliser la fonction `isalnum(c)` qui retourne `1` si le caractère `c` est alphanumérique et `0` sinon.

Par exemple, `longueur_mot("le langage C")` ainsi que `longueur_mot("le?!lele")` doivent retourner `2` = la longueur du mot "le". Par contre l'appel `longueur_mot("... sofferte onde serene ...")` doit retourner `0`, car le premier caractère n'est pas alphanumérique. Pour la chaîne vide la valeur de retour doit également être `0`.

Solution de l'exercice a)

Solution proposée :

```c
int longueur_mot(const char *string)
{
    int delta;
    for (delta = 0; isalnum(string[delta]); delta++)
        ;
    return delta;
}
```

Il faut incrémenter un compteur du début de la chaîne tant que `isalnum` est vrai.

b) (2 points)

Écrivez une fonction qui reçoit en paramètre une chaîne de caractères `string` et un entier `n` positif et qui retourne un pointeur vers une nouvelle chaîne de caractères contenant une copie des `n` premiers caractères de `string`. Si `string` contient moins que `n` caractères, alors on copie tout le contenu de `string`.

Par exemple `copier("supercalifragilisticexpialidocious", 5)` retourne la chaîne `"super"`.

Solution de l'exercice b)

Solution proposée :

```
1  char *copier(const char *string, int n)
2  {
3      int len = strlen(string);
4      n = len < n ? len : n;
5      char *result = malloc(n + 1);
6      for (int i = 0; i < n; i++)
7      {
8          result[i] = string[i];
9      }
10     result[n] = '\0';
11
12     return result;
13 }
```

Points importants :

— Il faut trouver la taille de la nouvelle chaîne en comparant la taille de `string` et `n`. Attention au `'\0'` (donc `n+1` à la ligne 5).

— Utiliser `malloc()`.

— Pas oublier le caractère fin-de-chaîne `'\0'`.

On définit les deux struct suivantes :

```
typedef struct _cell
{
    char *mot;
    struct _cell *next;
} cell_t;
typedef struct _liste
{
    cell_t *premier;
    cell_t *dernier;
} liste_t;
```

c) (3 points)

Écrivez une fonction qui reçoit une liste et un pointeur vers une chaîne de caractères contenant un seul mot. Cette fonction rajoute une nouvelle cellule ayant pour contenu le pointeur `un_mot` **à la fin** de la liste chaînée `liste`. Cette fonction doit mettre à jour les champs `premier` et `dernier`. Pour une liste vide ces champs auront la valeur `NULL`.

Solution de l'exercice c)

```
void enfiler_mot(liste_t *plist,
                 char *mot)
{
    cell_t *wc = malloc(sizeof(cell_t));
    wc->mot = mot;
    wc->next = NULL;

    if (plist->premier == NULL)
    {
        plist->premier = wc;
    }
    else
    {
        plist->dernier->next = wc;
    }
    plist->dernier = wc;
}
```

Points importants :

- Allouer la nouvelle cellule et l'initialiser.
- Mettre à jour le dernier élément.
- Cas de base : la liste est vide – aussi mettre à jour le premier élément.

d) (3 points)

Utilisez les trois fonctions ci-dessus pour implémenter une fonction qui prend en paramètre une chaîne de caractères `texte` et retourne la liste de mots qui s'y trouvent dans l'ordre de leur apparition.

Solution de l'exercice d)

```
liste_t *créer_liste(const char *texte)
{
    liste_t *pliste = malloc(sizeof(liste_t));
    pliste->premier = pliste->dernier = NULL;
    const char *s = texte;
    while (*s)
    {
        int longueur = longueur_mot(s);
        if (longueur)
        {
            char *mot = copier(s, longueur);
            enfiler_mot(pliste, mot);
            s += longueur;
        }
        else
        {
            s++;
        }
    }
    return pliste;
}
```

Points importants :

— Utiliser `longueur_mot` et `copier` pour créer une nouvelle chaîne qui contient le prochain mot.

— Utiliser `enfiler_mot` pour populer la liste.

— Attention à la condition de fin de boucle.

Une solution récursive est aussi envisageable.

e) (2 points)

Implémentez une fonction qui libère la mémoire d'une liste.

Solution de l'exercice e)

```c
void libérer(liste_t *pliste)
{
    while (pliste->premier != NULL)
    {
        cell_t *p = pliste->premier;
        pliste->premier = pliste->premier->next;
        free(p->mot);
        free(p);
    }
    free(pliste);
}
```

Points importants :

- Itérer sur tous les éléments de la liste – attention à la condition d'arrêt.
- Libérer la chaîne `mot` qui est stockée dans chaque cellule.
- Ne pas effacer une cellule avant d'avoir récupéré le pointeur vers l'élément suivant !
- Libérer la liste elle-même.