IFT 1227 – Architecture des ordinateurs

Devoir 3

- À faire en groupe de **deux** étudiants.
- Remise : Le 5 avril 2018 avant minuit.
- Il y aura une pénalité de **10%** par jour de retard.

Programmation en assembleur MIPS

En ayant un texte composé des phrases, on veut découper ce texte en mots en formant un tableau de mots et ensuite, trier les éléments de ce tableau. Pour simplifier le traitement, on va définir un mot comme une entité formé des lettres de l'alphabet latin (26 lettres) en minuscules ou majuscules ou en mélange de deux se terminant par le symbole de fin de ligne `'\0'`. Un tableau de mots pourrait être modélisé par une table des adresses sur les chaines représentant des mots.

Votre programme doit gérer les textes avec un maximum de 300 caractères. Si la saisie dépasse la capacité du tampon il faudra afficher le message d'erreur et terminer le programme.

Pour accomplir cette tâche il faudra comme minimum créer les fonctions suivantes :

1. **`saisir`** – une fonction qui permet stocker le texte saisi par l'utilisateur dans un tampon de 300 octets. La fonction doit retourner la taille effective du texte saisi en nombre d'octets. Le texte pourrait contenir des lignes, c'est-à-dire que le symbole `'\n'` pourrait être présent. La fin de lecture est définie par les deux symboles `'\n''\n'` consécutifs.
2. **`bool lettre(char l)`** – une fonction qui permet identifier si le caractère passé en paramètre représente une lettre d'alphabet latin (26 lettres d'alphabet anglais) minuscule ou majuscule.
3. **`bool strCmp(str1, str2)`** – une fonction permettant de comparer les deux mots. **`str1`** et **`str2`** sont les adresses des chaînes respectives. La comparaison des lettres doit être effectuée selon leurs encodages ASCII.
4. **`decMots(texte, taille)`** – permet de découper en mots le texte passé en paramètre (l'adresse du tampon contenant le texte saisi et la taille du texte en nombre d'octets). Les valeurs de retour sont l'adresse de la table de mots et sa taille en nombre d'éléments. Le découpage doit être fait directement dans le texte original sans copier le contenu en mettant des `'\0'` a des endroits opportuns et en sauvegardant l'adresse de début de chaque mot dans la table de mots. La taille maximal de la table de mots est fixée à 600 octets (on considère un mot d'une lettre comme l'entité minimum qu'on pourrait avoir, donc, 150 mots pour un tampon de 300 octets et chaque mot est représenté par son adresse dans la table de mots, ce qui nous donne la taille de 4*150 = 600 octets).
   S'il y a des mots dupliqués, on doit avoir deux entrées dans la table de mots.

La saisie et le découpage consistent les deux tâches séparées.

5. **`trier(tabMots, taille`**) - permet de réorganiser les adresses associées à chaque mot de sorte que les mots soient triés en ordre alphabétique croissant. Le paramètre taille représente un nombre d'éléments de la table de mots.
6. **`afficher(tabMots, taille)`** – affiche le contenu d'une table de mots. Le paramètre **`taille`** correspond au nombre des mots effectivement stockés dans cette table. Les mots doivent être affichés 4 par ligne.
7. **`main`** – dans la fonction principale les actions suivantes doivent être faites :
    - **`jal saisir`** – saisir le texte
    - Afficher le texte saisi
    - **`jal decMots –`** découper le texte en mots et créer une table de mots
    - **`jal afficher –`** afficher les mots non triés
    - **`jal trier –`** trier une table de mots
    - **`jal  afficher –`** afficher les mots triés

Vous devez créer toutes les fonctions additionnelles que vous jugez utiles dans la réalisation de votre programme.

Les exemples des entrées et des sorties :

Entrée 1:

```
b_a

Vous avez saisi le texte suivant:
b_a

Tableau de mots non trié:

b a

Tableau de mots trié:

a b
```

Entrée 2:

```
"Assembly language: a low level language
that is a little more human-friendly than machine
language".

Vous avez saisi le texte suivant:

"Assembly language: a low level language
that is a little more human-friendly than machine
language".

Tableau de mots non trié:
```

```
Assembly  language  a         low
level     language  that      is
a         little    more      human
friendly  than      machine   language

Tableau de mots trié:

Assembly  a         a         friendly
human     is        language  language
language  level     little    low
machine   more      than      that
```

Entrée 3 :

```
homework123done
Done!!!

Vous avez saisi le texte suivant:

homework123done
Done!!!

Tableau de mots non trié:

homework done Done

Tableau de mots trié:

Done done homework
```

Voici l'ébauche du programme:

```
#segment de la mémoire contenant les données globales
.data
#tampon résérvé pour le texte
buffer:   .space    300
tabMots : .word     150
…
msg:      .asciiz "…"
#segment de la mémoire contenant le code
.text
main:
     …
     li   $v0,10    #terminer le programme
     syscall
#fonction saisir - n'oubliez pas les commentaires!
```

```
saisir:
     …
#fonction lettre
lettre:
     …
#fonction strCmp
strCmp:
     …
#fonction decMots
decMots:
     …
#fonction trier
trier:
     …
#fonction afficher
afficher:
     …
```

Directives

Dans toutes les questions, il faut obligatoirement respecter toutes les conventions MIPS et utiliser les bonnes pratiques de la programmation. Utilisez le bouton Help => onglet MIPS pour chercher toute l'information concernant MIPS.

L'information concernant les appels système se trouve dans le simulateur Mars, bouton Help => onglet MIPS => onglet Syscalls. Voici l'extrait des appels système :

Saisir une valeur à l'entrée standard et la faire stockée dans le registre `$t0` :

```
li   $v0, 5    # charger le numéro de service
syscall        # faire appel de ce service
# récupérer le résultat de $v0 dans $t0
add  $t0, $v0, $0
```

Imprimer une valeur stockée dans le registre `$t0` :

```
li  $v0, 1   # service 1 imprime un entier
# charger la valeur à imprimer dans le registre $a0.
add $a0, $t0, $zero # Notre valeur se trouve dans $t0
syscall             # faire appel du service
```

Imprimer une chaîne se trouvant à l'adresse msg

```
la   $a0, msg
li   $v0, 4    # Print string (Code 4 de syscall)
syscall        # faire appel du service
```

Remise

Remettre le fichier source: **`decouperTrier.asm`**

Évaluation

Bon fonctionnement du programme :		
	Reproduire les 3 essais cités dans l'énoncé	30
	Les essais complémentaires	30
Programme adéquat (algorithme, déroulement du programme)		20
Commentaires		10
Contient en en-tête la description complète de ce que doit faire le programme		5
Contient les descriptifs: but, date, auteur(s), adresse(s) de courriel, code(s) permanent(s)		5
Votre note:		/100