Nom :	Prénom :
N° Étudiant :	Groupe de TD :

TME Solo 2020 – 2021 – Sujet n°1

Architecture des ordinateurs 1 – LU3IN029

Durée : 0h50

Documents autorisés : Aucun document ni machine électronique n'est autorisé à l'exception du mémento MIPS.

Le barème indiqué pour chaque question n'est donné qu'à titre indicatif. Le barème total est lui aussi donné à titre indicatif. Merci de rendre la feuille.

Étapes préliminaires à suivre scrupuleusement :

1. Créez un répertoire pour le TME solo à la racine de votre compte qui devra contenir les codes réalisés, en tapant les commandes suivantes dans un terminal :
   `> cd`
   `> mkdir TMESolo_<nom>` (où <nom> est votre nom en minuscule sans espace ni accent)
   `> chmod -R go-rwx .`
   Attention : cette dernière commande est très importante car elle empêche d'autres utilisateurs d'accéder à vos fichiers. Si vous ne la faites pas correctement et qu'un autre étudiant copie vos fichiers, vous risquez d'obtenir la note de 0.
   Remarque : la détection de plagiat sera faite automatiquement par logiciel.
2. Ouvrez un éditeur de texte et sauvegardez le fichier `~/TMESolo_<nom>/rapport.txt`. Ce fichier contiendra votre rapport de TME solo. Vous devrez y copier-coller tous vos codes réponses. Vous indiquerez bien le numéro de la question auquel se rapporte chaque code.
   Attention : ce fichier doit se trouver dans le répertoire `TMESolo_<nom>` et s'appeler `rapport.txt`
3. Lancez Mars et composez le TME solo en répondant aux questions ci-dessous. Enregistrez bien tous vos codes dans le répertoire `TMESolo_<nom>`.

Soumission de votre devoir à la fin du TME solo

1. Créez une archive contenant vos codes et votre rapport avec les commandes suivantes :
   `> cd`
   `> tar -cvf tmesolo_<nom>.tar TMESolo_<nom>/`
2. Déposez l'archive dans Moodle (dans la section "rendu de TME" il y a une remise de devoir intitulée TME solo). Attention vous devez valider votre soumission et vous ne pourrez soumettre qu'une seule fois.

Consigne importante : il vous est demandé de mettre des commentaires dans vos codes pour indiquer la correspondance entre les registres utilisés et les variables des programmes.

Le TME solo comporte une question principale sur 15 points (la note finale est sur 20) et une question pour atteindre 20 avec 5 points bonus.

Exercice 1 : Bon parenthésage de chaîne – 25 points

Question 1.1 : 15 points

Écrivez dans un fichier nommé `Q1.s` (et enregistré dans le répertoire pour le TME solo) un programme assembleur correspondant au code C ci-dessous. Ce programme teste, via la fonction `bon_parenthesage` si une chaîne entrée au clavier est bien parenthésée. Un message différent est affiché selon le cas.

```
char s[20];
char ch_ok[]  = "bien parenthésé\n";
char ch_nok[] = "mal parenthésé\n";

int bon_parenthesage(char * ch){
  int d = 0;
  int i = 0;

  while (ch[i] != '\0'){
    if (ch[i] == '('){
        d += 1;
    }
    if (ch[i] == ')'){
        d += -1;
    }
    if (d < 0){
      return d;
    }
    i++;
  }

  return d;
}

void main(){
  int ok;
  scanf("%20s", s);  /* max 20 caractères, caractère nul inclus */

  printf("%s", s);   /* affichage de la chaine lue*/
  ok = bon_parenthesage(s);

  if (ok == 0){
    printf("%s", ch_ok);
  }
  else{
    printf("%s",ch_nok);
  }
  exit();
}
```

Testez votre programme. Par exemple, les chaînes `"((a)(b)((c)))"` et `"aaa"` sont bien parenthésées alors que `"(((()()))"` et `"(ab))"` ne le sont pas.
N'oubliez pas de copier-coller votre code assembleur dans votre rapport.

Solution:

Programme principal et les données globales :

```
.data
s:      .space 20
ch_ok:  .asciiz   "bien parenthésé\n"
ch_nok: .asciiz   "mal parenthésé\n"
```

```
.text
   # nv = 1, na = 1
   addiu $29, $29, -8

   lui $4, 0x1001    # scanf
   ori $5, $0, 20
   ori $2, $0, 8
   syscall

   lui $4, 0x1001    # printf
   ori $2, $0, 4
   syscall
   # bon_parenthesage(s)
   lui $4, 0x1001
   jal bon_parenthesage
   sw  $2, 4($29)           # seulement si pas optimise

   lw  $2, 4($29)     # seulement si pas optimise
   bne $2, $0, nok
   lui $4, 0x1001           # affiche ch_ok
   ori $4, $4, 20
   ori $2, $0, 4
   syscall
   j epilogue_main
nok :
   lui $4, 0x1001          # affiche ch_nok
   ori $4, $4, 37
   ori $2, $0, 4
   syscall

epilogue_main :
   addiu $29, $29, 8
   ori   $2, $0, 10
   syscall
```

Solution:

Fonction `bon_parenthesage` :

```
bon_parenthesage :
   # nr = 0 + $31, nv = 2, na = 0
   addiu $29, $29, -12
   sw    $31, 8($29)

   xor   $2,  $2, $2    # $2 = d
   xor   $8,  $8, $8    # $8 = i
   ori   $10, $0, '('
   ori   $11, $0, ')'

while :
   addu  $12, $4, $8      # @ch[i]
   lbu   $12, 0($12)      # lb est ok aussi
   beq   $12, $0,  fin_while

   bne   $12, $10, if2
   addiu $2,  $2,  1
   j     if3
if2 :
   bne   $12, $11, if3
   addi  $2,  $2, -1
if3 :
   bltz  $2,  fin_while
fin_alt :
   addiu $8, $8, 1         # i++
   j     while
fin_while :
```

```
lw    $31, 8($29)
addiu $29, $29, 12
jr    $31
```

Question 1.2 : 10 points

Copiez votre fichier contenant le code réponse de la question précédente en l'appelant `Q2.s` et ouvrez ce dernier dans Mars. On souhaite ajouter au code précédent la fonction `bon_parenthesage_rec` calculant si une chaîne est bien parenthésée de manière récursive. Le programme principal appelle désormais les deux fonctions à la suite.

```
int bon_parenthesage_rec(char * ch, int index){
  int d;
  if (ch[index] == '\0'){
     return 0;
  }
  d = bon_parenthesage_rec(ch, index + 1);
  if (d > 0){
     return d;
  }
  if (ch[index] == ')'){
     return d - 1;
  }
  if (ch[index] == '('){
     return d + 1;
  }
  else { return d; }
}

void main(){
  int ok;
  scanf("%20s", s);  /* max 20 caractères caractère nul inclus*/

  printf("%s", s);   /* affichage de la chaine lue*/
  ok = bon_parenthesage(s);

  if (ok == 0){
    printf("%s", ch_ok);
  }
  else{
    printf("%s",ch_nok);
  }
  // ajout par rapport à la question précédente
  ok = bon_parenthesage_rec(s,0);
  if (ok == 0){
    printf("%s", ch_ok);
  }
  else{
    printf("%s",ch_nok);
  }
  exit();
}
```

Programmez la fonction `bon_parenthesage_rec` (8 points).

Ensuite, modifiez le programme principal (2 points).

Testez votre programme pour vérifier qu'il fonctionne.

N'oubliez pas d'ajouter des commentaires à votre code.

Copiez votre programme réponse dans votre rapport.

Solution:

Programme principal modifié et ses mêmes données globales :

```
.data
s:       .space 20
ch_ok:   .asciiz  "bien parenthésé\n"
ch_nok:  .asciiz "mal parenthésé\n"
.text
   # nv = 1, na = 2
   addiu $29, $29, -12  #<--- modifie

   lui   $4,  0x1001    # scanf
   ori   $5,  $0, 19
   ori   $2,  $0, 8
   syscall

   lui   $4,  0x1001    # printf
   ori   $2,  $0, 4
   syscall

   lui   $4,  0x1001
   jal   bon_parenthesage
   sw    $2,  4($29)

   lw    $2,  4($29)    # seulement si pas optimise
   bne   $2,  $0, nok
   lui   $4,  0x1001
   ori   $4,  $4, 20
   ori   $2,  $0, 4
   syscall
   j ajout           #<---- modifie
nok:
   lui   $4,  0x1001
   ori   $4,  $4, 37
   ori   $2,  $0, 4
   syscall
#--------------- ajout
ajout:
   lui   $4,  0x1001
   xor   $5,  $5, $5
   jal   bon_parenthesage_rec
   sw    $2,  4($29)

   lw    $2,  4($29)    # seulement si pas optimise
   bne   $2,  $0, nok2
   lui   $4,  0x1001
   ori   $4,  $4, 20
   ori   $2,  $0, 4
   syscall
   j     epilogue_main
nok2:
   lui   $4,  0x1001
   ori   $4,  $4, 37
   ori   $2,  $0, 4
   syscall
#--------------- fin ajout
epilogue_main:
   addiu $29, $29, 12
   ori   $2, $0, 10
   syscall
```

Solution:

Fonction `bon_parenthesage_rec` :

```
   # nr = 0 + $31, nv = 1, na = 2
   addiu $29, $29, -16
   sw    $31, 12($29)

   add   $9,  $4,  $5       # @ch[i]
```

```
   lbu   $9,  0($9)
   bne   $9,  $0,  not_zero
   xor   $2,  $2,  $2       # cas ch[index] == 00
   j     bpr_epilogue
not_zero:
   sw    $4,  16($29)    # sauvegarde param
   sw    $5,  20($29)
   addi  $5,  $5, 1      # index + 1
   jal   bon_parenthesage_rec
   sw    $2,  8($29)     # sauvegarde resultat dans d, peut etre optimisé

   blez  $2,  d_neg
   j     bpr_epilogue   # cas d positif => fin
d_neg:
   lw    $4,  16($29)
   lw    $5,  20($29)
   add   $8,  $4, $5
   lbu   $8,  0($8)       # ch[index]
   ori   $10, $0, ')'
   bne   $8,  $10, not_par_close
   addi  $2,  $2, -1
   j     bpr_epilogue
not_par_close:
   ori   $10, $0, '('
   bne   $8,  $10, bpr_epilogue
   addi  $2,  $2, 1
bpr_epilogue:
   lw    $31, 12($29)
   addiu $29, $29, 16
   jr $31
```