

## TP2 ASSEMBLEUR

**Les codes des deux questions sont fournis en annexe pour des questions de clarté.**

1. Suppression des espaces.

Cette partie du TP nous aura permis de découvrir deux points importants en assembleur :

- La manipulation de chaînes de caractères ;
- Le parcours d'une structure avec des pointeurs.

En effet, nous avons dû implémenter une chaîne de caractère pour en supprimer les espaces. Le parcours a été réalisé à l'aide de deux pointeurs sur le début et la fin de la chaîne. Nous avons été surpris de voir qu'il est très simple de déclarer une chaîne de caractère en assembleur. Nous pensions à la base qu'étant un langage de très bas niveau, il faudrait déclarer les caractères un par un, alors qu'il est en fait possible de remplir l'ensemble de la chaîne en une seule ligne de code.

Concernant le parcours de la chaîne à l'aide des pointeurs, celui-ci a été beaucoup plus simple que prévu. Le fait de manipuler directement les registres avec les pointeurs permet de se défaire d'un niveau d'abstraction que l'on peut observer dans les langages de plus haut niveau tels que le C. La manipulation des pointeurs dans ce dernier peut être confuse par moments, car il n'est pas toujours aisé de bien se représenter à quoi ceux-ci correspondent.

Vous trouverez ci dessous l'état de la mémoire avant et après l'exécution du programme appliqué à la chaîne suivante : « H e l l o W o r l d » :

```

000000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000080 00 00 a0 e1 34 10 9f e5 30 20 9f e5 01 00 d1 e4 ....4...0....
000000090 20 00 50 e3 00 00 00 00 0a 01 00 c2 e4 00 00 50 e3 ..P.....P...
0000000a0 f9 ff ff 1a fe ff ff ea 20 48 20 65 20 6c 20 6c .....H e l l
0000000b0 20 6f 20 20 57 20 6f 20 72 20 6c 20 64 00 00 00 ..o W o r l d...
0000000c0 a8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000000d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
0000000f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

```

00000030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000040 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000050 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000060 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000070 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
00000080 00 00 a0 e1 34 10 9f e5 30 20 9f e5 01 00 d1 e4 ....4...0....
00000090 20 00 50 e3 00 00 00 0a 01 00 c2 e4 00 00 50 e3 ...P.....P...
000000a0 f9 ff ff 1a fe ff ff ea 48 65 6c 6c 6f 57 6f 72 .....HelloWor
000000b0 6c 64 00 20 57 20 6f 20 72 20 6c 20 64 00 00 00 ld.W o r l d...
000000c0 a8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000d0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000e0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....
000000f0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 .....

```

## 2. Tri par sélection.

Lors de cet exercice, nous avons eu à traduire un algorithme qui nous était proposé, nous laissant donc une faible marge de manœuvre par rapport à ce dont nous avons l'habitude. Cependant, nous avons constaté qu'il était très simple de le coder. Car bien que le langage soit peu intuitif au départ, une fois maîtrisé il se révèle simple d'utilisation, puisque rien n'est caché à l'utilisateur.

Vous trouverez ci dessous l'état de la mémoire avant et après l'exécution du programme appliqué à la liste suivante :

Avant : (les valeurs sont à partir de l'adresse e0. Elles commencent à 06).

[illegible]

Pendant :

[illegible]

Après :

[illegible]

On constate que les valeurs ont bien été triées par ordre croissant par la méthode du tri par insertion.