

ARCHITECTURE DES ORDINATEURS

TD : 06

STRUCTURES

Exercice 1 : Structure en mémoire

Écrivez le code assembleur équivalent au code C suivant :

```
struct t {  
    long i;  
    long j;  
};  
  
struct t a = { 2, 3 };  
struct t b = { 4, 1 };
```

Exercice 2 : Liste

Réalisez en assembleur l'équivalent du code C suivant :

```
struct list {  
    long x;  
    long y;  
    struct list *next;  
};  
  
struct list l2 = {  
    .x = 4,  
    .y = 6,  
    .next = NULL,  
};  
  
struct list l = {  
    .x = 1,  
    .y = 2,  
    .next = &l2,  
};  
  
long f(struct list *l) {  
    return l->x;  
}  
  
long g(struct list *l) {  
    return l->y;  
}
```

```

struct list *h(struct list *l) {
    return l->next;
}

long length(struct list *l) {
    long n = 0;
    while (l) {
        n++;
        l = l->next;
    }
    return n;
}

```

Pour des raisons d'efficacité, ne définissez pas d'emplacement mémoire pour `l`, mais utilisez un registre à la place.

Exercice 3 : Arbre

Utilisons cette structure

```

struct tree {
    long x;
    long y;
    struct tree *left;
    struct tree *right;
};

```

qui sert à établir un arbre binaire de recherche, implémentez une fonction de dictionnaire, i.e. une fonction

```
long dico(long cherche_x, struct tree *t)
```

qui cherche le nœud de l'arbre donc le champ `x` est égal à `cherche_x`, et en retourne le champ `y`.

Note : puisque c'est un arbre binaire de recherche, il n'y a pas besoin de récursion : il suffit de parcourir l'arbre en allant du côté gauche si `cherche_x < t->x`, et en allant du côté droit si `cherche_x > t->x`. Si l'on ne trouve pas `cherche_x`, retournez `-1`. Écrivez d'abord la version C, puis la version assembleur

Pour aller plus loin...

Écrire une fonction récursive `long hauteur(struct tree *t)` qui calcule la hauteur de l'arbre.