Structures de données

1 Listes

En C, pour définir une liste chaînée d'entiers, on utilise typiquement :

```
struct cell {
    long x;
    struct cell *next;
};
```

avec pour convention que dans le dernier maillon de la liste chaînée, le champ *next* est NULL.

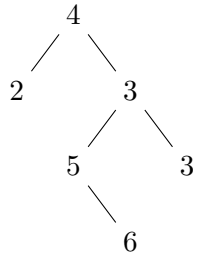
1. Le fragment de code Y86 ci-contre définit une liste chaînée u : dessiner cette liste avec les conventions habituelles de représentation manuelle d'une liste.
2. Soit une fonction C : `long somme(struct cell *l)` qui calcule la somme des éléments d'une liste chaînée dont l'adresse du premier maillon est le paramètre l. Ecrire le code de cette fonction en assembleur Y86. Tester avec un programme `main` qui calcule la somme des éléments de la liste u.
3. Définir une seconde liste chaînée v contenant les entiers 10, 11, 12 en entremêlant (pour le plaisir) les cellules de v avec celles de u.
4. Soit une fonction C qui concatène deux listes : `void concat(struct cell *l1, struct cell *l2)` en faisant pointer le dernier maillon de la liste $l1$ (supposée non vide) vers le premier maillon de la liste $l2$. Ecrire le code de cette fonction en assembleur Y86. Tester avec un programme `main` qui concatène les listes u et v.

```
main:   irmovl  0x200,%esp
        [...]
        halt

somme:  [...]
        ret

.align 4
u:      .long   1
        .long   0x100
.pos 0x100
        .long   3
        .long   0x120
        .long   7
        .long   0
.pos 0x120
        .long   5
        .long   0x108
```

2 Arbres

En C, pour définir un arbre, on utilise typiquement :

```
struct node {
    long x;
    struct node *left;
    struct node *right;
};
```



avec pour convention que si *left* ou *right* est NULL, c'est qu'il n'y a pas de fils de ce côté-là. En utilisant les mêmes principes que pour les listes :

1. représenter en mémoire (aussi simplement que possible) l'arbre u dessiné ci-dessus ;
2. écrire en assembleur Y86 une fonction récursive `long sum(struct node *t)` qui calcule la somme des éléments de l'arbre t. Tester en écrivant un programme `main` qui applique cette fonction à l'arbre u.

Listes : corrigé

Le programme de test concatène deux listes chaînées d'entiers, puis calcule la somme des éléments de la liste ainsi obtenue :

```
0x000: 308400020000 | test:   irmovl  0x200,%esp
0x006: 2045         |         rrmovl  %esp,%ebp
0x008: 30808c000000 |         irmovl  v,%eax
0x00e: a008         |         pushl   %eax
0x010: 308084000000 |         irmovl  u,%eax
0x016: a008         |         pushl   %eax
0x018: 804e000000   |         call    concat
0x01d: 8025000000   |         call    somme
0x022: 2054         |         rrmovl  %ebp,%esp
0x024: 10           |         halt
                    |
0x025: a058         | somme:  pushl   %ebp
0x027: 2045         |         rrmovl  %esp,%ebp
0x029: 6300         |         xorl    %eax,%eax       # s = 0
0x02b: 501508000000 |         mrmovl  8(%ebp),%ecx    # pointeur p
0x031: 6211         | sb:     andl    %ecx,%ecx       # fin de liste ?
0x033: 734b000000   |         je      sf              # oui
0x038: 502100000000 |         mrmovl  (%ecx),%edx     # p.val
0x03e: 6020         |         addl    %edx,%eax       # s += p.val
0x040: 501104000000 |         mrmovl  4(%ecx),%ecx    # p = p.next
0x046: 7031000000   |         jmp     sb              # boucle
0x04b: b058         | sf:     popl    %ebp
0x04d: 90           |         ret
                    |
0x04e: a058         | concat: pushl   %ebp
0x050: 2045         |         rrmovl  %esp,%ebp
0x052: 501508000000 |         mrmovl  8(%ebp),%ecx    # pointeur p
0x058: 6300         |         xorl    %eax,%eax       # pointeur q
0x05a: 6211         | cb:     andl    %ecx,%ecx       # fin de liste ?
0x05c: 736e000000   |         je      cf1             # oui
0x061: 2010         |         rrmovl  %ecx,%eax       # q = p
0x063: 501104000000 |         mrmovl  4(%ecx),%ecx    # p = p.next
0x069: 705a000000   |         jmp     cb              # boucle
0x06e: 6200         | cf1:    andl    %eax,%eax       # q valide?
0x070: 7381000000   |         je      cf2             # non
0x075: 50150c000000 |         mrmovl  12(%ebp),%ecx   # v
0x07b: 401004000000 |         rmmovl  %ecx,4(%eax)    # q.next = v
0x081: b058         | cf2:    popl    %ebp
0x083: 90           |         ret
                    |
0x084:              | .align 4
0x084: 01000000     | u:      .long   1
0x088: 00010000     |         .long   0x100
0x08c: 0a000000     | v:      .long   10
0x090: 10010000     |         .long   0x110
0x100:              | .pos 0x100
0x100: 03000000     |         .long   3
0x104: 20010000     |         .long   0x120
0x108: 07000000     |         .long   7
0x10c: 00000000     |         .long   0
0x110: 0b000000     |         .long   11
0x114: 30010000     |         .long   0x130
0x120:              | .pos 0x120
0x120: 05000000     |         .long   5
0x124: 08010000     |         .long   0x108
0x130:              | .pos 0x130
0x130: 0c000000     |         .long   12
0x134: 00000000     |         .long   0
```

Arbres : corrigé

```
test:   irmovl  0x200,%esp
        rrmovl  %esp,%ebp
        irmovl  a,%eax
        pushl   %eax
        call    somme
        rrmovl  %ebp,%esp
        halt

somme:  pushl   %ebp
        rrmovl  %esp,%ebp
        mrmovl  8(%ebp),%ecx    # racine r
        xorl    %eax,%eax       # s = 0
        andl    %ecx,%ecx       # arbre vide ?
        je      sf              # oui
        mrmovl  4(%ecx),%ecx    # fils gauche
        pushl   %ecx
        call    somme           # sg = somme(r.gauche)
        popl    %ecx
        mrmovl  8(%ebp),%ecx    # r
        mrmovl  (%ecx),%edx     # r.x
        addl    %edx,%eax       # r.x + sg
        pushl   %eax            # sauvegarde
        mrmovl  8(%ecx),%ecx    # fils droit
        pushl   %ecx            # argument
        call    somme           # sd = somme(r.droite)
        popl    %ecx
        popl    %edx            # r.x + sg
        addl    %edx,%eax       # r.x + sg + sd
sf:     rrmovl  %ebp,%esp
        popl    %ebp
        ret

.align 4
a:      .long   4
        .long   g
        .long   d
g:      .long   2
        .long   0
        .long   0
d:      .long   3
        .long   dg
        .long   dd
dg:     .long   5
        .long   0
        .long   dgd
dd:     .long   3
        .long   0
        .long   0
dgd:    .long   6
        .long   0
        .long   0
```