Logiciel de Base

Ensimag 1A Apprentissage

Examen — Juin 2013

Consignes :

- Durée : 2h.
- Tous documents autorisés.
- Le barème est donné à titre indicatif.
- On attend des réponses courtes et pertinentes, inutile de recopier le cours.
- Les parties sont indépendantes les unes des autres. La plupart des questions du problème sont également indépendantes. Pensez à lire le sujet en entier avant de commencer à répondre.

Consignes relatives à l'écriture de code C et assembleur Pentium :

- Pour chaque question, une partie des points sera affectée à la clarté du code et au respect des consignes ci-dessous.
- Pour les questions portant sur la traduction d'une fonction C en assembleur, on demande d'indiquer en commentaire chaque ligne du programme C original avant d'écrire les instructions assembleur correspondantes.
- Pour améliorer la lisibilité du code assembleur, il est conseillé d'utiliser des constantes (i.e. déclarations du type x=42) pour les déplacements relatifs à `%ebp` (i.e. paramètres des fonctions et variables locales). Par exemple, si une variable locale s'appelle `var` en langage C, on y fera référence avec `var(%ebp)`.
- Sauf indication contraire dans l'énoncé, on demande de traduire le code C en assembleur de façon systématique, sans chercher à faire la moindre optimisation : en particulier, **on stockera les variables locales dans la pile** (pas dans des registres), comme le fait le compilateur C par défaut.
- On respectera les conventions de gestions des registres Intel vues en cours, c'est à dire :
  - `%eax`, `%ecx` et `%edx` sont des registres scratch ;
  - `%ebx`, `%esi` et `%edi` ne sont pas des registres scratch.

1 Exercices sur le langage d'assemblage et GDB

On considère le programme assembleur suivant :

```
.text
        .globl main
main:
        pushl %ebp
```

```
        movl %esp, %ebp
        pushl %ebx
        pushl %edi

        movl $t, %ebx
        movl $after_t, %edi
after_assign:
        addl $-4, %edi
while:
        cmpl %edi, %ebx
        jae end_while

        movl (%ebx), %eax
        movl (%edi), %ecx
        movl %eax, (%edi)
        movl %ecx, (%ebx)

        addl $4, %ebx
        addl $-4, %edi

        jmp while
end_while:

        popl %edi
        popl %ebx
end_array:
        leave
        ret

.data
t:      .int 1, 9, 3, 12, 5
after_t:
fmt:    .asciz "%d - "
```

On assemble ce programme et on l'exécute dans GDB. Une trace incomplète est donnée ci-dessous :

```
(gdb) break after_assign
Breakpoint 1 at 0x80483a3: file array.S, line 12.
(gdb) run
Starting program: a.out

Breakpoint 1, after_assign () at array.S:12
12              addl $-4, %edi
(gdb) print /x $ebx
$1 = 0x8049588
(gdb) print /x $edi
$2 = 0x804959c
(gdb) next
while () at array.S:14
14              cmpl %edi, %ebx
(gdb) print /x $edi
$3 = 0x________
```

```
(gdb) x/16x $esp
0xbfffe940:     0x________      0x________      0xbfffe9c8      0xb7eafca6
0xbfffe950:     0x00000001      0xbfffe9f4      0xbfffe9fc      0xb7fe19b8
0xbfffe960:     0xbfffe9b0      0xffffffff      0xb7ffeff4      0x08048219
0xbfffe970:     0x00000001      0xbfffe9b0      0xb7ff0966      0xb7fffab0
(gdb) print $ebp
$4 = (void *) 0x________
(gdb) break end_array
Breakpoint 2 at 0x80483bc: file array.S, line 31.
(gdb) continue
Continuing.

Breakpoint 2, end_array () at array.S:31
31              leave
(gdb) print /x $edi
$5 = 0x0
(gdb) print /x $ebx
$6 = 0xb7fdbff4
(gdb) print /x $eax
$7 = 0x________
(gdb) x/1x 0x8049588
0x8049588 <t>:  0x________
```

Question 1 (2 points) *6 valeurs ont été remplacées par des* `________`*. Donnez ces 6 valeurs, avec pour chacune une explication d'une ou deux phrases.*

```
GNU gdb (GDB) 7.0.1-debian
Copyright (C) 2009 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "i486-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from a.out...done.
(gdb) break after_assign
Breakpoint 1 at 0x80483a3: file array.S, line 12.
(gdb) run
Starting program: a.out

Breakpoint 1, after_assign () at array.S:12
12              addl $-4, %edi
Current language:  auto
```

```
The current source language is "auto; currently asm".
(gdb) print /x $ebx
$1 = 0x8049588
(gdb) print /x $edi
$2 = 0x804959c
(gdb) next
while () at array.S:14
14              cmpl %edi, %ebx
(gdb) print /x $edi
$3 = 0x8049598
(gdb) x/16x $esp
0xbfffe940:     0x00000000      0xb7fdbff4      0xbfffe9c8      0xb7eafca6
0xbfffe950:     0x00000001      0xbfffe9f4      0xbfffe9fc      0xb7fe19b8
0xbfffe960:     0xbfffe9b0      0xffffffff      0xb7ffeff4      0x08048219
0xbfffe970:     0x00000001      0xbfffe9b0      0xb7ff0966      0xb7fffab0
(gdb) print $ebp
$4 = (void *) 0xbfffe948
(gdb) break end_array
Breakpoint 2 at 0x80483bc: file array.S, line 31.
(gdb) continue
Continuing.

Breakpoint 2, end_array () at array.S:31
31              leave
(gdb) print /x $edi
$5 = 0x0
(gdb) print /x $ebx
$6 = 0xb7fdbff4
(gdb) print /x $eax
$7 = 0x9
(gdb) x/1x 0x8049588
0x8049588 <t>:  0x00000005
```

2 Implémentation d'une fonction simple en assembleur

On considère la fonction C suivante :

```
void add(int *dest, int incr) {
        *dest += incr;
}
```

Question 2 (1 point) *Traduisez cette fonction en assembleur.*

3 Manipulation de liste

Soit une liste chaînée définie par :

```
struct cellule {
        int val;
        struct cellule *suiv;
};
```

Question 3 (1 point) *Écrire en C une fonction, la plus simple possible (une fonction plus compliqué ne donnera pas la note maximum), qui ajoute une valeur de type* `int` *en tête d'une liste.*

```
void ajout_en_tete(int val, struct cellule **l) {
        struct cellule *cell = malloc(sizeof(struct cellule));
        cell->val = val;
        cell->suiv = (*l);
        (*l)->suiv = cell;
}
```

4 Problème : Implémentation d'une liste chaînée avec ajout en fin et concaténation efficace

Une liste chaînée est une structure de donnée relativement flexible : on peut faire un ajout en tête en $O(1)$, ajouter un élément en milieu de liste assez simplement, ... Malheureusement, des opérations communes comme la concaténation de deux listes ou l'ajout en fin sont coûteuses ($O(n)$). On peut rendre ces opérations moins coûteuses en conservant un pointeur sur le dernier élément de la liste. Dans cette partie, nous implémenterons cette solution en représentant une liste (de `char`) par une structure contenant un pointeur sur le premier élément et un pointeur sur le dernier. La déclaration en C est la suivante :

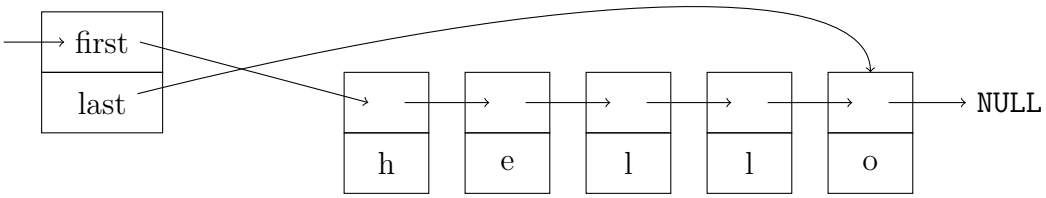
```
struct cell;

struct list {
        struct cell *first;
        struct cell *last;
};

// Internal use only
struct cell {
        struct cell *next;
        char val;
};
```

Une structure `struct cell` occupe 8 octets : 4 pour le pointeur `next`, 1 pour la valeur `val`, et 3 inutilisés (padding).

La chaîne « hello » serait représentée par la liste suivante :



La chaîne vide est représentée par une structure ou `first == NULL` et `last == NULL`.

Nous allons implémenter un module C permettant de manipuler cette structure de données. Une spécification partielle du module est donnée en annexe A.

On suppose les constantes suivantes définies en assembleur :

```
first=0
last=4
val=4
next=0
```

Pour nous convaincre que la concaténation est simple avec cette implémentation, voici une implémentation en C de la fonction `list_cat` :

```
void list_cat(struct list *dest, struct list *src) {
        if (dest->last == NULL) {
                dest->first = src->first;
        } else {
                dest->last->next = src->first;
        }
        if (src->last != NULL) {
                dest->last = src->last;
        }
        free(src);
}
```

Question 4 (0.5 point) *À quoi servent les deux `if` ?*

Aux cas des listes vides dans src et dest.

Question 5 (1 point) *Traduisez la fonction `list_cat` en assembleur.*

cf. `list_tail_asm.S`.

Voici une fonction en assembleur :

```
// void list_XXX(...) {
        .globl list_XXX
list_XXX:
        pushl %ebp
        movl %esp, %ebp
        subl $4, %esp
```

```
//      ...
        movl 8(%ebp), %eax
        movl first(%eax), %eax
        movl %eax, -4(%ebp)
for_XXX:
//      ...
        cmpl $0, -4(%ebp)
        je fin_for_XXX
//      ...
        movl -4(%ebp), %eax
        pushl val(%eax)
        pushl $fmt_c
        call printf
        addl $8, %esp
//      ...
        movl -4(%ebp), %eax
        movl next(%eax), %eax
        movl %eax, -4(%ebp)
//      ...
        jmp for_XXX
fin_for_XXX:
//      ...
        leave
        ret
.data
fmt_c:  .asciz "%c"
```

les commentaires `// ...` correspondent à des lignes de C.

Question 6 (0.5 point) *Que fait cette fonction ?*

C'est un affichage de la liste.

Question 7 (1.5 points) *Traduisez cette fonction de l'assembleur vers le langage C.*

cf. `list_print`.

Question 8 (1.5 points) *Écrire en assembleur une version optimisée de cette fonction, qui stocke la variable locale dans un registre au lieu de la stocker dans la pile. Réduisez autant que possible le nombre d'instructions assembleur.*

cf. `list_print_opt`.

On va maintenant implémenter un itérateur sur une liste, avec la fonction `list_iterate`. Cet itérateur est générique, et prends donc en argument un pointeur sur fonction. L'itérateur doit pouvoir modifier les éléments de la liste, donc la fonction reçoit les éléments de la liste (les `char`) par adresse. On veut pouvoir conserver un état entre deux appels à la fonction, donc en plus des éléments de la liste, la fonction reçoit

un pointeur sur une donnée quelconque (un `void *`, que la fonction devra caster). Le dernier argument de `list_iterate` est le pointeur passé en argument à la fonction à chaque itération (un exemple est fourni plus bas).

Question 9 (1.5 points) *Implémentez la fonction `list_iterate` en C.*

Voici un exemple d'utilisation de `list_iterate` :

```
static void f(char *c, void *arg) {
        (void)c;
        size_t *XXX = (size_t *)arg;
        ++(*XXX);
}

size_t list_YYY_iter(struct list *s) {
        size_t XXX = 0;
        list_iterate(s, f, &XXX);
        return XXX;
}
```

Question 10 (0.5 point) *Quelle est la signification du mot clé `static` dans ce contexte ? Pourquoi est-il utilisé pour la fonction `f` ?*

`f` n'est pas utilisé en dehors du fichier, `static` permet de ne pas exporter le symbole et donc d'éviter les clash de noms au link.

Question 11 (0.5 point) *Quelle est l'utilité de la ligne `(void)c;` ?*

Éliminer explicitement un warning sur argument non utilisé (mais nécessaire au bon typage de f).

Question 12 (2 points) *Que fait la fonction `list_YYY_iter` ? Traduisez les fonctions `f` et `list_YYY_iter` en assembleur (pour la fonction `f`, il peut être plus simple de faire une traduction globale plutôt que ligne à ligne).*

cf. `list_len_iter` et `plus_one`.

Question 13 (1.5 points) *Implémentez en C la fonction `list_to_upper(struct list *s)`, qui passe en majuscule tous les caractères de la liste qui étaient en minuscule. Pour vous aider, un programme manipulant des minuscules/majuscules est fourni en annexe B.*

cf. `list_to_upper`.

Question 14 (1.5 points) *Implémentez en C la fonction* `list_free(struct list *s)`*, qui libère toute la mémoire occupée par la liste* `s`*.*

cf. `list_free`.

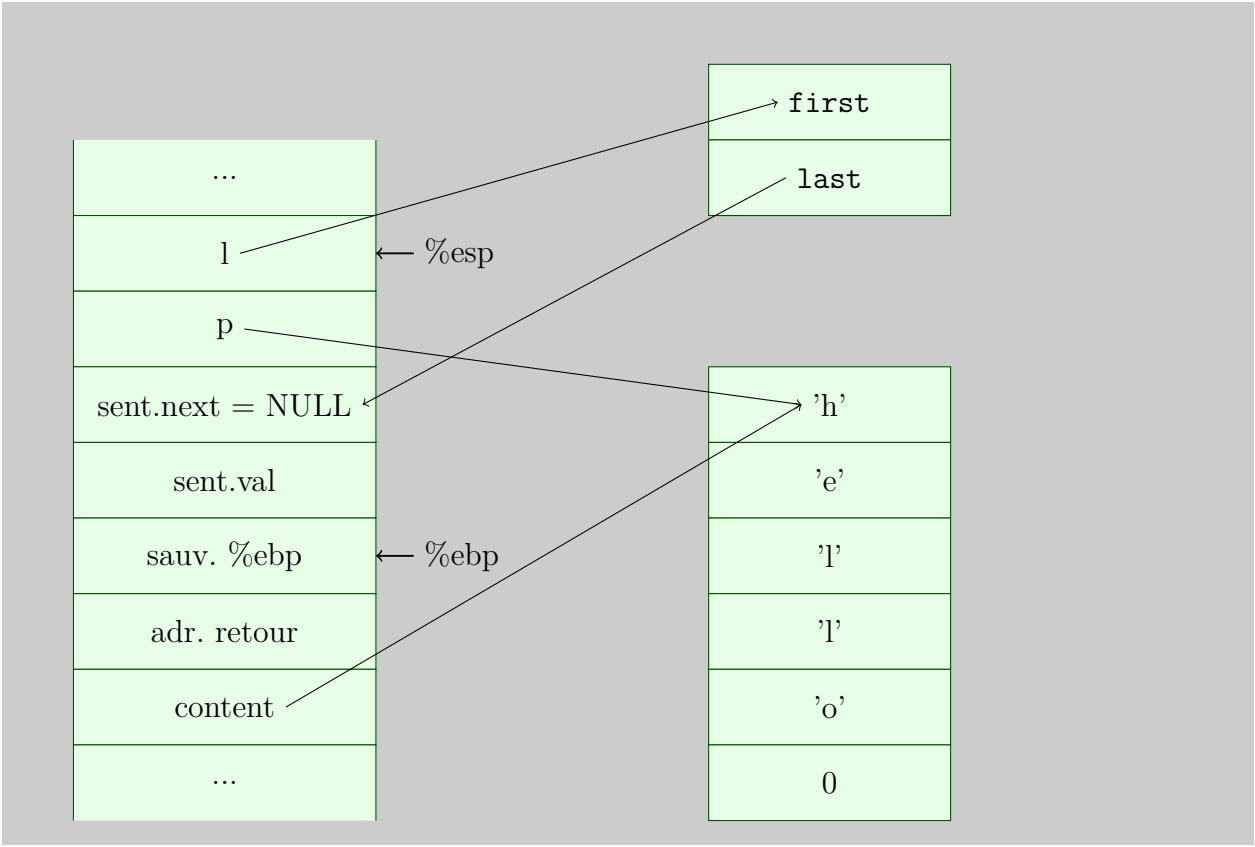
Question 15 (1.5 points) *Implémentez la fonction* `void list_append(struct list *s, char c)`*, qui ajouter le caractère* `c` *en fin de liste* `s`*.*

cf. `list_append`.

La fonction `list_new` permet de créer une liste à partir d'une chaîne de caractères C :

```
struct list *list_new(char *content) {
        struct cell *c;
        struct list *l = malloc(sizeof(struct list));
        char *p;
        struct cell sent; // 8 octets
        if (content[0] == '\0') {
                l->first = NULL;
                l->last = NULL;
                return l;
        }
        sent.next = NULL;
        l->last = &sent;
        for (p = content; *p != '\0'; ++p) {
                c = malloc(sizeof(struct cell));
                c->val = *p;
                c->next = NULL;
                l->last->next = c;
                l->last = c;
        }
        l->first = sent.next;
        return l;
}
```

Question 16 (1.5 points) *Dessinez la pile, la chaîne de caractère pointée par* `content` *(qui contient, pour l'exemple, la chaîne « hello »), et la case mémoire pointée par* `l` *à l'entrée de la boucle* `for` *(juste après l'initialisation* `p = content` *dans cette fonction. On représentera les pointeurs par des flèches.*



Question 17 (3.5 points) *Traduisez la fonction* `list_new` *en assembleur.*

cf. `list_new`.

23
23

A Spécification du paquetage de liste

```
#include <stdlib.h>

struct cell;

struct list {
        struct cell *first;
        struct cell *last;
};

struct cell {
        struct cell *next;
        char val;
};

/*
  Appelle la fonction f sur chaque caractère de la chaîne. À chaque
  appel, la fonction f reçoit le caractère, passé par adresse, en
  premier argument, et la valeur arg.

  f peut bien sûr choisir d'ignorer l'un des arguments.
*/
extern void list_iterate(struct list *s,
                         void (*f)(char *, void *),
                         void *arg);

/*
  Concatène src et dest. Le résultat est stoké dans dest, et la liste
  pointée par src est détruite (la mémoire anciennement occupée par
  src et qui n'est pas utilisée dans le résultat est libérée).
*/
extern void list_cat(struct list *dest, struct list *src);

/*
  Ajoute le caractère c à la liste l.
*/
extern void list_append(struct list *s, char c);

/*
  Renvoie une liste contenant les caractères de la chaine content.
*/
extern struct list *list_new(char *content);

/*
  Passe tous les caractères de la liste s en majuscule.
*/
extern void list_to_upper(struct list *s);

/*
  Libère la mémoire occupée par la liste s (toute la mémoire, y
  compris les cellules de la liste.
*/
extern void list_free(struct list *s);
```

B Exemple de code avec minuscules/majuscules

```
#include <stdio.h>
int main() {
        char c = 'a';
        if (c >= 'a' && c <= 'z') {
                printf("'c' est en minuscule\n");
                printf("la majuscule de 'c' est %c\n", c + 'A' - 'a');
        }
        return 0;
}
```

Ce programme donne la sortie :

```
'c' est en minuscule
la majuscule de 'c' est A
```