Cours de compilation
TD numéro 2

Jean Méhat

8 octobre 2015

Commentaires généraux

Puisque les résultats des TPs sont pris en compte dans la notation, il est nécessaire d'avoir assimilé le contenu du cours *avant* la séance de TP. Je trouve cela inquiétant d'avoir vu des étudiants étudier le chapitre 1 du support pendant le TP, visiblement pour la première fois.
Pour que je puisse tester votre code, il était nécessaire de m'envoyer les fichiers de code éventuellement en les regroupant dans une archive. Il y a plus de 30 étudiants dans le cours ; ce n'est pas envisageable pour moi d'extraire vos réponses d'un fichier de texte ou d'un PDF avec des copier-coller pour pouvoir les compiler. Pour la même raison, il était nécessaire de joindre le fichier en C qui accompagnait le fichier en assembleur.
Beaucoup d'entre vous ont utilisé du code avec des constantes pour tester vos fonctions. Ce n'est pas raisonnable car cela limite considérablement les tests effectués ; par exemple, plusieurs réponses pour la fonction qui calcule le minimum renvoyaient systématiquement le premier argument et n'avaient visiblement été testées qu'avec le minimum comme premier argument. Il vaut bien mieux utiliser les arguments de la fonction main (argc et argv) ce qui permet d'effectuer plusieurs tests sans avoir à recompiler le code, surtout si on est un peu familier avec la ligne de commande (*indication subliminale : familiariez-vous avec la ligne de commande*).

Question 1 : Écrire en assembleur une fonction `min2` qui reçoit deux arguments et renvoie le plus petit des deux.
Commentaire : à peu près toutes les réponses ont utilisé une méthode qui me semble peu naturelle *(placer les arguments dans les registres, si c'est le bon argument qui est dans %eax, effectuer un return, sinon les échanger et effectuer un return)* ; moi j'ai plutôt envie de faire *placer les arguments dans les registres, si ce n'est pas le bon qui est dans %eax, les échanger, faire le return.* Ça donne le code assembleur suivant :

```
1         .globl min2
2         .text
3 min2:
4         movl 4(%esp),%eax /* premier argument : a */
5         cmpl 8(%esp),%eax /* a - second argument b */
6         jle  ok           /* a - b <= 0 */
7         movl 8(%esp),%eax /* b est plus petit que a */
8 ok:
9         ret
```

Le code C pour le faire tourner :

```c
/* min2-d.c
  Driver pour tester la fonction min2
 */
# include <stdio.h>
# include <stdlib.h> // pour strtol

extern int min2(int, int); // defini dans min2.s

int
main(int argc, char * argv[]){
  int a, b;

  if (argc != 3){
    fprintf(stderr, "usage: %s N M\n", argv[0]);
    return 1;
  }
  a = strtol(argv[1], NULL, 0);
  b = strtol(argv[2], NULL, 0);
  printf("min2(%d, %d) = %d\n", a, b, min2(a, b));
  return 0;
}
```

Des commandes pour le tester :

```
$ gcc -Wall -m32 -g min2-d.c min2.s
$ a.out 1 2
min2(1, 2) = 1
$ a.out 2 1
min2(2, 1) = 1
```

On peut tester un peu plus sérieusement avec par exemple

```
$ for i in $(seq 1 5) ; do for j in $(seq 1 5) ; do a.out $i $j; done ; done
```

Question 2 : Traduire en assembleur la fonction `indexa` suivante, qui renvoie l'adresse du premier a dans une chaîne (ou 0 s'il n'y est pas).

```
char *
indexa(char string[]){
        int i;

        for(i = 0; string[i] != 0; i++)
                if (string[i] == 'a')
                        return & string[i];
        return 0;
}
```

ou bien :

```
char * indexa(char * p){
        for(; *p; p++)
                if (*p == 'a')
                        return p
        return 0;
}
```

Commentaire : Le code assembleur à écrire n'était pas trop difficle. Certains ont cependant commis une erreur grossière de comparer l'octet avec $'0' pour détecter la fin de chaîne. Certains ont renvoyé l'index au lieu de l'adresse. Plusieurs n'ont pas traité correctement le cas où la chaine était vide (mais les déficiences de leurs tests ne semblent pas avoir permis d'identifier le problème).

```
        .globl indexa
        .text
indexa: movl    4(%esp),%eax    /* placer l'argument (p) dans %eax */
        decl    %eax            /* p-- */
loop:
        incl    %eax            /* p++ */
        cmpb    $'a',(%eax)     /* *p == 'a' ? */
        je      fin             /*   alors on s'arrete */
        cmpb    $0,(%eax)       /* *p == 0 ? */
        jne     loop            /*   non */
        movl    $0,%eax         /*   oui : renvoyer NULL */
fin:
        ret
```

Le code de test était un peu plus délicat : on ne pouvait pas se contenter d'imprimer l'adresse renvoyée par indexa, sans s'assurer que cette adresse était correcte. Mon code utilise la fonction de bibliothèque strchr; si vous ne connaissiez pas cette fonction, il était possible d'utiliser le code de l'énoncé.

```
# include <stdio.h>
# include <string.h>

extern char * indexa(char *); // dans indexa.s

void
tester(char * string){
  char * trouve = indexa(string);
  char * correct = strchr(string, 'a');

  printf("Dans la chaine %s: ", string);
  if (correct != trouve)
    printf("ERREUR: indexa renvoie %p au lieu de %p\n", trouve, correct);
   else if (correct != NULL)
     printf("on trouve 'a' en %d\n", trouve - string);
   else
     printf("on n'a pas trouve 'a'\n");
}

int
main(int argc, char * argv[]){
  int i;

  for(i = 1; i < argc; i++)
    tester(argv[i]);
  return 0;
}
```

Un test possible (a dans toutes les positions d'une chaine de 5 caractères, en vérifiant qu'il rend bien le premier a quand il y en a plusieurs).

```
$  gcc -Wall -g -m32 indexa-d.c indexa.s
```

```
$ a.out xxx abcdea bacdea bcadea bcdaea bcdaea bcdeaa
Dans la chaine xxx: on n'a pas trouve 'a'
Dans la chaine abcdea: on trouve 'a' en 0
Dans la chaine bacdea: on trouve 'a' en 1
Dans la chaine bcadea: on trouve 'a' en 2
Dans la chaine bcdaea: on trouve 'a' en 3
Dans la chaine bcdaea: on trouve 'a' en 3
Dans la chaine bcdeaa: on trouve 'a' en 4
```

Question 3 : Écrire une fonction `rindexa` qui renvoie l'adresse du *dernier* caractère 'a' dans la chaîne.
Corrigé : Je trouve inquiétant que beaucoup d'entre vous aient répondu correctement à la question précédente mais pas à celle-ci dont le niveau de difficulté me semble équivalent. Je ne montre que le code assembleur (le code de test est à peu près le même que dans la question précédente en utilisant cette fois la fonction strrchr ; on devrait le trouver dans les fichiers de l'archive).

```
1         .globl rindexa
2         .text
3 rindexa:
4         movl    4(%esp),%ecx    /* placer l'argument (p) dans %ecx */
5         movl    $0,%eax         /* renvoyer NULL par defaut */
6 loop:
7         cmpb    $'a',(%ecx)     /* *p == 'a' ? */
8         jne     non
9         movl    %ecx,%eax       /*   si oui, garder p dans %eax */
10 non:
11        cmpb    $0,(%ecx)       /* *p == 0 ? */
12        je      fin             /*   oui : on s'arrete */
13        incl    %ecx            /* p++ */
14        jmp     loop
15 fin:
16        ret
```

Question 4 : Traduire la fonction suivante du C vers l'assembleur

```
int
fact(int n){
        int r;

        for(r = 1; n > 1; n--)
                r *= n;
        return r;
}
```

Corrigé : beaucoup de réponses correctes ; ce n'est pas très étonnant puisqu'il suffisant de retaper du code vu en cours. Il y a de nombreuses autres façons de faire ; par exemple :

```
1         .text
2 fact:
3         movl    4(%esp),%ecx
4         movl    $1,%eax
5         movl    $1,%edx
6 loop:
```

```
7         cmpl    %ecx,%edx
8         jne     encore
9         imull   %edx,%eax
10        ret
11 encore:
12        imull   %edx,%eax
13        incl    %edx
14        jmp     loop
15        .globl  fact
```

Le code C :

```
1 /* fact-d.c
2
3    Driver pour la fonction factorielle
4 */
5 # include <stdio.h>
6 # include <stdlib.h>
7
8 extern int fact(int n); // dans fact.s
9
10 int
11 main(int argc, char * argv[]){
12   int i;
13   int t;
14
15   for(i = 1; i < argc; i++){
16     t = strtol(argv[i], NULL, 0);
17     printf("fact(%d) = %d\n", t, fact(t));
18   }
19   return 0;
20 }
```

On peut tester avec :

```
$ gcc -g -m32 fact-d.c fact.s
$ a.out 1 2 3 4 5 6 7
fact(1) = 1
fact(2) = 2
fact(3) = 6
fact(4) = 24
fact(5) = 120
fact(6) = 720
fact(7) = 5040
```

Si on appelle la fonction avec un argument négatif, la boucle balaie presque tous les entiers (en passant par 0) en quelques secondes :

```
$ a.out -1
fact(-1) = 0
```

Question 5 : Traduire la fonction suivante du C vers l'assembleur

```
int
fib(int n){
        if (n < 2)
                return n;
```

```
        return fib(n - 1) + fib(n - 2);
}
```

Corrigé : Une réponse possible :

```
1         .text
2         .globl fib
3 fib:                              /* pile: ... n @retour */
4         movl    4(%esp),%eax      /* copier n dans eax */
5         cmpl    $2,%eax           /* n-2 */
6         jge     calcule           /* si n < 2 */
7         ret                       /*   renvoyer n */
8 calcule:
9         decl    %eax              /* eax = n-1 */
10
11        pushl   %eax              /* calculer fib(n-1) */
12        call    fib
13        add     $4,%esp
14        /* eax vaut fib(n-1) */
15
16        pushl   %eax              /* pile : ... n @retour fib(n-1) */
17        movl    8(%esp),%eax      /* copier n dans eax */
18        subl    $2,%eax           /* eax = n-2 */
19
20        pushl   %eax              /* calculer fib(n-2) */
21        call    fib
22        add     $4,%esp
23        /* eax vaut fib(n-2), pile : ... n @retour fib(n-1) */
24
25        popl    %ecx              /* pile: ... n @retour */
26        addl    %ecx,%eax
27        ret
```

Le fichier driver en C est presque le même que celui pour tester la fonction factorielle; vous pouvez le trouver dans l'archive. On peut tester par exemple avec

```
$ gcc -g -m32 fib.s fib-d.c
$ a.out $(seq 0 12)
fib(0) = 0
fib(1) = 1
fib(2) = 1
fib(3) = 2
fib(4) = 3
fib(5) = 5
fib(6) = 8
fib(7) = 13
fib(8) = 21
fib(9) = 34
fib(10) = 55
fib(11) = 89
fib(12) = 144
```

Commentaire : Le point crucial ici est de sauver la valeur de fib(n-1) dans la pile pendant le calcul de fib(n-2). J'ai compté comme fausses les réponses qui calculaient la valeur de la fonction de Fibonacci sans utiliser la récursion.

Question 6 : Traduire l'assembleur suivant en C

```
        .text
        .globl  heron
heron:
        pushl   %ebx
        movl    8(%esp),%eax
        movl    12(%esp),%ebx
        movl    16(%esp),%ecx
        movl    %eax,%edx
        addl    %ebx,%eax
        addl    %ecx,%eax
        sarl    $1,%eax
        subl    %eax,%ebx
        subl    %eax,%ecx
        subl    %eax,%edx
        imull   %ebx,%eax
        imull   %ecx,%eax
        imull   %edx,%eax
        negl    %eax
        popl    %ebx
        ret
```

Commentaire : pas une seule réponse correcte à cette question ! Je la met de coté pour une autre occasion.

Bonus

Les questions bonus ne sont bien sur à traiter qu'après avoir répondu aux questions principales ; j'ai eu quelques réponses peut-être correctes pour la fonction qui calcule le médian mais je ne les ai pas prises en compte parce qu'il manquait des réponses aux questions principales.