[T06] Esercitazione 6

Istruzioni per l'esercitazione:

- Aprite il form di consegna in un browser e loggatevi con le vostre credenziali `uniroma1`.
- Scaricate e decomprimete sulla scrivania il codice dell'esercitazione. Vi sarà una sotto-directory separata per ciascun esercizio di programmazione. Non modificate in alcun modo i programmi di test `*_main.c`.
- Rinominare la directory chiamandola `cognome.nome`. Sulle postazioni del laboratorio sarà `/home/studente/Desktop/cognome.nome/`.
- È possibile consultare appunti/libri e il materiale didattico online.
- Rispondete alle domande online sul modulo di consegna.
- **Finiti gli esercizi**, e non più tardi della fine della lezione:
  - **zippate la directory di lavoro** in `cognome.nome.zip` (`zip -r cognome.nome.zip cognome.nome/`).
- **Per consegnare**:
  - inserite nel form di consegna come autovalutazione il punteggio di ciascuno dei test forniti (inserite zero se l'esercizio non è stato svolto, non compila, o dà errore di esecuzione).
  - fate **upload** del file `cognome.nome.zip`.
  - **importante**: verificate di aver ricevuto mail di conferma per la sottomissione del form
- Se siete in laboratorio, prima di uscire:
  - **importante**: fate logout dal vostro account Google!
  - eliminate dal desktop la directory creata (`rm -rf cognome.nome`).
  - rimettete a posto eventuali **sedie** prese all'ingresso dell'aula!

Per maggiori informazioni fate riferimento al regolamento delle esercitazioni.

Esercizio 1 (Inserimento in testa in una lista collegata)

Tradurre nel file `E1/e1.s` la seguente funzione C contenuta in `E1/e1.c` che aggiunge un elemento in testa una lista collegata:

```c
#include <stdlib.h>
#include "e1.h"

int list_add_first(node_t **l, short elem) {
    node_t *p = *l;
    node_t *n = malloc(sizeof(node_t));
    if (n == NULL) return -1;           // allocation error
    n->elem = elem;
    n->next = p;
    *l = n;
    return 0;
}
```

NOTA IMPORTANTE: L'invocazione delle funzione `malloc` è una normale chiamata a funzione (istruzione `call`) in cui occorre passare un argomento attraverso la stack. Non è necessario conoscere nessun dettaglio implementativo della `malloc`.

Usare il main di prova nella directory di lavoro `E1` compilando con `gcc -m32 e1_main.c e1.s -o e1`.

Esercizio 2 (Uguaglianza di liste)

Tradurre nel file `E2/e2.s` la seguente funzione C contenuta in `E2/e2.c` che, date due liste collegate, restituisce 1 se sono uguali e 0 altrimenti. Riferirsi alla dichiarazione della struttura `node_t` dell'Esercizio 1.

```c
#include <stdlib.h>
#include "e2.h"

int list_equal(const node_t *l1, const node_t *l2) {
    while (l1!=NULL && l2!=NULL) {
        if (l1->elem != l2->elem) return 0;
        l1 = l1->next;
        l2 = l2->next;
    }
    return l1==NULL && l2==NULL;
}
```

Suggerimento: usare le istruzioni SETcc.

Usare il main di prova nella directory di lavoro E2 compilando con `gcc -m32 e2_main.c e2.s -o e2`.

Esercizio 3 (strpbrk alla vaccinara)

Si richiede di scrivere un'implementazione della funzione standard C strpbrk:

```c
char *my_strpbrk(const char *s1, const char *s2);
```

La funzione deve restituire il puntatore alla prima occorrenza in `s1` di un qualsiasi carattere presente nella stringa `s2` oppure `NULL` se alcun carattere di s2 appare in `s1` prima che `s1` stessa termini.

Esempio: Se `s1 = "Once again, this is a test"` e `s2="ftir"`, `my_strpbrk` deve restituire il puntatore alla prima occorrenza in `s1` di un qualsiasi carattere nella stringa `s2`, cioè il puntatore alla `i` di "again". La chiamata `my_strpbrk("Once again, this is a test",":#F")` deve invece restituire `NULL`. Ovviamente non è accettabile chiamare direttamente la `strpbrk` predefinita della `libc` (non impareresti nulla).

Usare il main di prova nella directory di lavoro E3 compilando con `gcc -m32 e3_main.c e3.c -o e3`.

Esercizio 4 (Minimo Comune Multiplo)

Tradurre nel file `E4/e4.s` la seguente funzione C contenuta in `E4/e4.c` che, dati due interi, ne calcola il minimi comune multiplo. Usare il file `E4/e4_eq.c` per sviluppare la versione C equivalente.

```c
#include "e4.h"

int lcm(int x, int y) {
    int greater = y;
    if (x > y)
        greater = x;
    while (1) {
        if ((greater % x == 0) && (greater % y == 0))
            return greater;
        greater++;
    }
}
```

Suggerimento: usare le istruzioni CMOVcc e SETcc.

Usare il main di prova nella directory di lavoro E4 compilando con `gcc -m32 e4_main.c e4.s -o e4`.

Domande

1. Assumendo che il registro %ecx contenga il valore 0xF0F0F0F0, dopo aver eseguito l'istruzione "shll $4, %ecx" quale delle seguenti affermazioni risulta vera:
    - A. Il bit più significativo di %ecx è 1
    - B. Il bit meno significativo di %ecx è 1
    - C. %cl contiene il valore 0
    - D. %ch contiene il valore 0
    - E. Se si esegue l'istruzione "sarl $4, %ecx" si riporta il valore di %ecx a 0xF0F0F0F0

2. Assumendo che il registro %ecx contenga il valore 0x00FF00FF, quale delle seguenti affermazioni risulta vera:

    - A. Se eseguiamo "sarl $3, %ecx" otteniamo in %ecx un numero più grande rispetto a prima
    - B. Se eseguiamo "sarl $8, %ecx" oppure "shrl $8, %ecx" otteniamo lo stesso valore in %ecx
    - C. Se eseguiamo "sarl $31, %ecx" otteniamo in %ecx un numero diverso da 0
    - D. Se eseguiamo "sarb $1, %cl" il valore contenuto in %ecx cambia
    - E. Nessuna delle precedenti

3. Quale delle seguenti istruzioni *NON* calcola lo stesso risultato di "shll $1, %eax":

    - A. imul $2, %eax
    - B. addl %eax, %eax
    - C. leal (%eax, %eax), %eax
    - D. leal 2(%eax), %eax
    - E. sall $1, %eax

4. Siano %eax=33 (decimale), %edx=0 (decimale) e %ecx=10 (decimale), quale delle seguenti affermazioni risulta vera:

    - A. Per calcolare %eax diviso 10 posso eseguire "idivl %eax" oppure "sarl $10, %eax"
    - B. "idivl $3" è un'istruzione valida e calcola "%edx:%eax" diviso 3
    - C. Non posso usare "sarl" per calcolare %eax diviso 2, perché %eax contiene un valore che non è una potenza di 2
    - D. "idivl %ecx" scrive il valore 3 nel registro %eax e nel registro %edx
    - E. Nessuna delle precedenti
5. Data la struct C "struct s { char x, short * y; int z;}", quanti byte (sizeof) sono necessari per memorizzarla in stack?
    - A. 7
    - B. 8
    - C. 10
    - D. 12
    - E. 14
    - F. 16
6. Data la struct C "struct s { char x, short y; char z}", quanti byte di padding sono presenti nella struct?
    - A. 0
    - B. 1
    - C. 2
    - D. 3
    - E. 4
7. Data la struct C "struct s { char x, int y; char * z; short q; char p; }", qual è l'offset del campo p?
    - A. 8
    - B. 10

- C. 12
- D. 14
- E. 16

8. Considerata la costante -72, quale tra le seguenti affermazioni risulta essere vera?
   - A. La sua rappresentazione binaria (8 bit) è 0100 1000
   - B. La sua rappresentazione binaria (16 bit) è 1111 1111 1011 0110
   - C. La sua rappresentazione binaria (16 bit) è 0000 0000 1011 0110
   - D. La sua rappresentazione binaria (16 bit) è 1111 1111 0100 1000
   - E. Se gli viene sommato 1 la sua rappresentazione binaria (16 bit) è 1111 1111 1011 1000
   - F. Nessuna delle precedenti

Soluzioni

Esercizio 1 (Inserimento in testa in una lista collegata)

es1_eq.c

```c
#include <stdlib.h>
#include "e1.h"

int list_add_first(node_t **l, short elem) {
    node_t **si = l;
    node_t *b = *si;
    node_t *a = malloc(sizeof(node_t));
    if (a != NULL) goto F;
    return -1;           // allocation error
F:;
    short d = elem;
    (*a).elem = d;
    (*a).next = b;
    *si = a;
    return 0;
}
```

es1.s

```
.globl list_add_first

# typedef struct node_t {    // base
#     short elem;            // offset: 0 |xx..|   (base)
#     struct node_t *next;   // offset: 4 |xxxx|  4(base)
# } node_t;                  // sizeof: 8

list_add_first: # int list_add_first(node_t **l, short elem) {
    pushl %ebx
    pushl %esi
    subl $4, %esp

    movl 16(%esp), %esi     # node_t **si = l;
    movl (%esi), %ebx       # node_t *b = *si;
    movl $8, (%esp)
    call malloc             # node_t *a = malloc(sizeof(node_t));
    testl %eax, %eax        # if (a != NULL)
    jnz F                   # goto F;
    movl $1, %eax           # return -1;           // allocation error
    jmp E
F:
    movl 20(%esp), %edx     # short d = elem;
    movw %dx, (%eax)        # (*a).elem = d;
```

```
    movl %ebx, 4(%eax)      # (*a).next = b;
    movl %eax, (%esi)       # *si = a;
    movl $0, %eax
E:
    addl $4, %esp
    popl %esi
    popl %ebx
    ret                             # return 0;
```

Esercizio 2 (Uguaglianza di liste)

list_equal.c

```c
#include <stdlib.h>
#include "e2.h"

int list_equal(const node_t *l1, const node_t *l2) {
    const node_t *a = l1;
    const node_t *c = l2;
L:
    if (a==0) goto E;
    if (c==0) goto E;
    short d = (*c).elem;
    if ((*a).elem == d) goto F;
    return 0;
F:
    a = (*a).next;
    c = (*c).next;
    goto L;
E:;
    char al = a == 0;
    char ah = c == 0;
    al = al & ah;
    return (int) al;
}
```

e2.s

```
.globl list_equal
list_equal: # int list_equal(const node_t *l1, const node_t *l2)
    movl 4(%esp), %eax      # node_t *a = l1;
    movl 8(%esp), %ecx      # node_t *c = l2;
L:
    testl %eax, %eax        # if (a==0)
    jz E                    # goto E;
    testl %ecx, %ecx        # if (c==0)
    jz E                    # goto E;
    movw (%ecx), %dx        # short d = (*c).elem;
    cmpw %dx, (%eax)        # if ((*a).elem == d)
    je F                    # goto F;
E1:
    movl $0, %eax
    ret                     # return 0;
F:
    movl 4(%eax), %eax      # a = (*a).next;
    movl 4(%ecx), %ecx      # c = (*c).next;
    jmp L                   # goto L;
E:
    testl %eax, %eax        # char al = a == 0;
    setz %al
```

```
    testl %ecx, %ecx        # char ah = c == 0;
    setz %ah
    andb %ah, %al           # al = al & ah;
    movsbl %al, %eax
    ret                     # return (int) al;
```

Esercizio 3 (strpbrk alla vaccinara)

my_strpbrk

```c
#include <stdio.h>
#include "e3.h"
#include <stdlib.h>

char *my_strpbrk(const char *s1, const char *s2) {
    const char* p;
    while (*s1) {
        for (p=s2; *p; p++)
            if (*s1 == *p) return (char*) s1;
        s1++;
    }
    return NULL;
}
```

Esercizio 4 (Minimo Comune Multiplo)

e4_eq.c

```c
#include "e4.h"

int lcm(int x, int y) {
    int si = x;
    int di = y;
    int c = di;
    if (si <= di)
        c = si;
    int a;
L:;
    a = c; // setta d in modo opportuno!
    int d = a % si;
    char bl = d == 0;
    a = c; // setta d in modo opportuno!
    d = a % di;
    char bh = d == 0;
    bl = bl & bh;
    if (bl == 0) goto F;
    a = c;
    return a;
F:
    c++;
    goto L;
}
```

e4.s

```
.globl lcm
lcm: # int lcm(int x, int y)
```

```
    pushl %esi
    pushl %edi
    pushl %ebx

    movl 16(%esp), %esi     # int si = x;
    movl 20(%esp), %edi     # int di = y;
    movl %edi, %ecx         # int c = di;
    cmpl %edi, %esi         # if (si > di)
    cmovg %esi, %ecx        # c = si;
L:
    movl %ecx, %eax         # a = c; // setta d in modo opportuno!
    movl %eax, %edx
    sarl $31, %edx
    idivl %esi              # int d = a % si;
    testl %edx, %edx        # char bl = d == 0;
    setzb %bl
    movl %ecx, %eax         # a = c; // setta d in modo opportuno!
    movl %eax, %edx
    sarl $31, %edx
    idiv %edi               # d = a % di;
    testl %edx, %edx        # char bh = d == 0;
    setzb %bh
    andb %bh, %bl
    jz F
    movl %ecx, %eax         # a = c;
    popl %ebx
    popl %edi
    popl %esi
    ret                 # return a;
F:
    incl %ecx           # c++;
    jmp L               # goto L;
```

Domande

Risposte corrette ai quiz:

1. C - %cl contiene il valore 0
2. B - Se eseguiamo “sarl $8, %ecx” oppure “shrl $8, %ecx” otteniamo lo stesso valore in %ecx
3. D - leal 2(%eax), %eax
4. D - “idivl %ecx” scrive il valore 3 nel registro %eax e nel registro %edx
5. D - 12
6. C - 2
7. D - 14
8. F - Nessuna delle precedenti