[T08] Esercitazione 8

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 (Ordinamento e ricerca su array di strutture)

Si vuole scrivere nel file `E1-find-person/e1.c` due funzioni che lavorano su array di strutture che rappresentano persone come definito di seguito:

```
typedef struct person_t person_t;

struct person_t {
    char *name;
    int age;
};
```

La prima funzione ha il seguente prototipo:

```
void sort_people(person_t p[], size_t n);
```

e deve ordinare per nome le persone nell'array `p` di dimensione `n`. La seconda funzione ha il seguente prototipo:

```
person_t *find_person(char *key, person_t sorted[], size_t n);
```

e, assumendo che `sorted` sia un array di `n` persone ordinate per nome e `key` è un nome da cercare, restituisce il puntatore a una persona nell'array che ha quel nome, oppure `NULL` se nessuna persona ha quel nome. Se vi sono più persone con il nome cercato, restituisce una qualunque.

NOTA: Si ricorda che per operazioni di ordinamento e ricerca è possibile utilizzare le funzioni `qsort` e `bsearch`.

Esercizio 2 (Parsing)

Si consideri un software per la gestione delle prenotazioni in un ristorante. L'elenco dei prenotati è salvato in un buffer, composto dalla concatenazione di *record* di 37 byte ciascuno. Ogni record è suddiviso nei seguenti campi:

- cognome associato alla prenotazione => stringa da 30 byte
- numero di posti => stringa da 2 byte
- orario della prenotazione => stringa da 5 byte

Tutti questi campi sono rappresentati da stringhe **senza** terminatore. Non esiste separatore tra i campi e tra i record. I bytes in eccesso sono costituiti da padding rappresentato con il carattere _. Il campo dell'orario ha il formato `hh:mm`.

Ad esempio, il record:

```
Rossi_____________________________0419:50
```

descrive una prenotazione a nome Rossi, per 4 posti, alle ore 19:50.

Si scriva in `e2.c` una funzione `getBookingsAfterTime` con il seguente prototipo:

```
void getBookingsAfterTime(struct booking ** list, const char * data, int size, const ch
```

che, dato in ingresso un buffer `data` (di dimensione `size` byte) con tutti i record dei tavoli prenotati e un orario `time` espresso in formato `hh:mm`, restituisce in `list` la lista di tutti i tavoli prenotati per un orario uguale o successivo a `time`. Si noti che `list` deve essere una lista collegata costituita da elementi rappresentati dalla `struct booking` (definita nel file `e2.h`):

```
struct booking {
   char * surname;
   int places;
   char * time;
   struct booking * next;
};
```

L'ordine delle prenotazioni in `list` deve essere coerente con l'ordine che le stesse hanno nel buffer `data`.

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

NOTA: Si ricorda che per convertire una stringa in un numero intero è possibile utilizzare la funzione `atoi`.

Esercizio 3 (Palestra IA32)

Tradurre in IA32 in un file `e3.s` la seguente funzione, tenendo presente che i codici ASCII dei caratteri '0' e '9' sono 48 e 57, rispettivamente.

`e3.c`

```
int count_digits(const char *s) {
    int cnt = 0;
    while (*s) {
        if (*s >= '0' && *s <= '9') cnt++;
        s++;
    }
```

```
    return cnt;
}
```

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

Domande

Rispondi alle seguenti domande, tenendo conto che una risposta corretta vale 1 punti, mentre una risposta errata vale 0 punti.

Domanda 1. Siano %eax=20 (decimale), %edx=0 (decimale) e %ecx=8 (decimale). Con riguardo alle due istruzioni “idivl %ecx” e “sarl $3, %eax”, quale delle seguenti affermazioni risulta vera:

- A. Le due istruzioni scrivono lo stesso valore in %eax, ma “idivl” è più efficiente di “sarl”
- B. Le due istruzioni scrivono lo stesso valore in %eax, ma “sarl” è più efficiente di “idivl”
- C. Le due istruzioni scrivono valori diversi in %eax
- D. In questo caso la “idivl” non modifica il valore del registro %edx
- E. Nessuna delle precedenti

Domanda 2. Assumendo %eax=0xFF000000, %ecx=1 (decimale) e %edx=10 (decimale), dopo aver eseguito l’istruzione “testl %eax, %eax” quale delle seguenti affermazioni risulta vera:

- A. Se eseguiamo “cmovnzl %edx, %ecx” viene scritto il valore 10 nel registro %ecx
- B. Se eseguiamo “cmovzl %edx, %ecx” viene scritto il valore 0 nel registro %ecx
- C. L’istruzione “cmovnzw %dx, %cx” non modifica il valore del registro %ecx
- D. L’istruzione “cmovel %edx, %ecx” è equivalente all’istruzione “cmovnzl $10, %ecx”
- E. Nessuna delle precedenti

Domanda 3. Assumendo %eax=10 (decimale), %ecx=7 (decimale) e %edx=2 (decimale), quale delle seguenti affermazioni risulta vera:

- A. Se eseguiamo “cmpl %ecx, %eax” e “cmovlel %edx, %eax” viene scritto il valore 2 nel registro %eax
- B. Se eseguiamo “cmpl %ecx, %eax” e “cmovbl %edx, %eax” viene scritto il valore 2 nel registro %eax
- C. Se eseguiamo “cmpb $0, %al” e “cmovel %edx, %eax” viene scritto il valore 2 nel registro %eax
- D. Se eseguiamo “subl %ecx, %eax” e “cmovgl %edx, %eax” viene scritto il valore 2 nel registro %eax
- E. Nessuna delle precedenti

Domanda 4. Quale delle seguenti è un’istruzione valida:

- A. cmovzb %al, %cl
- B. cmovzl $3, %eax
- B. cmovgew %ax, (%ecx)
- B. cmpl %ecx, $10
- B. Nessuna delle precedenti

Soluzioni

Esercizio 1 (Ordinamento e ricerca su array di strutture)

`e1.c`

```
#include "../../T08/E1-find-person/e1.h"
#include <string.h>
#include <stdlib.h>
```

```c
int cmp(const void *ap, const void *bp){
        person_t a = *(person_t*)ap;
        person_t b = *(person_t*)bp;
        return strcmp(a.name, b.name);
}

void sort_people(person_t p[], size_t nel) {
        qsort(p, nel, sizeof(person_t), cmp);
}

person_t *find_person(char *key, person_t sorted[], size_t nel) {
        person_t key_person;
        key_person.name = key;
        return bsearch(&key_person, sorted, nel, sizeof(person_t), cmp);
}
```

Esercizio 2 (Parsing)

e2.c

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

#include "../../T08/E2/e2.h"

int get_timestamp(const char * A){
    char Ah[3] = {A[0], A[1], '\0'};
    char Am[3] = {A[3], A[4], '\0'};
    return atoi(Ah) * 60 + atoi(Am);
}

void getBookingsAfterTime(struct booking ** list, const char * data, int size, const ch
    *list = NULL;
    struct booking * last = NULL;
    struct booking * elem;

    int start = get_timestamp(time);

    // Leggo dal buffer fino a quando riesco a trovare ulteriori prenotazioni
    int pos = 0;
    while(pos < size) {

        // Leggo la prossima prenotazione dal buffer
        const char* buf = &data[pos];
        pos += 37;

        // Estraggo l'ora di prenotazione
        int i=32;
        char time_tmp [6];
        while (i<37) {
            time_tmp[i-32] = buf[i];
            i++;
        }
        time_tmp[i-32]='\0';

        // Se l'ora di prenotazione è successiva a time
        if (get_timestamp(time_tmp) >= start) {
            // Alloco memoria per un nuovo elemento della lista
            elem = malloc(sizeof(struct booking));
```

```
            // Aggiungo all'elemento l'orario di prenotazione
            elem->time = malloc((strlen(time_tmp) + 1) * sizeof(char));
            strcpy(elem->time, time_tmp);

            // Estraggo ed aggiungo all'elemento il numero di posti prenotati
            char seats_tmp [3];
            seats_tmp[0] = buf[30];
            seats_tmp[1] = buf[31];
            seats_tmp[2] = '\0';
            elem->places = atoi(seats_tmp);

            // Estraggo ed aggiungo all'elemento il cognome di chi ha prenotato
            i=0;
            char surname_tmp [31];
            while (i<30 && buf[i]!='_') {
                surname_tmp[i] = buf[i];
                i++;
            }
            surname_tmp[i]='\0';
            elem->surname = malloc((strlen(surname_tmp) + 1) * sizeof(char));
            strcpy(elem->surname, surname_tmp);

            // Aggiungo l'elemento in coda alla lista
            elem->next=NULL;
            if (*list == NULL)
                *list = elem;
            else
                last->next = elem;
            last = elem;
        }
    }

    return;
}
```

Esercizio 3 (Palestra IA32)

e3_eq.c

```
int count_digits(const char *s) {
    int a = 0;
    const char* c = s;
L:
    char dl = *c;
    if (dl == 0) goto E;
    if (dl < '0') goto F;
    if (dl > '9') goto F;
    a += 1;
F:
    c++;
    goto L;
E:
    return a;
}
```

e3.s

```
.globl count_digits
count_digits: # int count_digits(const char *s) {
    xorl %eax, %eax     #     int a = 0;
```

```
    movl 4(%esp), %ecx  #     const char* c = s;
L:
    movb (%ecx), %dl    #     char dl = *c;
    testb %dl, %dl      #     if (dl == 0) goto E;
    je E
    cmpb $48, %dl       #     if (dl < '0') goto F;
    jl F
    cmpb $57, %dl       #     if (dl > '9') goto F;
    jg F
    incl %eax           #     a += 1;
F:
    incl %ecx           #     c++;
    jmp L               #     goto L;
E:
    ret                 #     return a;
```

Domande

1. B - Le due istruzioni scrivono lo stesso valore in %eax, ma “sarl” è più efficiente di “idivl”
2. A - Se eseguiamo “cmovnzl %edx, %ecx” viene scritto il valore 10 nel registro %ecx
3. D - Se eseguiamo “subl %ecx, %eax” e “cmovgl %edx, %eax” viene scritto il valore 2 nel registro %eax
4. E - Nessuna delle precedenti