[T09] Esercitazione 9

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 (Parser CSV)

Si vuole scrivere nel file `E1/e1.c` una funzione con il seguente prototipo:

```
int parseCSV(const char* file, person_t** out, int minYear);
```

che processa un file CSV e genera in output (sfruttando `*out`) una lista collegata di persone che sono nate a partire dall'anno `minYear`. Il valore di ritorno della funzione deve essere `0` in caso di successo e `-1` in caso di errore. Ogni riga del file CSV segue il seguente formato:

```
nome,cognome,dataDiNascita,haIlDottorato
```

dove:

- `nome` è una stringa che rappresenta il nome di una persona
- `cognome` è una stringa che rappresenta il cognome di una persona
- `dataDiNascita` è una stringa che rappresenta la data di nascita di una persona nel formato `DD-MM-YYYY`
- `haIlDottorato` è una stringa "`yes`" o "`no`" che indica se una persona ha ottenuto il dottorato (PhD)

La lista collegata generate dalla funzione deve avere nodi con la seguente struttura:

```
typedef struct person_t person_t;

struct person_t {
   char* name;
```

```
    char* surname;
    int year;       // anno di nascita
    int has_phd;    // 1 se ha il dottorato, altrimenti 0
    person_t* next;
};
```

L'ordine delle persone nella lista di output deve essere coerente con l'ordine delle righe nel file CSV.

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

NOTA #1: se il file non esiste la funzione deve tornare `-1` ed una lista vuota.

NOTA #2: se il file esiste ma è vuoto deve tornare `0` ed una lista vuota.

NOTA #3: se la lista in output è vuota, `*out` deve essere inizializzato a `NULL`.

NOTA #4: si può assumere che una riga del CSV non sia più lunga di 64 caratteri.

Esercizio 2 (Palestra IA32)

Il Cyclic Redundancy Check (CRC) è un algoritmo spesso utilizzato per individuare errori casuali durante la trasmissione di dati. In questo esercizio, viene richiesto di tradurre in assembly un'implementazione semplificata del CRC32. Nella directory E2, si traduca in assembly IA32 la seguente funzione C:

```c
#include "e2.h"

int crc32b(char *bytes, int n) {
    int magic = ~0; // operazione: not
    int crc = magic;
    while (n--) {
        int value;
        int byte = *bytes++; // attenzione!
        int index = crc ^ byte;
        get_constant(&value, index & 0xFF);
        crc = value ^ (crc >> 8);
    }
    return crc ^ magic;
}
```

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

Domande

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

Domanda 1. Si consideri un programma che usa la funzione matematica `sin`. Quale delle seguenti righe di comando potrebbe essere utilizzata?

- A. `gcc main.c -math -o mymath`
- B. `gcc main.c -o mymath`
- C. `gcc main.c -o mymath -lm`
- D. nessuna delle precedenti

Domanda 2. Si consideri una funzione comparatore `int compar(const void *a, const void *b)`. Quale delle seguenti affermazioni è falsa?

- A. se l'elemento puntato da a è minore di quello puntato da b la funzione restituisce un valore negativo
- B. se l'elemento puntato da a è minore o uguale di quello puntato da b la funzione restituisce un valore negativo
- C. se l'elemento puntato da a è maggiore di quello puntato da b la funzione restituisce un valore positivo
- A. se l'elemento puntato da a è uguale a quello puntato da b la funzione restituisce un valore zero

Domanda 3. Si consideri la seguente istruzione `fseek(f,0,SEEK_CUR)`. Quale delle seguenti affermazioni è vera?

- A. l'istruzione sposta la posizione corrente del file `f` all'inizio del file
- B. l'istruzione sposta la posizione corrente del file `f` alla fine del file
- C. l'istruzione non sposta la posizione corrente del file `f`
- D. l'istruzione non è valida

Domanda 4. Si consideri la seguente istruzione `sscanf(s,"%d %d %d", &a, &b, &c)`. Quale delle seguenti affermazioni è vera?

- A. l'istruzione restituisce il numero di byte letti dalla stringa `s`
- B. l'istruzione restituisce zero se riesce a leggere tutti gli argomenti `&a, &b, &c`
- C. l'istruzione restituisce il numero di argomenti letti
- D. l'istruzione restituisce il numero di argomenti letti che assumono un valore positivo

Soluzioni

Esercizio 1 (Parser CSV)

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

#include "../../T09/E1/e1.h"

char* my_strdup(const char* s) {
    char* r = malloc(strlen(s) + 1);
    r = strcpy(r, s);
    return r;
    // oppure...
    return strdup(s);
}

int parseCSV(const char* file, person_t** out, int minYear) {
    *out = NULL;

    // apro il file
    FILE* fp = fopen(file, "r");
    if (fp == NULL) return -1;

    // leggo dal file una riga per iterazione
    person_t* last = NULL;
    while (1) {

        char line[64];
        char* res = fgets(line, sizeof(line), fp);
        if (res == NULL) break;

        // printf("Line: %s\n", line);

        char name[64], surname[64], date[11], phd[5];
```

```
        int i = 0;
        for(i = 0; i < 4; i++) {
            char* token = strtok(res, ",");
            if (token == NULL) {
                fclose(fp);
                return -1;
            }
            // rimuovo il newline character
            int len = strlen(token);
            if (token[len - 1] == '\n')
                token[len - 1] = '\0';

            // printf("token (len=%d): %s\n", len, token);

            switch(i) {
                case 0: { strncpy(name, token, sizeof(name)); break; }
                case 1: { strncpy(surname, token, sizeof(surname)); break; }
                case 2: { strncpy(date, token, sizeof(date)); break; }
                case 3: { strncpy(phd, token, sizeof(phd)); break; }
                default: { fclose(fp); return -1; }
            }
            res = NULL;
        }

        int year = atoi(date+6);
        if (year >= minYear) {
            // printf("Creating node for %s %s %d %s\n", name, surname, year, phd);

            person_t* current = malloc(sizeof(person_t));
            if (current == NULL) { fclose(fp); return -1; }

            // le stringhe sono locali alla funzione
            // quindi occorre allocarle dinamicamente
            current->name = my_strdup(name);
            current->surname = my_strdup(surname);
            current->year = year;
            current->has_phd = strcmp(phd, "yes") == 0 ? 1 : 0;
            current->next = NULL;

            if (last == NULL)
                *out = current;
            else
                last->next = current;
            last = current;
        }
    }

    // chiudo il file
    fclose(fp);

    return 0;
}
```

Esercizio 2 (Palestra IA32)

```
#include "../../T09/E2/e2.h"

int crc32b(char *bytes, int n) {
    int magic = 0;
    magic = ~magic;
    int crc = magic;
```

```
L1:
    if (n <= 0) goto E1;
    int value;
    int byte = *bytes++; // attenzione!
    int index = crc ^ byte;
    index = index & 0xFF;
    get_constant(&value, index);
    crc = crc >> 8;
    crc = value ^ crc;
    n--;
    goto L1;
E1:
    crc = crc ^ magic;
    return crc;
}
```

```
.globl crc32b

# void get_constant(int* value, int index);

crc32b: # int crc32b(char *bytes, int n) {
    pushl %esi
    pushl %edi
    pushl %ebx
    pushl %ebp
    subl $12, %esp
    xorl %ebx, %ebx     # int magic = 0;
    notl %ebx           # magic = ~magic;
    movl %ebx, %ebp     # int crc = magic;
    movl 32(%esp), %esi
    movl 36(%esp), %edi
L1:
    cmpl $0, %edi       # if (n <= 0) goto E1;
    jle E1
                        # int value;
    movsbl (%esi), %edx # int byte = *bytes++; // attenzione!
    incl %esi
    xorl %ebp, %edx     # int index = crc ^ byte;
    andl $0xFF, %edx    # index = index & 0xFF;
    movl %edx, 4(%esp)
    leal 8(%esp), %edx
    movl %edx, (%esp)
    call get_constant   # get_constant(&value, index);
    sarl $8, %ebp       # crc = crc >> 8;
    xorl 8(%esp), %ebp  # crc = value ^ crc;
    decl %edi           # n--;
    jmp L1              # goto L1;
E1:
    xorl %ebx, %ebp     # crc = crc ^ magic;
    movl %ebp, %eax
    addl $12, %esp
    popl %ebp
    popl %ebx
    popl %edi
    popl %esi
    ret                 # return crc;
```

Domande

1. C. gcc main.c -o mymath -lm
2. B. se l'elemento puntato da a è minore o uguale di quello puntato da b la funzione restituisce un valore negativo

3. C. l’istruzione non sposta la posizione corrente del file `f`
4. C. l’istruzione restituisce il numero di argomenti letti