[T01] Esercitazione

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 (Palestra C)

Scrivere una funzione C `uint2bin` con il seguente prototipo

```
void uint2bin(unsigned x, char bin[32]);
```

che, dato un intero `x` senza segno a 32 bit e un buffer `bin` di 32 caratteri, ottiene la rappresentazione binaria del numero con il bit più significativo per primo. Gli 0 e 1 nel risultato devono essere rappresentati mediante i caratteri ASCII '0' e '1'.

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

Ad esempio, invocando la funzione con `0x0F0F0F0F`, il buffer di output sarà la stringa `"00001111000011110000111100001111"`. Infatti:

- `0xFF` in esadecimale corrisponde al valore decimale `255` ed al valore binario `1111`
- `0x0` in esadecimale corrisponde al valore decimale `0` ed al valore binario `0000`

Consultare la pagina di Wikipedia sul sistema numerico binario per avere alcuni ulteriori chiarimenti.

Suggerimento #1: Ragionare sul fatto che i dati (variabili del programma) in un sistema di calcolo sono sempre e solo memorizzati come un insieme di bit. Quindi quando si definisce una variabile intera `a` e la si stampa con ad esempio:

```
printf("%d\n", a);
```

Si sta chiedendo alla funzione di `printf` di produrre in output la rappresentazione in base decimale di `a`, tuttavia `a` è e solo memorizzata come una sequenza di 32 bit. Volendo è possibile stampare la rappresentazione esacimale di `a` con:

```
printf("%x\n", a);
```

La funziona `printf` non offre una stringa di formato (come `%x` e `%d` viste sopra) per stampare la rappresentazione binaria di una variabile. Si fa notare inoltre, anche se è fuori dagli scopi di questo esercizio, che anche le stringhe non sono altro che sequence di bit: ogni carattere è una sequenza di 8 bit.

Suggerimento #2: In C, per poter conoscere il valore del bit meno significativo (primo da destra) di una variabile intera si può procedere in due modi:

1. Usando l'operatore & (AND bitwise), quindi ad esempio:

   ```
   if (a & 1) printf("Bit LSB: 1\n");
   else printf("Bit LSB: 0\n");
   ```

2. Verificando se il numero è pari o dispari, quindi ad esempio:

   ```
   if (a % 2) printf("Bit LSB: 1\n");
   else printf("Bit LSB: 0\n");
   ```

Suggerimento #3: In C, per poter "spostare" verso destra i bit di una variabile intera si può procedere in due modi:

1. Usando l'operatore >> (shift a destra), quindi ad esempio:

   ```
   a = a >> 1; // sposta tutti i bit della variabile a verso destra di una posizione
   ```

2. Svolgendo la divisione per 2, quindi ad esempio:

   ```
   a = a / 2;
   ```

Esercizio 2 (Debugging GDB)

Il frammento di codice seguente calcola l'i-esimo numero della successione di Fibonacci specificato dall'utente. In particolare, esegue una implementazione ricorsiva ed una variante iterativa dell'algoritmo di calcolo:

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

unsigned int fib(unsigned int n) {
    if (n < 2) return n;
    else return fib(n - 1) + fib(n - 2);
}

unsigned int fib_iter(unsigned int n) {
    unsigned int i = 0, j = 1, k, t;
    for (k = 1; k <= n; ++k) {
        t = i + j;
        i = j;
        j = t;
    }
    return j;
}

int main() {
```

```
    unsigned int n = 10;
    unsigned int r1 = fib(n);
    unsigned int r2 = fib_iter(n);
    printf("[%u] ric: %u iter: %u\n", n, r1, r2);
    printf("Risultato: %d/%d\n", r1 == r2, 1);

    return 0;
}
```

I due metodi `fib` e `fib_iter` sono stati presi da Wikibooks ma restituiscono risultati difformi. Risalire alla causa di questa difformità con l'ausilio di GDB, in particolare utilizzando degli opportuni breakpoint. Poi correggere il problema.

Esercizio 3 (Debugging Valgrind)

Di seguito è riportata una implementazione buggata di una lista collegata in C:

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

typedef struct node {
    char * id;
    char * value;
    struct node * next;
} node_t;

static int count = 0;

node_t* add_node(node_t* l, char* value) {
    node_t* node = malloc(sizeof(node_t));

    char id[16];
    sprintf(id, "ID_%d", count++);
    node->id = id;
    node->value = value;

    if (l != NULL)
        node->next = l;

    return node;
}

void print_list(node_t* head) {
    node_t* current = head;

    while (current != NULL) {
        printf("%s\n", current->value);
        current = current->next;
    }
}

void delete_list(node_t * head) {
    node_t* current = head;

    while (current != NULL) {
        free(current);
        free(current->value);
        current = current->next;
    }
}

int main() {
```

```
    node_t* l;

    l = add_node(NULL, "Hello");
    l = add_node(l, " ");
    l = add_node(l, "World");
    l = add_node(l, "!");

    print_list(l);
    delete_list(l);

    printf("Risultato 1/1\n");
    return 0;
}
```

Individuare gli errori attraverso:

```
valgrind --leak-check=full ./e3
```

e proporre una soluzione che utilizzi correttamente la memoria: no segmentation fault e no memory leak.

Domande

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

Domanda 1. Un circuito combinatorio è:

- A. costruito mediante porte logiche e calcola funzioni booleane
- B. alla base della costruzione delle porte logiche
- C. un circuito che consente di mantenere uno stato ed è pertanto un possibile ingrediente per costruire memorie
- D. un circuito per la temporizzazione degli eventi in un sistema di calcolo

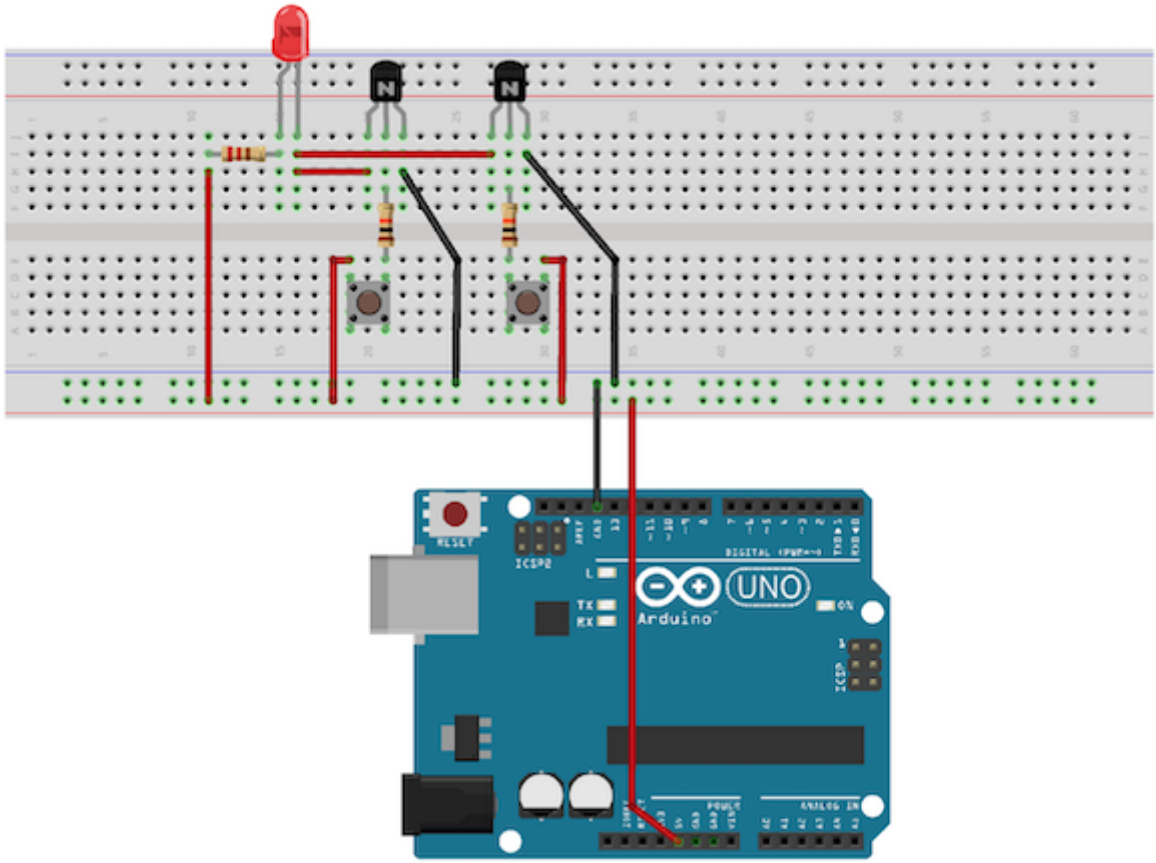
Domanda 2. Data una tensione da 12V e un LED che si illumina in presenza di una corrente nel range [50, 100] mA (oltre tale valore il LED si brucia) quale tra queste resistenze può essere inserita nel circuito per far illuminare il LED?

- A. 80 Ohm
- B. 100 Ohm
- C. 220 Ohm
- D. 450 Ohm
- E. 500 Ohm

Domanda 3. Quale tra queste affermazioni su un flip-flop SR (latch) è falsa:

- A. può essere utilizzato per memorizzare un bit di informazione
- B. la configurazione S=0 e R=1 non garantisce Q = NOT Q’
- C. la configurazione S=1 e R=1 potrebbe portare ad uno stato indefinito
- D. il flip flop non mantiene il suo stato quando non è alimentato (no corrente elettrica)

Domanda 4. Cosa calcola la seguente breadboard?



- A. NOT
- B. AND
- C. OR
- D. XOR
- E. Nessuna delle precedenti

Soluzioni

Esercizio 1 (Palestra C)

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

void uint2bin(unsigned x, char bin[32]) {
    int idx;
    for (idx = 31; idx>=0; idx--){
         bin[idx] = '0' + (x & 1);
         x = x >> 1;
    }
}
```

Esercizio 2 (Debugging GDB)

```
/*

Possibili step per analizzare questo programma con GDB:
1) Compilare il programma usando il flag -g
2) Lanciare GDB
3) Usare il comando "go" per far partire l'esecuzione, fermandosi sul main
```

```
4) mettere un "break" point alla riga stampa se il risultato e' corretto o meno; poi fa
5) Stampare usando il comando "print" le due variabili r1 e r2
6) Capire quale fra i due risultati non e' corretto: calcolare il valore di fibonacci s
7) Essendo r2 sbagliato, si vuole procedere ad analizzare fib_iter
8) Inserire "break" point sulla prima riga del corpo del for di fib_iter
9) Rifar partire esecuzione fino al comando "go"
10) Analizzare come cambia il valore di j ad ogni iterazione
11) Rendersi conto che il loop viene eseguito una volta in piu' del dovuto
12) Identificare quindi nel sorgente il motivo di questo problema
*/


#include <stdio.h>
#include <stdlib.h>

unsigned int fib(unsigned int n) {
    if (n < 2) return n;
    else return fib(n - 1) + fib(n - 2);
}

unsigned int fib_iter(unsigned int n) {
    unsigned int i = 0, j = 1, k, t;
    for (k = 1; k < n; ++k) { // conditione del loop deve essere "<" e non "<="
        t = i + j;
        i = j;
        j = t;
    }
    return j;
}

int main() {

    unsigned int n = 10;
    unsigned int r1 = fib(n);
    unsigned int r2 = fib_iter(n);
    printf("[%u] ric: %u iter: %u\n", n, r1, r2);
    printf("Risultato: %d/%d\n", r1 == r2, 1);

    return 0;
}
```

Esercizio 3 (Debugging Valgrind)

```
/*

Se eseguiamo Valgrind sopra al binario compilato con "-g", otteniamo:

1) Indicazione su uso di dati non inizializzati

==1088284== Conditional jump or move depends on uninitialised value(s)
==1088284==    at 0x1092A5: print_list (e3.c:29)
==1088284==    by 0x109375: main (e3.c:53)

Valgrind sta indicando che alle riga 29 usiamo una variabile non inizializzata.
In quella riga  viene utilizzata "current" che e' assegnata a riga 27 e riga 31.
Dopo una breve analisi, possiamo capire che il main passa correttamente un "head"
inizializzata e che quindi dobbiamo concretarci su riga 31 e non riga 27.
Quindi a riga 27 viene assegnato a "current" il campo "next" di un nodo.
Se andiamo a vedere righe 20-21, notiamo che questo campo viene assegnato solo
quando la condizione a riga 20 e' vera. Se e' falso non viene inizializzato.
Questo non e' corretto: le variabile devono sempre essere inizializzate.
Quindi aggiungiamo un assegnamento a NULL nel caso alternativo alla riga 20.
```

```
2) Accesso invalido

==1088534== Invalid read of size 8
==1088534==    at 0x1092D5: delete_list (e3.c:40)
==1088534==    by 0x109381: main (e3.c:54)
==1088534==  Address 0x4aa7168 is 8 bytes inside a block of size 24 free'd
==1088534==    at 0x484B27F: free (vg_replace_malloc.c:872)
==1088534==    by 0x1092D0: delete_list (e3.c:39)
==1088534==    by 0x109381: main (e3.c:54)
==1088534==  Block was alloc'd at
==1088534==    at 0x4848899: malloc (vg_replace_malloc.c:381)
==1088534==    by 0x1091F5: add_node (e3.c:13)
==1088534==    by 0x109365: main (e3.c:51)

Valgrind sta indicando che riga 40 fa un accesso non valido alla memeoria,
leggendo ad un indirizzo che punta ad un blocco liberato a riga 39 e allocato
a riga 13. Se facciamo attenzione possiamo notare che riga 39 effettua
una free su un puntatore che poi viene utilizzato alle righe successivo.
Questo non e' permesso: se libero un blocco non devo piu' accedervi.
Occorre quindi spostare la free dopo l'ultimo uso del puntatore
(vedere soluzione sotto).

3) Operazione free invalida

==1088534== Invalid free() / delete / delete[] / realloc()
==1088534==    at 0x484B27F: free (vg_replace_malloc.c:872)
==1088534==    by 0x1092E0: delete_list (e3.c:40)
==1088534==    by 0x109381: main (e3.c:54)
==1088534==  Address 0x10a018 is in a r-- mapped file /home/ercoppa/Desktop/code/SC/doc

Riga 40 sta liberando un blocco di memoria con free()
che non e' mai stato allocato con {malloc, realloc, calloc}.
Questo non e' permesso. Quel blocco in realta' risiede in
memoria globale perche' e' una stringa costante. Le stringhe
costanti non vanno mai deallocate con free.

4) L'ultimo errore potrebbe non essere rilevato da Valgrind.
Esso riguarda riga 15 in quanto viene allocato un buffer locale
che pero puo' essere acceduto anche al termine dell'attivazione
della funzione corrente. Una variabile locale o buffer locale
non devono mai essere acceduti dopo che l'attiviazione della
funzione e' terminata perche' non sono piu' validi. Occorre
quindi allocare questo buffer dinamicamente con malloc()
e poi pero' deallocarlo in delete_list().

*/

#include <stdio.h>
#include <stdlib.h>

typedef struct node {
    char * id;
    char * value;
    struct node * next;
} node_t;

static int count = 0;

node_t* add_node(node_t* l, char* value) {
    node_t* node = malloc(sizeof(node_t));

    char* id = malloc(16); // questo buffer deve essere allocato dinamicamente e non lo
                           // deve sopravvivere all'attivazione della funzione!
                           // NOTA: Valgrind potrebbe non rilevare questo errore!
```

```
                            //       Infatti, Valgrind non rileva tutti gli errori
                            //       ma solo un sottoinsieme!
    sprintf(id, "ID_%d", count++);
    node->id = id;
    node->value = value;

    if (l != NULL)
        node->next = l;
    else
        node->next = NULL; // se l e' NULL, dobbiamo comunque inizializzare questo campo

    return node;
}

void print_list(node_t* head) {
    node_t* current = head;

    while (current != NULL) {
        printf("%s\n", current->value);
        current = current->next;
    }
}

void delete_list(node_t * head) {
    node_t* current = head;

    while (current != NULL) {
        free(current->id); // devo liberare questo blocca di memoria altrimente c'e' un
        // free(current->value); // al campo viene assegnata una stringa NON allocata c
        node_t* c = current; // salviamo l'attuale current
        current = current->next; // aggiorniamo current
        free(c); // noi vogliamo deallocare il nodo precedente all'attuale current

    }
}

int main() {
    node_t* l;

    l = add_node(NULL, "Hello");
    l = add_node(l, " ");
    l = add_node(l, "World");
    l = add_node(l, "!");

    print_list(l);
    delete_list(l);
    printf("Risultato 1/1\n");
    return 0;
}
```

Domande

1. A - costruito mediante porte logiche e calcola funzioni booleane
2. C - 220
3. B - la configurazione S=0 e R=1 non garantisce Q = NOT Q’
4. C - OR