# [T04] Esercitazione del 18 marzo 2022

## 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**: fate logout dal vostro account Google!
- **Se avete domande** accedete a Zoom agli orari stabiliti per l'esercitazione, accedendo con la **mail istituzionale** uniroma1.it dello studente. Troverete online i docenti ed il tutor del corso. In alternativa, scrivete via mail ai docenti.
- **Suggerimento** Non traducete direttamente da C a IA32, ma scrivere prima una versione C intermedia `E1/e1_eq.c` equivalente a quella di partenza, ma più semplice da tradurre in assembly. Testatela con il main di prova prima di passare a scrivere la versione `.s`. E' inutile tradurre la versione C equivalente se è errata!

Per maggiori informazioni fate riferimento al regolamento delle esercitazioni.

## Esercizio 1 (Numeri di Fibonacci)

Tradurre nel file `E1/e1.s` la seguente funzione C contenuta in `E1/e1.c` che calcola i numeri di Fibonacci:

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

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

## Esercizio 2 (Conto numero elementi uguali)

Tradurre nel file `E2/e2.s` la seguente funzione C contenuta in `E2/e2.c` che, dati due array di `short`, conta il numero di indici per cui gli array hanno lo stesso valore:

```
int counteq(short* v1, short* v2, int n) {
    int i, cnt = 0;
    for (i=0; i<n; ++i) cnt += (v1[i]==v2[i]);
    return cnt;
}
```

*Suggerimento*: usare le istruzioni SETcc e MOVZ/MOVS.

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

## Esercizio 3 (Clonazione buffer di memoria)

Tradurre nel file `E3/e3.s` la seguente funzione C contenuta in `E3/e3.c` che clona un blocco di memoria di `n` byte all'indirizzo `src`. Il nuovo blocco deve essere allocato con `malloc` e deve avere lo stesso contenuto del blocco `src`. Sarà compito del chiamante di `clone` deallocare il blocco di memoria allocato da `clone`.

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

void* clone(const void* src, int n) {
    void* des = malloc(n);
    if (des==0) return 0;
    memcpy(des, src, n);
    return des;
}
```

*Suggerimento*: per copiare i dati `src` al nuovo blocco si suggerisce di usare la funzione `memcpy`. Si noti che è possibile chiamare funzioni di libreria C con call come se fossero normali funzioni scritte dall'utente.

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

***NOTA***: in caso di errore in fase di linking su `memcpy` per via di PIE, usare `gcc -m32 e3_main.c e3.s -o e3 -no-pie`

## Esercizio 4 (Palestra C)

Scrivere nel file `E4/e4.c` una funzione C `drop_spaces` che, data una stringa `text`, elimina tutti gli spazi.

```c
void drop_spaces(char* text);
```

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

## Esercizio 5 (Domande)

1. Se una funzione `foo` ha un prologo in cui vengono salvati due registri callee-save e vengono riservati 12 byte per ospitare variabili locali, argomenti ed eventuale padding, quale di questi operandi permette di accedere al secondo argomento di `foo`?
    - A. `(%esp)`
    - B. `4(%esp)`
    - C. `8(%esp)`
    - D. `12(%esp)`
    - E. `20(%esp)`
    - F. `24(%esp)`
    - G. `28(%esp)`
    - H. `32(%esp)`
2. Assumendo `%al = 5`, eseguire `movsbl %al, %eax` porta allo stesso risultato in `%eax` rispetto eseguire `movzbl %al, %eax`?

   - A. Sì
   - B. No
3. Assumendo di avere una funzione `foo` che chiama una funzione `baz`. Quale tra le seguenti affermazioni risulta essere vera:
   - A. `foo` non può utilizzare il registro `%eax`
   - B. `foo` non può utilizzare il registro `%ebx`
   - C. `baz` non può utilizzare il registro `%eax`
   - D. `baz` non può utilizzare il registro `%ebx`
   - E. Nessuna delle precedenti affermazioni è vera
4. Se una funzione `baz` viene chiamata da una funzione `foo`, quale delle seguenti affermazioni risulta essere **falsa**:
   - A. `baz` prima di poter utilizzare `%edi` deve salvare il suo contenuto e ripristinarlo prima di effettuare la `ret`
   - B. `baz` può utilizzare `%edx` senza dover preservare il suo contenuto iniziale
   - C. `foo` deve salvare il contenuto di `%ecx` se vuole preservarne il contenuto prima di chiamare `baz`
   - D. `foo` deve salvare il contenuto di `%esi` se vuole presevarne il valore prima di chiamare `baz`
5. Quale fra le seguenti istruzioni risulta essere valida:
   - A. `setl %eax`
   - B. `setba %al`
   - C. `leal (%eax, %edx, 6), %ecx`
   - D. `movl (%eax), 4(%esp)`
   - E. `leal -1(%ecx), %eax`
   - F. `addl %eax, $4`
6. Assumendo che `%eax=0x0000BEEF`, quanto vale `%ecx` dopo aver eseguito l'istruzione `movsbw %al, %cx`?
   - A. `0x000000EF`
   - B. `0xFFFFFFEF`
   - C. `0x0000FFEF`
   - D. `0x0000EFEF`
7. Se `%ecx=0`, qual è il valore di `%al` dopo le istruzioni `testl %ecx, %ecx` e `setge %al`
   - A. 0
   - B. 1
   - C. nessuna delle precedenti