UNIVERSITÀ DEGLI STUDI DI MILANO
matteo.re@unimi.it
https://homes.di.unimi.it/re/arch2-lab-2015-2016.html

DIPARTIMENTO DI INFORMATICA
Laboratorio di Architetture degli Elaboratori II
Turno B : (G – Z)

9

Allocazione dinamica

SOMMARIO:

1) Allocazione dinamica memoria
2) Heap
3) Chiamata di sistema sbrk
4) E1. Allocazione dinamica, scrittura e lettura nel blocco di memoria ottenuto
5) Record e struct
6) E2. Allocazione dinamica di record, inizializzazione, copia da record a record
7) E3. Richiesta all'utente dei dati da inserire nel record
8) E4. Allocazione dinamica di array di dimensione variabile
9) Esempio possibile progetto esame

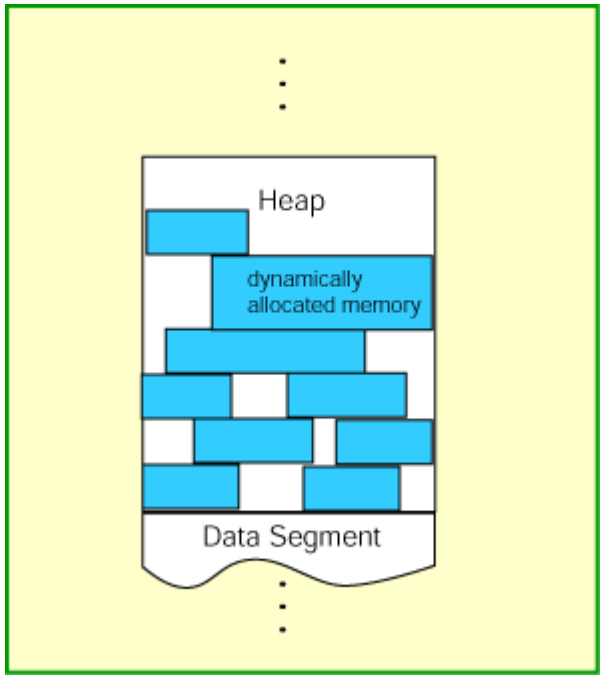
Ultimo aggiornamento: 6/5/2016

L’allocazione dinamica della memoria si verifica quando un programma richiede al sistema operativo un blocco di memoria dichiarando, preventivamente, la dimensione dello stesso.

Il blocco ricevuto può essere utilizzato in vari modi:

. Storage di un singolo valore
. Storage di diversi tipi di dati (ad es. interi, float …) all’interno di un’unica struttura dati

La memoria può essere restituita al sistema quando non è più necessaria al programma e non ci sono vincoli riguardo all’ordine in cui la memoria allocata viene restituita al sistema (differenza rispetto a quanto avviene quando si utilizza lo stack).



I blocchi di memoria richiesti dal programma vengono dallo heap, ossia la regione posta al di sopra della regione contenente i dati statici del programma.

A differenza dello stack che viene gestito tramite una logica di tipo LIFO lo heap (letteralmente «mucchio») assomiglia più ad una libreria in cui i libri vengono prelevati e ricollocati creando un pattern di arree utilizzate e non utilizzate.

Il modo in cui si effettua la richiesta per l'allocazione dinamica di un blocco di memoria è il seguente:

```
li      $a0,xxx    # $a0 contiene il numero di byte richiesti.
                   # xxx deve essere multiplo di 4.
li      $v0,9      # codice 9 == allocazione memoria (sbrk)
syscall            # chiemete sistema.
                   # $v0 <-- contiene indirizzo del primo byte
                   # del blocco allocato dinamicamente
```

NB: non c'è modo di conoscere in anticipo il range di memoria che verrà restituito. L'unica garanzia è che si tratterà di un blocco contiguo della dimensione richiesta

Esercizio 1

Scrivere un programma che:

i) Richieda l'allocazione dinamica di un blocco di memoria di 4 byte contigui
ii) Effettui una copia (di sicurezza) dell'indirizzo di memoria al quale inizia il blocco in **$s0**
iii) Salvi il valore di un intero nel blocco di memoria ottenuto
iv) Come dimostrazione del fatto che gli step 1-3 sono andati a buon fine leggere l'intero salvato nel blocco e stamparlo.

L'indirizzo del blocco è ottenuto a runtime. Non è possibile ottenerlo per utilizzarlo sotto forma di indirizzo simbolico nel programma. Tuttavia esso è disponibile in $s0.

Operazioni come questa in linguaggi di programmazione ad alto livello sono implementati come funzioni dedicate. Nel linguaggio C la medesima operazione di allocazione dinamica realizzata in questo esercizio è realizzata dalla funzione malloc().

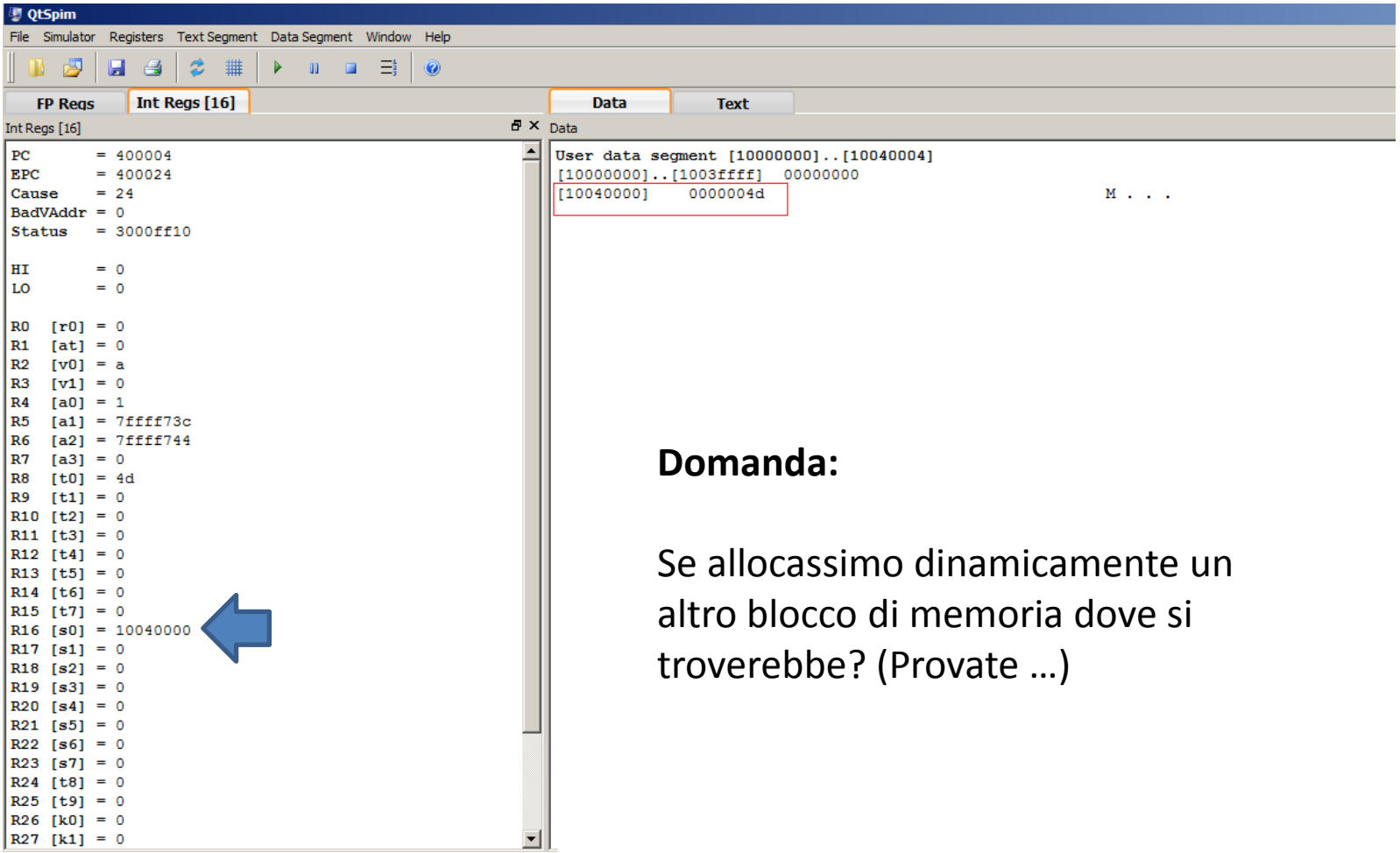
Esercizio 1

```
# DynAlloc.asm
# Allocate one block of memory, put an integer into it,
# print out the integer.

        .text
        .globl  main

main:
        li      $v0,9                   # (i) allocazione del blocco di memoria
        li      $a0,4                   # di 4 byte
        syscall                         # $v0 <-- indirizzo
        move    $s0,$v0                 # (ii) copia di sicurezza (non necessario ma è buona abitudine)

        li      $t0,77                  # (iii) salvare il valore 77
        sw      $t0,0($s0)              # nel blocco

        lw      $a0,0($s0)              # (iv a) caricare il contenuto del blocco in $a0
        li      $v0,1                   # caricare il valore 1 in $v0
        syscall                         # (iv b) stampare l'intero

        li      $v0,10                  # exit
        syscall

## end of file
```

Esercizio 1

Abbiamo bisogno di conoscere l'indirizzo del blocco di memoria allocato dinamicamente?
No. Esso è disponibile salvato all'interno di alcuni registri (e nella copia di backup che avevamo effettuato).

Dopo l'esecuzione del programma l'indirizzo è disponibile in **$s0** . Verifichiamo che il valore inserito (77) si trova effettivamente all'indirizzo disponibile in questo registro.



Domanda:

Se allocassimo dinamicamente un altro blocco di memoria dove si troverebbe? (Provate …)

Record e struct

Il blocco di memoria in grado di ospitare questa struttura consiste di **12** byte (4 per ognuno dei tre interi che la compongono). Il primo intero (eta) è collocato ad uno spiazzamento di **0** rispetto all'indirizzo in memoria del blocco della struttura, il secondo campo (l'intero salario) è collocato ad uno spiazzamento di **4** rispetto all'inizio del blocco ed il terzo (l'intero livello) ad uno spiazzamento di **8**.

```
struct
{
  int eta;
  int salario;
  int livello;       # della posizione lavorativa
};
```

Domanda: Se **$s1** contiene l'indirizzo in memoria al quale inizia il blocco contenente la struttura. Come dovremmo completare la seguente riga di codice per caricare nel registro **$t0** il contenuto del campo salario?

```
lw $t0, … (    …    )     # carica salario in $t0
```

Esercizio 2

Scrivere un programma che:

i) Allochi dinamicamente un blocco di memoria per la struttura rappresentata in questa slide
ii) Inserisca nei suoi campi i valori **34**, **24000** e **12** (in quest'ordine)
iii) Allochi dinamicamente la memoria per una seconda struttura
iv) Copi, nei campi della seconda struttura, i valori presenti nei campi della prima

```
struct
{
  int eta;
  int salario;
  int livello;       # della posizione lavorativa
};
```

Esercizio 2

```
# CopiaStruct.asm
.text
        .globl  main

main:
        # creazione prima struct
        li      $v0,9               # allocazione memoria
        li      $a0,12              # 12 byte
        syscall                     # $v0 <-- indirizzo
        move    $s1,$v0             #         prima struct   (backup in $s1)

        # inizializzazione della struct
        li      $t0,34              # salva (34)10 -> (22)16
        sw      $t0,0($s1)          # in eta
        lw      $t0,salario         # salva (24000) -> (5DC0)16
        sw      $t0,4($s1)          # in salario
        li      $t0,12              # salva (12)10 -> (C)16
        sw      $t0,8($s1)          # in livello

        # creazione seconda struct
        li      $v0,9               # allocazione memoria
        li      $a0,12              # 12 byte
        syscall                     # $v0 <-- indirizzo
        move    $s2,$v0             #         seconda struct   (backup in $s2)

        # copia dati dalla prima alla seconda struct
        lw      $t0,0($s1)          # copia eta dalla prima
        sw      $t0,0($s2)          # alla seconda struttura

        lw      $t0,.....($s1)      # copia salario dalla prima

        sw      $t0,.....($s2)      # alla seconda struttura

        lw      $t0,.....($s1)      # copia livello dalla prima

        sw      $t0,.....($s2)      # alla seconda struttura

        li      $v0,10              # exit
        syscall

        .data
salario: .word   24000              # salario (in memoria statica)
```

Domanda:

Che valori devo inserire in corrispondenza delle linee rosse?

Esercizio 3

Modificare il programma di esercizio 2 in modo che:

i) Esegua le stesse operazioni realizzate in esercizio 2 …
ii) ma chieda all'utente di inserire i tre valori interi da salvare nelle struct

suggerimento:

Service	System Call Code	Arguments	Result
print integer	1	$a0 = value	(none)
print float	2	$f12 = float value	(none)
print double	3	$f12 = double value	(none)
print string	4	$a0 = address of string	(none)
read integer	5	(none)	$v0 = value read
read float	6	(none)	$f0 = value read
read double	7	(none)	$f0 = value read
read string	8	$a0 = address where string to be stored $a1 = number of characters to read + 1	(none)
memory allocation	9	$a0 = number of bytes of storage desired	$v0 = address of block
exit (end of program)	10	(none)	(none)
print character	11	$a0 = integer	(none)
read character	12	(none)	char in $v0

Esercizio 3

```
# CopiaStruct_e3.asm
#
        .text
        .globl  main

main:
        # creazione prima struct
        li      $v0,9           # allocazione memoria
        li      $a0,12          # 12 byte
        syscall                 # $v0 <-- indirizzo
        move    $s1,$v0         # $s1 prima struct

        # inizializzazione della struct
                        # chiedere inserimento eta
                        li $v0, 4                       # codice system call per print_str
                        la $a0, str1                    # indirizzo stringa da stampare
                        syscall                         # stampa stringa
                        # leggere valore per eta
                        li $v0,5                         # codice system call per read_integer
                        syscall                          # lettura intero (e storage in $v0)

                        sw $v0,0($s1)                    # salva valore letto in eta

                        # chiedere inserimento salario
                        li $v0, 4                       # codice system call per print_str
                        la $a0, str2                    # indirizzo stringa da stampare
                        syscall                         # stampa stringa
                        # leggere valore per salario
                        li $v0,5                        # codice system call per read_integer
                        syscall                        # lettura intero (e storage in $v0)

                        sw $v0,4($s1)                   # salva valore letto in salario

                        # chiedere inserimento livello
                        li $v0, 4                       # codice system call per print_str
                        la $a0, str3                    # indirizzo stringa da stampare
                        syscall                         # stampa stringa
                        # leggere valore per livello
                        li $v0,5                        # codice system call per read_integer
                        syscall                         # lettura intero (e storage in $v0)

                        sw $v0,8($s1)                    # salva valore letto in livello
```

(Continua …)

Esercizio 3

```
# CopiaStruct_e3.asm

...
```

(Continua …)

```
 # creazione seconda struct
        li      $v0,9             # allocazione memoria
        li      $a0,12            # 12 byte
        syscall                   # $v0 <-- indirizzo
        move    $s2,$v0           #         seconda struct  (backup in $s2)

        # copia dati dalla prima alla seconda struct
        lw      $t0,0($s1)        # copia eta dalla prima
        sw      $t0,0($s2)        # alla seconda struttura

        lw      $t0,4($s1)        # copia salario dalla prima

        sw      $t0,4($s2)        # alla seconda struttura

        lw      $t0,8($s1)        # copia livello dalla prima

        sw      $t0,8($s2)        # alla seconda struttura

        li      $v0,10            # exit
        syscall

        .data
str1: .asciiz "inserire eta: "
str2: .asciiz "inserire salario: "
str3: .asciiz "inserire livello: "
```

Esercizio 4

5. [6] Un programma Assembly MIPS ha bisogno di utilizzare due array. Il primo, allocato staticamente, è costituito da 15 elementi, tutti contenenti inizialmente il valore 42. Il secondo viene allocato dinamicamente dopo averne chiesto all'utente, da terminale, il numero di elementi, e viene pure inizializzato a 42. Scrivere il codice che costruisce tali array.

Esercizio 4

5. [6] Un programma Assembly MIPS ha bisogno di utilizzare due array. Il primo, allocato staticamente, è costituito da 15 elementi, tutti contenenti inizialmente il valore 42. Il secondo viene allocato dinamicamente dopo averne chiesto all'utente, da terminale, il numero di elementi, e viene pure inizializzato a 42. Scrivere il codice che costruisce tali array.

```
# Data declarations
.data
str1: .asciiz "inserire numero elementi in array2: "
array1:     .word  42, 42, 42, 42, 42, 42, 42, 42, 42, 42
            .word  42, 42, 42, 42, 42

# code section
.text
.globl  main
.ent  main

main:

# chiedere inserimento numero elementi in array2
li $v0, 4                           # codice system call per print_str
la $a0, str1                        # indirizzo stringa da stampare
syscall                             # stampa stringa

# leggere valore inserito
li $v0,5                            # codice system call per read_integer
syscall                             # lettura intero (e storage in $v0)
addu $t0, $v0, 0                    # backup valore numero elementi N in array

# calcolo dimensione dell'array richiesto in byte
sll $t1, $t0, 2                     # 4 * N

subu $sp, $sp, 4                    # metto risultato sullo stack
```

Esercizio 4

5. [6] Un programma Assembly MIPS ha bisogno di utilizzare due array. Il primo, allocato staticamente, è costituito da 15 elementi, tutti contenenti inizialmente il valore 42. Il secondo viene allocato dinamicamente dopo averne chiesto all'utente, da terminale, il numero di elementi, e viene pure inizializzato a 42. Scrivere il codice che costruisce tali array.

```
# Data declarations
.data
str1: .asciiz "inserire numero elementi in array2: "
array1:     .word  42, 42, 42, 42, 42, 42, 42, 42, 42, 42
            .word  42, 42, 42, 42, 42

# code section
.text
.globl  main
.ent  main

main:

# chiedere inserimento numero elementi in array2
li $v0, 4                           # codice system call per print_str
la $a0, str1                        # indirizzo stringa da stampare
syscall                             # stampa stringa

# leggere valore inserito
li $v0,5                            # codice system call per read_integer
syscall                             # lettura intero (e storage in $v0)
addu $t0, $v0, 0                    # backup valore numero elementi N in array in $t0

# calcolo dimensione dell'array richiesto in byte  (usando Shift Left Logical, sll)
sll $t1, $t0, 2                     # 4 * N

subu $sp, $sp, 4                    # metto risultato sullo stack
sw   $t1, 0($sp)
```

Esercizio 4

5. [6] Un programma Assembly MIPS ha bisogno di utilizzare due array. Il primo, allocato staticamente, è costituito da 15 elementi, tutti contenenti inizialmente il valore 42. Il secondo viene allocato dinamicamente dopo averne chiesto all'utente, da terminale, il numero di elementi, e viene pure inizializzato a 42. Scrivere il codice che costruisce tali array.

```
# Data declarations
.data
str1: .asciiz "inserire numero elementi in array2: "
array1:     .word  42, 42, 42, 42, 42, 42, 42, 42, 42, 42
            .word  42, 42, 42, 42, 42

# code section
.text
.globl  main
.ent  main

main:

# chiedere inserimento numero elementi in array2
li $v0, 4                           # codice system call per print_str
la $a0, str1                        # indirizzo stringa da stampare
syscall                             # stampa stringa

# leggere valore inserito
li $v0,5                            # codice system call per read_integer
syscall                             # lettura intero (e storage in $v0)
addu $t0, $v0, 0                    # backup valore numero elementi N in array in $t0

# calcolo dimensione dell'array richiesto in byte  (usando Shift Left Logical, sll)
sll $t1, $t0, 2                     # 4 * N

subu $sp, $sp, 4                    # metto risultato sullo stack
sw   $t1, 0($sp)
```

Esercizio 4

5. [6] Un programma Assembly MIPS ha bisogno di utilizzare due array. Il primo, allocato staticamente, è costituito da 15 elementi, tutti contenenti inizialmente il valore 42. Il secondo viene allocato dinamicamente dopo averne chiesto all'utente, da terminale, il numero di elementi, e viene pure inizializzato a 42. Scrivere il codice che costruisce tali array.

(CONTINUA …)

```
# allocazione dinamica memoria per array
li      $v0,9                       # allocazione memoria (codice sbrk)
lw      $a0,0($sp)                  # dimensione array richiesto in byte
syscall                             # $v0 <-- indirizzo array allocato
move    $s2,$v0                     # backup indirizzo in $s2

# Inizializzazione array:
# num elementi array disponibile in $t0
# indirizzo in memoria del primo elemento dell'array disponibile in $s2
# valore da scrivere inserito in $a2 indice loop in $s3
li          $a2, 42
li          $s3, 0

DynArrInitLoop:
            sw $a2, 0($s2)
            addu $s2, $s2, 4
            addu $s3, $s3, 1
            blt $s3, $t0, DynArrInitLoop # if indice < len, loop

# exit
addu $sp, $sp, 4

li  $v0, 10
syscall

.end main
```

Esercizio 4

5. [6] Un programma Assembly MIPS ha bisogno di utilizzare due array. Il primo, allocato staticamente, è costituito da 15 elementi, tutti contenenti inizialmente il valore 42. Il secondo viene allocato dinamicamente dopo averne chiesto all'utente, da terminale, il numero di elementi, e viene pure inizializzato a 42. Scrivere il codice che costruisce tali array.

(CONTINUA …)

```
# allocazione dinamica memoria per array
li      $v0,9                       # allocazione memoria (codice sbrk)
lw      $a0,0($sp)                  # dimensione array richiesto in byte
syscall                             # $v0 <-- indirizzo array allocato
move    $s2,$v0                     # backup indirizzo in $s2

# Inizializzazione array:
# num elementi array disponibile in $t0
# indirizzo in memoria del primo elemento dell'array disponibile in $s2
# valore da scrivere inserito in $a2 indice loop in $s3
li          $a2, 42
li          $s3, 0

DynArrInitLoop:
            sw $a2, 0($s2)
            addu $s2, $s2, 4
            addu $s3, $s3, 1
            blt $s3, $t0, DynArrInitLoop # if indice < len, loop

# exit
addu $sp, $sp, 4

li  $v0, 10
syscall

.end main
```

Esercizio 4

5. [6] Un programma Assembly MIPS ha bisogno di utilizzare due array. Il primo, allocato staticamente, è costituito da 15 elementi, tutti contenenti inizialmente il valore 42. Il secondo viene allocato dinamicamente dopo averne chiesto all'utente, da terminale, il numero di elementi, e viene pure inizializzato a 42. Scrivere il codice che costruisce tali array.

(CONTINUA …)

```
# allocazione dinamica memoria per array
li      $v0,9                       # allocazione memoria (codice sbrk)
lw      $a0,0($sp)                  # dimensione array richiesto in byte
syscall                             # $v0 <-- indirizzo array allocato
move    $s2,$v0                     # backup indirizzo in $s2

# Inizializzazione array:
# num elementi array disponibile in $t0
# indirizzo in memoria del primo elemento dell'array disponibile in $s2
# valore da scrivere inserito in $a2 indice loop in $s3
li          $a2, 42
li          $s3, 0

DynArrInitLoop:
            sw $a2, 0($s2)
            addu $s2, $s2, 4
            addu $s3, $s3, 1
            blt $s3, $t0, DynArrInitLoop # if indice < len, loop

# exit
addu $sp, $sp, 4

li  $v0, 10
syscall

.end main
```

Esercizio 5

Si scriva un programma che gestisca una tabella di giocatori:

GIOCATORE_id | PUTEGGIO | NUM_PARTITE GIOCATE

Funzionalità:

- inserimento nuovo giocatore
- inserimento partita: l'utente specifica il punteggio finale (es 3 a 2) e il punteggio in classifica dei rispettivi giocatori viene aggiornato
- stampa classifica
- giocatore con numero massimo/minimo di partite
- altro…