

Niko Dalla Noce - niko.dalla@stud.unifi.it  
Alberto Giglioli - alberto.giglioli@stud.unifi.it  
Gabriele Vallario - gabriele.vallario@stud.unifi.it

Data di consegna - 3 luglio 2017

### **Esercizio di progetto (1): analisi di stringhe**

Utilizzando QtSpim, scrivere e provare un programma che prende in input una stringa qualsiasi di dimensione massima di 100 caratteri (es, "uno due ciao 33 tree tre uno Uno di eci"), e traduce ogni sua sottosequenza di caratteri separati da almeno uno spazio (nell'esempio, le sottosequenze "uno", "due", "ciao", "tree", "tre", "uno" "Uno" "di", "eci") applicando le seguenti regole:

- ogni sottosequenza "uno" si traduce nel carattere '1';
- ogni sottosequenza "due" si traduce nel carattere '2';
- ogni sottosequenza "nove" si traduce nel carattere '9';
- qualsiasi altra sottosequenza si traduce nel carattere '?'.

Nell'esempio considerato, se "uno due ciao 33 tree tre uno Uno di eci" è la stringa di input inserita da tastiera, a seguito della traduzione il programma dovrà stampare su console la seguente stringa di output:

- Risultato della traduzione: 1 2 ? ? ? ? 1 ? ??

### **Descrizione della soluzione adottata:**

Il programma legge la stringa di massimo 100 caratteri inserita da tastiera e restituisce la traduzione di ogni sottosequenza a cui è associato uno specifico carattere.

Abbiamo scelto di implementare una procedura che scorre ogni carattere fino al raggiungimento di uno spazio; trovato lo spazio il programma verifica se la sottosequenza ottenuta può essere tradotta, in caso negativo associa a quella parola un "?".

Il metodo per tradurre la parola inizia eseguendo un confronto con il primo carattere della sottosequenza con il primo carattere di un'altra stringa, contenente una delle tre parole traducibili, creata precedentemente.

Se durante ogni confronto combaciano ogni singolo carattere allora stamperà una stringa contenente la traduzione della parola, come per esempio "uno due nove tree" -> "1 2 9 ?".

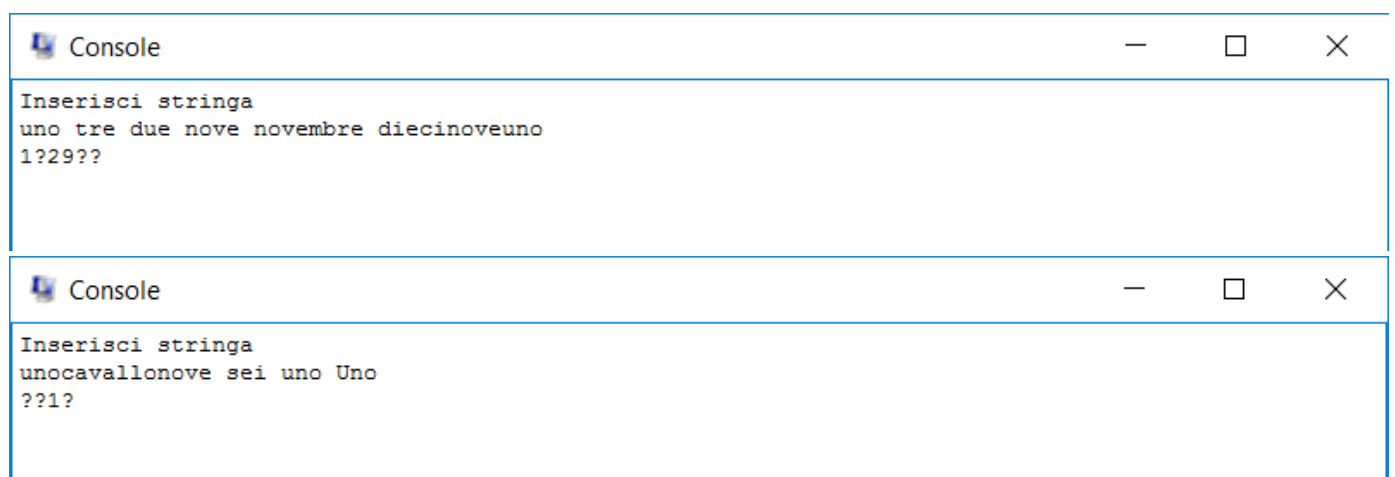
Il programma riesce ad individuare casi particolari come stringhe "novecento" perché, per esempio, dopo la sequenza di caratteri "nove" si aspetta uno spazio.

Nello svolgimento dell'esercizio sono stati utilizzati vettori per memorizzare le parole da tradurre ("uno", "due", "nove").

### Pseudo-linguaggio:

```
void traduction (String s)
{
    String[] split = s.split(" ");
    for each (word in split){
        switch (word)
        case "uno":
            print("1");
            break;
        case "due":
            print("2");
            break;
        case "nove":
            print("9");
            break;
        default:
            print("?");
            break;
    }
}
```

### Simulazione dell'esercizio:



```
Console
Inserisci stringa
uno tre due nove novembre diecinueveuno
1?29??

Console
Inserisci stringa
unocavallonove sei uno Uno
??1?
```

### Codice MIPS:

```

• .data
• comando : .ascii "Inserisci stringa \n"
• input: .space 102
• uno: .ascii "uno "
• due: .ascii "due "
• nove: .ascii "nove "
• appo: .space 101
•
• .text
• .globl main
•
• main:
•
• printRequest:
•
•         la $a0, comando
•         li $v0, 4
•         syscall
•
• takeInput:
•
•         li $v0, 8
•         #la $t2, str # t2 punta ad un carattere della stringa
•         la $a0, input
•         li $a1, 100
•         syscall
•
• init:
•         la $t3, appo      # t3 punta ad un carattere della stringa
•         li $t1, 0          # t1 conta la lunghezza
•         la $t2, input      # t2 punta ad un carattere della stringa
•         lb $s1, ($t2)
•         beq $s1, 32, end    #se la stringa è vuota esco
•
• addSpace:
•
•         lb $s1, ($t2)
•         beq $s1, 10, initializeInput
•         addi $t2, $t2, 1
• j addSpace
•
• initializeInput:
•
•         li $s2, 32
•         sb $s2, ($t2)

```

```

•          #addi $t2, $t2, 1
•          #sb $s2, ($t2)
•          la $t2, input
•
•
• nextCh: lb $t0, ($t2)          # leggo un carattere della stringa
•          sb $t0, ($t3)
•          add $t3, $t3, 1
•          add $t2, $t2, 1      # incremento la posizione sulla stringa
•          beqz $t0, end        # se è zero ho finito (pseudoistruzione)
•          bne $t0, 32, nextCh  # vedo se c'è uno spazio
•          j spaceFound
•          j nextCh            # continuo al prossimo carattere
•
• spaceFound:
•          la $t4, uno          #carico l'indirizzo di uno
•          la $t7, due          # carico due
•          la $t1, nove         # carico nove
•          la $t3, appo         #carico l'indirizzo di appo
•
•          j loopUno
•
• loopUno:
•          lb $t5, ($t4)        #carico in $t5 il carattere da confrontare di
•          uno
•          lb $t6, ($t3)        #carico in $t6 il carattere da confrontare di
•          appo
•
•          li $a0, 1
•          beqz $t5, print
•          bne $t5, $t6, loopDue #appena trovo un carattere che non coincide
•          salto al confronto
•          addi $t4,$t4, 1      #con il due, altrimenti incremento i
•          puntatori per poter
•          addi $t3, $t3,1      #confrontare i successivi
•          caratteri
•          addi $t8, $t8, 1
•          j loopUno
•
• loopDue:
•          lb $t5, ($t7)
•          lb $t6, ($t3)
•          li $a0, 2
•          beqz $t5, print      #loopDue esegue la solita procedura di
•          loopUno
•          bne $t5, $t6, loopNove

```

```

•      addi $t7,$t7, 1
•      addi $t3, $t3,1
•      addi $t8, $t8, 1
•      j loopDue
•
• loopNove: lb $t5, ($t1)
•          lb $t6, ($t3)
•          li $a0, 9
•          beqz $t5, print          #loopNove esegue la solita procedura di
loopUno
•          bne $t5, $t6, printBo
•          addi $t1,$t1, 1
•          addi $t3, $t3,1
•          addi $t8, $t8, 1
•          j loopNove
•
• print:
•          bgt $t8, 5, printBo      #se $t8 è maggiore di 5 stampa il ?
•          li $v0, 1                #sennò stampa la traduzione
•          syscall
•          la $t3, appo
•          j initializeAppo        #salto alla inizializzazione del vettore appo
•
•
• printBo:
•          li $a0, 63               #printBo esegue la stampa del punto interrogativo
•          li $v0, 11
•          syscall
•          la $t3, appo
•          j initializeAppo
•
• initializeAppo:
•
•          sb $t8, ($t3)
•          addi $t3, $t3, 1
•          lb $t5, ($t3)
•          beqz $t5, initAppo
•          j initializeAppo
•
• initAppo:
•          li $t8, 0
•          la $t3, appo
•          j nextCh
•
• end:
•          li $v0,10               #esce

```

- syscall

## Esercizio di progetto (2): procedure annidate e ricorsive

Siano

G e F due procedure definite come segue (in pseudo-linguaggio di alto livello):

```
Procedure
begin
  b
  for k := 0, 1, 2, . . . , n do
    begin
      end
      u :=
      b :=
return
end
```

$G(n)$   
 $0$   
 $F(k)$   
 $b^{2+u}$   
 $b$

Procedure									$F(n)$
begin									
if		$n$		$=$					0
then				<b>return</b>					1
else	<b>return</b>	$2 * F(n$	$-$	$1)$	$+$				$n$
end									

Utilizzando QtSpim, scrivere e provare un programma che legga inizialmente un numero naturale  $n$  compreso tra 1 e 8, e che visualizzi su console:

- il valore restituito dalla procedura  $G(n)$ , implementando  $G$  e  $F$  come descritto precedentemente. Le chiamate alle due procedure  $G$  ed  $F$  devono essere realizzate utilizzando l'istruzione jal (jump and link).

- la traccia con la sequenza delle chiamate annidate (con argomento tra parentesi) ed i valori restituiti dalle varie chiamate annidate (valore restituito fra parentesi), sia per  $G$  che per  $F$ .

Mostrare e discutere nella relazione l'evoluzione dello stack nel caso specifico in cui  $n=3$ .

Esempio di output su console nel caso in cui  $n=1$ :

Valore restituito dalla procedura:

$G(1)=4$

Traccia:

$G(1) \rightarrow F(0) \rightarrow F:\text{return}(1) \rightarrow F(1) \rightarrow F(0) \rightarrow F:\text{return}(1) \rightarrow F:\text{return}(3) \rightarrow G:\text{return}(4)$

Infatti:

$G(1)$  richiama  $F(0)$ , che ritorna il valore 1; quindi viene richiamata  $F(1)$ , che a sua volta richiama  $F(0)$ ;

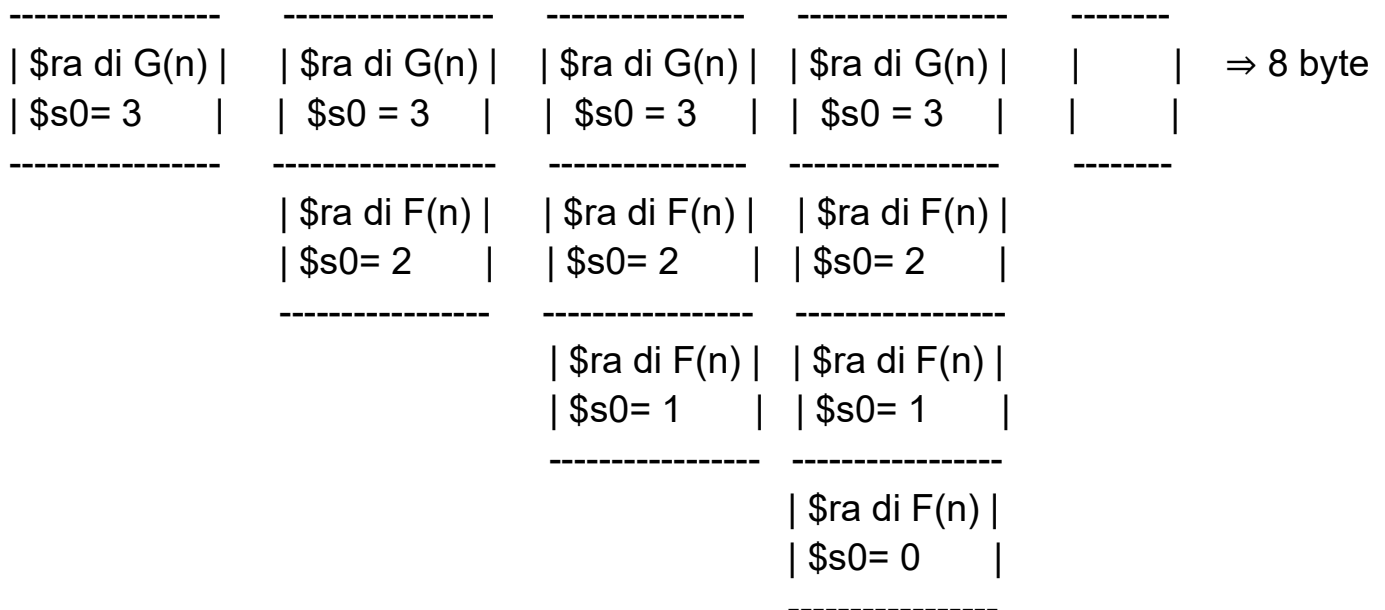
$F(0)$  ritorna il valore 1, quindi  $F(1)$  ritorna il valore 3, ed infine  $G(1)$  ritorna il valore 4 che è il risultato finale.

### Evoluzione dello stack:

La funzione  $G(n)$  chiama  $F(n)$ , la quale ricorsivamente si chiama  $n$  volte, ossia fino a quando non raggiunge il caso base. Per ogni chiamata viene memorizzato nello stack la  $n$  della chiamata e l'indirizzo da dove sono venuto (\$ra), nel primo caso è quello di jal, negli altri quello delle chiamate ricorsive della  $F(n)$ . Ad ogni chiamata

ricorsiva si abbassa lo stack pointer di 8 byte. Quando si arriva al caso base, si “procede al contrario” per calcolare tutti i valori che la F genera da ogni chiamata.

Per n=3:



Quando le chiamate terminano, lo stack si “riavvolge” e lo schema appena disegnato procede al contrario.

## Simulazione dell'esercizio:

Console

Inserisci numero:  
2  
F: 0 -> f: return 1 -> f: return 1 -> F: 1 -> F: 0 -> f: return 1 -> f: return 3 -> F: 2 ->  
F: 1 -> F: 0 -> f: return 1 -> f: return 8 -> G: return 24

Console

Inserisci numero:  
3  
F: 0 -> f: return 1 -> f: return 1 -> F: 1 -> F: 0 -> f: return 1 -> f: return 3 -> F: 2 ->  
F: 1 -> F: 0 -> f: return 1 -> f: return 8 -> F: 3 -> F: 2 -> F: 1 -> F: 0 -> f: return 1 ->  
f: return 19 -> G: return 595



Inserisci numero:

5

F: 0 -> f: return 1 -> f: return 1 -> F: 1 -> F: 0 -> f: return 1 -> f: return 3 -> F: 2 -> F: 1 ->  
 F: 0 -> f: return 1 -> f: return 8 -> F: 3 -> F: 2 -> F: 1 -> F: 0 -> f: return 1 -> f: return 19 ->  
 F: 4 -> F: 3 -> F: 2 -> F: 1 -> F: 0 -> f: return 1 -> f: return 42 -> F: 5 -> F: 4 -> F: 3 -> F: 2  
 -> F: 1 -> F: 0 -> f: return 1 -> Overflow

## Codice MIPS:

```

• .data
• richiestaIntero: .asciiz "Inserisci numero: \n"
• acapo: .asciiz "\n"
• ritorno: .asciiz "f: return "
• risultato: .asciiz "G: return "
• overflow: .asciiz "Overflow "
• effe: .asciiz "F: "
• freccia: .asciiz " -> "
•
• .text
• .globl main
•
• main:
•     li $v0, 4
•     la $a0, richiestaIntero
•     syscall
•
•     li $v0, 5
•     syscall
•
•     move $s5, $v0    # il numero inserito è in $a0
•     add $s5, $s5, 1
•
• G:
•     li $s4, 0        # b=0
•     li $s3, 1        # u=1
•
• for:
•
•     beq $s1, $s5, esci    #esci dal ciclo se $s1 è uguale a $s5
•     move $s0, $s1
•
•     la $a0, effe        #stampo la lettera F:
•     li $v0, 4
•     syscall
•

```

```

•      move $a0, $s1
•      li $v0, 1
•      syscall
•
•      la $a0, freccia          #stampo la freccia ->
•      li $v0, 4
•      syscall
•
•      jal F
•      mul $s4, $s4, $s4        # b al quadrato
•      mfhi $s7                  #controllo se il registro hi non è 0, perchè sennò
c'è overflow
•      bnez $s7, over
•      add $s4, $s4, $v0        # b^2 + u
•
•
•      move $t1, $v0
•
•      la $a0, ritorno          #stampo la stringa f: return
•      li $v0, 4
•      syscall
•
•      move $a0, $t1
•      li $v0, 1
•      syscall
•
•      la $a0, freccia
•      li $v0, 4
•      syscall
•
•      move $v0, $t1
•
•      addi $s1, $s1, 1
•      j for
•
• F:
•
•      bnez $s0, nonCasoBase
•      li $v0, 1
•
•      move $t1, $v0
•
•      la $a0, ritorno
•      li $v0, 4
•      syscall
•
•

```

```

•      move $a0, $t1
•      li $v0, 1
•      syscall
•
•      la $a0, freccia
•      li $v0, 4
•      syscall
•      move $v0, $t1
•
•      jr $ra
•
•
•
•
•      nonCasoBase:
•
•      addi $sp, $sp, -8 #abbasso lo stack pointer di 2 word
•      sw $s0, 0($sp)    # salvo la n di f(n)
•      sw $ra, 4($sp)    # salvo l'indirizzo da dove sono venuto
•      sub $s0, $s0, 1   # n-1
•
•      move $t1, $v0
•      la $a0, effe
•      li $v0, 4
•      syscall
•
•      move $a0, $s0
•      li $v0, 1
•      syscall
•
•      la $a0, freccia
•      li $v0, 4
•      syscall
•
•      move $v0, $t1
•
•      jal F
•
•      lw $s0, 0($sp) #Riprendo $s0 dallo stack
•      lw $ra, 4($sp) #Riprendo il punto da cui sono venuto.
•      addi $sp, $sp, 8 #Libero le due word dallo stack
•
•      mul $v0, $v0, 2   #moltiplico per 2 la chiamata precedente
•
•      add $v0, $v0, $s0 #faccio la somma n * $v0 e salvo il risultato in $v0
•      jr $ra           #torno da dove sono venuto.
•

```

```

• over:
•     la $a0, overflow
•     li $v0, 4
•     syscall
•     li $v0, 10          #stampo la stringa per avvertire che ce overflow
•     syscall
•
• esci:
•     la $a0, risultato
•     li $v0, 4
•     syscall
•
•     li $v0, 1
•     move $a0, $s4
•     syscall
•
•     li $v0, 10
•     syscall

```

### Esercizio di progetto (3): operazioni fra matrici

Utilizzando QtSpim, scrivere e provare un programma che visualizza all'utente un menù di scelta con le seguenti cinque opzioni *a*, *b*, *c*, *d*, *e*:

a) Inserimento di matrici. Il programma richiede di inserire da tastiera un numero intero  $0 < n < 5$ , e richiede quindi l'inserimento di due matrici quadrate, chiamate A e B, di dimensione  $n \times n$ , contenenti numeri interi. Quindi si ritorna al menù di scelta.

*Facoltativo.* Le matrici A e B dovranno essere allocate dinamicamente in memoria. Si consiglia l'utilizzo della system call 'sbrk' del MIPS.

Ogni volta che si seleziona l'opzione a) del menu, i nuovi valori inseriti di A e B dovranno essere salvati nella stessa area di memoria in cui erano stati salvati i vecchi valori: i nuovi valori sovrascriveranno quelli vecchi.

*Facoltativo:* Si dovrà allocare (con la 'sbrk') uno spazio aggiuntivo di memoria solo se le due nuove matrici dovessero richiedere più spazio di memoria rispetto a quello già allocato in precedenza.

Esempio di interfaccia per l'inserimento delle due matrici:

*Dimensione matrici: 3x3*

*Matrice A: Riga1: -2 44 5 Riga2: 1 1 1 Riga3: 3 0 1*

*Matrice B: Riga1: 0 0 10 Riga2: -1 1 -1 Riga3: 1 0 0*

b) Somma di matrici. Il programma effettua la somma fra le due matrici A e B, e visualizza su console il risultato A+B. Quindi si ritorna al menù di scelta. Ad esempio, se A e B sono state inserite come riportato nell'esempio al punto a), il programma dovrà visualizzare su console:

*Risultato di A+B: Riga1: -2 44 15 Riga2: 0 2 0 Riga3: 4 0 1*

c) Sottrazione di matrici. Il programma effettua la sottrazione fra le due matrici A e B, e visualizza su console il risultato A-B. Quindi si ritorna al menù di scelta.

Ad esempio, se A e B sono state inserite come riportato nell'esempio al punto a), il programma dovrà visualizzare su console:

*Risultato di A-B: Riga1: -2 44 -5 Riga2: 2 0 2 Riga3: 2 0 1*

d) Prodotto di matrici. Il programma effettua il prodotto fra le due matrici A e B, e visualizza su console il risultato A\*B. Quindi si ritorna al menù di scelta.

Ad esempio, se A e B sono state inserite come riportato nell'esempio al punto a), il programma dovrà visualizzare su console:

*Risultato di A\*B: Riga1: -39 44 -64 Riga2: 0 1 9 Riga3: 1 0 30*

e) Uscita. Stampa un messaggio di uscita ed esce dal programma.

**Descrizione della soluzione adottata:**

Alle opzioni *a*, *b*, *c*, *d* corrisponderanno le chiamate alle opportune procedure, e quindi il programma dovrà tornare disponibile per selezionare una nuova opzione. Alla scelta “e” corrisponderà la terminazione del programma.

Il programma come prima cosa chiede di inserire un comando tra quelli elencati da una stringa di benvenuto; se il primo comando inserito non è quello della creazione delle matrici il programma stamperà una stringa di errore, richiedendo in seguito di inserire nuovamente un comando.

Gli altri comandi sono l’addizione, la sottrazione, la moltiplicazione tra le due matrici e l’uscita dal programma.

La scelta di uno di questi comandi avviene tramite uno switch case selezionando per ogni comando una lettera scelta da “a” fino ad “e” come elencato precedentemente. La creazione delle matrici avviene grazie all’utilizzo del `sbrk`, il cui scopo principale è di aumentare o diminuire lo spazio di memoria associato ad un processo.

La somma e la sottrazione tra matrici avviene sommando o sottraendo le solite posizioni e riportando il risultato stampato in uscita così da non creare un’altra matrice che occuperebbe solo memoria in più.

La moltiplicazioni tra matrici è un’operazione un pò più complessa delle precedenti perché non sempre può essere eseguita: la moltiplicazione tra due matrici A e B può essere eseguita se la matrice A ha un numero di righe uguale al numero delle colonne della seconda matrice B; nel nostro caso la moltiplicazione è applicabile perché quando il programma richiede la grandezza delle matrici le crea quadratiche. Per eseguire la prima moltiplicazione il programma considera solo la prima riga della matrice A e solo la prima colonna della matrice B, moltiplica il primo elemento della riga della matrice A con il primo elemento della colonna della matrice B, somma il prodotto ad una variabile temporanea e si procede fino a che non esaurisce gli elementi della riga di A e gli elementi della colonna di B, una volta terminato la variabile temporanea che contiene il risultato delle somme delle moltiplicazioni verrà inserito nella posizione della prima riga e della prima colonna, cioè nella posizione dove le righe della matrice A e le colonne della matrice B si intersecano.

**Esempio di come è applicata la moltiplicazione:**

$$\begin{bmatrix} -1 & 3 & -2 \\ 2 & 0 & 1 \\ 1 & -5 & 3 \end{bmatrix} \times \begin{bmatrix} 1 & -2 \\ 0 & 2 \\ -5 & 0 \end{bmatrix} = \begin{array}{c|c} 9 & - \\ \hline - & - \\ \hline - & - \end{array}$$

$$9 = (-1) \times 1 + 3 \times 0 + (-2) \times (-5)$$

### Pseudo-linguaggio:

```
switch(comando inserito){
    case "a":
        creaLeMatrici();

    case "b":
        sommaTraMatrici();

    case "c":
        sottrazioneTraMatrici();

    case "d":
        moltiplicazioneTraMatrici();

    case "e":
        uscitaDalProgramma();
}

int[][] creaLeMatrici (int n){
    int[][] m1= new int [n][n];
    int[][] m2= new int [n][n];
    for(i=0 to n){
        for(j=0 to n){
            m1[i][j]=(numero inserito da tastiera);
        }
    }
    for(i=0 to n){
        for(j=0 to n){
            m2[i][j]=(numero inserito da tastiera);
        }
    }
}
```

```
}
```

```
int[][] sommaTraMatrici(int[][] m1, int[][] m2) {  
    int[][] res = new int[m1.length][m1.length];  
    for (i = 0 to m1.length) {  
        for (j = 0 to m1[i].length; j++) {  
            res[i][j] = m1[i][j] + m2[i][j];  
        }  
    }  
    return res;  
}
```

```
int[][] sottrazioneTraMatrici(int[][] m1, int[][] m2) {  
    int[][] res = new int[m1.length][m1.length];  
    for ( i = 0 to m1.length) {  
        for (j = 0 to m1[i].length) {  
            res[i][j] = m1[i][j] - m2[i][j];  
        }  
    }  
    return res;  
}
```

```
int[][] moltiplicazioneTraMatrici(int[][] m1, int[][] m2) {  
    int somma=0;  
    int riga=0, colonna=0;  
    int[][] res = new int[m1.length][m1.length];  
    for (i = 0 to m1.length) {  
        colonna=0;  
        for (j = 0 to m1[i].length) {  
            for (k=0 to m1[i].length){  
                somma = somma + m1[riga][k] * m2[k][colonna];  
            }  
            res[i][j] = somma;  
            somma=0;  
            colonna++;  
        }  
        riga++;  
    }  
    return res;  
}
```



**Simulazione dell'esercizio:**

Inserisci uno dei seguenti comandi:

- Comando a per eseguire l'inserimento delle matrici.
- Comando b per eseguire la somma tra le matrici create.
- Comando c per eseguire la sottrazione tra le matrici.
- Comando d per eseguire la moltiplicazioni tra matrici.
- Comando e per uscire dal programma.

a

Hai selezionato il comando a

Inserisci un numero intero n compreso tra  $0 < n < 5$  per definire la dimensione della matrice:

3

Matrice 1

inserisci riga 1

2

5

8

inserisci riga 2

6

3

2

inserisci riga 3

5

8

7

Matrice 2

inserisci riga 1

4

1

2

inserisci riga 2

3

6

4

inserisci riga 3

5

9

5

```
Console
inserisci un nuovo comando: b
Hai selezionato il comando b
Il risultato dell'operazione scelta:

6 6 10
9 9 6
10 17 12
inserisci un nuovo comando: c
Hai selezionato il comando c
Il risultato dell'operazione scelta:

-2 4 6
3 -3 -2
0 -1 2
inserisci un nuovo comando: d
Hai selezionato il comando d
Il risultato dell'operazione scelta:

63 104 64
43 42 34
79 116 77

inserisci un nuovo comando: e
Hai selezionato il comando e per uscire dal programma.
```

## Codice MIPS:

- .data
- strNuovoComando: .asciiz "\ninserisci un nuovo comando: "
- strA: .asciiz "\nInserisci un numero intero n compreso tra 0<n<5 per definire la dimensione della matrice: \n"
- strBenvenuto: .asciiz "\nInserisci uno dei seguenti comandi: \n Comando a per eseguire l'inserimento delle matrici.\n Comando b per eseguire la somma tra le matrici create.\n Comando c per eseguire la sottrazione tra le matrici.\n Comando d per eseguire la moltiplicazioni tra matrici.\n Comando e per uscire dal programma.\n"
- strErrore: .asciiz "\nIl comando selezionato non è accettabile.\n Riprovare altro comando.\n"
- strErrore1: .asciiz "\nIl numero inserito non è accettabile.\n Riprovare altro numero.\n"
- strUscita: .asciiz "\nHai selezionato il comando e per uscire dal programma.\n"
- stringaA: .asciiz "\nHai selezionato il comando a per creare le matrici\n"
- stringaB: .asciiz "\nHai selezionato il comando b che esegue la somma tra le matrici\n"
- stringaC: .asciiz "\nHai selezionato il comando c che esegue la sottrazione tra le matrici\n"
- stringaD: .asciiz "\nHai selezionato il comando d che esegue la moltiplicazione tra le matrici\n"
- stringaRis: .asciiz "Il risultato dell'operazione scelta: \n\n"
- secondaMatrice: .asciiz "\n Seconda matrice"
- str: .asciiz "inserisci riga "
- strColonne: .asciiz "Numero colonne: "
- insM1: .asciiz "\nMatrice 1\n"
- insM2: .asciiz "\nMatrice 2\n"
- acapo: .asciiz "\n"
- .text

```

● .globl main
●
● main:
●                                     # scelta della procedura o
dell'uscita
●         li $v0, 4                     # $v0 = codice della print_string
●         la $a0, strBenvenuto         # $a0 = indirizzo della stringa
●         syscall                      # stampa la strBenvenuto
●
● choice:
●
●         move $s4, $s3
●         mul $s4, $s4, $s4             # numero di celle della matrice
●
● # legge la scelta
●
●         li $v0, 12
●         syscall
●         move $t0, $v0                # $t2=scelta a,b,c,d o e
●         li $t1, 0                    # serve per andare a capo nella stampa delle matrici
●
●         beq $t0, 97, jA               #if($t2==a) vai a ja
●         beq $t0, 98, jB               #if($t2==b) vai a jb
●         beq $t0, 99, jC               #if($t2==c) vai a jc
●         beq $t0, 100, jD              #if($t2==d) vai a jd
●         beq $t0, 101, jE              #if($t2==e) vai a je
●
●         li $v0, 4
●         la $a0, strErrore             #legge la stringa errore perchè non è stato inserito
un comando corretto
●         syscall
●         j choice
●
● jA:
●
●         la $a0, stringaA              #stampa la stringa del comando a
●         li $v0, 4
●         syscall
●
● A:
●         la $a0, strA                  #stampa la richiesta di inserire un intero compreso
tra 0 e 5 esclusi
●         li $v0, 4
●
●         syscall
●
●         li $v0, 5
●         syscall                      #legge un intero inserito
●         move $s3, $v0                 #e lo sposta in $s3
●         move $s4, $s3
●         mul $s4, $s4, $s4
●         mul $s5, $s3, 4               # indirizzo per spostarmi alla cella della successiva
riga
●
●         sle $t2, $s3, $zero           # $t0=1 se $t1 <= 0
●         bne $t2, $zero, errore        # errore se scelta <=0
●         li $t0, 5
●         sle $t2, $t0, $s3

```

```

    bne $t2, $zero, errore # errore se scelta >=5

    li $v0, 4
    la $a0, insM1
    syscall

#ho inserito il numero della grandezza della matrice ed ora la riempio
allocazioneMatrice:
    mul $t4, $s3, $s3
    mul $t4, $t4, 4
    li $v0, 9          #il comando li $v0, 9 è il comando sbrk
che alloca in          #memoria le
matrici in sequenza
    move $a0, $t4
    syscall
    move $t0, $v0

    j printRequestRiga

inserimentoRiga:
    li $v0, 5
    syscall
    sw $v0, ($t0)      #inserisco il numero digitato da tastiera
nella posizione $t0
    addi $t0, $t0, 4    #incremento la posizione della matrice
    sub $t3, $t3, 1     #la sottrazione serve per vedere quante
                        #posizioni
mancono da riempire
    beqz $t3, printRequestRiga #se $t3 arriva a 0 salta alla
stampa di
# una nuova riga
    j inserimentoRiga

printRequestRiga:
    beq $s3, $s0, save #controllo il numero di righe da inserire
    li $v0, 4
    la $a0, str
    syscall

    li $v0, 1
    addi $s0, $s0, 1
    move $a0, $s0
    syscall

    li $v0, 4
    la $a0, acapo      #stampo \n per scrivere il numero seguente a
capo
    syscall

    move $t3, $s3
    j inserimentoRiga

save:
    li $s0, 0

```

```

●      addi $t5, $t5, 1          #il metodo save serve per decidere
quale delle due matrici
●
●      beq $t5, 1, indirizzoM1
●
●      beq $t5, 2, indirizzoM2
●      j allocazioneMatrice
●
indirizzoM1:
●      li $v0, 4                #questo metodo serve per salvare
la matrice numero 1
●
●      la $a0, insM2
●      syscall
●
●      mul $s1, $s3, $s3
●      mul $s1, $s1, 4
●      sub $t0, $t0, $s1
●      move $s1, $t0
●      j allocazioneMatrice
●
●
●
●
indirizzoM2:
●
●      mul $s2, $s3, $s3        #questo metodo serve per salvare la
matrice numero 2
●
●      mul $s2, $s2, 4
●      sub $t0, $t0, $s2
●      move $s2, $t0
●      li $v0, 4
●
●
●      altroComando:
●
●      la $a0, strNuovoComando
●      li $v0, 4                #in questo metodo resettiamo tutti i
registri pronti per
●
●      syscall
●
●      li $t5, 0
●      li $t6, 0
●      li $t9, 0
●      li $t2, 0
●      move $t3,$s1             #sposto l'indirizzo della prima matrice in
$t3
●      move $t4,$s2             #sposto l'indirizzo della seconda matrici
in $t4
●
●      j choice
●
●
●      jB:
●      la $a0, stringaB
●      li $v0, 4
●      syscall
●
●      la $a0, stringaRis

```

```

    li $v0, 4
    syscall

addizione:

    beqz $s4, altroComando

    lw $t7, ($t3) # carico primo elemento della prima matrice
    lw $t8, ($t4) # carico primo elemento della seconda matrice

    add $t7, $t7, $t8 #sommo il primo elemento con il secondo
elemento
    add $t3, $t3, 4 #mi sposto nella matrici numero 1
all'elemento seguente
    add $t4, $t4, 4 #stessa procedura della riga precedente
solo nella seconda matrice

    sub $s4, $s4, 1

    beq $t1, $s3, rigaDopoA
    add $t1, $t1, 1 #contatori
    move $a0, $t7
    li $v0, 1
    syscall

    li $a0, 32 #stampa uno spazio tra i numeri
    li $v0, 11
    syscall

    j addizione

rigaDopoA:

    li $t1, 1

    la $a0, acapo #questo metodo quando arriva in fondo alla
riga della matrice va a capo
    li $v0, 4 #mettendo tra ogni numero uno spazio per
distanziare i numeri
    syscall

    move $a0, $t7
    li $v0, 1
    syscall

    li $a0, 32
    li $v0, 11
    syscall

    j addizione

jC:
    la $a0, stringaC
    li $v0, 4
    syscall

```

```

    la $a0, stringaRis
    li $v0, 4
    syscall

sottrazione:
    beqz $s4, altroComando

    lw $t7, ($t3) # carico primo elemento della prima matrice
    lw $t8, ($t4) # carico primo elemento della seconda matrice

    sub $t7, $t7, $t8
    add $t3, $t3, 4
    add $t4, $t4, 4

    sub $s4, $s4, 1

    beq $t1, $s3, rigaDopoS # stampro la riga successiva quando $t1
arriva in fondo alla riga della matrice
    add $t1, $t1, 1 # contatori
    move $a0, $t7
    li $v0, 1
    syscall

    li $a0, 32
    li $v0, 11
    syscall

    j sottrazione

rigaDopoS:
    li $t1, 1

    la $a0, acapo
    li $v0, 4
    syscall

    move $a0, $t7
    li $v0, 1
    syscall

    li $a0, 32
    li $v0, 11
    syscall

    j sottrazione

jD:
    la $a0, stringaD
    li $v0, 4
    syscall

    la $a0, stringaRis
    li $v0, 4

```



```

●      syscall
●      li $t5, 0
●      move $t3, $s1      #salva in $t3 l'indirizzo della prima matrice
●      move $t4, $s2      #salva in $t4 l'indirizzo della seconda matrice,
vengono spostati in
●                                #questi due registri perchè in seguito
verranno incrementati
●      move $s6, $s2      #$s6 = indirizzo della colonna nella seconda
matrice                                # (usato come appoggio)
●
●      li $s7, 0
●
●      moltiplicazione:
●          beq $s7, $s3, altroComando
●          lw $t7, ($t3) # carico primo elemento della prima matrice
●          lw $t8, ($t4) # carico primo elemento della seconda matrice
●
●          mul $t7, $t7, $t8 #moltiplicazione dell'elemento della riga
della prima                                #matrice con l'elemento della
colonna della seconda matrice
●          add $t9, $t9, $t7  #$t9 sarà il risultato di una casella
della matrice da stampare
●
●          addi $t3, $t3, 4    #incremento di una posizione nella prima
matrice
●          add $t4, $t4, $s5    #mi sposto nella riga successiva nella
solita colonna
●
●          addi $t5, $t5, 1    #contatore che indica se sono arrivato in
fondo alla                                #colonna della seconda matrice
●
●          bne $t5, $s3, moltiplicazione
●
●          j colonnaDopo      #se ho finito una colonna mi sposto a quella
successiva
●
●      colonnaDopo:
●          addi $t1, $t1, 1      #contatore per sapere quando andare
alla riga dopo
●          beq $t1, $s3, rigaDopoM
●          li $v0, 1
●          move $a0, $t9          #stampa il risultato di quella casella
●          syscall
●          li $v0, 11
●          li $a0, 32            #$a0 in questo caso serve per spazioare
il risultato
●          syscall
●
●          li $t5, 0            #resetto il contatore delle colonne $t5
●          addi $s6, $s6, 4      #mi sposto nella colonna successiva
della seconda matrice
●          move $t4, $s6
●          sub $t3, $t3, $s5     #torna all'inizio della riga della prima
matrice
●          li $t9, 0            #resetto il registro $t9 per il prossimo
risultato
●

```

```

j moltiplicazione

rigaDopoM:
    beq $s7, $s3, altroComando    #se ho finito di scorre la matrice
    richiede un nuovo comando

    seconda matrice    li $t5,0        #resetto il contatore delle colonne della
                        li $v0, 1
                        move $a0, $t9    #stampa il risultato presente nel registro
    $t9                syscall
                        li $v0, 11
                        la $a0, 32
                        syscall
                        li $v0, 4
                        la $a0, acapo
                        syscall

                        li $t9,0        #resetto il risultato

    dopo                li $t1, 0        #resetto il contatore per andare alla riga

                        move $s6, $s2    #torna all'inizio della seconda matrice

                        move $t4, $s2    #resetto l'indice per la prossima operazione
    la matrice          addi $s7, $s7, 1    #$s7 serve per vedere se ho esaminato tutta
                        j moltiplicazione

jE:
    li $v0, 4
    la $a0, strUscita
    syscall

    j jExit

errore:
    li $v0, 4
    la $a0, strErrore1
    syscall # stampa la stringa strErrore1

    j A # ritorna alla richiesta di inserimento di un numero tra 1 e 4

jExit:
    li $v0, 10        # termina programma
    syscall

```