Architettura degli Elaboratori

MIPS:
Un esempio completo

Barbara Masucci
UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Punto della situazione

- Oggi vediamo un esempio completo di programma in assembly MIPS
  - Una procedura (ordina) che, preso in input un vettore di interi e la sua lunghezza, dà in output il vettore ordinato
  - Per svolgere il suo lavoro, la procedura fa uso di un'altra procedura (scambia) che consente di scambiare il contenuto di due locazioni consecutive del vettore
  - Quindi:
    - Caller: ordina
    - Callee: scambia

Procedura scambia

- Iniziamo a considerare la procedura scambia

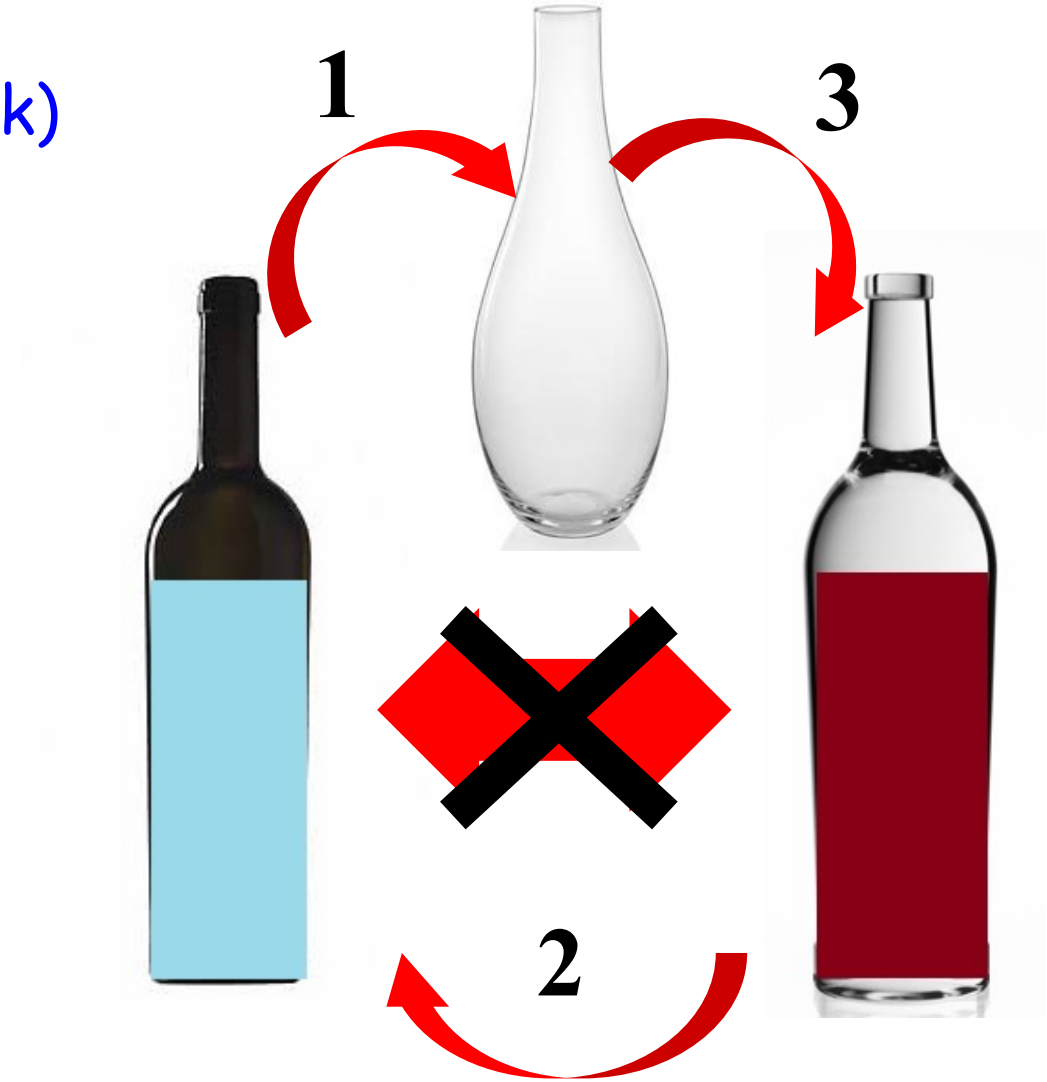
```
void scambia(int v[], int k)
 {
        int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
 }
```

Procedura scambia

```
void scambia(int v[], int k)
 {
        int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
 }
```

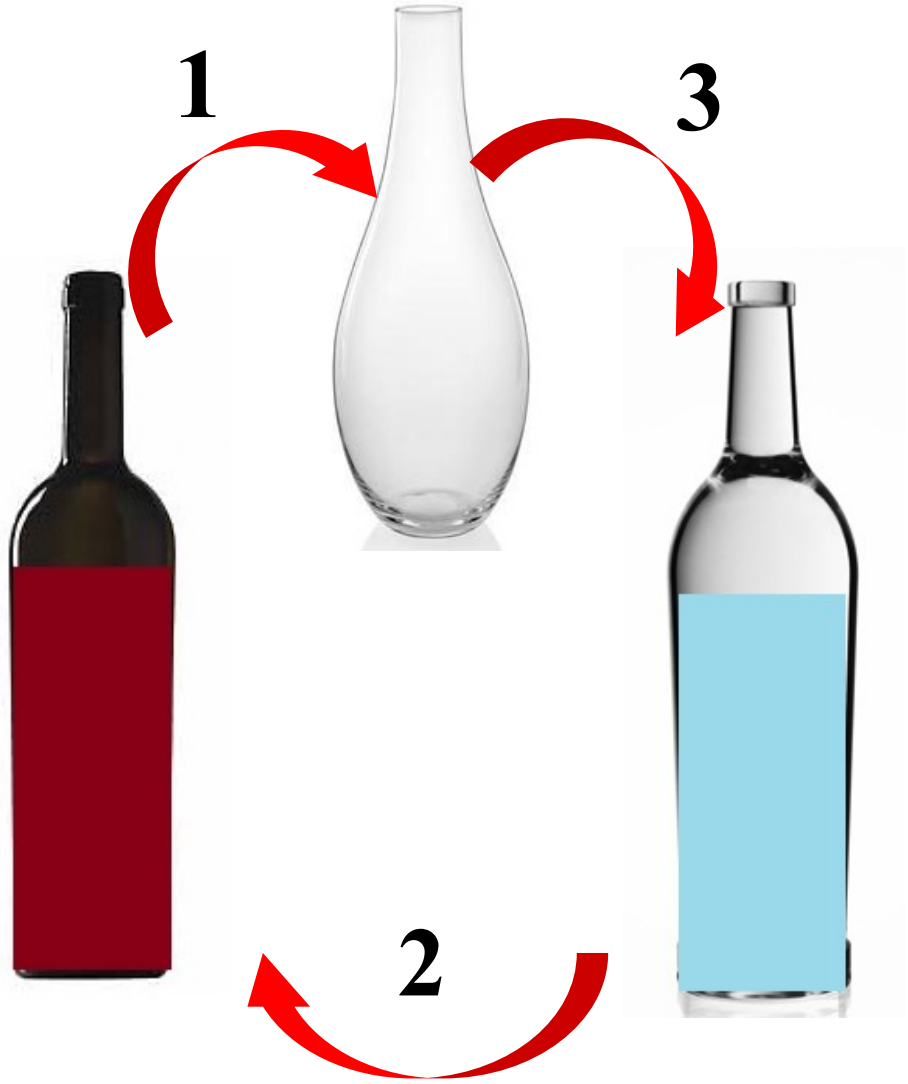
Procedura scambia

```
void scambia(int v[], int k)
 {
        int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
 }
```



Procedura scambia

```
void scambia(int v[], int k)
 {
        int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
 }
```



Procedura scambia

- Per tradurre da C ad assembler MIPS dobbiamo effettuare i passaggi seguenti:
  - Allocare i registri per le variabili del programma;
  - Scrivere il codice relativo al corpo della procedura;
  - Salvare il contenuto dei registri da eventuali modifiche effettuate dalla procedura.
  - Restituire il controllo al chiamante

```
void scambia(int v[], int k)
 {
        int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
 }
```

Procedura scambia

- Step 1: Allocazione dei registri
  - Il registro $a0 è utilizzato per l'indirizzo base del vettore v
  - Il registro $a1 è utilizzato per il parametro k
  - Infine, il registro $t0 è utilizzato per la variabile temp

```
void scambia(int v[], int k)
 {
        int temp;
        temp = v[k];
        v[k] = v[k+1];
        v[k+1] = temp;
 }
```

Procedura scambia

- Step 2: Scrivere il codice per il corpo della procedura
  - Innanzitutto otteniamo l'indirizzo per la word v[k]
    - sll $t1, $a1, 2 #$t1 contiene 4*k
    - add $t1, $a0, $t1 #$t1 contiene l'indirizzo di v[k]
  - Poi carichiamo v[k] in un registro temporaneo (temp)
    - lw $t0, 0($t1) #$t0 contiene v[k]
  - Inoltre, carichiamo v[k+1] in un altro registro temporaneo
    - lw $t2, 4($t1) #$t2 contiene v[k+1]
  - Infine, memorizziamo $t0 e $t2 e scambiamo la loro posizione in memoria
    - sw $t2, 0($t1) #registra v[k+1] in v[k]
    - sw $t0,4($t1) #registra v[k] in v[k+1]

```
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
```

Procedura scambia

- Step 3: Salvare il contenuto dei registri
  - Poiché la procedura non utilizza registri di tipo $s, ed essendo scambia una procedura foglia, non dobbiamo salvare il contenuto di nessun registro (nemmeno di $ra)
- Step 4: Restituire il controllo al chiamante
  - Quando la procedura termina il suo lavoro, restituisce il controllo al chiamante

jr $ra

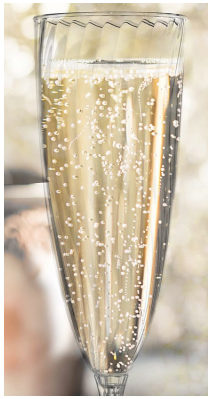
Procedura scambia

In definitiva, il codice assembler MIPS per la procedura scambia è quindi

```
SCAMBIA: sll $t1, $a1, 2  #$t1 contiene 4*k
         add $t1, $a0, $t1  #$t1 contiene l'indirizzo di v[k]
         lw $t0, 0($t1)  #$t0 contiene v[k]
         lw $t2, 4($t1)  #$t2 contiene v[k+1]
         sw $t2, 0($t1) #registra v[k+1] in v[k]
         sw $t0,4($t1) #registra v[k] in v[k+1]
         jr $ra #ritorno alla procedura chiamante
```

Procedura ordina

- Come possiamo usare la procedura scambia per effettuare l'ordinamento di un vettore?
- Vediamo un algoritmo che si chiama bubble sort
  - L'algoritmo scandisce più volte il vettore, confrontando elementi adiacenti e scambiando di posto quelli che non sono nell'ordine giusto
  - Il primo step posiziona l'elemento più grande all'ultima locazione del vettore
  - Il secondo step posiziona il secondo elemento più grande alla penultima locazione del vettore
  - Il nome deriva dal fatto che ad ogni iterazione l'elemento più grande si sposta verso destra, come una bolla in un bicchiere di champagne sale verso l'alto



Procedura ordina

Prima iterazione

Vettore iniziale (n=5)

Primo confronto: nessuno scambio

Secondo confronto: necessario scambio

Effettuato scambio

Terzo confronto: nessuno scambio

Quarto confronto: necessario scambio

Effettuato scambio

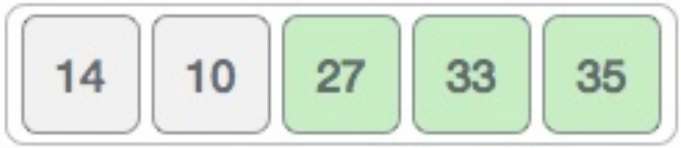
Al termine della prima iterazione, l'elemento più grande si trova in fondo al vettore

Procedura ordina

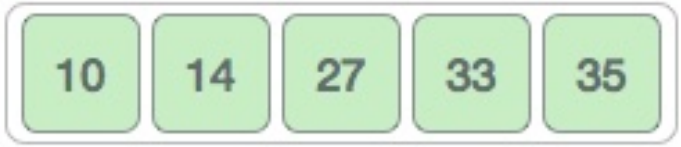
Al termine della prima iterazione

Al termine della seconda iterazione

Al termine della terza iterazione



Al termine della quarta iterazione



Se n è la taglia del vettore,
dopo n-1 iterazioni il vettore sarà ordinato

Ciascuna iterazione fa i confronti solo nella parte
di vettore non ordinata dalle iterazioni precedenti

Procedura ordina

Il codice C della procedura ordina è il seguente:

```
void ordina(int v[], int n)
 {
        int i, j;
        for (i =0; i < n-1; i++) {
            for (j=0; j<n-i-1; j ++){
                if v[j] >  v[j+1]
                    scambia(v,j);
            }
        }
 }
```

Procedura ordina

- Per tradurre da C ad assembler MIPS dobbiamo effettuare i passaggi seguenti:
  - Allocare i registri per le variabili del programma;
  - Scrivere il codice relativo al corpo della procedura;
  - Salvare il contenuto dei registri da eventuali modifiche effettuate dalla procedura.

Procedura ordina

- Step 1: Allocazione dei registri
  - Il registro $a0 è utilizzato per l'indirizzo base del vettore v
  - Il registro $a1 è utilizzato per il parametro n
  - Il registro $s0 è utilizzato per la variabile i
  - Il registro $s1 è utilizzato per la variabile j

Procedura ordina

- Problema: entrambe le procedure ordina e scambia hanno bisogno dei registri $a0 e $a1 per i loro parametri!

- Soluzione: copiare i parametri della procedura ordina in altri registri nella fase iniziale della procedura ordina, in modo da rendere disponibili i registri $a0 e $a1 per la procedura scambia

- Quindi all'inizio della procedura ordina copiamo il contenuto di $a0 e $a1 in altri due registri ($s2 e $s3):

```
move $s2, $a0 #copia il parametro $a0 in $s2 (salva $a0)
move $s3, $a1 #copia il parametro $a1 in $s3 (salva $a1)
```

- Chiaramente tutti i riferimenti ai registri $a0 e $a1 nella procedura ordina devono essere sostituiti con $s2 ed $s3

Procedura ordina

- Step 2: Scrivere il codice per il corpo della procedura
  - Consideriamo il ciclo for più esterno:

    for (i =0; i < n-1; i++) {…}
  - Innanzitutto inizializziamo il contatore i (contenuto in $s0)
    - move $s0, $zero #i=0
  - Usciamo dal ciclo quando i ≥ n-1
    - Calcoliamo n-1 e lo salviamo in un registro **NON** temporaneo

      addi $s4, $s3, -1 #$s4 contiene n-1
    - Effettuiamo il test per uscire dal ciclo

    CICLO1: slt $t0, $s0, $s4 #$t0 è zero se i ≥ n-1

    beq $t0, $zero, ESCI1 #salta a ESCI1 se i ≥ n-1
  - Prima di terminare il ciclo esterno (CICLO1), incrementiamo il contatore i e torniamo all'inizio di tale ciclo
    - ESCI2: addi $s0, $s0, 1 #i++
    - j CICLO1 #salta incondizionato a inizio del ciclo est.

Procedura ordina

- Step 2: Scrivere il codice per il corpo della procedura
  - Consideriamo il ciclo for più interno:

    for (j =0; j < n-i-1; j++) {…}
  - Innanzitutto inizializziamo il contatore j (contenuto in $s1)
    - move $s1, $zero #j=0
  - Usciamo dal ciclo quando j ≥ n-i-1
    - Calcoliamo n-i-1 e lo salviamo in un registro NON temporaneo

      sub $s5, $s4, $s0 #$s5 contiene n-i-1
    - Effettuiamo il test per uscire dal ciclo

    CICLO2: slt $t1, $s1, $s5 #$t1 è zero se j ≥ n-i-1

    beq $t1, $zero, ESCI2 #salta a ESCI2 se j ≥ n-i-1
  - Prima di terminare il ciclo interno (CICLO2), dopo aver verificato se va fatto o meno lo scambio, incrementiamo il contatore j e torniamo all'inizio di tale ciclo
    - ESCIIF: addi $s1, $s1, 1 #j++
    - j CICLO2 #salta incondizionato a inizio del ciclo int.

Procedura ordina

- Step 2: Scrivere il codice per il corpo della procedura
  - Scriviamo il corpo del ciclo for più interno:

    if v[j] > v[j+1] scambia(v,j);
  - Innanzitutto otteniamo l'indirizzo di v[j] in memoria

    sll $t2, $s1, 2 #$t2 contiene j*4

    add $t2, $s2, $t2 #$t2 contiene l'indirizzo di v[j]
  - Carichiamo v[j] in un registro

    lw $t3, 0($t2) #$t3 contiene v[j]
  - Poi carichiamo v[j+1] in registro

    lw $t4, 4($t2) #$t4 contiene v[j+1]
  - Effettuiamo il test v[j+1] ≥ v[j]

    slt $t5, $t4, $t3 #$t5 è zero se v[j+1] ≥ v[j]

    beq $t5, $zero, ESCIIF #salta a ESCIIF se v[j+1] ≥ v[j]
  - Se non c'è il salto, viene chiamata la procedura scambia

Procedura ordina

- Step 2: Scrivere il codice per il corpo della procedura
  - Ora possiamo passare i parametri alla procedura scambia

```
move $a0, $s2 #il primo parametro di scambia è v
move $a1, $s1 #il secondo parametro di scambia è j
jal SCAMBIA

ESCIIF: addi $s1, $s1, 1  #j++
        j CICLO2 #salta incondizionato a inizio del ciclo int.
ESCI2: addi $s0, $s0, 1  #i++
       j CICLO1 #salta incondizionato a inizio del ciclo est.

...
...
```

Procedura ordina

- Step 3: Salvare il contenuto dei registri
  - Poiché la procedura ordina utilizza 6 registri di tipo $s, dobbiamo salvare (nello stack) il loro contenuto all'inizio della procedura
  - Inoltre va salvato l'indirizzo di ritorno (contenuto nel registro $ra) per l'istruzione jr ra

```
ORDINA: addi $sp, $sp, -28  #crea spazio nello stack per 7 registri
        sw $ra, 24($sp) #salva $ra nello stack
        sw $s5, 20($sp) #salva $s5 nello stack
        sw $s4, 16($sp) #salva $s4 nello stack
        sw $s3, 12($sp) #salva $s3 nello stack
        sw $s2, 8($sp) #salva $s2 nello stack
        sw $s1, 4($sp) #salva $s1 nello stack
        sw $s0, 0($sp) #salva $s0 nello stack
```

Procedura ordina

- Step 3: Salvare il contenuto dei registri
  - La procedura ordina si conclude con il ripristino di tutti i registri precedentemente salvati e il salto all'indirizzo di ritorno

```
ESCI1:  lw $s0, 0($sp) #ripristina $s0 dallo stack
        lw $s1, 4($sp) #ripristina $s1 dallo stack
        lw $s2, 8($sp) #ripristina $s2 dallo stack
        lw $s3, 12($sp) #ripristina $s3 dallo stack
        lw $s4, 16($sp) #ripristina $s4 dallo stack
        lw $s5, 20($sp) #ripristina $s5 dallo stack
        lw $ra, 24($sp) #ripristina $ra dallo stack
        addi $sp, $sp, 28  #ripristina lo stack pointer
        jr $ra #salta all'indirizzo di ritorno del chiamante
```

Procedura ordina

- Step 3: Salvare il contenuto dei registri
  - La procedura ordina si conclude con il ripristino di tutti i registri precedentemente salvati e il salto all'indirizzo di ritorno

```
ESCI1:  lw $s0, 0($sp) #ripristina $s0 dallo stack
        lw $s1, 4($sp) #ripristina $s1 dallo stack
        lw $s2, 8($sp) #ripristina $s2 dallo stack
        lw $s3, 12($sp) #ripristina $s3 dallo stack
        lw $s4, 16($sp) #ripristina $s4 dallo stack
        lw $s5, 20($sp) #ripristina $s5 dallo stack
        lw $ra, 24($sp) #ripristina $ra dallo stack
        addi $sp, $sp, 28  #ripristina lo stack pointer
        jr $ra #salta all'indirizzo di ritorno del chiamante
```

Procedura completa

BubbleSort

Procedura scambia

```
SCAMBIA: sll $t1, $a1, 2 #$t1 contiene 4*k
         add $t1, $a0, $t1 #$ti contiene l'indirizzo di v[k]
         lw $t0, 0($t1) #$t0 contiene v[k]
         lw $t2, 4($t1) #$t2 contiene v[k+1]
         sw $t2, 0($t1) #registra v[k+1] in v[k]
         sw $t0, 4($t1) #registra v[k] in v[k+1]

         jr $ra #salta all'indirizzo di ritorno del chiamante (procedura ordina)
```

N.B.: Il salvataggio di $ra all'inizio della procedura non è necessario perché SCAMBIA è una procedura "foglia"

Procedura ordina

```
ORDINA: addi $sp, $sp, -28 #crea spazio nello stack per 7 registri
        sw $ra, 24($sp) #salva $ra nello stack
        sw $s5, 20($sp) #salva $s5 nello stack
        sw $s4, 16($sp) #salva $s4 nello stack
        sw $s3, 12($sp) #salva $s3 nello stack
        sw $s2, 8($sp) #salva $s2 nello stack
        sw $s1, 4($sp) #salva $s1 nello stack
        sw $s0, 0($sp) #salva $s0 nello stack

        move $s2, $a0 #copia il parametro $a0 in $s2 (salva indirizzo di v in $s2)
        move $s3, $a1 #copia il parametro $a1 in $s3 (salva n in $s3)

        move $s0, $zero #i=0
        addi $s4, $s3, -1 #$s4 contiene n-1 — NOTA: non può essere temporaneo
                          altrimenti #la funzione scambia potrebbe cambiarne il
                          contenuto (usiamo #$s3 al posto di $a1, che viene usato anche
                          da Scambia)
CICLO1: slt $t0, $s0, $s4 #$t0 e' zero se i ≥ n-1
        beq $t0, $zero, ESCI1 #salta a ESCI1 se i ≥ n-1
        move $s1, $zero #j=0
        sub $s5, $s4, $s0 #$s5 contiene n-i-1 — NOTA: non può essere temporaneo
                          #altrimenti la funzione scambia potrebbe cambiarne il
                          contenuto
        CICLO2: slt $t1, $s1, $s5 #$t1 e' zero se j ≥ n-i-1
                beq $t1, $zero, ESCI2 #salta a ESCI2 se j ≥ n-i-1
                sll $t2, $s1, 2 #$t2 contiene j*4
                add $t2, $s2, $t2 #$t2 indirizzo di v[j] (usiamo $s2 al posto di
                                  #$a0, che viene usato anche da Scambia)
                lw $t3, 0($t2) #$t3 contiene v[j]
                lw $t4, 4($t2) #$t4 contiene v[j+1]
                slt $t5, $t4, $t3 #$t5 e' zero se v[j+1] ≥ v[j]
                beq $t5, $zero, ESCIIF #salta a ESCIIF se v[j+1] ≥ v[j]

                move $a0, $s2 #il primo parametro di scambia è v
                move $a1, $s1 #il secondo parametro di scambia è j
                jal SCAMBIA #richiama la procedura SCAMBIA

                ESCIIF: addi $s1, $s1, 1 #j++
                        j CICLO2 #salto incondizionato verso il test del secondo ciclo
        ESCI2: addi $s0, $s0, 1 #i++
               j CICLO1 #salto incondizionato verso il test del primo ciclo
ESCI1: lw $s0, 0($sp) #ripristina $s0 dallo stack
lw $s1,4($sp) #ripristina $s1 dallo stack
lw $s2, 8($sp) #ripristina $s2 dallo stack
lw $s3, 12($sp) #ripristina $s3 dallo stack
lw $s4, 16($sp) #ripristina $s4 dallo stack
lw $s5, 20($sp) #ripristina $s5 dallo stack
lw $ra, 24($sp) #ripristina $ra dallo stack
addi $sp, $sp, 28 #ripristina lo stack pointer
jr $ra #salta all'indirizzo di ritorno del chiamante (programma main)
```