Ingegneria del Software

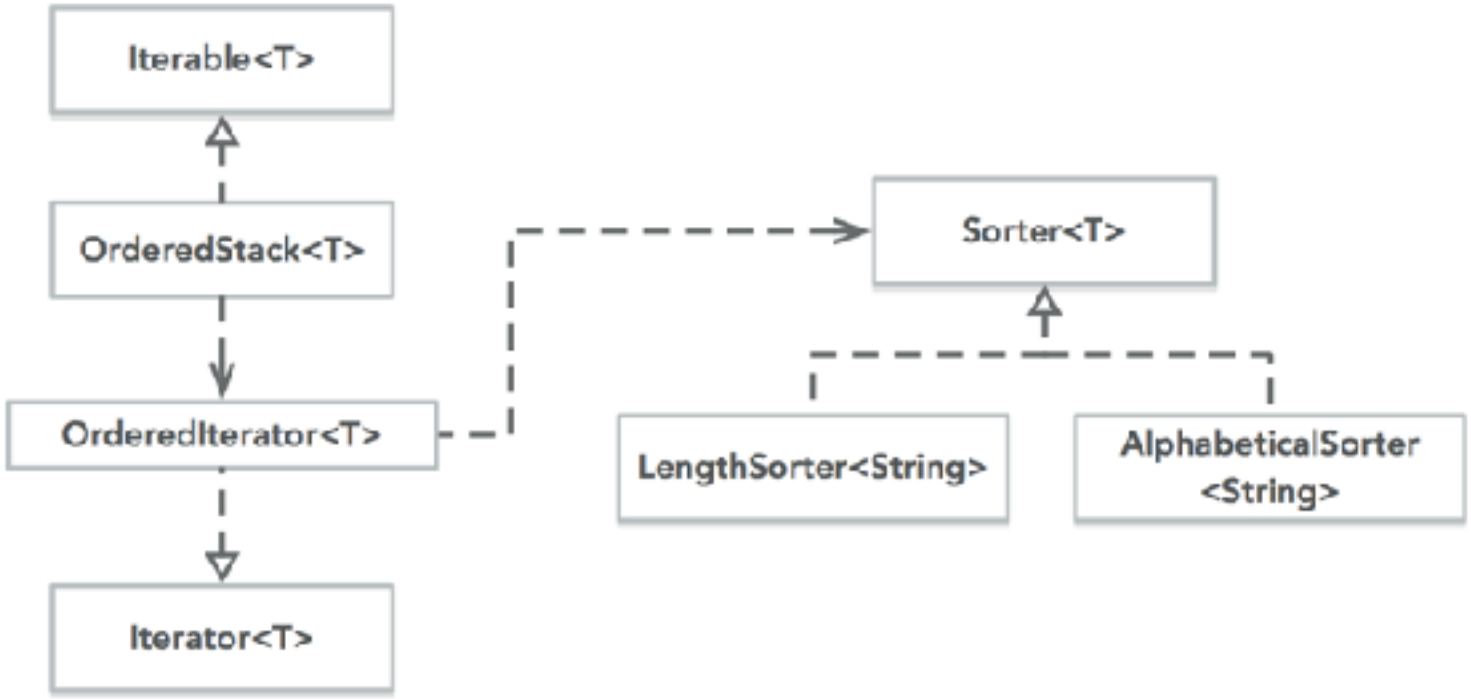
Esercitazione 7

OrderedStack

Definire in UML uno Stack generico che ritorni un iteratore che legga lo stack rispetto ad un criterio personalizzabile dall'utilizzatore. Ad esempio uno Stack di stringhe può essere letto in ordine di lunghezza delle stringhe o in ordine alfabetico.
Usare un pattern adeguato.

OrderedStack

Strategy Pattern



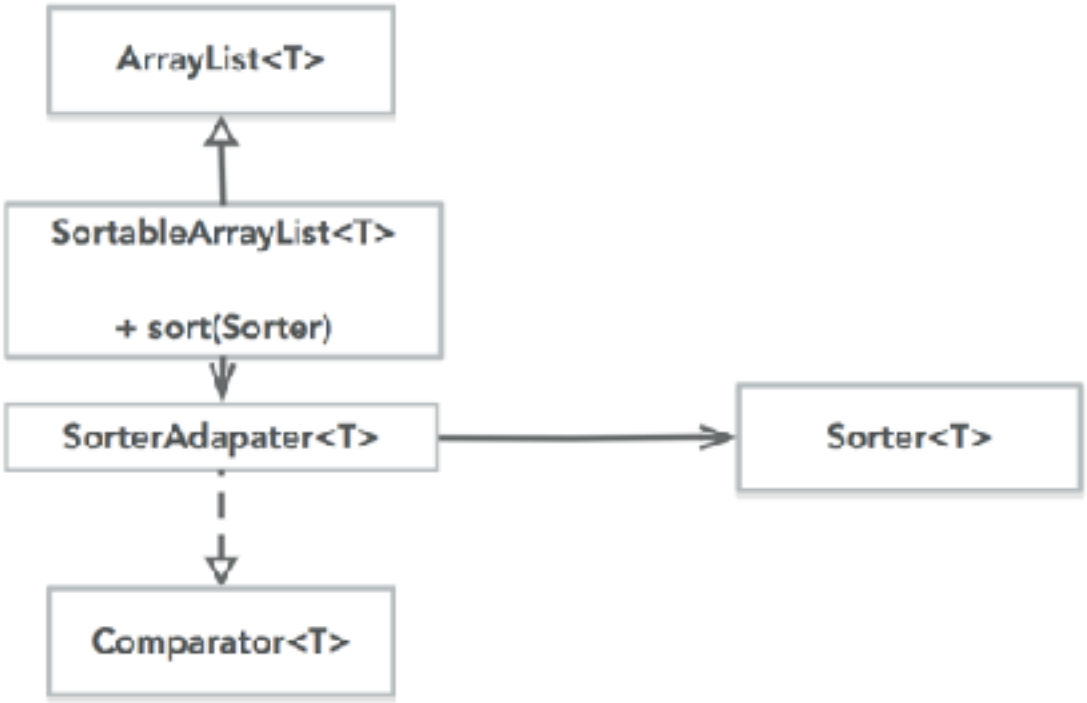
Sorter

E se volessimo usare un ArrayList con l'interfaccia Sorter per ordinarlo? ArrayList usa Comparator.

Che pattern possiamo utilizzare?

Sorter

Adapter Pattern



Weather Station

Implementare un sistema con il quale è possibile monitorare dati meteorologici. La classe WeatherData può essere aggiornata da un client con i dati aggiornati riguardo temperatura, pressione e umidità. La classe WeatherDisplay deve mostrare i dati non appena questi vengono aggiornati. Usare un pattern adeguato.

Specifica e Testing

Esercizio 1

```
public static boolean aggiungiStudente (Studente
s, Aula a);

class Aula {
    public boolean isIn (Studente s) {...}
}
```

Esercizio 1

```
/*@
@ requires s != null && a != null
@ ensures a.isIn(s) &&
@ (\forall Student t; !s.equals(t); a.isIn(t) <==> \old(a.isIn(t)))
@ \result <==> !\old(a.isIn(s))
@*/

public static boolean aggiungiStudente (Studente
s, Aula a);

class Aula {
    public boolean isIn (Studente s) {...}
}
```

Esercizio 2

```
class Indovina {
    static int count = 0;

    static int mistero(int x, String y, int[] z) {
        System.out.println(count);
        System.out.println(y + z[x]);
        x++;
        z[x] = z[x-1] + 1;
        count++;
        return x;
    }
}
```

Esercizio 2

```
/*@
@ requires z != null && y != null &&
@             0 <= x <= z.length-2
@ ensures \result == \old(x)+1 &&
@ count==\old(count)+1 &&
@ z[\result] == z[\result - 1] + 1;
@ assignable z[\result]
@*/
int mistero(int x, String y, int[] z);
```

Esercizio 3

Sia data la classe Interval

```
public class Interval{
    private float low, high;
    public float getLowerBound(){…}
    public float getUpperBound(){…}
    public static Interval getInterval(float[] times, float
timePoint)
}
```

Esercizio 3

- Definire in JML il contratto che rispetti le seguenti specifiche:
  - *times* non nullo
  - *times* con valori in ordine strettamente crescente
  - Restituisce un oggetto di tipo *Interval* che corrisponde a un intervallo temporale avente come estremi due punti contigui di times.
  - *timePoint* deve essere maggiore o uguale all'estremo minore e strettamente minore dell'estremo maggiore

Esercizio 3

```
/*@
@ assignable \nothing
@ requires times != null && times.length >= 2 &&
@ timePoint >= times[0] && timePoint < times[times.length -1]
@ && (\forall int i; 0<=i &&
i<times.length-1;times[i]<times[i+1])
@ ensures (\exists int i; 0<=i<times.length-1;
@  times[i] == \result.getLowerBound() &&
@  times[i+1] == \result.getUpperBound())) &&
@  \result.getLowerBound() <= timePoint &&
@  \result.getUpperBound() > timePoint );
@*/
Interval getInterval(float[] times, float timePoint);
```

Esercizio 4

public static boolean sottoStringa (char[] testo, char[] parola);

- Scrivere la specifica che renda l'operazione sensata, in modo che il metodo si comporti nel seguente modo:
  - testo = [abbcdddeff], parola = [bcddd] -> true
  - testo = [abbcdddeff], parola = [baaddd] -> false
- Dall'esempio, si deduce che se parola è contenuta in testo il risultato è true; false altrimenti
- In altre parole: se esiste una posizione in testo a partire dalla quale tutti i caratteri corrispondano nell'ordine a quelli di parola, per tutta la lunghezza di parola, il risultato è true (false altrimenti)

Esercizio 4

```
/*@@ requires testo!=null && parola!=null &&
@                          length(testo)>=length(parola);
@ assignable \nothing
@ ensures \result <==>
@     (\exists int i; 0<=i && i<testo.length-
parola.length;
@        (\forall int j; 0 <= j && j<parola.length;
testo[i+j] == parola[j])
@      );
@ */
boolean sottoStringa(char[] testo, char[] parola);
```

Esercizio 5

public static void highLowNums (int [] nums, int [] highs, int n);

- L'array nums contiene interi tutti diversi tra di loro
- L'array highs è lungo esattamente n
- Il metodo trova gli n numeri interi più grandi di nums e li inserisce in ordine decrescente nell'array highs
- L'array nums non viene modificato

Esercizio 5

```
/*@ assignable highs[*];
@
@ requires
@     nums != null && highs != null && highs.length == n
@     && nums.length >= n
@     && (\forall int i; 0<=i<nums.length-1;
@               !(\exists int j; i<j<nums.length; nums[i] ==
nums[j]));
@ ensures
@   (* highs contiene gli n numeri più grandi di nums *)
@   && (\forall int i; 0<=i<n-1; highs[i]>=highs[i+1]);
@*/
```

Esercizio 5

```
/*@ assignable highs[*];
@
@ requires
@     nums != null && highs != null && highs.length == n
@     && nums.length >= n
@     && (\forall int i; 0<=i<nums.length-1;
@               !(\exists int j; i<j<nums.length; nums[i] ==
nums[j]));
@ ensures
@    (* per ogni numero in highs esiste il corrispondente in
nums ed esistono in nums
          esattamente “posizione_in_highs” numeri maggiori
di esso *)
@   && (\forall int i; 0<=i<n-1; highs[i]>=highs[i+1]);
@*/
```

Esercizio 5

```
/*@ assignable highs[*];
@
@ requires
@     nums != null && highs != null && highs.length == n
@     && nums.length >= n
@     && (\forall int i; 0<=i<nums.length-1;
@                 !(\exists int j; i<j<nums.length; nums[i] ==
nums[j]));
@ ensures
@   (\forall int i; 0<=i<n;
@          (\exists int j; 0<=j<nums.length; highs[i] ==
nums[j])
@    && (\numof int k; 0<=k<nums.length; nums[k] >
highs[i]) == i) );
@   && (\forall int i; 0<=i<n-1; highs[i]>=highs[i+1]);
@*/
```

Esercizio 6

public static boolean isPermutation(int x[], int y[])

- True se y è una permutazione di x, false altrimenti
- Sotto l'ipotesi di assenza di duplicati nei due array

Esercizio 6

```
/*@ requires x != null && y != null &&
@ (\forall int i; 0 <= i < x.length -1;
@  (\forall int j; i < j < x.length; x[i] != x[j]))
@ && (* same for y *);
@
@ ensures (\result == true) <==> (x.length ==
y.length) &&
@    (\forall int i; 0 <= i < x.length;
@       (\exists int j; 0 <= j < y.length; x[i] == y[j]));
@*/
public static boolean isPermutation(int x[], int y[])
```

Esercizio 6bis

public static boolean isPermutation(int x[], int y[])

- True se y è una permutazione di x, false altrimenti
- Rimuovere l'ipotesi di assenza di duplicati nei due array

Esercizio 6bis

```
/*@ requires
@ x != null && y != null;
@ ensures
@(\result == true) <==> (x.length == y.length) &&
@(\forall int i; 0 <= i < x.length;
@    (\numof int j; 0 <= j < x.length; x[i] == x[j]))
@         ==
@    (\numof int k; 0 <= k < y.length; x[i] == y[k]));
@*/
public static boolean isPermutation(int x[], int y[])
```

Esercizio 7

Si consideri il metodo:

```
//@ requires in >= 0;
//@ ensures Math.abs(\result*\result -
in) < 0.0001;
public static float sqrt(float in)
```

Esercizio 7

Come possiamo creare sistemi robusti?

Il comportamento è predicibile anche nei casi in cui i patti non sono stati rispettati

Si possono:

- togliere le precondizioni

  ```
  //@ requires true
  ```

- evidenziare i casi problematici

  ```
  //@ ensures in >= 0 && (* altre postcondizioni *)
  ```

- sollevando le opportune eccezioni

  ```
  //@ signals (NegativeException e) in < 0;
  ```

Esercizio 7

Specifica con pre e post condizioni

```
//@ requires in >= 0;
//@ ensures Math.abs(\result*\result-in)<0.0001;
public static float sqrt(float in)
```

Specifica totale

```
//@ requires true;
//@ ensures in >= 0 && Math.abs(\result*\result-
in)<0.0001;
//@ signals(NegativeException ne) in < 0;
public static float sqrt(float in)
```

Esercizio 8

Rendere totali le specifiche di:
public static boolean sottoStringa (char[] testo, char[] parola);

- Scrivere la specifica che renda l'operazione sensata, in modo che il metodo si comporti nel seguente modo:
  - testo = [abbcdddeff], parola = [bcddd] -> true
  - testo = [abbcdddeff], parola = [baaddd] -> false
- Dall'esempio, si deduce che se parola è contenuto in testo il risultato è true; false altrimenti
- In altre parole: se esiste una posizione in testo a partire dalla quale tutti i caratteri corrispondano nell'ordine a quelli di parola, per tutta la lunghezza di parola, il risultato è true (false altrimenti)

Parziali:

```
/*@@ requires testo!=null && parola!=null &&
@                          length(testo)>=length(parola);
@ assignable \nothing
@ ensures \result <==>
@     (\exists int i; 0<=i && i<testo.length-
parola.length;
@        (\forall int j; 0 <= j && j<parola.length;
testo[i+j] == parola[j])
@      );
@ */
boolean sottoStringa(char[] testo, char[]
parola);
```

Totali:

```
/*@@ requires true
@ assignable \nothing
@ ensures testo != null && parola!= null  &&
@                       length(testo)>=length(parola) &&
@  \result <==>
@     (\exists int i; 0<=i && i<testo.length-parola.length;
@        (\forall int j; 0 <= j && j<parola.length; testo[i+j] ==
parola[j])
@      );
@ signals (NullPointerException npe) testo==null  | |
parola==null;
@ signals (InvertedDimensionException ide)
length(testo)<length(parola);
@ */
boolean sottoStringa(char[] testo, char[] parola);
```

Esercizio 9

Si deve progettare una classe Clock, che realizza oggetti che rappresentano ora, minuto e secondo. La classe offre le operazioni click(x), che fa avanzare il tempo di x secondi, e le operazioni getter getHour, getMin e getSec, che restituiscono l'ora, il minuto e il secondo di un oggetto Clock. La classe implementa l'interfaccia Comparable, di cui si riporta la definizione:

```
public interface Comparable {
  //@ensures (* if this < arg then \result ==−1
  //@                else if this == arg then \results == 0,
  //@                      else \result == 1 *);
  public int compareTo(T arg);
}
```

- Fornire la specifica JML del metodo click. Il metodo non solleva eccezioni.
- Definire una rappresentazione (rep) per la classe. Specificare l'eventuale rep invariant e fornire un'implementazione della funzione di astrazione, come implementazione dell'operazione toString.
- Definire un'implementazione del metodo compareTo

```
class Clock implements Comparable<Clock>{
        private int seconds;
        //@private invariant
        //@ seconds >= 0 && seconds < 3600*24 &&
        //@ seconds == getHour()*3600 + getMin()*60 + getSec();

        //@ public invariant
        //@ 0 <= (getSec() + getMin()*60 + getHour()*3600)<24*3600

        //@ requires x>=0
        //@ ensures (getSec() + getMin()*60 + getHour()*3600)==
        //@ \old(getSec() + getMin()*60 + getHour()*3600 +x)%(24*3600)
        void click(int x);

        int getHour();  int getMin();    int getSec();
```

```
    @Override
    public String toString() {
            int hours = seconds / 3600;
            int minutes = (seconds % 3600) / 60;
            int sec = seconds % 60;
            return "H:" + hours + ", M:" + minutes + ", S:" + sec);
    }

    public int compareTo(Clock arg){
            if(seconds < arg.seconds) return -1;
            else if (seconds == arg.seconds) return 0;
            else return 1;
    }
}
```

Esercizio 10

```
public class Math {
  //@ requires value >= 0;
  //@ ensures Math.abs(\result * \result - value) < 0.01;
  public double sqrt(double value) { … }
}
```

- Mettiamoci nell'ottica degli utenti:
  - Non useremo mai un parametro attuale negativo, in quanto le precondizioni del metodo sarebbero violate.
  - Se le precondizioni non sono soddisfatte il metodo è infatti libero di fare quello che vuole (il contratto è nullo)
  - Se stiamo rispettando i patti (value >= 0), ci viene garantito che il risultato è la radice quadrata

```
public class SafeMath extends Math {
  //@ also
  //@  requires value < 0;
  //@  ensures  \result = -1;
  public double sqrt(double value)
{…}
}
```

La classe SafeMath è un’estensione valida di Math?

La classe SafeMath è un'estensione valida di Math?

- Regola delle signature: sì, sqrt viene sovrascritta
- Regola dei metodi: le chiamate a sqrt di SafeMath si comportano come le chiamate a sqrt di Math
- Regola delle proprietà: tutti gli invarianti pubblici sono rispettati

Mettiamoci nei panni di un utente di Math: sarà sorpreso se gli viene passata un'istanza di tipo SafeMath?

- Per quanto ne sa lui, non è possibile calcolare la radice quadrata di un numero negativo, quindi non eserciterà mai le nuove funzionalità… di fatto per lui Math e SafeMath sono equivalenti

Ricaviamo il contratto complessivo di SafeMath.sqrt

Le precondizioni si ottengono mettendo in or quelle di SafeMath e quelle di Math:

```
//@ requires (value >= 0) || (value < 0)
```

Le postcondizioni si ottengono con la seguente formula:
(presup => postsup) && (presott => postsott)

```
//@ ensures (value >= 0 => Math.abs(...))
//@ && (value < 0  => \result = -1 )
```

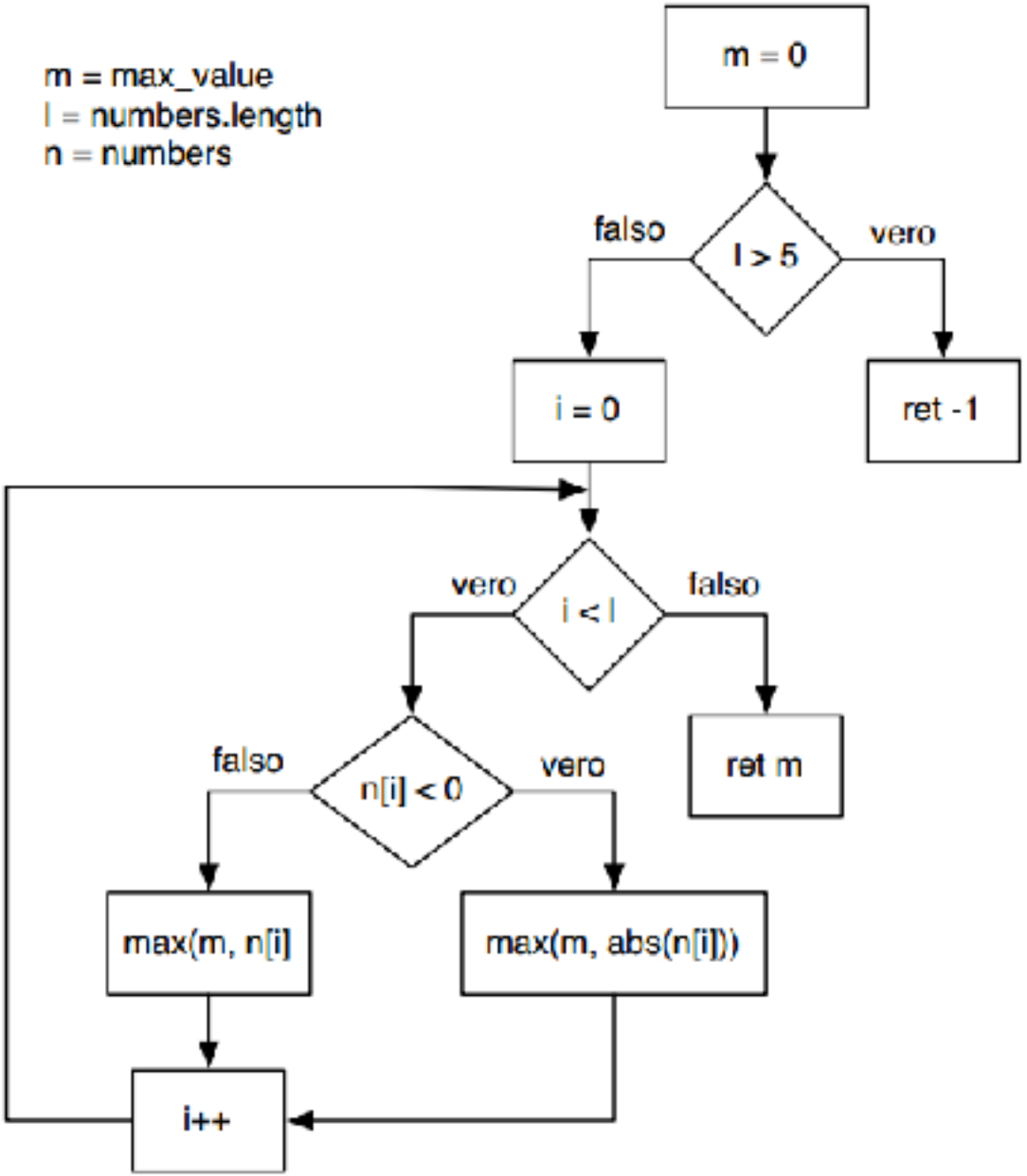
Esercizio 11

Il seguente metodo Java prende in ingresso un array di al più 5 interi e restituisce l'elemento più grande in valore assoluto, oppure 0 se l'array fosse vuoto oppure -1 se troppo grande.

```
public int max(int[] numbers) {
    int i, max_value = 0;
    if (numbers.length > 5) return -1;
    for (i = 0; i<numbers.length; i++) {
        if (numbers[i] < 0)
            max_value = Math.max(max_value,Math.abs(numbers[i]));
        else max_value = Math.max(max_value, numbers[i]);
    }
    return max_value;
}
```

- Disegnare il diagramma del flusso di controllo.

Soluzione:


m = max_value
l = numbers.length
n = numbers
m = 0
falso
l > 5
vero
i = 0
ret -1
vero
i < l
falso
falso
n[i] < 0
vero
ret m
max(m, n[i]
max(m, abs(n[i]))
i++

- Il metodo è collaudato con i casi di test seguenti (input; valore restituito):
  - T1: {0,0,0,0,0}; 0;
  - T2: {1,2,3,4,5}; 5;
  - T3: {-1,-2,-3,-4,-5}; 5;
  - T4: {1,2,3,4,5,6}; -1;
  - T5: {-10,10,3,5,-6}; 10;
  - T6: {}; -1;

1. Definire le percentuali di copertura delle istruzioni (*statement*) e delle decisioni (*branch*) ottenute eseguendo ogni test separatamente.

2. Quale test darebbe errore?

Soluzione:

T	statement	branch/archi
T1	8	8
T2	8	8
T3	8	8
T4	3	2
T5	9	10
T6	5	4

T6: restiuisce 0 e non -1;