Università degli Studi di Palermo

Corso di Laurea
Ingegneria Gestionale

Lezione 9

Le Istruzioni di Controllo (parte 2)

Ing. Massimo Cossentino

Sistemi Informativi Aziendali

a.a. 2008/2009

Utilizzare le espressioni booleane
Il tipo `boolean`

- George Boole (1815-1864): pioniere nello studio della logica
- Il valore dell'espressione `amount < 1000` è `true` o `false`.
- Il tipo boolean ha due valori: `true` e `false`

Utilizzare le espressioni booleane
Metodi predicativi

- Un metodo predicativo restituisce un valore booleano

```
public boolean isOverdrawn()
{
   return balance < 0;
}
```

Il valore restituito dal metodo può essere utilizzato come condizione di un enunciato `if`:

```
if (harrysChecking.isOverdrawn()) . . .
```

Utilizzare le espressioni booleane
Metodi predicativi

- Nella classe `Character` sono presenti parecchi utili metodi predicativi statici:

```
isDigit
isLetter
isUpperCase
isLowerCase
```

- che consentono di verificare i caratteri

```
if (Character.isUpperCase(ch)) . . .
```

- La classe `Scanner` ha metodi predicativi utili per verificare se una successiva richiesta di dati in ingresso andrà a buon fine: `hasNextInt` e `hasNextDouble()`

```
if (in.hasNextInt()) n = in.nextInt();
```

Utilizzare le espressioni booleane
Gli operatori booleani

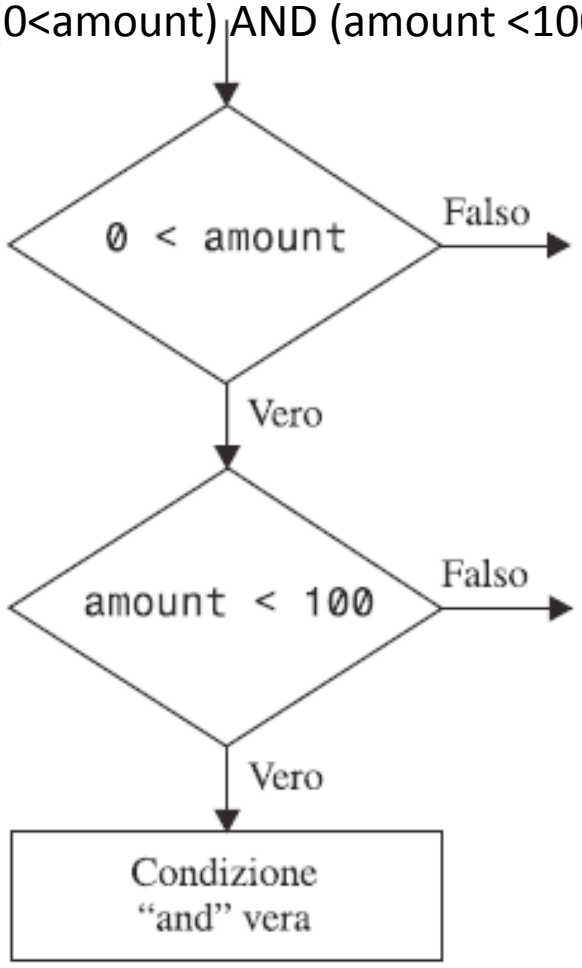
- `&&` and
- `||` or
- `!` not

```
if (0 < amount && amount < 1000) . . .
```

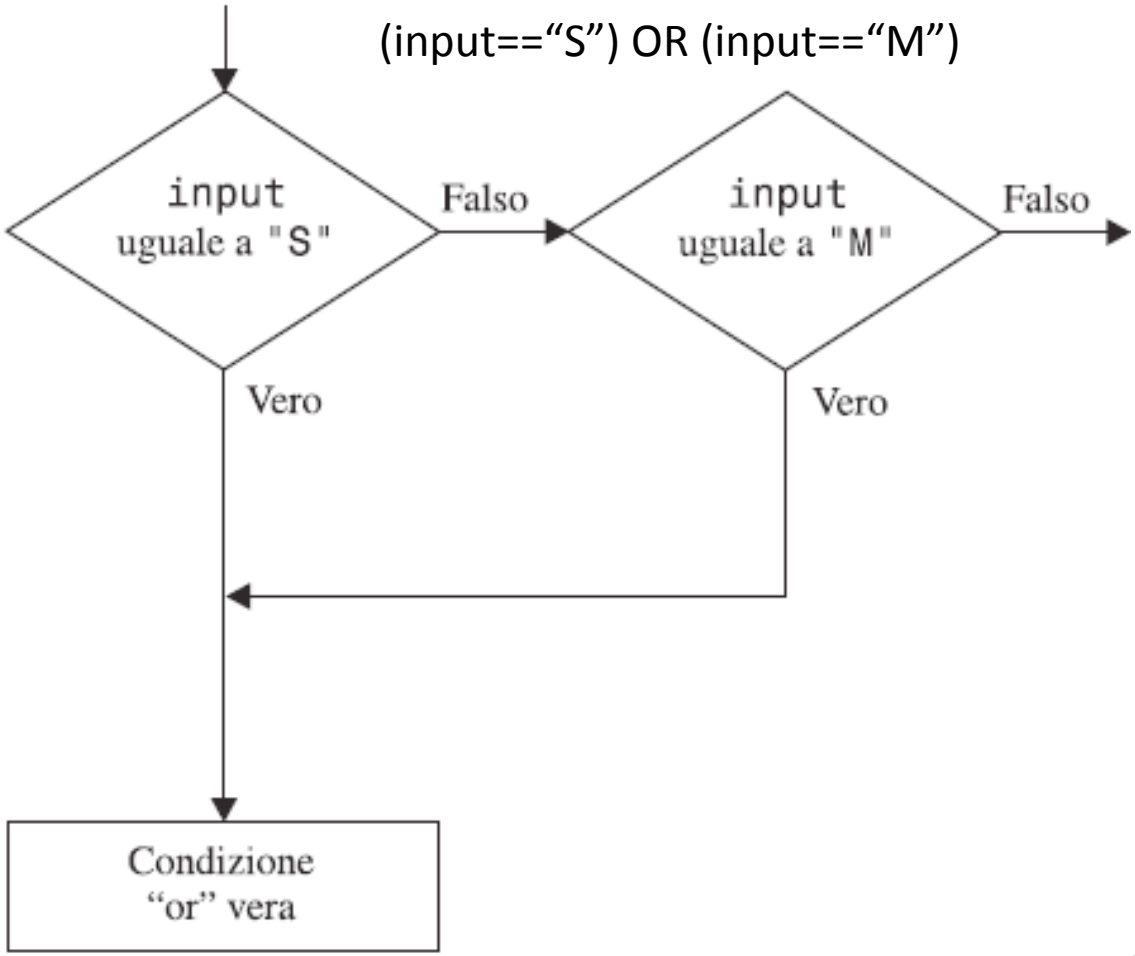
```
if (input.equals("S") || input.equals("M")) . . .
```

Gli operatori booleani

(0<amount) AND (amount <100)



(input==“S”) OR (input==“M”)



Riepilogo delle tre operazioni logiche

A	B	A && B
vero	vero	vero
vero	falso	falso
falso	*qualsiasi valore*	falso

A	B	A || B
vero	*qualsiasi valore*	vero
falso	vero	vero
falso	falso	falso

A	!A
vero	falso
falso	vero

Utilizzare variabili booleane

```
private boolean married;
```

- Il risultato della verifica di una condizione può essere memorizzato in una variabile booleana.

```
if (married) . . . else . . .
if (!married) . . .
```

- Le variabili booleane sono dette *flag*, perché hanno solo due stati, "su" o "giù".

Utilizzare variabili booleane

- Valutare attentamente i nomi da assegnare alle variabili booleane dà i risultati migliori.
  - Da non fare

```
if (married == true) . . . // Da non fare
```

  - Usare una condizione semplice

```
if (married) . . .
```

Operatori di assegnamento composti

- Gli operatori di assegnamento composti permettono di abbreviare le espressioni di assegnamento.

variabile = variabile operatore espressione

variabile operatore = espressione

esempio:

c = c +3;

c += 3;

Operatori di assegnamento composti

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

Operatori di incremento e decremento

- L' assegnazione di un valore ad una variabile non è un'uguaglianza matematica, cioè il segno uguale che si usa in un'assegnazione non sta ad indicare che il valore a sinistra è uguale a quello a destra.
- L'operatore di assegnazione è una istruzione per fare qualcosa
- l'istruzione

  x = x +1;

  ha senso solo tenuto conto di quanto detto sopra.

Operatori di incremento e decremento

- Operatori unari:
  - L'operatore di incremento unario (++) aggiunge 1 al suo operando
  - L'operatore di incremento unario (--) sottrae 1 al suo operando
  - L'operatore di incremento (decremento) prefisso
    - cambia il valore del suo operando e poi usa il valore del nuovo operando nella espressione nella quale l'operazione appare
  - L'operatore di incremento (decremento) postfisso
    - usa il valora corrente del suo operando nell'espressione nella quale l'operazione appare, poi cambia il valore dell'operando

Operatori di incremento e decremento

Operator	Called	Sample expression	Explanation
++	prefix increment	++a	Increment a by 1, then use the new value of a in the expression in which a resides.
++	postfix increment	a++	Use the current value of a in the expression in which a resides, then increment a by 1.
--	prefix decrement	--b	Decrement b by 1, then use the new value of b in the expression in which b resides.
--	postfix decrement	b--	Use the current value of b in the expression in which b resides, then decrement b by 1.

Università degli Studi di Palermo

Cicli While

Variabili di controllo nei cicli while

Calcolare la crescita di un investimento

- Saldo iniziale $10,000, tasso di interesse annuo 5%

Anno	Saldo
0	$10,000.00
1	$10,500.00
2	$11,025.00
3	$11,576.25
4	$12,155.06
5	$12,762.82

Calcolare la crescita di un investimento

- Quando il conto raggiunge un particolare saldo?

```
while (balance < targetBalance)
{
   years++;
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

File Investment.java

```
01: /**
02:    Una classe per controllare la crescita di un investimento
03:    che accumula interessi a un tasso annuale fisso.
04: */
05: public class Investment
06: {
07:    /**
08:       Costruisce un oggetto Investment con un saldo iniziale
09:       e un tasso di interesse.
10:       @param aBalance il saldo iniziale
11:       @param aRate il tasso di interesse percentuale
12:    */
13:    public Investment(double aBalance, double aRate)
14:    {
15:       balance = aBalance;
16:       rate = aRate;
17:       years = 0;
18:    }
19:
```

Segue

File Investment.java

```
20:  /**
21:     Continua ad accumulare interessi finché il saldo
22:           non raggiunge un valore desiderato.
23:     @param targetBalance the desired balance
24:  */
25:  public void waitForBalance(double targetBalance)
26:  {
27:     while (balance < targetBalance)
28:     {
29:        years++;
30:        double interest = balance * rate / 100;
31:        balance = balance + interest;
32:     }
33:  }
34:
35:  /**
36:           Restituisce il saldo attuale dell'investimento.
37:     @return il saldo attuale
38:  */
```

File Investment.java

```
39:   public double getBalance()
40:   {
41:      return balance;
42:   }
43:
44:   /**
45:      Restituisce il numero di anni per i quali
46:      l'investimento ha accumulato interessi.
47:      @return il numero di anni trascorsi dall'inizio
            dell'investimento
48:   */
49:   public int getYears()
50:   {
51:      return years;
52:   }
53:
54:   private double balance;
55:   private double rate;
56:   private int years;
57: }
```

File InvestmentRunner.java

```
01: /**
02:    Questo programma calcola quanto tempo occorre per il
03:    raddoppio di un investimento.
04: */
05: public class InvestmentRunner
06: {
07:    public static void main(String[] args)
08:    {
09:       final double INITIAL_BALANCE = 10000;
10:       final double RATE = 5;
11:       Investment invest
             = new Investment(INITIAL_BALANCE, RATE);
12:       invest.waitForBalance(2 * INITIAL_BALANCE);
13:       int years = invest.getYears();
14:       System.out.println("The investment doubled after "
15:             + years + " years");
16:    }
17: }
```

Segue

File `InvestmentTester.java`

Visualizza

```
The investment doubled after 15 years
```

Errori comuni: cicli infiniti

```
int years = 0;
while (years < 20)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
   // manca years++
}
```

- Un ciclo che viene eseguito ininterrottamente e che si può fermare solo annullando l'esecuzione del programma o riavviando il computer.

```
int years = 20;
while (years > 0)
{
   years++; // Avrebbe dovuto essere years--
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

Università degli Studi di Palermo

Cicli Do

Cicli do

- Eseguire un ciclo almeno una volta:

```
do
  enunciato
while (condizione);
```

- Esempio: verificare l'input

```
double value;
do
{
  System.out.print("Please enter a positive number: ");
  value = in.nextDouble();
}
while (value <= 0);
```

Cicli do

- Alternativa, utilizzare una variabile di controllo booleana:

```
boolean done = false;
while (!done)
{
   System.out.print("Please enter a positive number: ");
   value = in.nextDouble();
   if (value > 0) done = true;
}
```

Università degli Studi di Palermo

Cicli For

Ciclo `for`

-

```
for (inizializzazione; condizione; aggiornamento)
  enunciato
```

Esempio:

```
for (int i = 1; i <= n; i++)
{
   double interest = balance * rate / 100;
   balance = balance + interest;
}
```

Ciclo `for`

- Si usa un ciclo `for` quando una variabile viene modificata da un valore iniziale a un valore finale con un incremento o decremento costante.
- Equivale a

```
inizializzazione;
while (condizione)
{ enunciato; aggiornamento; }
```

- Altri esempi:

```
for (years = n; years > 0; years--) . . .
```

```
for (x = -10; x <= 10; x = x + 0.5) . . .
```

Esempi di uso dell'istruzione FOR

- Cambiare la variabile di controllo nei cicli FOR:
  - Cambiare la variabile di controllo da 1 a 100 con incrementi di 1
    - `for ( int i = 1; i <= 100; i++ )`
  - Cambiare la variabile di controllo da 100 a 1 con incrementi di –1
    - `for ( int i = 100; i >= 1; i-- )`
  - Cambiare la variabile di controllo da 7 a 77 con incrementi di 7
    - `for ( int i = 7; i <= 77; i += 7 )`
  - Cambiare la variabile di controllo da 20 a 2 con decrementi di 2
    - `for ( int i = 20; i >= 2; i -= 2 )`
  - Cambiare la variabile di controllo nella sequenza: 2, 5, 8, 11, 14, 17, 20
    - `for ( int i = 2; i <= 20; i += 3 )`
  - Cambiare la variabile di controllo nella sequenza: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0
    - `for ( int i = 99; i >= 0; i -= 11 )`

File Investment.java

```
01: /**
02:    Una classe per controllare la crescita di un investimento
03:    che accumula interessi a un tasso annuale fisso.
04: */
05: public class Investment
06: {
07:    /**
08:       Costruisce un oggetto Investment con un saldo iniziale
09:       e un tasso di interesse.
10:       @param aBalance il saldo iniziale
11:       @param aRate il tasso d'interesse percentuale
12:    */
13:    public Investment(double aBalance, double aRate)
14:    {
15:       balance = aBalance;
16:       rate = aRate;
17:       years = 0;
18:    }
```

Segue

File Investment.java

```
19:
20:   /**
21:      Continua ad accumulare interessi finché il saldo
22:      non raggiunge un valore desiderato.
23:      @param targetBalance il saldo desiderato
24:   */
25:   public void waitForBalance(double targetBalance)
26:   {
27:      while (balance < targetBalance)
28:      {
29:         years++;
30:         double interest = balance * rate / 100;
31:         balance = balance + interest;
32:      }
33:   }
34:
```

Segue

File Investment.java

```
35:  /**
36:     Continua ad accumulare interessi per un dato numero di anni.
37:     @param n il numero di anni
38:  */
39:  public void waitYears(int n)
40:  {
41:     for (int i = 1; i <= n; i++)
42:     {
43:        double interest = balance * rate / 100;
44:        balance = balance + interest;
45:     }
46:     years = years + n;
47:  }
48:
49:  /**
50:     Restituisce il saldo attuale dell'investimento.
51:     @return il saldo attuale
52:  */
```


Segue

(parte 2)

File `Investment.java`

```
53:   public double getBalance()
54:   {
55:      return balance;
56:   }
57:
58:   /**
59:      Restituisce il numero di anni per i quali l'investimento
60:      ha accumulato interessi.
61:      @return il numero di anni dall'inizio dell'investimento
62:   */
63:   public int getYears()
64:   {
65:      return years;
66:   }
67:
```

Segue

File `Investment.java`

```
68:    private double balance;
69:    private double rate;
70:    private int years;
71: }
```

File InvestmentRunner.java

```
01: /**
02:    Questo programma calcola di quanto aumenta un investimento
03:    in un dato numero di anni.
04: */
05: public class InvestmentTester
06: {
07:    public static void main(String[] args)
08:    {
09:       final double INITIAL_BALANCE = 10000;
10:       final double RATE = 5;
11:       final int YEARS = 20;
12:       Investment invest = new Investment(INITIAL_BALANCE, RATE);
13:       invest.waitYears(YEARS);
14:       double balance = invest.getBalance();
15:       System.out.printf("The balance after %d years is %.2f\n",
16:             YEARS, balance);
17:    }
18: }
```

Segue

File `Investment.java`

Visualizza

```
The balance after 20 years is 26532.98
```

Errori comuni: punto e virgola

- Punto e virgola nella posizione sbagliata

```
sum = 0;
for (i = 1; i <= 10; i++);
   sum = sum + i;
System.out.println(sum);
```

Cicli annidati

- I cicli possono essere annidati. Un esempio tipico di ciclo annidato è quello che visualizza tabelle con righe e colonne.
- Creare una forma triangolare

```
[]
[][]
[][][]
[][][][]
```

- Stampare un certo numero di righe

```
for (int i = 1; i <= n; i++)
{
  // costruisci una riga del triangolo
  ...
}
```

Cicli annidati

- Usare un altro ciclo per mettere insieme le parentesi quadre [ ] per la riga

```
for (int j = 1; j <= i; j++)
  r = r + "[]";
```

Cicli annidati

- Se si mettono assieme entrambi i cicli si ottengono due cicli annidati

```
public class Costruisci_figure
{
    int width=6;
     public void triangolo()
     {
          String r;
          for (int i=1;i<=width;i++)
          {
               r="";
               // costruisci una riga del triangolo
               for (int j = 1; j <= i; j++)
                    r = "[]"+r;
                    System.out.println(r);
          }
     }
}
```

cicli for non annidati

- Modificare la classe precedente per disegnare un quadrato
- Algoritmo:

```
//Costruisci la riga del quadrato
r="";
Per j da 1 a dimensione_quadrato
   r=r+"[]";
Per i da 1 a dimensione_quadrato
   Stampa riga
```

- Si tratta di due cicli NON annidati !!

Cicli for non annidati

```
public class Costruisci_figure
{
    int width=6;
     public void triangolo()
     {…}
     public void quadrato()
     {
         String r="";
         //costruisci una riga del quadrato
         for (int j = 1; j <= width; j++)
             r = "[]"+r;
             for (int i=1;i<=width;i++)
                 System.out.println(r);
     }
}
```

File Triangle.java

```
01: /**
02:    Questa classe descrive oggetti triangolari che possono
03:    essere visualizzati in questo modo:
04:    []
05:    [][]
06:    [][][]
07: */
08: public class Triangle
09: {
10:    /**
11:       Construisce un triangolo.
12:       @param aWidth il numero di [] nell'ultima riga
          del triangolo.
13:    */
14:    public Triangle(int aWidth)
15:    {
16:       width = aWidth;
17:    }
18:
```

Segue

File `Triangle.java`

```
19:  /**
20:     Calcola una stringa che rappresenta il triangolo.
21:     @return una stringa composta di caratteri []
        e ritorni a capo
22:  */
23:  public String toString()
24:  {
25:     String r = "";
26:     for (int i = 1; i <= width; i++)
27:     {
28:        // costruisci una riga del triangolo
29:        for (int j = 1; j <= i; j++)
30:           r = r + "[]";
31:        r = r + "\n";
32:     }
33:     return r;
34:  }
35:
36:  private int width;
37: }
```

File `TriangleRunner.java`

```
01: /**
02:    Questo programma visualizza due triangoli.
03: */
04: public class TriangleRunner
05: {
06:    public static void main(String[] args)
07:    {
08:       Triangle small = new Triangle(3);
09:       System.out.println(small.toString());
10:
11:       Triangle large = new Triangle(15);
12:       System.out.println(large.toString());
13:    }
14: }
```

File `TriangleRunner.java`

Visualizza

```
[]
[][]
[][][]

[]
[][]
[][][]
[][][][]
[][][][][]
[][][][][][]
[][][][][][][]
[][][][][][][][]
[][][][][][][][][]
[][][][][][][][][][]
[][][][][][][][][][][]
[][][][][][][][][][][][]
[][][][][][][][][][][][][]
[][][][][][][][][][][][][][]
[][][][][][][][][][][][][][][]
```

Utilizzo di valori sentinella

- Metodo molto utilizzato per segnalare la fine di un insieme di dati
- Non è una buona idea scegliere 0 oppure -1 come sentinelle: questi valori possono essere validi come dati del problema. E' meglio utilizzare la lettera Q.

```
System.out.print("Enter value, Q to quit: ");
String input = in.next();
if (input.equalsIgnoreCase("Q"))
   Abbiamo finito
else
{
   double x = Double.parseDouble(input);
   . . .
}
```

Condizione di uscita calcolata nel ciclo

- A volte la condizione di terminazione di un ciclo può essere valutata soltanto nel mezzo di un ciclo, che può essere controllato mediante una variabile booleana.

```
boolean done = false;
while (!done)
{
   Visualizza una richiesta di dati
   String input = leggi un dato
   if (i dati sono terminati)
      done = true;
   else
   {
      // elabora il dato letto
   }
}
```

File DataAnalyzer.java

```
01: import java.util.Scanner;
02:
03: /**
04:   Questo programma calcola il valore medio e il valore massimo
05:   di un insieme di dati forniti in ingresso.
06: */
07: public class DataAnalyzer
08: {
09:    public static void main(String[] args)
10:    {
11:       Scanner in = new Scanner(System.in);
12:       DataSet data = new DataSet();
13:
14:       boolean done = false;
15:       while (!done)
16:       {
```

Segue

File DataAnalyzer.java

```
17:       System.out.print("Enter value, Q to quit: ");
18:       String input = in.next();
19:       if (input.equalsIgnoreCase("Q"))
20:          done = true;
21:       else
22:       {
23:          double x = Double.parseDouble(input);
24:          data.add(x);
25:       }
26:    }
27:
28:    System.out.println("Average = " + data.getAverage());
29:    System.out.println("Maximum = " + data.getMaximum());
30:  }
31: }
```

File DataSet.java

```
01: /**
02:    Calcola informazioni relative a un insieme di dati.
03: */
04: public class DataSet
05: {
06:    /**
07:       Costruisce un insieme di dati vuoto.
08:    */
09:    public DataSet()
10:    {
11:       sum = 0;
12:       count = 0;
13:       maximum = 0;
14:    }
15:
16:    /**
17:       Aggiunge un valore all'insieme dei dati
18:       @param x un valore
19:    */
```

Segue

File DataSet.java

```
20:   public void add(double x)
21:   {
22:      sum = sum + x;
23:      if (count == 0 || maximum < x) maximum = x;
24:      count++;
25:   }
26:
27:   /**
28:      Restituisce la media dei valori inseriti.
29:      @return la media, o 0 se non ci sono dati
30:   */
31:   public double getAverage()
32:   {
33:      if (count == 0) return 0;
34:      else return sum / count;
35:   }
36:
```

Segue

File DataSet.java

```
37:   /**
38:      Restituisce il valore massimo tra i valori inseriti.
39:      @return il massimo, o 0 se non ci sono dati
40:   */
41:   public double getMaximum()
42:   {
43:      return maximum;
44:   }
45:
46:   private double sum;
47:   private double maximum;
48:   private int count;
49: }
```

File `DataSet.java`

Visualizza

```
Enter value, Q to quit: 10
Enter value, Q to quit: 0
Enter value, Q to quit: -1
Enter value, Q to quit: Q
Average = 3.0
Maximum = 10.0
```