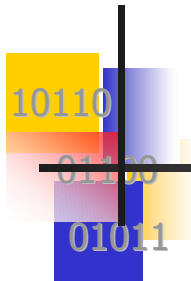


## Algebra di Boole: Concetti di base



# Algebra di Boole

- E' un'algebra basata su tre operazioni logiche
  - **OR**
  - **AND**
  - **NOT**
- Ed operandi che possono avere due soli valori:
  - **vero** (1)
  - **falso** (0).
- Tramite questa algebra di possono comporre **espressioni logiche** che possono essere **vere** o **false**.
- Ad esempio: **(X AND Y) OR Z** oppure **(NOT A) OR B**

10110

01100

01011

# Algebra di Boole : Tavole di verità

Le operazioni dell'algebra di Boole sono definite tramite le **Tavole di verità**.

<b>AND</b>	<i>0</i>	<i>1</i>
<i>0</i>	0	0
<i>1</i>	0	1

<b>OR</b>	<i>0</i>	<i>1</i>
<i>0</i>	0	1
<i>1</i>	1	1

<b>NOT</b>	<i>0</i>	<i>1</i>
<i>1</i>	1	0

10110

01100

01011

# Algebra di Boole

- **Esempi:**
- vero AND vero = vero  $\iff$  1 AND 1 = 1
- falso AND vero = falso  $\iff$  0 AND 1 = 0
- vero OR vero = vero  $\iff$  1 OR 1 = 1
- falso OR vero = vero  $\iff$  0 OR 1 = 1
- falso OR falso = falso  $\iff$  0 OR 0 = 0
  
- NOT vero = falso  $\iff$  NOT 1 = 0
- NOT falso = vero  $\iff$  NOT 0 = 1

10110

01100

01011

# Algebra di Boole

- Esempio di una tavola di verità per due espressioni logiche

A	B	C	A OR (B AND C)	A AND (B OR (NOT C))
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	1	1

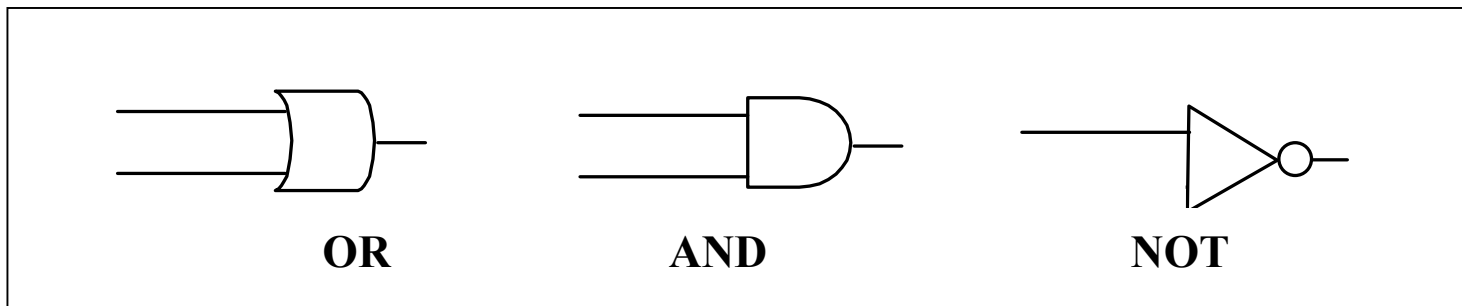
10110

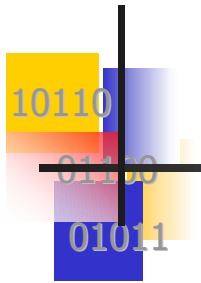
01100

01011

# Algebra di Boole e Porte Logiche

- **Porte Logiche**
- Sono elementi circuitali che corrispondono alle operazioni logiche e che possono essere combinati per effettuare operazioni più complesse.
- Ad esempio, tramite le porte logiche possono essere definiti circuiti sommatore.





# IL LINGUAGGIO JAVA

## Istruzione switch

## Ciclo for

## Array

10110

01100

01011

# Istruzione switch

- Mentre l'istruzione **if-else** corrisponde ad una scelta a due vie, esiste una istruzione che permette una scelta a molte vie: l'istruzione **switch**.

```
switch (intexpr)
{
    case intexpr1 : istruzioni;
    case intexpr2 : istruzioni;
    . . .
    default : statement;
}
```

- L'istruzione valuta l'espressione ed esegue le istruzioni del ramo **case** il cui valore dell'etichetta è uguale all'espressione e le istruzioni dei rami seguenti.
- Il tipo dell'espressione deve essere **byte**, **char**, **short** o **int**.



10110

01100

01011

## Istruzione switch

- Per ogni ramo possono essere inserite una o più istruzioni.
- Le istruzioni possono essere condivise tra più rami.
- Il ramo **default** viene usato per considerare valori non indicati negli altri rami **case**.
- L'istruzione **break** serve per terminare l'esecuzione dell'istruzione **switch** ed evitare di controllare/eseguire i rami seguenti.

```
. . . . .  
    case intexpr1 : istruzioni; break;  
. . . . .
```

10110

01100

01011

# Istruzione switch

- Esempio di istruzione **switch** per controllare i numeri pari e dispari tra 1 e 10:

```
switch (x)
{
  case 1:
  case 3:
  case 5:
  case 7:
  case 9: System.out.println("Il valore è dispari"); break;
  case 2:
  case 4:
  case 6:
  case 8:
  case 10: System.out.println("Il valore è pari"); break;
  default: System.out.println("Il valore non è compreso tra
    1 e 10");
}
```

10110

01100

01011

# Istruzione switch : esempio

```
public class Calcolatrice
{ public static void main(String args[])
  { double x, y, ris;
    String str; char op;

    x = Console.readDouble("Inserire operando :");
    y = Console.readDouble("Inserire operando :");
    str = Console.readString("Inserire operazione :");
    op = str.charAt(0);
    switch (op)
    { case '+' : ris = x+y;
      System.out.println("Risultato = " + ris); break;
      case '-' : ris = x-y;
      System.out.println("Risultato = " + ris); break;
      case '*' : ris = x*y;
      System.out.println("Risultato = " + ris); break;
      case '/' : ris = x/y;
      System.out.println("Risultato = " + ris); break;
      default :
        System.out.println("Operazione sconosciuta");
    }
  }
}
```



# Iterazioni

- Ogni volta che bisogna ripetere una o più istruzioni per un certo numero di volte occorre eseguire una **iterazione** o **ciclo**.
- Java offre tre istruzioni cicliche per eseguire operazioni iterative:
  - **for**
  - **while**
  - **do while**
- Presentano caratteristiche diverse ma sono equivalenti. Il loro uso dipende da quale è più semplice in un dato punto del programma.



# Iterazioni : istruzione for

- L'istruzione **for** permette di eseguire un ciclo per un numero prestabilito di volte e/o quando è vera una condizione.

```
for (inizializzazione_indice; condizione; operazione_indice)  
    { istruzioni;
```

```
ESEMPIO:   for (i = 1; i <= 10; i++)  
            { y = 2 * z + 4;  
              z = y - 1;  
            }
```

- Esegue le operazioni tra parentesi per 10 volte aggiornando i valori di *y* e *z*. (*y*, *z* e *i* devono essere stati dichiarati)

# Iterazioni : istruzione for

- L'indice può essere dichiarato nel for.

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

- Ognuna delle tre parti dentro le parentesi può essere vuota.

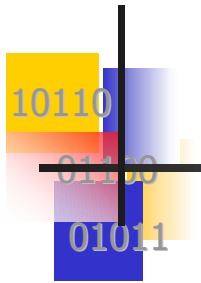
```
for(int i=0; ; i++)  
    System.out.println(i);
```

**COSA FA ?**

```
for( ; ; )  
    System.out.println(i);
```

**COSA FA ?**

- E se manca anche l'istruzione ?



# Iterazioni : istruzione for

```
for (inizial_indice; condizione; op_indice)  
{  
    istruzioni;  
}
```

1. Si esegue l'*inizializzazione di uno o più indici*,
2. Si controlla la *condizione*,
  - a. Se la *condizione* è *falsa* il **for** termina;
  - b. se la *condizione* è *vera* si eseguono le *istruzioni* e quindi si effettua l'*incremento/decremento dell'indice(i)*.
3. Quindi finché la *condizione* è *vera* si eseguono le *istruzioni* e la gestione dell'indice.

# Iterazioni : istruzione for

## Esempi

### Somma di n-esima di un numero (accumulo)

```
int somma = 0; int x= 2;
for (i = 1; i<20; i++)
    somma = somma + x;
```

### Calcolo del fattoriale di un numero

```
int fatt = 1; int x, i;
x = Console.readInt("Inserire x :");
for (i = 1; i <= x; i++)
    fatt = fatt * i;
System.out.print("Fattoriale di " + x + "= " + fatt);
```



# Ciclo for: esempio calcolo della potenza

```
public class ElevPotenza
{
    . . . .
    public void potenza()
    {
        int i, n, p, x=1;
        n = Console.readInt("Inserisci il numero");
        p = Console.readInt("Inserisci l'espon.");
        for (i=1; i<=p; i++)
            x = x * n;
        System.out.println("Risultato = " + x);
    }
    . . . .
}
```

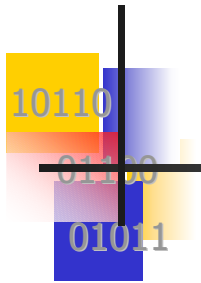
# Ciclo for

- L'operazione sull'indice può essere un decremento.

```
for(i=10; i >= 1; i=i-1)
{
    x = x + 2;
    y = y - 2 ;
}
```

- Oppure può essere una qualsiasi operazione aritmetica. Ad esempio: `i +=2` o `i-=5` o `i*=2`.
- Si possono avere più operazioni di inizializzazione e sull'indice separate da una `,` (virgola)

```
for(i=10, j=1; i > 0; j++, i--)
    x = 2*j + i;
```



# Array in Java

---

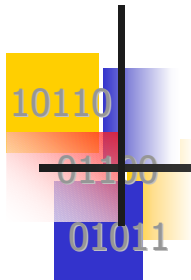
- **Costruttore di tipo array**
- Il costruttore **array** serve per memorizzare una sequenza (vettore) di variabili dello stesso tipo.
- Si possono avere array di interi, caratteri, float, etc;
- Tuttavia non si possono avere array che contengono elementi di tipo diverso.
- Esempio di dichiarazione di un array di interi:  
`int x[];`      OPPURE      `int[] x;`



# Array in Java

---

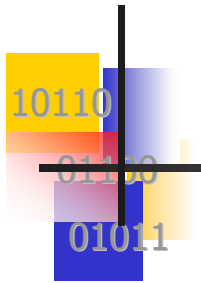
- Dopo la dichiarazione, per essere usato, l'array deve essere creato: `x = new int[20];`
- La creazione assegna uno spazio in memoria per l'array e lo rende usabile nel programma.
- Dichiarazione e creazione possono avvenire nella stessa istruzione:
  - `int x[] = new int[20];`
  - `int y[] = {1, 3, 5, 7, 9, 11};`
- Nel secondo caso abbiamo un array di sei elementi.



# Array in Java

- Ogni elemento dell'array ha nome uguale e viene individuato tramite un indice :
  - `x[i]`
  - `x[3] = 45;`
- Gli elementi di un array di lunghezza n in Java hanno indice [0] .. [n-1]:

```
int x[] = new int[20];  
  
. . . . .  
x[20] = 560;      /* istruzione errata ! */
```



# Array e ciclo for

- La scansione degli array in un ciclo è diretta:

```
int sequenza[] = new int[10]
int x = 4;
for (int i=0; i < 10; i++)
    sequenza[i] = x * i;
```

Corrisponde a

```
sequenza[0] = x * 0 → 0
sequenza[1] = x * 1 → 4
sequenza[2] = x * 2 → 8
. . . .
sequenza[8] = x * 8 → 32
sequenza[9] = x * 9 → 36
```

0
4
⋮
32
36

# Array e ciclo for

- Esempio di scansione:

```
int sequenza[] = new int[10]
```

```
. . . . .
```

```
for (int i=0; i < 10; i+=2)
```

```
    sequenza[i] = Console.readInt();
```

```
. . . . .
```

```
for (int i=0; i < 10; i++)
```

```
    System.out.print(sequenza[i]);
```