Operatori relazionali

Per confrontare i valori di due espressioni si usano gli operatori relazionali.

Questi operatori confrontano due o più elementi e restituiscono come risultato un valore logico di tipo **bool** che può assumere soltanto uno dei due valori booleani **true** o **false**, a seconda del risultato del confronto.

Gli operatori relazionali del C++ sono i seguenti:

LINGUAGGIO C++	L'operatore restituisce
a = = b	true se a è uguale a b
a != b	true se a è diversa da b
a < b	true se a è strettamente minore di b
a > b	true se a è strettamente maggiore di b
a <= b	true se a è minore o uguale a b
a >= b	true se a è maggiore o uguale a b

Operatori relazionali

Esempi

Supponiamo che **`a=2`**, **`b=3`** e **`c=6`**

`(a == 5)` risultato **`false`**.

`(a*b >= c)` risultato **`true`** in quanto (2*3 >= 6).

`(b+4 > a*c)` risultato **`false`** in quanto (3+4 > 2*6).

`((b=2) == a)` risultato **`true`**.

Operatori relazionali

Attenzione!

L'operatore **=** (un solo uguale) è differente dal simbolo **==** (doppio uguale), il primo è l'operatore di assegnamento (assegna il valore dell'espressione alla sua destra alla variabile alla sua sinistra e ritorna tale valore) mentre il secondo è l'operatore relazionale di uguaglianza che confronta i valori delle due espressioni che stanno ai suoi lati e ritorna il valore booleano **true** o **false** a seconda che esse abbiano lo stesso valore o valori diversi.

Quindi nell'ultima espressione `((b=2) == a)` viene dapprima assegnato il valore `2` a `b` e quindi tale valore viene confrontato con il valore di `a`, che è pure `2`, e quindi il risultato che si ottiene è il valore `true`.

Differenza tra = e ==

= Sam (You are Sam)

== Sam (Are you Sam?)

Operatori logici

Gli operatori logici sono operatori che richiedono operandi di tipo booleano e producono un risultato booleano.

Sono anche detti connettivi logici perché tengono insieme più espressioni logiche, formando un'unica espressione logica composta.

Vedremo i seguenti tre operatori logici:

! NOT

&& AND

|| OR

- L'operatore logico NOT è un operatore unario (ha un solo operando).
- Gli operatori logici AND e OR sono operatori binari (richiedono due operandi)

Operatori logici

L'operatore **!** è l'operatore logico di negazione NOT.

Esso ha un unico operando (di tipo **bool**) posto alla sua destra e il suo risultato è l'opposto del valore dell'operando:

- se l'operando è **true** esso ritorna **false**,
- se l'operando è **false** esso ritorna **true**.

NOT	
X	!X
true	false
false	true

Ad esempio:

!(5 == 5) ritorna **false** in quanto (5 == 5) è **true**

!(6 <= 4) ritorna **true** in quanto (6<=4) è **false**

!true ritorna **false**

!false ritorna **true**.

Operatori logici

Gli operatori `&&` e `||` sono gli operatori logici di congiunzione (AND) e disgiunzione (OR).

Essi hanno due operandi (di tipo `bool`) che indicheremo con X e Y.

L'espressione logica AND

```
X && Y
```

è vera se **entrambi** gli operandi X **e** Y sono veri, ed è falsa altrimenti

L'espressione logica OR

```
X || Y
```

è vera se **almeno uno** degli operandi X **o** Y è vero, ed è falsa altrimenti.

Operatori logici

Il risultato è quello riportato nella seguente tabella (detta tabella di verità):

X	Y	LINGUAGGIO C++	
		X && Y	X || Y
false	false	false	false
false	true	false	true
true	false	false	true
true	true	true	true

`((5 == 5) && (3 > 6))` ritorna **`false`** (*true* && *false*).

`((5 == 5) || (3 > 6))` ritorna **`true`** (*true* || *false*).

Strutture di controllo

Normalmente un programma non è semplicemente una sequenza di istruzioni da eseguire una dopo l'altra.

Durante la sua esecuzione esso può:

- effettuare una scelta tra due (o più) possibili alternative
- o ripetere più volte uno stesso gruppo di istruzioni.

A questo scopo il C++ fornisce delle strutture di controllo che servono a specificare come si deve comportare il programma.

Per parlare di strutture di controllo occorre introdurre il concetto di ***blocco di istruzioni***.

Un blocco di istruzioni è un gruppo di istruzioni separate da punti e virgola (`;`) e racchiuse tra parentesi graffe: `{` e `}` .

Costrutto di selezione

Selezione binaria

Il costrutto di selezione binaria permette di effettuare una scelta fra due possibili alternative. Per effettuare la scelta, dobbiamo valutare una condizione.

Un esempio tratto dalla vita di tutti i giorni potrebbe essere il seguente:

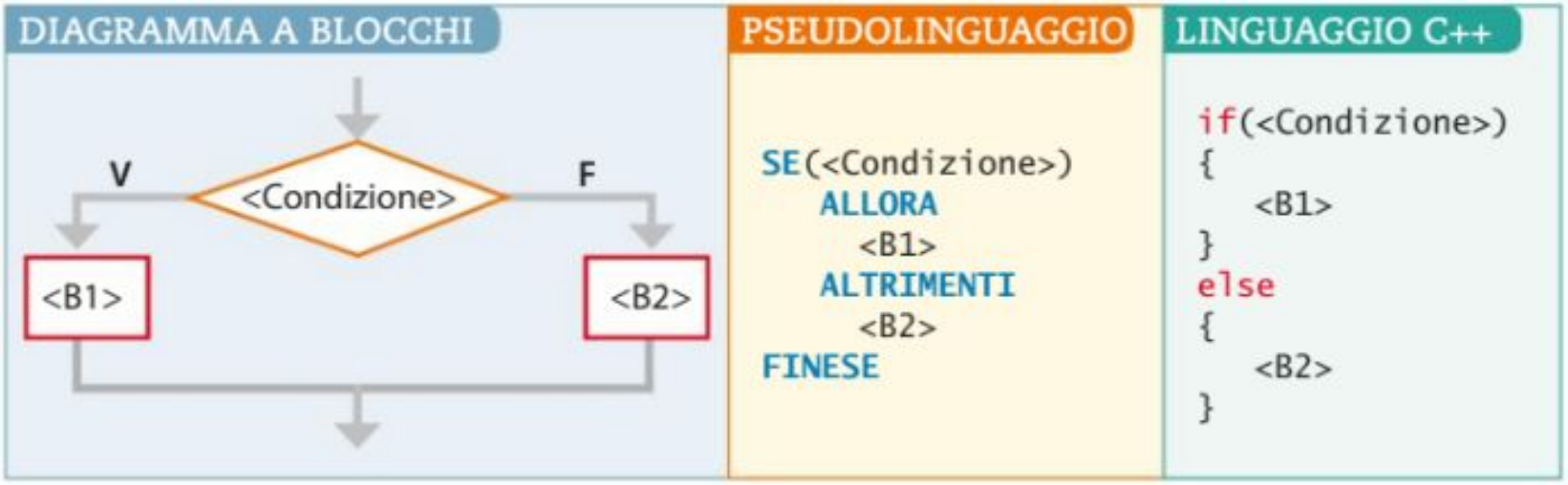
SE pioverà ALLORA prenderò l'autobus ALTRIMENTI farò una passeggiata

In termini più genera li possiamo scrivere:

- SE LA CONDIZIONE È VERA ALLORA
- ESEGUI ISTRUZIONE1
- ALTRIMENTI
- ESEGUI ISTRUZIONE2

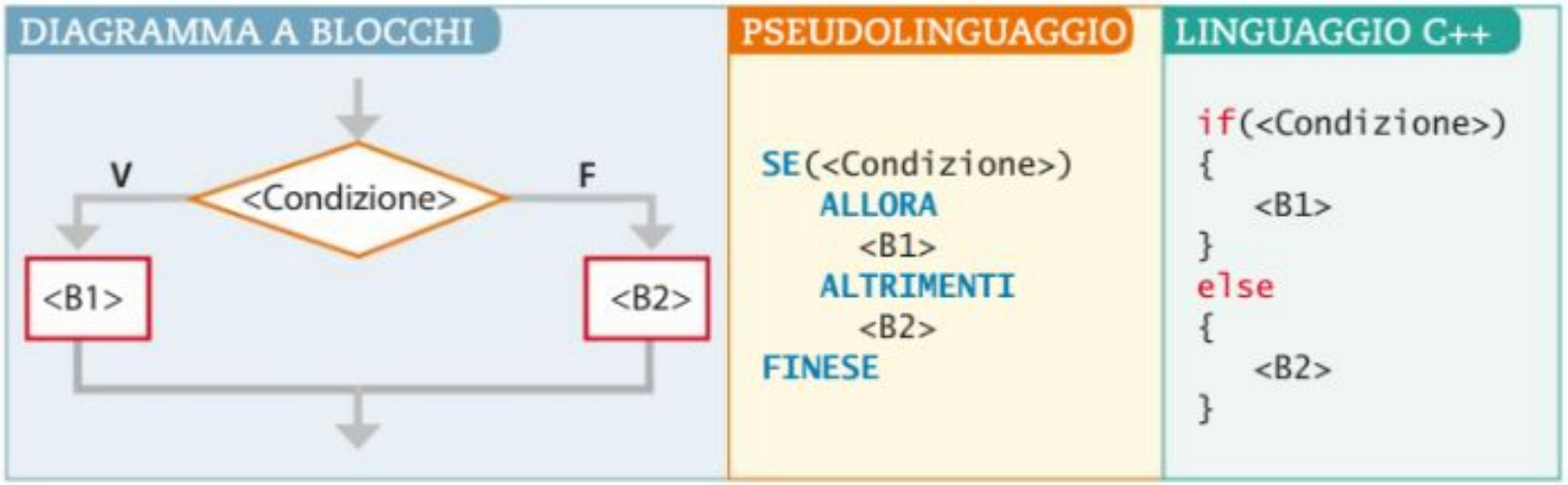
Costrutto di selezione binaria

La sintassi di questo costrutto, chiamato più precisamente selezione binaria, è la seguente:



- **Condizione** è una espressione di tipo **`bool`** (ossia una espressione il cui risultato è uno dei due valori di verità **`true`** o **`false`**).
- **B1** e **B2** possono indicare una singola istruzione o un blocco di istruzioni

Costrutto di selezione binaria



Quindi se la **Condizione** e Vera (**V**) si esegue il blocco di istruzioni **B1** altrimenti, cioè se la Condizione e Falsa (**F**) si esegue il blocco di istruzioni **B2**.

Condizione

Per formulare la condizione vengono utilizzati gli operatori relazionali già visti:

>

<

>=

<=

==

=

La condizione consiste nel confrontare la variabile scritta a sinistra dell'operatore relazionale con quanto scritto a destra dell'operatore stesso.

Condizione

Confronto con un valore immediato.

Ad esempio:

```
if (x>=5)
```

dove il valore della variabile x viene confrontato con il numero 5. Il risultato di questa condizione:

- è vero se il contenuto di x è maggiore di 5 (per esempio se x = 7) o è uguale a 5 (cioè se x=5),
- è falso se il contenuto di x è minore di 5 (per esempio se x = 3).

Attenzione: il falso di **<** (minore) è **>=** (maggiore e uguale) non solo **>** (maggiore) cosi come il falso di **>** (maggiore) è **<=** (minore e uguale) non solo **<** (minore).

Condizione

Confronto con una variabile.

Ad esempio:

```
if (y!=x)
```

dove il contenuto della variabile y che si trova a sinistra dell'operatore logico viene confrontato con il valore della variabile che si trova a destra.

Il risultato di questa condizione:

- è vero se la variabile x contiene un valore diverso dalla variabile y
- è falso se la variabile x contiene lo stesso valore della variabile y

Condizione

Confronto con una espressione.

Ad esempio:

```
if (t==(k*5)/b)
```

Dapprima viene calcolato il valore dell'espressione e successivamente viene confrontato il valore della variabile che si trova a sinistra dell'operatore logico `==` con il valore dell'espressione appena calcolata.

Se nel nostro esempio `t` vale `10`, `k` vale `3` e `b` vale `2`. la condizione sarà falsa poiché `t` è diverso da `7.5` (risultato di `(k*5)/b`)

Condizione

Se è necessario concatenare più condizioni in un **`if`** si utilizzano gli operatori logici AND, OR, NOT: **`&&`**, **`||`**, **`!`**

Esempi:

```
if (num >= 0 && num <=10)
if (a==b || b!=0)
```

Attenzione!

La condizione è racchiusa tra parentesi tonde e **non** è seguita dal punto e virgola.

Il blocco delle istruzioni è sempre racchiuso tra parentesi graffe.

Solo quando il blocco è composto da una sola istruzione è possibile omettere le parentesi, ma, per una questione di comodità di lettura e di abitudine, è consigliabile usarle sempre.

Indentazione

Hai notato i "rientri" utilizzati per gli if, gli else, le parentesi graffe e le istruzioni?

Le istruzioni vengono incolonnate in questo modo per indicare la loro dipendenza. Si evidenzia, in tal modo, che un'istruzione e contenuta in un'altra.

È buona abitudine curare questi incolonnamenti per garantire una buona leggibilità del codice.

Questa tecnica è detta indentazione e il suo utilizzo viene considerato come una norma fondamentale di buona programmazione.

La tecnica si basa sull'idea di usare i rientri allo scopo di separare più chiaramente le istruzioni e, in particolare, di rappresentare esplicitamente le relazioni di annidamento.

Esempio

Dato in input un numero intero N, comunicare se è minore di zero oppure maggiore o uguale a zero.

Dobbiamo quindi acquisire il numero N dall'esterno e confrontarlo con 0.

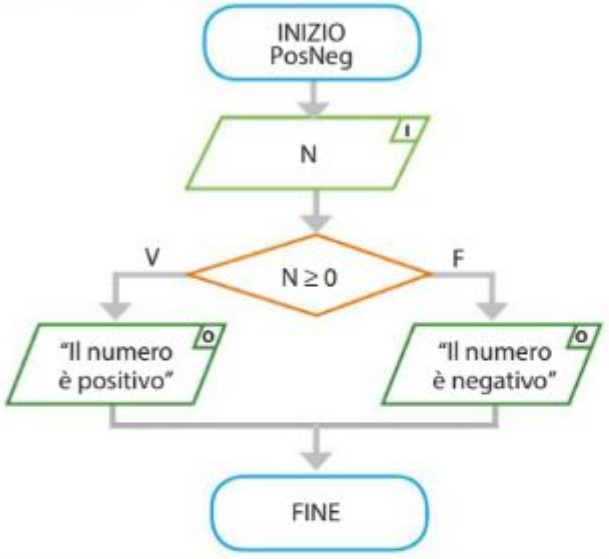
Analisi dei dati

NOME	FUNZIONE	TIPO	DESCRIZIONE
N	INPUT	INTERO	Valore da verificare
Messaggio	OUTPUT	STRINGA	Numero positivo oppure negativo

Soluzione

Formalizzazione dell'algoritmo

DIAGRAMMA A BLOCCHI



PSEUDOLINGUAGGIO

```
ALGORITMO PosNeg
VARIABILI
   N: INTERO
INIZIO
   LEGGI(N)
   SE(N≥0)
      ALLORA
         SCRIVI("Il numero è positivo")
      ALTRIMENTI
         SCRIVI("Il numero è negativo")
   FINESE
FINE
```

Realizzazione del programma

LINGUAGGIO C++

```
#include <iostream>
using namespace std;
int main()
{
   int n;
   cin >> n;
   if(n>=0)
      cout << "Il numero e' positivo";
   else
      cout << "Il numero e' negativo";
   return 0;
}
```

Esempio: IF nidificato

Dato in input un numero intero N, comunicare se è positivo, negativo o nullo.

L'analisi del problema è simile all'esempio precedente, con un controllo in più: se N è maggiore di zero è positivo, ma, se non lo è, potrebbe essere negativo o nullo, quindi controlliamo se è minore di zero: se è vero il numero è negativo, altrimenti è nullo.

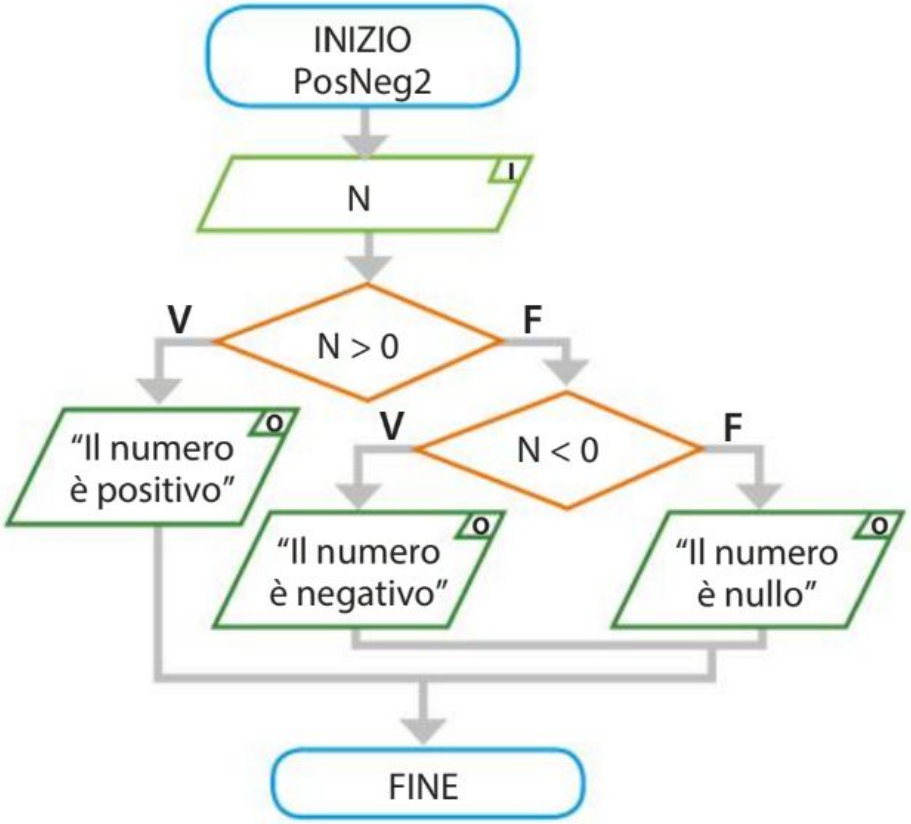
Analisi dei dati

NOME	FUNZIONE	TIPO	DESCRIZIONE
N	I	INTERO	Valore da verificare
Messaggio	O	STRINGA	Numero positivo oppure negativo oppure nullo

Flow chart e pseudocodice

Formalizzazione dell'algoritmo

DIAGRAMMA A BLOCCHI



PSEUDOLINGUAGGIO

```
ALGORITMO PosNeg2
VARIABILI
   N: INTERO
INIZIO
   LEGGI(N)
   SE(N>0)
      ALLORA
         SCRIVI("Il numero è positivo")
      ALTRIMENTI
         SE(N<0)
            ALLORA
               SCRIVI("Il numero è negativo")
            ALTRIMENTI
               SCRIVI("Il numero è nullo")
         FINESE
   FINESE
FINE
```

Programma C++

Questo esercizio presenta un costrutto di selezione all'interno di un altro, quindi nel ramo "else" del primo costrutto di selezione se ne esegue un altro. Un costrutto inserito in un altro dello stesso tipo si dice "annidato" o "nidificato" (*nested if*).

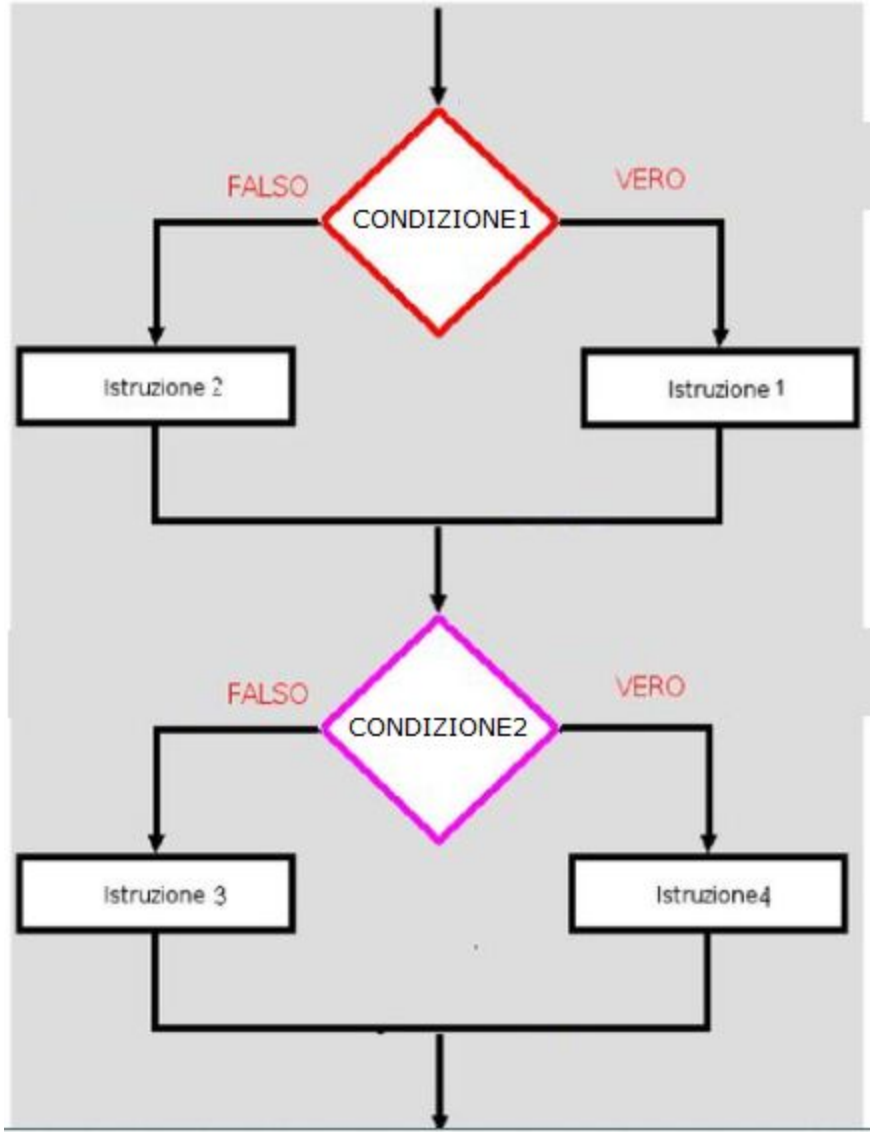
LINGUAGGIO C++

```
#include <iostream>
using namespace std;
int main()
{
   int n;
   cin >> n;
   if(n>0)
       cout << "Il numero e' positivo";
   else
       if(n<0)
          cout << "Il numero e' negativo";
       else
          cout << "Il numero e' nullo";
   return 0;
}
```

Selezioni in cascata

Quando due o più condizioni devono essere valutate una dopo l'altra come illustrato in figura si parla di selezioni in cascata: prima viene testata la condizione CONDIZIONE1 e di seguito la condizione CONDIZIONE2, indifferentemente dal risultato della condizione CONDIZIONE1.

La selezione a cascata si usa, quindi, quando le due condizioni sono indipendenti.



Selezioni in cascata

Dato in input il prezzo totale degli articoli scelti dal cliente, calcola e stampa il prezzo da pagare sapendo che:

- se il prezzo e superiore a 100 € si applica il 20% di sconto e, inoltre,
- se il pagamento avviene con strumenti di pagamento elettronici, devi applicare un ulteriore sconto del 5%.

Le due condizioni sono indipendenti?

Selezioni in cascata

Sì, le due condizioni sono indipendenti, quindi devo utilizzare la selezione in cascata.

Analisi del problema.

Variabili di input:

- prezzo di tipo float
- Poi occorre una variabile che mi informi riguardo il fatto che il cliente paghi con strumenti elettronici o in contanti, quindi devo dichiarare una variabile pos di tipo bool.

Poi potrei dichiarare delle costanti che definiscano:

- la soglia di prezzo al di sopra della quale si applica lo sconto del 20% (nel nostro caso: PSCONTO=100)
- le 2 percentuali di sconto (20% e 5%)

Selezioni in cascata

Analisi del problema.

Strategia risolutiva:

- Il primo costrutto `if` esamina la condizione: prezzo è maggiore o uguale a 100?
  - Se la condizione è vera si applica lo sconto del 20%
  - Altrimenti il prezzo resta invariato
- Il secondo costrutto `if` esamina la condizione: il pagamento avviene con strumenti elettronici?
  - Se la condizione è vera si applica lo sconto del 5%
  - Altrimenti il prezzo resta invariato

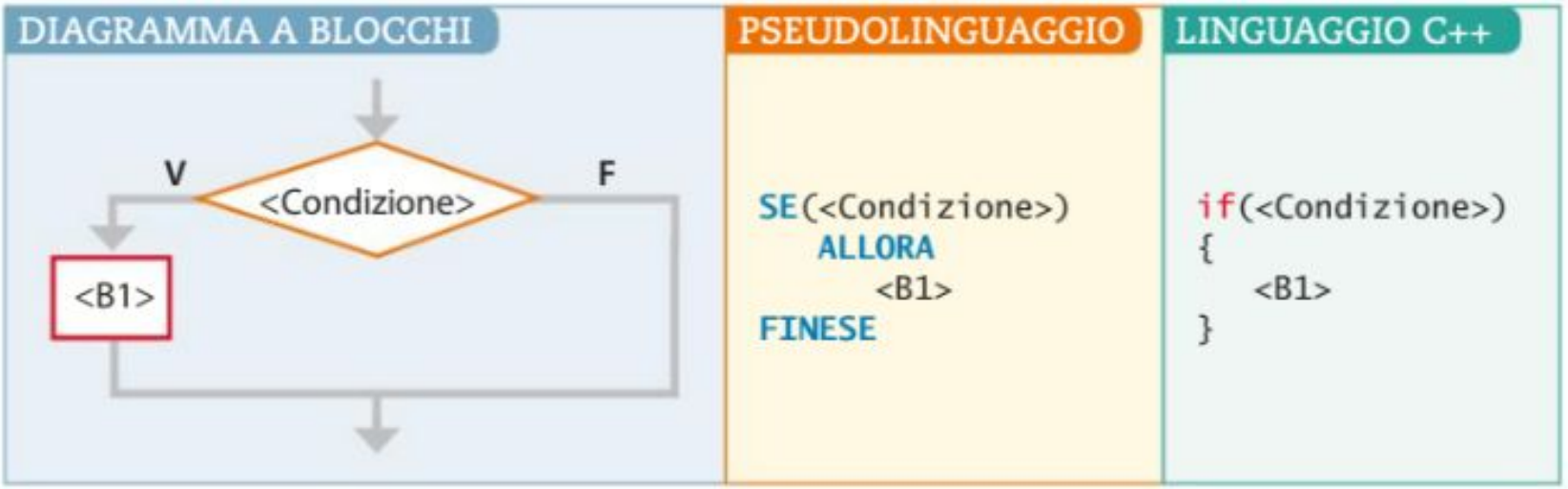
Programma C++

```cpp
#include <iostream>
using namespace std;
int main ()
{
  float prezzo;                 //Dichiarazione delle variabili: prezzo e' un numero reale
  bool pos;                     //La variabile pos (tipo bool) vale 1 per pagamenti elettronici, 0 altrimenti
  const int PSCONTO = 100;      //La costante PSCONTO indica la soglia al di sopra della quale si ottiene lo sconto

  cout << "Inserire il prezzo degli articoli" << endl;
  cin >> prezzo;                //l'utente inserisce il prezzo
  if (prezzo >= PSCONTO)        //Primo costrutto di selezione
    {
      prezzo = prezzo * 0.8;
    }
  else
    {
      prezzo = prezzo;
    }
  cout << "Il cliente deve pagare: " << prezzo << ". Pagamento elettronico?" <<endl;

  cin >> pos;                  //l'utente deve inserire 1 (true) se utilizza pagamenti elettronici, 0 (false) altrimenti
  if (pos == 1)                //Secondo costrutto di selezione
    {
      prezzo = prezzo * 0.95;
    }
  else
    {
      prezzo = prezzo;
    }
  cout << "Il prezzo finale ammonta a: " << prezzo << endl;

  return 0;
}
```

Costrutto di selezione unaria

Il costrutto selezione può presentarsi anche con un solo ramo, cioè senza l'alternativa **ALTRIMENTI**.

Questo caso, chiamato Selezione unaria (o semplice), si rappresenta sintatticamente nel seguente modo:



Esempio di selezione unaria

Scriviamo un algoritmo che calcola il valore assoluto di un numero.

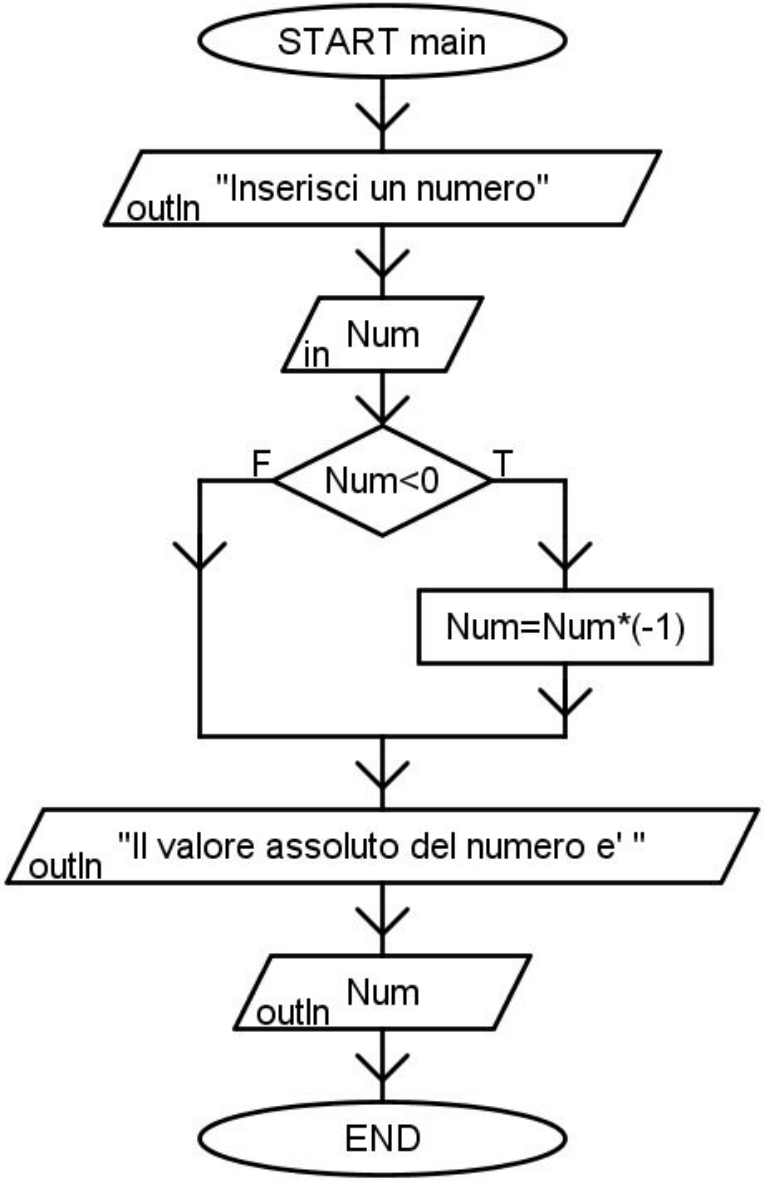
In questo caso:

- se il numero è già positivo non dobbiamo effettuare nessuna operazione, mentre
- se è negativo lo moltiplichiamo per –1, in modo da trasformarlo in positivo.

Analisi del problema

- Variabile di input: **`Num`**
- Variabile di output: Valore assoluto di **`Num`**, che:
  - Coincide con **`Num`** se esso è positivo
  - è uguale a **`Num*(-1)`** se esso è negativo

Diagramma di flusso



Pseudocodice

```
inizio
   scrivi "Inserisci un numero"
   leggi Num
   se Num <0
      allora
         esegui Num = Num *(-1)
   finese
   scrivi "Il valore assoluto del numero è: "
   scrivi Num
fine
```

Programma C++

```
//Programma che calcola il valore assoluto di un numero intero

#include <iostream>
using namespace std;

int main ()
{
  int Num;                          //Dichiarazione delle variabili

  cout << "Inserisci un numero intero: " << endl;
  cin >> Num;

  if (Num < 0)                      // Selezione unaria
    Num = Num * (-1);

  cout << "Il valore assoluto del numero da te inserito e': " << Num << endl;

  return 0;
}
```

Calcolare la potenza di un numero in C++

- Innanzitutto per poter usare la funzione predefinita per il calcolo della potenza è necessario importare la libreria di funzioni che la contiene ovvero la **libreria cmath** con la direttiva che deve essere inserita all'inizio del programma:

  ```
  #include <cmath>
  ```

- La funzione da utilizzare poi si chiama **`pow`** (abbreviazione di power, potenza in inglese), questa funzione riceve due parametri di tipo **`double`**, il primo è la base e il secondo è l'esponente, ma può essere utilizzata anche per numeri interi (tipo **`int`**) e numeri **`float`**.
- Se si vuole calcolare la potenza di a elevato alla b si dovrà scrivere la seguente istruzione:

  ```
  cout << pow(a,b);
  ```

Esempio

Il seguente programma chiede all'utente di inserire due numeri, **`base`** ed **`esponente`**, e calcola il primo elevato al secondo:

```
#include <iostream>
#include <cmath>                        //Direttiva per importare la libreria cmath
using namespace std;
int main ()
{
  double base;                          //Dichiarazione delle variabili
  double esponente;
  cout << "inserisci la base ";
  cin >> base;
  cout << "inserisci l'esponente ";
  cin >> esponente;
  cout << "la potenza di " << base << " elevato alla " << esponente << " e' ";
  cout << pow (base, esponente);        //Calcolo della potenza
  return 0;
}
```

Calcolare la radice quadrata di un numero in C++

- Per poter usare la funzione predefinita per il calcolo della radice quadrata è necessario importare la libreria delle funzioni matematiche, cioè la **libreria cmath** con la direttiva che deve essere inserita all'inizio del programma:

  ```
  #include <cmath>
  ```

- La funzione da utilizzare si chiama `sqrt` (abbreviazione di square root, radice quadrata in inglese); questa funzione prende come parametro un numero con la virgola (tipo `float` o `double`) ma può essere utilizzata anche per numeri interi (tipo `int`) e restituisce un risultato con la virgola.

Esempio

Il seguente programma chiede all'utente di inserire un numero **`n`** e ne calcola la radice quadrata **`r`**:

```
//Programma per calcolare la radice quadrata di un numero inserito dall'utente

#include <iostream>
#include <cmath>
using namespace std;
int main ()
{
  float n;
  cout << "Inserisci il numero di cui vuoi calcolare la radice quadrata: " <<endl;
  cin >> n;
  float r = sqrt (n);
  cout << "La radice quadrata del numero " << n << " vale: " << r << endl;
  return 0;
}
```