Informatica - Mod. Programmazione
Lezione 03

Prof. Giuseppe Psaila

Laurea Triennale in Ingegneria Informatica
Università di Bergamo

Istruzioni di Controllo

- Le **Istruzionoi di Controllo** consentono di alterare il **Flusso Sequenziale** del programma
- In altre parole, consentono di controllare l'esecuzione del programma
- adattando il suo comportamento alle diverse situazioni
- Due famiglie:
  Istruzioni Condizionali
  Cicli

Istruzioni Condizionali

Prima Forma:

`if(`*Condizione*`)`

Azione

- Se la situazione espressa dalla *Condizione* è verificata,
  $\Rightarrow$ Si esegue l'*Azione*
- Altrimenti si passa oltre

Programma: Divisione_04.cpp

```
{
    int a;
    int b;
    float r;

    cout << "inserisci due valori" << endl;
    cin >> a;
    cin >> b;

    if( b != 0)
    {
        r = a / (float)b;
        cout << "r: " << r << endl;
    }

    return 0;
}
```

Istruzioni Condizionali

- `b != 0`
  è la *Condizione*
  L'operatore `!=` corrisponde all'operatore matematico
  $\neq$
- Quindi la *Condizione* è verificata se il valore della variabile `b` è diverso da `0`

Istruzioni Condizionali

- L'*Azione* è contenuta tra parentesi graffe,
- perché è costituita da più di una istruzione
- Se l'*Azione* è costituita da **Una Sola Istruzione**, le parentesi graffe possono essere omesse

Seconda Forma:

`if(`*Condizione*`)`

    AzioneVera

`else`

    AzioneFalsa

- Se la situazione espressa dalla *Condizione* è verificata,
  $\Rightarrow$ Si esegue l'*AzioneVera*
- Altrimenti
  $\Rightarrow$ Si esegue l'*AzioneFalsa*

Programma: Divisione_05.cpp

```
int main()
{
    int a;
    int b;
    float r;

    cout << "inserisci due valori" << endl;
    cin >> a;
    cin >> b;

    if( b != 0)
    {
        r = a / (float)b;
        cout << "r: " << r << endl;
    }
    else
        cout << "ATTENZIONE: DIVISIONE PER ZERO";

    return 0;
}
```

Istruzioni Condizionali

- La **Prima Forma** va bene quando, se la *Condizione* è verificata, si deve eseguire qualche cosa in più rispetto al resto del programma
- La **Seconda Forma** va bene quando ci sono **Due Azioni Alternative** da eseguire: una se la *Condizione* è verificata, l'altra se la *Condizione* non è verificata

Condizioni

- Una **Condizione** esprime una situazione sui valori (delle variabili)
- Se la situazione espressa è `Verificata`,
  si dice che la condizione è **VERA** (True)

  Se la situazione espressa **Non È Verificata**
  si dice che la condizione è **FALSA** (False)

Operatori di Confronto

- *Espressione1* > *Espressione2*
  Espressione1 >= *Espressione2*
  Vero se il valore di *Espressione1* è maggiore (maggiore o uguale) del valore di *Espressione2*
- *Espressione1* < *Espressione2*
  Espressione1 <= *Espressione2*
  Vero se il valore di *Espressione1* è minore (minore o uguale) del valore di *Espressione2*

Operatori di Confronto

- *Espressione1* == *Espressione2*
  Vero se il valore di *Espressione1* è uguale al valore di *Espressione2*
- *Espressione1* != *Espressione2*
  Vero se il valore di *Espressione1* è diverso al valore di *Espressione2*

Operatori di Confronto

- Gli **Operatori di confronto** consentono di specifica i **PREDICATI LOGICI**
- Un Predicato Logico è una **Condizione Elementare**

Condizioni Composte

- Si possono costruire **Condizioni Complesse** partendo da **Condizioni Elementari** (Predicati)
- Si usa la cosiddetta **Algebra di Boole**
- In quest'algebra le variabili possono assumere due soli valori: **Falso** o **Vero**
  Tre operatori base: **AND**, **OR** e **NOT**

Operatore AND

C_1 && C_2

- C_1 e C_2 sono condizioni
- La *Tabella di Verità* definisce il comportamento dell'operatore

C_1	C_2	C_1 && C_2
F	F	F
F	V	F
V	F	F
V	V	V

Operatore OR

$C_1 \;||\; C_2$

- C_1 e C_2 sono condizioni
- La *Tabella di Verità* definisce il comportamento dell'operatore

C_1	C_2	$C_1 \;\|\|\; C_2$
F	F	F
F	V	V
V	F	V
V	V	V

Operatore NOT

$!(C)$

- C è una condizione
- La *Tabella di Verità* definisce il comportamento dell'operatore

C	$!(C)$
F	V
V	F

Condizioni

- Scriviamo una serie di programmi che fanno la stessa cosa
- Chiedono un valore all'utente
- e dicono se quel valore è (o non è) nell'intervallo $[0, 10]$

Programma: Intervallo_01.cpp

```
int main()
{
    int x;

    cout << "inserisci un valore" << endl;
    cin >> x;

    if( x >= 0 && x <= 10)
        cout << "x=" << x
             << " e` in [0, 10]"<< endl;
    else
        cout << "x=" << x
             << " non e` in [0, 10]"<< endl;

    return 0;
}
```

Condizioni

- La condizione è basata sull'AND
- è Vera se il valore della variabile `x` è nell'intervallo

Programma: Intervallo_02.cpp

```
#include <iostream>

using namespace std;

int main()
{
    int x;

    cout << "inserisci un valore" << endl;
    cin >> x;

    if( x < 0 || x > 10)
        cout << "x=" << x
             << " non e` in [0, 10]"<< endl;
    else
        cout << "x=" << x
             << " e` in [0, 10]"<< endl;

    return 0;
}
```

Condizioni

- La condizione è basata sull'OR
- è Vera se il valore della variabile `x` NON è nell'intervallo

Programma: Intervallo_03.cpp

ATTENZIONE: ERRORE dei Principianti

```
#include <iostream>

using namespace std;

int main()
{
    int x;

    cout << "inserisci un valore" << endl;
    cin >> x;

    if( x < 0 && x > 10)
        cout << "x=" << x
             << " non e` in [0, 10]"<< endl;
    else
        cout << "x=" << x
             << " e` in [0, 10]"<< endl;

    return 0;
}
```

Condizioni

- La condizione è **SEMPRE FALSA**
- perchè non è possibile che il valore della variabile `x` sia contemporaneamente
  minore di 0
  e maggiore di 10

Programma: Intervallo_04.cpp

```
#include <iostream>

using namespace std;

int main()
{
    int x;

    cout << "inserisci un valore" << endl;
    cin >> x;

    if( !(x >= 0 && x <= 10))
        cout << "x=" << x
             << " non e` in [0, 10]"<< endl;
    else
        cout << "x=" << x
             << " e` in [0, 10]"<< endl;

    return 0;
}
```

Condizioni

- Vogliamo che la condizione sia vera se il valore della variabile `x` è fuori dall'intervallo
- Ma ci viene più facile scrivere la condizione che è vera quando il valore della variabile `x` è nell'intervallo
- Soluzione: prendiamo quest'ultima e la neghiamo con l'operatore NOT

Teoremi di DeMorgan

Date due condizioni C_1 e C_2

- $(C_1 \text{ AND } C_2) \equiv \text{NOT}(\text{ NOT}(C_1) \text{ OR NOT}(C_2))$
- $(C_1 \text{ OR } C_2) \equiv \text{NOT}(\text{ NOT}(C_1) \text{ AND NOT}(C_2))$

Teoremi di DeMorgan

```
x<0 || x>10 ≡
!( !(x<0) && !(x>10) ) ≡
!( x>=0 && x<=10 )
```

Gestione del Valore Vero/Falso

- Il Linguaggio C (e il C++ lo ha ereditato) ha introdotto una convenzione particolar per gestire i valori Vero/Falso
- Il valore 0 viene Sempre considerato Falso
- Un qualsiasi valore diverso da 0 è considerato Vero
- Quindi vengono usati i numeri interi per riportare i risultati dei confronti
- Pertanto, potremmo assegnare il risultato di una condizione ad una variabile intera

Programma: Intervallo_05.cpp

```
int x;
int r;

cout << "inserisci un valore" << endl;
cin >> x;

r = !(x >= 0 && x <= 10);
cout << "Confronto: " << r << endl;
if( r )
    cout << "x=" << x
         << " non e` in [0, 10]"<< endl;
else
    cout << "x=" << x
         << " e` in [0, 10]"<< endl;

return 0;
```

Condizioni

- Gli operatori di confronto e quelli logici producono sempre `0` (Falso) e `1` (Vero)
- Il risultato della valutazione della condizione viene assegnato alla variabile `r`
- Nell'`if`, la condizione è espressa dalla sola variabile `r`: se vale `0`, viene considerata Falsa, se vale `1`, viene considerata vera

Condizioni

Domanda: Si Può Scrivere

```
if( 0 <= x <= 10 )
```

?

Condizioni

Domanda: Si Può Scrivere
`if( 0 <= x <= 10 ) ?`

- Il compilatore non si arrabbia
- Ma è **SEMPRE VERA**
- Va letta così (proprietà associativa):
  `(0 <= x ) <= 10`
- Se `x` è minore di `0`, `(0<=x)` vale `0`, altrimenti vale `1`
- ma sia `0` che `1` sono minori di `10`
  quindi è **SEMPRE VERA**

ATTENZIONE Agli Errori di Battitura

```
if( a = 0 )
```

Invece di

```
if( a == 0 )
```

- Il compilatore non segnala errore
- Ma la condizione viene considerata **SEMPRE FALSA**
- e il valore della variabile a diventa 0

Evitate il Problema Scrivendo

```
if( 0 == a )
```

- Dimenticando un = diventa

  ```
  if( 0 = a )
  ```

- Il compilatore segnala errore

Programma: Errore.cpp

```
int main()
{
    int a;

    cout << "inserisci un valore" << endl;
    cin >> a;

    if( 0 = a )
        cout << "ZERO" << endl;
     else
        cout << "NON ZERO" << endl;

    return 0;
}
```

Programma: Errore.cpp

```
 7    int a;
 8
 9    cout << "inserisci un valore" << endl;
10    cin >> a;
11
12    if( 0 = a )
13       cout << "ZERO" << endl;
14     else
15        cout << "NON ZERO" << endl;
16
17    return 0;
18 }
```

(2) Resources | Compile Log | Debug | Find Results | Close

File	Message
C:\Users\Utente\Documents\Lavoro\corsi\Informatic...	In function 'int main()':
C:\Users\Utente\Documents\Lavoro\corsi\Informatica\2...	[Error] lvalue required as left operand of assignment

Condizioni

Leggiamo il messaggio di errore

`lvalue required as left operand of assignment`

- "lvalue" sta per "left value"
- un lvalue è qualcosa che può stare a sinistra di un assegnamento
- Per quello che abbiamo visto fino ad ora, SOLO le variaili possono stare a sinistra di un assegnamento

SFIDA

Scrivere un programma (di nome **Sfida_03a.cpp**) che

- legge da tastiera due numeri interi `a` e `b`
- se `b` è diverso da `0`
  calcola il quoziente `q` e il resto `r` della divisione
- - se `q` è minore di `r`, scrive in output il valore della differenza `r - q`
  - altrimenti scrive in output il valore del prodotto di `q` per `r`