Istruzioni del Linguaggio C/C++

Istruzione di assegnamento

Istruzione di assegnamento

- Denotata mediante il simbolo **=**

  (l’operatore relazionale di uguaglianza è denotata con il simbolo **==**)
- nome_variabile = espressione ;
- Viene utilizzata per assegnare ad una variabile (non ad una costante!) il valore di un’espressione
- L'identificatore a sinistra rappresenta l'indirizzo della variabile di cui vogliamo modificare il valore, tale indirizzo è detto **lvalue**
- Il valore dell'espressione che compare a destra rappresenta il nuovo valore, tale valore è detto **rvalue**

Istruzione di assegnamento

- Denotata mediante il simbolo **=**

  (l'operatore relazionale di uguaglianza è denotata con il simbolo **==**)
- Viene utilizzata per assegnare ad una variabile (non ad una costante!) il valore di un'espressione

Esempio

```
int  N;
```

simbolo	*indirizzo*
N	**1600**

	...
1600	
	...

L'esecuzione di una **definizione** provoca l'allocazione di uno spazio in memoria equivalente a quello necessario a contenere un dato del tipo specificato

```
N = 150;
```

simbolo	*indirizzo*
N	**1600**

	...
1600	**150**
	...

L'esecuzione di un **assegnamento** provoca l'inserimento nello spazio relativo alla variabile del valore indicato a destra del simbolo **=**

Concatenazione

Elenco delle istruzioni da eseguire, nell'ordine in cui vogliamo che vengano eseguite (sequenza)

ESEMPI

```
int N ;

N = 3 ;

cout<<N<<endl;
```

equivalente a:

```
int N; N = 3; cout<<N<<endl ; // MENO LEGGIBILE !!!
```

Istruzioni espressione

Qualsiasi espressione (logica, condizionale, assegnamento) seguita da un ; è un'*istruzione (semplice)*

ESEMPI

```
x = 0;  y = 1;        /*  due istruzioni */

k++;

3;                    /*  non fa nulla */

;                     /*  istruzione nulla */
```

Istruzione di assegnamento (*cont.*)

- L'esecuzione di un'istruzione di assegnamento comporta innanzitutto la valutazione di tutta l'espressione a destra dell'assegnamento.
Es.,

```
int c, d;
c = 2;
d = (c+5)/3 - c;
d = (d+c)/2;
```

- **Solo dopo** si inserisce il valore risultante (**rvalue**) nella locazione di memoria relativa alla variabile (posta a sinistra dell'assegnamento)
  - Risultato espressione assegnamento: indirizzo della variabile

- Il primo assegnamento di un valore ad una variabile dichiarata viene detto **inizializzazione**.
In C/C++, l'inizializzazione si può effettuare anche al momento della dichiarazione. Es.,

```
int a, b=56;
```

Esercizio 1 (*Specifica*)

- Inverti l'ordine delle cifre che formano un intero positivo non multiplo di 10 che sia compreso fra 101 e 999.

- Per esempio: **234** → **432**

Esercizio 1 (*Algoritmo*)

Idea!: *Utilizzare le operazioni di modulo e di divisione fra numeri interi*.

Dato un *numero*, valgono le seguenti relazioni:

- **Unita = (numero/10^0)%10;**
  - **Es., (234/1)%10 = 4**
- **Decine = (numero/10^1)%10;**
  - **Es., (234/10)%10 = 3**
- **Centinaia = (numero/10^2)%10;**
  - **Es., (234/100)%10 = 2**

Esercizio 1 (*Programma*)

```
main()
{
  int numero;
  int unita, decine, centinaia, risultato;

  cin>>numero;

  unita = (numero)%10;
  decine = (numero/10)%10;
  centinaia = (numero/100)%10;

  risultato =unità*100+decine*10+centinaia;
  cout<<risultato;
}
```

Esercizi da svolgere

Esercizio 1bis

Si risolva lo stesso problema posto nell'Esercizio 1 accettando in input un qualsiasi numero intero (anche divisibile per 10) compreso fra 100 e 999.
Esempio: 100 va ristampato come 001

Esercizio 1ter (servirà conoscere le istruzioni cicliche)

Si risolva lo stesso problema posto nell'Esercizio 1 accettando in input un qualsiasi numero intero positivo (si richieda in input sia il numero sia il numero di cifre che lo compongono).

Tipo booleano

Tipo booleano

- Disponibile solo in C++
- Nome del tipo: *bool*
- Valori possibili: vero (*true*), falso (*false*)
  - *true* e *false* sono due letterali booleani
- Esempio di definizione:

```
bool u, v = true ; // seconda variabile
                   // inizializzata a vero
```

- Operazioni possibili: ...

Operatori logici

operatore logico	*operatore*	**C/C++**
not logico (negazione)	*unario*	**!** (prefisso)
and logico (congiunzione)	*binario*	**&&** (infisso)
or logico (disgiunzione)	*binario*	**||** (infisso)

Tabella di verità degli operatori

<table>
<tr><th>AND</th><th>Ris.</th><th>OR</th><th>Ris.</th><th>NOT</th><th>Ris.</th></tr>
<tr><td>V && V</td><td>V</td><td>V || V</td><td>V</td><td>!V</td><td>F</td></tr>
<tr><td>V && F</td><td>F</td><td>V || F</td><td>V</td><td>!F</td><td>V</td></tr>
<tr><td>F && V</td><td>F</td><td>F || V</td><td>V</td><td></td><td></td></tr>
<tr><td>F && F</td><td>F</td><td>F || F</td><td>F</td><td></td><td></td></tr>
</table>

Tipo booleano e tipi numerici

- Se un oggetto di tipo booleano è usato dove è atteso un valore numerico
  - *true* è convertito a 1
  - *false* è convertito a 0

In C, non esistendo il tipo boolean, gli operatori logici operano su interi e restituiscono un intero:
- il valore 0 viene considerato **falso**
- ogni valore diverso da 0 viene considerato **vero**
- **il risultato è 0 o 1**

Esempi

5 && 7 0 || 33 !5

Espressioni

Espressioni

- Costrutto sintattico formato da letterali, identificatori, operatori, parentesi tonde, ...
  - **aritmetica**: produce un risultato di tipo aritmetico
  - **logica**: produce un risultato di tipo booleano
- Gli operatori di confronto producono un risultato di tipo booleano
- Mediante le parentesi tonde possiamo forzare l'ordine di valutazione dei termini

Espressioni

- Esempi:

Espressioni aritmetiche

2 + 3

(2 + 3) * 5

4 > 2

true || (2 > 5)

Espressioni logiche

Sintassi del C++

Sintassi del C/C++

- Ora che abbiamo più familiarità col linguaggio, fissiamo un po' meglio la sintassi ...
- Programma C/C++: sequenze di parole (*token*) delimitate da spazi bianchi (*whitespaces*)
  - Spazio bianco: carattere spazio, tabulazione, a capo
- Token possibili: identificatori, parole chiave, espressioni letterali, operatori, separatori
  - Operatore: denota una operazione nel calcolo delle epsressioni
  - Separatore: ( ) , ; : { }

Uso degli spazi bianchi

- Una parola chiave ed un identificatore vanno separati da spazi bianchi
  - Es: int a; // inta sarebbe un identificatore !
- In tutti gli altri casi gli spazi bianchi non sono necessari

Altri operatori

- Assegnamento abbreviato: +=, -=, *=, /=, ...
  - a += b ; <==> a = a + b ;
- Incremento e decremento: ++ --
  - int a; a++; a--; ++a; --a;
- Prefisso: prima si effettua l'incremento/decremento, poi si usa il valore della variabile. Restituisce un **lvalue**
  - int a = 3; cout<<++a; // stampa 4
  - (++a) = 4; // valido
- Postfisso: prima si usa il valore della variabile, poi si effettua l'incremento/decremento. Restituisce un r**value**
  - int a = 3; cout<<a++; // stampa 3
  - (a++) = 4; // ERRORE !!!

Proprietà operatori

- **Posizione** rispetto ai suoi operandi (o argomenti): prefisso, postfisso, infisso
- **Numero di operandi** (**arietà**)
- **Precedenza** (o **priorità**) nell'ordine di esecuzione
  - Es: 1 + 2 * 3 è valutato come 1 + (2 * 3)
    k<b+3 è valutato come *k<(b+3), e non (k<b) +3*
- **Associatività**: ordine con cui vengono valutati due operatori con la stessa precedenza.
  - Associativi a sinistra: valutati da sinistra a destra
    - Es: / è associativo a sinistra, quindi 6/3/2 ⇔ (6/3)/2
  - Associativi a destra: valutati da destra a sinistra
    - Es: = è associativo a destra ...

Associatività dell'assegnamento

- L'operatore di assegnamento può comparire più volte in un'istruzione.
- L'associatività dell'operatore di assegnamento è a destra
  Es.,

```
k = j = 5;
```

equivale a

```
j = 5;
k = j;
```

```
k = j+2 = 5;
```

NON SI PUO' FARE!

Ordine valutazione espressioni

- Ordine
  - *Fattori*: dalle espressioni letterali e calcolo delle funzioni e degli operatori unari
  - *Termini*: dal calcolo degli operatori binari
    - Moltiplicativi: * / %
    - Additivi: + -
    - Traslazione: << >>
    - Relazione: < > <= >=
    - Eguaglianza: == !=
    - Logici: && ||

Sintesi priorità degli operatori

Fattori

! ++ --

Termini

* / %

+ -

> >= < <=

== !=

&&

||

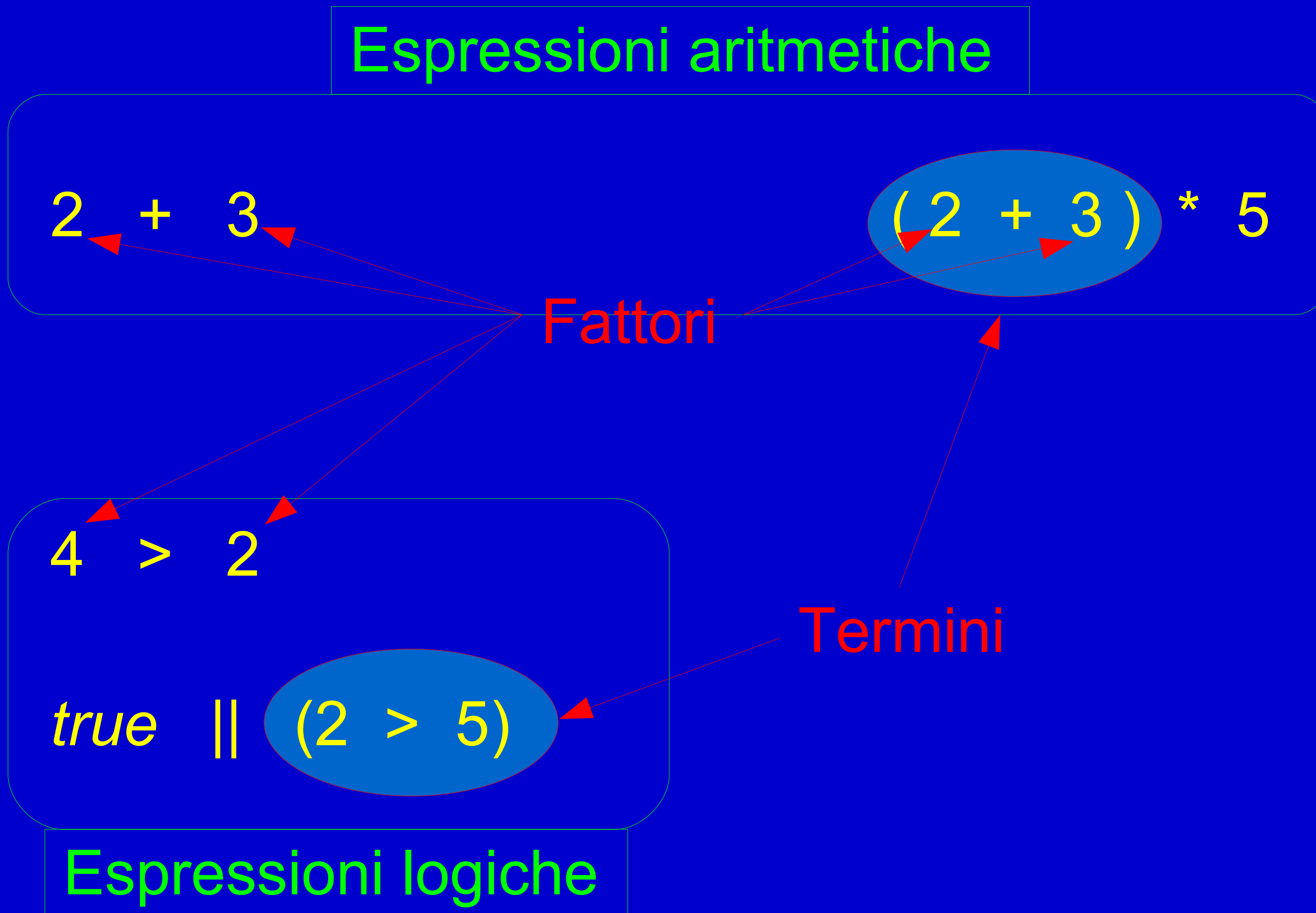
? :

Assegnamento

=

Espressioni

- Esempi:



Programmazione strutturata

Programmazione “strutturata”

- **Si parla di programmazione strutturata se si utilizzano <u>solo le seguenti strutture</u> per alterare il flusso di controllo** [*Dijkstra, 1969*]:
  - **concatenazione** (o **composizione**)
    legata al concetto di enumerazione (sequenza)
  - **selezione o (istruzione condizionale**)
    ramifica il flusso di controllo in base al valore vero o falso di una espressione detta “*condizione di scelta*”
  - **iterazione**
    dà luogo all’esecuzione ripetuta di un’istruzione (semplice o composta) finché permane vera una espressione detta “*condizione di iterazione*”

- **OBIETTIVO: rendere i programmi più *leggibili*, *modificabili* e *manutenibili***

Teorema di Jacopini-Böem

Le strutture di ***concatenazione***, ***iterazione*** e ***condizione*** costituiscono un insieme ***completo***
in grado di esprimere ***tutte le funzioni calcolabili***

- **Dunque, l'uso di queste *sole* strutture di controllo *non limita* il potere espressivo.**

  P.es., un linguaggio con i seguenti

  Tipi di dato: *Naturali con l'operazione di somma (+)*

  Istruzioni: *assegnamento*
  istruzione composta
  istruzione condizionale
  istruzione di iterazione

  è un ***linguaggio completo***, cioè è un linguaggio in grado di esprimere tutte le funzioni calcolabili.

Programmazione "strutturata" in C/C++

- **Istruzioni composte**
  - concatenazione di istruzioni "semplici" → **;**
  - blocco → **{ }**

- **Istruzioni espressione**
  - esempio: assegnamento → **=**

- **Istruzioni condizionali**
  - alternativa → **if (), if () else**
  - selezione → **switch ()**

- **Istruzioni di iterazione**
  - ciclo → **while (), do while (), for (; ;)**

Istruzioni Condizionali

Istruzioni condizionali

Solitamente vengono rese disponibili due tipi di istruzioni condizionali:

- Istruzione di **SCELTA** (semplice) o **ALTERNATIVA**
- Istruzione di **SCELTA MULTIPLA** (questa non è essenziale, ma migliora l'espressività del linguaggio)

Istruzione di scelta semplice (o Alternativa)

- **Consente di scegliere fra *due istruzioni alternative* in base al verificarsi di una particolare *condizione***

```
<istruzione-di-scelta> ::=
        if (condizione)   < istruzione1>
        [  else  < istruzione2> ]
```

Opzionale

- ***Condizione* è un'espressione logica** che viene valutata al momento dell'esecuzione dell'istruzione **if**
- Se *condizione* risulta vera si esegue <istruzione1>, altrimenti si esegue <istruzione2>
- In entrambi i casi l'esecuzione continua poi con l'istruzione che segue l'istruzione **if**.

  NOTA: Se *condizione* è falsa e la parte **else** (opzionale) è omessa, si passa subito all'istruzione che segue l'istruzione **if**

Le due istruzioni di scelta semplice

```
if (condizione)   istruzione1;


if (condizione)
        istruzione1;

if (condizione)   istruzione1;
else  istruzione2;

if (condizione)   istruzione1;
else
      istruzione2;
```

Esempio

```
int a=3, n=-6, b=0;
if (n > 0)
        a = b + 5;
else
        n = b*5;
```

----

a → ?

b → ?

n → ?

Problema

- **E se vogliamo eseguire più di una istruzione in uno dei due rami o in entrambi?**
- **Esempio:**

if (*condizione*)
<varie *istruzioni*>

else
<varie *istruzioni*>

Istruzioni Composte

Istruzione composta

- Sequenza di istruzioni racchiuse tra parentesi graffe

```
{
    <istruzione1>
    <istruzione2>
    ...
}
```

- E' un caso particolare di *blocco*
- Ovunque la sintassi preveda una istruzione si può inserire una istruzione composta
- Ai fini della sintassi e della semantica è trattata come una istruzione singola
- L'esecuzione di una istruzione composta implica l'esecuzione ordinata di tutte le istruzioni della sequenza

Istruzioni Condizionali
Seconda parte

Forma completa

if (*condizione*)
<istruzione>

[else **<**istruzione>]

- *istruzione* può essere qualsiasi istruzione, ovvero espressione, condizionale, iterativa, composta (blocco di istruzioni)

Esempio

- Qualora occorra specificare più istruzioni, si dovrà quindi utilizzare un *blocco*

```
if (n > 0)
    {                          /* inizio blocco */
        a = b + 5;
        c = x + a - b;
    }                          /* fine  blocco */
else n = b*5;
```

Osservazione 2

Istruzioni *if* annidate

- Come caso particolare, <istruzione1> o <istruzione2> potrebbero essere un'altra <istruzione-di-scelta>
- In questo caso *occorre attenzione ad associare le parti* ***else*** *(che sono opzionali) all'* ***if*** *corretto*

In base alla sintassi del linguaggio C, l'*else* è sempre associato all'*if* più interno

Se questo comportamento non soddisfa o crea ambiguità, occorre inserire esplicitamente un blocco **{ }**

Osservazione 2 (*cont.*)

```
?  if (n > 0)
        if (a>b) n = a;
   else n = b*5;
```

/* else riferito a if (a>b) */

Per far sì che l'else si riferisca al primo if →

```
if (n > 0)
     { if (a>b) n = a; }
else n = b*5;
```

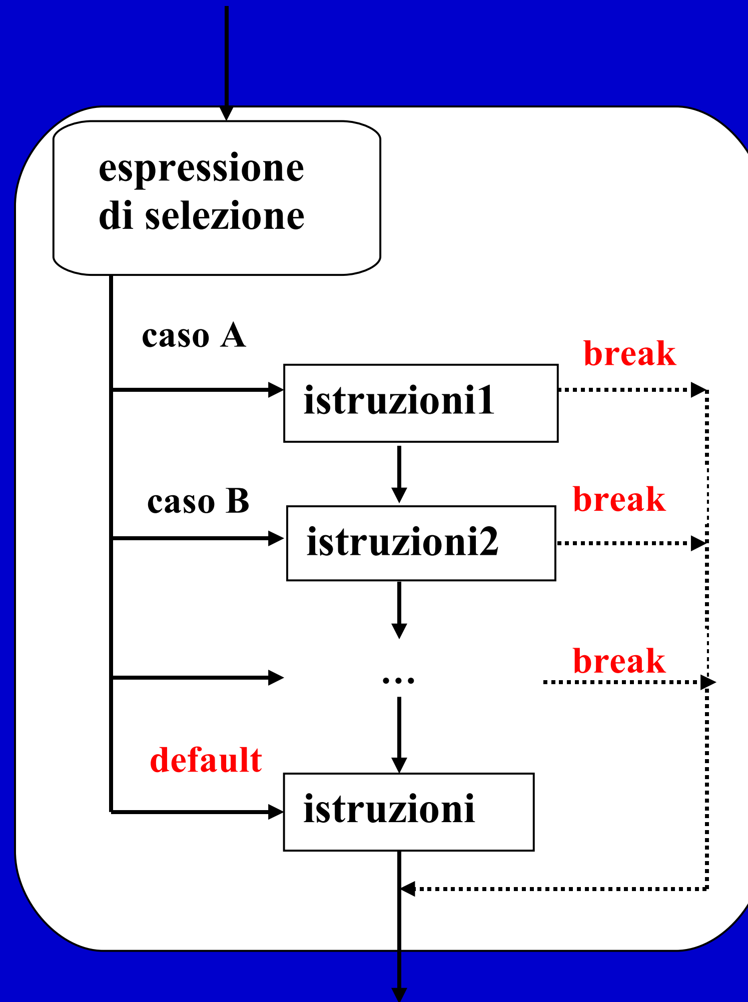
Per leggibilità, usare sempre parentesi →

```
if (n > 0)
     {    if (a>b) n = a;
          else n = b*5;   }
```

Istruzione di scelta multipla: switch

- Consente di scegliere fra ***molte istruzioni*** in base al valore di un'*espressione di selezione*
- L'espressione di selezione deve denotare un **valore numerabile** (*intero*, *carattere*,…)
- Il valore di tale espressione viene confrontato con le costanti che etichettano i vari casi: l'esecuzione prosegue dal ramo dell'etichetta corrispondente (se esiste)
- Se nessuna etichetta corrisponde al valore dell'espressione, si segue il ramo **default** (se specificato)
- Se neanche questo esiste, l'esecuzione prosegue con l'istruzione successiva all'istruzione **switch**

Istruzione di scelta multipla (*cont.*)



Istruzione di scelta multipla (sintassi)

<istruzione-di-scelta-multipla> ::=

switch (*espressione di selezione*)

{

case <*etichetta1*> **:** < istruzioni> [**break;**]

case <*etichetta2*> **:** < istruzioni> [**break;**]

…

[**default :** < istruzione>]

}

***espressione di selezione* è un'espressione che denota un valore numerabile** che viene valutata al momento dell'esecuzione dell'istruzione **switch**

Tutte le costanti <*etichetta*> devono essere dello stesso tipo dell'espressione di *selezione*

Esempio

```
int a, n;
cin>>a>>n;
switch (n)
        {
            case 1:
                cout<<”Ramo A”<<endl;
                break;
            case 2:
                cout<<”Ramo B”<<endl;
                a =  a*a;
                break;
            case 3:
                cout<<”Ramo C”<<endl;
                a = a*a*a;
                break;
            default: a=0;
        }
cout<<a<<endl;
```

Osservazioni

- <istruzioni> denota una *sequenza* di istruzioni per cui *non è necessario un blocco* per specificare più istruzioni
- I vari rami ***non* sono mutuamente esclusivi:** una volta imboccato un ramo, l'esecuzione *prosegue in generale con le istruzioni dei rami successivi*
- Per avere rami mutuamente esclusivi occorre **forzare esplicitamente l'uscita** mediante l'istruzione **break**

Esempio

```
int a, n, b = 1;
cin>>a>>n;
switch (3 - n)
  {
        case 0:
              b *=  a;
        case 1:
              b *= a;
        case 2:
              b *= a;
              break;
        default:
              a=0;
     }
cout<<a<<endl;
```

Pro e contro della scelta multipla

- L'istruzione **switch** garantisce maggiore leggibilità rispetto all'**if** quando c'è da scegliere tra più di due alternative

- Tuttavia:
  - è utilizzabile solo con espressioni ed etichette di tipo **numerabile** (int, char)
  - **non è utilizzabile** con numeri **reali** (float, double) o con tipi **strutturati** (stringhe, vettori, strutture...)