Introduzione alla programmazione C++

Roberto Borelli

Aprile 2022

Introduzione alla programmazione C++

Obiettivi:

1. Rispondere alle domande chiave.
2. Introdurre i concetti fondamentali.
3. Dei punti di seguito presentati **non deve** rimanere alcun dubbio, altrimenti sarà impossibile comprendere argomenti più avanzati.

Outline:

1. Struttura di un programma.
2. Funzioni per l'input e l'output.
3. Variabili, tipi, e operazioni.
4. Esecuzione condizionale (if).
5. I cicli (for e while).

Introduzione (1)

1. Cosa vuol dire programmare?
2. Cos'è un linguaggio di programmazione?

Introduzione (2)

Programmare significa dare una serie di istruzioni al calcolatore, che in un secondo momento le eseguirà.
Il linguaggio di programmazione è il linguaggio che ci mette a disposizione costrutti e comandi con cui possiamo costruitre i programmi.
I linguaggi di programmazione hanno una **sintassi** precisa, ogni singolo carattere di ogni singola istruzione va scritto con accuratezza.
Il calcolatore però sà solo interpretare zeri ed uni...
Il nostro programma sarà quindi un file di testo contenente delle istruzioni, il programma verrà poi **compilato** (ovvero trasformato in zeri ed uni) e sarà creato un file **eseguibile** che poi potremo eseguire sulla nostra macchina.

La struttura base di un programma

1. Corpo di un programma.
2. Istruzioni.
3. Commenti a singola linea e multi-linea.
4. Flusso di esecuzione.
5. Librerie.

Una prima istruzione: stampa (1)

1. La libreria `<iostream>`
2. Funzione per la stampa.
3. Andare a capo.
4. Concatenare delle stampe.

Una prima istruzione: stampa (2)

```
1 #include<iostream>
2 using namespace std;
3 int main(){
4    cout << "Ciao mondo. ";
5    cout << "Hello world!";
6    cout << endl;
7    cout << "Mi chiamo:" << endl << "Mario Rossi";
8    return 0;
9 }
```

Rappresentare i dati (1)

1. Il concetto di tipo
2. Numeri interi
3. Numeri con la virgola
4. Caratteri
5. Stringhe
6. Valori booleani

Rappresentare i dati (2)

- Ad ogni tipo di dato corrisponde una diversa rappresentazione in memoria.
  Il numero intero 10 viene rappresentato in memoria come:
  `0000 1010`
- Per i caratteri si usa la tabella ASCII. Il carattere 'A' corrisponde alla rappresentazione binaria:
  `0100 0001`

Rappresentare i dati (3)

- Ma anche il numero 65 in binario vale `0100 0001`.
- Quindi se in una zona di memoria trovo `0100 0001` si tratta del carattere 'A' o si tratta del numero 65???
- Devo sapere di che dato si tratta! I tipi ci permettono di interpretare correttamente i dati.
  `0100 0001` interpretato come numero vale 65.
  `0100 0001` interpretato come carattere vale 'A'.

Le variabili (1)

1. Cosa sono?
2. A cosa servono?
3. Dichiarazione.
4. Operatore di assegnazione.

Le variabili (2)

Sono un costrutto che ci permette di memorizzare dati. Possiamo vederle come delle scatole in cui mettere oggetti. Più nel dettaglio una variabile è composta da:

1. **Nome** (o identificatore): indica un area della memoria del calcolatore.
2. **Valore**: il valore memorizzato in quell'area di memoria.
3. **Tipo**: indica quali valori possiamo memorizzare. (In una scatola di scarpe possiamo mettere solo scarpe).
   Come visto prima il tipo indica anche come verranno rappresentati i dati in bit.

Le variabili (3)

Dichiarare una variabile significa creare un associazione tra il nome e un area di memoria.
Assegnare un valore significa memorizzarlo nello spazio di memoria della variabile.
Le variabili prima di essere usate in un programma, vanno dichiarate! Altrimenti staremmo usando un nome che il computer non saprebbe come interpretare.

Le variabili (4)

- `int x;`
  Viene creata una variabile di nome `x` che conterrà numeri interi.
- `x = 2;`
  Si legge: in `x` memorizzo il valore 2.
- `x = x + 1;`
  Prima viene valutata l'espressione `x+1` e poi il risultato viene memorizzato in `x`. Quindi ora `x` vale 3.
- `var = expr;`
  1. Viene valutato il valore di `expr`.
  2. Il risultato viene memorizzato in `var`.
- **Nota:** `expr` deve avere lo stesso tipo di quello dichiarato per `var`.

Altro sulla stampa.

1. Stampare il valore delle variabili.
2. Stampare dei valori.
3. Stampare il risultato di operazioni (slide successiva).

Operatori matematici.

Si applicano a valori e variabili:

1. x + y (Somma)
2. x - y (Sottrazione)
3. x * y (Moltiplicazione)
4. x / y (Divisione)
5. x % y (Resto della divisione x/y)

Si applicano a variabili:

1. x++ (Incremento della variabile x di 1)
2. x-- (Decremento della variabile x di 1)
3. x+=z (Incremento della variabile x del valore z)

Si possono combinare le varie operazioni con le parentesi tonde.

Operatori matematici. Esempi.

Esercizio.

Programma che stampa il resto della divisione per 5 del numero 5769.

Esercizio.

Calcolare:

1. 1.4*2
2. 4.0/5
3. 3%2
4. 4/0
5. 4/5

4/5 = 0. Operazioni tra valori dello stesso tipo.

Se le operazioni avvengono tra valori dello stesso tipo, restituiscono un valore sempre di quel tipo. Ad esempio:

```
int   + int   => int
float + float => float
```

In particolare:

```
int   / int   => int
```

Quindi:

```
4 / 5 => 0
```

4/5 = 0. Operazioni tra valori di tipi diversi.

Quando invece eseguiamo operazioni tra int e float, il risultato è di tipo float. Quindi:

```
int   + float => float
int   * float => float
int   / float => float
float / int   => float
```

Quindi:

```
4.0 / 5.0 => 0.8
4.0 / 5   => 0.8
  4 / 5.0 => 0.8
```

4/5 = 0. Conversione esplicita di tipo.

In conclusione, dividendo due numeri interi, per ottenere il risultato con la virgola, vorremmo che almeno uno dei due numeri fosse di tipo float. Possiamo dire al programma di interpretare uno degli operandi (o tutti e due) come float, semplicemente scrivendo `(float)`. Ad esempio:

```
(float)4 / 5          => 0.8
(float)4 / (float)5 => 0.8
       4 / (float)5 => 0.8
```

Funzioni per l'inserimento di dati.

- Come avere dell'input dall'utente?

Esercizio.

Programma che prende in input un numero e ne stampa il doppio.

Operatori logici.

Si applicano a valori o variabili di tipo primitivo (`int`, `float`, `char`, ecc...):

1. == (Uguale)
2. != (Diverso)

Si applicano a valori o variabili di tipo `bool`:

1. && (And)
2. || (Or)
3. ! (Not)

Operatori logici. Esempi.

Esercizio.

`bool A = true; bool B = false; bool C = true; bool D = true;` Calcolare:

1. (C || (3 == 2))
2. B && (C || B)
3. (3 != 2) && (5 != 3)
4. (A || (C && (D || ('c' == 'c' && 4 != 4))))

Esercizio.

`int x = 2; char c = 'm'; int y = 3;` Calcolare:

1. (x == 2) && (y != 3)
2. (c == 'n') || true
3. false || ((c == 'm') && (y == 3))

Controllo del flusso (1)

Esecuzione condizionale:

- Abbiamo bisogno di un costrutto che ci permetta di fare delle scelte.

Controllo del flusso (1)

Esecuzione condizionale:

- Abbiamo bisogno di un costrutto che ci permetta di fare delle scelte.
- Il costrutto if.

```
1   if(condizione){
2     // istruzione
3   }
```

Controllo del flusso (2)

Ripetere delle istruzioni:

- Maniera banale? Ripetiamo il codice tante volte.

Controllo del flusso (2)

Ripetere delle istruzioni:

- Maniera banale? Ripetiamo il codice tante volte.
- Il ciclo for.

```
1  for(inizializzazione; condizione; cambiamento){
2    // istruzione
3  }
```

```
1  // istruzioni precedenti
2
3  for(inizializzazione; condizione; cambiamento){
4    // istruzioni interne
5  }
6
7  // istruzioni sucessive
```

```
1  // istruzioni precedenti
2
3  for(inizializzazione; condizione; cambiamento){
4    // istruzioni interne
5  }
6
7  // istruzioni sucessive
```

1. Vengono eseguite tutte le istruzioni precedenti al ciclo.
2. Il programma vede il ciclo for.
3. Viene eseguita l'istruzione di inizilizzazione.
4. Se la condizione è vera:
   - Vengono eseguite tutte le istruzioni interne al ciclo.
   - Viene eseguita l'istruzione di cambiamento.
   - Si torna al punto 4.
5. Altrimenti se la condizione è falsa:
   - Termina il ciclo for.
   - Vengono eseguite tutte le istruzioni successive al ciclo.

Controllo del flusso (3)

Ripetere delle istruzioni (altro modo):

- Il ciclo while.

```
1  while(condizione){
2    // istruzione
3  }
```