4. IO e variabili in C++

Andrea Marongiu

(andrea.marongiu@unimore.it)

Paolo Valente

UNIMORE

UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Ingresso e uscita

- **Input/Output**
  - Ingresso di informazioni (da elaborare) all'interno di un processo
  - Uscita di informazioni (elaborate) da un processo

- **Esempio**: stampa di informazioni sullo schermo, lettura di valori da tastiera

Ingresso e uscita in C++

- Il linguaggio C++ non prevede istruzioni per l'ingresso/uscita
- Implementato mediante oggetti di libreria chiamati ***stream***
  - **stream**: flusso di caratteri
  - **ostream**: fusso di caratteri in uscita, output *formattato*
  - **istream**: fusso di caratteri in ingresso, input *formattato*

Flussi di caratteri (1/4)

- Flusso di caratteri
  - Contenuto tra doppi apici
  - Successione di righe
  - Ciascuna costituita da zero o più caratteri
  - E terminata dal carattere speciale newline (\n)

- Esempio:

```
"Questa è una stringa\n
\tcon testo formattato.\n"
```

Flussi di caratteri (2/4)

- I caratteri non visualizzabili (caratteri speciali) possono essere rappresentati mediante **sequenze di controllo** (escape sequence)

- `\n` *newline*
- `\t` *tabulazione*
- `\\` barra *inversa*
- `\'` *apice*
- `\”` *virgolette*

Flussi di caratteri (3/4)

- Quando un programma C++ inizia la propria esecuzione ci sono tre fussi di caratteri già aperti:
  - ***cin***: fusso standard di ingresso
  - ***cout***: fusso standard di uscita
  - ***cerr***: fusso standard di uscita per comunicare messaggi di errore

Flussi di caratteri (4/4)

- Se il programma è invocato da una *shell* Unix
  - Lettura da ***cin***:
    - Lettura dei caratteri immessi dal terminale in cui gira la *shell* (tastiera)
  - Scrittura su ***cout*** o su ***cerr***:
    - Visualizzazione sul terminale in cui gira la *shell*

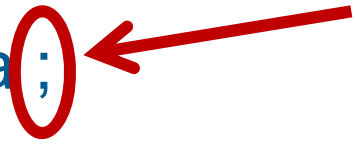
Operatore di uscita (<<)

- Scrittura formattata su ***cout***
- Forma più semplice:
  - ***cout***<<stringa ;
  - ove stringa è una sequenza di caratteri delimitata da doppi apici “

```
“Esempio di stringa"
```

Operatore di uscita (<<)

- Scrittura formattata su ***cout***
- Forma più semplice:
  - ***cout***<<stringa;



Nota il carattere terminatore di istruzione (;) Ogni istruzione C/C++ è terminata con questo carattere

  - ove stringa è una sequenza di caratteri delimitata da doppi apici “

```
“Esempio di stringa"
```

Operatore di uscita (<<)

- Scrittura formattata su ***cout***
- Gli operatori di uscita **<<** possono essere accodati l'uno all'altro

```
cout<<obj1<<obj2<<... ;
```

- Gli argomenti verranno stampati l'uno di seguito all'altro
- Il generico oggetto da stampare può essere una stringa, un *manipolatore* o una *variabile*
  - Vedremo tra qualche slide di cosa si tratta..

Manipolatori

- Oggetti che possono essere passati all'operatore di uscita
- Modificano in qualche modo la formattazione dell'ingresso/uscita
- Esempio:
  - **`endl`**: equivalente alla sequenza di controllo **`\n`**

```
cout<<obj1<<obj2<<...<<endl ;
```

Operatore di input (>>)

- Operatore di ingresso >> applicato ad un oggetto di tipo *istream*

```
cin>>nome_variabile ;
```

- Legge i caratteri in ingresso dallo standard input (abbreviato *stdin*)
  - ovvero la tastiera
- Li interpreta in base al *tipo* della variabile
- Assegna il valore letto alla variabile di nome `nome_variabile`

Operatore di input (>>)

Esempio di interpretazione

- **`cin`**`>>a;`
- La variabile a è di tipo **`int`**
- Se l'utente scrive 23 e va a capo, si leggono i caratteri 2, 3 e **`\n`**
- Vengono interpretati come le due cifre decimali del numero 23
- Il numero 23 viene memorizzato nella variabile a

Operatore di input (>>)

Ingresso inconsistente

- Cosa accade se la sequenza di caratteri letta non rappresenta alcun numero in notazione decimale?
- La lettura **fallisce** e l'oggetto cin entra in **stato di errore**
- Le successive letture falliranno
- Vedremo in futuro come resettare lo stato dello *stream* per non fare più fallire le successive letture

iostream

- Un programma che deve effettuare input/output 'classico' deve contenere le direttive

```
#include <iostream>
using namespace std;
```

- Tali direttive devono precedere il primo punto in cui viene efettuato l'input/output

Struttura di un programma

```
#include <iostream>
using namespace std;
```

direttive

```
int main()
{
   cin >> … ;
   cout << … ;

   return 0;
}
```

Struttura di un programma

```
#include <iostream>
using namespace std;


int main()
{
  cin >> … ;
  cout << … ;

  return 0;
}
```

istruzioni
(che usano iostream)

Ritorno valore finale

- Un *processo linux* (cioè, un programma in esecuzione) ritorna un valore quando termina
- Tale valore può essere letto ed utilizzato dal programma che lo ha invocato
  - *0 indica tipicamente che tutto è andato bene*
  - *Altri valori codificano un messaggio d'errore*

- La funzione **`main`** dovrebbe sempre "*ritornare*" un valore di tipo intero
- Il valore di ritorno della funzione **`main`** è il valore ritornato dal processo stesso

Struttura di un programma

```
#include <iostream>
using namespace std;


int main()
{
   cin >> … ;
   cout << … ;

   return 0;
}
```

Variabili

- Le *celle di memoria* non sono visibili in un programma C/C++.

- Il loro utilizzo è *astratto* tramite le **variabili**
  - Dei *contenitori* in cui *memorizzare* valori (dati)
  - Il contenuto di una variabile può cambiare durante l'esecuzione del programma

Variabili

- Sono associate ad uno specifico ***tipo*** di dato
- **Esempio:** in un programma C/C++ si possono scrivere dei numeri interi
  - **`6,  12,  700`**
- Tramite l'uso di *variabili* di tipo **`int`**
  - NOTA: Le variabili **`int`** in realtà possono contenere solo un sottoinsieme limitato dei numeri interi
  - come vedremo meglio in seguito

Definizione di una variabile

- In C/C++ è necessario elencare ogni variabile che sarà utilizzata nel programma, prima di utilizzarla
- In particolare si dice che bisogna **definire** ciascuna variabile. All'atto della definizione bisogna attribuire alla variabile
  - un tipo
  - un nome (identificatore) col quale ci si riferirà poi a tale variabile
  - eventualmente un valore iniziale (inizializzazione)

Esempi

- Ecco due esempi di definizione di variabili di tipo **int**

```
int a;   // definizione di una
         // variabile di nome a e di
         // tipo int

int k=5; // definizione di una
         // variabile di nome k e di
         // tipo int, inizializzata col
         // valore 5
```

Esempi

- Ecco due esempi di definizione di variabili di tipo **int**

```
int a;   // definizione di una
         // variabile di nome a e di
         // tipo int

int k=5; // definizione di una
         // variabile di nome k e di
         // tipo int, inizializzata col
         // valore 5
```

Una frase preceduta da una doppia sbarra (//) è interpretata dal compilatore C++ come un *COMMENTO*. Serve per aumentare la leggibilità del programma, ma non ne altera il funzionamento.

Valore iniziale

- Che valore assume una variabile se non viene inizializzata?
- Per il momento diciamo che assume un valore **casuale**
- Poi vedremo meglio i singoli casi

Nota sulla sintassi

- Nella descrizione della sintassi del linguaggio C/C++ utilizzeremo la notazione con parentesi quadre *[…]* per denotare elementi **opzionali**, ossia parti che possono o meno comparire
- Tutto ciò che non sarà contenuto tra tali parentesi *[…]* quadre sarà **obbligatorio**

Sintassi definizione variabile

- Sintassi della definizione di una variabile:
  - *nome_tipo nome_variabile [= valore_iniziale]* ;
- E' possibile raggruppare le definizioni di più variabili dello stesso tipo in una lista separata da ,
  - Forma generale definizione variabili:
  - *nome_tipo nome_variabile1 [=valore_iniziale],*
    nome_variabile2 [= valore_iniziale],
    ... ;

Completamento esempi

```
int a, c;        // definizione di due
                 // variabili di nome
                 // a e c di tipo int

int k=5, d;      // definizione di due
                 // variabili di nome k e d,
                 // di tipo int, di cui la
                 // prima è inizializzata
                 // col valore 5
```

- Vedremo successivamente esempi di definizioni di variabili di tipo diverso da **int**

Visibilità di una variabile

- Una variabile è **visibile**, ossia può essere utilizzata, solo a partire dal punto in cui viene definita nel testo del programma

Istruzione semplice

- Una definizione è di fatto una istruzione del C/C++
- In particolare si tratta di una cosiddetta istruzione semplice

Assegnamento

- Si può assegnare un nuovo valore ad una variabile mediante una **istruzione di assegnamento**

nome_variabile **=** *espressione* **;**

Esempi:

```
int v = 3;  // definizione variabile v
cout<<v<<endl;
v = 4;          // assegna il valore 4
                // alla variabile v
```