Corso “Programmazione 1”
Capitolo 02: Variabili, Costanti, Tipi

Docente: Marco Roveri - marco.roveri@unitn.it
Esercitatori: Giovanni De Toni - giovanni.detoni@unitn.it
Stefano Berlato - stefano.berlato-1@unitn.it

C.D.L.: Informatica (INF)
A.A.: 2021-2022
Luogo: DISI, Università di Trento
URL: https://bit.ly/2VgfYwJ

Ultimo aggiornamento: 15 settembre 2021

Terms of Use and Copyright

The material (text, figures) in these slides is authored mostly by Roberto Sebastiani, with contributions by Marco Roveri, Alessandro Armando, Enrico Giunchiglia e Sabrina Recla.

Le Variabili e le Costanti

- Per memorizzare un valore in un'area di memoria si utilizzano entità chiamate variabili o costanti.
- Le variabili permettono la modifica del loro valore durante l'esecuzione del programma.
- L' area di memoria corrispondente è identificata da un nome, che ne individua l'indirizzo in memoria.

Le Variabili

Le variabili sono caratterizzate da una quadrupla:

- il nome (identificatore)
- il tipo
- la locazione di memoria (l-value o indirizzo)
- il valore (r-value)

Definizione e Dichiarazione di Variabili I

- Definizione:
  - Formato: `tipo identificatore;`
  - Esempio: `int x;`
  - Se (e solo se!) una variabile è definita esternamente ad ogni funzione, `main()` inclusa, (variabile globale) viene automaticamente inizializzata al valore 0
- Definizione con inizializzazione:
  - Formato: `tipo identificatore=exp;`
  - Esempio: `int x=3*2;`
  - Esempio: `int y=3*z; // z definita precedentemente`
- Dichiarazione:
  - Formato: `extern tipo identificatore;`
  - Esempio: `extern int x;`

output di variabile

Per produrre in output il valore di una variabile x, si usa l'istruzione:
`cout << x ...`

Definizione e Dichiarazione di Variabili II

- Definizione: quando il compilatore incontra una definizione di una variabile, esso predispone l'allocazione di un'area di memoria in grado di contenere la variabile del tipo scelto
- Dichiarazione: specifica solo il tipo della variabile e presuppone dunque che la variabile venga definita in un'altra parte del programma

Esempio di definizioni di variabili:
{ ../ESEMPI_BASE/variabili.cc }
... con inizializzazione:
{ ../ESEMPI_BASE/variabili2.cc }

Dichiarazione di Costanti

- Sintassi:
  - Formato: `const tipo identificatore = exp;`
  - `exp` deve essere una espressione il cui valore deve poter essere calcolato in fase di compilazione
    (su alcuni compilatori è possibile inizializzare costanti a valore non costanti, ma il risultato è imprevedibile e varia a seconda dei casi $\Longrightarrow$ da evitare assolutamente)
- Esempi

```
const int kilo = 1024;
const double pi = 3.14159;
const int mille = kilo - 24;
```

Esempio di uso di costanti:
{ `../ESEMPI_BASE/costanti.cc` }

Concetto di stream

- Un programma comunica con l'esterno tramite uno o più flussi di caratteri (stream)
- Uno stream è una struttura logica costituita da una sequenza di caratteri, in numero teoricamente infinito, terminante con un apposito carattere che ne identifica la fine.
- Gli stream vengono associati (con opportuni comandi) ai dispositivi fisici collegati al computer (tastiera, video) o a file residenti sulla memoria di massa

Stream predefiniti

- In C++ esistono i seguenti stream predefiniti
  - cin (stream standard di ingresso)
  - cout (stream standard di uscita)
  - cerr (stream standard di errore)
- Le funzioni che operano su questi stream sono in una libreria di ingresso/uscita e per usarle occorre la direttiva:
  #include <iostream>

Stream di uscita

- Lo stream di uscita standard (`cout`) è quello su cui il programma scrive i dati
  - è tipicamente associato allo schermo
- Per scrivere dati si usa l'istruzione di scrittura:
  `stream << espressione;`
- L'istruzione di scrittura comporta:
  - il calcolo del valore dell'espressione
  - la conversione in una sequenza di caratteri
  - il trasferimento della sequenza nello stream

Nota:

La costante predefinita `endl` corrisponde ad uno `'\n'`, per cui viene iniziata una nuova linea con il cursore nella prima colonna

Scritture multiple

- L'istruzione di scrittura ha una forma più generale che consente scritture multiple, nel formato

  ```
  cout << espressione1 << espressione2 << ...
  ```

- Ad esempio,

  ```
  cout << x << y << endl;
  ```

  corrisponde a:

  ```
  cout << x;
  cout << y;
  cout << endl;
  ```

Esempio di uso di operazioni di output:

{ ../ESEMPI_BASE/esempio_cout.cc }

... con output multiplo:

{ ../ESEMPI_BASE/esempio_cout_multiplo.cc }

Stream di ingresso

- Lo stream di ingresso standard (`cin`) è quello da cui il programma preleva i dati
  - è tipicamente associato alla tastiera
- Per prelevare dati si usa l'istruzione di lettura:
  `stream >> espressione;`
  dove `espressione` deve essere un'espressione dotata di indirizzo (per ora, una variabile)
- L'istruzione di lettura comporta in ordine:
  1. il prelievo dallo stream di una sequenza di caratteri
  2. la conversione di tale sequenza in un valore che viene assegnato alla variabile

Lettura di un carattere da cin

- Consideriamo l'istruzione `cin >> x`, dove `x` è di tipo `char`

  □□□□*a*□□-14.53□□728□□ ...

  ⇑

  (qui "□" rappresenta uno spazio e "⇑" il cursore)
- se il carattere puntato dal cursore è una spaziatura:
  - il cursore si sposta in avanti per trovare un carattere che non sia una spaziatura
- se il carattere puntato dal cursore non è una spaziatura:
  1. il carattere viene prelevato
  2. il carattere viene assegnato alla variabile x
  3. il puntatore si sposta alla casella successiva

Lettura di un numero da cin

- Consideriamo l'istruzione `cin >> x`, dove `x` è un numero (ad esempio di tipo `int`)
- se il carattere puntato dal cursore è una spaziatura:
  - come nel caso precedente
- se la sequenza di caratteri è interpretabile come un numero compatibile con il tipo di `x`:

  `□□□□a□□-14.53□□728□□ ...`
  ⇑

  1. la sequenza di caratteri viene prelevata
  2. la sequenza viene convertita nel suo corrispondente valore (es, il valore intero `728`) che viene assegnato alla variabile x
  3. il puntatore si sposta alla casella successiva
- altrimenti, l'operazione di prelievo non avviene e lo stream si porta in uno stato di errore

Letture multiple

- L'istruzione di ingresso ha una forma più generale che consente letture multiple, nel formato
  `cin >> var1 >> var2 >> ...`
- Ad esempio, se `x`, `y` , `z` sono risp. `char`, `double` e `int`:

```
cin >> x >> y >> z;
```

corrisponde a:

```
cin >> x;
cin >> y;
cin >> z;
```

```
□□□□a□□-14.53□□728□□ ...
  ⇑
```

Esempi

Esempio di uso di operazioni di input:
{ `../ESEMPI_BASE/esempio_cin.cc` }
... con input multiplo:
{ `../ESEMPI_BASE/esempio_cin2.cc` }
... prima intero e poi reale:
{ `../ESEMPI_BASE/esempio_cin3.cc` }

Alcune funzioni utili della libreria `<iostream>`

- `cin.eof()`: ritorna un valore diverso da 0 se lo stream cin ha raggiunto la sua fine (End Of File)
  - va usato sempre dopo almeno un'operazione di lettura
  - richiede un separatore dopo l'ultimo elemento letto
- `cin.fail()`: ritorna un valore diverso da 0 se lo stream cin ha rilevato uno stato di errore (e.g. stringa per `int`) o un end-of-file
  - non necessariamente usato dopo almeno un'operazione di lettura
  - non richiede un separatore dopo l'ultimo elemento letto
- `cin.clear()`: ripristina lo stato normale dallo stato di errore

Fine Lezione 02