



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

## Istruzioni, dichiarazioni, definizioni (\*ioni)

Stefano Ghidoni



DIPARTIMENTO  
DI INGEGNERIA  
DELL'INFORMAZIONE



# Agenda

- Istruzioni
- Dichiarazioni e definizioni
- Inizializzazione



# Statements

- Un'istruzione o statement è una parte di codice C++ che specifica un'azione
  - **Non sono istruzioni le direttive del preprocessore (es., #include)**
- Termina con ;
  - Necessario per interpretare il codice
- Vari tipi di istruzioni
  - Qualche esempio?



# Tipi di istruzioni

- Expression statements
  - L'istruzione è la valutazione dell'espressione
- Dichiarazioni
- Selezione (if-statement, switch-statement)
- Iterazione (while-statement, for-statement)
- ...

Istruzioni per gestire le  
variabili (e non solo):  
dichiarazioni e definizioni



# Dichiarazioni

- Dichiarazione: istruzione che **introduce un nome in uno scope**
  - Specifica il nome
  - Specifica il tipo
  - Opzionalmente, specifica un inizializzatore (valore iniziale)
- Un nome deve essere dichiarato prima di essere usato
- Dichiarazione di:
  - Variabili
  - Funzioni
  - ... (molto altro)

es: creando il main e una variabile in esso, questa esiste fino alla fine del main; è la porzione di codice in cui la variabile è stata dichiarata.



# Dichiarazioni

- Esempi di dichiarazioni – per ciascuna delle seguenti, verificate:
  - Specifica il nome?
  - Specifica il tipo?
  - Opzionalmente, specifica un inizializzatore (valore iniziale)?

```
int a = 7;
const double cd = 8.7;
double sqrt (double);           // una funzione con un
                                // argomento double
vector<Token> v;               // vettore di variabili Token
```

→ il tipo è "vettore di Token"



# Dichiarazioni

- Una dichiarazione ha effetto sulla memoria?

```
int a = 7;
const double cd = 8.7;
double sqrt (double);           // una funzione con un
                                // argomento double
vector<Token> v;              // vettore di variabili Token
```

Una dichiarazione è "quando un nome entra in circolo".

Nel terzo esempio, la dichiarazione della funzione, non vi sono effetti sulla memoria.  
Quindi non ha sempre effetto sulla memoria.



# Definizioni

- Definizione: una dichiarazione che specifica completamente l'entità dichiarata
  - Una definizione è anche una dichiarazione



```
int a = 7;  
vector<double> v;  
double sqrt (double d) { /* ... */ }
```

- Alcune dichiarazioni *non* sono definizioni

```
double sqrt (double);  
extern int a;
```

→ Esiste una variabile "a" dichiarata in un altro file, dove le è stata riservata la memoria.  
In questo modo viene riconosciuto il simbolo ma non gli viene riservata altra memoria.  
Non è consigliato usarlo



# Definizioni

- Una definizione:
  - "Fornisce tutte le informazioni necessarie per creare l'entità nella sua interezza"
  - Può avere un significato diverso a seconda dell'entità definita:
    - **Variabile:** fornisce tipo e nome, riserva la memoria
    - **Funzione:** fornisce il corpo della funzione
    - **Classe:** fornisce variabili e funzioni membri della classe
- Quando un'entità è definita, per forza è anche dichiarata



# Dichiarazioni vs definizioni

- In alcuni contesti (**non in questo corso**), dichiarazione e definizione sono considerati mutuamente esclusivi
  - Dichiarazione intesa come *dichiarazione che non è anche una definizione*



# Dichiarazioni vs definizioni

- Variabili
  - Una dichiarazione fornisce il nome della variabile e il tipo (nulla è detto sulla memoria)
  - Una definizione fornisce un oggetto (riserva memoria) con un determinato nome e un determinato tipo
- Funzioni
  - Una dichiarazione fornisce il nome della funzione e il tipo (degli argomenti e del valore ritornato)
  - Una definizione fornisce anche il corpo
    - Completamente specificata
- Per le funzioni la distinzione dichiarazione vs definizione è più evidente



# Esempio con le funzioni

- Dichiarazione vs definizione di una funzione

```
int func();           // dichiarazione della funzione func

int main()
{
    int x = func(); // N.B. Utilizzo prima della
                     // definizione
}

int func()           // definizione della funzione func
{
    return 2;
}
```



# Dichiarazioni vs definizioni

- Perché esistono sia dichiarazioni che definizioni?
- Per poter usare entità prima di fornirne la definizione completa
  - Per gestire file sorgenti multipli – es: header file
    - La dichiarazione permette a tutti i file interessati di conoscere un'entità senza bisogno di conoscere i dettagli implementativi



# Dichiarazioni vs definizioni

- Dichiarazione vs definizione riflette interfaccia vs implementazione
  - Interfaccia: di cosa abbiamo bisogno per usare qualcosa (dichiarazione)
  - Implementazione: di cosa abbiamo bisogno affinché qualcosa faccia ciò che deve (definizione)



# Dichiarazioni vs definizioni

- La sola dichiarazione ci permette di riferirci a un'entità (variabile, funzione, ...)
- Spesso vogliamo la definizione altrove
  - Più sotto nel file
  - In un altro file
- Dichiarazioni raccolte in un header file permettono di condividere le funzioni



# Esempi di dichiarazioni e definizioni

```
int x = 7;                                // definizione
extern int x;                               // dichiarazione
extern int x;                               // un'altra
                                            // dichiarazione

double sqrt(double);                      // dichiarazione
double sqrt(double d) /* ... */;          // definizione
double sqrt(double);                      // altra dichiarazione
double sqrt(double);                      // altra dichiarazione

int sqrt(double);                          // errore: dichiarazione
                                            // incoerente
```

- Extern specifica che la dichiarazione non è una definizione
  - Poco usato, poco utile

La dichiarazione doppia identica è permessa anche per le funzioni.

Non vanno in conflitto perché la variabile è stata definita da qualcun'altro, quindi non sto riservando di nuovo di nuova. Non è avvenuta una double definition.



# Definizioni

- In C++ si possono definire
  - Variabili
  - Costanti
  - Funzioni
  - Namespace
  - Tipi (es., classi, enum)
  - Template

# Inizializzazione



# Dichiarazioni/definizioni e inizializzazione

- L'inizializzazione è spesso presente nelle dichiarazioni e definizioni
- Abbiamo visto più modi di inizializzare le variabili:  
=, () {},

```
int a;
double d = 7;
vector<int> vi(10);    // inizializzatore con sintassi ()
vector<int> vi2 {1, 2, 3, 4};      // inizializzatore con
                                    // sintassi {}

const int x = 7;          // inizializzatore obbligatorio
const int x2 {9};        // inizializzatore con sintassi {}
const int y;              // errore: nessun inizializzatore
```



Se non si inizializzasse una costante, non le si potrebbe attribuire alcun valore in seguito



# Dichiarazioni/definizioni e inizializzazione

```
int a;  
double d = 7;  
vector<int> vi(10);    // inizializzatore con sintassi ()  
vector<int> vi2 {1, 2, 3, 4};        // inizializzatore con  
// sintassi {}  
  
const int x = 7;          // inizializzatore obbligatorio  
const int x2 {9};        // inizializzatore con sintassi {}  
const int y;              // errore: nessun inizializzatore
```

- La sintassi {} è preferibile per l'inizializzazione
  - È più esplicita
  - Ancora non molto diffusa
- L'operatore = è usato per inizializzazioni semplici



# Variabili non inizializzate

- Che succede se non inizializziamo le variabili?
- "La ricetta per bug oscuri..." (BS)

```
void f(int z)
{
    int x;      // non inizializzato
    // ... Nessun assegnamento su x
    x = 7;
    // ...
}
```

- Sembra codice accettabile, ma può essere pericoloso!
  - Dipende dal codice nel primo "..."



# Variabili non inizializzate

- Esempio di ... problematico

```
void f(int z)
{
    int x;      // non inizializzato
    // ... Nessun asssegnamento su x
    if (x > z)
    {
        // ...
    }
    // ...
    x = 7;    assegnamento
    // ...
}
```



UNIVERSITÀ  
DEGLI STUDI  
DI PADOVA

## Istruzioni, dichiarazioni, definizioni (\*ioni)

Stefano Ghidoni



DIPARTIMENTO  
DI INGEGNERIA  
DELL'INFORMAZIONE