Introduzione al linguaggio C++

Corso di Algoritmi e Strutture Dati
Esercitazioni

Domenico Redavid

Stefano Ferilli

redavid@di.uniba.it

ferilli@di.uniba.it

Università degli Studi di Bari

A.A. 2009/2010

Il linguaggio C++

- Estensione ad oggetti del C
  - Sovrainsieme dei costrutti disponibili

Testi consigliati

- C++ Linguaggio, libreria standard, principi di programmazione – 3a ed.
  - B. Stroustrup (Addison-Wesley)
- C++ Fondamenti di programmazione
  - H.M. Deitel, P.J. Deitel (Apogeo)

Prerequisiti - Obiettivi

- Prerequisiti
  - Programmazione imperativa
    - Astrazione funzionale
  - Linguaggio C
    - Presentazione delle sovrastrutture OO
- Obiettivi
  - Operatività in tempi ridotti
    - Approfondimenti successivi

Scaletta

- I/O base
- Programmazione OO
- Costrutti OO del C++

Legenda

- Caratteri:
  - In tondo
    - A scelta del programmatore
      - Es.: identificatori
  - In **grassetto**
    - Definiti dalla sintassi
      - Es.: parole chiave
  - In *corsivo*
    - A scelta fra valori dati
      - Es.: tipi

I/O base

- `#include <iostream>` (sostituisce `#include <stdio.h>`)
- `cout<<` (sostituisce `printf`)
- `cin>>` (sostituisce `scanf`)

- Stampa

```
cout << espr;
cout << espr1 << … <<
   esprn;
```

- Lettura

```
cin >> var;
cin >> var1 >> … >>
   varn;
```

  - Gestione a righe
  - Interpretato
  - Separatori spaziatura

Formattazione Output

```
#include <iomanip>
```

- Larghezza campo di output *(n* colonne*)*

```
cout << setw(n);
```

  - Va ripetuto per ogni singolo output
- Numero di cifre decimali dopo la virgola (*n* cifre)

```
cout << setprecision(n);
```

- Visualizzazione degli zeri non significativi

```
cout << fixed;
cout << setiosflags(ios::fixed);
```

Esempio

```
cout << fixed << setprecision(2) << setw(8) << x;
```

Stringhe

```
#include <string>
```

- Sequenze di caratteri
  - Racchiuse tra doppi apici
  - Lunghezza non fissata
    - Flessibilità
    - Evita errori di “sconfinamento”

string id [**= “…”**]

Stringhe

- Lettura stringhe con spaziature

  getline(cin, id**)**

- Numero di caratteri di una stringa

  id**.length()**

  – “” stringa vuota (lunghezza 0)

- Estrazione di sottostringa

  id**.substr(***ind*, *lungh***)**

- Concatenazione di stringhe

  str_1 **+** str_2 **+** … **+** str_n

  – Almeno uno degli operandi deve essere una variabile stringa

Stringhe
Esempio

```
string s1 = "Salve,";
string s2 = "mondo!";
string saluto;


saluto = "Ba" + "ri";
```

- *NO!!!*

```
saluto = s1 + " " + s2;
```

  - Vale "Salve, mondo!"

```
saluto.length()
```

  - Vale 13

```
saluto.substr(0,5)
```

  - Vale "Salve"

```
saluto.substr(7,5)
```

  - Vale "mondo"

Stringhe
Esempio

```
string nome;
double prezzo;
int voto;
string continua
```

cin : Intel p4\n750\n8\ny\n

```
getline(cin, nome);
```
- nome : “Intel p4”
- cin : 750\n8\ny\n

```
cin >> prezzo;
```
- prezzo : 750
- cin : \n8\ny\n

```
cin >> voto;
```
- voto : 8
- cin : \ny\n

```
getline(cin, continua);
```
- continua : “”
- cin : y\n

Programmazione OO
Concetti

- Metodi di organizzazione umana per comprendere il mondo reale
  - 3 metodi che pervadono tutto il modo di pensare:
    - Differenziazione dell'esperienza in oggetti particolari e nei loro attributi
    - Distinzione tra oggetti interi e le parti che li compongono
    - Creazione di, e distinzione fra, differenti classi di oggetti

Programmazione OO
Concetti

- Scala
  - Relazione a 3 termini:
    - Parti
    - Tutto
    - Osservatore
  - Le proporzioni delle parti col tutto devono essere armonizzate con l'osservatore

Programmazione OO
Concetti

- Categorie di comportamento
  - I 3 tipi usati più spesso si basano:
    - Sulla causalità immediata
    - Sulle somiglianze della storia evolutiva (cambiamenti nel tempo)
    - Sulla somiglianza di funzioni

Programmazione OO
Concetti

- Orientato agli oggetti =
  - Classi e oggetti +
  - Ereditarietà +
  - Comunicazione tramite messaggi
- Vantaggi
  - Intuitività
  - Astrazione
  - Riuso

Programmazione OO
Principi

- Astrazione
  - Il principio di ignorare gli aspetti di un soggetto che non sono importanti per lo scopo attuale, per concentrarsi maggiormente su quelli che lo sono

Programmazione OO
Principi

- Astrazione procedurale
  - Il principio che ogni operazione che ottiene un effetto ben definito possa essere considerata dai suoi utenti come un'entità singola, nonostante tale operazione sia effettivamente realizzata da una sequenza di operazioni di livello inferiore
- Astrazione dei Dati
  - Il principio di definire un tipo di dato in termini delle operazioni applicabili agli oggetti del tipo, col vincolo che i valori di tali oggetti si possano modificare ed osservare solo usando tali operazioni

Programmazione OO
Principi

- Incapsulamento (occultamento dell'informazione)
  - Ogni componente del programma deve incapsulare, o nascondere, una singola scelta di progetto
  - L'interfaccia di ciascun modulo è definita in modo tale da rivelare il meno possibile del suo funzionamento interno
  - Utilizzato nello sviluppo della struttura completa di un programma

Programmazione OO
Principi

- Ereditarietà
  - Un meccanismo per esprimere le somiglianze tra le Classi, semplificando la definizione delle Classi simili ad una (alcune) precedentemente definita(e)
  - Rappresenta la generalizzazione e la specializzazione, rendendo espliciti gli Attributi ed i Servizi comuni entro una gerarchia o un reticolo di Classi

Programmazione OO
Principi

- Associazione
  - L'unione o la connessione fra oggetti
- Messaggio
  - Ogni comunicazione inviata tra entità

Programmazione OO
Elementi

- Classe
  - Un insieme di persone o di cose raggruppate a causa di certe somiglianze o caratteristiche comuni
    - Una descrizione di uno o più Oggetti aventi lo stesso insieme di Attributi e Servizi, compresa la descrizione di come creare nuovi Oggetti della Classe
- Classe-&-Oggetti
  - Una Classe e gli Oggetti di quella Classe

Programmazione OO
Elementi

- Oggetto (o *istanza)*
  - Una persona o una cosa verso la quale sono diretti un'azione, un pensiero o un sentimento. Qualsiasi cosa visibile o tangibile; un prodotto materiale o una materia
    - Un'astrazione di qualche cosa nel dominio del problema, che riflette la capacità del sistema a mantenere dell'informazione su di essa, ad interagire con essa o ad entrambe le cose; un incapsulamento di valori di Attributi e servizi esclusivi su tali Attributi

Programmazione OO
Elementi

- Attributo
  - Qualunque proprietà, qualità o caratteristica che si può ascrivere ad una persona o ad una cosa
    - Un dato (informazione di stato) per il quale ciascun Oggetto di una Classe ha un proprio valore
      - Stato e Cambiamento nel tempo

Programmazione OO
Elementi

- Servizio
  - Un’attività eseguita per fornire alla gente l’uso di qualcosa
    - Un comportamento specifico della cui esibizione un Oggetto è responsabile
      - Somiglianza di funzioni
      - Causalità immediata

Classi e Oggetti

- Classi ≈ Tipi di dati (complessi)
  - Rappresentano meglio le entità del mondo reale
    - Oggetti ≈ Istanze di classi
  - 2 categorie
    - Predefinite
      - int, double, …
    - Definite dal programmatore

Classi
Definizione

- Specifica dell'*interfaccia*

```
class id {
    public:
     …  // dichiarazioni dei costruttori
     …  // dichiarazioni delle funzioni membro
    private:
     …  // campi dati
};
```

  - Incapsulamento (*Information Hiding*)
    - Solo le funzioni di una classe possono accedere ai campi dati

Classi
Definizione

- Esempio

```
class Punto {
public:
Punto ();
Punto (double x, double y);
void sposta (double dx, double dy);
double get_x() const;
double get_y() const;
private:
double x;
double y;
};
```

Funzioni Membro
Definizione

tipo Classe::funzione(*par*$_1$,…,*par*$_n$) [**const**] **{**

…

};

- Funzioni di accesso dichiarate *const*
  - Non modificano il valore dei campi dati
    - Lettura del valore
    - Confronto

Funzioni Membro
Definizione

- Esempio

```
void Punto::sposta(double dx, double dy) {
x = x + dx;
y = y + dy;
};
double Punto::get_x() const {
return x;
};
double Punto::get_y() const {
return y;
};
```

Funzioni Membro
Costruttori

Classe::*Classe*(*par*$_1$,…,*par*$_n$) **{**

…

}

- Creano un nuovo oggetto
  - Istanza della classe
- Inizializzano tutti i campi dati dell'oggetto

- Stesso nome della classe

Funzioni Membro
Costruttori Predefiniti

Classe::*Classe*() **{**

. . .

}

- Inizializzano l'oggetto con valori predefiniti
  - Campi numerici da definire manualmente
  - Campi di tipo classe definiti automaticamente

Costruttori
Definizione

- Esempio

```
Punto::Punto(double asc, double ord) {
x = asc;
y = ord;
};
Punto::Punto() {
x = 0.0;
y = 0.0;
};
```

Costruttori
Uso

- Generici
  - Forma estesa

Classe id **=** *Classe***(**parametri**)**

  - Forma abbreviata

Classe id**(**parametri**)**

- Predefiniti
  - Forma estesa

Classe id **=** *Classe***()**

  - Forma abbreviata

Classe id

    - Senza parentesi!!!

Funzioni Membro
Uso

oggetto.funzione(parametri)

- Applicate a variabili oggetto
  - Oggetto a cui ci si riferisce implicito
    - Notazione punto
  - Tipicamente lettura o impostazione di campi
    - Non necessaria corrispondenza diretta col campo
      - get_
      - set_

Funzioni Membro
Uso

- Esempio

```
Punto o = Punto();          // o : (0.0,0.0)
  // oppure  Punto o;
Punto p = Punto(12.0,10.0);  // p : (12.0,10.0)
  // oppure  Punto p(12.0,10.0)

double largh = p.get_x(); // largh : 12.0
double lungh = o.get_y(); // lungh : 0.0

p.sposta(-2.0,5.0);     // p : (10.0,15.0)
```

Overloading

- Funzioni generiche
  - Uguale nome ma diverso Numero/Tipo di parametri
- Funzioni membro
  - Uguale nome ma classi diverse
- Operatori
  - Uguale simbolo ma diverse classi degli operandi

Confronto fra funzioni

- Funzioni generiche
  - Non appartengono a nessuna classe
  - Utilizzabili autonomamente
  - Parametri espliciti
- Funzioni membro
  - Definite in una specifica classe
  - Utilizzabili in una specifica classe
  - Notazione punto

Template in C++

- Costrutto per scrivere funzioni e classi molto generali che possono applicarsi a dati di tipo diverso
  - così come una classe è un modello per istanziare oggetti (della classe) a tempo d'esecuzione, un template è un modello per istanziare classi o funzioni (del template) a tempo di compilazione
- I template sono funzioni e classi generiche, implementate per un tipo di dato da definirsi in seguito
  - per utilizzarli il programmatore deve solo specificare i tipi con i quali essi debbono lavorare

Template di funzioni

- Forniscono un meccanismo per creare funzioni generiche, che possa cioè supportare simultaneamente differenti tipi di dato
- Molto utili quando bisogna utilizzare la stessa funzione con differenti tipi di argomenti

```
template <class tipo> tipo funzione (tipo arg1, tipo arg2,...)
{
   // Corpo della funzione
}
```

Dichiarazione tipica:

```
template <class T> T max (T a, T b)
{
 if (a > b)
    return a;
 else return b;
}
```

Template di classi

- Permettono di definire classi parametriche che possono gestire differenti tipi di dato

```
template <typename T>
class tipopar {
   ...
};
```

dove `T` è il nome del tipo utilizzato dal template e `tipopar` è il nome del tipo parametrizzato del template; `T` non è limitato a tipi di dato predefiniti

```
template <typename T>
struct Punto {
   T x, y;
};
...
Punto<int> pt = {45, 15};
```