Introduzione alla Programmazione ad Oggetti in C++

Lezione 1

Cosa è la Programmazione Orientata agli Oggetti

- ✓ **Metodologia per costruire prodotti software di grosse dimensioni che siano affidabili e facilmente modificabili**
  - **Tecnica di programmazione introdotta agli inizi degli anni 80**
- ✓ **Principali linguaggi utilizzati sono ad oggetti**
  - **C++, Java**
- ✓ **Le tecniche di progettazione più diffuse sono ad oggetti**
  - **database, interfacce grafiche, protocolli di rete, applicazioni Web**

Laboratorio di Algoritmi e Strutture Dati 2001/02

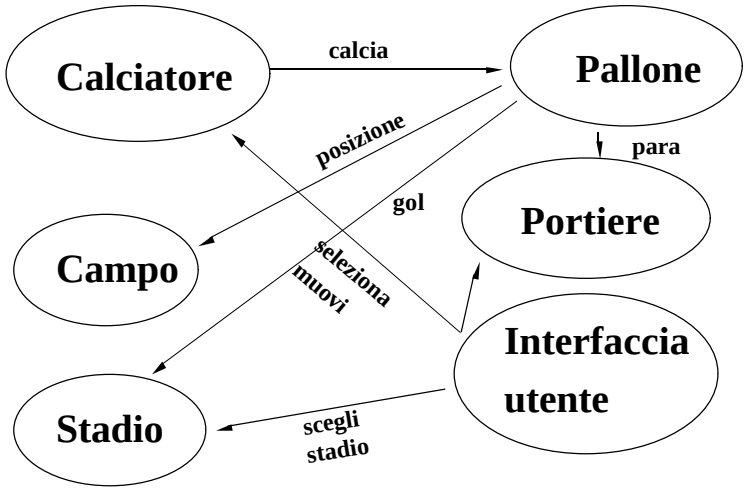
1

Idea Base della Programmazione Orientata agli Oggetti

- ✓ Un progetto è costituito da più oggetti che operano indipendentemente e interagiscono secondo modalità prefissate
- ✓ **Ogni oggetto rappresenta un elemento del dominio del problema, che può rispondere a determinati stimoli provenienti dal mondo esterno**
  - **ai fini della corretta interazione tra gli oggetti non è importante sapere come queste risposte vengono determinate**
  - **Ogni oggetto viene implementato separatamente e indipendentemente dagli altri**

Laboratorio di Algoritmi e Strutture Dati 2001/02
2

Esempio: FIFA 2004



Laboratorio di Algoritmi e Strutture Dati 2001/02
3

Perché la Programmazione Orientata agli Oggetti

- ✓ Si riesce a produrre codice più pulito ed in minor tempo
  - Aumenta produttività del programmatore
- ✓ E' più semplice mantenere il codice
  - E' possibile far fronte a nuove esigenze
- ✓ Fornisce un coerente modello di sviluppo del software
  - Maggiori possibilità di realizzare un programma che soddisfa le esigenze del cliente

Laboratorio di Algoritmi e Strutture Dati 2001/02
4

Maggiore Produttività

- ✓ La OOP (Object Oriented Programming) consente
  - **costruzione di librerie di routine che implementano funzioni comuni**
  - **utilizzo delle funzioni delle librerie indipendente dalla implementazione**
  - **facile adattabilità delle funzioni di libreria a differenti esigenze**
- ✓ **L'utilizzo sistematico di componenti software riutilizzabili aumenta enormemente la produttività dei programmatori**

Laboratorio di Algoritmi e Strutture Dati 2001/02
5

Mantenibilità Del Software

- ✓ Un programma consiste di componenti distinte ed autonome (oggetti) che interagiscono secondo interfacce ben definite
  - **dettagli della specifica localizzati negli oggetti**
  - **se una parte delle specifiche viene modificata devono essere aggiornati solo gli oggetti interessati alle modifiche**
- ✓ **La localizzazione aiuta a controllare grossi progetti e ne favorisce l'evoluzione**

Laboratorio di Algoritmi e Strutture Dati 2001/02
6

Modello di Sviluppo del Software

- ✓ **La OOP fornisce un modello coerente per descrivere le esigenze del cliente e le proposte del progettista**
  - **Cliente e progettista fanno riferimento alle stesse entità astratte, chiamandole con gli stessi nomi**
- ✓ **Favorisce la comunicazione tra cliente e sviluppatore ed aumenta la possibilità che il programma soddisfi le reali esigenze del cliente**

Laboratorio di Algoritmi e Strutture Dati 2001/02
7

Il C++

- ✓ Introdotto da B. Stroustrup nel 1984 all' AT&T
- ✓ **Nel 1998-99 definito lo standard ANSI del linguaggio e della libreria**
- ✓ **Inizialmente era C + classi**
- ✓ **Supporta la programmazione basata sugli oggetti, la programmazione orientata agli oggetti e la programmazione generica**

Laboratorio di Algoritmi e Strutture Dati 2001/02
8

Perché il C++

- ✓ **Il C++ è un linguaggio ibrido**
  - **concilia l'efficienza di un linguaggio di basso livello (C) con le potenzialità di un linguaggio ad alto livello (SIMULA)**
- ✓ **Il C++ è quasi totalmente compatibile con il C**
  - **utilizzabile su tutte le piattaforme con efficienza paragonabile al C**
- ✓ **Il C++ è un linguaggio multi-stile**
  - **consente di utilizzare lo stile di programmazione più adatto senza costi aggiuntivi**

OGNUNO PAGA SOLO PER QUELLO CHE USA

Laboratorio di Algoritmi e Strutture Dati 2001/02
9

Linguaggio Multi-Stile

- ✓ **Un linguaggio di programmazione serve per**
  - **specificare le azioni della macchina**
  - **descrivere i concetti necessari a rappresentare le funzionalità del programma**
- ✓ **Il C++ consente di controllare la macchina a basso livello e supporta l'astrazione dei dati e la OOP**
- ✓ **Supportare uno stile di programmazione significa fornire strumenti che rendono quello stile semplice ed efficiente**
  - **operatori, controlli in compilazione ed esecuzione**

Laboratorio di Algoritmi e Strutture Dati 2001/02

10

Stili Supportati In C++

- ✓ Procedurale
- ✓ Modulare
- ✓ Programmazione basata sugli oggetti
- ✓ Programmazione orientata agli oggetti
- ✓ Programmazione generica

Il programma paga solo per le risorse che richiede

Laboratorio di Algoritmi e Strutture Dati 2001/02

11

Caso di Studio: il Vettore

- ✓ Definizione di una struttura dati Vettore
  - **Insieme di elementi, dello stesso tipo, identificabili tramite una posizione**
  - **Possibilità di specificare la dimensione del vettore ed il tipo degli elementi**

Operazioni del tipo Vettore

- ✓ Creare e inizializzare un vettore
- ✓ Lettura e scrittura di singoli elementi, individuati tramite la posizione nell'insieme
- ✓ Controllo sulla correttezza dell'indicizzazione
- ✓ Assegnazione di un vettore ad un altro vettore
- ✓ Confronto tra due vettori
- ✓ Ricerca di un elemento, del minimo e del massimo
- ✓ Ordinamento degli elementi del vettore

Programmazione Procedurale

1. **Definire le operazioni da eseguire**
2. **Individuare gli algoritmi migliori per eseguire tali operazioni**

Il C++ supporta la programmazione procedurale attraverso il meccanismo di chiamata a funzione

Laboratorio di Algoritmi e Strutture Dati 2001/02

14

Problemi della Programmazione Procedurale

- ✓ Non facilita il riutilizzo del codice
- ✓ La conoscenza dei dettagli delle specifiche è distribuita in tutto il codice

Laboratorio di Algoritmi e Strutture Dati 2001/02

15

Implementazione di Vettore in Programmazione Procedurale

- ✓ **Il Vettore può essere implementato tramite il tipo array**
  - **insieme di elementi dello stesso tipo memorizzati sequenzialmente in memoria ed indirizzati tramite un puntatore contenente l'indirizzo del primo elemento**

```
int a[10];
a[3] = 1;
int i = a[0];
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

16

Creazione e Inizializzazione

- ✓ **E' possibile definire e inizializzare solo vettori di dimensione costante**

```
int a[4] = {1, 2, 3, 4};
```

- ✓ **Vettori di dimensione variabile devono essere allocati dinamicamente**
  - **L'inizializzazione deve essere eseguita tramite ciclo for**

```
int size = 4;
int *a = new int[size];
for(int i = 0; i < size; i++)
    a[i] = i+1;
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

17

Accesso agli elementi

- ✓ Effettuato tramite l'operatore di indicizzazione []
  - **calcola l'indirizzo dell'elemento da accedere a partire dall'indirizzo del primo elemento del vettore**
  - **gli indici partono da 0**
  - **Il compilatore controlla solo se l'indice non è negativo**
- ✓ **Il compilatore non segnala come errato un indice oltre la dimensione dell'array**
  - **Il programma deve provvedere a controllare l'indice esplicitamente**

```
if( i < size)
    a[i] = 0;
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

18

Operazioni su Vettori

- ✓ **Non esistono operazioni su array**
  - **Ogni funzione su un vettore deve essere implementata da una funzione che accede gli elementi uno per volta**
  - **Si deve passare alla funzione sia il puntatore al primo elemento che la dimensione**

```
int min(int* vett, int size);
int ricerca(int* vett, int size, int x);
int ordina(int* vett, int size);
void assegna(int* vett1, int* vett2, int size);
int confronta(int* vett1, int* vett2, int size1, int size2);
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

19

Operazioni sui Vettori

- ✓ Si deve definire una funzione diversa per ogni tipo di vettore
- ✓ Non è possibile modificare il comportamento di una funzione

Programmazione Modulare

- ✓ **Un modulo è**
  - **Insieme di dati e di funzioni che operano su tali dati.**
  - **Accesso ai dati del modulo tramite un'interfaccia utente.**

Attenzione spostata dalle operazioni ai dati (principio del data hiding)

Programmazione Modulare

1. **Individuare i moduli necessari**
2. **Suddividere il programma in modo che i dati siano nascosti nei moduli.**

- ✓ **Il C la consente attraverso la compilazione di unità separate.**
- ✓ **Il C++ la supporta con le classi**

Laboratorio di Algoritmi e Strutture Dati 2001/02

22

Vantaggi e Svantaggi dei Moduli

- ✓ Vantaggi
  - Dettagli delle specifiche confinati nei moduli
  - Definizione di strutture dati opache (visibile solo l'interfaccia ma non l'implementazione)
- ✓ Svantaggi
  - Moduli non flessibili

Laboratorio di Algoritmi e Strutture Dati 2001/02

23

Modulo Vettore

- ✓ organizzato su due file
  - vettore.cc (oppure .C o .cpp)
  - vettore.h
- ✓ vettore.cc contiene le definizioni delle funzioni che implementano le operazioni sui vettori
- ✓ vettore.h contiene i prototipi delle funzioni e le definizioni delle variabili globali e delle costanti
- ✓ per usare una variabile di tipo vettore si include vettore.h nel proprio programma e si linka vettore.cc
- ✓ Per evitare conflitti di nome tutte le definizioni vengono inserite in uno spazio dei nomi (namespace) VETTORE

Laboratorio di Algoritmi e Strutture Dati 2001/02

24

Gestione delle Eccezioni

- ✓ Può accadere che un'eccezione provocata da un modulo venga individuata da un altro modulo
  - L'eccezione è un comportamento che si verifica in esecuzione
  - Il modulo che individua l'eccezione non è in grado di gestirla
- ✓ In C++ esiste un metodo standard per gestire le eccezioni
  - Il modulo che individua l'eccezione lancia un messaggio
  - Questo messaggio viene raccolto dalla funzione addetta che provvede a risolvere l'eccezione
  - Se il messaggio non viene raccolto il programma termina

Laboratorio di Algoritmi e Strutture Dati 2001/02

25

Moduli ed Astrazione dei Dati

- ✓ La programmazione modulare consente di definire tipi di dati astratti …
  - Un modulo di gestione *X* può controllare tutte la variabili di tipo *X* del programma
  - L'utilizzo delle funzioni dell'interfaccia del modulo di gestione è indipendente dalla loro implementazione
- ✓ … ma non supporta l'astrazione dei dati
  - Il tipo di dato creato dal modulo di gestione è sostanzialmente diverso dai tipi di dati predefiniti

Laboratorio di Algoritmi e Strutture Dati 2001/02

26

Programmazione Basata sugli Oggetti

- ✓ Un linguaggio che supporta la programmazione basata sugli oggetti deve consentire di definire nuovi tipi di dato che si comportino come quelli predefiniti
- ✓ la programmazione basata sugli oggetti consente di definire dei tipi di dato che siano
  - chiusi (implementazione non visibile)
  - aperti (facilmente modificabili)

Laboratorio di Algoritmi e Strutture Dati 2001/02

27

La Programmazione Basata sugli Oggetti

1. **Individuare i tipi di dato da utilizzare**
2. **Fornire l'interfaccia di ciascun tipo di dato**

- ✓ **Il C++ supporta la programmazione basata sugli oggetti con le classi**

Laboratorio di Algoritmi e Strutture Dati 2001/02

28

Classe Vettore

```
class intVettore {
public:
   intVettore(int size);                          // inizializzazione
   intVettore(const intVettore&);
   ~intVettore();                                 // distruzione
    int& operator[](int);                         // indicizzazione
   bool operator==(const intVettore&) const;      // uguaglianza
   bool operator!=(const intVettore&) const;      // diseguaglianza
   IntVettore& operator=(const intVettore&);      // assegna
   int size() const;                              // dimensione
   int min() const;                               // minimo
   int max() const;                               // massimo
   int ricerca(int x) const;                      // ricerca
   void ordina();                                 // ordinamento
private:
   int dimensione, *a;
};
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

29

Incapsulamento dei Dati

- ✓ La parte pubblica della classe costituisce l'interfaccia pubblica
  - L'interfaccia contiene i metodi di intVettore che possono essere invocati dagli utenti della classe
  - Ogni funzione che utilizza oggetti intVettore deve conoscere soltanto la sua interfaccia pubblica
- ✓ La parte privata contiene l'implementazione della classe
  - L'implementazione è indipendente dalla definizione dell'interfaccia e può essere cambiata senza effetti sulle funzioni che utilizzano la classe

Laboratorio di Algoritmi e Strutture Dati 2001/02

30

Creazione e Inizializzazione

- ✓ E' possibile definire variabili di tipo intVettore di qualunque dimensione

```
int dim = 10;
intVettore vett(dim);
```

- ✓ Una funzione può prendere come argomento e restituire come risultato variabili di tipo intVettore

```
intVettore f(intVettore);
```

- ✓ E' possibile inizializzare una variabile intVettore copiandola da un'altra variabile dello stesso tipo

```
intVettore vett2 = vett;
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

31

Accesso agli elementi

- ✓ L'accesso agli elementi del vettore è effettuato tramite l'operatore []
- ✓ L'implementazione della funzione operator[] può essere realizzata in modo che
  - gli indici partono da 1 (o da qualunque altro valore)
  - È possibile impedire accessi a elementi non esistenti

```
vett[5] = 10;
vett[15] = 0;    // scrittura non eseguita
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

32

Operazioni sui Vettori

- ✓ E' possibile invocare una delle funzioni della classe utilizzando l'operatore di selezione
  - Lo stesso operatore utilizzato per selezionare i campi di uno struct del C

```
int x = vett.min();
```

- ✓ L'utilizzo di queste funzioni è indipendente dalla loro implementazione
  - Se qualcuno modifica l'implementazione della funzione ordina tutti i programmi che utilizzavano tale funzione devono solo essere ricompilati con la nuova libreria

Laboratorio di Algoritmi e Strutture Dati 2001/02

33

Problemi della classe intVettore

- ✓ La classe intVettore implementa solo vettori di interi
  - Per avere vettori di char si deve definire una nuova classe
  - La nuova classe è identica alla precedente ma contiene char
- ✓ Non è possibile creare nuovi classi che modifichino alcuni comportamenti della classe
  - Per definire la classe intVettoreSorted si deve partire da zero

Laboratorio di Algoritmi e Strutture Dati 2001/02

34

Programmazione Orientata agli Oggetti

- ✓ Ereditarietà
  - definisce un tipo di dati come specializzazione di un altro tipo esistente
- ✓ Legame dinamico (dynamic binding)
  - accede ad una intera gerarchia di classi attraverso una interfaccia comune e lascia al compilatore il compito di individuare a quale delle diverse implementazioni si sta facendo riferimento

Laboratorio di Algoritmi e Strutture Dati 2001/02

35

Programmazione Orientata agli Oggetti

1. **Individuare i tipi di dati da utilizzare**
2. **Fornire l'interfaccia di ciascun tipo di dati**
3. **Individuare gli elementi comuni tra i vari tipi di dati**

- ✓ **I linguaggi Object Oriented supportano la programmazione per differenze ed il polimorfismo**
- ✓ **Il C++ supporta queste tecniche con la derivazione di classi e le funzioni virtuali**

Laboratorio di Algoritmi e Strutture Dati 2001/02

36

Programmazione per Differenze

- ✓ Per ogni nuovo tipo di dati
  - cerca se esistono in libreria tipi simili
  - individua tutti gli elementi che possono essere ereditati
  - implementa soltanto le funzioni totalmente nuove

Laboratorio di Algoritmi e Strutture Dati 2001/02

37

Definizione di intVettoreSorted

- ✓ Si definisce la nuova classe intVettoreSorted come derivata da intVettore
  - Tutte le funzioni definite per intVettore vengono estese a intVettoreSorted
  - Si modifica l'operatore [] per far in modo da mantenere l'ordine dopo ogni modifica

```
#include "intVettore.h"
class intVettoreSorted : public intVettore;
int& intVettoreSorted::operator[](int i) { ... }
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

38

Polimorfismo

- ✓ Il tipo di dati costruito attraverso l'eredità può avere stesse funzionalità del tipo base ma con implementazioni differenti
- ✓ oggetti del tipo base e di quello ereditato possono essere trattati alla stessa maniera
  - il compilatore capisce quale implementazione utilizzare

```
void riempi(intVettore* v) {
   int x;
   for( int i = 0; i < v->dimensione; i++ ) {
         cin >> x;
         v[i] = x;  }
}
intVettoreSorted *v; riempi(v);
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

39

Uso Polimorfico delle Classi intVettore e intVettoreSorted

Definiamo la classe intVettore in modo che

- ✓ Ad una funzione che ha come argomento un intVettore può essere passato sia un oggetto intVettore che intVettoreSorted
- ✓ In esecuzione il programma capisce di che tipo è l'argomento che gli è stato passato
- ✓ Se l'argomento è di tipo intVettoreSorted gli accessi vengono eseguiti con l'operatore [] modificato

Laboratorio di Algoritmi e Strutture Dati 2001/02

40

Programmazione Generica

1. **Decidere quali algoritmi utilizzare**
2. **Parametrizzarli in modo che possano operare su un'ampia gamma di tipi di dato e di strutture dati**

Il C++ implementa la programmazione generica con i template

Laboratorio di Algoritmi e Strutture Dati 2001/02

41

Template di Classe Vettore

- ✓ il template è un modello di classe
  - descrive la classe senza specificare che elementi contiene
- ✓ il compilatore può ricavare dal modello la definizione della classe Vettore e di tutte le sue funzioni, relative ad ogni tipo di dati contenuto
  - ogni istanza viene creata solo quando richiesto dal programma

```
template <class T> class Vettore { ... };
Vettore<int> vi;
Vettore<double> vd;
Vettore<string> vs;
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

42

Template Classe Vettore

```
template <class T> class Vettore {
public:
   Vettore(int size);                                  // inizializzazione
   Vettore(const Vettore<T>&);
   virtual ~Vettore();                                 // distruzione
   virtual T& operator[](int);                         // indicizzazione
   bool operator==(const Vettore<T>&) const;           // uguaglianza
   bool operator!=(const intVettore<T>&) const;        // diseguaglianza
   Vettore<T>& operator=(const Vettore<T>&);           // assegna
   int size() const;                                   // dimensione
   virtual T min() const;                              // minimo
   virtual T max() const;                              // massimo
   virtual T ricerca(int x) const;                     // ricerca
   void ordina();                                      // ordinamento
private:
   int dimensione;  T *a;
};
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

43

Librerie standard del C++

Le librerie standard del C++ offrono vasto supporto alla programmazione generica (STL)

- ✓Classi contenitore
  - classi contenenti collezioni di elementi dello stesso tipo
- ✓Iteratori
  - Costrutti che permettono di accedere agli elementi di un contenitore indipendentemente da come è implementato
- ✓Algoritmi generici
  - Algoritmi parametrizzati per poter lavorare su ogni tipo di contenitore

Laboratorio di Algoritmi e Strutture Dati 2001/02

44

La classe Vettore della libreria standard

La libreria standard contiene un template vector

- ✓ Consente di modificare dinamicamente la dimensione del vettore
- ✓ Contiene solo gli operatori strettamente necessari ad accedere agli elementi
  - begin(), end(), [], ++
  - Tutte le altre operazioni sono implementate da algoritmi generici che operano su contenitori

```
vector<int> A;
sort(A.begin(), A.end());
```

Laboratorio di Algoritmi e Strutture Dati 2001/02

45