Programmazione 2
e Laboratorio di Programmazione

Corso di Laurea in

Informatica

Università degli Studi di Napoli "Parthenope"

Anno Accademico 2023-2024

Prof. Luigi Catuogno

1

Informazioni sul corso

Docente	Luigi Catuogno **luigi.catuogno@uniparthenope.it**
Orario	Lun: 9:00-11:00 Mer: 11:00-13:00
Sede	Centro Direzionale Napoli **Aula Magna**
Ricevimento	Mer: 14:00-16:00 (previo appuntamento) Ufficio docente oppure Team: **cxxa3bo**

2

Libri di testo

Introduzione al linguaggio – costrutti e tecniche di base

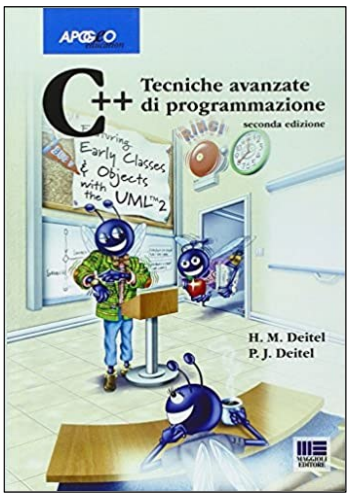
[FdP] H. M. Deitel, P. J. Deitel
C++ Fondamenti di programmazione
II ed. (2014) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8571-9

3

Libri di testo

Tecniche avanzate e strutture dati elementari

[TAP] H. M. Deitel, P. J. Deitel
C++ Tecniche avanzate di programmazione
II ed. (2011) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8572-6



4

Risorse on-line

Team del corso

Programmazione 2 AA 2023-24 - Prof. Catuogno
Comunicazioni, incontri e avvisi per il corso
Codice: **ftomzjx**

Piattaforma e-learning

Programmazione II e Laboratorio di Programmazione II - A.A. 2023-24
Materiale didattico, manualistica, esercitazioni.
URL: https://elearning.uniparthenope.it/course/view.php?id=2386

5

Dal C al C++

6

Le **`class`** in C++

7

Esercizio: *Simulazione di un C/C bancario #3*

Modifichiamo la classe MiniCCB per aggiungere le seguenti funzionalità:

Un metodo che cambi il PIN

```
bool cambiaPIN(string oldPIN, string newPIN)
```

Il PIN deve essere di almeno 5 caratteri

8

Esercizio: *Simulazione di un C/C bancario #3*

Modifichiamo la classe MiniCCB per aggiungere le seguenti funzionalità:

Un codice PUK che possa essere utilizzato per sbloccare un C/C dopo tre errori di PIN consecutivi (e cambia il PIN).

```
bool sblocca(string mioPUK, string newPIN)
```

Il PUK deve essere di almeno 10 caratteri

10 errori di PUK consecutivi bloccano il C/C

9

Esercizio: *sugli angoli e i gradi…*

Scrivere una classe **`tellAngle()`** con la seguente interfaccia:

```
tellAngle(int gradi, int primi, int secondi)
```

Il costruttore che imposta i tre attributi indicati. Se i parametri passati risultassero superiori risp. a 360, 60 e 60, imposta gli stessi valori *modulo* la rispettiva soglia.

```
void show()
```

visualizza i tre valori nel formato **`ggg:pp:ss`**

19

Esercizio: *sugli angoli e i gradi…*

```
int getGr();
int getPr();
int getSc();
```

restituisce il valore dell'attributo corrispondente

```
void setGr(int g);
void setPr(int p);
void setSc(int s);
```

imposta il valore dell'attributo corrispondente (modulo risp. 360, 60 e 60)

20

Funzioni (e classi) **`friend`**

Regole di visibilità dei membri di una classe:

public: l'accesso ai membri (sia attributi, sia funzioni) public è consentito a qualunque funzione

private: l'accesso ai membri private è consentito:

⇨ alle funzioni membro della stessa classe

⇨ alle funzioni e ai metodi di altre classi dichiarate **`friend`**

23

Funzioni (e classi) **friend**

```
class classWithFriends
{
      friend int friendFunction(classWithFriends, int);
      …
private:
      int privateNum;
public:
…
};
```

I prototipi delle funzioni *friend* vanno dichiarati nella classe, preceduti dalla parola chiave **friend**

24

Funzioni (e classi) **friend**

```
int friendFunction(classWithFriends c, int i)
{
      …
      return c.privateNum=i;
};
```

La funzione è definita *al di fuori* della classe e non è un suo metodo (anche se il suo prototipo compare all'interno della sua definizione).

La funzione **friendFunction** accede ai membri privati della classe **classWithFriends**

25

Esercizio: *contatori*

```
class counter0 {
private:
        int cnt;
public:
        counter0() { cnt=0; }
        int getCounter() { return cnt; }
        int increase() { return ++cnt; }
}
```

Scrivere la funzione **counterAling()** che prenda due oggetti della classe **counter0**, imposti gli attributi **cnt** di entrambe al valore minore tra i due e restituisca quello maggiore. Modificare la definizione della classe di conseguenza.

26

Esercizio: *contatori*

```
10  class counter0 {
11          friend int counterAlign(counter0&, counter0&);
12  private:
13          int cnt;
14  public:
16          counter0() { cnt=0; }
16          int getCounter() { return cnt; }
17          int increase() { return ++cnt; }
18  }
```

Si modifichi la classe introducendo la dichiarazione della funzione *friend* **counterAlign()**

27

Esercizio: *contatori*

```
20  int counterAlign(counter0 &a, counter0 &b)
21  {
22          int max;
23          if (a.cnt<=b.cnt){
24                  max=b.cnt;
25                  b.cnt=a.cnt;
26                  }
27          else {
28                  max=a.cnt;
29                  a.cnt=b.cnt;
30          }
31          return max;
32  }
 …  …
```

Il codice della funzione accede ai membri della classe, inclusi quelli privati

28

Esercizio: *Simulazione di un C/C bancario #4*

Si implementi una funzione esterna alla classe MiniCCB che permetta di effettuare un bonifico da un conto all'altro:

```
bool bonifico( MiniCCB &mioCC,
               string mioPIN,
               MiniCCB &suoCC, double importo )
```

La funzione prende il riferimento al conto che eroga il bonifico e il suo PIN, il riferimento al conto ricevente e l'importo della transazione

Chi ordina un bonifico conosce il PIN del suo conto, ma non quello del ricevente…

29

Le **class** in C++

La definizioni dei metodi può essere effettuata:

All'interno della dichiarazione della classe stessa

All'esterno. In questo caso, la definizione della classe conterrà solo i prototipi.

33

Le **class** in C++

```
class Punto {
private:
       double x;
       double y;
public:
       Punto() {x=0; y=0}
       double getX() {return x;}
       double getY() {return y;}
       void setX(double newx)
              {x=newx;}
       void setY(double newy)
              {y=newy;}
};
```

```
class Punto {
private:
       double x;
       double y;
public:
       Punto() {x=0; y=0}
       double getX();
       double getY();
       void setX(double);
       void setY(double);
};
…
void Punto::setX(double newx){
       x=newx;
}
…
```

34

Le **class** in C++

può essere preferibile tenere separata la definizione dell'interfaccia di una classe dalla sua implementazione

```
class Punto {
private:
        double x;
        double y;
public:
        Punto() {x=0; y=0}
        double getX() {return x;}
        double getY() {return y;}
        void setX(double newx)
                {x=newx;}
        void setY(double newy)
                {y=newy;}
};
```

Qui il metodo è implementato nella definizione della classe (metodo *inline*)

```
class Punto {
private:
        double x;
        double y;
public:
        Punto() {x=0; y=0}
        double getX();
        double getY();
        void setX(double);
        void setY(double);
};
…
void Punto::setX(double newx){
        x=newx;
}
…
```

Qui, nella definizione della classe appare soltanto il *prototipo* del metodo…

Il metodo è quindi definito successivamente (anche in un altro file). Naturalmente occorre indicare esplicitamente la classe a cui si riferisce mediante l'operatore di *risoluzione dello scope* ::

35

Le **class** in C++

Perché tenere separata la definizione dell'interfaccia di una classe dalla sua implementazione?

Perché questi due aspetti possono scaturire da cicli di sviluppo differenti.

- La definizione della classe è spesso contenuta in un file header che è utilizzato nella sua forma *sorgente*. Eventuali modifiche alla classe impattano sulla sua *riusabilità.*
- Il codice dei metodi è maggiormente utilizzato quanto è già compilato e incluso nelle librerie. Modifiche dell'implementazione, che salvaguardino il prototipo e le funzionalità, sono trasparenti alle app.

36

Esercizio: *tic-tac-toe #1*

Scrivere una classe **`tttPlayGround`** che abbia, tra gli altri, i seguenti attributi privati:

```
int playground[3][3];
int prossimoG;
```

37

Esercizio: *tic-tac-toe #1*

La classe **`tttPlayGround`** presenta la seguente interfaccia:

```
tttPlayGround()
```

il costruttore di default. Azzera il contenuto di **`playground`** e imposta **`prossimoG=1`**;

```
void show()
```

visualizza il contenuto di **`playground`** in modo che, le celle contenenti 0 appaiano vuote, quelle a 1 riportino il carattere **`O`** e quelle con 2 il carattere **`X`**

38

Esercizio: *tic-tac-toe #1*

`char prossimo()`

restituisce il il carattere **`O`** se **`prossimoG`** vale 1, oppure **`X`** se **`prossimoG`** vale 2;

`bool muovi(int riga, int col)`

assegna il valore corrente di **`prossimoG`** alla cella indicata da **`riga`** e **`col`** . Quindi aggiorna il valore di **`prossimoG`** per dare il turno all'altro giocatore (se è 1 passa a 2, se è 2 passa a 1). restituisce false se non è possibile fare la mossa (true se è ok).

`void reset()`

39

Esercizio: *tic-tac-toe #1*

```
10  class tttPlayGround {
11  private:
12          int playground[3][3];
13          int prossimoG;
14  public:
15          tttPlayGround()
16          {
17                  for(i=0;i<3;i++)
18                          for(j=0;j<3;j++)
19                                  playground[i][j]=0;
20                  prossimoG=1;
21          }
22          void show();
23          char prossimo();
24          bool muovi(int,int);
25          void reset();
26  };
```

40