Programmazione 2 e Lab. di programmazione 2

Corso di Laurea in Informatica - Anno Accademico 2022-23

Docenti
Prof. Angelo Ciaramella
[angelo.ciaramella@uniparthenope.it]
Prof. Luigi Catuogno
[luigi.catuogno@uniparthenope.it]
Tutor
Dott. Antonio Vanzanella
[antonio.vanzanella@studenti.uniparthenope.it]

1

Descrizione del Corso

Libro di testo

[FdP]

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

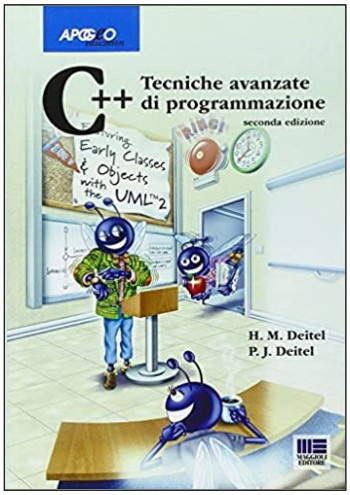
2

Descrizione del Corso

Libro di testo H. M. Deitel, P. J. Deitel

[TAP] C++ Tecniche avanzate di programmazione

II ed. (2011) Maggioli Editore (Apogeo Education)
ISBN: 978-88-387-8572-6



3

Orari e modalità di ricevimento studenti

Docenti:

Prof. Angelo Ciaramella	Martedì dalle 14:00 alle 16:00 - telematico (codice Teams r3p3w0z)
Prof. Luigi Catuogno	Giovedì dalle 11:00 alle 13:00 - telematico (codice Teams)

Tutor:

Dott. Antonio Vanzanella	Martedì dalle 11:00 alle 13:00 - telematico (codice Teams 92dbag0)

5

Il Linguaggio C++

(per programmatori C)

Parte prima

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

```
 4  class MiniCCB {
 5  private:
 6          double dare;
 7          double avere;
 8          int tentativiPIN;
 9          int tentativiPUK;
10          bool blocco;
11          string PIN;
12          string PUK;
…   …
```

10

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

```
13          bool CheckPIN(string p)
14          {
…                   …
26          }
27
28          bool CheckPUK(string p)
29          {
30                  if (tentativiPUK>10)
31                          return false;
32                  if(p==PUK){
33                          tentativiPUK=0;
34                          return true;
35                  }
36                  tentativiPUK++;
37                  return false;
38          }
```

11

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

```
39  public:
40          MiniCCB() {
41                  dare=avere=0;
42                  tentativiPIN=tentativiPUK=0;
43                  PIN="11111";
44                  PUK="1111111111";
44                  blocco=false;
46          };
47
48          MiniCCB(string p, string q) {
49                  dare=avere=0;
50                  tentativiPIN=tentativiPUK=0;
52                  PIN=p;
52                  PUK=q;
53                  blocco=false;
54          };
```

12

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

```
55          bool Saldo(double &importo, string p)
56          {
…                   …
62          };
63          bool Deposito(double importo, string p)
64          {
…                   …
70          };
71          bool Prelievo(double importo, string p)
72          {
…                   …
78          };
79          bool Bloccato()
80          {
81                  return blocco;
82          }
```

13

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

```
83          bool CambiaPIN(string oldp, string newp)
84          {
85                  if(blocco || !CheckPIN(oldp))
86                          return false;
87                  if(newp.size()<5)
88                          return false;
89                  PIN=newp;
90                  return true;
91          }
92
```

14

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

```
93          bool Sblocca(string pk, string newp)
94          {
95                  if(!CheckPUK(pk))
96                          return false;
97                  if(!blocco||newp.size()<5)
98                          return false;
99                  PIN=newp;
100                 tentativiPIN=0;
101                 blocco=false;
102                 return true;
103         }
104 };
```

15

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

```
105  int main()
106  {
107          double saldo=0, importo=0;
108          bool ancora=true;
109          int scelta=0;
110          MiniCCB Conto("12345","abcdefghil");
111          string mioPIN,nuovoPIN,mioPUK;
 …           …
```

16

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

```
164          case 4:
165                  do {
166                          cout << "Inserisci il nuovo PIN (min. 5 caratteri): ";
167                          cin >> nuovoPIN;
168                  } while (nuovoPIN.size()<5);
169                  cout << "Inserisci il PIN: ";
170                  cin >> mioPIN;
171                  if(!Conto.CambiaPIN(mioPIN,nuovoPIN)){
172                          cout<<"L'operazione non è andata a buon fine!" << endl;
173                          cout<<"Verificare che il PIN sia corretto o …"<<endl;
174                  }
175                  break;
```

17

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

```
176         case 5:
177                 do {
178                         cout << "Inserisci il nuovo PIN (min. 5 caratteri): ";
179                         cin >> nuovoPIN;
180                 } while (nuovoPIN.size()<5);
181                 cout << "Inserisci il PUK: ";
182                 cin >> mioPUK;
183                 if(!Conto.Sblocca(mioPUK,nuovoPIN)){
184                         cout<< "L'operazione non è andata a buon fine!" << endl;
185                         cout << "Verificare che il PUK sia … la banca"<<endl;
186                 }
187                 break;
188         } // end switch
  …      …
```

18

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

Esercizio: *sugli angoli e i gradi…*

```
10  class tellAngle {
11  private:
12          int gradi;
13          int primi;
14          int secondi;
15  public:
16          tellAngle(int g, int p, int s)
17          {
18                  gradi=g%360;
19                  primi=p%60;
20                  secondi=s%60;
21          }
 …  …
```

21

Esercizio: *sugli angoli e i gradi…*

```
22          void show()
23          {
24                  cout << setw(3)<<setfill('0')<<right<<gradi<<":";
25                  cout << setw(2)<<setfill('0')<<right<<primi<<":";
26                  cout << setw(2)<<setfill('0')<<right<<secondi<<endl;
27          }
28          int getGr() { return gradi; }
29          void setGr(int g) { gradi=g%360; }
30          // etc. …
31  };      // fine def. classe
32  int main()
33  {
34          tellAngle x(12,6,35);
35          x.show();
36  }
```

22

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

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

```
10  class MiniCCB {
11          friend bool bonifico(MiniCCB &, string, MiniCCB &, double);
12  private:
13          double dare;
14          double avere;
16          int tentativiPIN;
16          int tentativiPUK;
17          bool blocco;
18          string PIN;
19          string PUK;
20          …
```

bonifico è una funzione *friend* della classe **MiniCCB**.

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

```
118 }; // end class MiniCCB
119
120 bool bonifico(MiniCCB &mioCC, string mioPIN, MiniCCB &suoCC, double importo)
121 {
122         if(!mioCC.Prelievo(importo,mioPIN))      {
123                 cout << "L'operazione non è andata       fine!" << endl;
124                 cout << "Verificare che il PIN sia c
125                 return false;
126         }
127         suoCC.avere+=importo;
128         return true;
129 }
130
131 int main()
132 {
…
136         MiniCCB Conto("12345","abcdefghil"), AltroConto("54321","ciao!ciao!");
```

Per l'addebbito non è necessario implementare nulla di nuovo, basta utilizzare il metodo **Prelievo**

Effettua l'accredito incrementando direttamente l'attributo **importo** (che è privato) di **suoCC**

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

```
211         case 6:
212                 cout << "Inserisci l'importo del bonifico: ";
213                 cin >> importo;
214                 cout << "Inserisci il PIN: ";
215                 cin >> mioPIN;
216                 if(!bonifico(Conto,mioPIN,AltroConto,importo)) {
217                         cout<< "L'operazione non è andata a buon fine!" << endl;
218                         cout << "Verificare che il PIN sia corretto …"<<endl;
219                 }
220                 else
221                  {
222                         AltroConto.Saldo(importo,"54321");
223                         cout << "saldo altro conto: "<< importo << endl;
224                 }
225                 break;
  …
…
```

32

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;
}
…
```


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 ::*

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