

## Descrizione del Corso

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



1

## 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



2

# 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)

4

# Il Linguaggio C++

*(per programmatori C)*

Parte prima

5

# Tipi di dati aggregati

6

## Tipi di dati aggregati

- Definizione di nuovi tipi di dato «combinando» i tipi già esistenti
- Il C++ fornisce tre metodi di aggregazione:
  - **struct:**
  - **union:**
  - **class:**
    - Evoluzione della struct finalizzata alla *programmazione orientata agli oggetti (OOP)*
    - Definizione di operatori sui nuovi tipi (con accesso «esclusivo» ai campi)
    - Ridefinizione degli operatori nativi ( `'++'`, `'--'`, `'%'` ) sui record

7

## Le **class** in C++

8

## Le **class** in C++

La definizione di una classe in C++ include:

Definizione delle sue componenti (membri):

**Attributi:** variabili di vari tipi/classi da essa aggregati

**Funzioni:** funzioni *specializzate* nella manipolazione ed elaborazione degli attributi della classe stessa, spesso in maniera *esclusiva*

9

## Le **class** in C++

La definizione di una classe in C++ include:

Definizione delle sue componenti (membri)

*Regole di visibilità* dei suoi membri:

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

**private:** l'accesso ai membri `private` è consentito soltanto alla funzione membro dello stesso *oggetto* (con alcune eccezioni)

10

## Classi e oggetti.

In prima approssimazione:

Con la keyword `class`, si *descrive* una nuova struttura dati, elencando i dati che essa aggrega e definendo (tramite le funzioni membro) le operazioni elementari che si effettuano con essa. La classe descrive un nuovo tipo di dato.

Quando si dichiara/alloca una «variabile» di questo nuovo tipo, questa prende il nome di *oggetto*: un'area di memoria strutturata secondo la descrizione della classe

11

## Esempio: *ancora i punti sul piano*

Si *definisce* la classe **Punto** fornendone l'elenco dei *membri*

```
class Punto {
public:
    double x;
    double y;

    Punto() { x=y=0; };

    Punto(double c1,double c2) {
        x=c1;
        y=c2;
    };
};
```

12

## Esempio: *ancora i punti sul piano*

Tutti i membri dichiarati successivamente alla keyword public sono visibili anche dall'esterno della classe

Si *definisce* la classe **Punto** fornendone l'elenco dei *membri*

```
class Punto {
public:
    double x;
    double y;

    Punto() { x=y=0; };

    Punto(double c1,double c2) {
        x=c1;
        y=c2;
    };
};
```

La classe ha due attributi, costituiti da due variabili di tipo double

La classe ha due funzioni membro

13

## Esempio: *ancora i punti sul piano*

Si definisce la classe **Punto** fornendone l'elenco dei *membri*

```
class Punto {
public:
    double x;
    double y;

    Punto() { x=y=0; };

    Punto(double c1,double c2) {
        x=c1;
        y=c2;
    };
};
```

Le funzioni membro che hanno lo stesso nome della classe sono i suoi *costruttori*. Si tratta delle funzioni invocate ogni volta che viene creato un nuovo *oggetto*.

14

## Esempio: *ancora i punti sul piano*

```
24 double distanza (Punto p1, Punto p2)
25 {
26     return sqrt(pow((p2.x-p1.x),2)+pow((p2.y-p1.y),2));
27 }
28
29 int main()
30 {
31     Punto p1(3,4), *p2;
32     p2= new Punto();
33     cout << "Distanza= " << distanza(p1,*p2) << endl;
34
35 }
```

15

## Esempio: *ancora i punti sul piano*

```

24 double distanza (Punto p1, Punto p2)
25 {
26     return sqrt(pow((p2.x-p1.x),2)+pow((p2.y-p1.y),2));
27 }
28
29 int main()
30 {
31     Punto p1(3,4), *p2;
32     p2= new Punto();
33     cout << "Distanza= " << distanza(p1,*p2) << endl;
34
35 }

```

La funzione **distanza** accede ai membri degli oggetti di classe **Punto**, poiché questi sono **public**

16

## Esempio: *ancora i punti sul piano*

```

24 double distanza (Punto p1, Punto p2)
25 {
26     return sqrt(pow((p2.x-p1.x),2)+pow((p2.y-p1.y),2));
27 }
28
29 int main()
30 {
31     Punto p1(3.0,4.5), *p2, p3;
32     p2 = new Punto();
33     cout << "Distanza= " << distanza(p1,*p2);
34
35 }

```

La dichiarazione di oggetti di classe **Punto**, fatta indicando i due parametri **double**, invoca il costruttore **Punto(double, double)**

L'operatore **new** invoca esplicitamente il costruttore indicato (in questo caso **Punto()**)

La dichiarazione di puntatori non richiede la chiamata ai costruttori. La dichiarazione di una variabile senza alcun parametro, causa la chiamata al *costruttore standard* (in questo caso **Punto()**)

17



## Esempio: *Simulazione di un C/C bancario #1*

```

4  class MiniCCB {
5  public:
6      double Dare;
7      double Avere;
8
9      MiniCCB() { Dare=Avere=0; };
10     double Saldo() {
11         return Avere-Dare;
12     };
13     void Deposito(double importo) {
14         Avere+=importo;
15     };
16     void Prelievo(double importo) {
17         Dare+=importo;
18     };
19 };

```

18

## Esempio: *Simulazione di un C/C bancario #1*

```

20 int main()
21 {
22     bool ancora=true;
23     int scelta=0;
24     MiniCCB Conto;
25     double saldo=0, importo=0;
26
27     cout << "Simulazione di un C/C Bancario v.1" << endl << endl;
28     cout << "MENU"<<endl<<endl;
29     do {
30         cout << "MENU"<<endl<<endl;
31         cout << "1: Deposito" << endl;
32         cout << "2: Prelievo" << endl;
33         cout << "3: Saldo  " << endl;
34         cout << "0: Fine  " << endl<<endl;
35         cout << "Inserisci la scelta:";
36         cin >> scelta;

```

19

## Esempio: *Simulazione di un C/C bancario #1*

```

37         switch(scelta){
38             case 0:
39                 ancora=false;
40                 break;
41             case 1:
42                 cout << "Importo da versare: ";
43                 cin >> importo;
44                 Conto.Deposito(importo);
45                 break;
46             case 2:
47                 cout << "Importo da prelevare: ";
48                 cin >> importo;
49                 Conto.Prelievo(importo);
50                 break;

```

20

## Esempio: *Simulazione di un C/C bancario #1*

```

51
52
53             case 3:
54                 cout << "Saldo del conto: ";
55                 cout << Conto.Saldo() << endl;
56                 break;
57             } // end switch()
58         } while(ancora);

```

21

## La classe **string** del C++

Una stringa è una sequenza di caratteri di lunghezza variabile

Il linguaggio C implementa le stringhe utilizzando gli array di caratteri (**char**[]) con l'accorgimento di porre un carattere '\0' alla fine della sequenza.

La libreria standard del C fornisce anche una serie di funzioni per effettuare le operazioni elementari tra stringhe: concatenazione, copia, estrazione di sottostringhe etc..

Il C++ eredita *in toto* queste funzionalità ma...

22

## La classe **string** del C++

La libreria standard del C++ introduce un modo più moderno (e *object-oriented*) per usare le stringhe: la classe **string**

Per creare/manipolare oggetti della classe **string** è necessario scrivere:

```
#include<string>
using std::string
```

Poiché la classe è definita nel namespace **std**

23

## La classe **string** del C++

Principali operazioni con le stringhe:

### Dichiarazione e inizializzazione

```
string s1;           // cost. default: s1 è vuota
string s2=s1;       // s2 è una copia di s1;
string s3= "ciao";  // s3 copia di una costante
string s4(10, 'x'); // s4 è "xxxxxxxxxxx"
string s5(s1);      // s5 è una copia di s1;
string s6("ciao");  // Inizializzazione diretta
```

24

## La classe **string** del C++

Principali operazioni con le stringhe:

### Dichiarazione e inizializzazione

```
char s1[]= "ciao";  // vecchie e ...
string s2=s1;       // nuove stringhe;
```

25

## La classe **string** del C++

Principali operazioni con le stringhe:

**Input/Output di stringhe:**

```
#include<iostream>
#include<string>
using namespace std;
...
string s("ciao"), t;
cout << "stringa s: " << s <<endl;
```

26

## La classe **string** del C++

Principali operazioni con le stringhe:

**Input/Output di stringhe:**

```
...
string t;
cin >> t;
cout << "stringa t: " << t <<endl;
```

**cin** legge caratteri fino a quando non incontra uno spazio (*whitespace*). Pertanto, se l'utente inserisce «*uno due*», in **t** ci va solo «*uno*»

27

## Esempio: *Leggere più stringhe...*

```
int main()
{
    string word;
    while (cin >> word)
        cout << "Stringa: \"" << word << "\"" << endl;

    return 0;
}
```

Il `while` controlla la validità dell'input dallo stream `cin`. In caso di errore (o di end-of-file), lo stream restituisce **false** e il ciclo termina.

28

## Esempio: *Leggere righe intere (spazi inclusi)*

```
int main()
{
    string line;
    while (getline(cin, line))
        cout << "Line: \"" << line << "\"" << endl;

    return 0;
}
```

`getline()` riempie la stringa `line` con tutti i caratteri immessi fino al `'\n'` (*newline*). In caso di errore (o di end-of-file), la funzione restituisce **false** e il ciclo termina.

29

## La classe **string** del C++

Operatori di confronto tra stringhe:

```
string s1, s2, s3, s4, s5, s6;

s1==s2;    // true se s1 e s2 sono uguali
s2!=s3;    // true se s1 e s2 sono diverse
s3<=s4;    // true se s3 "precede s4 nell'ordine
            // lessicografico" (oppure se sono uguali).
```

Gli altri operatori relazionali <, > e >= sono utilizzati come di consueto e si riferiscono all'ordinamento lessicografico

30

## Esempio: *concatenazione di stringhe*

Impiego degli operatori + e +=

```
int main()
{
    string s1 = "Hello, ", s2("World!"), s3;
    string s4[3]={"Goodbye","Blue","Sky"}, s5;
    s3=s1 + s2;
    s5=s4[0];
    for (int i=1;i<3;i++)
        s5 += " " + s4[i];
    cout << "s3=" << s3 << endl << " s5= " << s5 << endl;
}
```

Ad ogni iterazione, a s5 si somma una nuova stringa estratta dall'array di stringhe s4;

```
s3=Hello, World! s5=Goodbye Blue Sky
```

31

## La classe **string** del C++

Principali operazioni con le stringhe:

```
bool t;
string s;
...
If(s.empty())
    cout << "s è vuota!" << endl;
...
```

Il metodo **empty()** restituisce **true** se la «sua» stringa è vuota.

32

## La classe **string** del C++

Principali operazioni con le stringhe:

```
int lens;
string s("Marea");
...
cout << "s è lunga: " << s.size() << "caratteri." << endl;
...
```

Il metodo **size()** restituisce la lunghezza (il numero di caratteri che contiene) della «sua» stringa. Il metodo **length()** è equivalente.

33



## La classe **string** del C++

Scorrere i caratteri contenuti nelle stringhe:

```
string s("Marea");
...
for(int i=0;i<s.size();i++)
    cout << "s[" << i << "]= " << s[i] << endl;
// oppure
//    cout << "s[" << i << "]= " << s.at(i) << endl;
```

34

## Esempio: *Scorrere stringhe e array di stringhe*

```
int main()
{
    string
    strarr[4]={"Fabbrica","Italiana","Automobili","Torino"};
    string single="FIAT";

    for (int i=0;i<4;i++){
        cout << " single["<<i<<"]=" << single[i];
        cout << " strarr["<<i<<"]=" << strarr[i] << endl;
    }
}
```

35

## Esempio: *Scorrere stringhe e array di stringhe*

```
int main()
{
    string
strarr[4]={"Fabbrica","Italiana","Automobili","Merino"};
    string single="FIAT";

    for (int i=0;i<4;i++){
        cout << " single["<<i<<"]=" << single[i];
        cout << " strarr["<<i<<"]=" << strarr[i] << endl;
    }
}
```

Qui si estrae l'i-esimo carattere da una singola stringa.

Qui estraiamo l'i-esima stringa da un array di stringhe

36

## La classe **string** del C++

Continua...

37

## Esempio: *Simulazione di un C/C bancario #2*

Modifichiamo la classe MiniCCB per aggiungere la gestione di un PIN, richiesto per autorizzare ogni operazione su C/C.

Il PIN di una istanza della classe è impostato dal costruttore

Deve essere inserito dall'utente prima di ogni operazione e passato al metodo che la implementa;

La classe verifica il PIN e rifiuta l'operazione in caso sia sbagliato;

Tre errori di fila bloccano il C/C;

38

## Esempio: *Simulazione di un C/C bancario #2*

Classe MiniCCB

**Costruttori:**

**MiniCCB ()**

*Crea una istanza della classe e imposta il PIN «11111»;*

**MiniCCB (string PIN)**

*Crea una istanza della classe e imposta il PIN indicato;*

39

## Esempio: *Simulazione di un C/C bancario #2*

Classe MiniCCB

**Metodi:**

```
bool Deposito( double importo, string myPIN)
bool Prelievo( double importo, string myPIN)
bool Saldo(double &importo, string myPIN)
```

*Effettuano l'operazione indicata previo verifica del PIN. In caso di errore (PIN errato o conto bloccato) restituiscono false*

40

## Esempio: *Simulazione di un C/C bancario #2*

Classe MiniCCB

**Metodi:**

```
bool Bloccato()
```

*Restituisce true se il conto è bloccato (tre errori di PIN consecutivi)*

41

## Esempio: *Simulazione di un C/C bancario #2*

```

4  class MiniCCB {
5  private:
6      double dare;
7      double avere;
8      int tentativiPIN;
9      bool blocco;
10     string PIN;
11     ...

```

42

## Esempio: *Simulazione di un C/C bancario #2*

```

12     bool CheckPIN(string p)
13     {
14         bool r;
15         if (blocco)
16             return false;
17         if(p==PIN){
18             tentativiPIN=0;
19             return true;
20         }
21         tentativiPIN++;
22         if(tentativiPIN>3)
23             blocco=true;
24         return false;
25     }
26

```

43

## Esempio: *Simulazione di un C/C bancario #2*

```

27 public:
28     MiniCCB () {
29         dare=avere=0;
30         tentativiPIN=0;
31         PIN="11111";
32         blocco=false;
33     };
34     MiniCCB(string p) {
35         dare=avere=0;
36         tentativiPIN=0;
37         PIN=p;
38         blocco=false;
39     };

```

44

## Esempio: *Simulazione di un C/C bancario #2*

```

40     bool Saldo(double &importo, string p)
41     {
42         if(blocco || !CheckPIN(p))
43             return false;
44
45         importo=avere-dare;
46         return true;
47     };
48
49     bool Deposito(double importo, string p)
50     {
51         if(blocco || !CheckPIN(p))
52             return false;
53
54         avere+=importo;
55         return true;
56     };

```

45

## Esempio: *Simulazione di un C/C bancario #2*

```

57     bool Prelievo(double importo, string p)
58     {
59         if(blocco || !CheckPIN(p))
60             return false;
61         dare+=importo;
62         return true;
63     };
64
65     bool Bloccato()
66     {
67         return blocco;
68     }
69 };

```

46

## Esempio: *Simulazione di un C/C bancario #2*

```

70 int main()
71 {
72     double saldo=0, importo=0;
73     bool ancora=true;
74     int scelta=0;
75     string mioPIN;
76     MiniCCB Conto("12345");
77
78     cout << fixed << setprecision(2);
79     do {
...

```

47

## Esempio: *Simulazione di un C/C bancario #2*

```

125         case 3:
126             cout << "Inserisci il PIN: "
127             cin >> mioPIN;
128             if(!Conto.Saldo(importo,mioPIN)){
129                 cout << "L'operazione è fallita!" << endl;
130                 cout << "Verificare che il PIN sia corretto ";
131                 cout << "o contattare la banca« << endl;
132             }
133             else
134                 cout << "Saldo del CC: "<< importo << endl;
135             break;
136         } // switch (ancora)
137     } while(ancora);
138 }

```

48

## Esercizio: *Simulazione di un C/C bancario #2*

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

49



## Esercizio: *Simulazione di un C/C bancario #2*

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