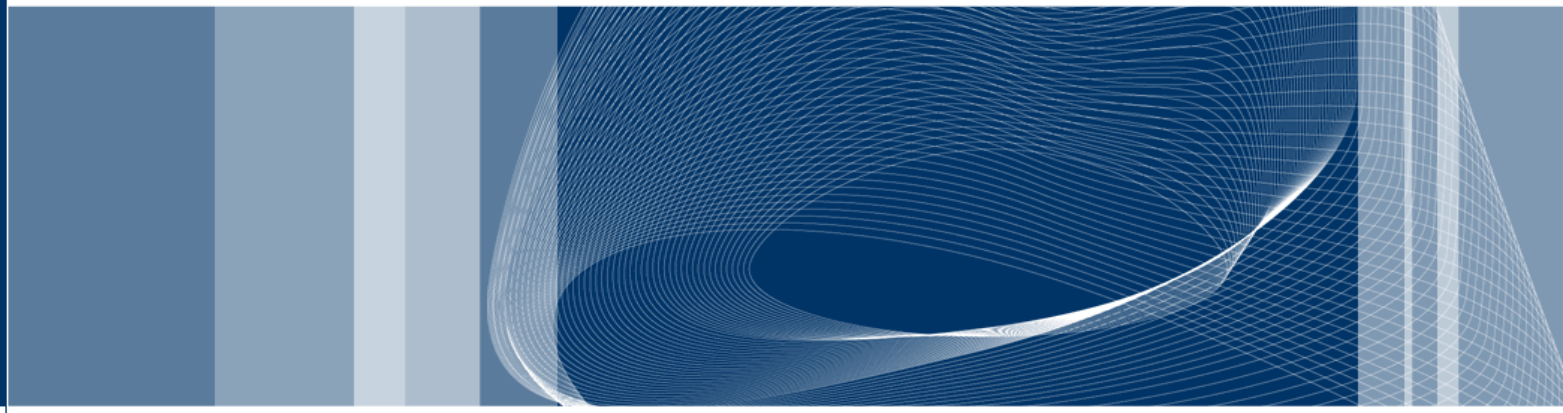
POLITECNICO DI MILANO



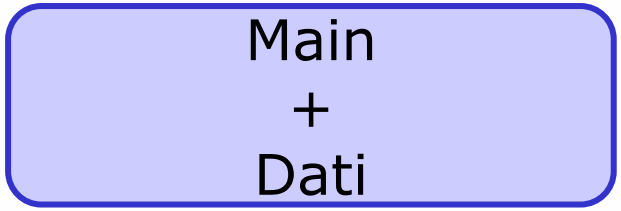
Algoritmi e Strutture Dati

Tutorial C++

Daniele Loiacono

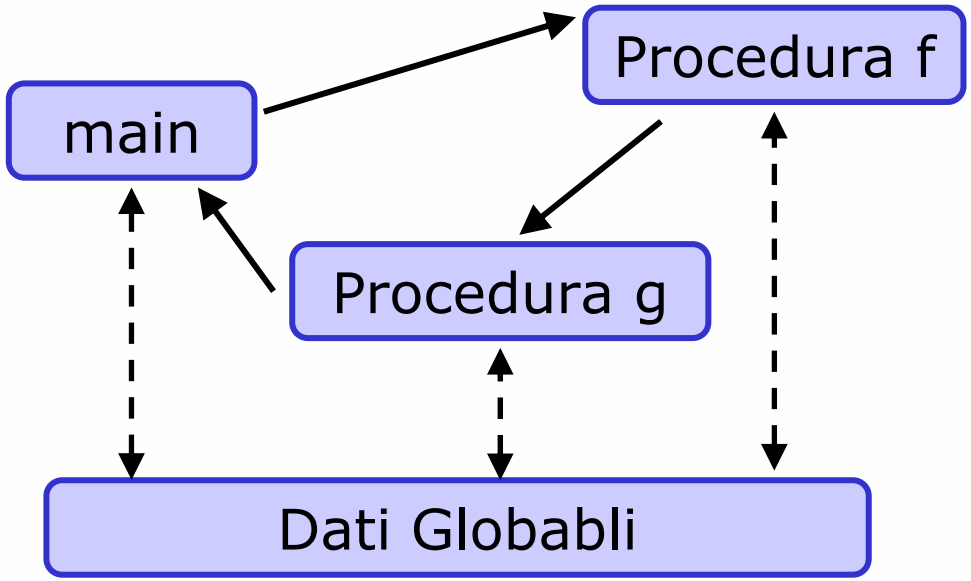
Paradigmi di programmazione

- Programmazione non strutturata



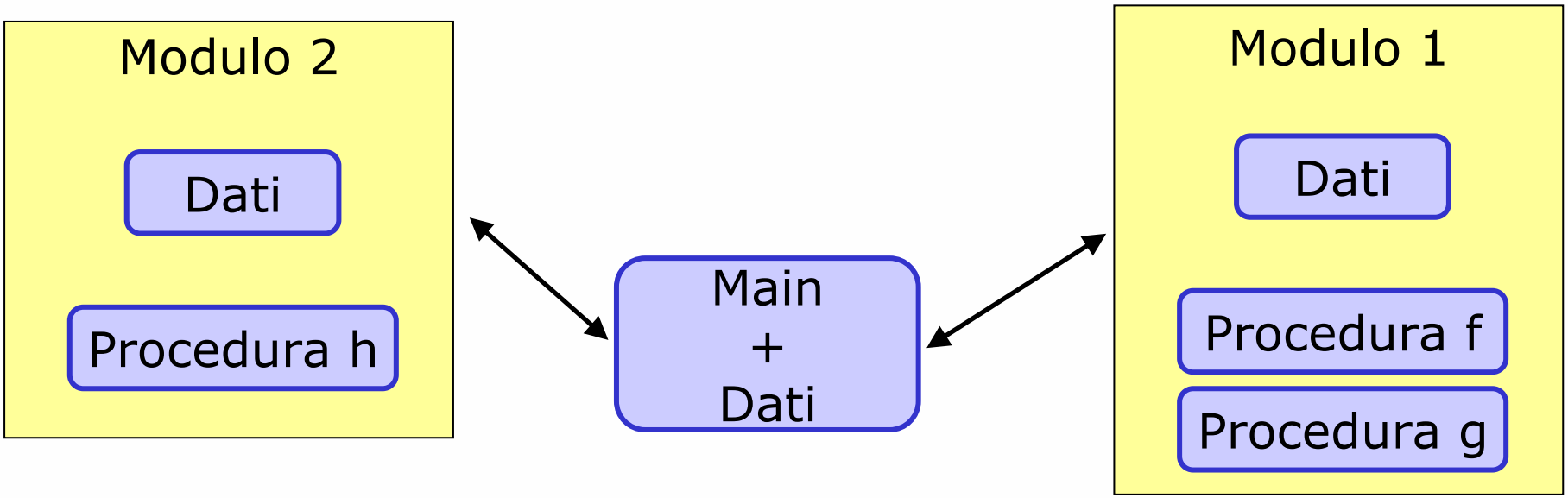
Paradigmi di programmazione

- Programmazione non strutturata
- Programmazione procedurale



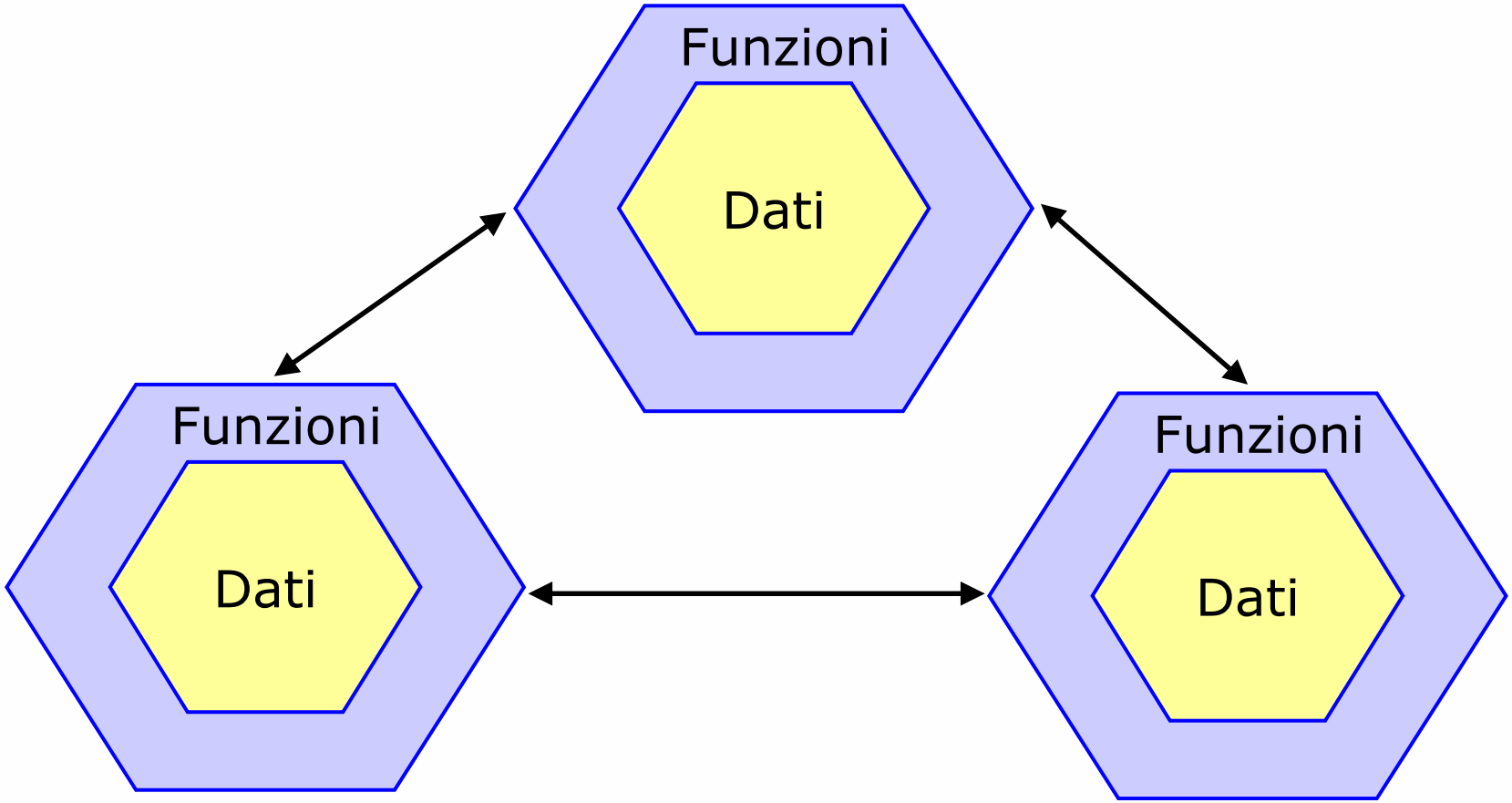
Paradigmi di programmazione

- Programmazione non strutturata
- Programmazione procedurale
- Programmazione modulare



Paradigmi di programmazione

- Programmazione non strutturata
- Programmazione procedurale
- Programmazione modulare
- Programmazione ad oggetti



Gli oggetti in C++

- Si utilizza il costrutto class:

```
class NomeClasse {
private:
  // attributi e metodi visibili solo all'interno
  // della classe

public:
  // attributi e metodi visibili da tutti

...
};
```

Esempio: la classe automobile


```
class Automobile{
private:
        int cilindrata;
        int numPorte;
        string marca;
public:
…
};
```

- Normalmente in C++ i dati non sono accessibile direttamente dall'esterno

```
Automobile a;
A.cilindrata=10;
```

- Come si accede ai dati?

Esempio: la classe automobile


```
class Automobile{
…
public:
       int getCilindrata (){ return cilindrata;}
       void setCilindrata (int cilindrata){
                 this->cilindrata = cilindrata;
       }
       string getMarca (){return marca; }
       void setMarca (int marca){this->marca = marca;}
       int getNumPorte ();
       void setNumPorte (int numPorte);
};

int Automobile::getNumPorte (){
    return numPorte;
}
void Automobile::setNumPorte (int numPorte){
     this->numPorte = numPorte;
}
```

Dichiarazione

Definizione

tipo_ritornato
nome_classe::nome_funzione
(parametri)

Costruttore

- L'inizializzazione della classe avviene attraverso il costruttore

```
class Automobile{
...
public:

        Automobile (int cilindrata, int numPorte, string marca){
                this->cilindrata = cilindrata;
                this->numPorte = numPorte;
                this->marca = marca;
        };
…
};
```

nome_classe (parametri)

Costruttore (2)

- Il costruttore viene chiamato una sola volta nella vita di un oggetto: quando l'oggetto viene istanziato
  - Nel caso delle variabili statiche durante l'inizializzazione
    ```
    Automobile a; //errore!
    Automobile a2(2000,5,"bmw");
    ```
  - Nel caso delle variabili dinamiche quando viene allocata la memoria
    ```
    Automobile *a = new Automobile(2000,5,"bmw");
    ```
- Cosa succede se un attributo è una classe a sua volta?
- Esempio

```
class Colore{
public:
     int codice;
     string nome;
     colore(int codice, string nome){
          this->codice = blu;
          this->nome = nome;
     }
};
```

Costruttore (3)


```
class Automobile{
private:
        int cilindrata;
        int numPorte;
        string marca;
        Colore colore;
public:

        Automobile (int cilindrata, int numPorte, string marca,
   int colCodice, int colNome ) : colore(colCodice,colNome) {
                    this->cilindrata = cilindrata;
                    this->numPorte = numPorte;
                    this->marca = marca;
        };
...
};
```

Distruttore


```
class Automobile{
...
public:
        ~Automobile (){…};
…
};
```

~nome_classe ()

- È una funzione speciale che viene chiamata una sola volta nella vita di un oggetto, quando viene deallocato
  - Nel caso delle variabili statiche quando escono dallo scope

```
int f(){
  Automobile a(2000,5,"bmw");
  …
}
```

  - Nel caso delle variabili dinamiche quando viene deallocata la memoria

```
Automobile *a = new Automobile(2000,5,"bmw");
…
delete a;
```

- Le funzioni di I/O per il terminale sono definite nella libreria **iostream**

```
#include <iostream>
```

  - Gli operatori **<<** e **>>** sono usati per definire la direzione del flusso
  - **cin**, **cout** e **cerr** rappresentano lo standard input, output e error del programma

```
#include <iostream>
int main()
{
  cout << “Hello, world !” << endl;
  return 0;
}
```

End of line

- Le funzioni di I/O che riguardano i file sono sono definite nella libreria **fstream** e funzionano in maniera del tutto analoga alle precedenti

I/O in C++: Esempio


```
int main()
{
    Automobile a(2000,5,"bmw");
    int numPorte;
    ofstream fileOut("out.txt");

    cout << "Numero porte: ";
    cin >> numPorte;
    a.setNumPorte(numPorte);
    cout << "Numero porte settato a " << a.getNumPorte() << endl;
    fileOut << "Numero porte settato a " << a.getNumPorte() << endl;
}
```

Namespace

- Per evitare che funzioni diverse (definite in librerie diverse) con lo stesso nome possano interferire (*name clash*), il C++ implementa il concetto di namespace

```
namespace mynames{
   int i;                                              // la mia
   dichiarazione di i
   float max(float, float);  // la mia dichiarazione di max
}

float mynames::max(float a, float b){        // implementazione della
   return (a>b) ? a : b;                               // funzione max
   appartenente
}
    // al namespace mynames
```

Namespace (2)

- Per utilizzare variabili e funzioni racchiuse in un namespace si può:
  - accedere all'intero namespace

```
using namespace mynames;
...
float r = max (2.1f, 5.3f);
```

  - accedere alla singola variabile o funzione

```
float r = mynames::max (2.1f, 5.3f);
```

  - dichiarare la singola funzione

```
using mynames::max;
...
float r = max (2.1f, 5.3f);
```

Template

- In C++ il costrutto template consente di definire classi parametrizzate rispetto a uno o più tipi di dato

```
template <typename T>
class NomeClasse {
private:
  // attributi e metodi visibili solo all'interno
  // della classe
   T ...
public:
  // attributi e metodi visibili da tutti

...
};
```

Template: parametri


```
template <typename T, size_t SIZE>
  class Stack {
  public:
    void push (const T& new_item);
    T top () const;
    void pop ();
    bool isEmpty () const;
    bool isFull () const;
  private:
    size_t top_;
    T stack_[SIZE];
  };
```

Ereditarietà

- Nuove classi sono create/derivate a partire da classi preesistenti
- La classe derivata eredita gli attributi e i metodi della classe definita precedentemente
- È possibile avere ereditarietà multipla o singola

```
class NomeClasse: public NomeSuperClasse {
private:
…
public:
…
};
```

Se la parola chiave public non è presente tutti gli attributi e i metodi della super classe diventano privati nella classe derivata

Ereditarietà: parola chiave virtual


```
class Automobile{
private:
        int cilindrata;
        int numPorte;
        string marca;
        Colore colore;
public:
        virtual void stampa() {
                cout << marca << " " << cilindrata
                cout << "cc di colore" << colore.nome;
                cout << " con " << numPorte << “ porte" << endl;
        }
…
};
```

Ereditarietà: overriding


```
class AutoSportiva : public Automobile{
private:
        float da0a100;
public:
        virtual void stampa() {
           Automobile::stampa();
           cout << "Da 0 a 100 km/h in " << da0a100 << " s" << endl;
        }

        AutoSportiva (int cilindrata, int numPorte, string marca,
                      int colCodice, string colNome, float da0a100):
                      Automobile(cilindrata,numPorte,marca,colCodice,
                      colNome){
                this->da0a100 = da0a100;
        }
};
```