

Pietro Ducange

Complementi di programmazione a oggetti in C++

a.a. 2020/2021

Funzioni e classi modello

**Si ringrazia la prof. Nicoletta De Francesco per aver messo a disposizione
la maggior parte delle slide utilizzate nella presente lezione**

Classi ed Oggetti

Classe:

Descrizione di un gruppo di oggetti con proprietà (attributi), comportamento (operazioni), relazioni e semantica comuni.

La classe è l'astrazione che:

- Enfatizza caratteristiche rilevanti
- Sopprime le altre caratteristiche

Oggetto:

Manifestazione concreta di un'astrazione.

Istanza di una classe.

Alcuni vantaggi della programmazione a oggetti

Incapsulamento

decomposizione

riuso

manutenzione

affidabilità

Meta-Programmazione

Possibilità di applicare lo stesso codice a **tipi diversi, parametrizzando i **tipi** utilizzati:**

Indipendenza degli algoritmi dai dati a cui si applicano: per esempio, un algoritmo di ordinamento può essere scritto una sola volta, qualunque sia il tipo dei dati da ordinare.

Funzioni modello

```
int i_max(int x, int y) {  
    return (x>y) ? x : y;  
};  
  
double d_max(double x, double y) {  
    return (x>y) ? x : y;  
};  
  
void main() {  
    int b; double c;  
    // ...  
    a= i_max(3,b);  
    d=d_max(3.6,c);  
}
```

Le due funzioni hanno **la stessa definizione** con tipi diversi

Funzioni modello: costruito template

```
#include<iostream.h>

template<class tipo>
tipo max(tipo x, tipo y) {
    return (x>y) ? x : y;
}

void main() {
    int b; double c;
    // ...
    b= max(3,b);

    // tipo=int max<int>(int,int)

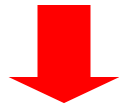
    c=max(3.6,c);

    // tipo = double max<double>(double,double)
}
```


Funzioni modello: compilazione

Risultato della compilazione

a = max(3,b);



tipo=int

max<int>(int x, int y) { return (x>y) ? x : y;}

d = max(3.6,c);



tipo=double

max<double>(double x, double y) {return (x>y) ? x : y;}

Funzioni modello: argomenti impliciti

```
template<class tipo>  
tipo max(tipo x, tipo y) .....
```

```
void main() {  
    int b=2; double c=6.0, d; int array[2]={3,4};  
  
    cout << max(array[0],b);    // OK: int max<int >(int,int)  
  
    d = max(3.6,c);  
        // OK: double max<double>(double, double)  
  
    b = max(3.6,c);  
        // OK: double max<double>(double, double) e conversione  
  
    // d = max(3,c);    errore: non si deduce il tipo:  
        // 3 e' intero, c e' double  
  
}
```

I tipi devono essere deducibili dalla chiamata

Esempio di funzione modello (I)

```
template<class tipo>
void primo ( tipo *x ) {
    tipo y= x[0] ;
    cout << y << endl;
};
```

```
void main() {
    int array1[2]={3,4};
    double array2[2]={3.5,4.8};

    primo(array1);
```

```
    // 3      tipo=int      void primo<int>(int*)
```

```
    primo(array2);
```

```
    // 3.5    tipo=double    void primo<double>(double*)
```

```
}
```

Esempi di funzioni modello (I) (cont.)

primo(array1);



```
void primo<int> ( int *x ) {  
    int y= x[0] ;  
    cout << y << endl;  
};
```

primo(array2);



```
void primo< double > (double *x ) {  
    double y= x[0] ;  
    cout << y << endl;  
};
```

Esempi di funzioni modello (II)

```
template<class tipo>  
void primo ( tipo x ) {  
    cout << x[0] << endl;  
};  
  
void main() {  
    int array1[2]={3,4};  
    double array2[2]={3.5,4.8};  
  
    primo(array1);  
  
        // 3    tipo=int*    void primo<int*>(int*)  
  
    primo(array2);  
  
        // 3.5    tipo=double*    void primo<double*>(double*)  
  
    }
```

Esempi di funzioni modello (II) (cont.)

primo(array1);



```
void primo<int*> ( int * x ) {  
    cout << x[0] << endl;  
};
```

primo(array2);



```
void primo<double*> ( double * x )  
{  
    cout << x[0] << endl;  
};
```

Esempi di funzioni modello (III)

```
template<class tipo>  
void primo ( tipo x ) {  
    tipo y= x[0] ;  
    cout << y << endl;  
};
```

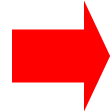
```
void main() {  
    int array1[2]={3,4};  
  
    primo(array1);
```

```
    //  tipo=int*    void primo<int*>(int*)  
    //errore non si può convertire un int in int*!!!
```

```
}
```

Esempi di funzioni modello (III) (cont.)

primo(array1);



```
void primo<int*> ( int*x ) {  
    int * y= x[0] ;  
    cout << y << endl;  
};
```

errore: non si può convertire un int in int*!!!

funzioni modello con più parametri

```
template<class tipo1, class tipo2>  
tipo1 max(tipo1 x, tipo2 y) {  
    return (x>y) ? x : y;  
}
```

```
void main() {  
  
    int b=2; double c=6;  
  
    cout << max(3,b);    // int max<int,int>(int,int)  
  
        // tipo1=int, tipo2= int  
  
    b = max(3,c);    // int max<int,double>(int,double)  
  
        // tipo1=int, tipo2= double  
  
}
```

funzioni modello con più parametri

```
template<class tipo1, class tipo2, class tipo3>  
tipo1 nuovomax(tipo2 x, tipo3 y) {  
    return (x>y) ? x : y;  
}
```

```
void main() {  
    int b; double c=6;  
  
    b = nuovomax(3,c);
```

```
    // NO: tipo1=? , tipo2=int, tipo3=double
```

```
}
```

Funzioni modello: parametri espliciti

```
template<class tipo>  
tipo max(tipo x, tipo y) {  
    return (x>y) ? x : y;  
}
```

```
void main() {  
    double d;  
    cout << max<int>(3,5.5);  
        // 5 max<int>(int,int); conversione del parametro  
  
    cout << max<double>(3,5.5);  
        // 5.5 max<double>(double,double) conversione  
        //del parametro  
  
    d= max<int>(3,5.5);  
        // max<int>(int,int); conversione del valore  
        // assegnato: 5  
}
```

Funzioni modello: parametri espliciti e impliciti

```
template<class tipo1, class tipo2, class tipo3>
tipo1 fun(tipo2 x, tipo3 y) {
    ....
}
```

Gli argomenti espliciti sono indicati nell'ordine del template

```
fun<int>(9,8.8);  // tipo1= int : int fun<int,int,double>
```

```
fun<int,double>(9,8.8);  // tipo1=int, tipo2=double :  
                        int fun<int,double,double>
```

```
fun<int,int,double>(.,..);  // int fun<int,int,double>
```

```
fun(9,8);  // errore tipo3=tipo2=int, tipo1?
```

Funzioni modello: parametri costanti

```
template<int n, double m >  
void funzione(int x=n){  
    double y=m;  
    int array[n];  
    .....  
}
```

```
void main () {  
    funzione<1+2,2>(8); // n=3, m=2      funzione<3,2>(int)  
  
    funzione<2,2>(9); // n=2, m=2      funzione<2,2>(int)  
}
```

I parametri costanti sono necessariamente espliciti:

Le istanziazioni di n e m devono essere ESPRESSIONI COSTANTI

Funzioni modello: parametri costanti (cont.)

funzione<1+2,2>(8);



```
void funzione<3,2>(int x=3){  
    double y=2;  
    int array[3];  
    ....  
}
```

funzione<2,2>(9);



```
void funzione<2,2>(int x=2){  
    double y=2;  
    int array[2];  
    ....  
}
```

Funzioni modello: parametri costanti e no

```
template< int n, class T>  
int gt(T x){  
return x>n;  
}
```

```
void main(){  
    cout << gt<50+6>(101);
```

```
    // 1  n=56, T=int  int gt<56,int>(int)
```

```
    //  risoluzione implicita di T
```

```
    cout << gt<8, double>(7);
```

```
    // 0  n=8, T=double  int gt<8, double>(double)
```

```
    //  risoluzione esplicita di T
```

Funzioni modello: parametri costanti e no (cont.)

gt<50+6>(101);



```
int gt<56,int>(int x){  
    return x>56;  
}
```

gt<8, double>(7);



```
int gt<8,double>(double x){  
    return x>8;  
}
```


Funzioni modello con variabili statiche

```
template<class tipo>
tipo maxT(tipo x, tipo y) {
    static int a; a++; cout << a << endl;
    return (x>y) ? x : y;
}
```

```
void main(){
```

```
cout << maxT<int>(101,102) << endl;           // 1 102
cout << maxT<int>(101,102)<< endl;             // 2 102
cout << maxT<double>(101,102) << endl;         // 1 102
}
```

Ogni istanza della funzione ha la sua variabile statica

Dichiarazione e definizione di template

// file templ.h

```
template<class tipo>
void boh(tipo x){
    // ... definizione
}
```

// file main

#include"templ.h"

```
void main() {
    //..
    boh(6);

    // ..
}
```

Una funzione modello **non può essere compilata senza conoscere le chiamate: non si può fare una compilazione separata**

Anche le classi possono essere definite come classi modello:

```
template<class tipo1, class tipo2, int n .....>  
class obj { ....
```

I parametri in questo caso sono **sempre espliciti**

Classi modello: stack

```
class stack{
    int size;
    int * p;
    int top;
public:
    stack(int n){
        size = n;
        p = new int [n];
        top = -1;
    };

    ~stack() { delete [] p; };

    int empty(){
        return (top==-1); };

    int full(){
        return (top==size-1); }; }
```

```
int push(int s){
    if (top==size-1) return 0;
    p[++top] = s;
    return 1;
};

int pop(int& s){
    if (top==-1) return 0;
    s = p[top--];
    return 1;
}
```

Stack modello parametrico rispetto al tipo degli elementi

//file stack.h

template<class tipo>

class stack {
 int size;
 tipo* p;
 int top;

public:

stack(int n){
 size = n;

p = new tipo [n];
 top = -1; };

~stack() { delete [] p; };

int empty() {
 return (top==-1);};

int full() {
 return (top==size-1);};

int push(tipo s) {
 if (top==size-1) return 0;
 p[++top] = s;
 return 1; };

int pop(tipo& s){
 if (top==-1) return 0;
 s = p[top--];
 return 1; }
};

Stack modello parametrico rispetto al tipo degli elementi

```
#include "stack.h"
```

```
void main(){
```

```
    stack<int> s1 (20), s2 (30);
```

```
    stack<char> s3 (10);
```

```
    stack<float> s4 (20);
```

```
    s1.push(3);
```

```
    s3.push('a');
```

```
    s4.push(4.5);
```

```
}
```

Stack modello parametrico rispetto al tipo degli elementi

```
class persona {  
    char nome [20];  
    int eta;  
public:  
    persona() {}  
    persona (char* n, int e){  
        strcpy(nome,n);  
        eta=e;}  
};  
  
void main() {  
    persona p ("anna",22);  
    stack<persona> pila(10);  
    pila.push(p);  
}
```

Classi modello

```
class stack{
    int size;
    int * p;
    int top;
public:
    stack(int);
    ~ stack();
    int empty();
    int full();
    int push(int);
    int pop(int&);
};

stack::stack(int n){
    size = n;
    p = new int [n];
    top = -1; }
```

```
stack::~~stack(){ delete [] p; }

int stack::empty(){
    return (top== -1); }

int stack::full(){
    return (top==size-1); }

int stack::push(int s){
    if (top==size-1) return 0;
    p[++ top] = s;
    return 1; }

int stack::pop(int& s){
    if (top== -1) return 0;
    s =p[top--];
    return 1; }
```


Classi modello

// file stack.h

template<class tipo>

class stack{

int size;

tipo * p;

int top;

public:

stack(int);

~ stack();

int empty();

int full();

int push(**tipo**);

int pop(**tipo**&);

};

template<class tipo>

stack<tipo>::stack(int n){

size = n;

p = new **tipo** [n];

top = -1; }

Classi modello (cont.)

```
template<class tipo>  
stack<tipo>::~~stack(){ delete [] p; }
```

```
template<class tipo>  
int stack<tipo>::empty(){ return (top==-1); }
```

```
template<class tipo>  
int stack<tipo>::full(){ return (top==size-1); }
```

```
template<class tipo>  
int stack<tipo>::push( tipo s ){  
    if (top==size-1) return 0;  
    p[++top] = s;  
    return 1; }
```

```
template<class tipo>  
int stack<tipo>::pop( tipo& s ){  
    if (top==-1) return 0;  
    s = p[top--];  
    return 1; }
```

Classi modello

// file stack.h
// contiene dichiarazioni e definizioni

```
template<class tipo>
class stack{
    // ..
public:
    ...
};

// definizioni
```

// file principale

```
# include "stack.h"
```

```
...
```

**bisogna includere
anche la definizione**

Stack parametrico anche rispetto alla dimensione

```
template<class tipo, int size>
class stack {
    tipo* p;
    int top;
public:
    stack(){
        p = new tipo [size];
        top = -1; };

    ~stack() { delete [] p; };

    int empty() {
        return (top==-1);};
    int full() {
        return (top==size-1);};

    int push(tipo s) {
        if (top==size-1) return 0;
        p[++top] = s;
        return 1; };
};
```

```
int pop(tipo& s){
    if (top==-1) return 0;
    s = p[top--];
    return 1; }
};
```

...

```
stack<int,10> pila1;
stack<double,20> pila2;

stack<char,20> pila3;
stack<int,20> pila4;
```

Esempio

```
template<class tipo, int size>  
class stack {  
// ..  
  
};
```

```
...  
stack<int, 100> pila1;  
stack<int, 300> pila2;
```

```
stack<int,100>* ptr = &pila1;
```

```
// ptr = &pila2;   errore
```

stack<int, 300> e stack<int, 100> sono tipi diversi

Classi modello con membri statici

```
template<int n>
class cmod{
    static int istanze;
    int m;
public:
    cmod();
    void stampa();
};
```

```
template<int n>
int cmod<n>::istanze=0;
```

```
template<int n>
cmod <n>::cmod(){
    m=n;
    istanze ++;
};
```

```
template<int n>
void cmod <n>::stampa(){
    cout << istanze << '\\t' << m << endl;
}
```

```
void main(){
    cmod<9> nove_a;
    nove_a.stampa();
```

```
    cmod<7> sette;
    sette.stampa();
```

```
    cmod<9> nove_b;
    nove_b.stampa();
```

```
}
```

Classi modello con membri statici

```
template<int n>
class cmod{
    static int istanze;
    int m;
public:
    cmod();
    void stampa();
};
```

```
template<int n>
int cmod<n>::istanze=0;
```

```
template<int n>
cmod <n>::cmod(){
    m=n;
    istanze ++;
};
```

```
template<int n>
void cmod <n>::stampa(){
    cout << istanze << '\\t' << m << endl;
}
```

```
void main(){
    cmod<9> nove_a;
    nove_a.stampa();

    // 1 9

    cmod<7> sette;
    sette.stampa();

    // 1 7

    cmod<9> nove_b;
    nove_b.stampa();

    // 2 9
}
```