Laurea triennale in Informatica

modulo (CFU 6) *di*

Programmazione II e Lab.

prof. Mariarosaria Rizzardi

Centro Direzionale di Napoli – Isola C4
stanza: n. 423 – IV piano Lato Nord
tel.: 081 547 6545
email: mariarosaria.rizzardi@uniparthenope.it

Programmazione generica in C++: cenni ai template

In C++ è possibile definire delle *funzioni* o *classi generiche* ovvero che abbiano come parametro il tipo di dato, potendo usare lo stesso codice per tipi differenti. In tal modo il C++ fornisce un metodo per creare un ***polimorfismo parametrico***.

Una funzione o classe generica si chiama ***funzione*** o ***classe template***.

Esempio

in P2_05_Cpp

```
#include <iostream>
#include <vector>
using namespace std;
int main()
{   vector <int> v = {7, 5, 16, 8};
    v.pop_back();
    v.push_back(25);
    v.push_back(13);
    for (auto n : v)
        cout << n << endl;
    vector <int> v2;
    v2=v;
    return 0;
}
```

la classe **vector** è un **template**

Laboratorio di Programmazione in Informatica
prof. Maria Rizzardi

Sintassi e uso dei Template

Funzione generica

```
template <class Ttype> ret_type func_name(parameter list)
oppure
template <typename Ttype> ret_type func_name(parameter list)
{    // corpo della funzione
}
```

Esempio 1a

esempio1a.hpp

```
template <typename X> void swapvar(X &a, X &b)
{    X tmp = a;
     a = b;
     b = tmp;
}
```

esempio1a.cpp

```
#include <iostream>
#include "esempio1a.hpp"
using namespace std;

int main()
{
    int i=10, j=20;
    cout << "Prima: i, j: " << i << ',' << j << '\n';
    swapvar<int> (i, j);
    cout << "Dopo:  i, j: " << i << ',' << j << '\n';

    double x=10.1, y=23.3;
    cout << "Prima: x, y: " << x << ',' << y << '\n';
    swapvar<double> (x, y);
    cout << "Dopo:  x, y: " << x << ',' << y << '\n';

    char a='A', b='B';
    cout << "Prima: a, b: " << a << ',' << b << '\n';
    swapvar<char> (a, b);
    cout << "Dopo:  a, b: " << a << ',' << b << '\n';
    return 0;
}
```

`swapvar (i, j);`

`swapvar (x, y);`

`swapvar (a, b);`

```
Prima: i, j: 10,20
Dopo:  i, j: 20,10
Prima: x, y: 10.1,23.3
Dopo:  x, y: 23.3,10.1
Prima: a, b: A,B
Dopo:  a, b: B,A
```

Il compilatore automaticamente genera il codice corretto per il tipo di dato usato nella chiamata alla funzione (over-loading automatico)

Sintassi e uso dei Template

Funzione generica: esempio 1b

esempio1b.hpp

```
template <class X> X maxelem(X a, X b)
{
    if ( a > b )
        return a;
    else
        return b;
}
```

esempio1b.cpp

```
#include <iostream>
#include "esempio1b.hpp"
using namespace std;

int main()
{
    int i=10, j=20;
    cout << "input:  i, j: " << i << ',' << j << endl;
    cout << "\toutput: " << maxelem(i,j) << endl << endl;

    double x=10.1, y=23.3;
    cout << "input:  x, y: " << x << ',' << y << endl;
    cout << "\toutput: " << maxelem(x,y) << endl << endl;

    char a='A', b='B';
    cout << "input:  a, b: " << a << ',' << b << endl;
    cout << "\toutput: " << maxelem(a,b) << endl << endl;

    return 0;
}
```

```
input:  i, j: 10,20
        output: 20

input:  x, y: 10.1,23.3
        output: 23.3

input:  a, b: A,B
        output: B
```

Laboratorio di ... Informatica
Rizzardi

Sintassi e uso dei Template

Funzione generica: esempio 1c

esempio1c.hpp

```
#include <iostream>
using namespace std;

template <typename X, typename Y> void show(X a, Y b)
{
    cout << "X a = " << a << ", Y b = " << b << endl;
}
```

esempio1c.cpp

```
#include "esempio1c.hpp"

int main()
{
    int i=10, j=20;
    show<int,int>(i,j);

    double x=10.1; int y=23;
    show(x,y);

    char a='A'; float b=1.5f;
    show(a,b);

    return 0;
}
```

```
X a = 10, Y b = 20
X a = 10.1, Y b = 23
X a = A, Y b = 1.5
```

Programmazione e Laboratorio di ... A.A. 2019/2020 Massimo Rizzardi

Sintassi e uso dei Template

Classe generica

```
template <class Ttype> class class_name
oppure
template <typename Ttype> class class_name
{
     // definizione di classe
}
```

Istanzia oggetto di classe generica

```
class_name <type> ob;
```

Il compilatore automaticamente genera il codice corretto per il tipo di dato usato nella chiamata alla funzione (overloading automatico)

Esempio 2

calcola.hpp

```
template <typename T> class calcola {
  public:
    T add(T x, T y);
    T mult(T x, T y);
};

template <typename T>
T calcola<T>::add(T x, T y)
{   return x+y;   }

template <typename T>
T calcola<T>::mult(T x, T y)
{   return x*y;   }
```

esempio2.cpp

```
#include <iostream>
#include "calcola.hpp"
using namespace std;

int main()
{
    calcola<int> *p1 = new calcola<int>;

    int i=10, j=20;

    cout << "input:  i, j: " << i << ',' << j << endl;
    cout << "\tadd : " << p1->add(i,j) << endl;
    cout << "\tmult: " << p1->mult(i,j) << endl << endl;
    delete p1;

    return 0;
}
```

```
input:  i, j: 10,20
        add : 30
        mult: 200
```