C++ avanzato: template e librerie STL

CLASSE 4E

A.S. 2018-2019

«INFORMATICA PER ISTITUTI TECNICI TECNOLOGICI» Vol. A

pagg. 264-285

Prof. **Barbara Cattani**

Argomenti

- Template di funzioni
- Libreria standard STL
- Contenitore per le stringhe (argomento affrontato nella classe 3)
- Contenitore vector
- Iteratori
- Contenitori List e Map
- Algoritmi standard

Template di funzioni

- Il termine **template** indica un modello utilizzabile per un insieme di problemi dello stesso tipo e avente parti o componenti che contengono parametri, in modo che possano essere adattate per specifiche esigenze

- Consentono di inserire in un programma funzioni che contengono un parametro, che può poi essere sostituito con un tipo al momento della chiamata della funzione stessa.

Template di funzioni

- Esempio: scambio del contenuto di due variabili

```
void ScambiaInteri(int& a, int& b)
{
  int temp;

  temp = a;
  a = b;
  b = temp;
};

void ScambiaStringhe(string& a, string& b)
{
  string temp;

  temp = a;
  a = b;
  b = temp;
};
```

Template di funzioni

```
template<typename T> void Scambia(T& a, T& b)
{
  T temp;

  temp = a;
  a = b;
  b = temp;
};
```

Nel *main* o nella *funzione chiamante*:

```
   int x, y;
   string s1, s2;

   Scambia(x,y);
   Scambia(s1,s2);
```

Libreria standard STL

- **STL** (*Standard Template Library*): libreria dei template standard.

- Creata da **Alexander Stepanov** e **Meng Lee**, ricercatori di HPL (Hewlett-Packard-Laboratories) e diventata una parte dello standard ANSI/ISO del linguaggio C++
- **Programmazione generica**: dispone di strutture di dati e di algoritmi che possono essere adattati e applicati in numerosi contesti di elaborazione.
- Le componenti fondamentali della **STL**:
  - contenitori
  - algoritmi
  - iteratori

Contenitori

- **Containers**: oggetti che contengono altri oggetti. Le comuni strutture dati (vettori,liste, code, stack ...)
- Implementati per mezzo di template
- Progettati in modo da avere metodi uguali che si adattano alle specificità del container (*polimorfismo*)
- Alcuni containers nella STL:

```
vector

list

queue

deque

stack

map    // container associativo

// per utilizzarli includere
// file header opportuno

#include <list>
...
list<int> numeri;
```

Contenitore Vector

- *string* è una classe della libreria standard che rappresenta un caso particolare di **contenitore**, costituito da un array di caratteri.
- *vector* contenitore standard utilizzato nei problemi che richiedono l'organizzazione dei dati in array dati omogenei tra loro.

```
#include <vector>
```

```
vector<int> v;
```

Contenitore Vector

- Alcuni **Metodi** (o funzioni membro) della classe Vector

  - **push_back(dato)**
  - **pop_back()**
  - **clear()**
  - **size()**
  - **empty()**

Contenitore Vector

- **Esempio:** inserire da tastiera un elenco di numeri interi e visualizzarli

```
1  // CVettore.cpp: contenitore vector
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5  // funzione principale
6  int main()
7  {
8   vector<int> v;
9   int d;
10  // inserimento componenti
11  cout << "Numero (0=fine): ";
12  cin >> d;
13  //aggiunge in CODA alle componenti del vettore già presenti
14  while (d != 0) {
15    v.push_back(d);
16    cout << "Altro numero (0=fine): ";
17    cin >> d;
18  }
19  // visualizza componenti
20  for (int i=0; i<v.size(); i++)
21  cout << v[i] << endl;
22  return 0;
23 }
```

Contenitore Vector

- **Esempio:** inserire da tastiera un elenco di numeri interi e visualizzarli

```
1  // CVettore.cpp: contenitore vector
2  #include <iostream>
3  #include <vector>
4  using namespace std;
5  // funzione principale
6  int main()
7  {
8   vector<int> v;
9   int d;
10  // inserimento componenti
11  cout << "Numero (0=fine): ";
12  cin >> d;
13  //aggiunge in CODA alle componenti del vettore già presenti
14  while (d != 0) {
15    v.push_back(d);
16    cout << "Altro numero (0=fine): ";
17    cin >> d;
18  }
19  // visualizza componenti
20  for (int i=0; i<v.size(); i++)
21  cout << v[i] << endl;
22  return 0;
23 }
```

Iteratori

- **Iterators** sono una generalizzaione dei puntatori e consentono di attraversare un container qualunque esso sia e operare con gli elementi. Intermediari tra containers e algoritmi
- Due metodi:

  begin() restituisce un iteratore che punta al primo elemento del contenitore

  end() restituisce un iteratore che punta alla posizione successiva all'ultimo elemento del contenitore

Iteratori

- Un iteratore deve essere dichiarato con un'istruzione che lo associa al contenitore secondo la seguente **sintassi** generale

contenitore<T>::iterator nome;

- **Esempio**: Iteratore per un vettore di interi

```
vector<int>::iterator i;
```

Iteratori

- **Esempio**: Iteratore per un vettore di interi

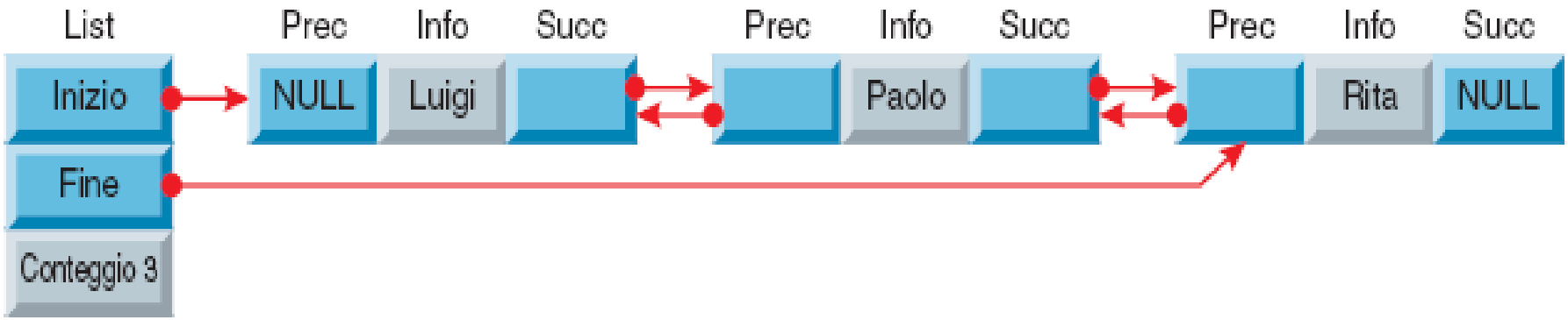
```
vector<int> v;
vector<int>::iterator i;

for (i=v.begin(); i!=v.end(); i++)
  cout << *i << endl;
```

Alla variabile *i* viene applicata l'aritmetica dei puntatori per passare dal valore iniziale v.begin() al valore finale v.end() con incremento unitario

La notazione **i* indica come per i puntatori, il valore della variabile puntata

Contenitore List

- **Lista bidirezionale** dove ogni nodo della lista contiene, oltre alla parte dati, due puntatori, uno all'elemento successivo e uno all'elemento precedente.




```
#include <list>
```

```
list<int> l;
```

Contenitore List

- Alcuni **Metodi** (o funzioni membro) della classe Vector

  - **push_back()**
  - **push_front()**
  - **size()**
  - **pop_back()**
  - **pop_front()**
  - **clear()**
  - **empty()**
  - **sort()**
  - **reverse()**
  - **merge()**

Contenitore List

- **Esempio:** memorizzare in una lista un insieme di nomi, acquisendoli da tastiera. Visualizzare poi l'elenco dei nomi prima in ordine alfabetico e poi in ordine inverso

```
1  // CLista.cpp: contenitore list
2  #include <iostream>
3  #include <list>
4  #include <string>
5  using namespace std;
6  // funzione principale
7  int main()
8  {
9      list<string> elenco;
10     string nome;
11 // inserimento componenti
12     cout << "Nome (*=fine): ";
13     cin >> nome;
14     while (nome != "*") {
15         elenco.push_back(nome);
16         cout << "Altro nome (*=fine): ";
17         cin >> nome;
18     }
19 //attivazione iteratore
20     list<string>::iterator i;
21     elenco.sort();
22     for (i=elenco.begin(); i!=elenco.end(); i++)
23         cout << *i << endl;
24     elenco.reverse();
25     for (i=elenco.begin(); i!=elenco.end(); i++)
26         cout << *i << endl;
27     return 0;
28 }
```

Contenitore Map

- **Struttura dati** organizzata come un insieme di coppie di valori: il primo rappresenta un valore **chiave** univoco all'interno della mappa, il secondo rappresenta il valore associato.

chiave 1	valore 1
chiave 2	valore 2
…	….
chiave *n*	valore *n*

Contenitore Map

- Esempio di **contenitore associativo** della libreria STL.

```
#include <map>
```

```
map<int, string> m;
```

Contenitore Map

- Esempio

```
1  // CMap.cpp: contenitore map
2  #include <iostream>
3  #include <map>
4  #include <string>
5  using namespace std;
6  // funzione principale
7  int main()
8  {
9      map<int, string> note;
10     int freq;
11     string nomenota;
12     // inserimento note
13     cout << "Frequenza della nota (0=fine): ";
14     cin >> freq;
15     while (freq != 0) {
16         cout << "Nome della nota: ";
17         cin >> nomenota;
18         note[freq] = nomenota;
19         cout << "Frequenza della nota (0=fine): ";
20         cin >> freq;
21     }
22     // visualizza le note: prima il nome, poi la frequenza
23     map<int, string>::iterator i;
24     for (i = note.begin(); i != note.end(); i++)
25         cout << i->second << ": " << i->first << " Hz" << endl;
26     return 0;
27 }
```

Algoritmi standard

- Gli **algoritmi** sono funzioni che implementano procedimenti di uso comune per la gestione dei contenitori.
- Richiedono inclusione header:

```
#include <algorithm>
```

Algoritmi standard

- Algoritmo **find()**

cerca un valore all'interno di un intervallo di un contenitore e restituisce un iteratore corrispondente alla posizione trovata:

```
p = find(v1.begin(), v1.end(), cercato);
```

- Algoritmo **count()**

conta gli elementi di un contenitore che hanno un valore uguale a un valore specificato.

```
int conta =
count(prov.begin(), prov.end(), cercata);
```

Algoritmi standard

- Esempio:

```
1  // CVettoreConta.cpp: contenitore vector e algoritmo count
2  #include <iostream>
3  #include <vector>
4  #include <algorithm>
5  #include <string>
6  using namespace std;
7  // funzione principale
8  int main()
9  {
10     vector<string> elenco;
11     string specie;
12     // inserimento componenti
13     cout << "Quale specie (*=fine): ";
14     cin >> specie;
15     while (specie != "*") {
16         elenco.push_back(specie);
17         cout << "Altra specie (*=fine): ";
18         cin >> specie;
19     }
20     // conteggio della specie richiesta
21     string cercata;
22     cout << "Specie da contare: ";
23     cin >> cercata;
24     cout << "Risultato conteggio = "
25         << count(elenco.begin(), elenco.end(), cercata)
26         << endl;
27     return 0;
28 }
```

Algoritmi standard

- Algoritmo **for_each()**

applica il codice contenuto in una funzione agli elementi di un contenitore:

```
for_each(v.begin(),v.end(),f)
```

Algoritmi standard

- Esempio:

```
 1  // CorsiUniv.cpp: codici dei corsi universitari
 2  #include <iostream>
 3  #include <list>
 4  #include <vector>
 5  #include <algorithm>
 6  #include <string>
 7  using namespace std;
 8  struct corso {
 9      string descrizione;
10      char anno;
11      string docente;
12  };
13  list<corso> daticorsi;
14  vector<string> codici;
15  // prototipi delle funzioni
16  void CaricaCorsi();
17  void CreaCodice(corso);
18  void VisualizzaCodici();
19  // funzione principale
20  int main()
21  {
22      CaricaCorsi();
23      for_each(daticorsi.begin(), daticorsi.end(), CreaCodice);
24      VisualizzaCodici();
25      return 0;
26  }
```

Algoritmi standard

- Esempio:

```
27  // caricamento corsi
28  void CaricaCorsi()
29  {
30      corso c;
31  // inserimento dati
32      cout << "Descrizione del corso (*=fine): ";
33      cin >> c.descrizione;
34      while (c.descrizione != "*") {
35          cout << "Anno del corso (1-3): ";
36          cin >> c.anno;
37          cout << "Cognome del docente: ";
38          cin >> c.docente;
39          daticorsi.push_back(c); // aggiunge alla lista
40          cout << "Altra descrizione (*=fine): ";
41          cin >> c.descrizione;
42      }
43  } // CaricaCorsi
44  void CreaCodice(corso c)
45  {
46      string cod;
47      cod = c.descrizione.substr(0,3) + c.anno + c.docente.substr(0,3);
48      codici.push_back(cod);
49  } // CreaCodice
50  // visualizzazione codici creati
51  void VisualizzaCodici()
52  {
53      vector<string>::iterator i;
54      cout << "--- Elenco codici dei corsi ---" << endl;
55      for (i=codici.begin(); i!=codici.end(); i++)
56          cout << *i << endl;
57  } // VisualizzaCodici
```

Algoritmi standard

- Algoritmo **copy()**

Copia gli elementi di un contenitore, secondo un intervallo specificato in un altro contenitore:

```
vector <int> v1;      → 1  2  3  1  1  4  5  7
vector <int> v2(3);

//copy: copia i primi tre elementi
copy(v.begin(), v.begin()+3, v2.begin());
  →1  2  3 in v2
```

Algoritmi standard

- Algoritmo **replace()**

Sostituisce tutti gli elementi di un contenitore, compresi in un intervallo specificato e aventi un valore prefissato, con un nuovo valore fornito come parametro all'algoritmo:

```
vector <int> v1;      → 1  2  3  1  1  4  5  7


//replace: cambia i valori 1 con il valore 100
replace(v1.begin(),v1.end(), 1, 100);
  →100  2  3  100  100  4  5  7
```

Algoritmi per l'ordinamento

- Algoritmo **sort()** mette in ordine crescente gli elementi.

```
vector<string> termini;
```

```
sort(termini.begin(), termini.end());
```

- Per l'ordine descrescente si usa l'algoritmo generico **reverse()**.

```
sort(termini.begin(), termini.end());
reverse(termini.begin(), termini.end());
```

Algoritmi per l'ordinamento

- Algoritmo **merge()** fonde due contenitori in un unico contenitore ordinato.

```
vector<string> v1, v2, v3;
...
sort(v1.begin(), v1.end());
sort(v2.begin(), v2.end());
merge(v1.begin(), v1.end(), v2.begin(),
v2.end(), v3.begin());
```

Algoritmi per l'ordinamento

- Algoritmi **max_element()** e **min_element()** restiuiscono gli iteratori agli elementi con valore massimo e valore minimo del contenitore

```
vector <int> v;
vector <int>::iterator i;

i=max_element (v.begin(), v.end());
cout<< ''Massimo:  ''<<*i<<endl;
i=min_element (v.begin(), v.end());
cout<< ''Minimo:  ''<<*i<<endl;
```