Università della Calabria

Corso di Laurea in Ingegneria Elettronica

Programmazione Orientata agli Oggetti

A.A. 2006/2007

Standard Template Library

Paolo Trunfio *

* DEIS, Università della Calabria – http://si.deis.unical.it/~trunfio

STL

La *Standard Template Library* (*STL*) comprende tre tipi di componenti:

- **Contenitori**
- **Iteratori**
- **Algoritmi**

I *contenitori* sono strutture dati come *vettori*, *liste* e *mappe*, implementati con il meccanismo dei template, in grado di memorizzare oggetti di qualsiasi tipo.

Gli *iteratori* sono meccanismi utilizzati per accedere e manipolare gli oggetti memorizzati nei contenitori.

Gli *algoritmi* comprendono funzioni d'utilità per l'*ordinamento*, la *ricerca*, ed altre operazioni standard sui contenitori.

Esempio di contenitore: la classe vector

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

int main() {
  vector<int> a;  // crea un vector di interi a
  for (int i = 1; i <= 10; i++) {
    a.push_back(i);  // appende il valore i in fondo ad a
  }
  for (int i = 0; i < a.size(); i++) {
    cout << a[i] << " ";
  }
  cout << endl;
  system("pause");
}

// 1 2 3 4 5 6 7 8 9 10
```


La classe vector (1)

Dichiarazioni (esempi):

```
vector<int> v;  // crea un vettore v di interi, inizialmente vuoto
vector<double> v(10);  // vettore di double con dimensione iniziale 10
vector<T> v;  // vettore di oggetti della classe T
```

Metodi principali:

```
size_type size() const; // restituisce il numero di elementi presenti nel vettore
bool empty() const; // restituisce true se il vettore è vuoto
void push_back(const T& x); // appende x alla fine del vettore
void pop_back(); // rimuove l'ultimo elemento del vettore
void clear(); // rimuove tutti gli elementi nel vettore
T& at(size_type i); // restituisce un riferimento all'elemento in posizione i
```

Operatori:

I *contenitori* possono essere confrontati utilizzando gli operatori **==**, **!=**, **<=**, **>=**, **<**, **>,** e possono essere assegnati utilizzando l'operatore **=**. L'operatore **[]** può essere usato nei *vector* per accedere ai singoli elementi (in alternativa ad **at)**.

La classe vector (2)

Iteratori:

vector<int>::iterator it; // definisce un iteratore *it* ad un vettore di interi

it++ // avanza di un elemento

it-- // retrocede di un elemento

***it** // restituisce l'oggetto puntato da *it*

Metodi di vector che restituiscono o fanno uso di iteratori:

iterator begin(); // restituisce un iteratore al primo elemento del vettore

iterator end(); // restituisce un iteratore che punta *subito dopo* l'ultimo elemento del vettore

iterator insert(iterator it, const T& x); // inserisce *x* nella posizione precedente a quella puntata dall'iteratore *it*; restituisce un iteratore all'elemento inserito

iterator erase(iterator it); // cancella l'elemento nella posizione dell'iteratore; restituisce un iteratore all'elemento successivo a quello cancellato

Scansione degli elementi mediante iteratori

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

int main() {
  vector<int> a;
  for (int i = 1; i <= 10; i++) {
    a.push_back(i);
  }
  vector<int>::iterator iter;
  for (iter = a.begin(); iter != a.end(); iter++) {
    cout << *iter << " ";
  }
  cout << endl;
  system("pause");
}

// 1 2 3 4 5 6 7 8 9 10
```

Iteratori costanti

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

void s1(vector<int> &v) {
  vector<int>::iterator it = v.begin();
  while (it != v.end()) {
    cout << *it << " ";
    it++;
  }
  cout << endl;
}
```

```
void s2(const vector<int> &v) {
  vector<int>::const_iterator it =
    v.begin();
  while (it != v.end()) {
    cout << *it << " ";
    it++;
  }
  cout << endl;
}

int main() {
  vector<int> a;
  for (int i = 1; i <= 10; i++)
    a.push_back(i);
  s1(a);
  s2(a);
  system("pause");
}
```


Vettori di oggetti

```
class Esame {
  public:
    string nome;
    int voto;
    int crediti;
    Esame(string n, int v, int c) {
      nome = n; voto = v; crediti = c;
    }
};

double media(const
                vector<Esame> &v) {
  double n = 0;
  int d = 0;
  vector<Esame>::const_iterator it;
  it = v.begin();
  while (it != v.end()) {
    int voto = (*it).voto;
    int crediti = (*it).crediti;
    n += voto*crediti;
    d += crediti;
    it++;
  }
  return n/d;
}

int main() {
  vector<Esame> v;
  Esame e1("Calcolo 1",21,4);
  Esame e2("Calcolo 2",25,4);
  Esame e3("Fisica 1",28,6);
  v.push_back(e1);
  v.push_back(e2);
  v.push_back(e3);
  double m = media(v);
  cout << "media = " << m << endl;
  system("pause");
}
```

La classe list (1)

Dichiarazioni (esempi):

list<int> l; // crea una lista vuota di interi

list<T> l; // lista di oggetti della classe *T*

Metodi principali (esempi):

size_type size() const; // restituisce il numero di elementi presenti nella lista

bool empty() const; // restituisce *true* se la lista è vuota

void push_back(const T& x); // appende *x* alla fine della lista

void push_front(const T& x); // inserisce *x* all'inizio della lista

void pop_back(); // rimuove l'ultimo elemento della lista

void pop_front(); // rimuove il primo elemento della lista

void clear(); // rimuove tutti gli elementi nella lista

void reverse(); // inverte la lista

void sort(); // ordina gli elementi della lista in ordine crescente usando l'operatore < associato al tipo degli elementi

void remove(const T& x); // rimuove l'elemento *x* dalla lista usando l'operatore == associato al tipo degli elementi

La classe list (2)

Iteratori:

list<int>::iterator it; // definisce un iteratore *it* ad una lista di interi

it++ // avanza di un elemento

it-- // retrocede di un elemento

***it** // restituisce l'oggetto puntato da *it*

Metodi di list che restituiscono o fanno uso di iteratori:

iterator begin(); // restituisce un iteratore al primo elemento della lista

iterator end(); // restituisce un iteratore che punta *subito dopo* l'ultimo elemento della lista

iterator insert(iterator it, const T& x); // inserisce *x* nella posizione precedente a quella puntata dall'iteratore *it*; restituisce un iteratore all'elemento inserito

iterator erase(iterator it); // cancella l'elemento nella posizione dell'iteratore; restituisce un iteratore all'elemento successivo a quello cancellato

La classe list: un esempio

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

void eliminaPari(list<int> &l) {
  list<int>::iterator it;
  it = l.begin();
  while (it != l.end()) {
    int val = *it;
    if (val % 2 == 0)
      it = l.erase(it);
    else
      it++;
  }
}
```

```
void stampa(const list<int> &l) {
  list<int>::const_iterator it;
  for(it = l.begin(); it != l.end(); it++)
    cout << *it << " ";
  cout << endl;
}

int main() {
  list<int> l;
  for (int i = 1; i <= 10; i++)
    l.push_back(i);
  eliminaPari(l);
  stampa(l);
  system("pause");
}

// 1 3 5 7 9
```


La classe map (1)

Dichiarazioni (esempi):

map<string,int> m; // crea una mappa indicizzata da chiavi di tipo string e contenente valori di tipo int

map<int,T> m; // mappa con chiavi di tipo int e valori di tipo *T*

Metodi principali (esempi):

m.size() // restituisce il numero di elementi nella mappa *m*

m.empty() // restituisce *true* se la mappa *m* è vuota

m.insert(valType("POO", 27)) // inserisce nella mappa il valore 27 con chiave "POO"; l'inserimento avviene soltanto se nella mappa non esiste già un'istanza con la stessa chiave

m.find("POO") // restituisce un iteratore all'istanza nella mappa la cui chiave è "POO"; se l'istanza non è presente restituisce un iteratore uguale a end()

m.count("POO") // restituisce il numero di istanze nella mappa la cui chiave è "POO"; il valore restituito può essere 0 o 1

La classe map (2)

Il tipo utilizzato per le operazioni di inserimento è definito tramite un typedef:

typedef map<string,int>::value_type valType;

Iteratori:

map<string,int>::iterator it; // un iteratore ad una mappa <string,int>

it++ // avanza di un elemento

it-- // retrocede di un elemento

(*it).first // restituisce la chiave dell'oggetto puntato da *it*

(*it).second // restituisce il valore dell'oggetto puntato da *it*

Metodi di map che restituiscono o fanno uso di iteratori:

iterator begin(); // restituisce un iteratore al primo elemento della mappa

iterator end(); // restituisce un iteratore che punta *subito dopo* l'ultimo elemento della mappa

iterator erase(iterator it); // cancella l'elemento nella posizione dell'iteratore; restituisce un iteratore all'elemento successivo a quello cancellato

La classe map: un esempio

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

int main() {
  map<string,string> m;
  typedef map<string,string>::value_type val;
  m.insert(val("Matteo", "0984-1234"));
  m.insert(val("Francesca", "0963-2468"));
  m.insert(val("Giovanni", "+393474567"));
  map<string,string>::iterator it;
  for(it = m.begin(); it != m.end(); it++)
    cout << it->first << ": " << it->second << endl;
  system("pause");
}
```

inserimento

Francesca: 0963-2468
Giovanni: +393474567
Matteo: 0984-1234

Uso dell'operatore [] per inserire valori in una mappa

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

int main() {
  map<string,string> m;
  m["Matteo"] = "0984-1234";
  m["Francesca"] = "0963-2468";
  m["Giovanni"] = "+393474567";
  map<string,string>::iterator it;
  for(it = m.begin(); it != m.end(); it++)
    cout << it->first << ": " << it->second << endl;
  system("pause");
}
```

inserimento

```
Francesca: 0963-2468
Giovanni: +393474567
Matteo: 0984-1234
```


Algoritmi (1)

Direttiva al preprocessore:

#include <algorithm>

Metodi principali:

iterator find(iterator first, iterator last, const T& value);

Gli elementi nell'intervallo [first,last) sono confrontati con *value* usando l'operatore == associato al tipo T. Se si trova una corrispondenza, la ricerca finisce e restituisce un iteratore all'elemento. Se non trova alcuna corrispondenza viene restituito last.

void replace(iterator first, iterator last, const T& *oldVal,* const T& *newVal*)

Sostituisce tutte le istanze di *oldVal* con *newVal* nell'intervallo specificato dalla coppia di iteratori [first,last)

void sort(iterator first, iterator last)

Riordina gli elementi nell'intervallo [first,last) in ordine crescente usando l'operatore < associato al tipo degli elementi. Nota: la classe *list* definisce un metodo specializzato per l'operazione di ordinamento. Quando si lavora con oggetti list è quindi preferibile utilizzare il metodo sort della classe list anziché quello generico.

Algoritmi (2)

iterator set_difference(iterator first1, iterator last1, iterator first2, iterator last2, iterator *result*)

Costruisce la sequenza ordinata degli elementi trovati nella prima sequenza (individuata da [first1,last1)) ma non contenuti nella seconda (individuata da [first2,last2)). L'iteratore *result* specifica la posizione del contenitore in cui deve essere inserito il risultato (tale contenitore deve essere di dimensione sufficiente). L'iteratore restituito punta a una posizione oltre l'ultimo elemento posto nel contenitore individuato da *result*. E' necessario che le due sequenze in input siano ordinate in base all'operatore < associato al tipo degli elementi.

iterator set_intersection(iterator first1, iterator last1, iterator first2, iterator last2, iterator *result*)

Come il precedente, ma calcola l'intersezione delle due sequenze.

iterator set_union(iterator first1, iterator last1, iterator first2, iterator last2, iterator *result*)

Come il precedente, ma calcola l'unione delle due sequenze.

Esempio: uso della funzione find

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

int main() {
  list<int> lista;
  for (int i = 1; i <= 10; i++)
    lista.push_back(i);
  int x; cout << "Numero da cercare: "; cin >> x;
  list<int>::iterator it = find(lista.begin(), lista.end(), x);
  if (it != lista.end())
    cout << "Numero presente\n";
  else
    cout << "Numero non presente\n";
  system("pause");
}
```