Linux@evening

seminari di informatica libera

solira

C++ avanzato: template e librerie STL

Nunzio Brugaletta

Di cosa si parla: programmazione generica

- Stile di programmazione in cui gli algoritmi sono scritti in termini di **to-be-specified-later types**.
- Si passa come parametro non un valore ma un **tipo**.
- Basato sui **template** e sulle librerie **STL** (*Standard Template Library*).
- STL originariamente sviluppate negli HP Research Labs da Alexsander Stepanov assieme a David Musser e Meng Lee.
- Incluse nella libreria standard C++

Riusabilita' del codice?

```
class libro {
  public:
    void setLibro(datilib);
    void getLibro(datilib&);
    bool getDisponibile(){return presente;};
    bool prestitoOk();
    bool restituitoOk();
  private:
    datilib libbib;
    bool presente;
};
...
```

```
...
class libreria{
  public:
    int aggiungi(libro);
    bool estrai(int, libro&);
    bool aggiorna(int, libro);
    int dotazione(){return bib.size();};
  private:
    vector<libro> bib;
};
```

```
class libSocio : public libro {
  public:
    libSocio();
    void getSocio(datisoc& sget){sget = socbib;};
    bool prestitoOk(datisoc);
    bool restituitoOk();
  private:
    void setSocio(datisoc s1){socbib = s1;};
    datisoc socbib;
};
```

- Per libreria come contenitore di oggetti di tipo libSocio?
- Ricopiare classe precedente sostituendo ricorrenze di libro con libSocio?
- E se ci sono pure riviste?
- Stesse funzionalità ma oggetti diversi

Classi template

```
template <class tag>
class libreria{
  public:
    int aggiungi(tag);
    bool estrai(int, tag&);
    bool aggiorna(int, tag);
    int dotazione(){return bib.size();};
  private:
    vector<tag> bib;
};

// Aggiunge nella libreria un elemento generico

template <class tag>
int libreria<tag>::aggiungi(tag elem){
    bib.push_back(elem);
    return dotazione()-1;
};
...
// libreria contenente oggetti tipo libro

libreria<libro> lib1;

// libreria contenente oggetti tipo libSocio

libreria<libSocio> lib2;
```

Fra parentesi angolari si specifica un **segnaposto** sostituito da un oggetto specifico quando si instanzia la classe

Nei metodi le istruzioni fanno riferimento al segnaposto

Quando si definiscono le istanze della classe si specifica anche il tipo (**template specialization**)

Funzioni template (1)

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

class libro{
public:
  void setLibro(string,string,float);
  friend ostream& operator<<(ostream& output, const libro& l);
  bool operator<(const libro &l) const;
private:
  string titolo;
  string autore;
  float prezzo;
};

void libro::setLibro(string t,string a,float p){
  this->titolo=t;
  this->autore=a;
  this->prezzo=p;
};

// significato operatore << per libro

ostream& operator<<(ostream& output, const libro& l){
  output << l.titolo << ' ' << l.autore << ' ' << l.prezzo << endl;
  return output;
};

// significato operatore < per libro

bool libro::operator<(const libro &l) const{
    return (this->prezzo<l.prezzo);
};
```

Classe libro generico con solo metodo per generazione oggetto

Si fa riferimento agli attributi dell'oggetto stesso

Specializzazione degli operatori

Funzioni template (2)

Calcolo del più grande fra due dati qualsiasi a condizione che il tipo dei dati implementi l'overloading dell'operatore.

Utilizza un segnaposto per il tipo

```
// calcolo maggiore dati di qualsiasi tipo

template <typename T>
T PiuGrande(T x, T y){
    return x < y ? y : x;
};
```

La funzione viene specializzata per il tipo libro.
Si comporta come fosse:

```
libro  PiuGrande(libro x, libro y)
```

```
main(){
  libro l1,l2,l3;
  string titolo,autore;
  float prezzo;

  // stampa dati libro piu' costoso

  cout << "Titolo1 ";
  getline(cin,titolo);
  cout << "Autore1 ";
  getline(cin,autore);
  cout << "Prezzo1 ";
  cin >> prezzo;
  cin.ignore();

  l1.setLibro(titolo,autore,prezzo);

  cout << "Titolo2 ";
  getline(cin,titolo);
  cout << "Autore2 ";
  getline(cin,autore);
  cout << "Prezzo2 ";
  cin >> prezzo;
  cin.ignore();

  l2.setLibro(titolo,autore,prezzo);

  l3 = PiuGrande(l1,l2);
  cout << l3;
}
```

STL in sintesi

Parte fondamentale della libreria standard basata su classi e funzioni template

Componenti principali:

- **Containers**: oggetti che contengono altri oggetti. Le comuni strutture dati (vettori, liste, code, stack ...)
- **Iterators**: entità che consentono di attraversare un container qualunque esso sia e operare con gli elementi. Intermediari tra containers e algoritmi
- **Algoritmi**: i più comuni algoritmi (find, sort, count ...) che, per mezzo degli iteratori si applicano a qualsiasi container.

Interfaccia di alto livello verso le strutture dati. Il compilatore si occupa dell'efficienza dell'implementazione

Containers

- 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;
```

Metodi comuni

```
push_front   // non disponibile per vector
             // inserisce elemento in testa
pop_front    // non disponibile per vector
             // estrae elemento dalla testa
push_back    // inserisce in coda

pop_back     // estrae elemento dalla coda

size         // quantità elementi nel container

resize       // modifica dimensione

insert       // inserisce elemento

erase        // elimina elemento

clear        // elimina tutti gli elementi

front        // riferimento al primo elemento

back         // riferimento all'ultimo elemento

at           // solo per vector e deque
             // accesso all'elemento
```

esempio uso mappa

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

class alunno{
public:
  string getMatricola(){return matr;};
  string getNome(){return nome;};
  void putNuovo(string mt, string nm)
               {matr=mt;nome=nm;};
private:
  string matr;
  string nome;
};

main(){
  map<string,alunno> elenco;
  alunno temp;
  string m,n;

  // inserisce alunni in elenco

  while(true){
    cout << "Matricola alunno ";
    getline(cin,m);

    if(m=="")
      break;

    cout << "Nome alunno ";
    getline(cin,n);

    temp.putNuovo(m,n);
    elenco[temp.getMatricola()]=temp;
  };
```

```
  // stampa nome conoscendo matricola

  cout << "Quale matricola? ";
  getline(cin,m);
  n = elenco[m].getNome();

  cout << "Nome: " << n << endl;
}
```

- È possibile inserire solo un elemento con quella chiave.
- Le registrazioni con chiavi duplicate non vengono prese in considerazione.
- È possibile accedere velocemente all'elemento specificando la chiave

Associazione fra chiave e valore

Iteratori

```
// tipi di iteratori

iterator                // per scorrere gli elementi
                        // dal primo all'ultimo
reverse_iterator        // per scorrere gli elementi
                        // in ordine inverso
const_iterator          // in avanti ma non consente
                        // la modifica dell'elemento
const_reverse_iterator  // come precedente ma
                        // all'indietro
```

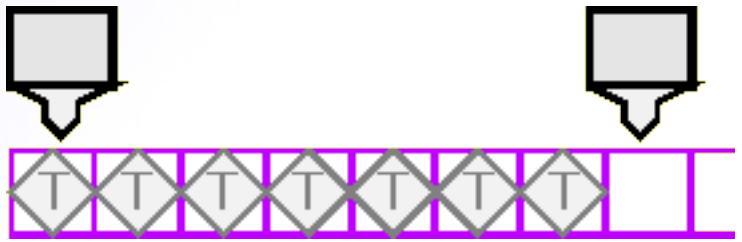
- Metodo per accedere agli elementi di un container.
- Ogni container fornisce iteratori specifici per la propria implementazione.
- Il programmatore gestisce 4 tipi sempre uguali di iteratori. La classe adatta alla propria specificità (polimorfismo)

Iteratori: esempio di uso

```
// stampa gli elementi di una lista

void stampa(list<libro> l){
  list<libro>::const_iterator i;

  for(i=l.begin();i != l.end(); i++){

    // stampa oggetto se overloading operatore <<

    cout << *i;

    // richiama metodo

    cout << i->getNome();
  };

};
```

Iteratore di un container generico

Accesso agli elementi o ai metodi degli elementi del container



`begin()` punta al primo elemento del container.

`end()` punta alla posizione successiva all'ultimo elemento del container

Algoritmi

- Parte delle STL che implementa algoritmi frequentemente utilizzati: find, sort, count ...
- Scelta progettuale: invece di utilizzare ereditarietà e polimorfismo dei container (algoritmi come comportamenti della classe container) si separano gli algoritmi dai container.
- Algoritmi indipendenti dai dettagli di implementazione dei container su cui si applicano.
- Per mezzo degli iteratori l'algoritmo si applica a tutti i container che soddisfano i requisiti.
- Richiedono inclusione header: `#include <algorithm>`

```
// operazioni che non modificano elementi

for_each            // applica operazione ad
                    // ogni elemento del container
find e find_if      // trova elemento uguale a
                    // valore o che soddisfa cond.
count e cont_if     // conta per valore o per
                    // condizione
max_element         // trova valore max sequenza

min_element         // trova valore min sequenza
```

```
// operazioni che modificano elementi

copy                // copia sequenza

remove_copy_if      // estrazione
                    // sottoinsieme
// sort

sort                // ordina sequenza
```

Algoritmi: classe esempio

```
class clibro{
public:
  string getTitolo(){return titolo;};
  string getAutore(){return autore;};
  float  getPrezzo(){return prezzo;};
  void   setPrezzo(float p){prezzo = p;};
  friend ostream& operator<<(ostream& output, const clibro& l);
  bool   operator==(const string &t) const;
private:
  string titolo;
  string autore;
  float  prezzo;
};

ostream& operator<<(ostream& output, const clibro& l){
  output << l.titolo << ' ' << l.autore << ' '
      << l.prezzo << endl;
  return output;
};

bool clibro::operator==(const string &t) const{
  return (this->titolo==t);
};

// vettore ma potrebbe essere altro container

vector<clibro> vLibri;
```

Definizione classe e metodi utilizzati negli esempi.

Ridefinizione di operatori per la classe

Algoritmi: iteratori e sort

```
// stampa elementi del container

void stampa(const vector<clibro> vl){
  vector<clibro>::const_iterator i;

  for(i=vl.begin();i!=vl.end();i++)
    cout << *i;
};
```

Stampa in accordo all'overloading dell'operatore <<

Uso iteratore per scansione lineare container

```
bool ordT(clibro l1,clibro l2){return l1.getTitolo()<l2.getTitolo();};
bool ordP(clibro l1,clibro l2){return l1.getPrezzo()<l2.getPrezzo();};

void ordina(vector<clibro>& vl){
  int scelta;

  cout << "Ordina per titolo(1) o prezzo (2)" << endl;
  cin >> scelta;

  if(scelta==1)
    sort(vl.begin(),vl.end(),ordT);
  if(scelta==2)
    sort(vl.begin(),vl.end(),ordP);

};
```

Tipo di confronto: funzione con 2 parametri tipo contenuto nel container e bool come tipo ritornato

Range elementi da ordinare e tipo di ordinamento. Senza funzione ordine basato su operatore <

Find e Find_if

```
void cerca1(vector<clibro> vl){
  string t;
  vector<clibro>::const_iterator pos;

  cout << "Titolo da cercare ";
  getline(cin,t);

  pos=find(vl.begin(),vl.end(),t);

  if(pos != vl.end())
    cout << *pos;
  else
    cout << "Libro inesistente :-(" << endl;
};
```

Cerca un libro per titolo.
Si basa su overloading dell'operatore ==

```
class cercaTit{
private:
  string tit;
public:
  cercaTit (string n): tit(n) {};

  bool operator() (clibro value) const{
    return (value.getTitolo()==tit);
  }
};

void cerca2(vector<clibro> vl){
  ...
  pos=find_if(vl.begin(),vl.end(),cercaTit(t));
  ...
};
```

Funzione oggetto

Costruttore inizializza variabile

Anche () è un operatore!! (overloading)
funzione richiamata quando istanziata

Cerca utilizzando una funzione che deve ridefinire l'operatore ()

For_each

```
class aumentaPercent{
private:
  int pc;
public:
  aumentaPercent (int n): pc(n) {};

  void operator() (clibro& value){
    float nuovo;
    nuovo = (float) value.getPrezzo()+value.getPrezzo()*pc/100;
    value.setPrezzo(nuovo);
  }
};

void aumenta(vector<clibro>& vl){
  int p;

  cout << "Percentuale aumento prezzo ";
  cin >> p;

  for_each(vl.begin(),vl.end(),aumentaPercent(p));
};
```

Aumenta i prezzi dei libri di una data percentuale

A tutti gli elementi del range specificato si applica la funzione

count_if e max_element

```
class prezzoRif{
private:
  float pz;
public:
  prezzoRif (float n): pz(n) {};

  bool operator() (clibro value){
    return value.getPrezzo()>pz;
  }
};

void conta(vector<clibro> vl){
  float pz;
  int q;

  cout << "Quanti libri con prezzo maggiore di? ";
  cin >> pz;
  cin.ignore();

  q = count_if(vl.begin(),vl.end(),prezzoRif(pz));

  cout << "Ci sono " << q << " libri con prezzo superiore" << endl;
};
```

Predicato unario

un valore passato come parametro

bool come tipo ritornato

```
bool ordP(clibro l1,clibro l2){return l1.getPrezzo()<l2.getPrezzo();};

void prezzoMag(vector<clibro> vl){
  vector<clibro>::const_iterator pos;

  cout << "Libro con prezzo maggiore" << endl;

  pos = max_element(vl.begin(),vl.end(),ordP);

  cout << *pos;
};
```

Stampa il libro con il prezzo più alto.

Utilizza la funzione di ordinamento per prezzo definita in precedenza

copy e remove_copy_if

```
#include <iterator>

void stampaOstr(vector<clibro> vl){
  ostream_iterator<clibro> video (cout);
  cout << "Stampa libri" << endl;

  copy(vl.begin(),vl.end(), video);

};
```

```
class nomeAutore{
private:
  string na;
public:
  nomeAutore (string n): na(n) {};

  bool operator() (clibro value){
    return value.getAutore()!=na;
  }
};

void sottoins(vector<clibro> vl){
  string a;
  vector<clibro> libriAutore;

  cout << "Stampa i libri di un autore" << endl;
  cout << "Nome autore ";
  getline(cin,a);

  libriAutore.clear();

  remove_copy_if(vl.begin(),vl.end(),
           inserter(libriAutore,libriAutore.begin()),
           nomeAutore(a));

  stampaOstr(libriAutore);
};
```

Si possono stampare gli elementi di un container copiando con utilizzo di particolare iteratore

Estrazione di un sottoinsieme. `remove_copy_if` richiede:

- Range da esaminare.
- Tipo di operazione. Nell'esempio inserimento in nuovo vettore dalla posizione iniziale.
- Funzione da applicare agli elementi. Vengono ricopiati elementi che **non** soddisfano la condizione desiderata

e non e' finita qui ...

http://www.cplusplus.com
reference sul linguaggio con esempi di uso

http://www.sgi.com/tech/stl/index.html
STL programmer's guide. Sito ufficiale del team di sviluppo delle librerie

http://www.mochima.com/tutorials/STL.html
rapida ma ottima introduzione alla STL. Non tutto ma di tutto

...