Realizzazione di Liste (II)

Laboratorio di Algoritmi e Strutture Dati

Domenico Redavid
redavid@di.uniba.it

Materiale di base gentilmente concesso dal dott. Nicola Di Mauro Ricercatore presso l'Univ. di Bari

Obiettivi

- Realizzazione di LISTA di interi mediante vettori → OK
- Realizzazione di LISTA di elementi di tipo più complesso (libro) mediante vettori → OK
- **Realizzazione di LISTA di elementi di tipo non specificato mediante vettori e uso di template**
- Realizzazione di LISTA circolare mediante puntatori
- Applicazione alla risoluzione di esercizi

Evoluzione

- Modificare la realizzazione introducendo i Template
  - Aiuto: Dove è “nascosta” la definizione del tipo degli elementi nella lista? Che impatto ha la variazione del tipo?

I template di classe

E' possibile comprendere che cosa sia una lista indipendentemente dal tipo di elementi che contiene.

Se si vuole istanziare una lista dobbiamo specificare necessariamente il tipo di dato contenuto.

I template di classe in C++ sono uno strumento per descrivere il concetto lista a un livello generale, tale da consentire di istanziare versioni specifiche.

I template di classe sono detti *tipi parametrici* perché hanno bisogno di uno o più parametri per generare l'istanza del template di classe desiderato.

Per generare una collezione di classi basta scrivere la definizione di un solo template di classe e, ogni volta che si ha bisogno di una nuova istanza specifica, è il compilatore che la genererà.

Un template di classe Lista

Un template di classe **Lista**, per esempio, può essere la base per molte classi **Lista** come “**Lista** di **double**”, “**Lista** di **int**”, “**Lista** di **Libri**”, e così via.

```
template< class T >
class Lista {
   public:
      ...
   private:
      ...
   ...
}
```

Definizione del template di classe **Lista**

- template< class T > indica che si tratta di un template
- il parametro di tipo **T** indica il tipo di classe **Lista** da creare
- il tipo di elemento da memorizzare in Lista è menzionato genericamente come **T** in tutta l'intestazione della classe **Lista** e nelle definizioni delle funzioni membro

Template
Sintassi

Definizione della classe template

```
template <typename variabile_tipo>
  class Nome_classe {
      caratteristiche
  }
```

e della funzione membro template

```
template <typename variabile_tipo>
tipo_restituito Nome_classe <variabile_tipo>::
     nome_funzione(parametri) const_opt {
     istruzioni
}
```

Template di classe Lista
interfaccia

```
#ifndef _LISTAVTEMP_H
#define _LISTAVTEMP_H
#include <iostream>
using namespace std;

template <class T, int max>  class Lista
{
  public:
    typedef int posizione;
    typedef T tipoelem;
    Lista();    //costruttore
    Lista(const Lista<T,max>&);    //costruttore di copia
    ~Lista();   //distruttore
    // operatori
    void crealista();
    bool listavuota() const;
    tipoelem leggilista(posizione) const;
    void scrivilista(tipoelem, posizione);
    posizione primoLista() const;
    bool finelista(posizione) const;
    posizione succLista(posizione) const;
    posizione predLista(posizione) const;
    void inslista(tipoelem, posizione);
    void cancLista(posizione);
// funzioni di servizio
    void stampaLista();

  private:
    T elementi[max];
    int lunghezza; // lunghezza della lista
  };

#endif // _LISTAVTEMP_H
```

listavtemp.h

Template di classe Lista
implementazione (1)

```
// Implementazione della classe Lista
// costruttore
template <class T, int max> Lista<T,max>::Lista()
{
  crealista();
}
// costruttore di copia
/* il costruttore di copia effettua una copia o clone di un oggetto. E' invocato, ad esempio,
   quando un oggetto è passato per valore ad una funzione ovvero quando una funzione
      restituisce    un oggetto
*/
template <class T, int max> Lista<T,max>::Lista(const Lista<T,max>& L)
{
  lunghezza=L.lunghezza;
  for (int i=0; i<lunghezza; i++)
      elementi[i]=L.elementi[i];
}

// distruttore
template <class T, int max> Lista<T,max>::~Lista()
{}

// operatori
template <class T, int max> void Lista<T,max>::crealista()
{
  lunghezza = 0;
}

template <class T, int max> bool Lista<T,max>::listavuota() const
  {
    return(lunghezza == 0);
  }
```

listavtemp.h

Template di classe Lista
implementazione (2)

```
template <class T, int max> int Lista<T,max>::primoLista() const
  {
    return(1);
  }
template <class T, int max> int Lista<T,max>::succLista(posizione p) const
  {
    if ((p>0) && (p<lunghezza+1)) // precondizione
      return(p+1);
    else
      return(p);
  }
template <class T, int max> int Lista<T,max>::predLista(posizione p) const
  {
    if ((p>0) && (p<lunghezza+1)) // precondizione
      return(p-1);
    else
      return(p);
  }
template <class T, int max> bool Lista<T,max>::finelista(posizione p) const
  {
    if ((p>0) && (p<=lunghezza+1)) // precondizione
      return( p == lunghezza+1);
    else
      return(false);
  }
template <class T, int max> T Lista<T,max>::leggilista(posizione p) const
  {
    if ((p>0) && (p<lunghezza+1)) // precondizione
      return(elementi[p-1]);
  }
```

listavtemp.h

Template di classe Lista
implementazione (3)

```
template <class T, int max> void Lista<T,max>::scrivilista(tipoelem a, posizione p)
{
  if ((p>0) && (p<lunghezza+1)) // precondizione
    elementi[p-1]=a;
}

template <class T, int max> void Lista<T,max>::inslista(tipoelem a, posizione p)
{
  if ((p>0) && (p<=lunghezza+1)) // precondizione
    {
      for (int i=lunghezza; i>=p; i--)
        elementi[i] = elementi[i-1];
      elementi[p-1]=a;
      lunghezza++;
    }
}

template <class T, int max> void Lista<T,max>::cancLista(posizione p)
{
  if ((p>0) && (p<lunghezza+1)) // precondizione
    if (!listavuota())
      {
        for (int i=p-1;i<(lunghezza-1);i++)
          elementi[i]=elementi[i+1];
        lunghezza--;
      }
}
```

listavtemp.h

Template di classe Lista
Test

```
#include <iostream>
#include "listavTemp.h"
#include "libro.h"
using namespace std;
int main() {
  Lista<Libro,10> l;
  Libro libro;
  Lista<Libro,10>::posizione indiceElemento = l.primoLista();
  libro.setTitolo("LIBRO1");
  l.inslista(libro,indiceElemento=l.succLista(indiceElemento));
  libro.setTitolo("LIBRO2");
  l.inslista(libro,indiceElemento=l.succLista(indiceElemento));
  libro.setTitolo("LIBRO2");
  l.inslista(libro,indiceElemento=l.succLista(indiceElemento));
  libro.setTitolo("LIBRO3");
  l.inslista(libro,indiceElemento=l.succLista(indiceElemento));
  libro.setTitolo("LIBRO4");
  l.inslista(libro,indiceElemento=l.succLista(indiceElemento));
  libro.setTitolo("LIBRO5");
  l.inslista(libro,indiceElemento=l.succLista(indiceElemento));
  l.stampaLista();
  cout <<"\nInserimento nuovo elemento in seconda posizione...\n";
  libro.setTitolo("LIBRO INSERITO");
  l.inslista(libro,l.succLista(l.primoLista()));
  l.stampaLista();
  cout << "Lista di interi: " << endl;
  Lista<int,5> l2;
  Lista<int,5>::posizione indice2 = l2.primoLista();
  l2.inslista(10,indice2=l2.succLista(indice2));
  l2.inslista(1,indice2=l2.succLista(indice2));
  l2.inslista(5,indice2=l2.succLista(indice2));
  l2.inslista(5,indice2=l2.succLista(indice2));
  l2.inslista(1,indice2=l2.succLista(indice2));
  l2.stampaLista();
  return 0;}
```

testlistavtemp.cpp

Realizzazioni Proposte

- Nella realizzazione senza template, CellaLista definiva e “nascondeva” il tipo degli elementi nella lista.
- Nella realizzazione con template, il generico tipo T “sostituisce” CellaLista;
- Lista: listavTemp.h (Lista realizzata tramite template);
- ProvaListaTemplate: testlistavTemp.cpp (main per prova Lista template)

Obiettivi

- Realizzazione di LISTA di interi mediante vettori → OK
- Realizzazione di LISTA di elementi di tipo più complesso (libro) mediante vettori → OK
- Realizzazione di LISTA di elementi di tipo non specificato mediante vettori e uso di template → OK
- **Realizzazione di LISTA circolare mediante puntatori**
- Applicazione alla risoluzione di esercizi

Rappresentazione collegata circolare (con sentinella) realizzata mediante doppi puntatori (o simmetrica)

```
#ifndef _CELLALP_H
#define _CELLALP_H

template <class T> class Cella
  {
  public:
    typedef T tipoelem;
    Cella();
    Cella(tipoelem);
    void setElemento(tipoelem);
    tipoelem getElemento() const;
    void setSucc(Cella*);
    Cella* getSucc() const;
    void setPrec(Cella*);
    Cella* getPrec() const;
    bool operator ==(Cella);

  private:
    tipoelem elemento;
    Cella* prec;
    Cella* succ;
  };
```

cellalp.h

Classe Cella

parte pubblica
- setElemento e getElemento
- setSucc e setPrec
- getSucc e getPrec

parte privata
- elemento
- prec
- succ

Implementazione della classe Cella

```
// Implementazione della classe Cella

template <class T> Cella<T>::Cella() {}
template <class T> Cella<T>::Cella(tipoelem e) {elemento = e;}
template <class T> void Cella<T>::setElemento(tipoelem label)
{
  elemento = label;
}
template <class T>  T Cella<T>::getElemento() const {return elemento;}
template <class T> void Cella<T>::setSucc(Cella* p)
{
  succ=p;
}
template <class T> Cella<T>* Cella<T>::getSucc() const
{
    return succ;
}
template <class T> void Cella<T>::setPrec(Cella* p)
{
  prec=p;
}
template <class T> Cella<T>* Cella<T>::getPrec() const {return prec;}
// sovraccarico dell'operatore ==
template <class T> bool Cella<T>::operator==(Cella cella)
{
  return (getElemento == cella.getElemento);
}
#endif // _CELLALP_H
```

Cellalp.h

La classe Lista

```
#ifndef _LISTAP_H
#define _LISTAP_H
#include "cellalp.h"
#include <iostream>
using namespace std;

template<class T>
class circLista
  {
  public:
    circLista();
    ~circLista();
    // circLista(const circLista<T>&);
    /* posizione è un puntatore a cella */
    typedef Cella<T>* posizione;
    typedef T tipoelem;
    /* Prototipi degli operatori */
    void crealista();
    bool listavuota() const;
    tipoelem leggilista(posizione) const;
    void scrivilista(tipoelem, posizione);
    posizione primoLista() const;
    bool finelista(posizione) const;
    posizione succlista(posizione) const;
    posizione preclista(posizione) const;
    void inslista(tipoelem,posizione&);
    void canclista(posizione&);
    // funzioni di servizio
    void stampaLista();
  private:
    posizione lista; //la lista è un puntatore ad
      oggetto Cella
  };
```

listap.h

Classe Lista

parte pubblica
- posizione (puntatore a Cella)
- operatori

parte privata
- lista (puntatore a Cella)

Implementazione della classe Lista
Prima parte

```
template <class T> circLista<T>::circLista() {crealista();}

template <class T> circLista<T>::~circLista()
{
    while (lista->getSucc() != lista->getPrec())
    {
      Cella<T>* posizione = lista->getSucc();
      canclista(posizione);
    }
    delete lista;
}

template <class T> void circLista<T>::crealista()
{
  T ElementoNullo;
  lista = new Cella<T>;
  lista->setElemento(ElementoNullo);
  lista->setSucc(lista);
  lista->setPrec(lista);   //la sentinella punta a se stessa
}

template <class T> bool circLista<T>::listavuota() const
 { return ((lista->getSucc() == lista) &&  (lista->getPrec()==lista)); }

template <class T> Cella<T>* circLista<T>::primoLista() const
 { return lista->getSucc();}

template <class T> Cella<T>* circLista<T>::succlista(posizione p) const
{ return p->getSucc(); }
```

listap.h

Implementazione della classe Lista
Seconda parte

```
template <class T> Cella<T>* circLista<T>::preclista(posizione p) const
{ return p->getPrec(); }

template <class T>  bool circLista<T>::finelista(posizione p) const
{return (p==lista);}

template <class T> T circLista<T>::leggilista(posizione p) const
{return p->getElemento();}

template <class T> void circLista<T>::scrivilista(tipoelem a, posizione p)
{p->setElemento(a);}

template <class T> void circLista<T>::inslista(tipoelem a, posizione &p)
{Cella<T>* temp = new Cella<T>;
  temp->setElemento(a);
  temp->setPrec(p->getPrec());
  temp->setSucc(p);
  (p->getPrec())->setSucc(temp);
  p->setPrec(temp);
  p=temp; // se p era la posizione dell'elemento n-mo, adesso lo è temp
}

template <class T> void circLista<T>::canclista(posizione &p)
{
  Cella<T>* temp = new Cella<T>;
  temp=p;
  (p->getSucc())->setPrec(p->getPrec());
  (p->getPrec())->setSucc(p->getSucc());
  p=p->getSucc();
  delete(temp);
}
```

listap.cpp

Implementazione della classe Lista
Stampa

```
template <class T> void circLista<T>::stampaLista()
{
  cout<<"[";
  Cella<T>* indice=primoLista();
  while (!finelista(indice))
  {
      cout << leggilista(indice);
      if (!finelista(succlista(indice)))
        cout << ", ";
      indice = succlista(indice);

    }
  cout<<"]\n";
}

#endif // _LISTAP_H
```

listap.h

Implementazione della classe Lista
Test

```
#include <iostream>
#include "listap.h"
#include "libro.h"
using namespace std;

int main()
{
  circLista<Libro> l;
  Libro libro;
  circLista<Libro>::posizione indiceElemento = l.primoLista();
  libro.setTitolo("LIBROUNO");
  l.inslista(libro,indiceElemento=l.succlista(indiceElemento));
  libro.setTitolo("LIBRODUE");
  l.inslista(libro,indiceElemento=l.succlista(indiceElemento));
  libro.setTitolo("LIBRODUE");
  l.inslista(libro,indiceElemento=l.succlista(indiceElemento));
  libro.setTitolo("LIBROTRE");
  l.inslista(libro,indiceElemento=l.succlista(indiceElemento));
  libro.setTitolo("LIBROQUATTRO");
  l.inslista(libro,indiceElemento=l.succlista(indiceElemento));
  libro.setTitolo("LIBROCINQUE");
  l.inslista(libro,indiceElemento=l.succlista(indiceElemento));
  l.stampaLista();
  cout <<"\nInserimento nuovo elemento in seconda posizione...\n";
  libro.setTitolo("LIBROINSERITO");
  indiceElemento = l.primoLista();
  l.inslista(libro,indiceElemento=l.succlista(indiceElemento));
  l.stampaLista();
  indiceElemento = l.primoLista();
  l.canclista(indiceElemento);
  l.stampaLista();
}
```

main.cpp

Realizzazioni Proposte

- Cella: cellalp.h (realizzazione tramite template);
- Lista: listap.h (Lista circolare simmetrica realizzata tramite template e puntatori);
- Test: main.cpp

Obiettivi

- Realizzazione di LISTA di interi mediante vettori → OK
- Realizzazione di LISTA di elementi di tipo più complesso (libro) mediante vettori → OK
- Realizzazione di LISTA di elementi di tipo non specificato mediante vettori e uso di template → OK
- Realizzazione di LISTA circolare mediante puntatori → OK
- **Applicazione alla risoluzione di esercizi**

Esercizi

- Rimozione duplicati
- Ordinamento di una lista
- Ricerca di un elemento in una lista lineare ordinata
- Calcolo delle occorrenze di elementi con determinate caratteristiche (es.: numero di interi positivi in una lista di int)
-