Laboratorio di Algoritmi e Strutture Dati

Marco de Gemmis
degemmis@di.uniba.it

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

Lista in C++ senza astrazione realizzazione con puntatori

```
#include <cstdlib>
#include <iostream>
using namespace std;
/* Definizione tipo nodo */
typedef int elemento;
typedef struct elem_lista{
elemento valore;
elem_lista* succ;
} nodo;
typedef nodo* Lista; /* Definizione tipo lista di elemento */

int main(){
/* creazione della lista */
Lista testa = new nodo; // testa è un puntatore al primo nodo
testa->valore = 1; testa->succ = NULL; /* inserimento primo elemento */
/* inserimento secondo elemento */
Lista iter = testa;
iter->succ = new nodo;
iter->succ->valore = 2; iter->succ->succ = NULL;
/* inserimento terzo elemento */
iter = iter->succ;
iter->succ = new nodo;
iter->succ->valore = 3; iter->succ->succ = NULL;
/* visualizzazione elementi */
iter = testa;
cout << "stampa lista: ";
while (iter != NULL){
cout << iter->valore << " ";
iter = iter->succ;
}
/* eliminazione secondo elemento */
iter = testa->succ;
testa->succ = iter->succ;
delete iter;
iter = testa;
cout << "stampa lista: ";
while (iter != NULL){
   cout << iter->valore << " ";
   iter = iter->succ;
}
}
```

listapuntatori.cpp

Lista in C++ senza astrazione realizzazione con vettori

```
#include <cstdlib>
#include <iostream>
using namespace std;
/* Definizione tipo nodo */
typedef int elemento;
typedef struct elem_lista{
elemento valore;
elem_lista* succ;
} nodo;
typedef nodo* Lista; /* Definizione tipo lista di elemento */

int main(){
/* creazione della lista */
Lista testa = new nodo; // testa è un puntatore al primo nodo
testa->valore = 1; testa->succ = NULL; /* inserimento primo elemento */
/* inserimento secondo elemento */
Lista iter = testa;
iter->succ = new nodo;
iter->succ->valore = 2; iter->succ->succ = NULL;
/* inserimento terzo elemento */
iter = iter->succ;
iter->succ = new nodo;
iter->succ->valore = 3; iter->succ->succ = NULL;
/* visualizzazione elementi */
iter = testa;
cout << "stampa lista: ";
while (iter != NULL){
cout << iter->valore << " ";
iter = iter->succ;
}
/* eliminazione secondo elemento */
iter = testa->succ;
testa->succ = iter->succ;
delete iter;
iter = testa;
cout << "stampa lista: ";
while (iter != NULL){
   cout << iter->valore << " ";
   iter = iter->succ;
}
}
```

listapuntatori.cpp

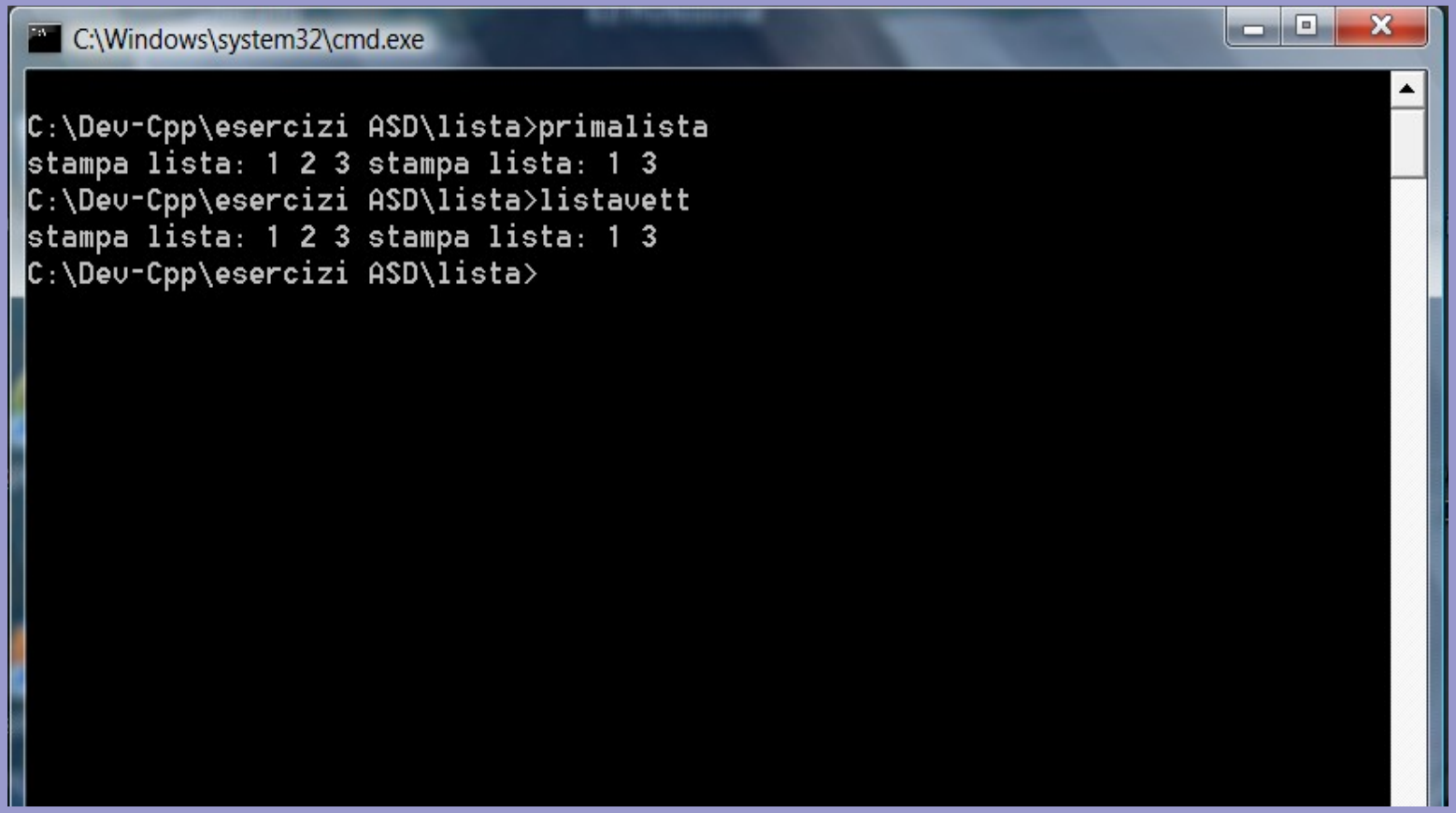
```
#include <cstdlib>
#include <iostream>
using namespace std;

/* Definizione tipo lista di interi */
const int DIMMAX = 50;
typedef struct _lista {
int elementi [DIMMAX];
int nelem;
} Lista;

int main() {
int i;
Lista L;   /* creazione della lista */
L.nelem = 0;
/* inserimento primo elemento */
L.elementi[L.nelem] = 1;
L.nelem++;
/* inserimento secondo elemento */
L.elementi[L.nelem] = 2;
L.nelem++;
/* inserimento terzo elemento */
L.elementi[L.nelem] = 3;
L.nelem++;
/* visualizzazione elementi */
i = 0;
cout << "stampa lista: ";
while (i < L.nelem){
cout << L.elementi[i] << " ";
i++;
}
/* eleminazione secondo elemento */
L.elementi[1] = L.elementi[2];
L.nelem--;
i = 0;
cout << "stampa lista: ";
while (i < L.nelem){
cout << L.elementi[i] << " ";
i++;
}
}
```

listavett.cpp

Output dei due programmi



Applicare l'astrazione

- Realizzazione lista mediante puntatori
- Applicare l'astrazione richiede:
  - Verifica della specifica sintattica
  - Verifica della specifica semantica

Lista in C++ con astrazione realizzazione mediante puntatori

```
#ifndef _LISTAP_H
#define _LISTAP_H


typedef int tipoelem;

typedef struct _cella {
    tipoelem elemento;
    struct _cella* succ;
    struct _cella* prec;
} cella;

typedef cella* posizione;
typedef cella* Lista;

void crealista(Lista);
bool listavuota(Lista);
tipoelem leggilista(posizione); // warning! non rispetta la specifica sintattica
void scrivilista(tipoelem, posizione); // warning! non rispetta la specifica sintattica
posizione primoLista(Lista);
bool finelista(posizione, Lista);
posizione succlista(posizione); // warning! non rispetta la specifica sintattica
posizione preclista(posizione); // warning! non rispetta la specifica sintattica
void inslista(tipoelem,posizione); // warning! non rispetta la specifica sintattica
void canclista(posizione); // warning! non rispetta la specifica sintattica

#endif // _LISTAP_H
```

listap.h

Lista in C++ con astrazione violazioni / 1

```
#include "listap.h"

void crealista(Lista L)
{
  tipoelem ElementoNullo;
  L = new cella;
  L->succ = L;   L->prec = L;  //la sentinella punta a se stessa
}
bool listavuota(Lista L) { return ((L->succ == L) &&  (L->prec == L));}
posizione primoLista(Lista L) {return L->succ;}
posizione succlista(posizione p) {return p->succ;}
posizione preclista(posizione p) {return p->prec;}
bool finelista(posizione p, Lista L) {return (p==L);}
tipoelem leggilista(posizione p) {return p->elemento;}
void scrivilista(tipoelem a, posizione p) {p->elemento = a}
void inslista(tipoelem a, posizione p)
{  posizione temp = new cella;
  temp->elemento = a;
  temp->prec = p->prec;
  temp->succ = p;
  p->prec->succ = temp;
  p->prec = temp;
  p=temp;
}

void canclista(posizione p)
{ posizione temp;
  temp=p;
  p->succ->prec = p->prec;
  p->prec->succ = p->succ;
  p=p->succ;
  delete temp;
}
```

SPECIFICA SINTATTICA NON RISPETTATA

```
// esempio di specifica rispettata con
// realizzazione mediante vettore

tipoelem leggilista(posizione p, Lista L)
{return L.elementi[p];}
```

listap.cpp

Lista in C++ con astrazione violazioni / 2

```
#include "listap.h"

int main()
{
  Lista l;
  crealista(l);
  posizione indiceElemento = primoLista(l);
  tipoelem a;
  a = 14;
  inslista(a,indiceElemento);
  a = 4;
  indiceElemento = succlista(indiceElemento);
  inslista(a,indiceElemento);
  a = 2007;
  indiceElemento = succlista(indiceElemento);
  inslista(a,indiceElemento);
```

main1.cpp

```
#include "listap.h"

int main()
{
  Lista l;
  posizione p;
  crealista(l);
  crealista(p); // violazione: crealista invocato
      passando una posizione
  posizione indiceElemento = primoLista(l);
  tipoelem a;
  a = 14;
  inslista(a,indiceElemento);
  l->elemento=2; // violazione: accesso diretto
```

main2.cpp

Obiettivi

- Realizzazione di LISTA di interi mediante vettori
- Realizzazione di LISTA di elementi di tipo più complesso (libro) mediante vettori
- 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

Realizzazione sequenziale con vettore

```
#ifndef _LISTAV_H
#define _LISTAV_H

// lunghezza massima della lista
const int DIMENSIONE = 1024;

// classe Lista
class Lista{
    public:
      typedef int tipoelem;
      typedef int posizione;
      Lista();      // costruttore
      ~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);
   private:
      tipoelem elementi[DIMENSIONE];
      int lunghezza;
};
#endif // _LISTAV_H
```

listav.h (v0)

Interfaccia:
- direttive al preprocessore
- dimensione massima della lista
- classe Lista
  - parte pubblica
    - tipoelem
    - posizione
    - costruttore e distruttore
    - operatori
  - parte privata
    - vettore elementi
    - lunghezza

Come generalizzare un elemento (cella) di una lista?

La classe Cella
Tipo elemento intero

```
#ifndef _CELLALV_H
#define _CELLALV_H

typedef int tipoelem;

// classe CellaLista
class CellaLista{
   public:
      CellaLista();     //costruttore
      CellaLista(tipoelem);
      ~CellaLista(){};  //distruttore
      void scriviCella(tipoelem);
      tipoelem leggiCella() const;
      bool operator == (CellaLista);
    private:
      tipoelem etichetta;
};

#endif // _CELLALV_H
```

cellalv.h (v0)

Definizione della classe CellaLista
Interfaccia
- classe CellaLista
  - parte pubblica
    - tipoelem
    - costruttori e distruttore
    - scriviCella e leggiCella
    - sovraccarico dell'operatore ==
  - parte privata
    - etichetta

Come generalizzare il dato tipoelem?

La classe Libro

```
#ifndef _LIBRO_H
#define _LIBRO_H

#include <string>
#include <iostream>

using namespace std;

class Libro{
   public:
      Libro();
      Libro(string);
      void setTitolo(string);
      string getTitolo() const;
      bool operator ==(Libro);
   private:
      string titolo;
};

#endif // _LIBRO_H
```

libro.h

- classe libro
  - costruttori
  - metodi per scrittura e lettura
  - sovraccarico dell'operatore ==
  - titolo

```
#include "libro.h"

Libro::Libro(){  titolo = "";  }

Libro::Libro(string t){ setTitolo(t); }

void Libro::setTitolo(string t){
  titolo = t;
}

string Libro::getTitolo() const{
  return (titolo);
}

// sovraccarico dell'operatore ==
bool Libro::operator==(Libro l){
  return (getTitolo() == l.getTitolo());
}
```

libro.cpp

La classe Cella
Tipo elemento Libro

```
#ifndef _CELLALV_H
#define _CELLALV_H

#include “libro.h”

typedef Libro tipoelem;

// classe CellaLista
class CellaLista{
   public:
      CellaLista();     //costruttore
      CellaLista(tipoelem);
      ~CellaLista(){};  //distruttore
      void scriviCella(tipoelem);
      tipoelem leggiCella() const;
      bool operator == (CellaLista);
    private:
      tipoelem etichetta;
};

#endif // _CELLALV_H
```

cellalv.h

Implementazione della classe CellaLista

```
#include "cellalv.h"

CellaLista::CellaLista() {}

CellaLista::CellaLista(tipoelem label){
  etichetta = label;
}

void CellaLista::scriviCella(tipoelem label){
  etichetta = label;
}

tipoelem CellaLista::leggiCella() const{
  return (etichetta);
}

bool CellaLista::operator==(CellaLista cella){
  return(leggiCella() == cella.leggiCella());
}
```

cellalv.cpp

La classe Lista

CellaLista contiene un elemento di tipo tipoelem

```
#ifndef _LISTAV_H
#define _LISTAV_H

#include "cellalv.h"

// lunghezza massima della lista
const int DIMENSIONE = 1024;

// classe Lista
class Lista{
    public:
       typedef int posizione;
       \\ typedef int tipoelem;
       Lista();      // costruttore
       ~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);
   private:
       CellaLista elementi[DIMENSIONE];
       int lunghezza;
       bool checkPos(posizione) const;
};
#endif // _LISTAV_H
```

listav.h

Implementazione della classe Lista
Prima parte

```
#include “lista.h”

Lista::Lista(){ crealista(); }

Lista::~Lista() {};

void Lista::creaLista(){ lunghezza = 0; }

bool Lista::listaVuota() const {
  return(lunghezza == 0);
}

Lista::posizione Lista::primoLista() const{
    return(1);  // e quindi pos(1)=pos(n+1) se la lista è vuota (n=0)
}

 bool Lista::checkPos(posizione p) const{

    return ( (0 < p) && (p < lunghezza+1))

 }


Lista::posizione Lista::succLista(posizione p) const{
  if ( checkPos(p)) // precondizione
      return(p+1);
  else return(p);
}

Lista::posizione Lista::predLista(posizione p) const{
  if  ( checkPos(p)) // precondizione
      return(p-1);
  else return(p);
}
```

listav.cpp

Implementazione della classe Lista
Seconda parte

```
bool Lista::fineLista(posizione p) const{
  if  ( checkPos(p)) // precondizione
    return( (p == 0) || (p == lunghezza+1));
  else return(false);
}

tipoelem Lista::leggiLista(posizione p) const{
  if  ( checkPos(p)) // precondizione
     return(elementi[p-1].leggiCella());
}

void Lista::scriviLista(tipoelem a, posizione p){
  if  ( checkPos(p)) // precondizione
      elementi[p-1].scriviCella(a);
}

void Lista::insLista(tipoelem a, posizione p){
  if  ( checkPos(p)) { // precondizione
      for (int i=lunghezza; i>=p; i--) elementi[i] = elementi[i-1];
      // indice elementi tra 0 e lunghezza-1
      elementi[p-1].scriviCella(a);
      lunghezza++;
    }
}

void Lista::cancLista(posizione p){
  if  ( checkPos(p)) // precondizione
  if (!listaVuota()){
          for (int i=p-1;i<(lunghezza-1);i++) elementi[i]=elementi[i+1];
          lunghezza--;
        }
}
```

listav.cpp

Funzioni di servizio

```
#ifndef _SERVIZIOLV_H
#define _SERVIZIOLV_H

#include <lista.h>

void stampaLista(Lista &);
void epurazioneLista(Lista &);
....

#endif // _SERVIZIOLV_H
```

serviziolistav.h

Funzioni di servizio utente per le liste

- stampaLista: visualizzazione elem.
- epurazioneLista: eliminazione duplicati
- ...

```
#include <iostream>
#include <lista.h>

using namespace std;

void stampaLista(Lista &l){
   cout<<"[";
   Lista::posizione indice;
   for (indice = l.primoLista();
        ((!l.fineLista(indice)) && (indice < DIMENSIONE));
        indice = l.succLista(indice)){
          cout << l.leggiLista(indice).getTitolo();
          if (!l.fineLista(l.succLista(indice))) cout << ", ";
   }
   cout<<"]\n";
}
.....
```

serviziolistav.cpp

Epurazione

Funzione di servizio **epurazione** (servizioLista.cpp)

```
void epurazioneLista(Lista &l){
  Lista::posizione p,q,r;

  p = l.primoLista();
  while (!l.fineLista(p)){
    q = l.succLista(p);
    while (!l.fineLista(q)){
      if (l.leggiLista(p) == l.leggiLista(q)){
        r = l.succLista(q);
        l.cancLista(q);
        q = r;
      }
      else
        q = l.succLista(q);
    }
    p=l.succLista(p);
  }
}
```

Test di Lista

```
#include <stdio.h>
#include <iostream>
#include <listav.h>
#include <servizioLista.h>

using namespace std;
int main(){
  Lista l;
  Lista::posizione indiceElemento = l.primoLista();
  Libro libro;

  libro.setTitolo("Primo");
  l.insLista(libro,indiceElemento=l.succLista(indiceElemento));
  libro.setTitolo("Secondo");
  l.insLista(libro,indiceElemento=l.succLista(indiceElemento));
  libro.setTitolo("Secondo");
  l.insLista(libro,indiceElemento=l.succLista(indiceElemento));
  libro.setTitolo("Quarto");
  l.insLista(libro,indiceElemento=l.succLista(indiceElemento));
  cout <<"\nL\'attuale lista e\': ";
  stampaLista(l);
  cout <<"\nOra inserisco l\'elemento Dieci nella seconda posizione...\n";
  libro.setTitolo("Dieci");
  l.insLista(libro,l.succLista(l.primoLista()));
  cout << "\nLista inserita: " << endl;
  stampaLista(l);
  cout << "\nEpurazione lista: " << endl;
  epurazioneLista(l);
  stampaLista(l);
  return 0;
}
```

testlista.cpp

Realizzazioni Proposte

- Libro: libro.h, libro.cpp
- CellaLista: cellalv.h, cellalv.cpp; (la cella contiene elementi di tipo libro);
- Lista: realizzazione lista (listav.h, listav.cpp) e funzioni di servizio (servizioLista.h, servizioLista.cpp);
- TestLista: testlistav.cpp (main per prova lista di libro)

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