PUNTATORI ***listaNUM.ccp***

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

struct Nodo {
  int    n;
  Nodo* pSucc;
};

  Nodo* pTesta = NULL;
  Nodo* pNew   = NULL;

  int nInp;

void fIns(); void fCan();
void fVis(); void fElim();

int main(){
  char sce;
    cout<< "\n---Gestione INS/CAN/VIS Lista numeri---";
    cout<< "\n\nscegliere tra :";
    cout<<   "\t i - inserisci  numero\n";
    cout<< "\t\t c - cancella   numero\n";
    cout<< "\t\t v - visualizza lista\n";
    cout<< "\t\t e - elimina    lista\n";
    cout<< "\t\t u - USCITA\n";
    cout<< "\ndigitare scelta:\n";
    do {
       cin >> sce;
       switch (sce) {
         case 'i': { cin>>nInp; fIns();  break; }
         case 'c': { cin>>nInp; fCan();  break; }
         case 'v': {            fVis();  break; }
         case 'e': {            fElim(); break; }
         case 'u': return 0;
       }

    } while (1);
}
```

container LIST ***listaNUM_list.ccp***

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

#include <list>

 list <int> n;              //-- dichiarazione list di interi di nome n

  int nInp;

void fIns(); void fCan();
void fVis(); void fElim();


int main(){
                          // i main dei due programmi sono uguali
```

I due programmi inseriscono i nuovi nodi in ordine numerico crescente, quindi dopo aver individuato la posizione corretta in cui va inserito il valore digitato.

Nella gestione della lista mediante **PUNTATORI**, data la struttura **Nodo**, il puntatore **pTesta** per il tipo strutturato Nodo viene utilizzato per puntare al **primo elemento** della lista (inizialmente, a lista vuota, conterrà NULL); il puntatore **pNew** viene utilizzato per salvare temporaneamente l'indirizzo di memoria dove viene allocato un **nuovo Nodo**; il puntatore **pSucc**, contenuto nella struttura Nodo, permette di collegare ogni Nodo al successivo.

Nella gestione mediante **container LIST**, vanno utilizzati **metodi** e **iteratori** specifici del container che riducono la gestione da parte del programmatore.

Gli elementi della lista non sono accessibili in modo diretto, ma **solo in sequenza**, a partire **dal primo della lista** (indicato da **pTesta** nella versione a puntatori, dall'**iteratore** restituito dal **metodo .begin()** nella versione con container **list**) e scorrendo la lista di Nodo in Nodo (guidati dal puntatore **pSucc** nella versione a puntatori, da **un iteratore** nella versione con **list**)

```
   } while (1);
}
```

```
void fLisVuota(){
    cout<<"ATTENZIONE!!! Lista vuota, ";
    cout<<"pTesta=" << pTesta <<endl;
}
void  fVis() {
    cout<< "\n---contenuto Lista---\n";
  Nodo* pNext  = NULL;
    if ( pTesta == NULL )              //-- lista vuota
        fLisVuota();
    else {
        pNext = pTesta;
        while ( pNext != NULL ) {
            cout << pNext -> n <<' ';//-- contenuto del nodo
            pNext = pNext -> pSucc;
        }                  //-- avanzamento del puntatore pNext
    }
    cout<< "\n\ndigitare scelta (i/c/v/e/u):\n";
}

void  fElim() {
      cout<< "\n---eliminazione Lista---\n";
  Nodo* pNext  = NULL;

    if ( pTesta == NULL )
        fLisVuota();
      else {
        cout<<"pTesta= \t\t" << pTesta << endl;
        while (pTesta != NULL){
          cout<<"elim. n=" << pTesta->n;
          pNext  = pTesta;
          pTesta = pTesta->pSucc;
          delete pNext;
          cout<< "  pTesta=\t" << pTesta << '\n';
        }
      }


      cout<< "\ndigitare scelta (i/c/v/e/u):\n";
}
```

```
void fLisVuota(){
    cout<<"ATTENZIONE!!! Lista vuota, ";
    cout<<"size=" << n.size() <<endl;
}
void  fVis() {
    cout<< "\n---contenuto Lista---\n";
  list <int>::iterator pNext;
    if ( n.empty() )                  //-- lista vuota
        fLisVuota();
    else {
        pNext = n.begin();
        while (pNext != n.end() ) {
            cout << *pNext <<' ';  //-- contenuto del nodo
            pNext++;   //-- avanzamento dell'iteratore pNext
        }
    }
    cout<< "\n\ndigitare scelta (i/c/v/e/u):\n";
}

void  fElim() {
      cout<< "\n---eliminazione Lista---\n";

      if ( n.empty()  )
            fLisVuota();
      else
            n.clear();
```

Per svuotare la lista gestita mediante container LIST, si può utilizzare il **metodo .clear()** che rilascia tutta la memoria allocata dal conteiner **n**, metodo che riduce significativamente la gestione a carico del programmatore.

Nella gestione della lista mediante PUNTATORI occorre invece partire **dal primo Nodo della lista** e liberare la memoria (**delete**) un elemento alla volta, facendo avanzare il puntatore **pTesta**.

```
      cout<< "\ndigitare scelta (i/c/v/e/u):\n";
}
```

```
void fIns() {
  Nodo* pPrec  = NULL;
  Nodo* pNext  = NULL;
    pNew       = new Nodo;
    pNew ->n   = nInp;
    pNew ->pSucc = NULL;
     cout<<"\tind.nuovo Nodo:\t"<<pNew<<endl;      //-deb
    if ( pTesta == NULL ) {
        pTesta = pNew;
        cout<<"\tpTesta=\t"<<pTesta<<endl;          //-deb
        cout<<"\tpTesta->pSucc=\t";                 //-deb
        cout<<pTesta->pSucc<<endl;                  //-deb
     }
     else
     if (pNew ->n <= pTesta->n){
        pNew -> pSucc = pTesta;
        pTesta        = pNew;
        cout<<"\tpTesta       =\t";                 //-deb
        cout<<pTesta<<endl;                         //-deb
        cout<<"\tpTesta->pSucc=\t";                 //-deb
        cout<<pTesta->pSucc<<endl;                  //-deb
     }
     else {
        pNext = pTesta;
        while ( pNext     != NULL
           &&   pNew ->n  >  pNext -> n) {
                pPrec = pNext;
                pNext = pNext -> pSucc;
        }
        pNew ->pSucc  = pNext;// inser. nuovo prima di pNext
        pPrec -> pSucc  = pNew;// agg. puntatore precedente
        cout<<"\tpPrec->pSucc=\t";                  //-deb
        cout<<pPrec->pSucc<<endl;                   //-deb
        cout<<"\tpNew->pSucc =\t";                  //-deb
        cout<<pNew->pSucc<<endl;                    //-deb
    }
}
```

```
void fIns() {

  list <int>::iterator pNext;
```

per approfondimenti consultare :
http://m.cplusplus.com/reference/list/list/

```
    if ( n.empty() )
        n.push_front(nInp);   //-- inserimento in testa alla lista
```

Nella gestione della lista mediante **PUNTATORI** l'inserimento di un nodo prima del **primo Nodo della lista** deve aggiornare:

- il puntatore **pTesta** con l'indirizzo del nuovo Nodo allocato (**pNew**)
- invece l'inserimento tra 2 nodi deve aggiornare il puntatore del nodo che precede (**pPrec -> pSucc**) con l'indirizzo del nuovo Nodo dopo aver salvato nel puntatore del nuovo Nodo (**pNew ->pSucc**) l'indirizzo contenuto in **pNext**

Per la lista gestita mediante **container** basta utilizzare il metodo **.insert**(posizione, valore) dove la posizione è quella del nodo che deve seguire (**pNext**) il nuovo Nodo.

```
    else {
        pNext = n.begin();
        while ( pNext    != n.end()
           &&   nInp     >  *pNext ) {
                pNext++;
        }
        n.insert( pNext , nInp);
      //-- inserimento prima della posizione indicata da pNext
    }
}
```

```
void fCan(){
  bool trovato  = false;
  Nodo* pPrec   = NULL;
  Nodo* pNext   = NULL;
    if ( pTesta == NULL )
        fLisVuota();
    else
    if (pTesta->n == nInp){
        trovato = true;
cout<<"\ndelete mem pTesta =\t"<<pTesta<<endl;    //-deb
        pNext   = pTesta;
        pTesta  = pTesta->pSucc;
        delete pNext;
        cout<< "\t\tpTesta=\t" << pTesta << '\n';
    } else
    if (pTesta->n < nInp) {
cout << "scorrimento lista: "<<pTesta->n <<' ';  //-deb
        pNext = pTesta->pSucc;
        pPrec = pTesta;
        while (pNext != NULL &&  ! trovato)  {
          cout << pNext->n <<' ';                //-deb
          if (pNext->n == nInp ){
            trovato = true;
            pPrec->pSucc = pNext->pSucc;
cout<<"\ndelete mem pNext =\t"<< pNext <<endl;    //-deb
            delete pNext;
          }
          if (pNew->n > nInp )
            pNext   = NULL;
          else {
            pPrec = pNext;
            pNext = pNext->pSucc;
          } }  }
    if ( ! trovato ) {
        cout<<"\nATTENZIONE! elemento NON presente\n";
        cout<< "\ndigitare scelta (i/c/v/e/u):\n";
   }  }
```

```
void fCan(){
  bool trovato  = false;
  list <int>::iterator pNext;

    if ( n.empty() )
        fLisVuota();
```

Anche per la cancellazione degli elementi della lista (rilascio della memoria allocata dal Nodo) la gestione della lista mediante **PUNTATORI** procede in modo diverso per il primo elemento dalla lista e per un elemento tra 2 nodi

```
    else {
        pNext = n.begin();
        if ( *pNext <= nInp ) {
          cout << "scorrimento lista: ";         //-deb
          while ( pNext != n.end()  &&   ! trovato)  {
            cout << *pNext <<' ';                //-deb
            if ( *pNext == nInp ){
              trovato = true;
cout<<"\ndelete mem pNext =\t"<< *pNext <<endl; //-deb
              n.erase(pNext);
            }
            if ( *pNext > nInp )
                pNext = n.end();
            else
              pNext++;
          }  }   }
    if ( ! trovato ) {
        cout<<"\nATTENZIONE! elemento NON presente\n";
        cout<< "\ndigitare scelta (i/c/v/e/u):\n";
    }  }
```

PUNTATORI ***listaNUM.ccp***	**container LIST** ***listaNUM_list.ccp***

---Gestione INS/CAN/VIS Lista numeri---

scegliere tra : i - inserisci numero
c - cancella numero
v - visualizza lista
e - elimina lista
u - USCITA

digitare scelta:
e

---eliminazione Lista---
ATTENZIONE!!! Lista vuota, pTesta=0

digitare scelta (i/c/v/e/u):
v

---contenuto Lista---
ATTENZIONE!!! Lista vuota, pTesta=0

digitare scelta (i/c/v/e/u):
c 8
ATTENZIONE!!! Lista vuota, pTesta=0
ATTENZIONE! elemento NON presente

digitare scelta (i/c/v/e/u):
i 7
ind.nuovo Nodo: 0x78a088
pTesta= 0x78a088
pTesta->pSucc= 0
i 9
ind.nuovo Nodo: 0x78a018
pPrec->pSucc= 0x78a018
pNew->pSucc = 0
i 3
ind.nuovo Nodo: 0x789f78
pTesta = 0x789f78
pTesta->pSucc= 0x78a088
i 11
ind.nuovo Nodo: 0x789ff8
pPrec->pSucc= 0x789ff8
pNew->pSucc = 0
i 3
ind.nuovo Nodo: 0x789fb8
pTesta = 0x789fb8
pTesta->pSucc= 0x789f78
v

. . . . . .
come l'altro programma
. . . . . .

digitare scelta (i/c/v/e/u):
i 7
i 9
i 3
i 11
i 3
v

---contenuto Lista---
3 3 7 9 11

. . . . . .
come l'altro programma
. . . . . .

PUNTATORI listaNUM.ccp	container LIST listaNUM_list.ccp
---contenuto Lista--- **3 3 7 9 11** digitare scelta (i/c/v/e/u): **c 8** **scorrimento lista: 3 3 7 9** ATTENZIONE! elemento NON presente digitare scelta (i/c/v/e/u): **c 2** ATTENZIONE! elemento NON presente digitare scelta (i/c/v/e/u): **c 12** **scorrimento lista: 3 3 7 9 11** ATTENZIONE! elemento NON presente	. . . . . . **come l'altro programma** . . . . . .
digitare scelta (i/c/v/e/u): **c 7** **scorrimento lista: 3 3 7** delete mem pNext = **0x78a088** **c 11** **scorrimento lista: 3 3 9 11** delete mem pNext = **0x789ff8** **c 3** delete mem pTesta = **0x789fb8** pTesta= 0x789f78 **v**	digitare scelta (i/c/v/e/u): **c 7** **scorrimento lista: 3 3 7** delete mem pNext = **7** **c 11** **scorrimento lista: 3 3 9 11** delete mem pNext = **11** **c 3** **scorrimento lista: 3** delete mem pNext = **3** **v**
---contenuto Lista--- **3 9** digitare scelta (i/c/v/e/u): **e**	**---contenuto Lista---** **3 9** digitare scelta (i/c/v/e/u): **e**
---eliminazione Lista--- pTesta= **0x789f78** elim. **n=3** pTesta= **0x78a018** elim. **n=9** pTesta= 0 **digitare scelta (i/c/v/e/u):** **c 5** ATTENZIONE!!! Lista vuota, pTesta=0 ATTENZIONE! elemento NON presente digitare scelta (i/c/v/e/u): **u**	**---eliminazione Lista---** **digitare scelta (i/c/v/e/u):** **c 5** . . . . . . **come l'altro programma** . . . . . .