```
#include <iostream>
#include <cassert>

using namespace std;


class Node{
public:
  Node(int aKey):mKey(aKey),mNext(0),mPrev(0){}
public:
  int mKey;
  Node* mNext;
  Node* mPrev;
};

class List{
  friend ostream& operator<<(ostream& out, const List& aList);
  friend istream& operator>>(istream& in, List& aList);
public:
  List();
  ~List();
  List(const List& aList);
  List& operator=(const List& aList);
  void Insert(int aKey);
  void Delete(int aIndex);
  void Output(ostream& out) const;
  int& operator[](int aIndex);
  bool IsEmpty();
private:
  void Init();
  Node* Find(int aIndex);
private:
  Node* mHead;
};

bool List::IsEmpty()
{
  return mHead->mNext==mHead;
}

void List::Init()
{
  mHead=new Node(0);
  mHead->mNext=mHead;
  mHead->mPrev=mHead;
}

ostream& operator<<(ostream& out, const List& aList)
{
  aList.Output(out);
  return out;
}

istream& operator>>(istream& in, List& aList)
{
  int key;
  while (in>>key) aList.Insert(key);
  return in;
}

List::List()
{
  Init();
}

List::~List()
{
  Node* cursor=mHead;
  do
    {
      Node* tmp=cursor->mNext;
      delete cursor;
      cursor=tmp;
    } while (cursor!=mHead);
}

List::List(const List& aList)
{
  Init();
  *this=aList;
}

List& List::operator=(const List& aList)
{
  if (!IsEmpty()) delete this;
  Init();
  Node* cursor=aList.mHead->mNext;
  do
    {
      Insert(cursor->mKey);
      cursor=cursor->mNext;
    } while (cursor!=aList.mHead);
}

void List::Insert(int aKey)
{
  Node* new_node=new Node(aKey);
  mHead->mPrev->mNext=new_node;
  new_node->mNext=mHead;
  new_node->mPrev=mHead->mPrev;
  mHead->mPrev=new_node;
}

void List::Delete(int aIndex)
{
  assert(aIndex>=0);
  Node* cursor=Find(aIndex-1);
  assert(cursor!=0);
  Node* tmp=cursor->mNext;
  cursor->mNext=cursor->mNext->mNext;
  tmp->mNext->mPrev=cursor;
  delete tmp;
}
```

```
int& List::operator[](int aIndex)
{
  assert(aIndex>=0);
  Node* cursor=Find(aIndex);
  assert(cursor!=0);
  return cursor->mKey;
}

Node* List::Find(int aIndex)
{
  int index=-1;
  Node* cursor=mHead;
  do
    {
      if (index==aIndex) return cursor;
      index++;
      cursor=cursor->mNext;
    } while (cursor!=mHead);
  return 0;
}

void List::Output(ostream& out)const
{
  Node* cursor=mHead->mNext;
  do
    {
      out<<cursor->mKey<<" ";
      cursor=cursor->mNext;
    } while (cursor!=mHead);
}

int main(){
  List list;
  for (int i=0;i<10;i++) list.Insert(i);
  cout<<list<<endl;
  cout<<"Elemento di posizione 4: "<<list[4]<<endl;
  cout<<"Elemento di posizione 9: "<<list[9]<<endl;
  cout<<"Cancella elemento di posizione 4"<<endl;
  list.Delete(4);
  cout<<"Elemento di posizione 4: "<<list[4]<<endl;
  cout<<list<<endl;
  cout<<"Cancella elemento di posizione 8"<<endl;
  list.Delete(8);
  cout<<list<<endl;
  cout<<"Cancella elemento di posizione 0"<<endl;
  list.Delete(0);
  cout<<list<<endl;
  cout<<"Copia lista"<<endl;
  List list2;
  List list3(list);
  list2=list;
  cout<<list2<<endl;
  cout<<list3<<endl;
  return 0;
}
```