IntList.h

```
#ifndef __INTLIST_H
#define __INTLIST_H

class CIntList
{
private:

protected:
      struct IntNode
      {
            int       m_iValue;
            IntNode * m_pNext;
      };

      IntNode * m_pHead;
      int       m_iCount;

public:
      CIntList();
      ~CIntList();

      bool insert( int  iValue );
      bool remove( int& iValue );
      void removeAll( void );
      bool search( int  iValue );
      void showAll( void );

      int  size() { return m_iCount; }
      bool is_empty() { return ( m_pHead == NULL ); }
};


#endif
```

IntList.cpp

```
#include <iostream.h>
#include "IntList.h"


// construtor por defeito
CIntList::CIntList()
{
	m_pHead		= NULL;
	m_iCount	= 0;
}


// destrutor
CIntList::~CIntList()
{
	removeAll();
}


// insere um novo elemento na cabeça da lista
bool CIntList::insert( int iValue )
{
	IntNode * pNode = new IntNode;

	if ( pNode == NULL ) return false;

	pNode->m_iValue = iValue;
	pNode->m_pNext  = m_pHead;
	m_pHead		= pNode;

	m_iCount++;

	return true;
}

// remove o primeiro elemento da lista e retorna o seu valor em iValue
bool CIntList::remove( int& iValue )
{
	if ( is_empty() ) return false;

	IntNode * pAux	= m_pHead;
	iValue		= m_pHead->m_iValue;
	m_pHead		= m_pHead->m_pNext;

	delete pAux;

	m_iCount --;

	return true;
}


// elimina todos os nós da lista
void CIntList::removeAll(void)
{
	int dummy;

	while ( remove( dummy ) );
}
```

```
// verifica se existe um nó com o valor passado em iValue
bool CIntList::search( int iValue )
{
	IntNode * pAux = m_pHead;

	while ( pAux != NULL )
	{
		if ( pAux->m_iValue == iValue ) return true;
		pAux = pAux->m_pNext;
	}

	return false;
}

// visualiza o conteúdo da lista
void CIntList::showAll( void )
{
	IntNode * pAux = m_pHead;

	while( pAux ){
		cout << pAux->m_iValue << endl;
		pAux=pAux->m_pNext;
	}
}
```