Список

Линейный список
Двусвязный список

Задача

- Создайте линейный список, состоящий из целых чисел.

Операции над списком:

- Создание и добавление списка;
- Удаление элемента списка;
- Вывод на экран элементов списка;
- Поиск элемента.

Объявление узла списка

```
struct DoubleList //описание узла списка
{
int data; //информационное поле
DoubleList *next; //указатель на следующий элемент
};
DoubleList *head; //указатель на первый элемент списка
```

Добавление элементов списка

```
void AddList(int value, int position)
{
DoubleList *node=new DoubleList; //создание нового элемента
node->data=value; //присвоение элементу значения
if (head==NULL) //если список пуст
{
        head=node; //определяется голова списка
        node->next=NULL; //установка указателя next
}
else
{     DoubleList *p=head;
        for(int i=1; i<position; i++) p=p->next;
        p->next=node;//добавление элемента
        node->next=NULL;
}
cout<<"\nЭлемент добавлен...\n\n";
}
```

Удаление элемента.

Поиск идет по позиции элемента

```
int DeleteList(int position)
{
if (head==NULL) { cout<<"\nСписок пуст\n\n"; return 0; }
if (head==head->next) //если это последний элемент в списке
{
delete head; //удаление элемента
head=NULL;
}
else
{
DoubleList *a=head;
for (int i=position; i>1; i--) a=a->next;
if (a==head) head=a->next;
a->next=a->next;
delete a;
}
cout<<"\nЭлемент удален...\n\n";
}
```

Вывод элементов списка

```
void PrintList()
{
if (head==NULL) cout<<"Список пуст\n";
else
{
  DoubleList *a=head;
  cout<<"\nЭлементы списка: ";
  while (a){
      cout<<a->data<<" ";
      a=a->next;}
  cout<<"\n\n";
}
}
```

Поиск элемента

```
void Find(int value)
{
if (head==NULL) cout<<"Список пуст\n";
else
{
    DoubleList *a=head; int k=0;
    cout<<"\nЭлементы списка: ";
    while (a && a->data != value)
    {a=a->next; k++;}
    cout<<" Найден эл.  "<<a->data<<"  позиция  "<<k<<endl;
}
}
```

Листинг main

```
#include "stdafx.h“
#include <iostream>
#include <cstdio>
#include <windows.h>
#include <conio.h>
using namespace std;
#define clrscr system("cls")
// Вставьте   функции
```

```
void main()
{setlocale(LC_ALL, "Rus");
int value, position, x;
do{
cout<<"1. Добавить элемент"<<endl;
cout<<"2. Удалить элемент"<<endl;
cout<<"3. Вывести список"<<endl;
cout<<"4. Найти элемент список"<<endl;
cout<<"0. Выйти"<<endl;
cout<<"\nНомер операции > "; cin>>x;
switch (x)
{
case 1:cout<<"Значение > "; cin>>value;
        cout<<"Позиция > "; cin>>position;
       AddList(value, position); break;
case 2:cout<<"Позиция > "; cin>>position;
         DeleteList(position); break;
case 3: PrintList(); break;
case 4:cout<<"Элемент > "; cin>> value; Find(value);break;
}
} while (x!=0);
}
```

Задача2

```
#include <iostream>
#include <cstdio>
#include <windows.h>
#include <conio.h>
#define _CRTDBG_MAP_ALLOC
#include <crtdbg.h>
using namespace std;
#define clrscr system("cls")

struct node
{
   int x;
   node* next;
};
node *first =0 ; //Указатель на ПЕРВЫЙ элемент списка
node *endp =NULL ;   //Указатель на ПОСЛЕДНИЙ элемент списка
int s =0 ;
```

Задача2

```
void addToEnd(int a){
   node *newp = new node;
   endp->next = newp;
   endp = newp;
   endp->x = a;
   endp->next =NULL ;
}

void display() {
   node* current = first;
   while(current)
   {
      cout << current->x << endl;
      current = current->next;
   }
}
```

Задача2

```
int main(){
	_CrtSetDbgFlag(_CRTDBG_ALLOC_MEM_DF | _CRTDBG_LEAK_CHECK_DF);//Для контроля утечки памяти
	setlocale(LC_ALL,"Rus");

   int n,value;
   char c;
   cout<<"Введите элемент списка:\n";
   first = new node;
   cin>>value;
   first->next = NULL;
   first->x = value;
   endp = first;
   s++;
   c = 'y';
   while (c!='n')
   {
      cout<<"Добавьте еще один элемент:\n";
      cin>>value;
      addToEnd(value);
      s++;
      cout<<"Новый элемент?\n";
      do
      {
          c = getch();
      } while(c!='n' && c!='y');
      clrscr;
   };
   cout << "Текущий список:\n";
   display();
   getch();
}
```

Линейный список

- Узел списка **Node** представляет собой структуру, которая содержит три поля - строку, целое число и указатель на такой же узел.

```
struct Node {
char word[40]; // область данных
int count;
Node *next; // ссылка на следующий узел
};
typedef Node *PNode; // тип данных: указатель на
  узел
```

Указатель **Head** указывает на начало списка, то есть, объявлен в виде

```
PNode Head = NULL;
```

Создание элемента списка

- Для того, чтобы добавить узел к списку, необходимо создать его, то есть выделить память под узел и запомнить адрес выделенного блока.

```
PNode CreateNode ( char NewWord[] )
{
PNode NewNode = new Node; // указатель на новый узел
strcpy(NewNode->word, NewWord); // записать слово
NewNode->count = 1; // счетчик слов = 1
NewNode->next = NULL; // следующего узла нет
return NewNode; // результат функции – адрес узла
}
```

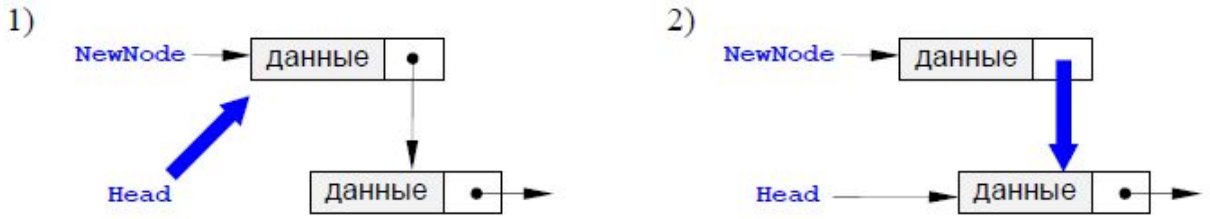
- После этого узел надо добавить к списку (в начало, в конец или в середину).

Добавление узла в начало списка

- При добавлении нового узла NewNode в начало списка надо

1) установить ссылку узла NewNode на голову существующего списка

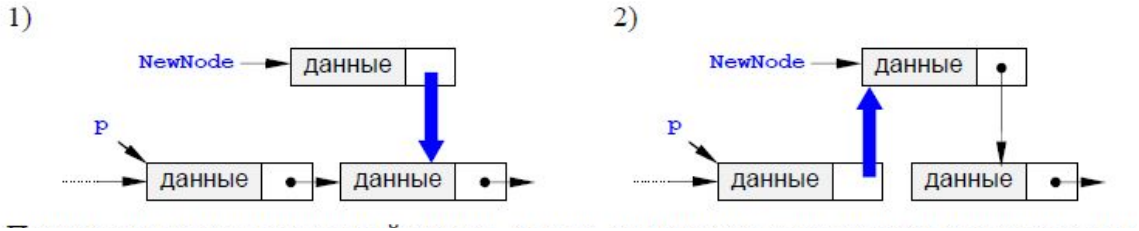
2) установить голову списка на новый узел.




```
void AddFirst (PNode &Head, PNode NewNode)
{
NewNode->next = Head;
Head = NewNode;
}
```

Добавление нового узла после заданного узла с адресом p

- Выполняется в два этапа:
- 1) установить ссылку нового узла на узел, следующий за данным;
- 2) установить ссылку данного узла p на NewNode.




```
void AddAfter (PNode p, PNode NewNode)
{
NewNode->next = p->next;
p->next = NewNode;
}
```

Добавление узла перед заданным p

- Для того, чтобы получить адрес предыдущего узла, нужно пройти весь список сначала. Затем задача сведется либо к вставке узла в начало списка (если заданный узел – первый), либо к вставке после заданного узла.

```
void AddBefore(PNode &Head, PNode p, PNode NewNode)
{
PNode q = Head;
if (Head == p)
 {
AddFirst(Head, NewNode); // вставка перед первым узлом
return;
}
while (q && q->next!=p) // ищем узел, за которым следует p
q = q->next;
if ( q ) // если нашли такой узел,
AddAfter(q, NewNode); // добавить новый после него
}
```

Добавление узла в конец списка

Для решения задачи надо сначала найти последний узел, у которого ссылка равна **NULL,** а затем воспользоваться процедурой вставки после заданного узла. Отдельно надо обработать случай, когда список пуст.

```
void AddLast(PNode &Head, PNode NewNode)
{
PNode q = Head;
if (Head == NULL)
{ // если список пуст,
   AddFirst(Head, NewNode); // вставляем первый элемент
   return;
}
while (q->next) q = q->next; // ищем последний элемент
AddAfter(q, NewNode);
}
```

Проход по списку

```
PNode p = Head; // начали с головы списка
while ( p != NULL ) // пока не дошли до конца
{
// делаем что-нибудь с узлом p
p = p->next; // переходим к следующему узлу
}
```

Поиск узла в списке

Алгоритм:

1) начать с головы списка;
2) пока текущий элемент существует (указатель – не NULL), проверить нужное условие и перейти к следующему элементу;
3) закончить, когда найден требуемый элемент или все элементы списка просмотрены.

```
//ищем слово NewWord в поле word
PNode Find (PNode Head, char NewWord[])
{
PNode q = Head;
while (q && strcmp(q->word, NewWord))
  q = q->next;
return q;    //возвращает адрес узла
}
```

Удаление узла


```
void DeleteNode(PNode &Head, PNode OldNode)
{
PNode q = Head;
if (Head == OldNode)
    Head = OldNode->next; // удаляем первый элемент
else {
    while (q && q->next != OldNode) // ищем элемент
    q = q->next;
    if ( q == NULL ) return; // если не нашли, выход
    q->next = OldNode->next;
}
delete OldNode; // освобождаем память
}
```

Задача (алфавитно-частотный словарь).

- В файле записан текст.
- Нужно записать в другой файл в столбик все слова, встречающиеся в тексте, в алфавитном порядке, и количество повторений для каждого слова.
- Проблемы:

1) количество слов заранее неизвестно (статический массив);
2) количество слов определяется только в конце работы (динамический массив).

- Решение – список.
- Алгоритм:

1) создать список;
2) если слова в файле закончились, то стоп.
3) прочитать слово и искать его в списке;
4) если слово найдено – увеличить счетчик повторений, иначе добавить слово в список;
5) перейти к шагу 2.

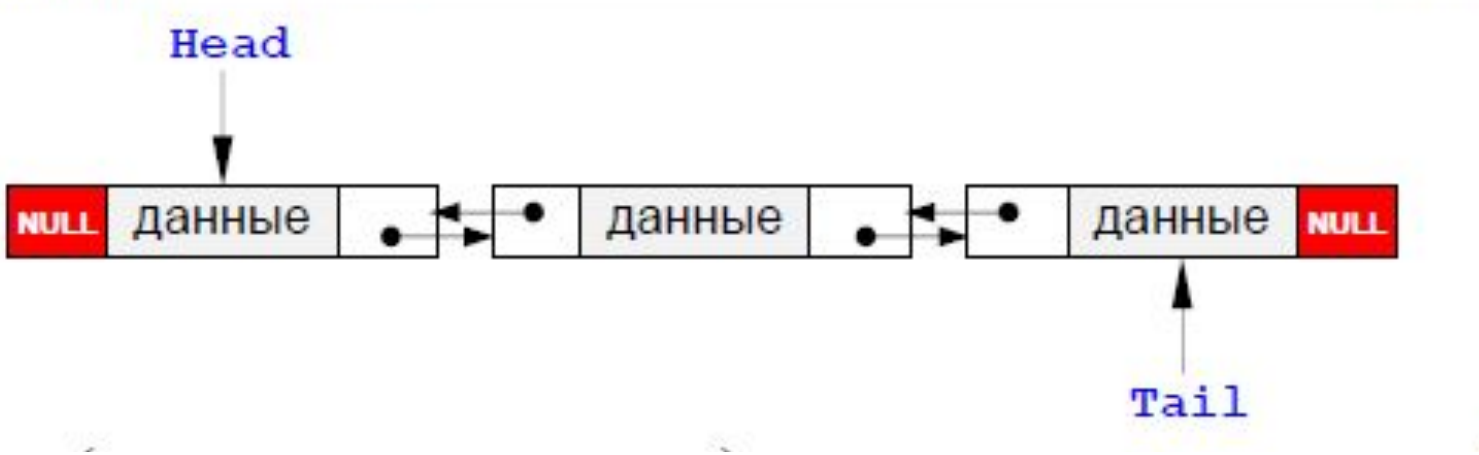
- Для того, чтобы добавить новое слово NewWord в нужное место (в алфавитном порядке), требуется найти адрес узла, *перед кото*рым надо вставить новое слово. Это будет первый от начала списка узел, для которого «его» слово окажется «больше», чем новое слово.
- Функция **strcmp** возвращает «разность» первого и второго слова.

```
PNode FindPlace (PNode Head, char
  NewWord[])
{
PNode q = Head;
while (q && (strcmp(q->word, NewWord) >
  0))
q = q->next;
return q;
}
```

Программа обрабатывает файл **input.txt** и составляет для него алфавитно-частотный словарь в файле **output.txt.**

```
void main()
{   PNode Head = NULL, p, where;
     FILE *in, *out;//объявляем файлы
     char word[80];
     int n;
     in = fopen ( "input.dat", "r" ); //открываем файл input.dat для чтения
while ( 1 ) {
    n = fscanf ( in, "%s", word ); // читаем слово из файла
    if ( n <= 0 ) break;
    p = Find ( Head, word ); // ищем слово в списке
    if ( p != NULL ) // если нашли слово,
     p->count ++; // увеличить счетчик
    else {     p = CreateNode ( word ); // создаем новый узел
     where = FindPlace ( Head, word ); // ищем место
     if ( ! where )      AddLast ( Head, p );
     else   AddBefore ( Head, where, p );
          }}
fclose(in);
out = fopen ( "output.dat", "w" );
p = Head;
while ( p ) { // проход по списку и вывод результатов
    fprintf ( out, "%-20s\t%d\n", p->word, p->count );
    p = p->next; }
fclose(out);
}
```

Двусвязный список



Каждый узел содержит (кроме полезных данных) также ссылку на следующий за ним узел (поле next) и предыдущий (поле prev).

Узел объявляется так:

```
struct Node {
char word[40]; // область данных
int count;
Node *next, *prev; // ссылки на соседние
  узлы
};
typedef Node *PNode; // тип данных «указатель на
  узел»
```

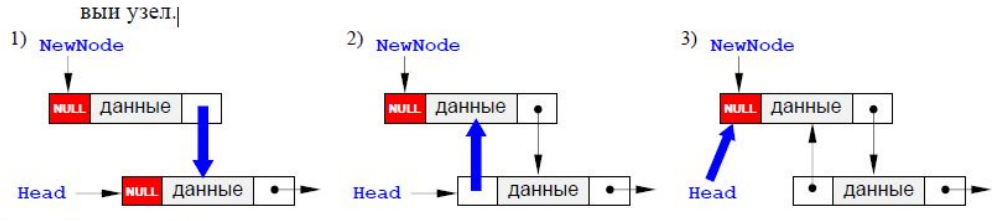
Указатель Head указывает на начало списка, а указатель Tail – на конец списка:

```
PNode Head = NULL, Tail = NULL;
```

Добавление узла в начало списка

При добавлении нового узла NewNode в начало списка надо:

1) установить ссылку next узла NewNode на голову существующего списка и его ссылку prev в NULL;
2) установить ссылку prev бывшего первого узла (если он существовал) на NewNode;
3) установить голову списка на новый узел;
4) если в списке не было ни одного элемента, хвост списка также устанавливается на новый узел.




```
void AddFirst(PNode &Head, PNode &Tail, PNode NewNode)
{
NewNode->next = Head;
NewNode->prev = NULL;
if ( Head ) Head->prev = NewNode;
Head = NewNode;
if ( ! Tail ) Tail = Head; // этот элемент – первый
}
```

Добавление узла в конец списка

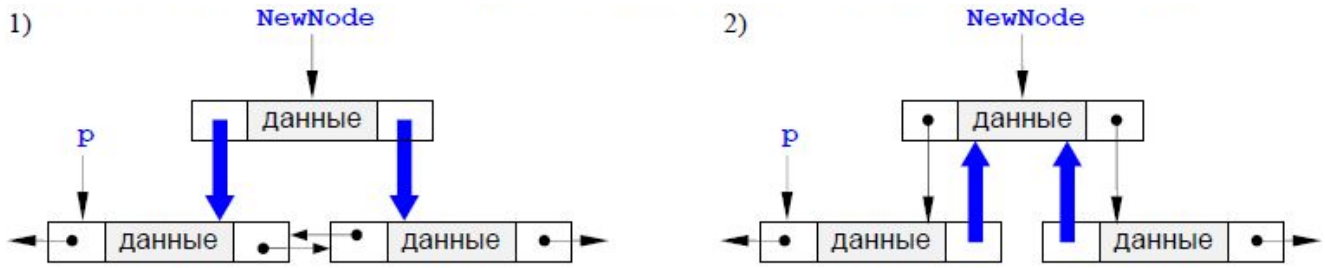
- Благодаря симметрии добавление нового узла NewNode в конец списка проходит совершенно аналогично, в процедуре надо везде заменить Head на Tail и наоборот, а также поменять prev и next.

Добавление нового узла после заданного p

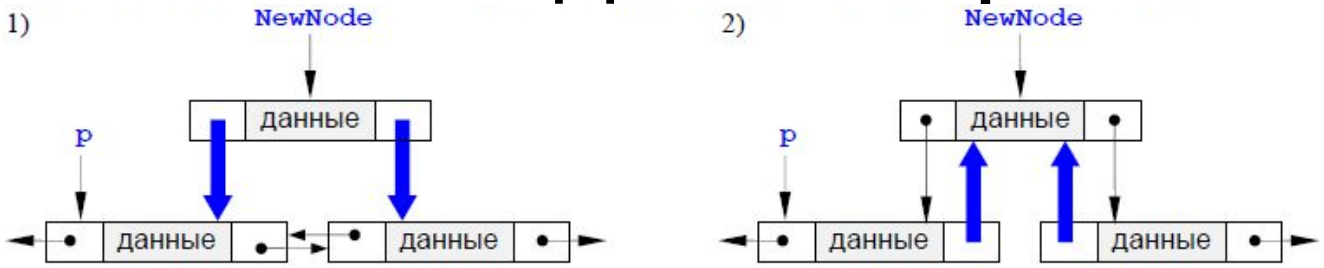
Если узел p является последним, то операция сводится к добавлению в конец списка (см. выше). Если узел p – не последний, то операция вставки выполняется в два этапа:

1) установить ссылки нового узла на следующий за данным (*next) и предшествующий* ему (*prev);*

2) установить ссылки соседних узлов так, чтобы включить NewNode в список.

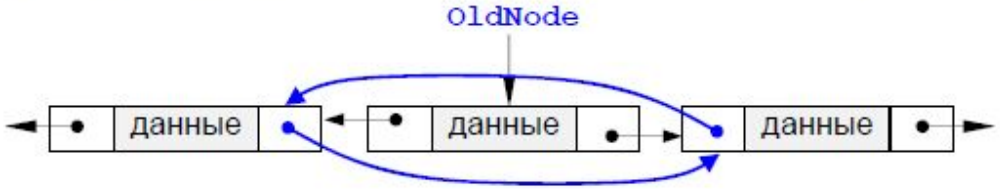


Добавление нового узла после заданного p




```
void AddAfter (PNode &Head, PNode &Tail,
PNode p, PNode NewNode)
{
if ( ! p->next )
AddLast (Head, Tail, NewNode); // вставка в конец списка
else {
NewNode->next = p->next; // меняем ссылки нового узла
NewNode->prev = p;
p->next->prev = NewNode; // меняем ссылки соседних узлов
p->next = NewNode;
}
}
```

Удаление узла




```
void Delete(PNode &Head, PNode &Tail, PNode OldNode)
{
if (Head == OldNode) {
   Head = OldNode->next; // удаляем первый элемент
if ( Head )
   Head->prev = NULL;
else Tail = NULL; // удалили единственный элемент
}
else {
   OldNode->prev->next = OldNode->next;
if ( OldNode->next )
   OldNode->next->prev = OldNode->prev;
else Tail = NULL; // удалили последний элемент
}
delete OldNode;
}
```

Циклические списки

Иногда список (односвязный или двусвязный) замыкают в кольцо, то есть указатель next последнего элемента указывает на первый элемент, и (для двусвязных списков) указатель prev первого элемента указывает на последний.

В таких списках понятие «хвоста» списка не имеет смысла, для работы с ним надо использовать указатель на «голову», причем «головой» можно считать любой элемент.

Файл : IntList.h // Класс IntList

```
#ifndef __INT_LIST
#define __INT_LIST
#include <stdlib.h>
class IntList {
/* класс ListItem представляет элемент списка, связанный со
   следующим с помощью поля next
*/
 struct ListItem {
   int item;        // значение элемента списка
   ListItem *next;   // указатель на следующий элемент списка
   // Конструктор для создания нового элемента
   ListItem(int i, ListItem *n = NULL) { item = i; next = n; }
 };
 int count;           // счетчик числа элементов
 ListItem *first;      // первый элемент списка
 ListItem *last;       // последний элемент списка

public :
 // Конструктор "по умолчанию" - создание пустого списка
 IntList() { count = 0; first = last = NULL; }

 // Конструктор "копирования" – создание копии имеющегося
списка
 IntList(const IntList & src);

 // Деструктор списка
 ~IntList();

 // Доступ к первому элементу списка
 int head() const { return first->item; }
 int & head() { return first->item; }

 // Доступ к последнему элементу списка
 int tail() const { return last->item; }
```

```
// Добавить элементы в конец списка
 void addLast(const IntList & src);

 // Добавить один элемент в начало списка
 void addFirst(int item);

 // Добивать один элемент в конец списка
 void addLast(int item);

 // Удалить первый элемент
 int removeFirst();

 // Удаление заданного элемента
 bool remove(int n);

 // Searching and removing an element
 void insert(int n);

 // количество элементов списка
 int getCount() { return count; }

 // Printing all the elements to the standard output stream
 void printAll();
};   // Конец определения класса IntList

#endif
```

Файл : IntList.cpp

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

using namespace std;

// Реализация конструктора копирования
IntList::IntList(const IntList & src) {
  count = 0;
  first = last = NULL;
  addLast(src);   // добавляет список src в конец списка this
}

// Реализация деструктора
IntList::~IntList() {
  ListItem *current = NULL; // указатель на элемент, подлежащий удалению
  ListItem *next = first;   // указатель на следующий элемент
  while (next) {          // пока есть еще элементы в списке
    current = next;
    next = next->next;      // переход к следующему элементу
    delete current;        // освобождение памяти
  }
}

// Добавление элементов заданного списка в конец определяемого
void IntList::addLast(const IntList & src) {
  for (ListItem *cur = src.first; cur;  cur = cur->next)
    addLast(cur->item);  // добавление одного элемента – см. ниже
}
```

продолжение

```
// Добавление одного элемента в начало списка
void IntList::addFirst(int item) {
  // создаем новый элемент списка
  ListItem *newItem = new ListItem(item, first);
  if (!first) {
    // список был пуст – новый элемент будет и первым, и последним
    last = newItem;
  }
  first = newItem;
  count++;    // число элементов списка увеличилось.
}

// Добавление одного элемента в конец списка
void IntList::addLast(int item) {
  // создаем новый элемент списка
  ListItem *newItem = new ListItem(item);
  if (!last) {
    // список был пуст – новый элемент будет и первым, и последним
    first = newItem;
  } else {
    // новый элемент присоединяется к последнему элементу списка
    last->next = newItem;
  }
  last = newItem;
  count++;    // число элементов списка увеличилось.
}

// Удаление первого элемента из списка
int IntList::removeFirst() {
  int res = first->item;  // содержимое первого элемента
  first = first->next;    // второй элемент становится первым
  count--;                // число элементов списка уменьшилось
  return res;   // удаленный элемент возвращается в качестве результата
}
```

```
// Удаление заданного элемента
bool IntList::remove(int n) {
  ListItem *pred = 0,        // указатель на предыдущий элемент
           *current = first;  // указатель на текущий элемент
  while (current) {
    if (current->item == n) {
      if (pred) {           // корректируем ссылку на удаляемый элемент
        pred->next = current->next;
      }
      if (current == last) { // удаляется последний элемент
        last = pred;         // корректируем ссылку на последний элемент
      }
      delete current;       // освобождаем память
      count--;              // уменьшаем количество элементов
      return true;
    } else {                // переходим к следующему элементу
      pred = current;
      current = current->next;
    }
  }
  // удаляемый элемент не найден
  return false;
}

// Вставка нового элемента в упорядоченный список
void IntList::insert(int n) {
  ListItem *pred = NULL,    // элемент, предшествующий вставляемому
           *succ = first;   // элемент, следующий за вставляемым
  while (succ != NULL && succ->item < n) { // поиск места вставки
    pred = succ;
    succ = succ->next;
  }
  // генерируем новый элемент:
  ListItem *newItem = new ListItem(n, succ);
  if (succ == NULL) {      // вставляемый элемент – последний
    last = newItem;
  }
  // вставляем новый элемент в список
  if (pred == NULL) {
    first = newItem;
  } else {
    pred->next = newItem;
  }
  count++;
}
// Вывод элементов списка в текстовом виде в стандартный выходной поток
void IntList::printAll() {
  ListItem *current = first;  // Указатель на элемент
  while (current) {
    cout << current->item << ' ';
    current = current->next;  // Переход к следующему элементу
  }
  cout << endl;
}
```