Описание однонаправленного линейного динамического списка (L1)

```
struct L1 {
                char name[20]; // Имя переменной
                char value[10]; // Значение переменной
                comp* next; //Ссылка на следущий элемент списка
};
```

```
 struct Form {
                L1* head; // Первый элемент (голова) списка
                L1* tail; // Последний элемент (хвост) списка
        };
```

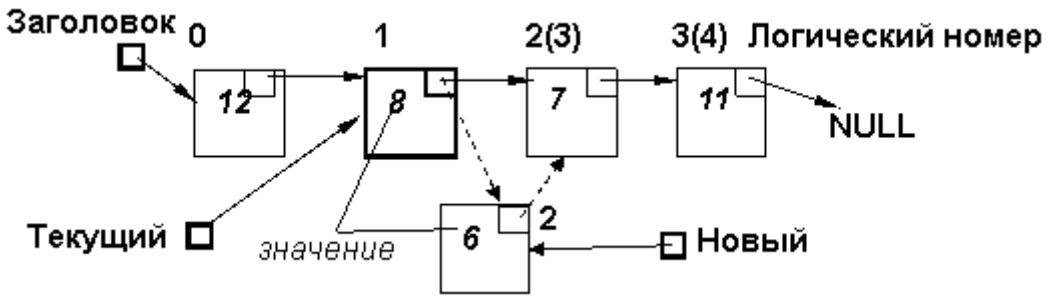
```
// Создание пустого списка
void constr_list(Form &l)
{ l.head = NULL; }
// Проверка списка на пустоту
bool chk_empty(Form l)
{ return (l.head==NULL); }

Form vars; // Динамический список
constr_list(vars);
```

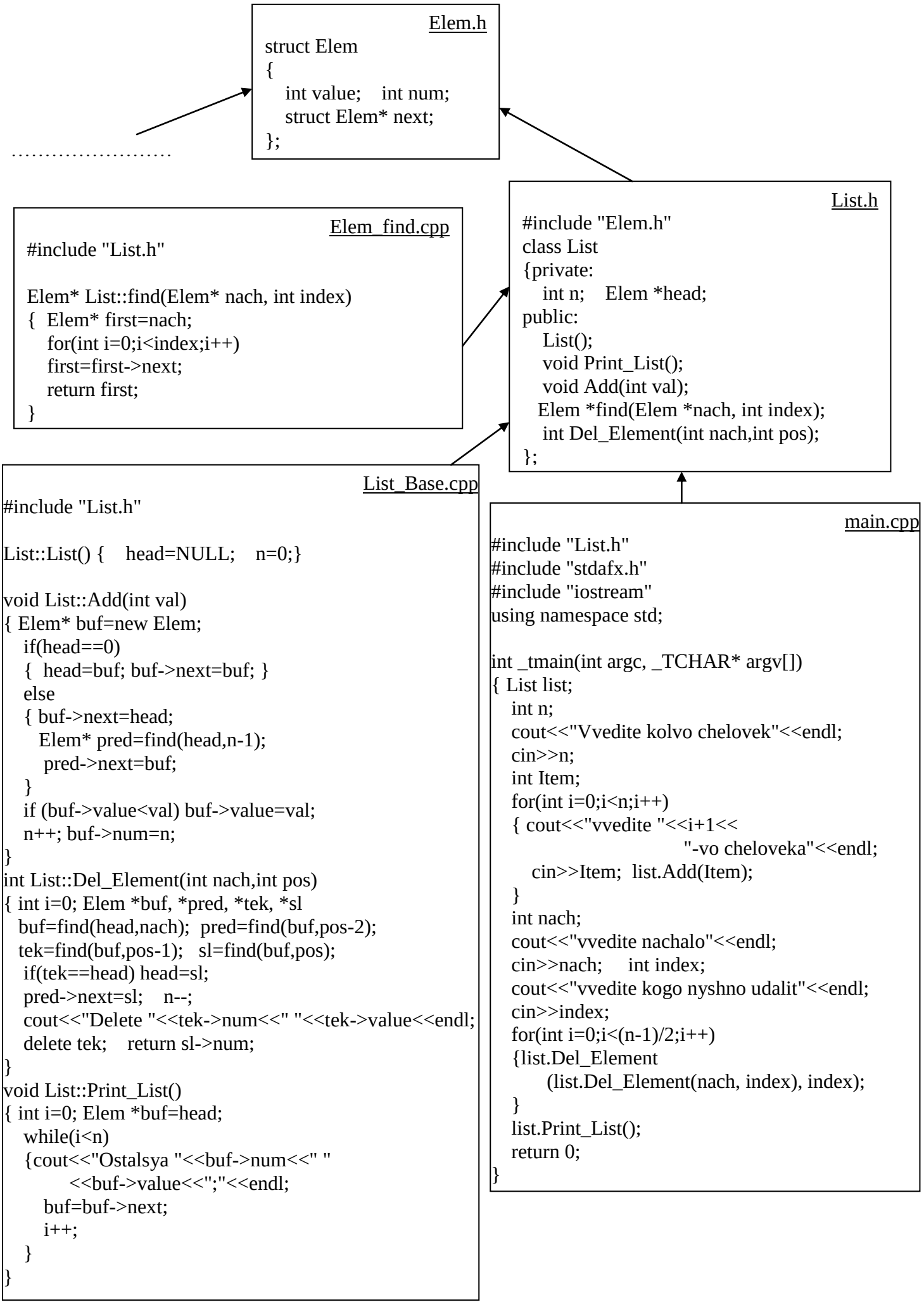
```
// Изменение значения элемента
void Elem_edit(L1 &c, char* v)
{ strcpy_s(c.value, 10, v);}
```

```
// Включение в список нового компонента
void Elem_in(Form &l, char* n, char* v)
{L1* c = new L1();
        strcpy_s(c->name, 20, n);
        strcpy_s(c->value, 10, v);
        c->next = NULL;
        if (chk_empty(l))
                l.head = c;
        else
                l.tail->next = c;
        l.tail = c;
}
```

```
// Поиск компонента в списке по имени
L1* search(Form l, char *n)
{       L1* r;
        r = l.head;
        while (r != NULL)
        {       if (!strcmp(r->name,n))
                        return r;
                r = r->next;
        }
        return r;
}
```

```
// Удаление элемента из списка
void Elem_del(Form &l, comp* c)
{if (c == l.head)
        {       l.head = c->next;
                delete c;
                return;
        }
        L1* r;
        r = l.head;
        while (r->next != c)
                r = r->next;
        r->next = c->next;
        delete c;
}
```




Elem.h
struct Elem
{
int value; int num;
struct Elem* next;
};
……
List.h
#include "Elem.h"
class List
{private:
int n; Elem *head;
public:
List();
void Print_List();
void Add(int val);
Elem *find(Elem *nach, int index);
int Del_Element(int nach,int pos);
};
Elem_find.cpp
#include "List.h"
Elem* List::find(Elem* nach, int index)
{ Elem* first=nach;
for(int i=0;i<index;i++)
first=first->next;
return first;
}
List_Base.cpp
#include "List.h"
List::List() { head=NULL; n=0;}
void List::Add(int val)
{ Elem* buf=new Elem;
if(head==0)
{ head=buf; buf->next=buf; }
else
{ buf->next=head;
Elem* pred=find(head,n-1);
pred->next=buf;
}
if (buf->value<val) buf->value=val;
n++; buf->num=n;
}
int List::Del_Element(int nach,int pos)
{ int i=0; Elem *buf, *pred, *tek, *sl
buf=find(head,nach); pred=find(buf,pos-2);
tek=find(buf,pos-1); sl=find(buf,pos);
if(tek==head) head=sl;
pred->next=sl; n--;
cout<<"Delete "<<tek->num<<" "<<tek->value<<endl;
delete tek; return sl->num;
}
void List::Print_List()
{ int i=0; Elem *buf=head;
while(i<n)
{cout<<"Ostalsya "<<buf->num<<" "
<<buf->value<<";"<<endl;
buf=buf->next;
i++;
}
}
main.cpp
#include "List.h"
#include "stdafx.h"
#include "iostream"
using namespace std;
int _tmain(int argc, _TCHAR* argv[])
{ List list;
int n;
cout<<"Vvedite kolvo chelovek"<<endl;
cin>>n;
int Item;
for(int i=0;i<n;i++)
{ cout<<"vvedite "<<i+1<<
"-vo cheloveka"<<endl;
cin>>Item; list.Add(Item);
}
int nach;
cout<<"vvedite nachalo"<<endl;
cin>>nach; int index;
cout<<"vvedite kogo nyshno udalit"<<endl;
cin>>index;
for(int i=0;i<(n-1)/2;i++)
{list.Del_Element
(list.Del_Element(nach, index), index);
}
list.Print_List();
return 0;
}

```
class SListNode // SListNode - название класса, представляющего
{             // узел односвязного списка
public:         int data;   // целая переменная хранит данные узла
                SListNode* next; // указатель  на следующий узел списка
};
```

```
SListNode::SListNode ()
{
  data = 0;
  next = NULL;
};
```

```
SListNode* list = new SListNode;
list->data = 0;
list->next = new SListNode; // создание следующего узла
list->next->data = 1;
list->next->next = new SListNode;
list->next->next->data = 2;
```

```
void SListNode::InsertAfter( int d)
{public:
  SListNode* new_node = new SListNode; // Создаём указатель  на узел.
  new_node->data = d;    // Заполняем поле data.
  new_node->next = next; // Заполняем поле next. Этому полю присваиваем
                         // значение поля next текущего элемента
  next = new_node;      // Меняем поле next текущего элемента. Теперь, текущий
};                       // узел указывает  на вновь созданный
```

```
class SLinkedList
{public:
  SListNode* head; // первый элемент списка
  SListNode* tail; // последний элемент списка
  int count;
};
```

```
SLinkedList::SLinkedList ()
          : count(0), head(NULL), tail(NULL) {}
SLinkedList::~SLinkedList()
{  SListNode* delNode = head; SListNode* temp;
  while (delNode != NULL)
    {temp = delNode->next; delete delNode;
     delNode = temp;
    }
}
```

```
void SLinkedList::PushBack (int d)
{if (count == 0)  //  В списке нет элементов.
  {head = tail = new SListNode;
   head->data = d;
  }else //  В списке есть хотя бы один элемент
     {tail->InsertAfter(d);
      tail = tail->next;
     }
  count++;
}
```

```
void SLinkedList::PopBack()
{if (count == 1)
  { delete tail; head = tail = NULL; count--;}
  if (count > 1)
  { SListNode* temp = new SListNode;
    temp = head;
    while (temp->next != tail) node = temp->next;
    tail = temp; delete temp->next;
    tail->next = NULL;
    count--;
  }
}
```

```
void SLinkedList::PushFront (int d)
{if (count == 0)
  {head = tail = new SListNode;
   head->data = d;
  }else
    { SListNode* new_node = new SListNode;
       new_node->data = d;
       new_node->next = head;
       head = new_node;
     }
  count++;
}
```

```
void SLinkedList::PopFront()
{if (count != 0)
  { SListNode* temp = head;
    head = head->next;
    delete temp;
    count--;
   if (head == NULL) //  в списке был один элемент
      tail = head;
  }
}
```

```
SLinkedList list;                SListNode* node;
list.PushBack(25);      list.PushBack(12);      list.PushBack(3);   list.PushFront(1);  list.PushFront(2);
node = list.head;   node = node->next;          node = node->next;      cout << node->data << "\n"; // 25
```

```
---------------------------------------------------------------------------
1    typedef struct tag_lib {
2    char title[100];
3    char author[100];
4    int value;
5    } LIB;
6
7    /*Структура, которая описывает связи между строками таблицы, и представляет собой объект
8    данных.
9    Здесь *prev и *next – указатели на предыдущую и следующую строки соответственно.*/
10   ////////////////////////////////////////////////////
11   typedef struct tag_obj {
12   LIB lib;
13   typedef struct tag_obj* prev, *next;
14   } OBJ;
15
16   ////////////////////////////////////////////////////
17   OBJ* add_obj(char* title, char* author, int value)//функция использует три параметра для ввода
18   данных в структуру LIB.
19   {
20    OBJ* current = (OBJ *)malloc(sizeof(OBJ));//создается новая структура типа OBJ.
21    strcpy(current->lib.title, title); //запись информации в структуру LIB
22    strcpy(current->lib.author, author); //запись информации в структуру LIB
23    current->lib.value = value; //запись информации в структуру LIB
24    current->prev = tail;//инициализируются указатель prev добавленного объекта
25    //prev указывает на предыдущий объект, т.е. равен указателю tail.
26    current->next = NULL;//инициализируются указатель next добавленного объекта
27    //добавление осуществляется в конец списка, то указатель next должен быть равен NULL
28    if(tail != NULL) tail->next = current;
29    /*объект, на который указывает указатель tail, становится предпоследним и его
30    указатель next должен указывать на последний объект, т.е. быть равным указателю current.*/
31    if(head == NULL) head = current;
32    /*Затем проверяется, является ли добавляемый объект первым (head == NULL),
33    и если это так, то указатель head приравнивается указателю current */
34    tail = current;//указатель tail инициализируется на последний объект
35    return current;//возвращает указатель на созданный объект
36   }
37
38   ////////////////////////////////////////////////////
39   OBJ* del_obj(OBJ* current)//функция удаления элемента.
40   { //del_obj() в качестве аргумента использует указатель на объект, который следует удалить.
41    if(current == head)
42   //Сначала выполняется проверка для инициализации указателя head, в том случае,
43   //если удаляется первый объект, на который он указывает.
44   if(current->prev != NULL) head = current->prev;
45    else head = current->next;
46    if(current == tail) //Аналогичная проверка осуществляется для tail.
47   if(current->next != NULL) tail = current->next;
48    else tail = current->prev;
49    if(current->prev != NULL)
50   //проверка: если предыдущий объект относительно текущего существует,
51   //то его указатель на следующий объект следует переместить.
52   current->prev->next = current->next;
53    if(current->next != NULL)
54   //Аналогичная проверка выполняется и для следующего объекта относительно удаляемого.
```

```
current->next->prev = current->prev;
 free(current);//удаления объекта из памяти
return head;//возвращается указатель на первый объект.
}

//////////////////////////////////////////////////
int main()
{
 OBJ *current = NULL;
 int value;
 char title[100], author[100];
 do
 {
 printf("Введите название книги: ");
 scanf("%s",title);
 printf("Введите автора: ");
 scanf("%s",author);
 printf("Введите стоимость: ");
 scanf("%d",&value);
 current = add_obj(title,author,value);//формирует связанный список на основе введенных данных.
 printf("Для выхода введите 'q'");
 } while(scanf("%d",&value) == 1);
 current = head; //указатель current передвигается на первый объект
 while(current != NULL)
 {
 printf("Title: %s, author %s, value = %d\n",
 current->lib.title, current->author.old, current->lib.value);
 current = current->next;
 }
 while(head != NULL)
 del_obj(head);
 return 0;
}
```