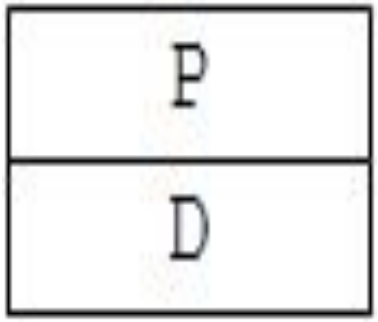
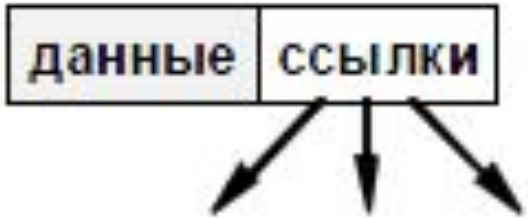
ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

1. Основные понятия
2. Объявление динамических структур
3. Доступ к данным
4. Работа с памятью
5. Списки
6. Основные операции со списками

Элемент динамической структуры



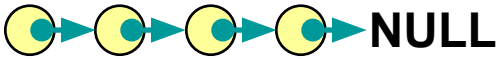


P – указатель;
D – данные.

Типы структур:

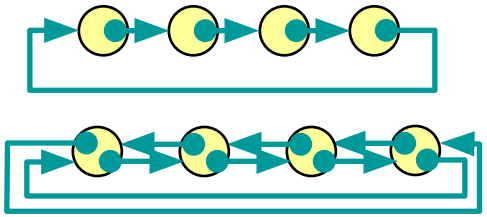
списки

односвязный

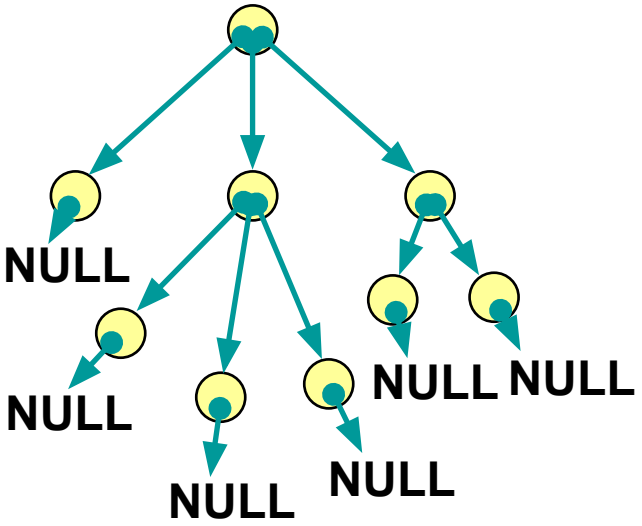


двунаправленный (двусвязный)

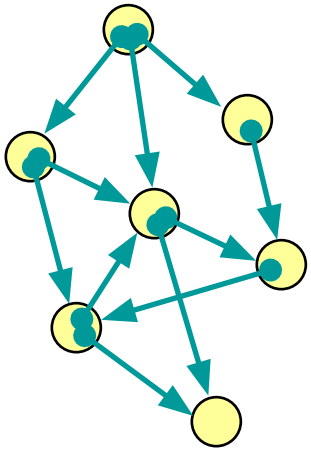
циклические списки (кольца)



деревья



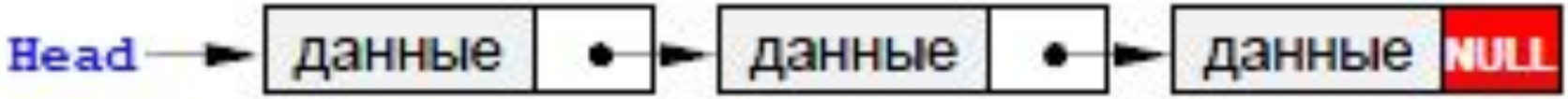
графы

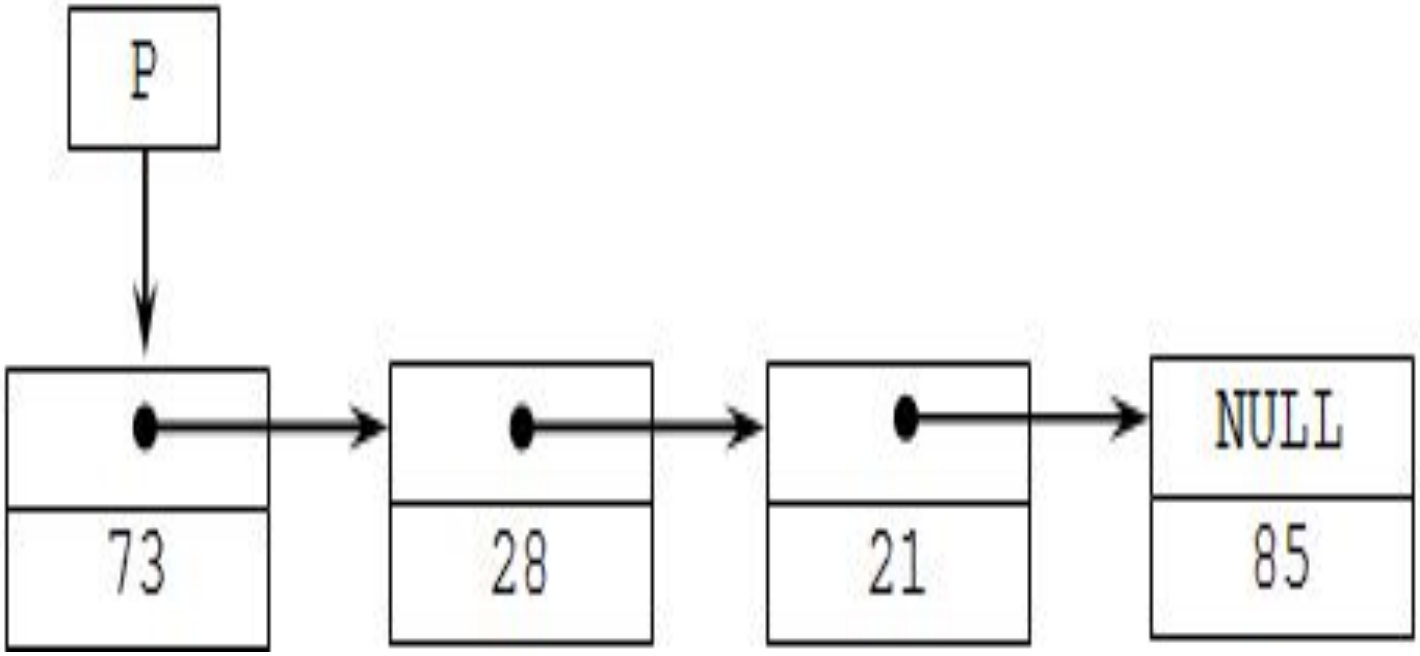


Объявление динамических структур

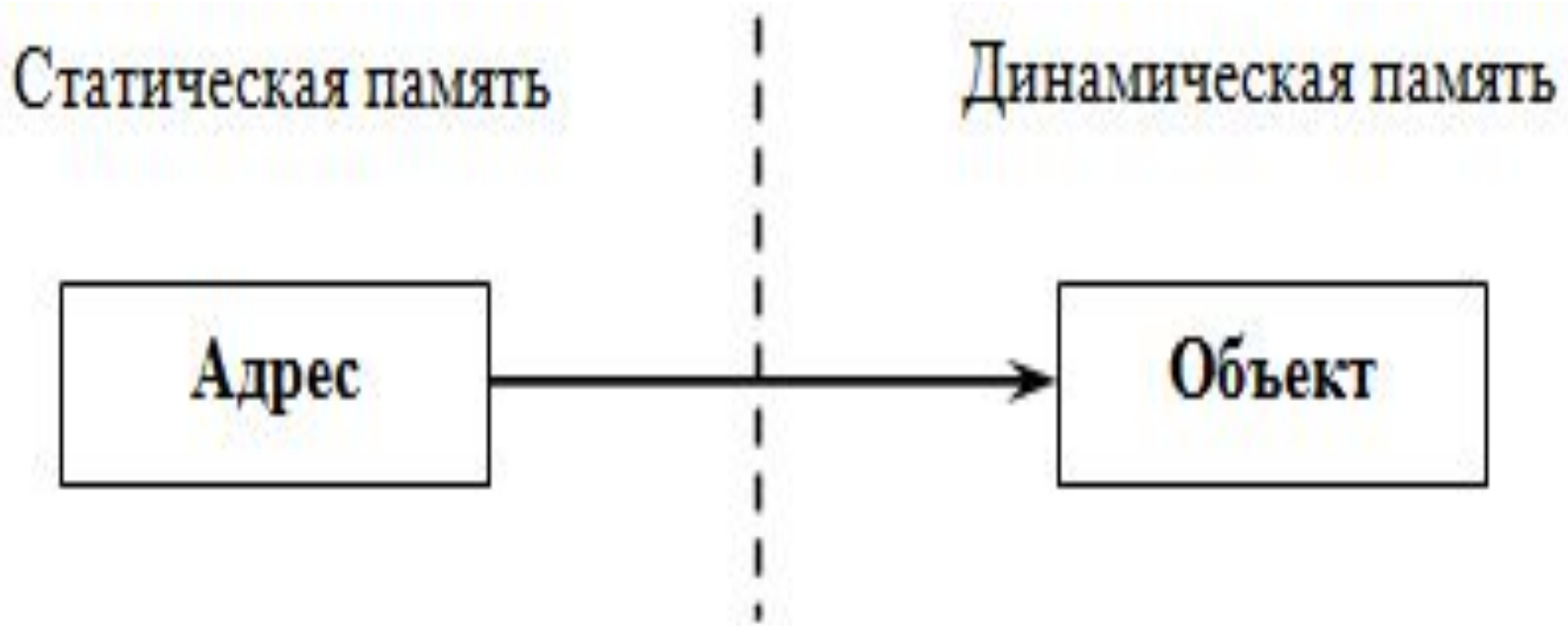
```
struct имя_типа {
        информационное поле;
        адресное поле;
                };
```

```
struct TNode {
//информационное поле
int Data;
//адресное поле
TNode *Next;
};
```


Head
данные
данные
данные
NULL


P
73
28
21
NULL
85

Доступ к данным в динамических структурах



```
УказательНаСтруктуру->ИмяЭлемента
```

Например:

```
p->Data;
p->Next;
```

```
struct Node
{
char Name[20];
int Value;
Node *Next;
     };
```

```
int _tmain()
{ //объявляется указатель
Node *PNode;
//выделяется память
PNode = new Node;
//присваиваются значения
strcpy(PNode->Name , "STO");
PNode->Value = 28;
PNode->Next = NULL;
cout<< "name="<<PNode->Name
<<"\nvalue="<< PNode->Value;
//освобождение памяти
delete PNode;}
```

```
Memo1->Clear();
Node *PNode;
PNode = new Node;
AnsiString St="STO";
strcpy(PNode->Name ,St.c_str());
PNode->Value = 28;
PNode->Next = NULL;

Memo1->Lines->Add("name="+
AnsiString(PNode->Name)+
" value="+(PNode->Value));

delete PNode;
```

```
struct Node
{
String Name;
int Value;
Node *Next;
        };
```

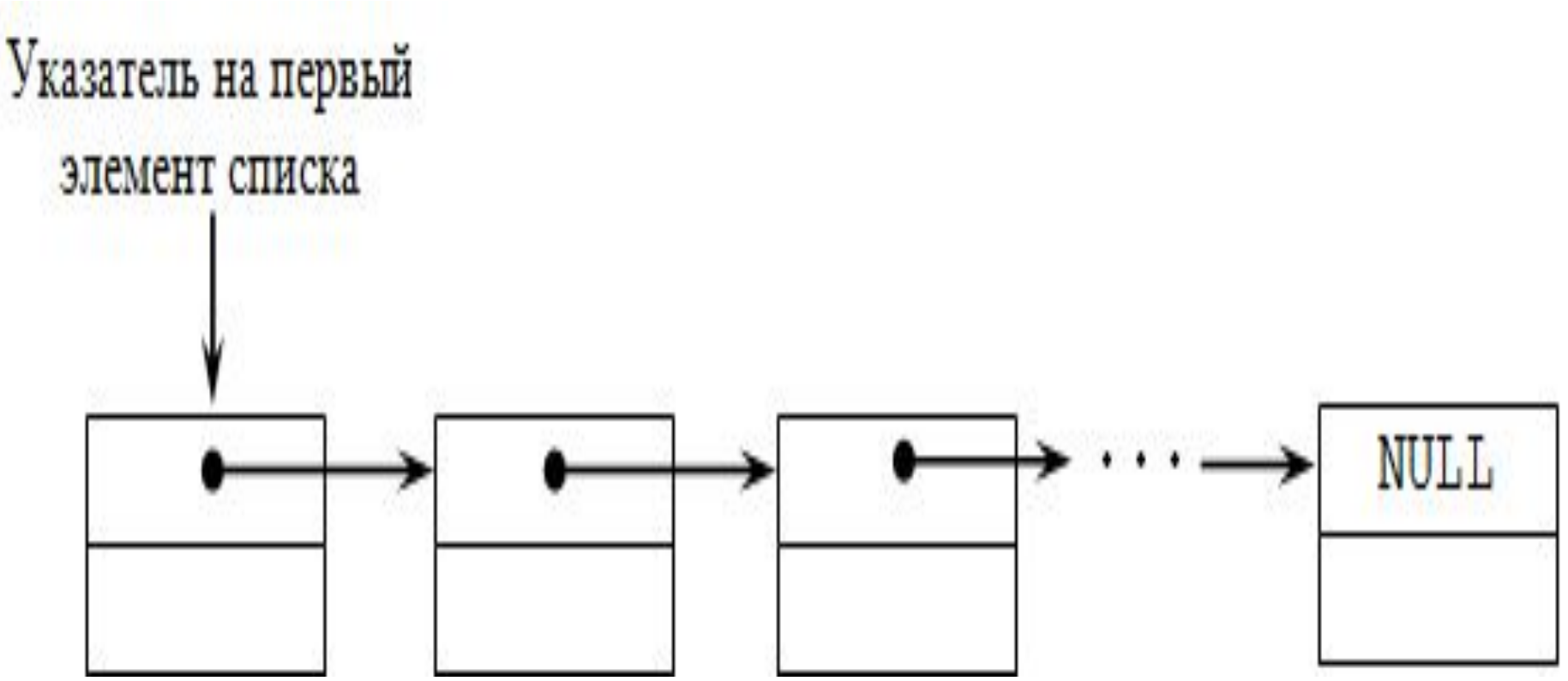
```
Node *PNode;
PNode = new Node;

PNode->Name= "STO";
PNode->Value = 28;
PNode->Next = NULL;

Memo1->Lines->Add("name="+(PNode->Name)+
" value="+(PNode->Value));
delete PNode;
```

Списки

Однонаправленный (односвязный) список



```
struct имя_типа {
 информационное поле;
адресное поле;
 };
struct Node {
int key;
Node *pnext;
};
struct point {
char name[10];   //String name;
 int age;
point *pnext; };
```

```
struct  List {//структура данных
   int  Data; //информационное поле
   List *Next; //адресное поле
 };
```

Создание однонаправленного списка

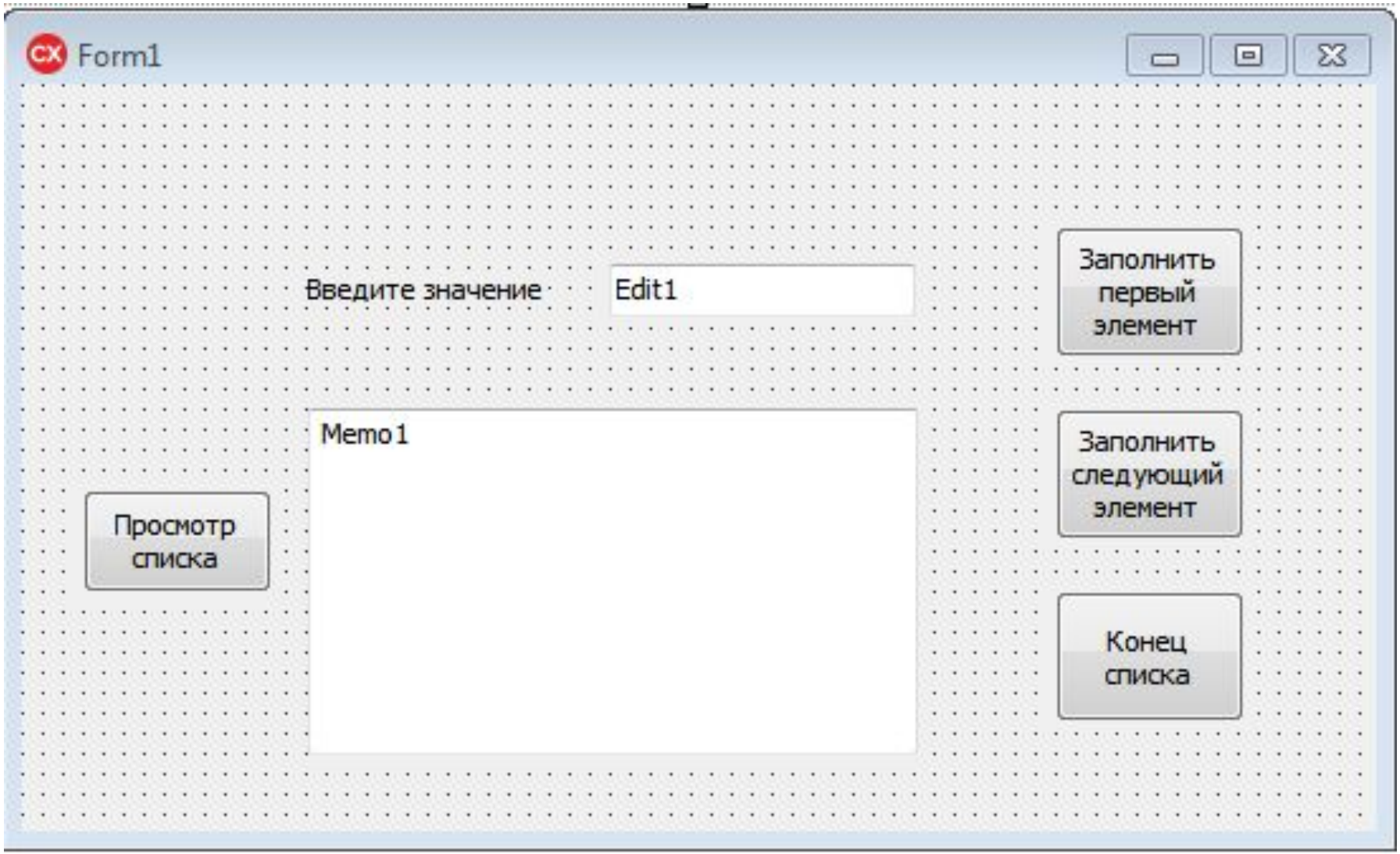
```
int n;
cout<<"vvedite kol-vo elementov spiska:”;
cin>>n;
//указатель на текущий элемент списка
List *head,
*first;//указатель на первый элемент
//выделяется память под первый элемент
head=new List;
//сохранили адрес первого элемента,
//чтобы потом обратиться к нему
first=head;
```

```
    //заполняем первый элемент
    cout << "Vvedite znachenie ";
    cin >>head->Data;
//заполняем остальные элементы списка
    for(int i=1;i<n;i++)
    { head->Next= new List;
    head=head->Next;
    cout << "Vvedite znachenie ";
    cin >>head->Data;
    }
    //определяем конец списка для
    //последнего элемента
    head->Next=NULL;
```

просмотр однонаправленного списка

```
cout<<"elementi spiska:\n";
head=first;
//пока не встретится признак
//конца списка NULL
    while(head != NULL)
{ //вывод на экран значения
//информационного поля
cout << head->Data << "\t";
//переход к следующему элементу
    head=head->Next;
  }  cout << "\n";
```

В визуальной среде



```
TForm1 *Form1;

struct  List {//структура данных
int  Data; //информационное поле
List *Next; //адресное поле
 };

List *head,//указатель на текущий элемент списка
     *first;//указатель на первый элемент списка
```

Обработчик события нажатие на кнопку «Заполнить первый элемент»

```
 head=new List;
//сохранили адрес первого элемента,
//чтобы потом обратиться к нему
first=head;

//заполняем первый элемент
head->Data=StrToInt(Edit1->Text);
```

Обработчик события нажатие на кнопку «Заполнить следующий элемент»

```
head->Next= new List;
head=head->Next;
head->Data=StrToInt(Edit1->Text);
```

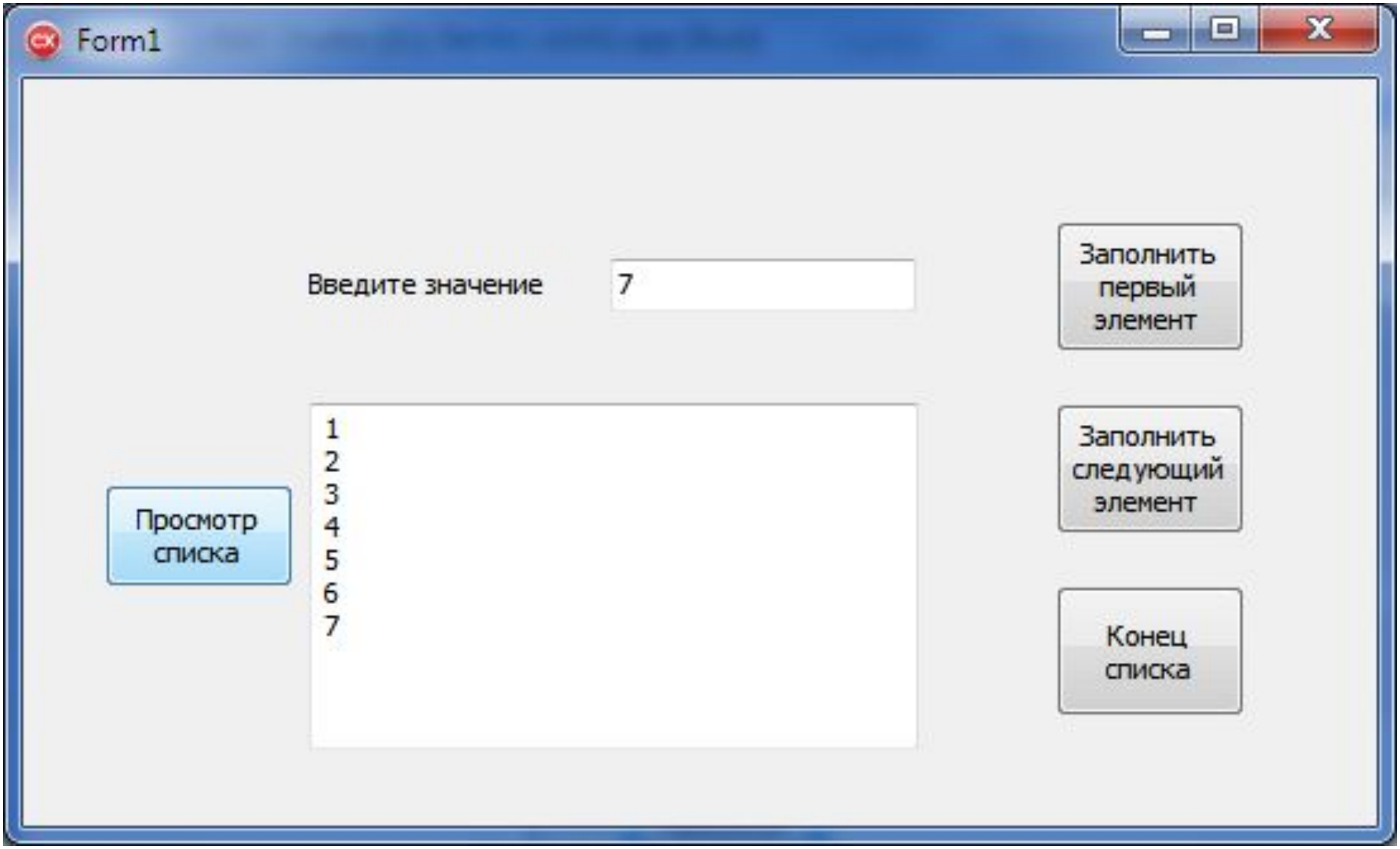
Обработчик события нажатие на кнопку «Конец списка»

```
 //определяем конец списка для
//последнего элемента
    head->Next=NULL;
```

Обработчик события нажатие на кнопку

«Просмотр списка»

```
Memo1->Clear();
head=first;
//пока не встретится признак конца списка NULL
    while(head != NULL)
{
//вывод на экран значения информационного поля
Memo1->Lines->Add(head->Data);
head=head->Next;
  }
```

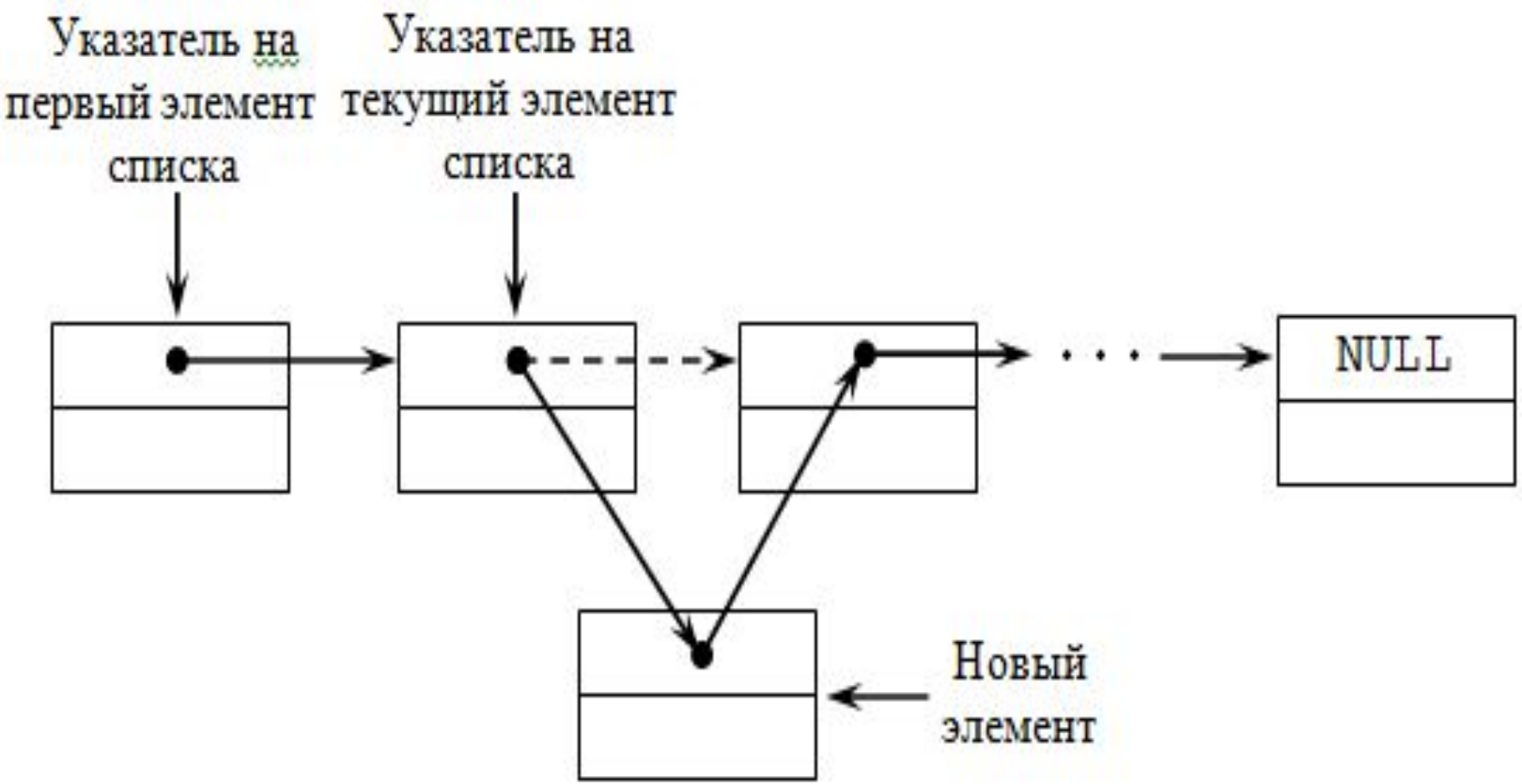

Form1
Введите значение
7
Заполнить первый элемент
Заполнить следующий элемент
Конец списка
Просмотр списка
1
2
3
4
5
6
7

```
void PrintM(List *head) {
//head=first;
//пока не встретится признак конца списка NULL
    while(head != NULL) {
Form1->Memo1->Lines->Add(head->Data);
head=head->Next;
  }
}
```

Обработчик события кнопки
«Просмотр списка»

```
 Memo1->Clear();
 PrintM(first);
```

Вставка элемента в однонаправленный список



```
/*функция добавления элемента с заданным номером в однонаправленный список*/
List* InsertM(List* Head,
int Number,          //номер добавляемого элемента списка
int DataItem  ) {  //значение добавляемого элемента списка
Number--; //значение номер уменьшается на 1
//выделяется память для вспомогательного списка
    List *NewItem=new(List);
//в поле дата вспомогательного списка
//заносится значение добавляемого элемента списка
    NewItem->Data=DataItem;
// признак конца списка NULL в адресное поле
    NewItem->Next = NULL;
//если список пуст
if (Head == NULL) {
    Head = NewItem; //создаем первый элемент списка
  }
```

```
else {
//список не пуст
    List *Current=Head;
    for(int i=1; i < Number && Current->Next!=NULL; i++)
     Current=Current->Next;
     if (Number == 0) {
//вставляем новый элемент на первое место
       NewItem->Next = Head;
       Head = NewItem;
     }
else {//вставляем новый элемент на не первое место
      if (Current->Next != NULL)
        NewItem->Next = Current->Next;
      Current->Next = NewItem;
     }
  }  return Head;}
```

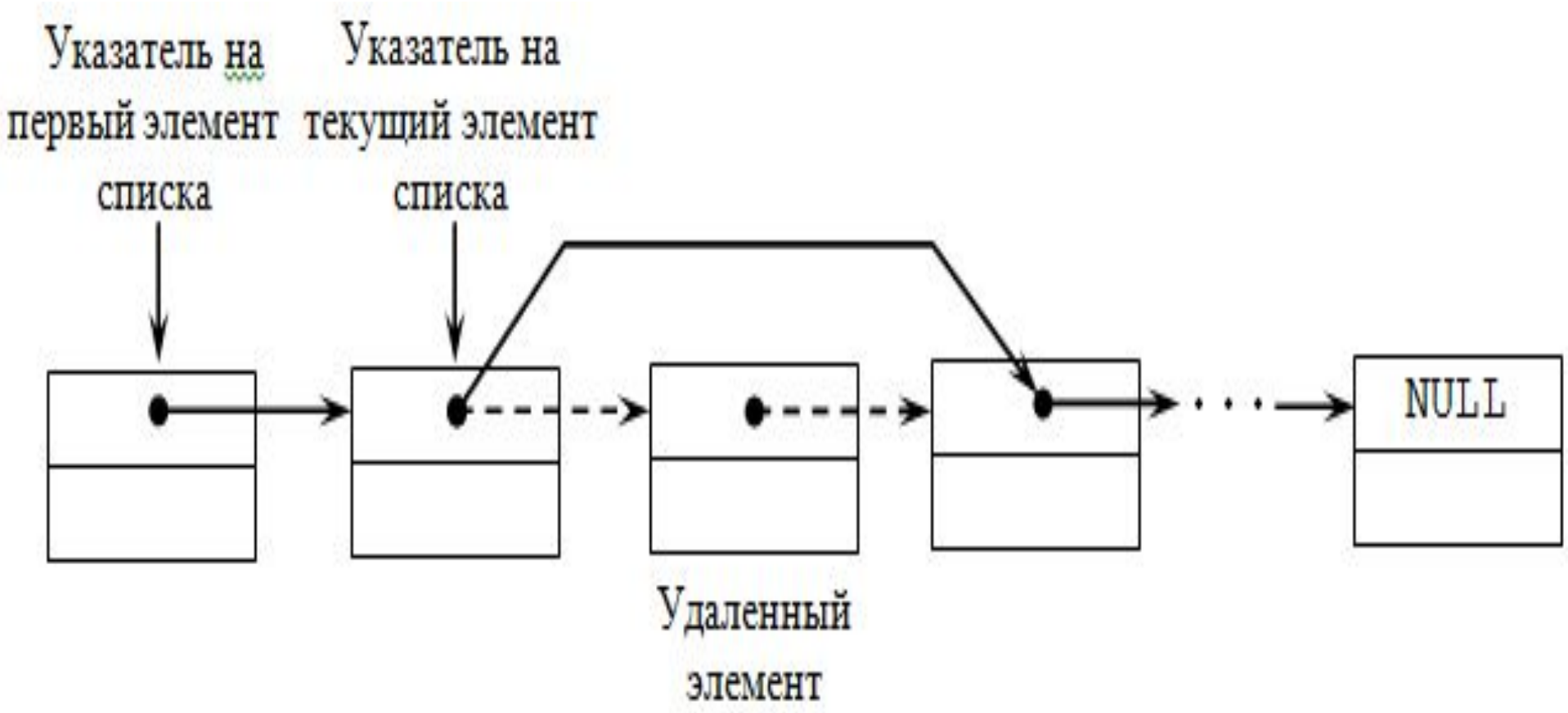
Обработчик события кнопки

«Добавить элемент»

```
 int Numb, DataIt;
 Numb=StrToInt(Edit3->Text);
 DataIt=StrToInt(Edit1->Text);
//добавляем элемент, в качестве аргумента функции используем сохраненный
//указатель на первый элемент first
head=InsertM(first, Numb, DataIt);
Memo1->Clear();

 PrintM(head);
```

Удаление элемента из однонаправленного списка



```
/*удаление элемента с заданным номером из однонаправленного списка*/
List* DeleteM(List* Head,   int Number){
  List *ptr;//вспомогательный указатель
  List *Current = Head;
  for (int i = 1; i < Number && Current != NULL; i++)
    Current = Current->Next;
  if (Current != NULL){//проверка на корректность
    if (Current == Head){//удаляем первый элемент
      Head = Head->Next;
      delete(Current);
      Current = Head;
    }
```

```
else {//удаляем не первый элемент
        ptr = Head;
        while (ptr->Next != Current)
           ptr = ptr->Next;
        ptr->Next = Current->Next;
        delete(Current);
        Current=ptr;
      }
   }
   return Head;
}
```

Обработчик события нажатие на кнопку «Удалить элемент»

```
  int Numb;
 Numb=StrToInt(Edit3->Text);

head=DeleteM(head,Numb);
 Memo1->Clear();

 PrintM(head);
```

/*функция поиска элемента с заданным значением в однонаправленном списке результат работы функции: true − искомый элемент имеется, false − искомый элемент отсутствует*/

```
bool FindM (List* Head, int DataItem){
//вспомогательный указатель
  List *ptr;
  ptr = Head;
//пока не конец списка
  while (ptr != NULL){
//последовательное сравнение элементов списка с заданным значением
    if (DataItem == ptr->Data) return true;
//значение найдено
    else ptr = ptr->Next; // переход к следующему
  }
  return false;  }   //значение не найдено
```

//функция удаления однонаправленного списка

```
void Delete_ListM(List* Head){
  if (Head != NULL){
     Delete_ListM(Head->Next);
     delete Head;
  }
}
```

Пример класса для работы с односвязным списком

```
struct element {

//значения из x
//будут передаваться в список
//сюда добавить другие поля вашей структуры
int x;
//Адресное поле
element *Next;
};
```

```
class List //Класс Список
{
 element **Head, *First;
 public:
  List() {
  Head=new element;
  First=Head;
  Head->x=1;} //Конструктор и инициализация
                  //первого элемента списка
 ~List(); //Деструктор. определен вне класса
//Функция для добавления значений в список
void Add(int x);
//Функция для отображения списка на экране
void Show();
};
```

```
List::~List() //Деструктор определен вне класса
{
//Пока по адресу не пусто
    while (Head!=NULL)      {
//Временная переменная для хранения адреса
//следующего элемента
        element *temp=Head->Next;
//Освобождаем адрес обозначающий начало
delete Head;
//Меняем адрес на следующий

        Head=temp;
}
}
```

```
//Функция добавления элементов в список
//после x указываются все заполняемые поля
//структуры
void List::Add(int x) {
//При каждом вызове выделяется память
 Head->Next= new element;
Head=Head->Next;
Head->x=x;
}
```

```
//Функция отображения списка на экране
void List::Show() {
//Определяем указатель, который изначально
//равен адресу начала списка
  element *temp=First;
 //До тех пор пока не встретит пустое значение
while (temp!=NULL) {
//Выведет элемент x из списка в Memo1 на форме с
именем Form3 – указывайте свою форму
   Form3->Memo1->Lines->Add(temp->x);
//Указываем, что далее нам нужен следующий
//элемент
temp=temp->Next;
 }  }
```

```
//использование класса
 List ob;//Переменная, тип которой список объявляем
         //глобально
 . . . .
```

Обработчик события нажатие на кнопку

«Заполнить следующий элемент»

```
 //Считывание из визуальных компонентов
int x=StrToInt(Edit1->Text);
//Добавление элемента в список
ob.Add(x);
```

Обработчик события нажатие на кнопку

«Просмотр списка»

```
 Memo1->Clear();
ob.Show();
```

Можно использовать другие варианты для заполнения и отображения содержимого списка

см. devcpp_4.pdf

При описании класса в отдельном модуле не забудьте указать, если используете для заполнения визуальные компоненты или тип String

```
#include <vcl.h>
```