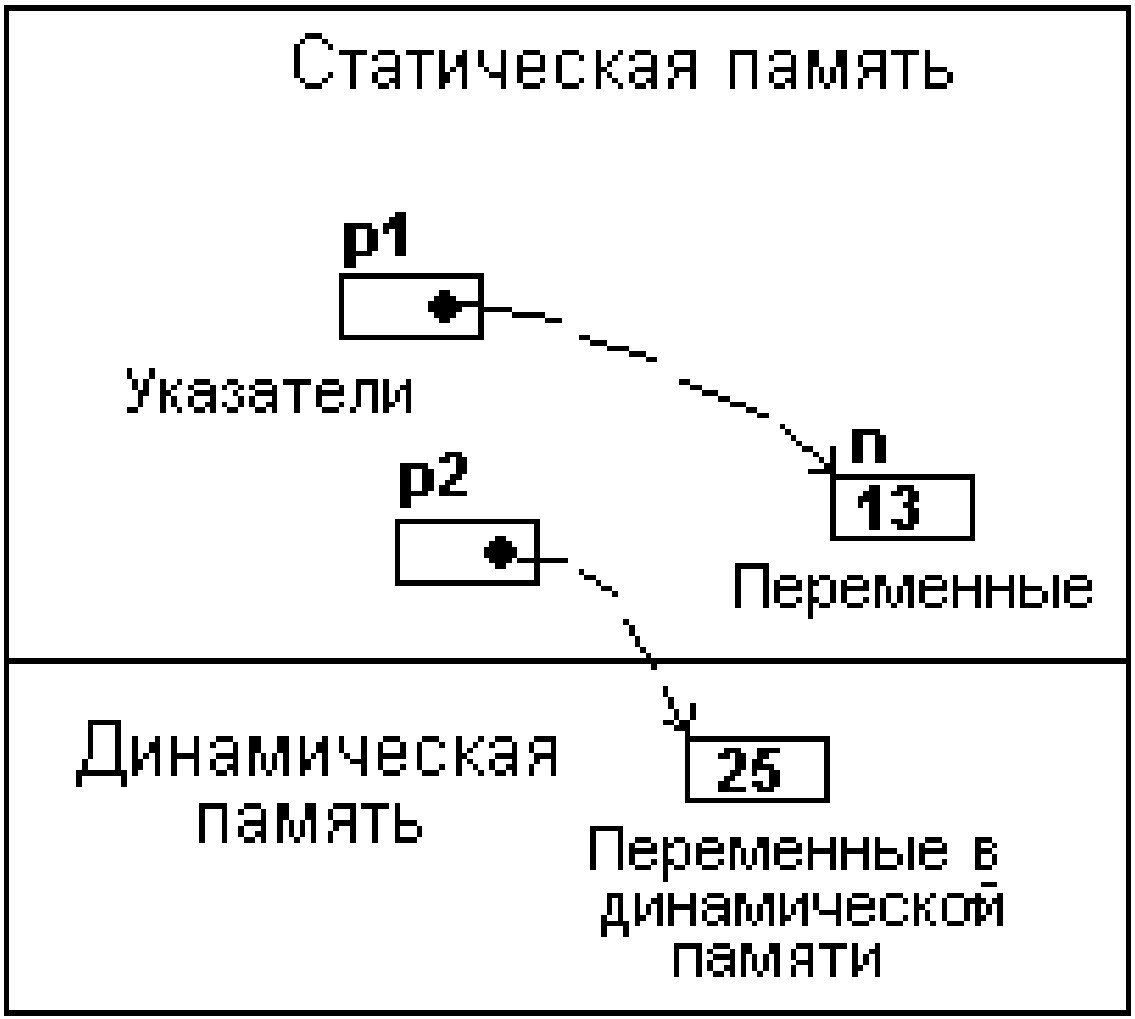
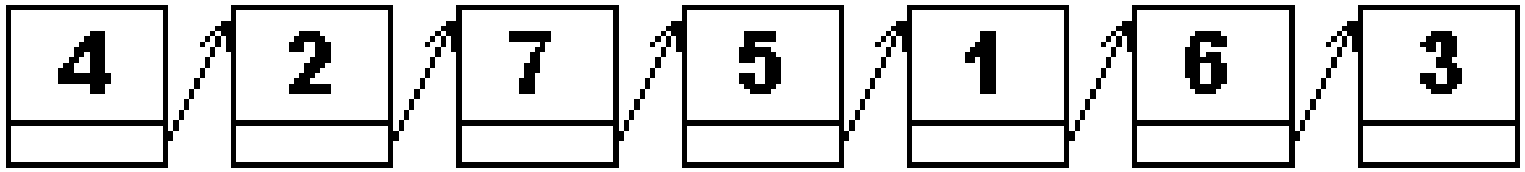
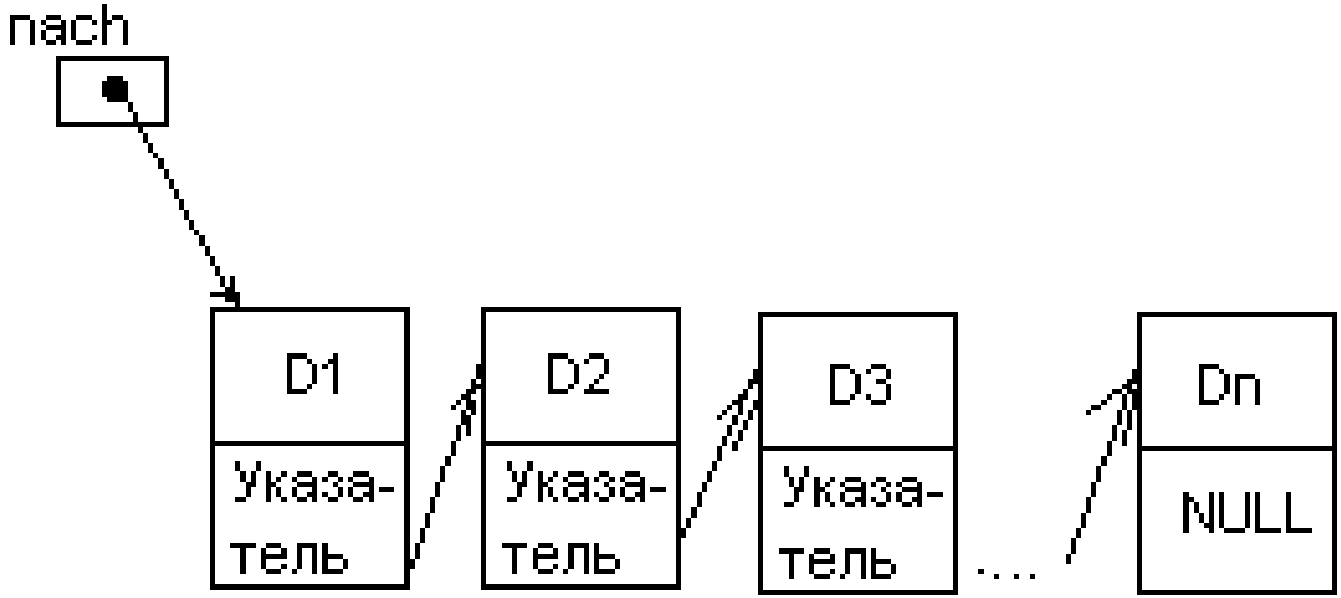
# Лекция 5.
# Линейный список
# в динамической памяти

# Статические и динамические переменные



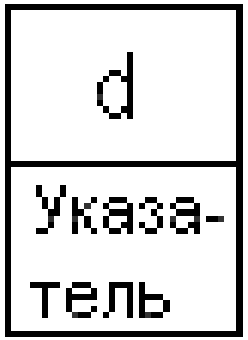
# Общее представление о списке в динамической памяти



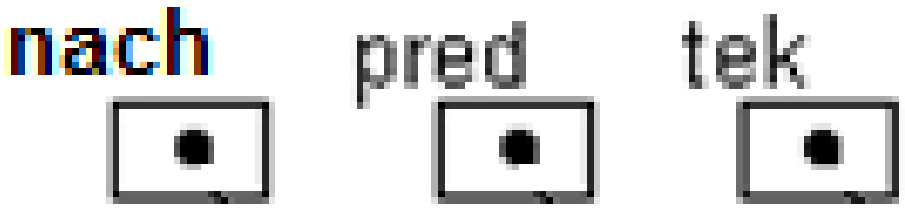
# Описание типа для построения линейного списка

```
struct  node
   {
     int d;
     struct node *link;
    };
```

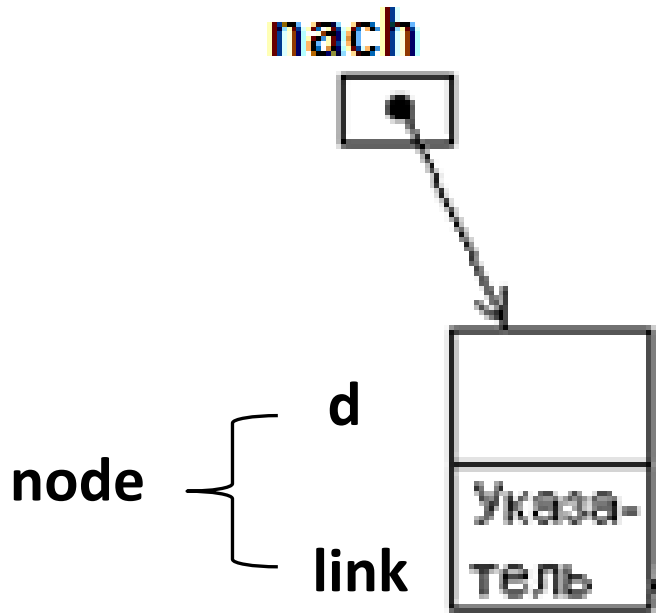


## Создание элементов в динамической памяти

**struct node *nach, *pred, *tek;**

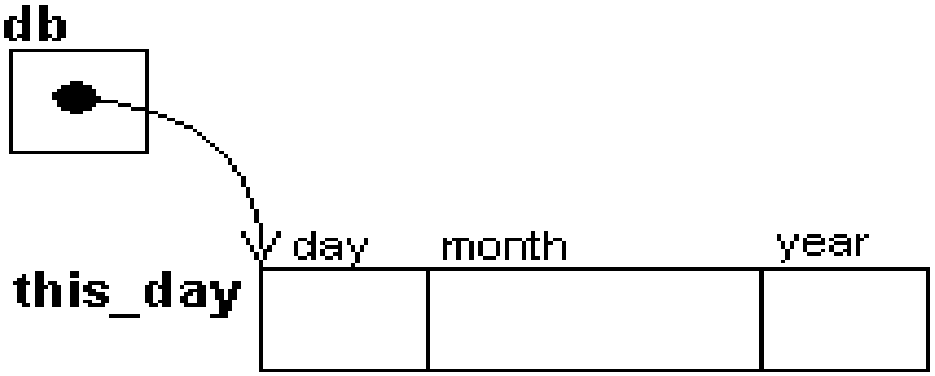



**first=(struct node *)malloc(sizeof(struct node));**



# Обращение к полям структуры через указатель (в статической памяти)

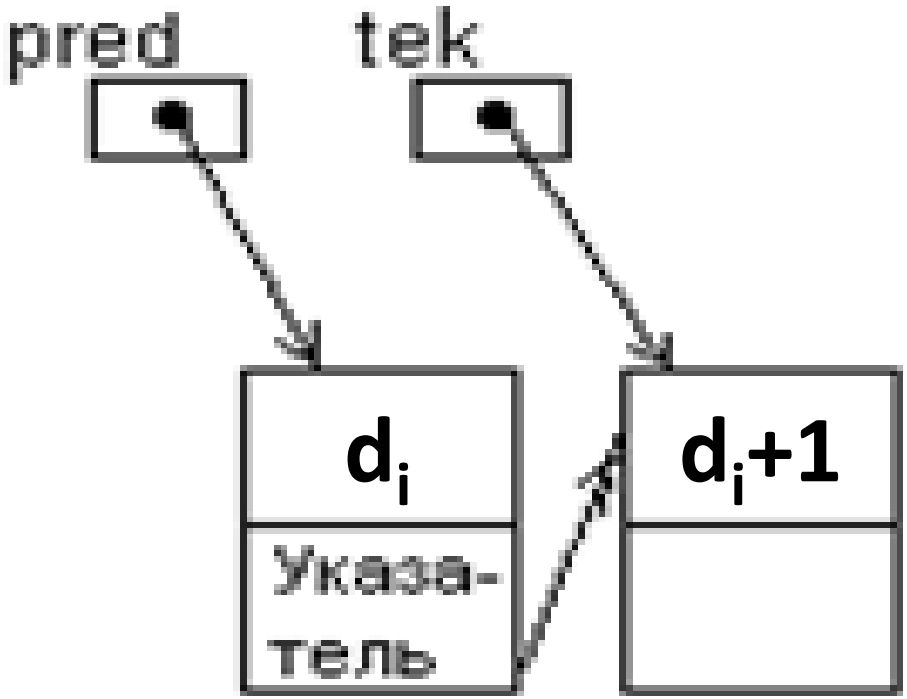
```
1.  struct date {
2.        int day;
3.        char month[10];
4.        int year;  };
5.  struct date this_day, *db;
6.  db = &this_day;
    . . .
7.  (*db).day = 25;
8.  db -> day = 25;
```



# Связывание элементов

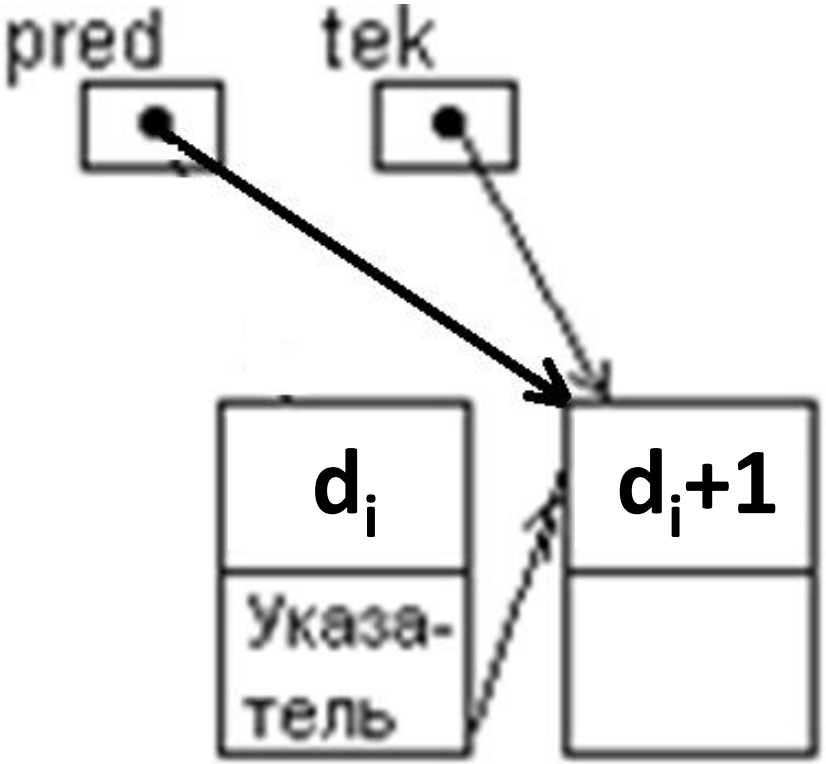
**pred -> link=tek;** // обращение к полю **link**

// структуры через указатель

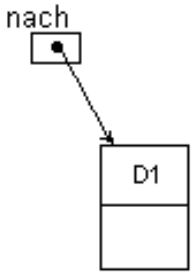
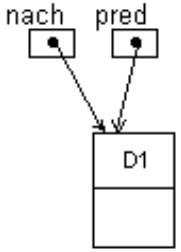
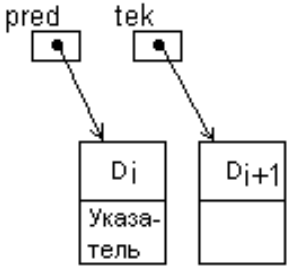
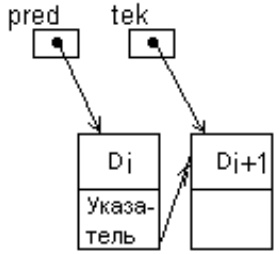


# Переприсваивание указателей

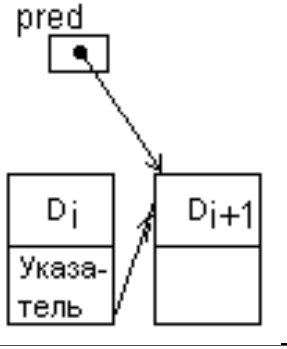
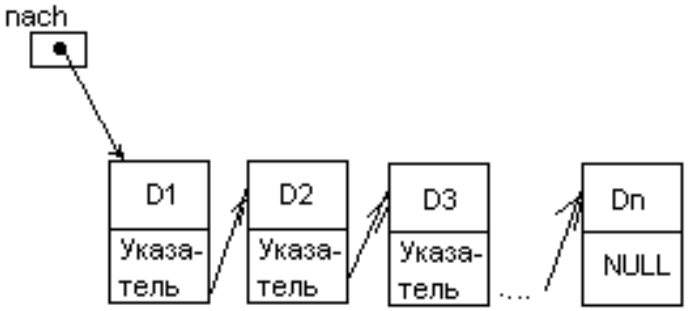
**pred=tek;** // pred и tek указывают на
// один и тот же элемент



# Алгоритм построения списка в прямом порядке

| | |
|---|---|
| 1. Создать в динамической памяти первый элемент nach;<br>Ввести nach->d; |  |
| 2. pred=nach; |  |
| 3. Создать текущий элемент tek:<br>Ввести tek->d; |  |
| 4. Установить связь предыдущего с текущим:<br>pred->link=tek; |  |

# Продолжение. Алгоритм построения списка в прямом порядке

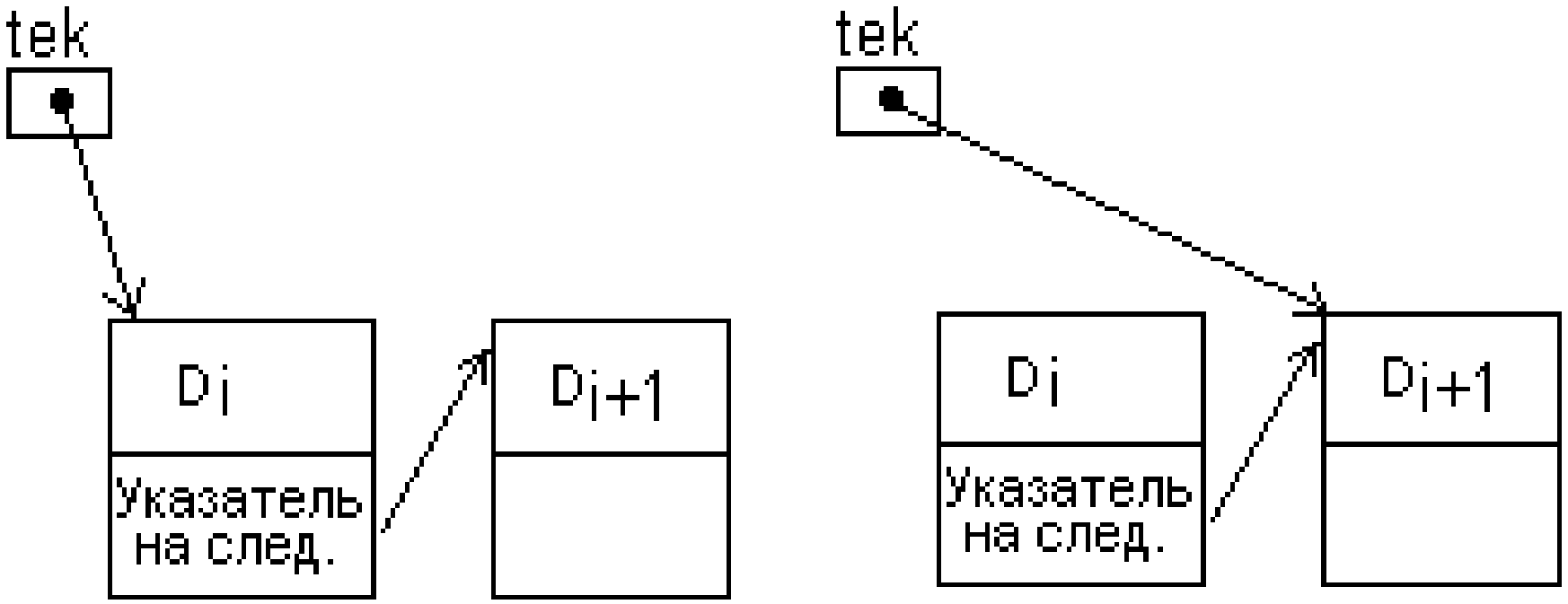
| | |
|---|---|
| 5. Текущий элемент становится предыдущим: pred=tek; |  |
| 6. Если не конец ввода то перейти на 3. | |
| 7. Обнулить адресную часть последнего элемента: tek->link=NULL; |  |

# Глобальные описания

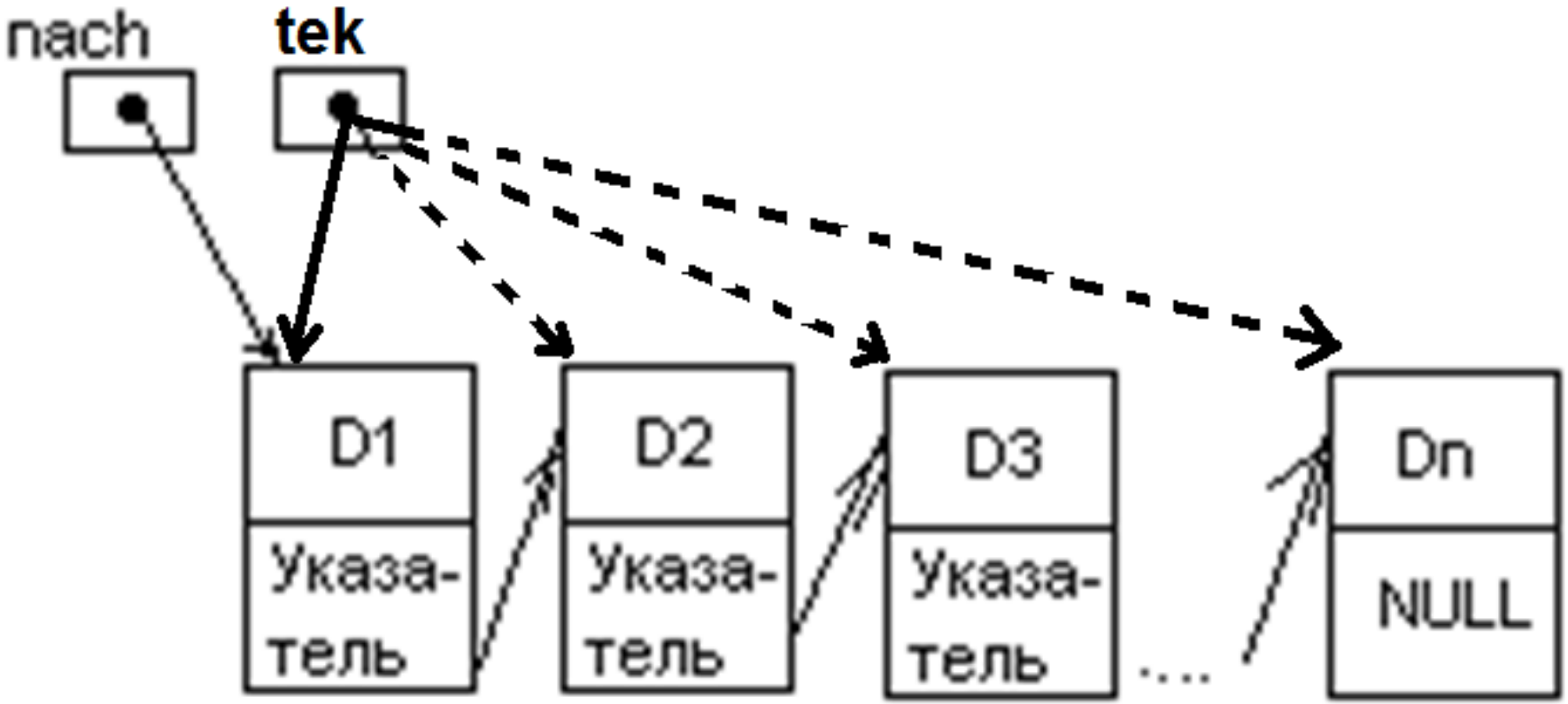
```
struct node
        {
            int d;
            struct node *link;
        };

 struct node *nach;    // Указатель на
                       // начало   списка
```

# Шаг по связи

**tek = tek->link;**



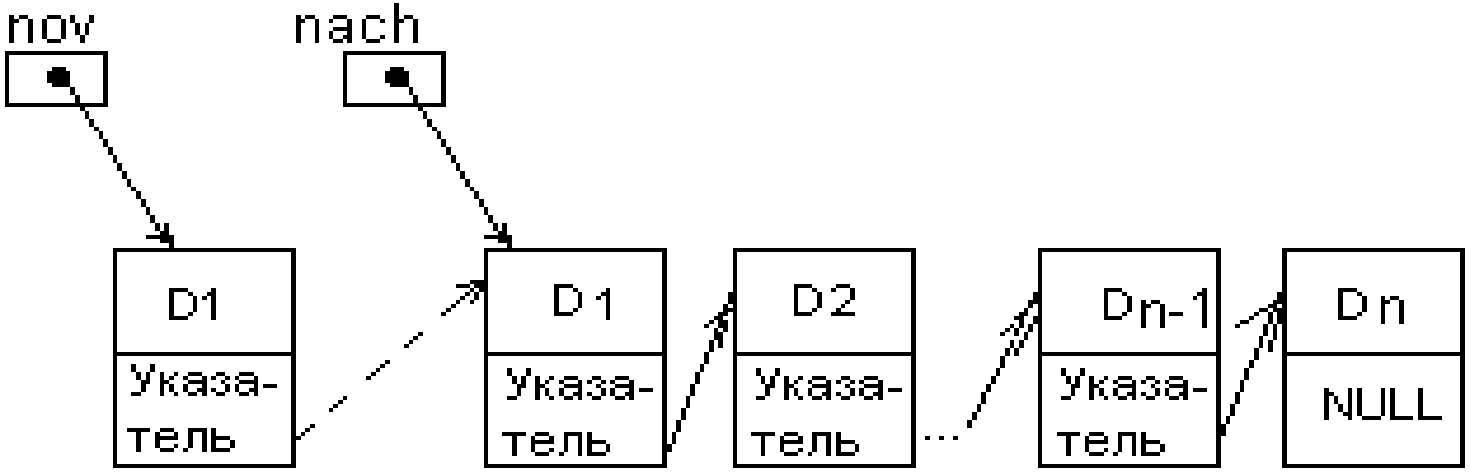
# Просмотр списка



# Функция просмотра списка

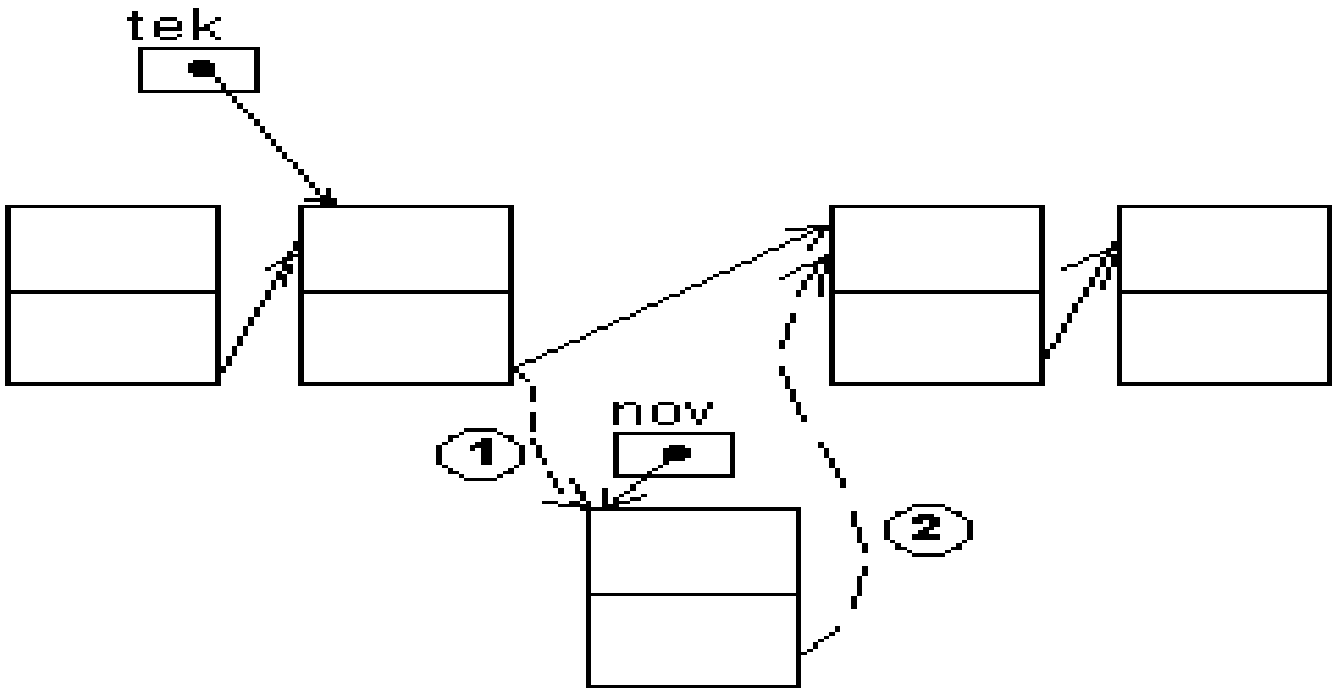
```
1.  void prosmotr()
2.   {
3.       struct node *tek;
4.       tek = nach;              // Встали на начало списка
5.       while(tek != NULL)       // Пока не конец списка
6.           {
7.              printf("%d", tek->d);
8.              tek = tek->link;        // Шаг по связи
9.           }
10.  }
```

# Добавление элемента в начало списка




1. Создать новый элемент **nov**.
   Ввести информационную часть **nov->d**.
2. Построить связь:
   **nov->link=nach;**
3. Новый элемент сделать начальным:
   **nach=nov;**

# Добавление элемента в середину списка



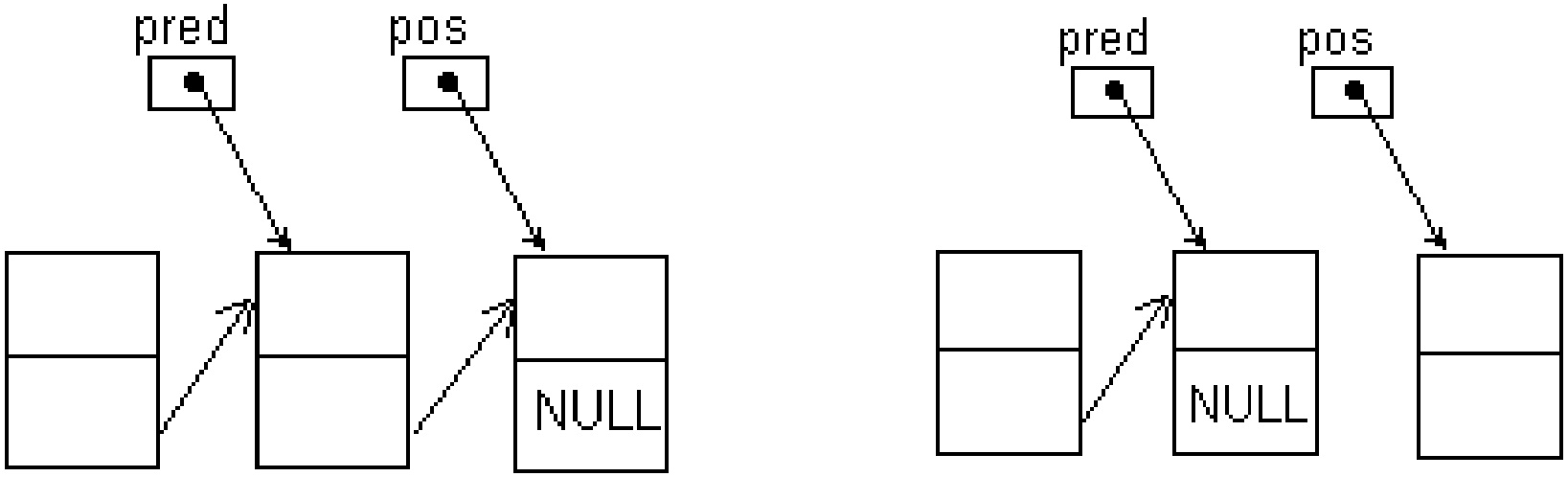

1. Заполнить связь 2 у элемента **nov**:

   **nov->link=tek->link;**

2. Соединить **tek** и **nov** (связь 1):

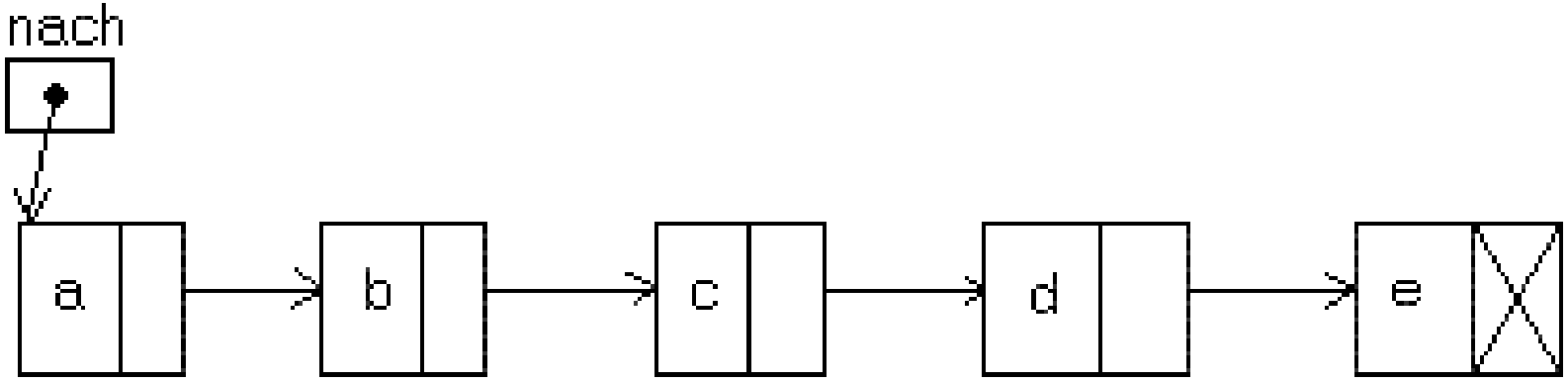
   **tek->link=nov;**

# Удаление последнего элемента



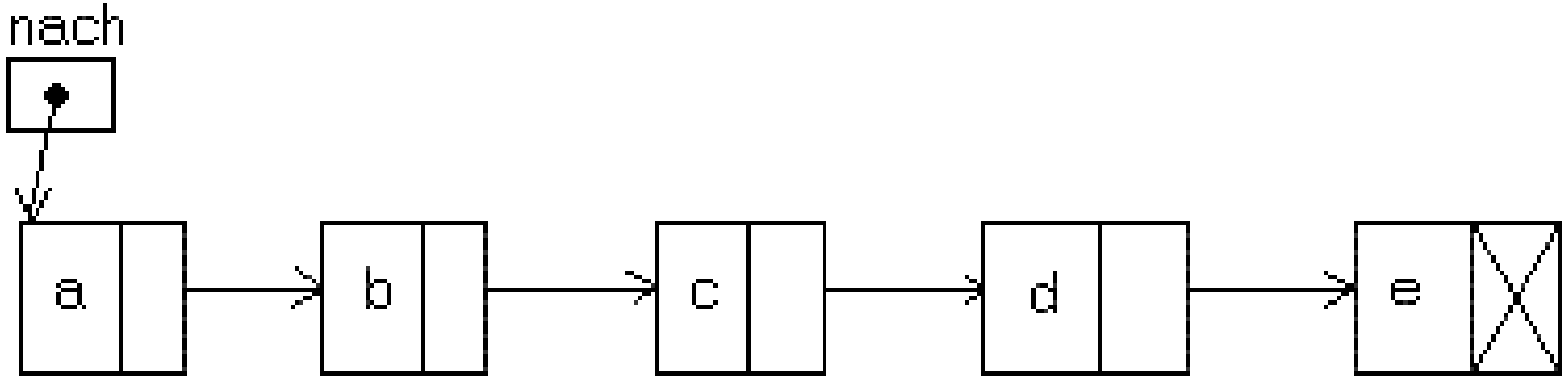

1. **pred->link=NULL;**
2. **free(pos);**

# Перемещение по списку




```
tek=nach;

while(tek->d != 'c')
        tek=tek->link;

printf("%s", tek->d);
```

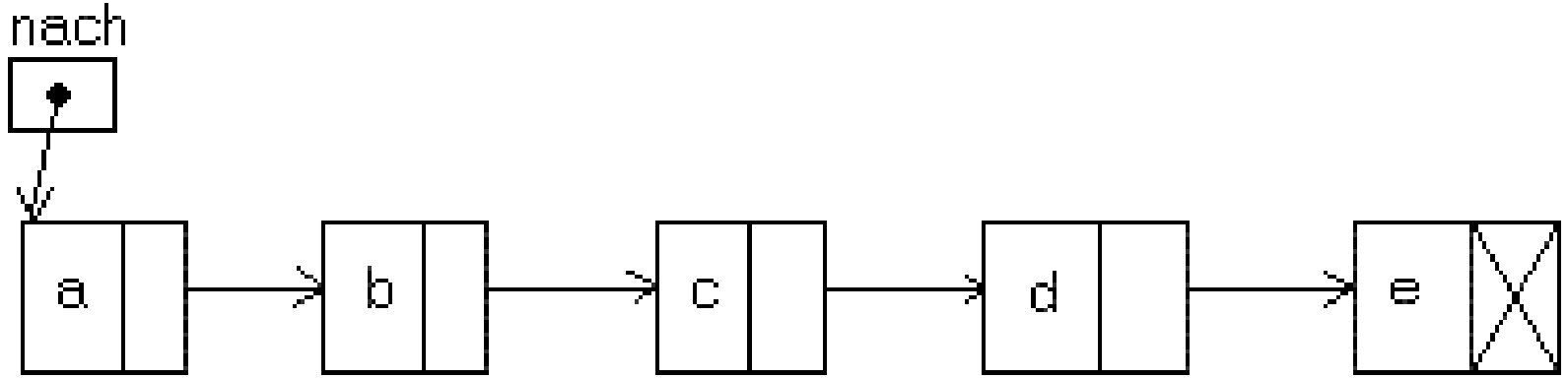
# Перемещение по списку




Что будет выведено на экран в результате выполнения фрагмента программы?

```
tek=nach;
for(i=1; i<=4; i++)
        tek=tek->link;
printf("%s", tek->d);
```

# Перемещение по списку




Что будет выведено на экран в результате выполнения фрагмента программы?

```
tek=nach;
while(tek->link != NULL)
        tek=tek->link;
printf("%s", tek->d);
```

## Динамическое выделение памяти в C++

**(**компилятор **g++**, файлы с расширением **.cpp )**

В Си работать с динамической памятью можно при помощи соответствующих функций распределения памяти (calloc, malloc, free), для чего необходимо подключить библиотеку malloc.h

C++ использует новые методы работы с динамической памятью при помощи операторов **new** и **delete**:

- **new** — для выделения памяти;
- **delete** — для освобождения памяти.

Оператор **new** используется в следующих формах:

**new тип;** — для переменных

**new тип[размер];** — для массивов

Память может быть распределена для одного объекта или для массива любого типа, в том числе типа, определенного пользователем. Результатом выполнения операции **new** будет указатель на отведенную память, или исключение std::bad_alloc в случае ошибки.

```
int *one;
double *array;
struct person *human;
…
one = new int;
array = new double[10];
human = new  person;
```

Память, отведенная в результате выполнения **new**, будет считаться распределенной до тех пор, пока не будет выполнена операция **delete**. Освобождение памяти связано с тем, как выделялась память – для одного элемента или для нескольких. В соответствии с этим существует и две формы применения **delete**:

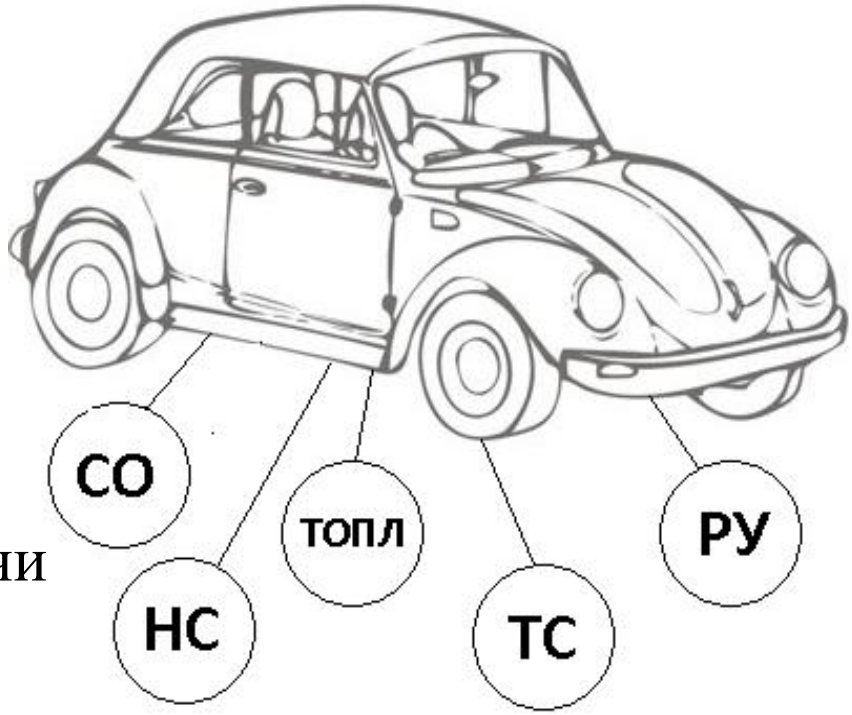
```
delete указатель;    // для одного элемента
delete[] указатель;  // для массивов
```

```
#include<iostream>
using namespace std;
struct   person{  char name[20];      int age;  };
int main() {
    int *one,   n;
    double *array;
    person *human;
    cout<<"Введите размер массива:"<<endl;
    cin>>n;
    one=new int;                //один элемент
    array=new double[n];   //массив элементов
    human=new person;     // один элемент типа person
   delete one, human;
   delete[] array;
   return 0;    }
```

# Объектно-ориентированное программирование (ООП)

Все программы состоят из кода и данных. ООП организует программу вокруг её данных (объектов) и наборов интерфейсов (методов) к этим данным.

**Абстракция. Автомобиль.**
Автомобиль - это программа (конечная цель) .




Состоит из нескольких подсистем: рулевое управление, тормозная система, система подачи топлива и т.д. (каждая система состоит из своих подсистем).
В целом же мы видим автомобиль.
Объекты подсистем взаимодействуют между собой и получается автомобиль.

# Объектно-ориентированное программирование

## Абстракция.Иерархия.

# Три принципа ООП

1. **Инкапсуляция** – механизм, который связывает код и данные, которыми он манипулирует, защищая оба этих компонента от внешнего вмешательства и злоупотреблений.

Доступ к коду и данным контролируется тщательно определенным интерфейсом. Каждый знает, как получить доступ к коду и может использовать код независимо от деталей реализации. Основой **инкапсуляции** является **класс**.

Класс определяет структуру и поведение (данные и код), которые будут совместно использоваться набором объектов.
Каждый объект данного класса содержит структуру и поведение, которые определены классом (копии переменных и методы).

При создании класса определяют данные и код (члены класса) - переменные и методы (функции) объекта.

# Три принципа ООП

Каждая переменная или метод могут быть *общедоступными* (**public**) или *приватными* (**private**).

Приватные (**private**)– доступны только для методов, которые являются членами класса.

Общедоступные (**public**) – доступны для всех методов вне класса (для внешних пользователей класса).

**Инкапсуляция** – это сокрытие реализации класса и отделение его внутреннего представления от внешнего.

# Три принципа ООП

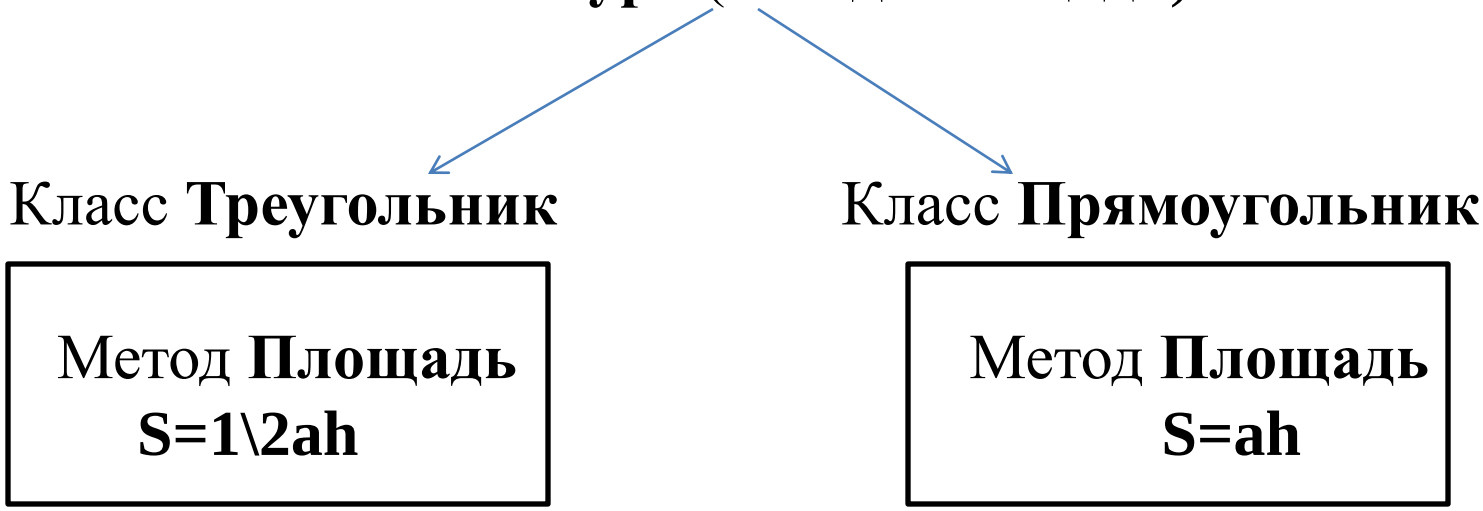
2. **Наследование** – процесс, посредством которого один объект получает свойства другого объекта.

   (отношения между классами, при которых класс использует структуру или поведение другого или других классов).

   Подклассы обычно переопределяют или дополняют унаследованную структуру и поведение.

3. **Полиморфизм** - («имеющий много форм») свойство, которое позволяет использовать один и тот же интерфейс для общего класса действий (один интерфейс – несколько методов (переопределение и перегрузка методов)).

# Три принципа ООП




Один и тот же метод (имя) работает по-разному в разных подклассах одного суперкласса – это полиморфизм.

# Три принципа ООП

Объектно-ориентированный подход способствует:

1. уменьшению сложности программного обеспечения;
2. повышению надежности программного обеспечения;
3. обеспечению возможности модификации отдельных компонентов без изменения остальных;
4. обеспечению возможности повторного использования отдельных компонентов ПО.

# Классы

**Класс** определяет новый тип данных. Этот тип можно использовать для создания объектов.

**Класс** – это шаблон для создания объекта.
**Объект** – экземпляр класса.

**Класс** определяет структуру объекта (переменные) и его методы, образующие функциональный интерфейс (функции для работы с переменными класса).

**Класс** – это логическая конструкция,
**объект** – физическое воплощение, участок памяти.