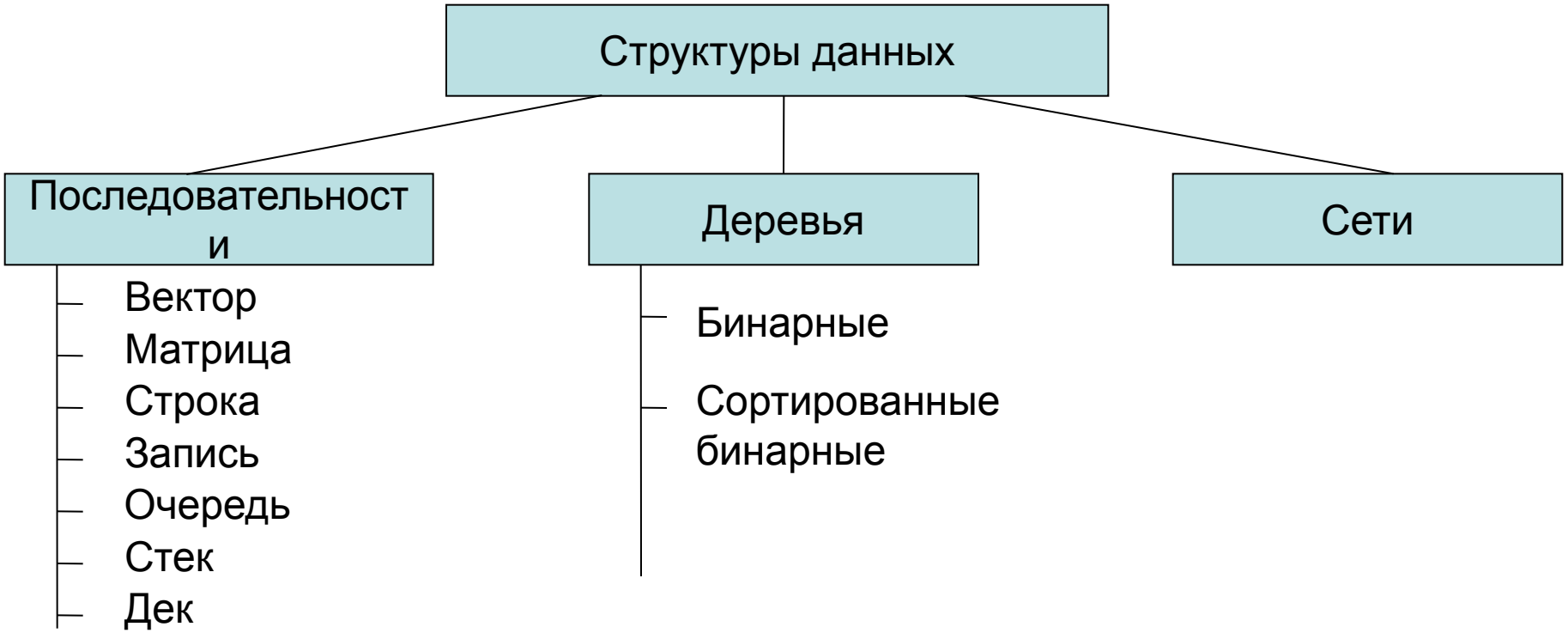
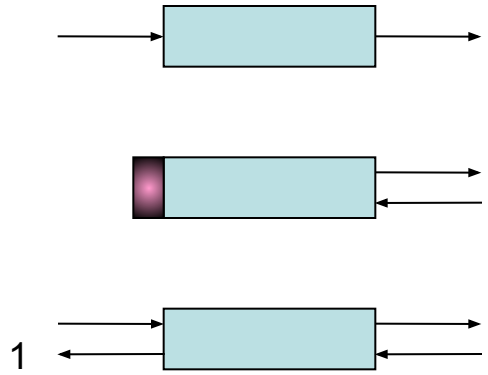
10 Динамические структуры данных



Динамические линейные структуры

1. Очередь – структура данных, реализующая: добавление – в конец, а удаление – из начала.
2. Стек – структура данных, реализующая: добавление и удаление с одной стороны.
3. Дек – структура данных, реализующая: добавление и удаление с двух сторон.



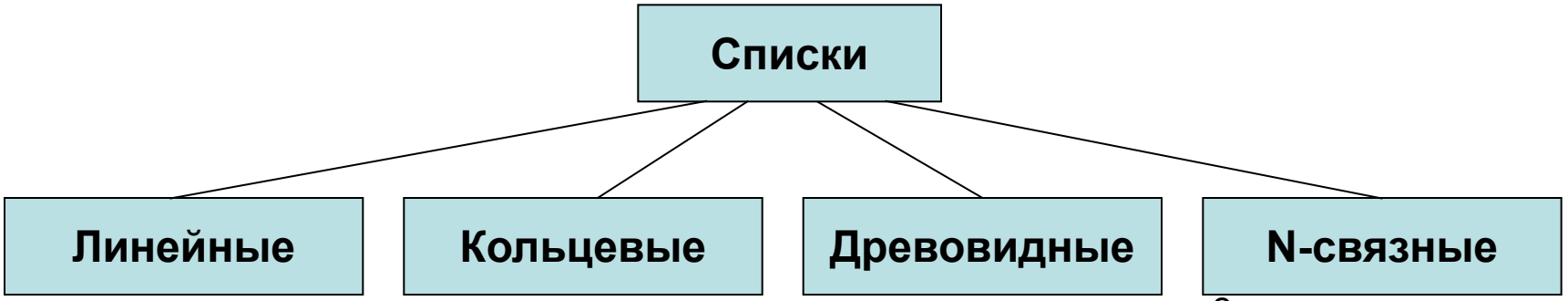
10.1 Списки

Список – способ организации данных, предполагающий использование указателей для определения следующего элемента.

Элемент списка состоит из двух частей: *информационной* и *адресной*.

Информационная часть содержит поля данных.

Адресная – включает от одного до n указателей, содержащих адреса следующих элементов. Количество связей, между соседними элементами списка определяет его связность: односвязные, двусвязные, n-связные.

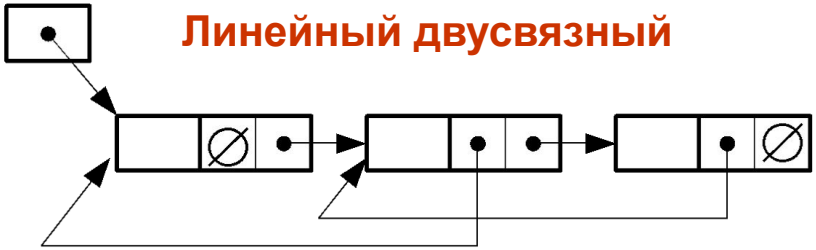


Виды списков

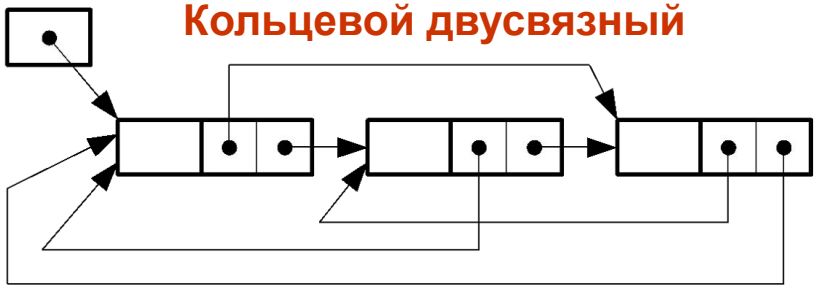
Линейный односвязный

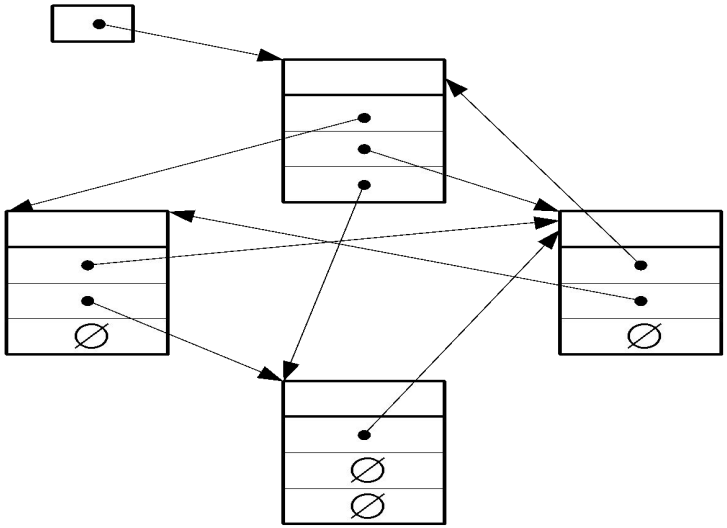
Кольцевой односвязный

Линейный двусвязный



Кольцевой двусвязный





Сетевой n-связный

Примеры описания элементов списка

Односвяный список:

```
struct element {                    {тип указателя}
       char  name[16];              {информационное поле 1}
       char  telefon[7];            {информационное поле 2}
        element *p;                 {адресное поле}
        };
```

Двусвяный список:

```
struct element {                    {тип указателя}
       char  name[16];              {информационное поле 1}
       char  telefon[7];            {информационное поле 2}
        element *prev;              {адресное поле «предыдущий»}
        element *next;              {адресное поле «следующий»}
        };
```

10.2 Односвязные списки

Аналогично одномерным массивам односвязные списки реализуют последовательность элементов. Однако в отличие от одномерных массивов позволяют:

- работать с произвольным количеством элементов, добавляя и удаляя их по мере необходимости;
- осуществлять вставку и удаление записей, не перемещая остальных элементов последовательности;

но

- не допускают прямого обращения к элементу по индексу;
- требуют больше памяти для размещения.

Основные приемы работы

Описание элемента списка:

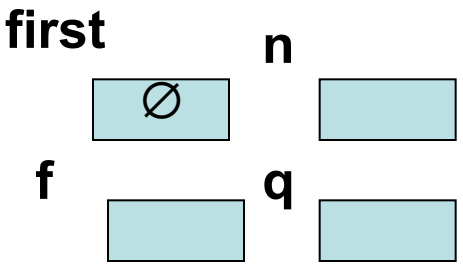
```
struct element      {тип на элемента}
{int num;           {число}
   element *p;      {указатель на следующий элемент}
  };
```

Описание переменной – указателя списка и нескольких переменных-указателей в статической памяти:

```
element * first,        {адрес первого элемента}
    *n,*f,*q;           {вспомогательные указатели}
```

Исходное состояние – «список пуст»:

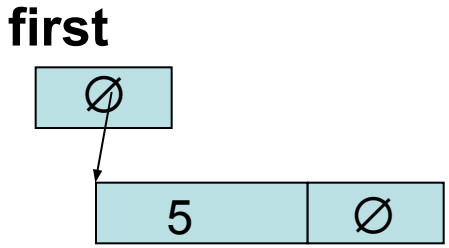
```
first=NULL;
```



Основные приемы работы (2)

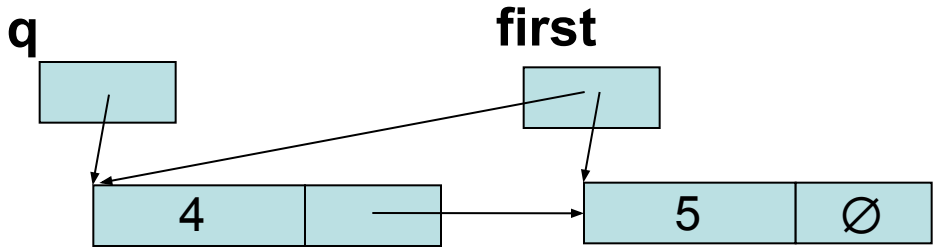
1 Добавление элемента к пустому списку:

```
first=new element;
first ->num=5;
first->p=NULL;
```



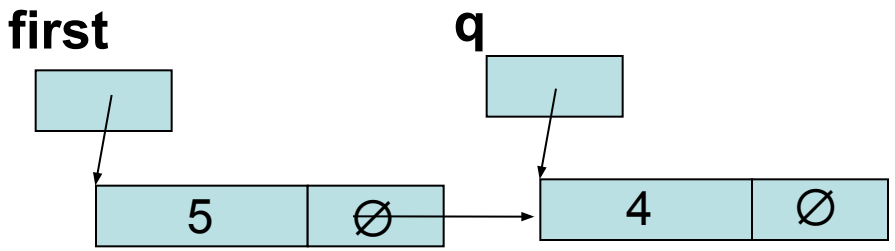
2 Добавление элемента перед первым (по типу стека):

```
q=new element;
q->num=4;
q->p=first;
first=q;
```



3 Добавление элемента после первого (по типу очереди):

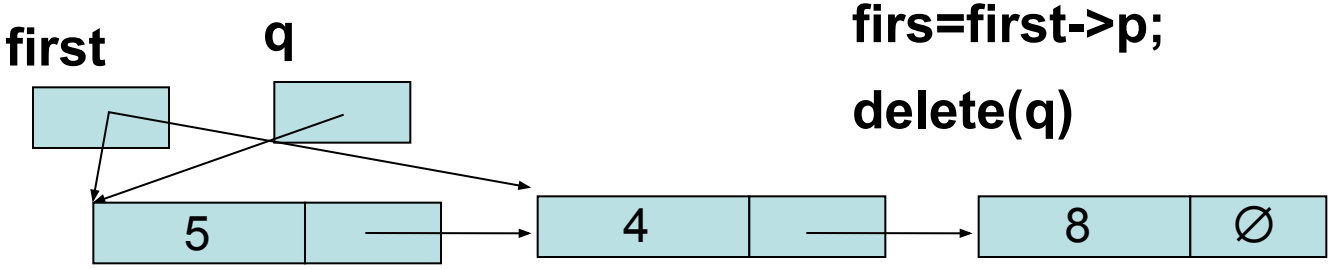
```
q=new element;
q->num=4;
q->p=NULL;
first->p=q;
```



Варианты удаления элементов

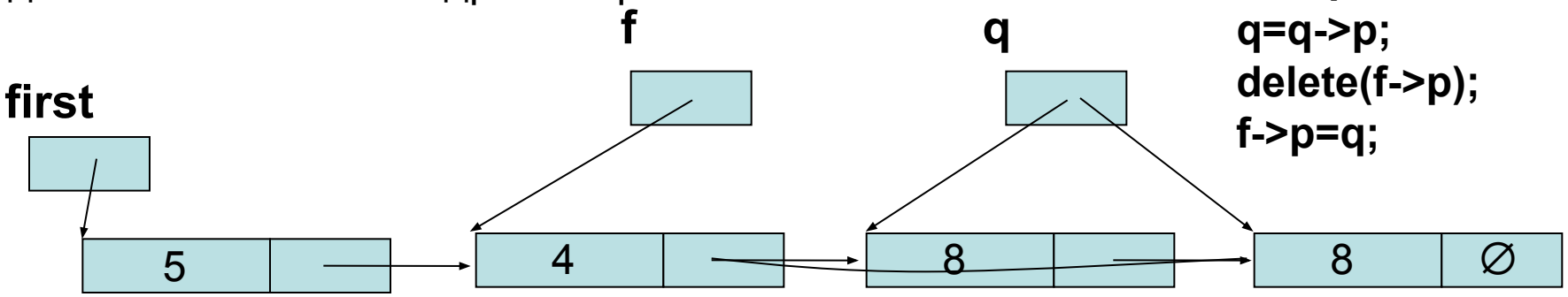
1. Удаление первого элемента

```
q=first;
firs=first->p;
delete(q)
```



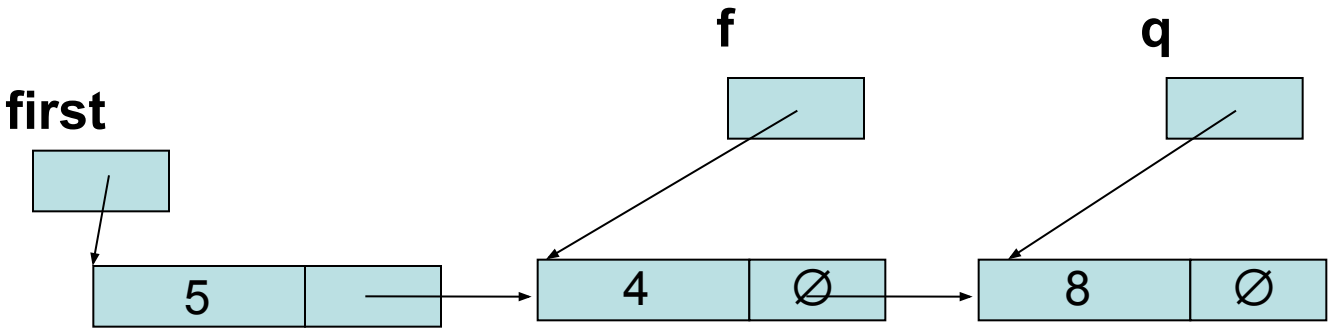
2. Удаление элемента с адресом q

```
f=first;
while(f->p!=q)
f=f->p;
q=q->p;
delete(f->p);
f->p=q;
```



3. Удаление последнего элемента

```
f=q=first;
while(q->p!=NULL)
{f=q; q=q->p;}
f->p=NULL;
delete(q);
```

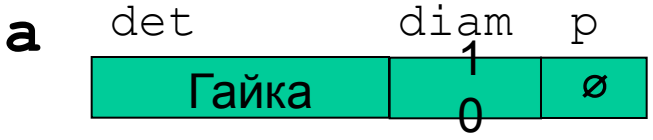


Динамические структуры данных (Ex10_1)

Пример. Стек записей.

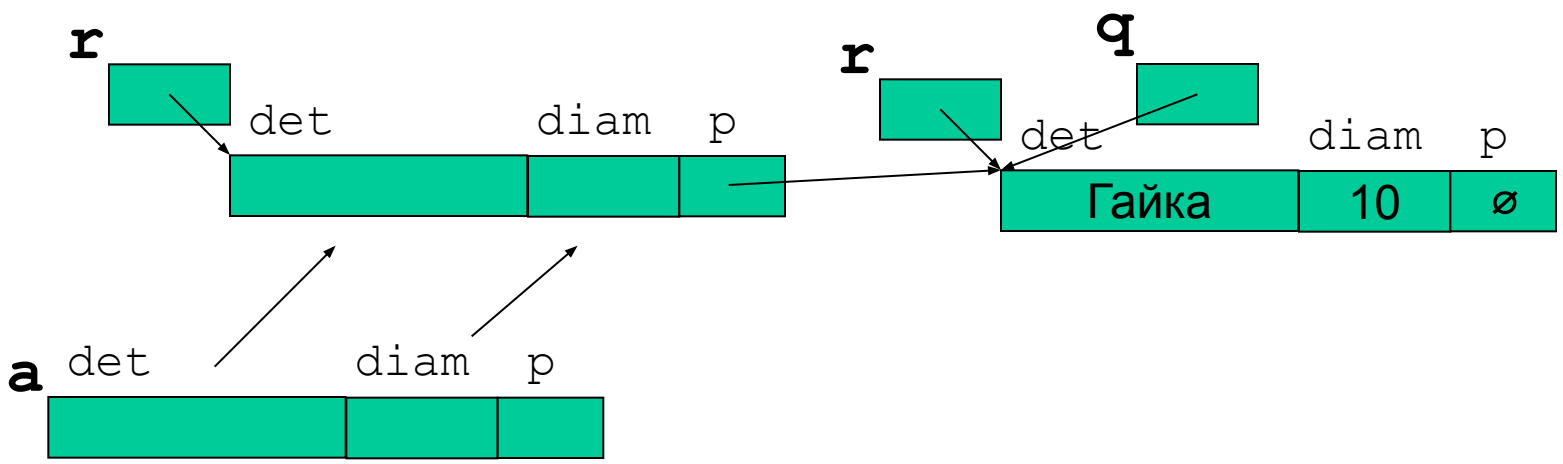
Написать программу, которая формирует список деталей, содержащих наименование детали и ее диаметр. Удалить из списка все детали с диаметром, меньшим 1.

```
#include "stdafx.h"
#include <stdio.h>
#include <string.h>
struct zap  {  char det[10];  float diam; zap *p;  };
int main(int argc, char* argv[])
{     zap a,*r,*q,*f;
      r=new zap;
      r->p=NULL;
      puts("Input strings");
      scanf("%s %f\n",r->det,&r->diam);
```



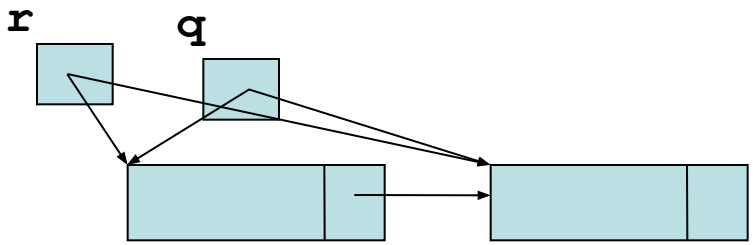
Динамические структуры данных (2)

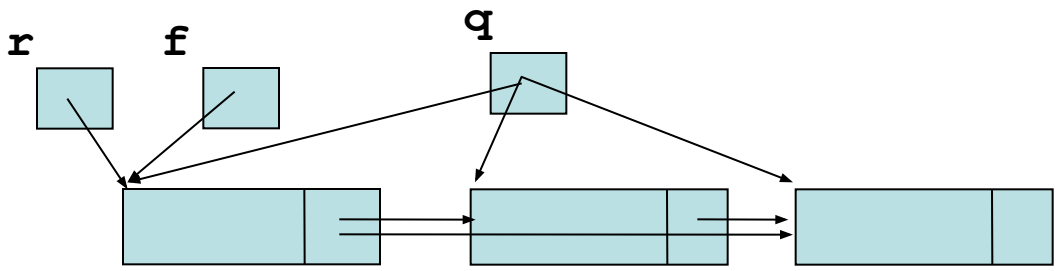
```
while((scanf("\n%s",a.det)),strcmp(a.det,"end")!=0)
        { scanf("%f",&a.diam);
           q=r;
           r=new zap;
           strcpy(r->det,a.det);
           r->diam=a.diam;
           r->p=q;
        }
```



Динамические структуры (3)
Удаление записей

```
q=r;
do
  {if (q->diam<1)
    if( q==r){
     r=r->p;
         delete(q);
 q=r;
  }
    else
       {q=q->p;
        delete(f->p);
        f->p:=q
       }
  else
    {f=q;
     q=q->p;
   }
}while(q!=NULL);
```

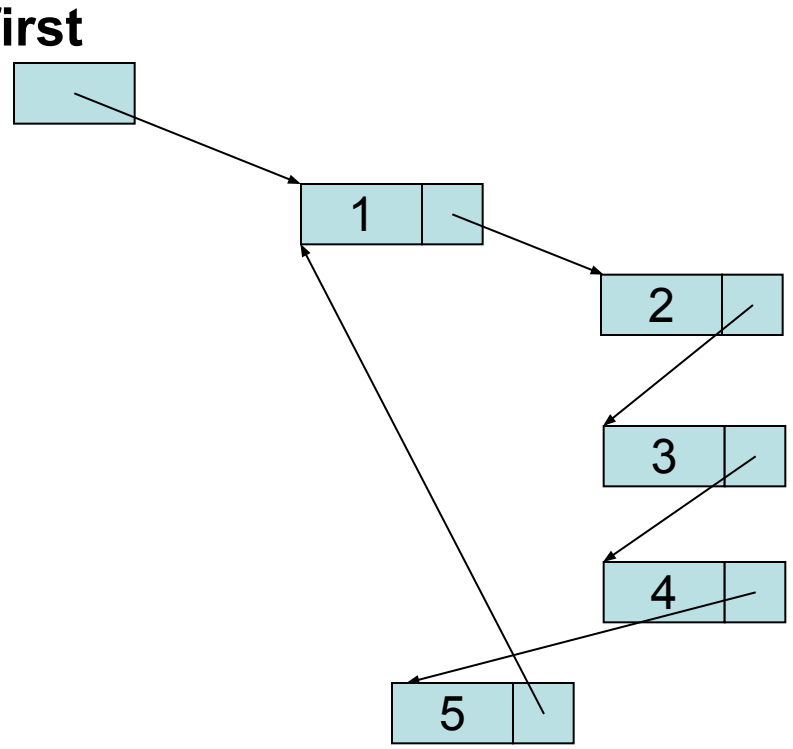




Динамические структуры данных (4)

```
    q=r;
    puts("Result");
    if(q==NULL) puts("No information");
    else
    do { printf("%s %5.1f\n",q->det,q->diam);
              q=q->p; }
         while (q!=NULL);
    return 0;
}
```

Кольцевой список

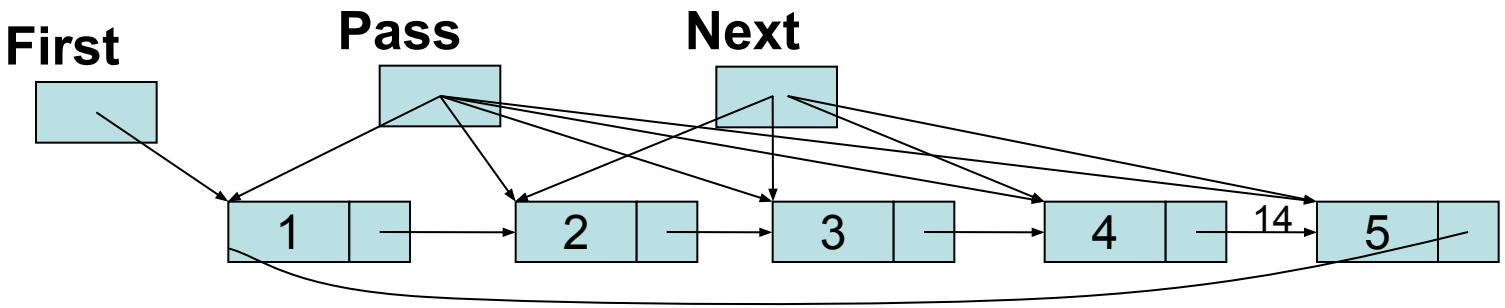



```
1 2 3 4 5

// Ex10_2.cpp
#include "stdafx.h"
#include <stdio.h>
int play(int n,int m)
{
  struct child {
         int name;
         child *p;};
 int i,j;
 child *first,*next,*pass;
```

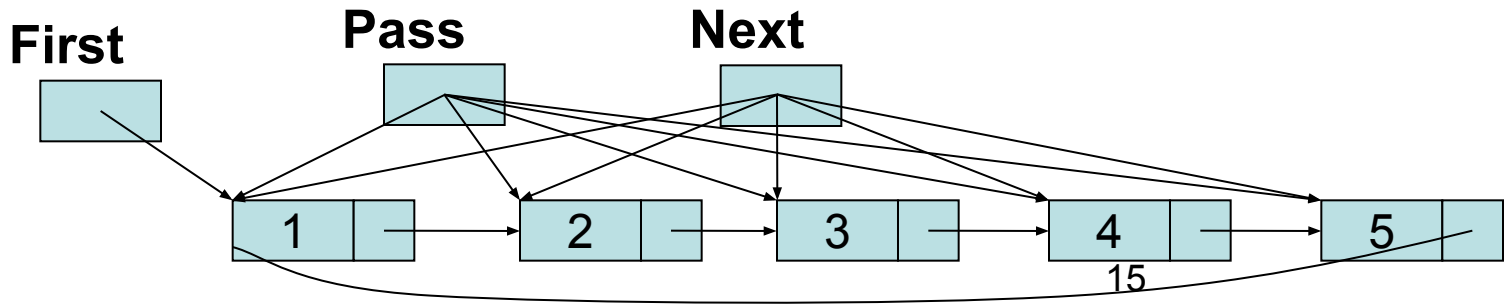
Создание списка

```
{ Создание списка }
  first=new child;
       first->name=1;
       pass=first;
       for( i=2;i<=n;i++)
          {next=new child;
                  next->name=i;
                  pass->p=next;
                  pass=next;          }
       pass->p:=first; {Замыкание круга}
```



Проход по кольцу m-1 раз

```
pass=first;
for{i=n;i>1;i++)
    {
        for(j=1;j<m;j++)
            { next=pass;
              pass=pass->p;}
```

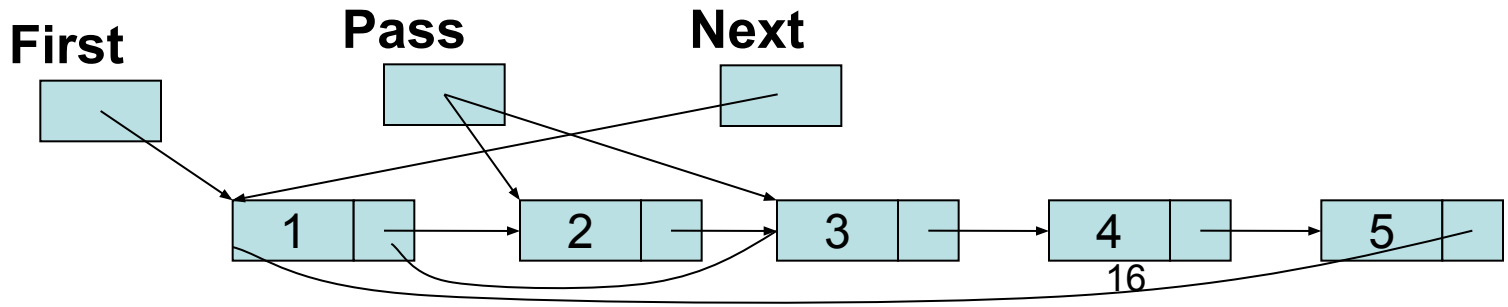


Удаление m-го элемента. Основная программа

```
printf("%2d\n",pass->
  name);
next->p=pass->p;
delete(pass);
pass=next->p;
 }
 //Возврат результата
return pass->name;
}
```

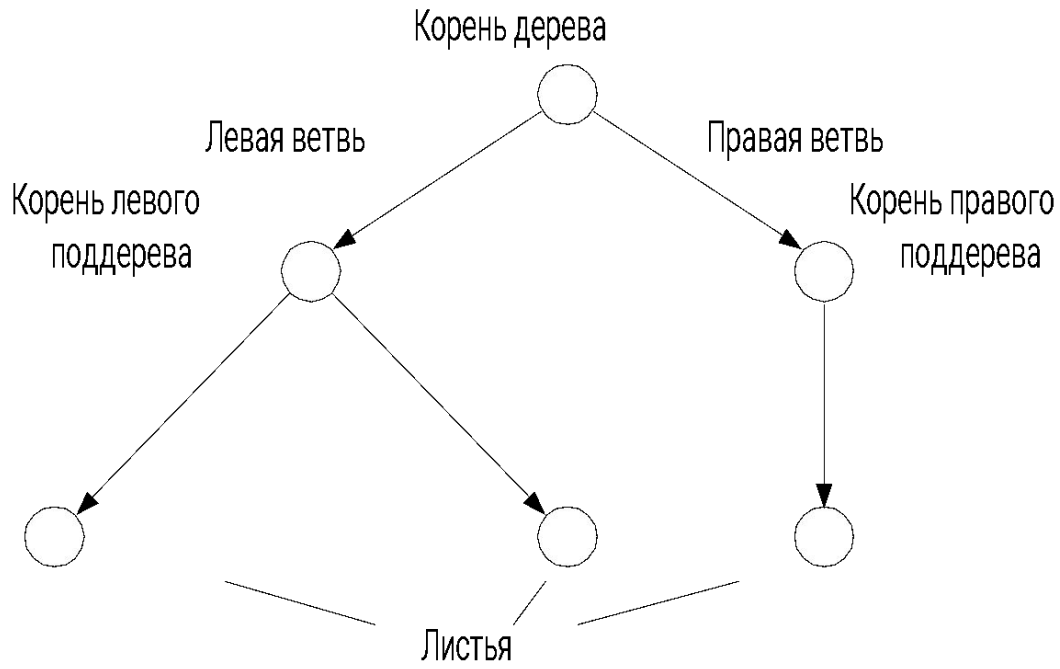
```
int main()
{
 printf("Result =");

printf("%4d\n",play(5,7));
}
```



10.3 Бинарные сортированные деревья

В математике ***бинарным деревом*** называют конечное множество вершин, которое либо пусто, либо состоит из корня и не более чем двух непересекающихся бинарных деревьев, называемых левым и правым поддеревьями данного корня.

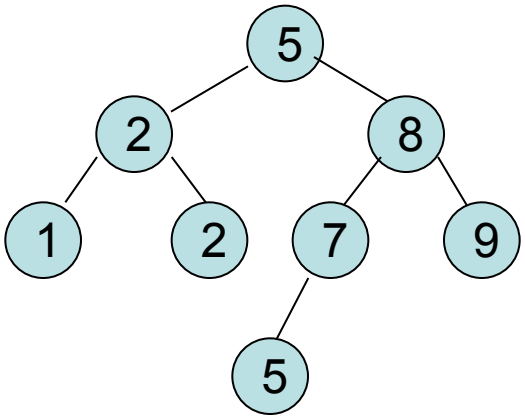


Вершины, из которых не выходит ни одной ветви, называют ***листьями***

Сортированные бинарные деревья, строятся по правилу: *ключевое поле левого поддерева должно содержать значение меньше, чем в корне, а ключевое поле правого поддерева – значение больше или равное значению в корне*.

Построение бинарного дерева

Рассмотрим последовательность целых чисел: {5, 2, 8, 7, 2, 9, 1, 5}

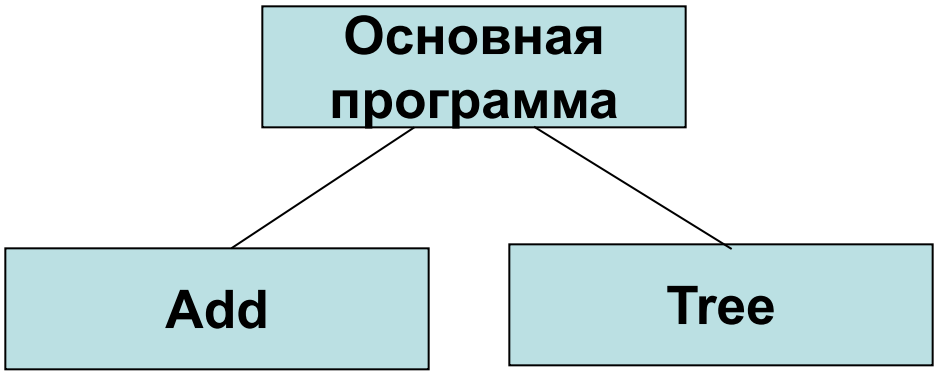


Пример. Разработать программу сортировки последовательности чисел с использованием бинарного дерева.

Описание элемента дерева

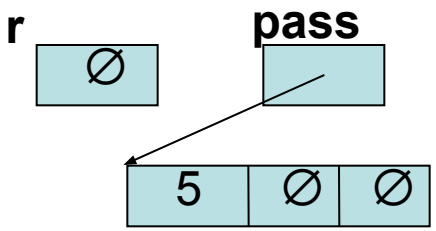
```
// Ex10_3.cpp
#include "stdafx.h"
#include <stdio.h>
#include <STDLIB.H>
#include <conio.h>
#define lim 100
struct top_ptr
{int value;
top_ptr * left;
top_ptr * right;};
int next_number;
top_ptr * r,*pass;
```

Схема структурная ПО



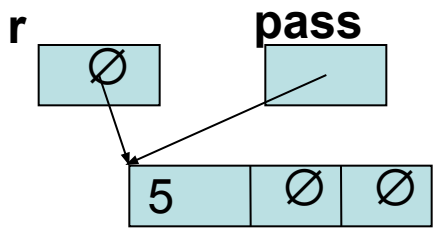
Основная программа

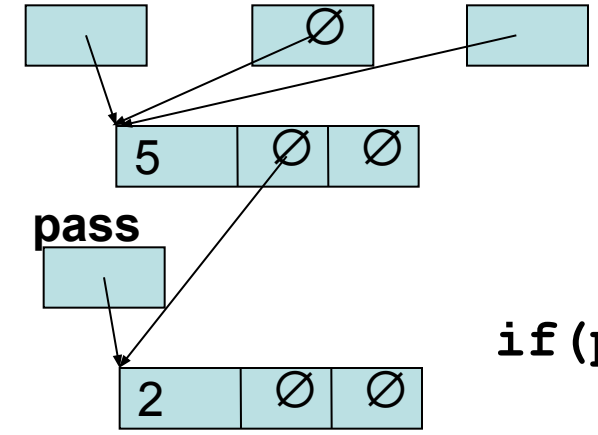
```
int main(int argc,char argv[])
 { r=NULL;
  puts("input value or 1000 for end");
  scanf("%d",&next_number);
  while(next_number!=1000)
    { pass=new top_ptr;
      pass->value=next_number;
      pass->left=NULL;
      pass->right=NULL;
      Add1(&r,pass);
      scanf("%d",&next_number);
      }
  puts("===Result===");
  Tree1(r); printf("\n");
 getch();return 0;
}
```



Нерекурсивная процедура построения дерева

```
void Add1(top_pt **r,top_ptr *pass);
{top_ptr *next,*succ;
 if(*r==NULL) *r=pass;
 else
 {succ=*r;
                  while(succ!=NULL)
r          succ    next {next=succ;
                            if(pass->value<succ->value)
                                succ=succ->left;
                            else succ=succ->right;
pass                    }
                  if(pass->value<next->value
                       next->left=pass;
                  else next->right=pass;
          }
}
```





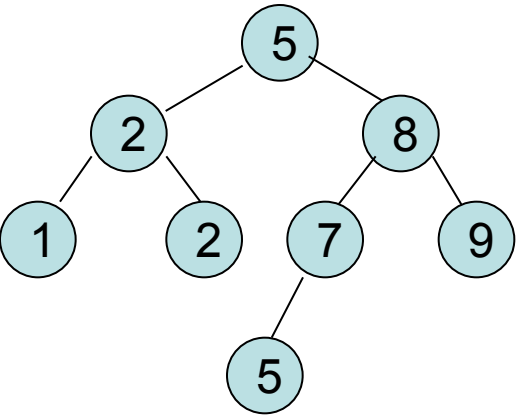
Рекурсивная процедура построения дерева

```
void Add2(top_ptr **r,top_ptr *pass)
{ top_ptr * rr;
rr=*r;
if(rr==NULL) *r=pass;
else
if (pass->value<rr->value)
Add2(&rr->left,pass);
else Add2(&rr->right,pass);
}
```

Нерекурсивная процедура обхода дерева

```
void Tree1(top_ptr *r)
{ struct memo{
short int nom;
top_ptr * adres[lim];} memo1;
top_ptr * pass;
memo1.nom=-1;
pass=r;
```

Нерекурсивная процедура обхода дерева (2)




```
while ((pass!=NULL)||(memo1.nom!=-1))
if (pass!=NULL)
{ if( memo1.nom==lim-1)
        { puts(" Error lim"); exit(1);}
          memo1.nom=memo1.nom+1;
          memo1.adres[memo1.nom]=pass;
          pass=pass->left;
        }
else{ pass=memo1.adres[memo1.nom];
memo1.nom=memo1.nom-1;
printf("%4d\n",pass->value);
pass=pass->right;}
}
```

Рекурсивная процедура обхода дерева

```
void Tree2(top_ptr *r)
{
if(r!=NULL)
{ Tree2(r->left);
printf("%4d",r->value);
Tree2(r->right);
}
}
```

Рекурсивная процедура удаления вершины дерева

```
void ud(top_ptr **r,top_ptr **q)
{//вложенная процедура удаления
   top_ptr * rr;
    if((*r)->right==NULL)
      {
       (*q)->value=(*r)->value;
        *q=*r;
        rr=*r;
        (*r)=(*r)->left;
         delete rr;
      }
      else
       ud(&((*r)->right),q);
  }
```

Рекурсивная процедура удаления вершины дерева(2)

```
void udaldr(top_ptr **d,int k)
{ top_ptr *q;
  top_ptr * dd=*d;
  if (*d==NULL)  // Первый случай
   puts(" Element not found or tree is empty");
  else //{Вершина есть - ищем ее}
     if (k<dd->value)udaldr(&dd->left,k);
     else
       if ( k>dd->value)udaldr(&dd->right,k);
       else { //{ Элемент найден - удаляем его}
         q=*d; //{ Второй случай}
         if (q->right==NULL){ *d=q->left;delete q;}
         else
           if (q->left==NULL){*d=q->right;delete q;}
           else //{ Третий случай удаления}
              ud(&q->left,&q); //{удаление q)}
           }   }
```

Рекурсивная процедура поиска вершины с номером k

При необходимости поиск вершины также можно осуществлять, используя рекурсию. Построим рекурсивную функцию, которая будет возвращать true, если элемент найден и false – в противном случае. Адрес найденного элемента будем возвращать в параметре pass:

```
bool Find(top_ptr*r,top_ptr **pass, int n)
{  if (r==NULL)return false; //значение не найдено
   else
     if (n==r->value)
      {  *pass=r;          // запомнили адрес
         return true;      //значение найдено
      }
     else
      if (n<r->value)
        return Find(r->left,pass,n);    // влево
      else
        return Find(r->right,pass,n);    //вправо
}
```