Коллекции и рекурсия

2013

Описание (из прошлой лекции)

```
struct el{
  int item;
  el* next;
};
```

```
class List{
 public:
   . . .
 private:
   el* head;
   el* tail;
   . . .
};
```

Метод печати в обратном порядке

- Нельзя сделать метод рекурсивным, т.к. List не является рекурсивной структурой данных
- Рекурсия заложена в структуре el
- Поэтому организовать рекурсию можно относительно поля head, но это-приватные данные

- Создадим два метода: открытый нерекурсивный, который будет вызывать закрытый рекрсивный

```
public:
void revPrint();


private:
void revP(el * f);
```

```
void List::revPrint()
{
        revP(head);
}
void List::revP(el * f)
{
    if (f)
        {
            revP(f->next);
            cout<<f->item<<" ";
        }
}
```

- вызываем публичный метод

```
List s1;
. . .
s1.revPrint();
```

- Как обойтись без методов-оберток?
- Нужно, чтобы класс представлял рекурсивную структуру

```
class List{
  public:
    . . .
  private:
   int   item;
   List * next;
    . . .
};
```

Бинарное дерево поиска

```
class TreeR {
private:
    int data;
    TreeR* lt,*rt;
public:
    TreeR(int a): data(a), lt(0), rt(0){ }
     TreeR(const TreeR& t);
    ~TreeR();
    void add(int a);
    void LKR();
};
```

Реализация

```
void TreeR:: add(int a)
{
  if (data > a) {
    if (lt) lt->add(a);
    else lt = new TreeR(a);
  }
  else {
    if (rt) rt->add(a);
    else rt = new TreeR(a);
 }
}
```

Реализация

```
void TreeR:: LKR()
 {
    if (lt) lt->LKR();
    cout<<"  "<<data<<"  ";
    if (rt) rt->LKR();
}
```

Реализация

```
TreeR:: ~TreeR()
{
   if (lt) delete lt;
   if (rt) delete rt;
}
```

Самостоятельно

- Конструктор копирования
- Сравнение на равенство двух деревьев
- Операцию присваивания
- Операцию сложения двух деревьев

Работа с деревом

```
TreeR* t1 ; // это пустое дерево
   t1= new TreeR(5);
```

- если дерево не пусто, в нем есть корневой узел
- нельзя реализовать метод проверки на пустоту

Работа с деревом

- поэтому вызовы методов нужно предварять проверкой

```
if (t1) t1->add(3);
if (t1)  t1->LKR();
if (t1)  delete t1 ;
```

- или всегда создавать дерево при объявлении указателя

```
TreeR* t1= new TreeR(5);
```

- это может создать алгоритмические проблемы
- кроме того может возникнуть трудность при удалении из дерева последней вершины