

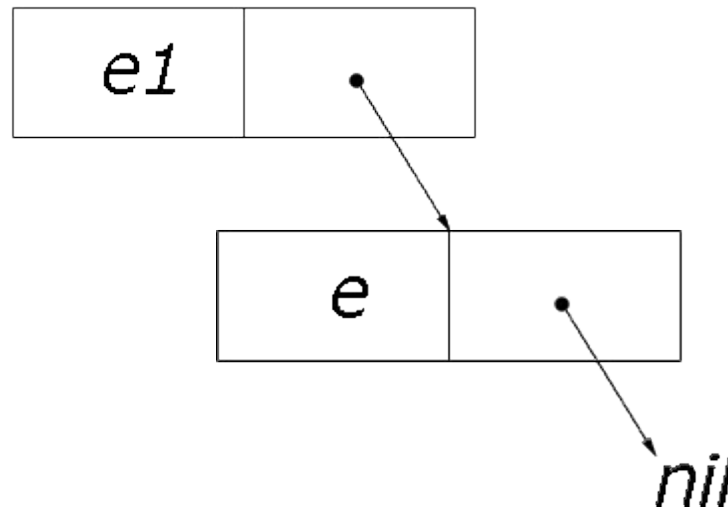


Рекурсивные структуры данных

Списки, двоичные деревья

Список

- Списком является пустой список (*nil*)
- Если l – список элементов типа a , e – элемент типа a , то $cons(e, l)$ – список элементов типа a





Работа со списками

```
program testlist;
```

```
type
```

```
  plist = ^list;
```

```
  list = record
```

```
    el : integer;
```

```
    next : plist;
```

```
  end;
```

```
function addEl(l : plist; el : integer) : plist;
```

```
var
```

```
  pl : plist;
```

```
begin
```

```
  New(pl);
```

```
  pl^.el := el;
```

```
  pl^.next := l;
```

```
  addEl := pl;
```

```
end;
```

```
var
```

```
  l : plist;
```

```
  i : integer;
```

```
begin
```

```
  l := nil;
```

```
  for i := 1 to 10 do l := addEl(l,i);
```

```
end.
```

```
type
```

```
  plist = ^list;
```

```
  list = object
```

```
    el : integer;
```

```
    next : plist;
```

```
  end;
```



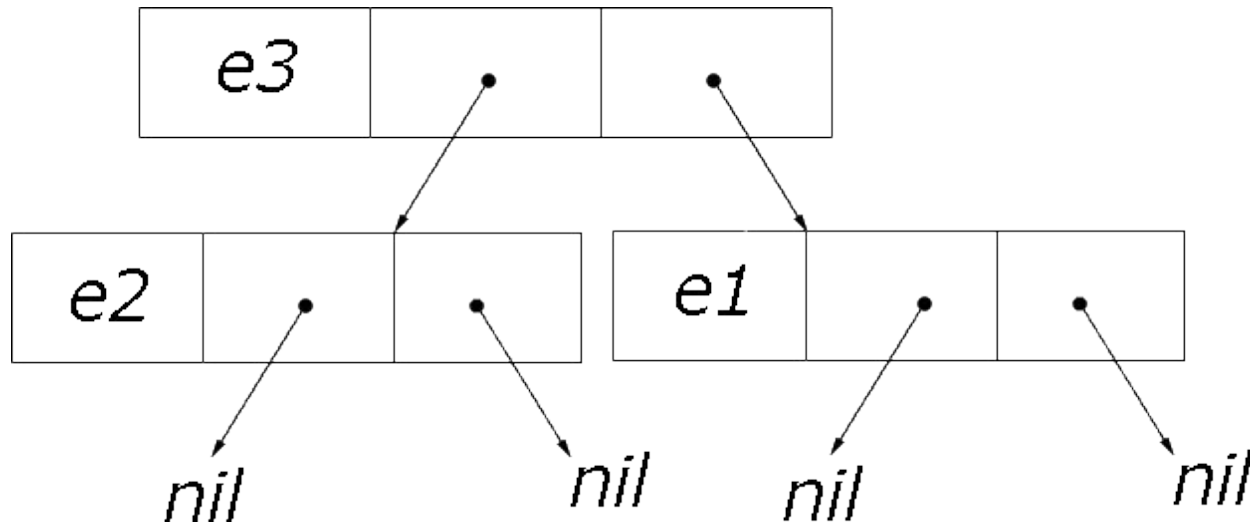
Работа со списками

```
procedure writelist(l : plist);  
begin  
  while l <> nil do begin  
    write(l^.el, ' ');  
    l := l^.next;  
  end;  
end;
```

```
procedure writelistrec(l : plist);  
begin  
  if l <> nil then begin  
    write(l^.el, ' ');  
    writelistrec(l^.next);  
  end;  
end;
```

Двоичное дерево

- Деревом является пустое дерево
- Если l_1, l_2 – деревья элементов типа a , e – элемент типа a , то $tree(e, l_1, l_2)$ – дерево элементов типа a





Работа с деревьями

```
program testtree;
```

```
type
```

```
  ptree = ^tree;  
  tree = record  
    el : integer;  
    left, right : ptree;  
  end;
```

```
function addEl(l,r : ptree; el : integer) : ptree;
```

```
var  
  pl : ptree;  
begin  
  New(pl);  
  pl^.el := el;  
  pl^.left := l;  
  pl^.right := r;  
  addEl := pl;  
end;
```

```
function copyTree(t : ptree) : ptree;  
var  
  pt : ptree;  
begin  
  if t = nil then copyTree := nil  
  else begin  
    New(pt);  
    pt^.el := t^.el;  
    pt^.left := copyTree(t^.left);  
    pt^.right := copyTree(t^.right);  
    copyTree := pt;  
  end;  
end;
```

```
prevt := nil; t := addEl(nil,nil,i);  
for i := 1 to 6 do begin  
  r := copyTree(prevt);  
  prevt := t;  
  t := addEl(t,r,i);  
end;
```



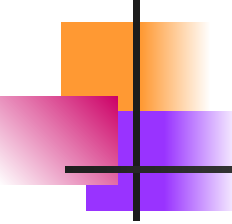
Работа с деревьями

```
procedure writetree(t : ptree; level : integer);
var
  i : integer;
begin
  if t <> nil then begin
    writetree(t^.left,level+1);
    for i := 1 to level do write(' ');
    writeln(t^.el);
    writetree(t^.right,level+1);
  end;
end;
```



Поиск в дереве

```
function search(t : ptree; el : integer) : boolean;
begin
  if t = nil then search := false
  else
    if el = t^.el then search := true
    else if el < t^.el then search := search(t^.left,el)
         else search := search(t^.right,el)
  end;
end;
```

Сортировка с помощью двоичного дерева

```
procedure insert(var t : ptree; el : integer);
begin
  if t = nil then begin
    New(t);
    t^.el := el;
    t^.left := nil;
    t^.right := nil;
  end
  else
    if el < t^.el then insert(t^.left,el)
      else insert(t^.right,el)
  end;
end;
```