Введение в C++

Линейный списки, двоичные деревья, hash-таблицы

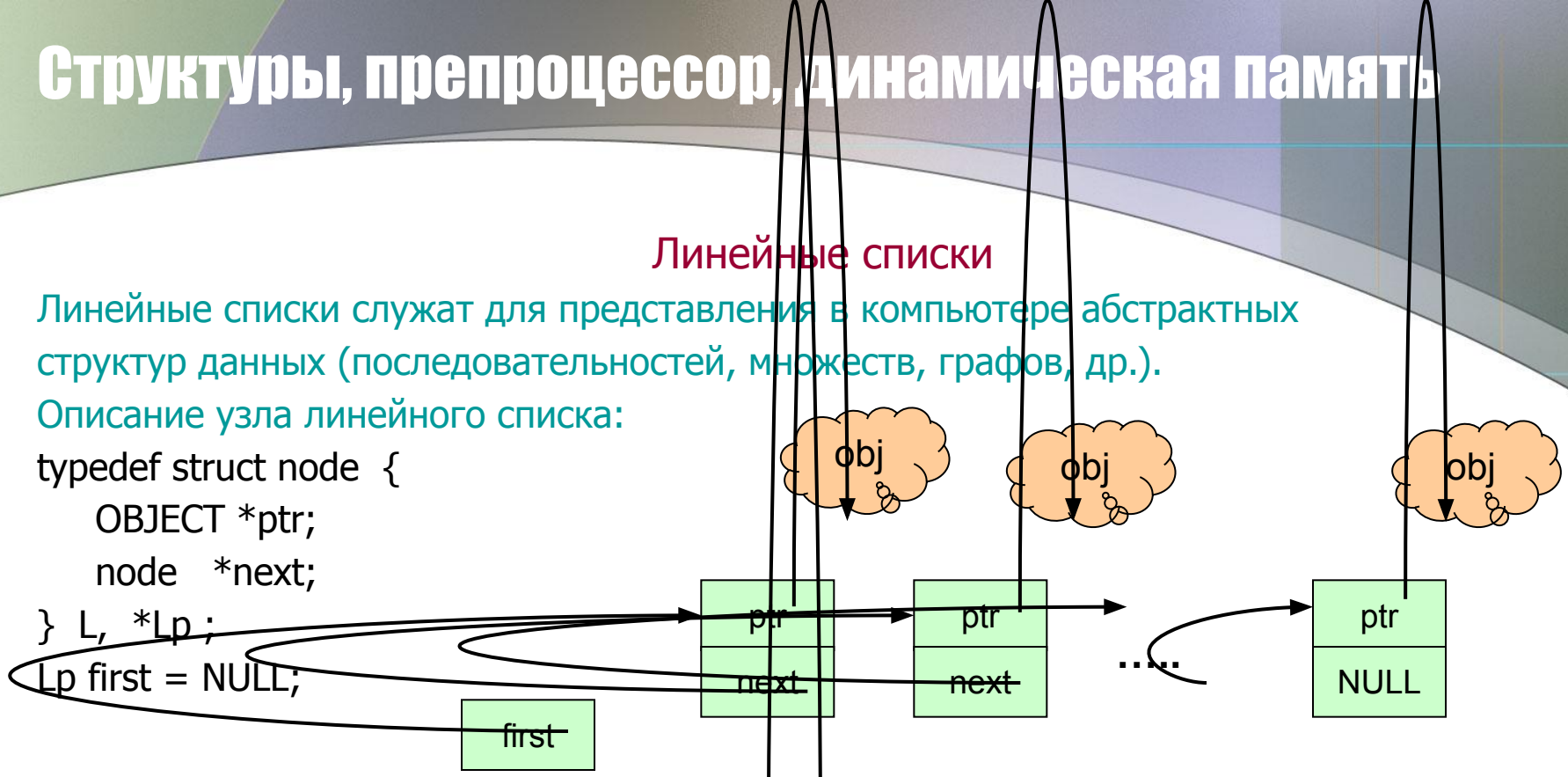
Структуры, препроцессор, динамическая память

Линейные списки

Линейные списки служат для представления в компьютере абстрактных структур данных (последовательностей, множеств, графов, др.).

Описание узла линейного списка:

```
typedef struct node  {
     OBJECT *ptr;
     node   *next;
}  L,  *Lp ;
Lp first = NULL;
```



пустой список

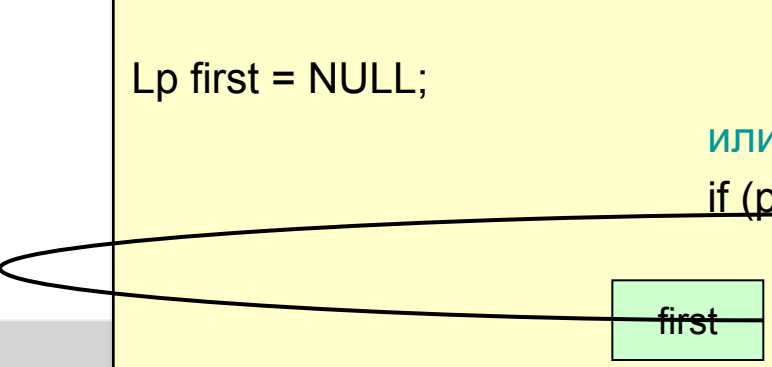
```
Lp first = NULL;
```

список из 1 элемента

```
if (p != NULL) p = p -> next;
```

или

```
if (p != NULL) p = p -> next;
```



продвижение вперед на 1 элемент

Структуры, препроцессор, динамическая память

Операции с линейными списками

Итеративный обход списка (в прямом направлении)

```
Lp p = first;
while ( p != NULL ) { R (p -> ptr );          // обработка объекта в узле
        p = p -> next; }
```

Рекурсивный обход списка в прямом направлении

```
void w1 (Lp p ) {
        if ( p != NULL ) { R (p -> ptr );      // обработка объекта в узле
        w1 ( p -> next ); }
}
```

Рекурсивный обход списка в обратном направлении

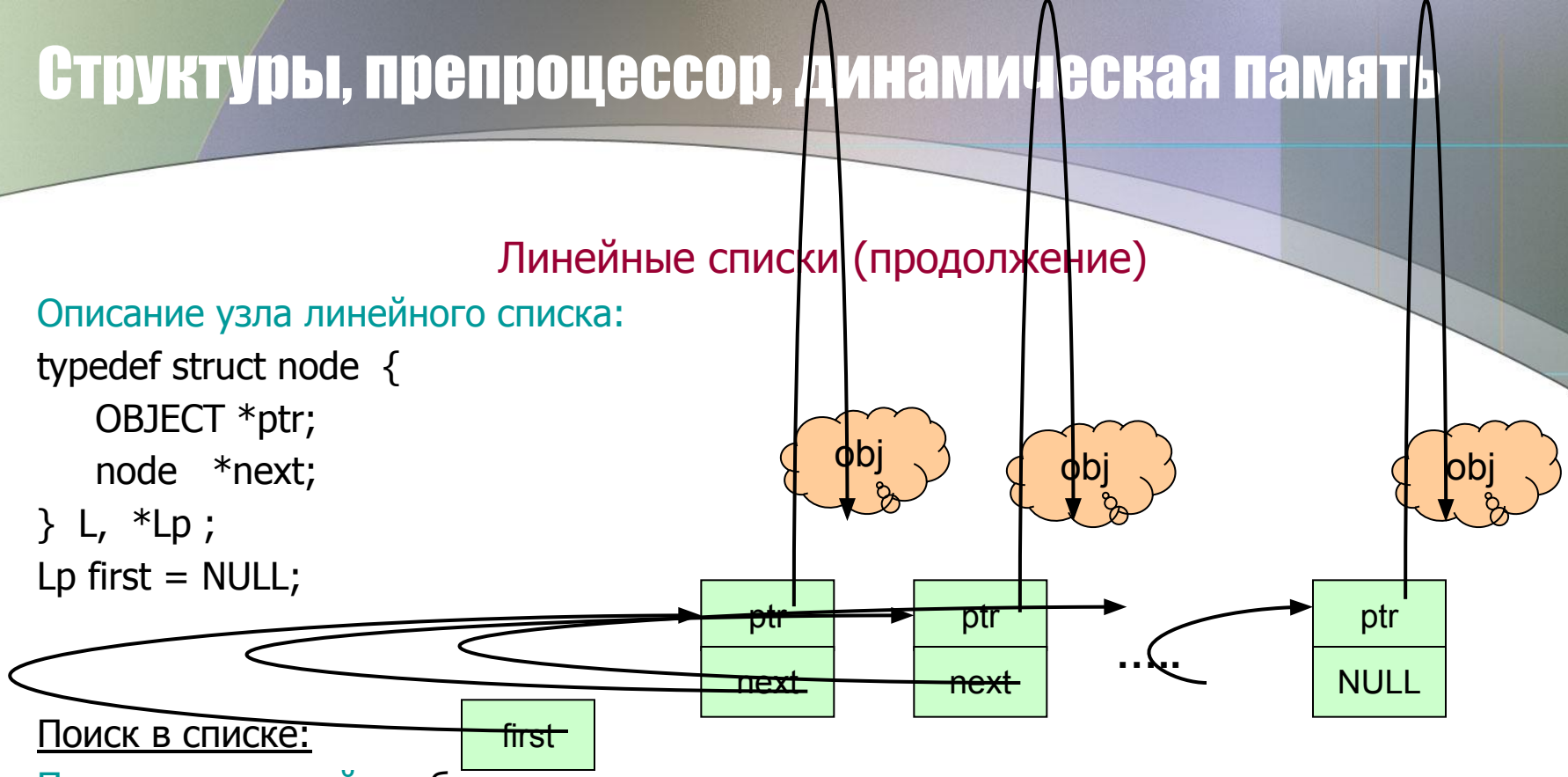
```
void w2 (Lp p ) {
        if ( p != NULL ) { w2 ( p -> next );
        R (p -> ptr );      }                  // обработка объекта в узле
}
```

Структуры, препроцессор, динамическая память

Линейные списки (продолжение)

Описание узла линейного списка:

```
typedef struct node  {
     OBJECT *ptr;
     node   *next;
}  L,  *Lp ;
Lp first = NULL;
```



Поиск в списке:

Пусть задан некий < образец> для поиска

```
Lp p = first;
while ( p != NULL && *p -> ptr != <образец> ) p = p -> next;
If ( p == NULL) < искомого элемента в списке нет > ;
       else  < p есть указатель на нужный узел списка > ;
```

Структуры, препроцессор, динамическая память

Вставка узла в начало списка:

```
Lp q = new L;   // здесь q – указатель на новый узел списка
q -> next = first;
first = q;          // новый узел становится первым узлом списка
```
Самостоятельно сделайте поясняющий рисунок к этому фрагменту кода

Удаление узла из начала списка:

```
if ( first != NULL ) first = first -> next;
```
Обратите внимание – память из-под освободившегося узла системе не возвращается

Вставка узла в произвольное место списка:

```
Lp q = new L;          // q – указатель на новый узел списка
q -> next = p - > next;      // здесь p – указатель на произвольный узел списка
p -> next = q;
```
Самостоятельно сделайте поясняющий рисунок к этому фрагменту кода

Структуры, препроцессор, динамическая память

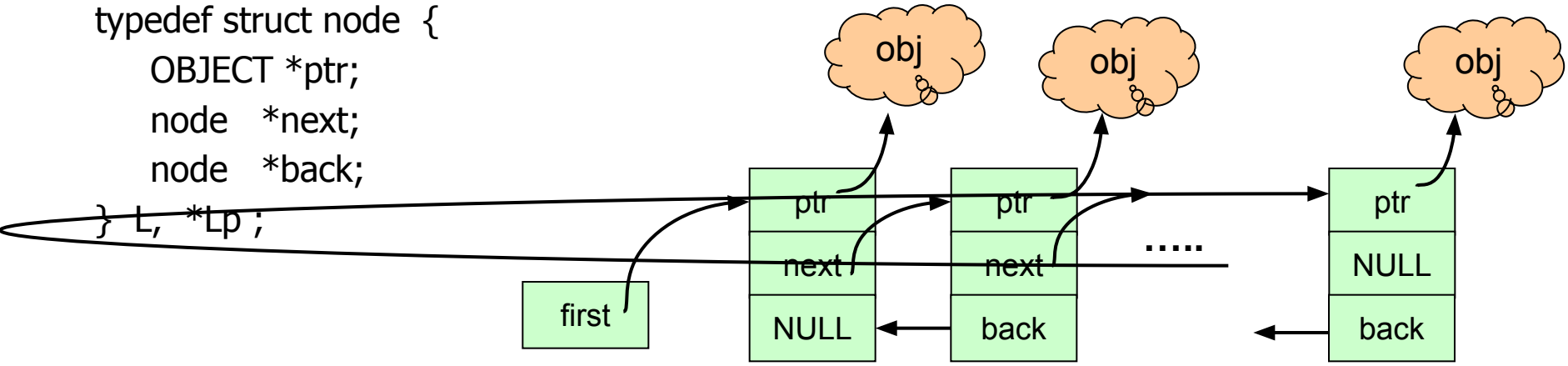
Удаление узла из произвольного места списка:

```
// пусть p – указатель на произвольный узел списка
If ( p != NULL  && p -> next  != NULL )
        p -> next = p -> next -> next;
```

Самостоятельно сделайте поясняющий рисунок к этому фрагменту кода

Описание узла двусвязного списка:

```
typedef struct node  {
     OBJECT *ptr;
     node   *next;
     node   *back;
} L,  *Lp ;
```



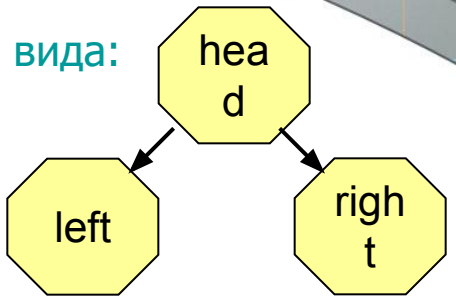
Также имеются циклические и многосвязные списки.

Структуры, препроцессор, динамическая память

Двоичные деревья

Определение:

Пустой граф и граф с одним узлом есть двоичное дерево. Граф вида:

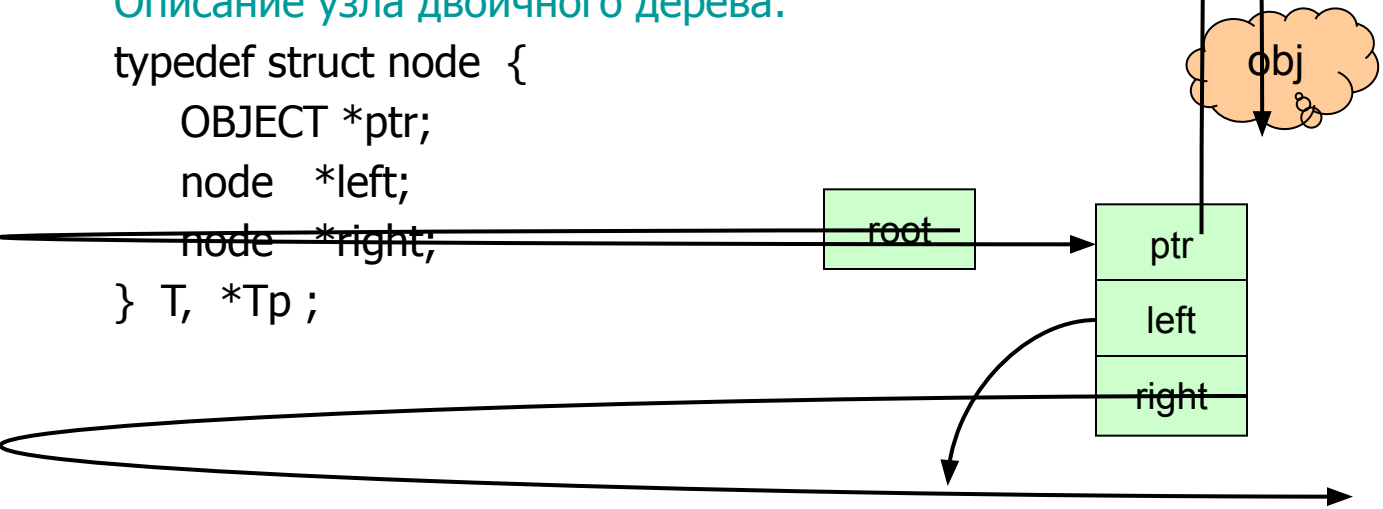


есть двоичное дерево.

Двоичное дерево, левая и правая части которого
есть двоичное дерево, также есть двоичное дерево.

Описание узла двоичного дерева:

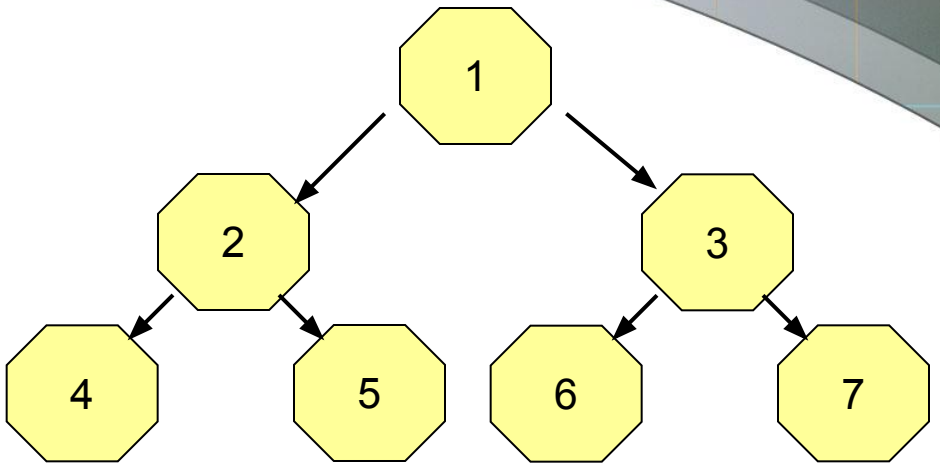
```
typedef struct node  {
    OBJECT *ptr;
    node   *left;
    node   *right;
}  T,  *Tp ;
```



Структуры, препроцессор, динамическая память

Способы обхода двоичных деревьев:

head, left, right (hlr – обход)
порядок посещения узлов: 1 2 3 4 5 6 7
left, head, right (lhr – обход)
порядок посещения узлов: 4 2 5 1 6 3 7
left, right, head (lrh – обход)
порядок посещения узлов: 4 5 2 6 7 3 1



Алгоритм hlr – обхода:

```
void hlr ( Tp p )
{
    if ( p ) {
       R ( p -> ptr );    // R - обработка объекта в узле
       hlr ( p -> left );
       hlr ( p -> right );
       }
}
```

Структуры, препроцессор, динамическая память

Алгоритм lhr – обхода:

```
void lhr ( Tp p )
{
    if ( p ) {
            lhr ( p -> left );
       R ( p -> ptr );    // R - обработка объекта в узле
       lhr ( p -> right );
       }
}
```

Алгоритм lrh – обхода:

```
void lrh ( Tp p )
{
    if ( p ) {
            lrh ( p -> left );
       lrh ( p -> right );
       R ( p -> ptr );    // R - обработка объекта в узле
       }
}
```

Структуры, препроцессор, динамическая память

Поиск, вставка, сортировка в двоичных деревьях

Двоичные деревья полезны, когда им присущ внутренний порядок (сорт. дерево).

Пусть определена некоторая хэш-функция h (<OBJECT>).

Итеративный поиск в сорт. дереве:

```
//   Пусть задан некий < образец>
void search1 ( Tp &p) {
        while ( p   && *p -> ptr != <образец> )
                if ( h ( *p -> ptr ) >= h ( <образец> ) )  p = p -> left;
                else  p = p -> right;
}       //   p – указатель на найденный узел или NULL
```

Рекурсивный поиск в сорт. дереве:

```
void search2 ( Tp &p) {
        if ( p )       if ( <образец> != *p -> ptr)
                if ( h (<образец>) <= h (*p -> ptr )  search2 ( p -> left );
                else search2 ( p -> right) ;
}       //   p – указатель на найденный узел или NULL
```

Структуры, препроцессор, динамическая память

Рекурсивный поиск в сорт. дереве с включением:

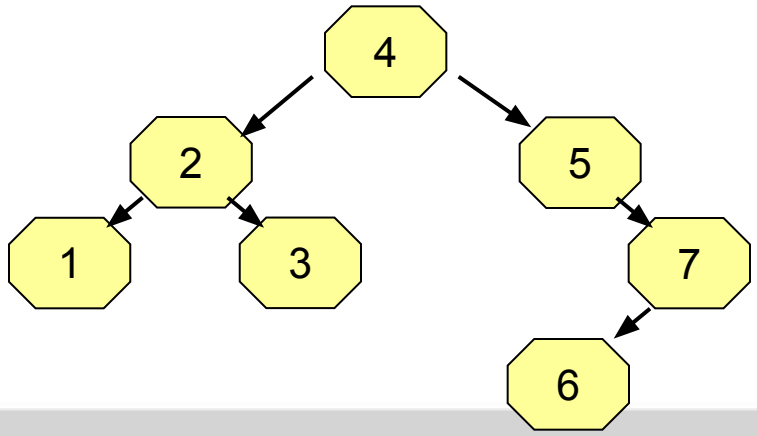
```
//   Пусть задан некий < образец> для поиска и вставки
void locate ( Tp &p )
{
    if ( !p ) { p = new T; p -> ptr = &<образец>;  p -> left = p -> right = NULL; }
    else if ( <образец> != *p -> ptr )
       if ( h ( <образец> ) <= h (*p -> ptr ) ) locate ( p -> left );
       else locate ( p -> right ) ;
}      //   p – указатель на найденный или созданный) узел
```

После отработки алгоритма locate сортировка есть просто lhr (rhl) –обход сорт. дерева.

Пример:

пусть данные поступают в дерево
в следующем порядке:
4 5 2 7 3 6 1

Осуществим lhr-обход, получим:
1 2 3 4 5 6 7



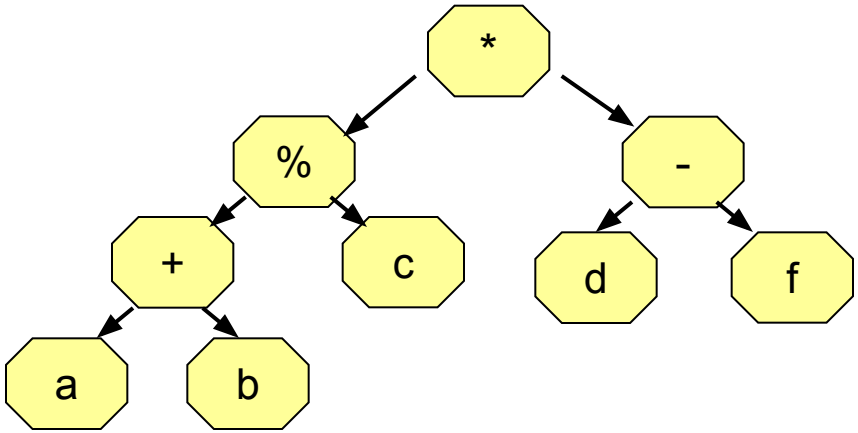
Структуры, препроцессор, динамическая память

Задача из теории компиляторов. Вычисление выражений:

Пусть требуется вычислить следующее выражение:

((a + b) % c) * (d – f)

Построим следующее двоичное дерево:



Обойдем данное двоичное дерево в порядке lrh и будем вычислять выражения в узлах, используя обратную польскую запись (постфиксную форму).

Проверьте, что при таком обходе дерева выражение будет вычислено корректно.

Исследуйте работу программного стека.

Структуры, препроцессор, динамическая память

Hash-таблицы (таблицы с перемешиванием)