Семантический уровень ЯП
или
Контекстные условия ЯП

Почему ?

A^nB^nC^n - не КСЯ

```
int   A^n;
int   main  (  )  {
      A^n =  2 * A^n;
}
```

Нельзя построить КСГ, описывающую правильность использования идентификаторов даже в такой примитивной программе

Примеры контекстных условий

Каждый объект должен быть описан

Языки разделяются на языки с умолчанием (например, Бейсик) и языки без умолчания (например, C++ , Java).
В языке без умолчания каждый объект должен быть описан ровно один раз, а в языке с умолчанием объект можно описать не более одного раза.

```
int *  index,    data,    index[100];
double   data;
```

Ошибки: дублирование идентификаторов index, data.

Вывод

1. Необходимо хранить в некоторой таблице идентификаторы, которые объявлены в программе
2. При обработке каждого очередного объявления необходимо выполнять проверку на дублирование имени

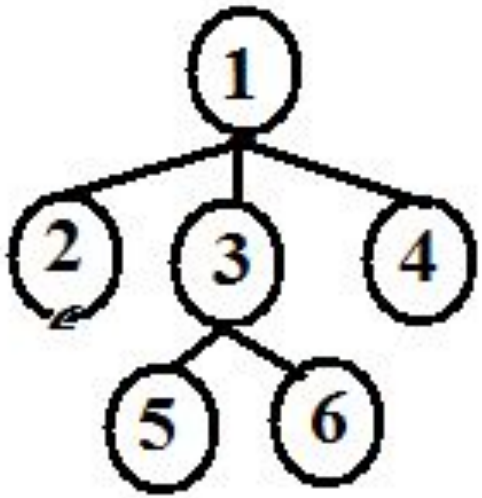
Область использования объекта должна быть согласована с областью его действия

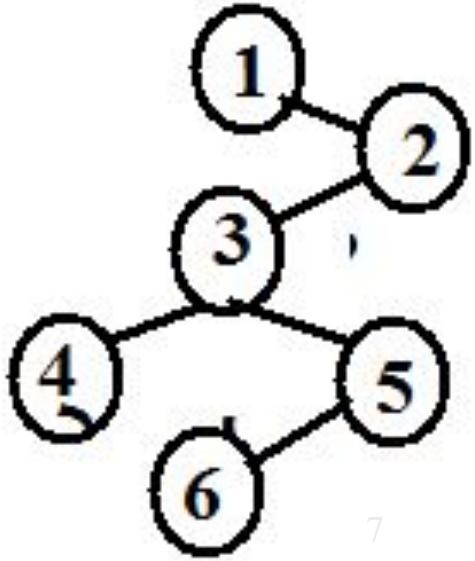
```
int i;
int   main  ( )  {
     int     j;
     i=1;   j=2; // глобальная  i   и   локальная    j
   {          int   i,  k;
              i=3; k=4;       // локальные в новом блоке
     }
      k=3;         // ошибка: недоступная    k
}
```

Вывод

1. Семантическая таблица должна иметь иерархическую структуру
2. В качестве иерархической структуры можно использовать дерево
3. Произвольное дерево в программе можно представить бинарным деревом

Слева – сосед,
Справа – потомок.





Типы формальных и фактических параметров функций и их количество должны совпадать

```
diuble   a[ 1000];   int   len;
bool fun (double  *,  int );
//  правильно
fun (a,   len) ;          fun ( & a  [len],   len  /  2);

// ошибки
Fun ( a[ len], a);      fun (a);
```

Вывод

1. Функции хранятся в семантическом дереве вместе с информацией о параметрах
2. Параметры функции являются локальными в теле функции, следовательно соответствующие вершины являются потомками вершины для функции
3. При обработке вызова функции требуется осуществлять контроль типа каждого параметра
4. Кроме контроля типов параметров при вызове функции требуется проверка совпадения количества формальных и фактических параметров

В программе разрешаются приведения типов данных (явные и неявные)

```
double   x, y, z , h [ 1000];
int  a= 100, b = 3, c = a / b;        // c = 33
x= a / b;         // x = 33.000000
 y = a;
z =  y / b;         // z = 33.333333
void * addr = ( void *)  h;

file * in;
y = in; //  неверное приведение типов
```

Вывод

1. При трансляции операций необходимо проверять типы данных операндов этой операции
2. Для каждой операции необходимо контролировать допустимость приведения типов

В ЯП допускается использование именованных констант

```
int  index = 100;
const   int   maxIndex = 100000;
 index =  maxIndex  / 2;    // правильно
maxIndex  =  maxIndex  / 2;    // Ошибка
```

ВЫВОД

1. В таблице необходимо хранить признак константы
2. При выполнении операции присваивания контролировать отсутствие этого признака

В ЯП допускается описание пользовательских типов

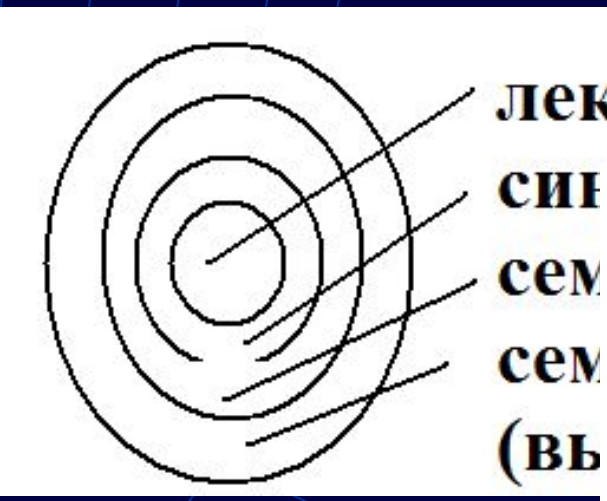
```
typedef  long  long  int   LL;
LL  ww,  zz;  //  верно:  LL  - тип
 ww  a, b ;       // ошибка:  ww  не  тип
 LL = 2;  //  ошибка:  LL – тип, а не переменная

struct  point {int x, y, z ; };
point a;   //  такой тип есть
 a  data; // ошибка  -  a не является типом
a.x = 0;     // верно;  a  - данные структурного типа
a.x.x – a.j;    // ошибки:  неверные поля  структуры
a.y = point.x;    //  ошибка: point – тип, а не данные
point.x = 0;
```

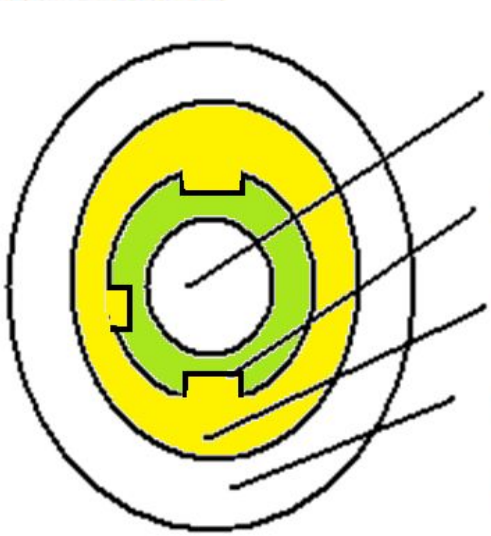
Вывод

1. В таблице необходимо хранить признак типа для каждого идентификатора
2. С типом можно объявлять данные, по нельзя выполнять вычисления
3. Иерархия полей и методов в классах и структурах должна контролироваться при обработке соответствующих объектов
4. Нельзя вызвать не функцию, нельзя (за некоторыми особыми случаями) использовать функции как данные

Семантика & синтаксис

Семантика ЯП – это контекстные условия ЯП. Она фактически неразрывно связана с синтаксисом. Схематически это означает уточнение картины:





Рефакторинг синтаксиса

Требования контекстной зависимости могут привести к корректировке синтаксического уровня.

Пример:

```
D -> D, id  | D , P      // список переменных в описании
P -> id = V              // присваивание
O -> P  | ; |  …         // оператор
```

Семантика , связанная с id, разная в P и O:

- в P надо занести id с соответствующим типом в таблицу и проконтролировать дублирование,
- в O id надо найти в таблице и узнать его тип.

Рефакторинг синтаксиса

Различия в семантике диктуют разделение синтаксической конструкции P на две :

```
D -> D, id  | D , id=V      // список в описании
P -> id = V                 // присваивание
O -> P  | ; |  …            // оператор
```

Семантика id одна и та же в конструкции D, независимо от того, инициализируется id или нет. Более того, такая подстановка упростит и синтаксический анализ, так как отпадает необходимость выбора ветви с инициализацией или без нее.

Лабораторная работа № 7

Тема : Анализ контекстных условий ЯП

.Перечислить типы данных реализуемого ЯП (например, int, double, …) с указанием диапазона значений и длины памяти для хранения значения

. Построить таблицу приведения типов

.Перечислить типы объектов программы (например, функции, простые переменные, структуры, …)

.Привести список контекстных условий для каждого типа объектов с привязкой к соответствующей СД

.Построить перечень данных, которые должны храниться в семантическом дереве для каждого типа объектов

Переходим к реализации семантического уровня

ПРОБЛЕМЫ:

- Как реализовать и хранить семантические типы?
- Что хранить в узле семантического дерева?
- Как реализовать класс семантического дерева?
- Какие функции должен реализовывать этот класс?
- Какую диаграмму классов мы в итоге должны реализовать?

Диаграмма классов

Контекстная зависимость семантики и синтаксиса приводит к необходимости связи между соответствующими классами.
Необходимость связи между классами лексики и семантики определяется необходимостью выдачи сообщений об ошибках.

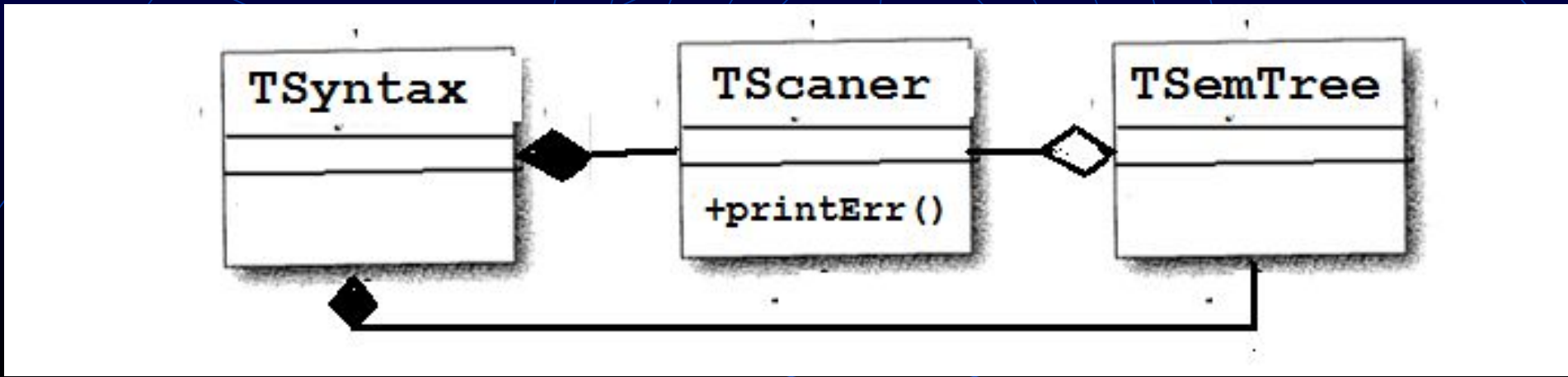


Диаграмма классов (варианты 1 и 2)

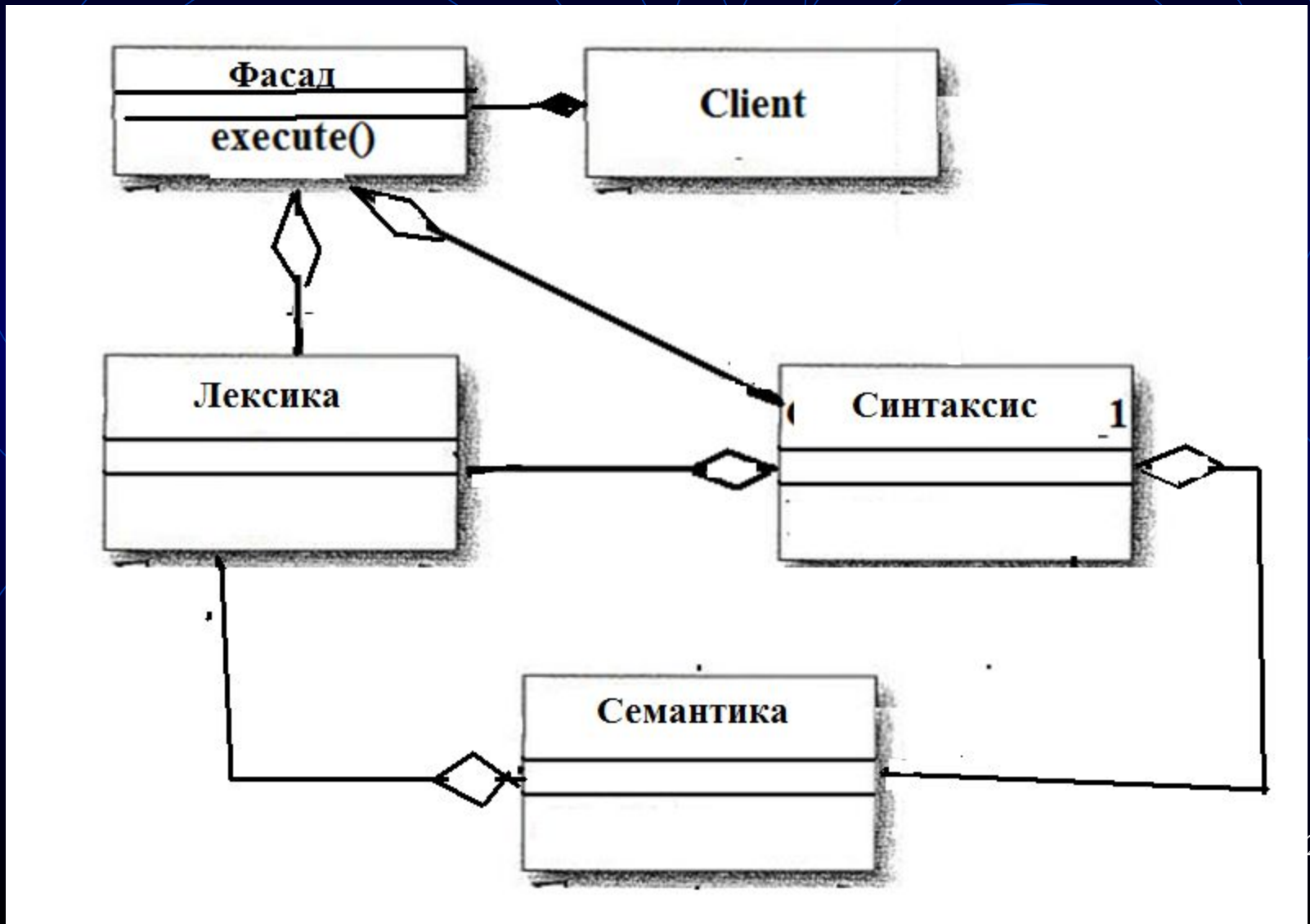
Классы (вариант 1):

- Лексический анализатор
- Синтаксический анализатор
- Семантическое дерево
- Семантика

Классы (вариант 2):

- Лексический анализатор
- Синтаксический анализатор
- Семантика на семантическом дереве

Диаграмма классов (вариант 2)



Семантические типы

```
enum TypeObject {ObjConst=1,     // константа
                 ObjLabel,       // метка
                 ObjVar,         // простая переменная
                 ObjTypeVar,     // простой тип
                 ObjArray,       // массив
                 ObjTypeArray,   // тип массива
                 ObjStruct,      // структура
                 ObjTypeStruct,  // тип  структуры
                 ObjFunct        // функция
…
};
```

Информация в вершине семантического дерева

```
struct Node     {
    LEX   id;           // идентификатор объекта
     TypeObject   DataType;     // тип  объекта
       //   дополнительные  возможности:
      int FlagConst;     // признак константы
       int Param;        // количество параметров функции
        int   sizeOfArray;          // размерность массива
         int   lenOfArray [MAX_N];     // длина измерения
          int FlagInit;      // Флаг начальной инициализации
 …..
};
```

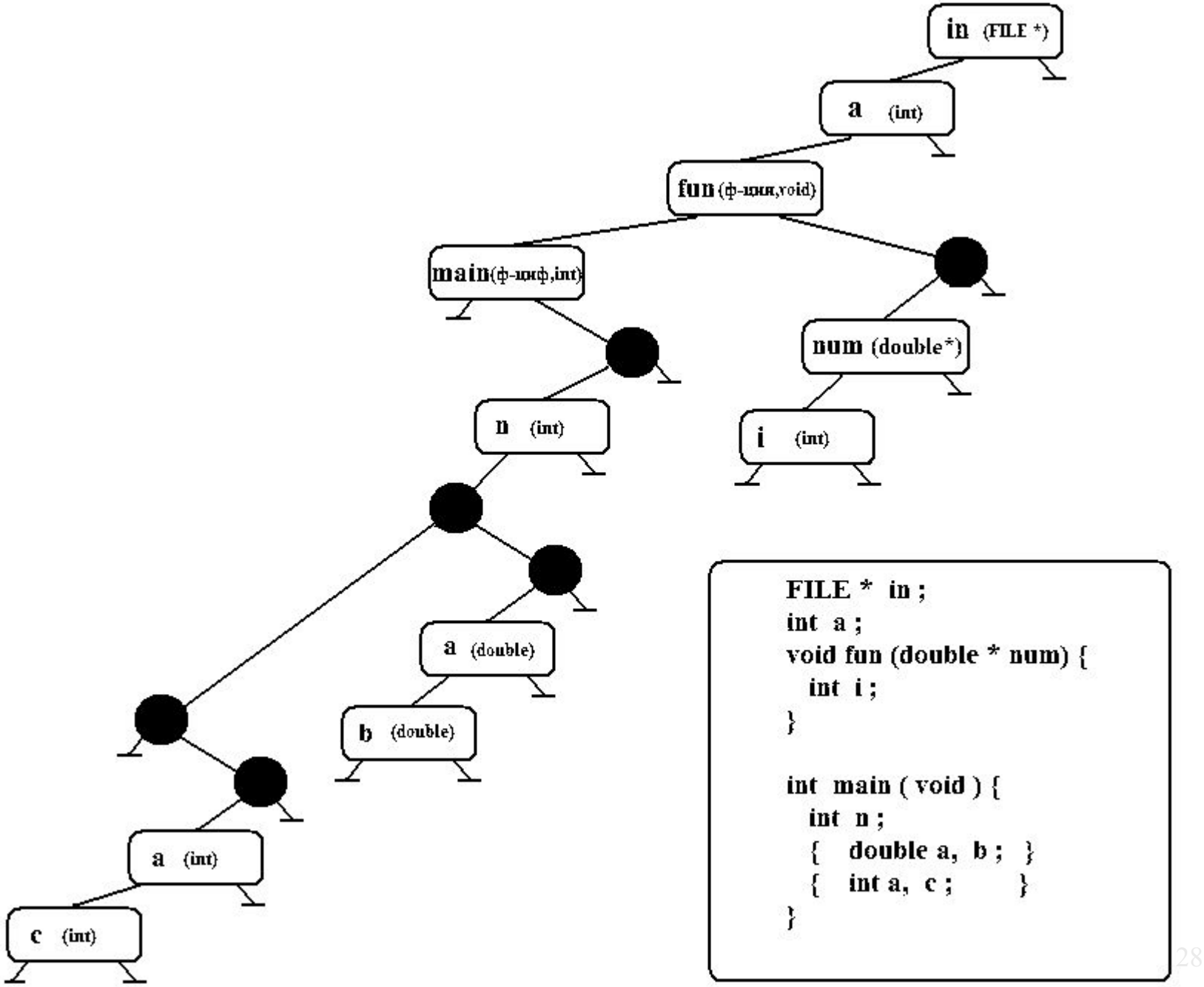
Класс «семантическое дерево»

```
class Tree   {
private:
     Node * n;          // данные таблицы
      Tree * Up, * Left, * Right;
                    // родитель, левый и правый потомок
  public:
    Tree(Tree * l, Tree * r, Tree * u, Node * Data);
     Tree();
      ~Tree();
       // необходимые функции  работы с таблицей:
.....
};
```

Функции поиска и дополнения класса «семантическое дерево»

```
class Tree   {
…..
  void SetLeft (Node * data);
  void SetRight(Node * data);
  Tree *  FindUp  (Tree * from, TypeLex   id);
  Tree *  FindUp  (TypeLex    id);
    Tree * Find UpOneLevel ( TypeLex   id);
  Tree *  FindDownLeft  (Tree * from, TypeLex   id);
  Tree *  FindDownLeft  (TypeLex   id);
  …
  void Print(Tree *   from);
  };
```

Особенности реализации семантического дерева: пустые вершины


in (FILE *)
a (int)
fun (ф-ция,void)
main(ф-ция,int)
num (double*)
n (int)
i (int)
a (double)
b (double)
a (int)
c (int)
FILE * in ;
int a ;
void fun (double * num) {
int i ;
}
int main (void) {
int n ;
{ double a, b ; }
{ int a, c ; }
}

Семантические подпрограммы

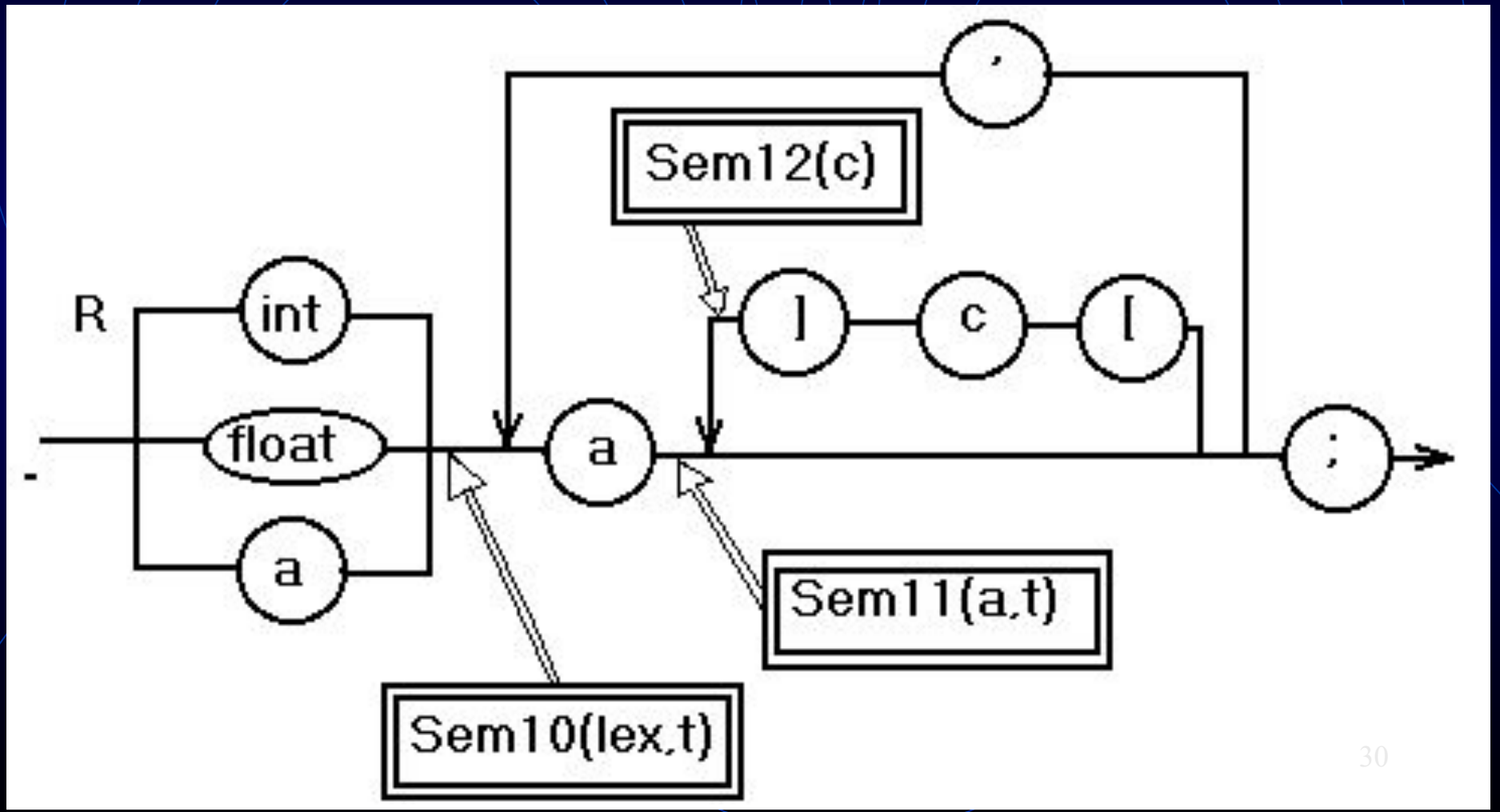
Семантика языка программирования - это контекстные условия, то есть некоторые условия, связанные с обрабатываемым текстом программы.

Следовательно, семантический контроль связан с синтаксисом: в том или ином синтаксическом окружении вызываются определенные фунецииЮ которые эти условия проверяют .

Такая связь реализуется ***семантическими подпрограммами***.

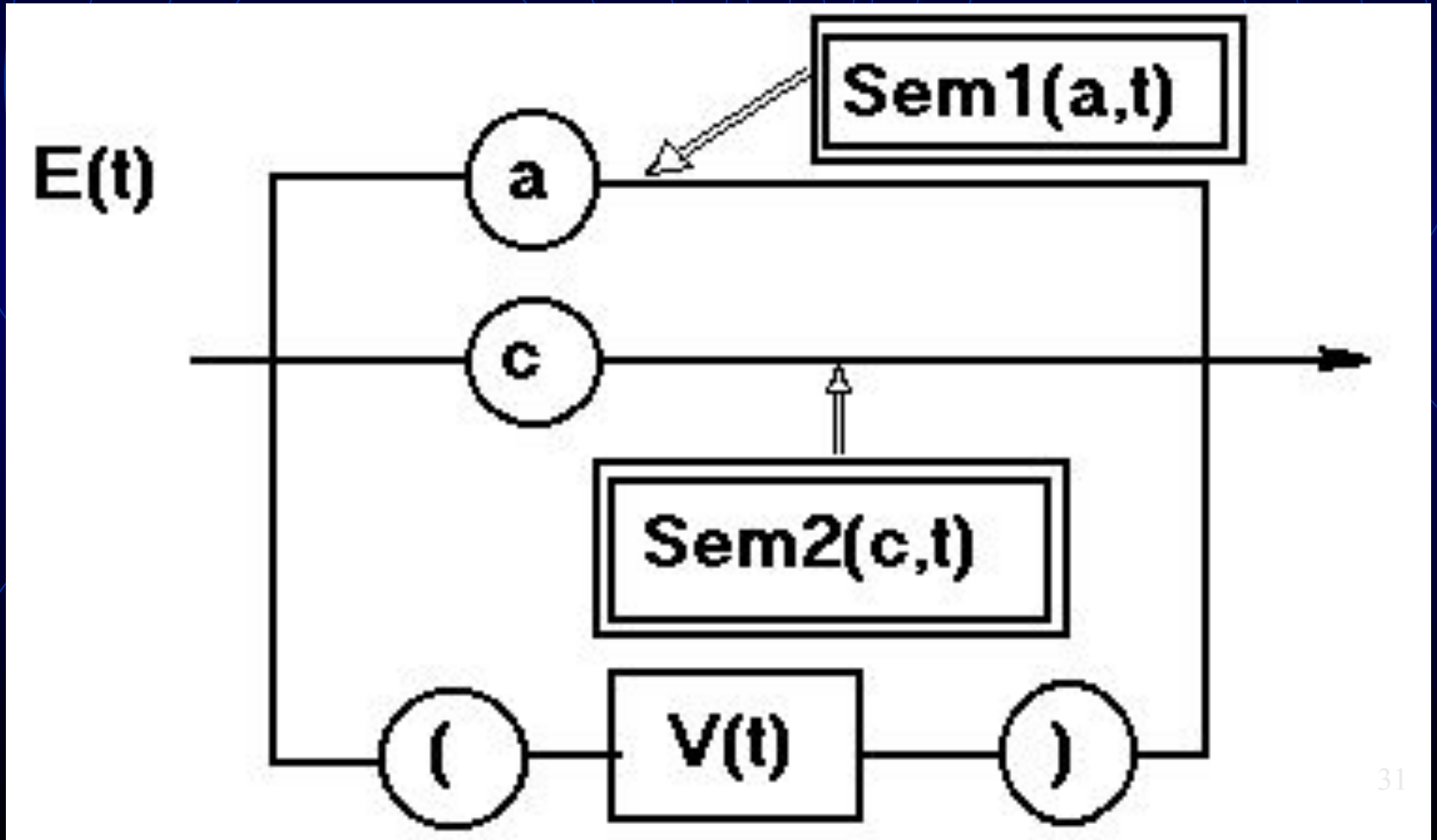
Семантика описаний

Цель - занести объект в таблицу со всеми параметрами



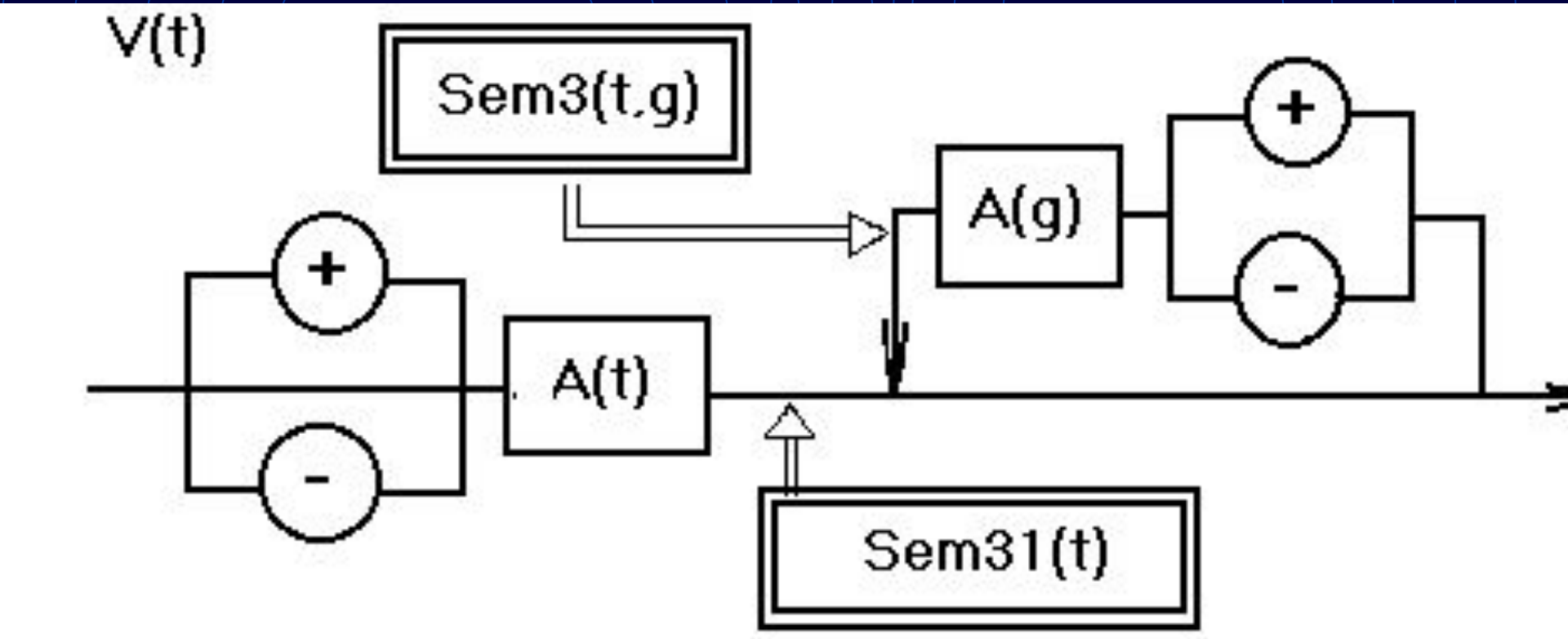
Семантика выражений

Цель – вычислить тип



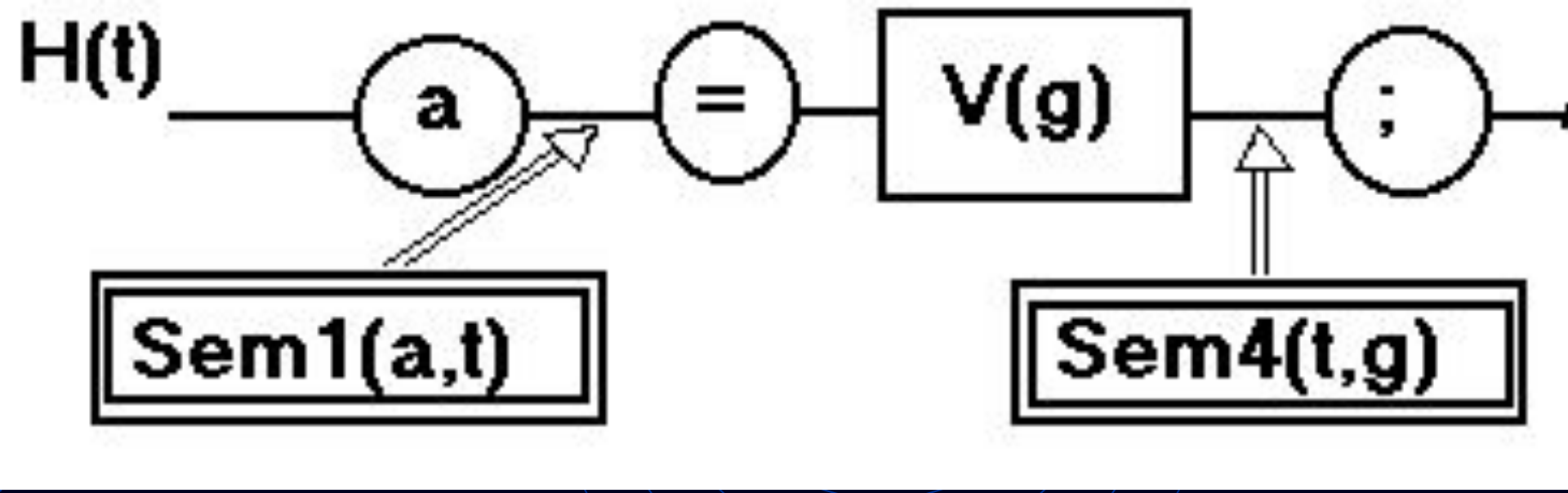
Семантика выражений

Цель – вычислить тип и проверить допустимость операции



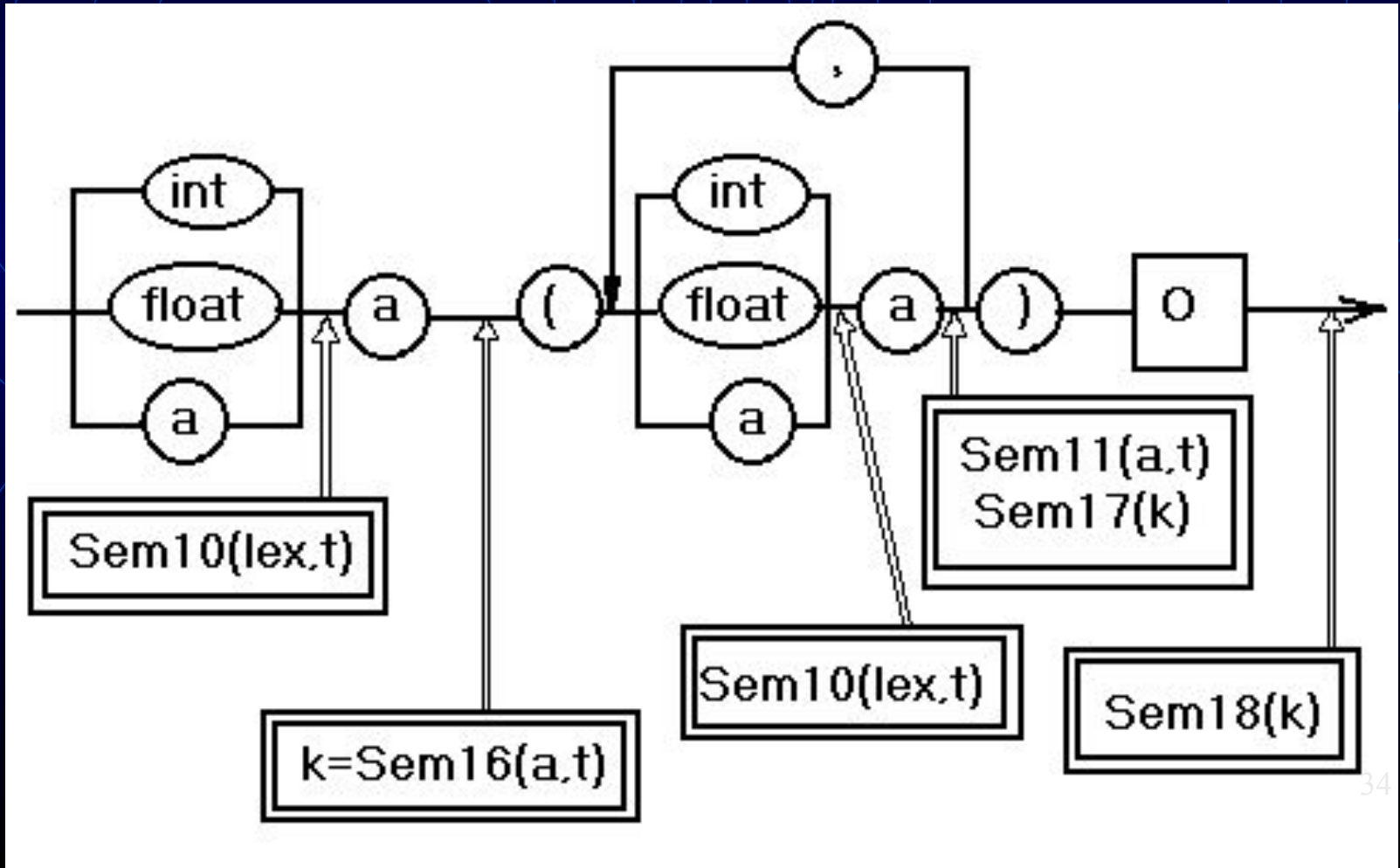
Семантика присваиваний

Цель – проверить допустимость операции



Семантика функций

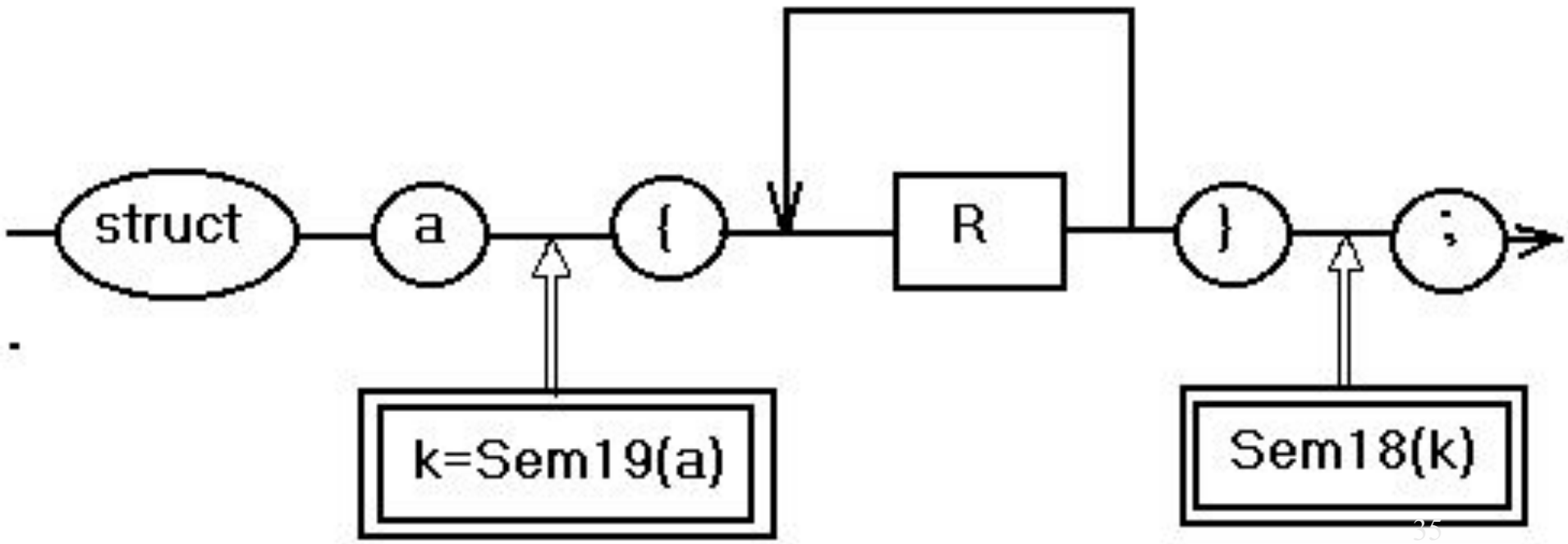
Цель при объявлении функции – занести поддерево функции в текщее семантическое дерево



Семантика классов и структур

Как и для функций цель оьработки семантики структур

- занести поддерево в общее семантическое дерево



Семантика классов и структур

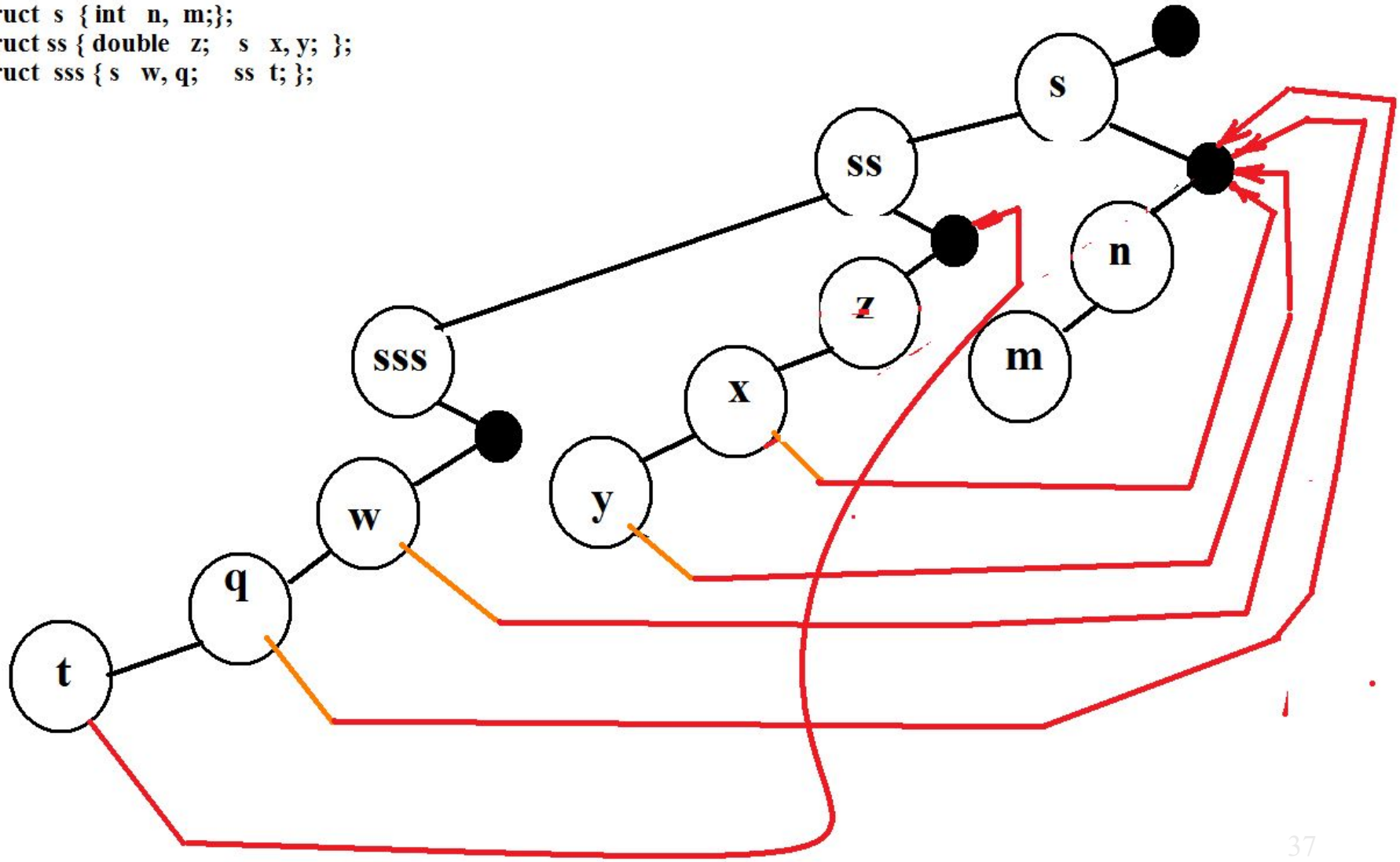
Дерево классов и структур имеет сложную структуру.

Пример:

```
    // типы
struct  s   { int   n,  m;};
struct ss { double   z;    s   x, y;  };
struct  sss { s   w, q;     ss  t; };

        //  данные
  s  a,  b;
   ss   c,  d;
   sss  f,  g;
```

Семантика классов и структур

```
struct  s  { int   n,  m;};
struct ss { double   z;    s   x, y;  };
struct  sss { s   w, q;     ss  t; };
```



Семантика классов и структур

Дерево классов и структур имеет сложную структуру вообще говоря, дерево становится потоковым графом:

```
struct s { int n, m;};
struct ss { double z; s x, y; };
```

Вершины x и y в качестве правых ссылок имеют указатели на правого потомка вершины s.

```
struct sss { s w, q; ss t; };
```

Вершина t в качестве правой ссылки имеет указатель на правого потомка вершины ss.

Семантика классов и структур

Для данных указатели строятся аналогично

```
s  a,  b;    // ссылки на правых потомков s
ss   c,  d; // ссылки на правых потомков ss
sss  f,  g; // ссылки на правых потомков sss
```

Корректность полей проверяется обходом дерева по ссылкам:

- структура верхнего уровня - поиск от текущей – (последней созданной) вершины вверх по дереву,
- каждая точка (стрелка) - переход по правой ссылке,
- каждое очередное поле - поиск по левым указателям вниз.

Лабораторные работы 7, 8, 9

Общая задача – реализовать семантический контроль

Работа 7 – проектирование
Работы 8 и 9 - реализация

Лабораторная работа № 7

Тема : Анализ контекстных условий ЯП

.Перечислить типы данных реализуемого ЯП (например, int, double, …) с указанием диапазона значений и длины памяти для хранения значения

. Построить таблицу приведения типов

.Перечислить типы объектов программы (например, функции, простые переменные, структуры, …)

.Привести список контекстных условий для каждого типа объектов с привязкой к соответствующей СД

.Построить перечень данных, которые должны храниться в семантическом дереве для каждого типа объектов

Лабораторная работа № 8

Тема - реализация создания семантического дерева.

1. Создать заголовочный файл семантического класса.
2. Реализовать семантические функции, связанные с созданием семантического дерева. Это конструкции описания данных и типов.
3. Особое внимание обратить на ошибки дублирования идентификаторов в одной области видимости, а так же возврат на предшествующий уровень после завершения блока.

Лабораторная работа № 9

Тема – полный семантический контроль.

1. Реализовать функции контроля использования объектов.
2. Обратить внимание на приведение типов в выражениях.
3. Проверить работоспособность на всех требованиях, выделенных в работе № 6.

Сообщения о семантических ошибках
и
нейтрализация семантических ошибок

Ошибка «неверное приведение типов»

Возможен повтор сообщения об ошибке при дальнейшей обработке выражения.

Нейтрализация:

1. внести новы семантический тип noType,
2. Все приведения с типом noType считаются корректными. Тип результата операции - noType.

Ошибка «нет описания идентификатора»

Возможен повтор сообщения об ошибке, если забыли описать идентификатор, но правильно использовали.
Нейтрализация:

1. внести идентификатор сразу после корня с типом noType,
2. Все операции и описания данных с с типом noType считаются корректными.

Недостаток - отсутствие ошибок при фактической ошибке в написании идентификатора при использовании.

Лабораторная работа № 10

Тема – нейтрализация семантических ошибок.

1. Предложить способы нейтрализации ошибок, выявленных в работе № 6
2. Реализовать нейтрализацию
3. Проверить работоспособность.