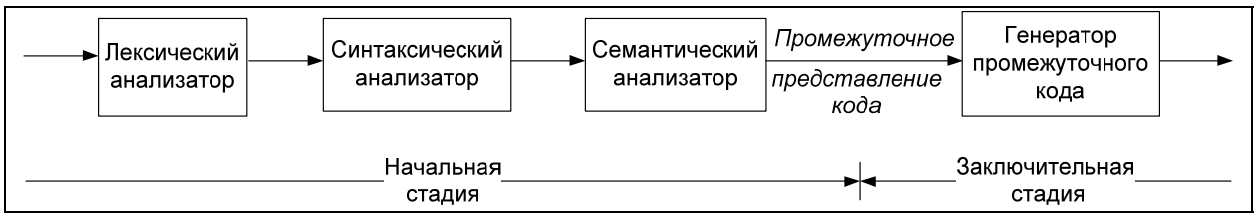
Лекция 23

БГТУ, ФИТ, ПОИТ, 3 семестр, Языки программирования

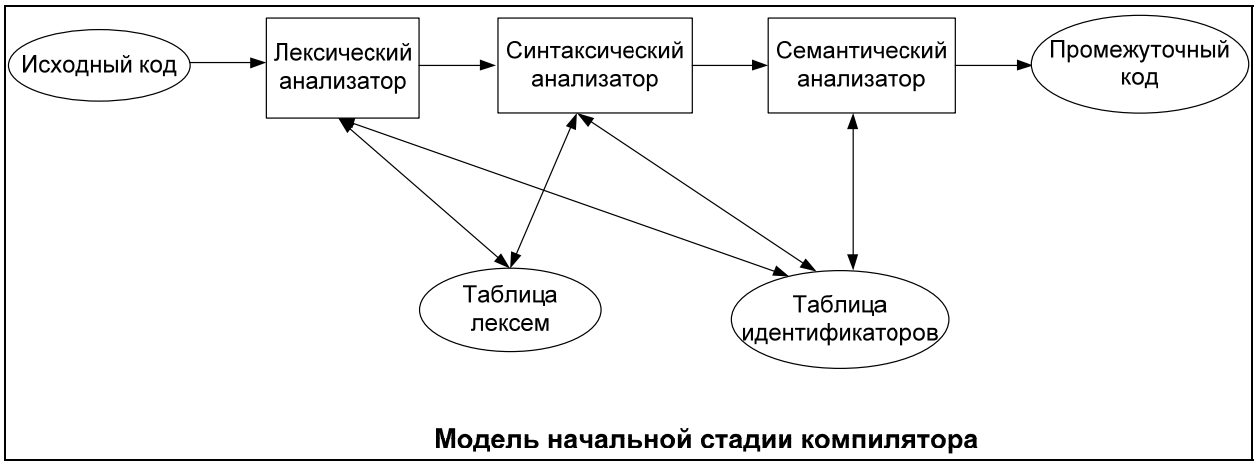
Семантический анализ и подготовка к генерации кода

1. Логическая структура транслятора



Начальная, анализирующая часть транслятора (или front end), отвечает за лексический, синтаксический и семантический анализ исходной программы и порождает промежуточное представление исходного кода. При изменении входного языка фронтальная часть может быть заменена независимо от других частей компилятора.

Заключительная, синтезирующая часть (back end) не зависит от входного языка и может быть изменена для другой целевой машины.



Модель начальной стадии компилятора

На начальной стадии компилятора анализируется исходная программа и создается промежуточное представление, из которого на заключительной стадии генерируется целевой код.

2. **Семантический анализ и подготовка к генерации кода:** назначение семантического анализа, этапы семантического анализа.

Назначение семантического анализа – проверка смысловой правильности конструкций языка программирования.

Входные данные для семантического анализатора:

- таблица идентификаторов;
- дерево разбора – результат разбора синтаксических конструкций входного языка.

Основные действия семантического анализатора:

1) проверка соблюдения в исходной программе семантических правил входного языка;
2) дополнение внутреннего представления программы в компиляторе операторами и действиями, неявно предусмотренными семантикой входного языка;
3) проверка элементарных семантических (смысловых) норм языка программирования.

1). Проверка соблюдения *семантических правил* входного языка – ***сопоставление*** входных цепочек программы с требованиями семантики входного языка программирования.

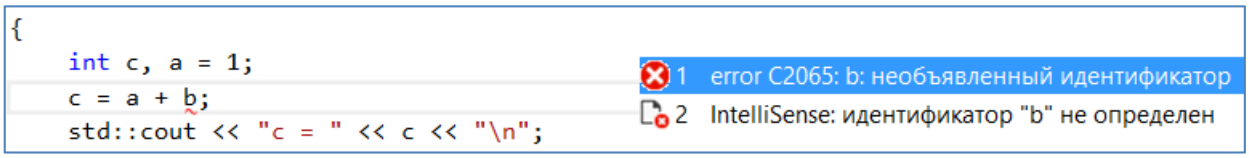
Примеры семантических правил:

- каждый идентификатор должен быть объявлен только один раз (с учетом блочной структуры объявлений);
- все операнды в выражениях и операциях должны иметь типы, допустимые для данного выражения или операции;
- типы переменных в выражениях должны быть согласованы между собой;
- при вызове процедур и функций количество и типы фактических параметров должны быть согласованы с количеством и типами формальных параметров.

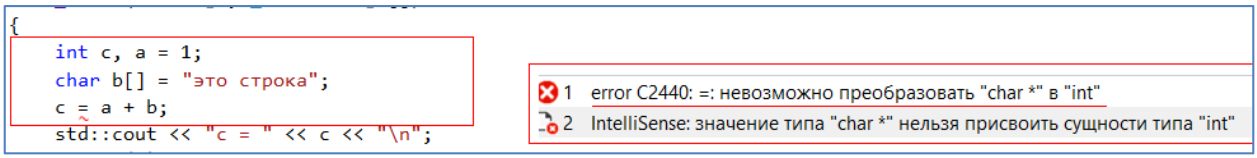
Пример. Оператор языка C++:

```
a = b + c;
```

1) Если хотя бы один из идентификаторов не объявлен, то это ошибка:



2) Не допускается, чтобы один из операндов был числовыми, а другой — строковым:



Построение таблицы идентификаторов (ТИ)

Требования:

- структура таблицы идентификатров должна обеспечивать эффективность поиска и вставки в таблицах;
- структура таблицы должна обеспечивать возможность динамического роста объема таблицы.

Необходимые действия:

- поиск имени в таблице идентификатров, соответствующего текущей лексеме, определяющей идентификатор;
- если такое имя найдено, то выдается сообщение об ошибке, в противном случае имя заносится в таблицу со значениями атрибутов.

Использование таблицы идентификаторов

Действия при повторном использовании имени в тексте программы:

- осуществляется поиск имени в ТИ;
- если имя найдено, то сопоставляется текущая семантика с указанной в ТИ (анализируются значения атрибутов, задающих контекст);
    - при отсутствии противоречий происходит дальнейший разбор;
    - в противном случае выдается сообщение об ошибке;
- если имя не найдено в ТИ, то формируется сообщение об ошибке.

2). *Дополнение внутреннего представления программы операторами* и действиями неявно предусмотренными семантикой входного языка.

Инструкции языка C++	Выполняемые действия
a = b + c;	- операция сложения; - операция присваивания результата сложения.

int c = 1;
float b = 2.5;
double a;
a = b + c;
	- преобразование целочисленной переменной **c** в формат чисел с плавающей точкой;
	- сложение двух чисел с плавающей точкой;
	- преобразование результата в число с плавающей точкой удвоенной точности;
	- присвоение результата переменной **a**.

3). *Проверка элементарных смысловых норм* языков программирования.

Примеры соглашений:

- каждая переменная или константа должна хотя бы один раз использоваться в программе;
- каждая переменная должна быть определена до ее первого использования при любом ходе выполнения программы;
- переменной должно всегда предшествовать присвоение ей какого-либо значения;
- результат функции должен быть определен при любом ходе ее выполнения;
- каждый оператор в исходной программе должен иметь возможность хотя бы один раз выполниться;
- операторы условия и выбора должны предусматривать возможность пути выполнения программы по каждой из своих ветвей;
- операторы цикла должны предусматривать возможность завершения цикла.

Пример.

Найдите в программе «неточности»:

```
int f_test(int a) {
    int b, c;
    b = 0;
    c = 0;
    if (b = 1) { std::cout << "a = " << a << "\n"; return a; }
    c = a + b;
    std::cout << "c = " << c << "\n";
    }
int _tmain(int argc, _TCHAR* argv[])
{
    f_test(3);
    system("pause");
    return 0;
}
```

Сообщения компилятора (уровень предупреждений 4 (/W4)):

```
1>------ Сборка начата: проект: L22, Конфигурация: Debug Win32 ------
1>  L22.cpp
1>d:\adel\lplab\l22\l22\l22.cpp(15): warning C4100: argv: неиспользованный формальный параметр
1>d:\adel\lplab\l22\l22\l22.cpp(15): warning C4100: argc: неиспользованный формальный параметр
1>d:\adel\lplab\l22\l22\l22.cpp(11): warning C4706: назначение в пределах условного выражения
1>d:\adel\lplab\l22\l22\l22.cpp(14): warning C4715: f_test: значение возвращается не при всех путях выполнения
1>  L22.vcxproj -> D:\Adel\LPLab\L22\Debug\L22.exe
========== Сборка: успешно: 1, с ошибками: 0, без изменений: 0, пропущено: 0 ==========
```

Результат выполнения:

```
a = 3
Для продолжения нажмите любую клавишу . . .
```

Семантика учебного компилятора для языка программирования svv-2015:

№	Правило
1	Наличие функции main
2	Усечение слишком длинных идентификаторов до 5 символов
3	Сначала осуществляется проверка на ключевые слова, а затем на идентификатор. Не допускаются идентификаторы совпадающие с ключевыми словами
4	Нет повторяющихся наименований функций
5	Нет повторяющихся объявлений идентификаторов
6	Предварительное объявление, применяемых функций
7	Предварительное объявление, применяемых идентификаторов.
8	Соответствие типов формальных и фактических параметров при вызове функций
9	Усечение слишком длинного значения string-литерала
10	Округление слишком большого значения integer-литерала
11	Если ошибка возникает на этапе лексического анализа, синтаксический анализ не выполняется
12	При возникновении ошибки в процессе лексического анализа, ошибочная фраза игнорируется (предполагается, что ее нет) и осуществляется попытка разбора следующей фразы. Граница фразы, любой сепаратор (пробел, скобка, запятая, точка с запятой и пр.)
13	Если 3 подряд фразы не разобраны, то работа транслятора останавливается
14	При возникновении ошибки в процессе синтаксического анализа, ошибочная фраза игнорируется (предполагается, что ее нет) и осуществляется попытка разбора следующей фразы. Граница фразы – точка с запятой.

3. **Место семантического анализатора в процессе компиляции**

Семантический анализ обычно выполняется частично на этапе лексического анализа, на этапе синтаксического разбора и вначале этапа подготовки к генерации кода.

Семантический анализ может быть выделен в отдельную фазу компиляции и выполняться после завершения фазы синтаксического разбора. В этом случае выполняется полный семантический анализ программы.

Дерево синтаксического разбора

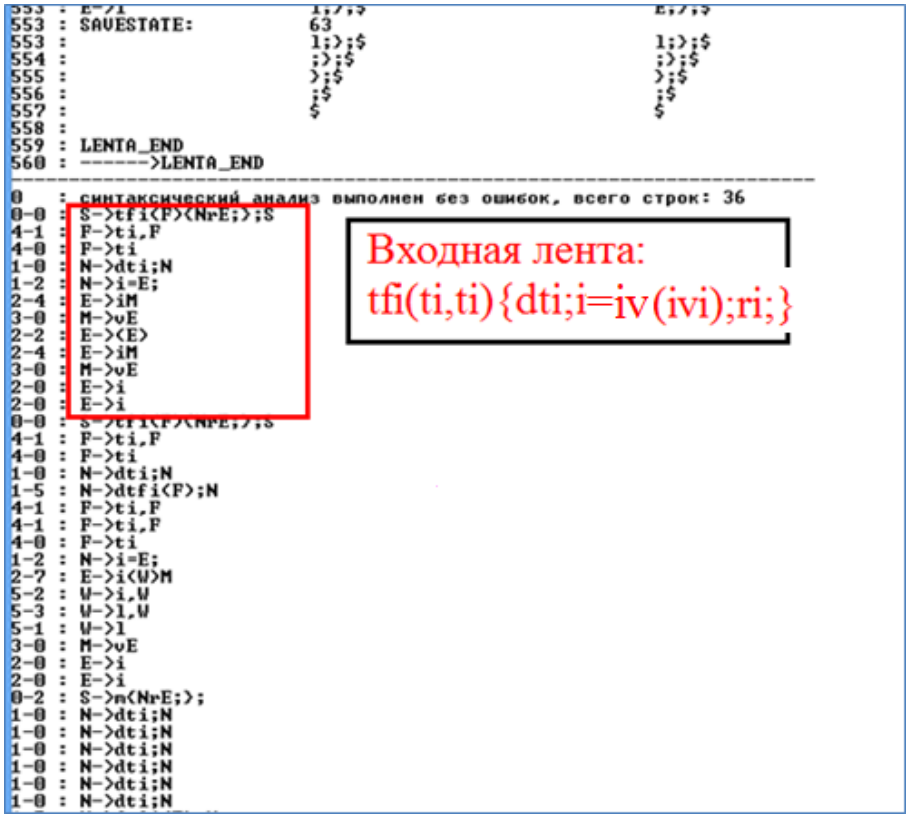
Фрагмент контрольного примера:

integer function fi(integer x, integer y)
{
declare integer z;
z= x*(x+y);
return z;
};
	tfi(ti,ti){dti;i=iv (ivi);ri;};

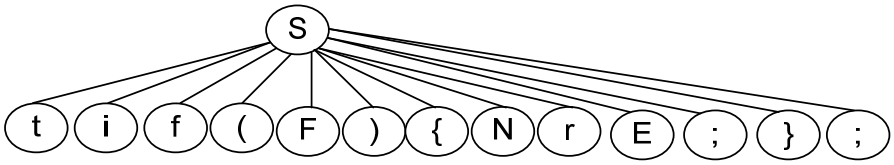
Грамматика:

Правила КС грамматики		Эквивалентная грамматика в нормальной форме Грейбах
S→C;|CS; C→tfi(F){B};|m{B}; B→NrE; N→O|ON O→dti;|rE;|i=E;|dtfi(F); E→i|l|(E)|EvE|i(W) F→ti|ti,F W→i|l|i,W|l,W	=>	1) S→ m{NrE;}; | tfi(F){NrE;};S | m{NrE;};S 2) N→ dti;|rE;|i=E;|dtfi(F);|dti;N|rE;N|i=E;N|dtfi(F);N 3) E→i | l | (E) | i(W) | iM | lM | (E)M | i(W)M 4) M→vE | vEM 5) F→ti | ti,F 6) W→i | l | i,W | l,W

Последовательность правил грамматики

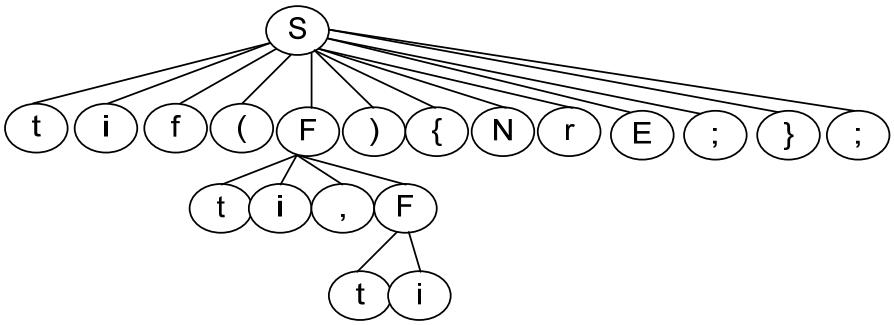


Дерево разбора:



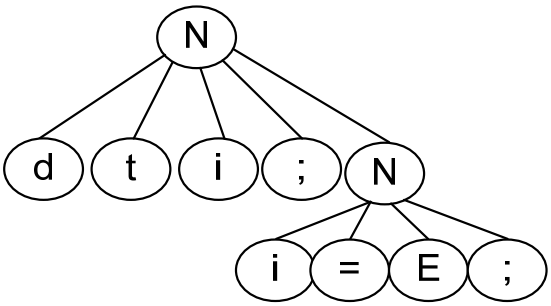
(1.2)
<=>

tfi(ti,ti){dti;i=iv(ivi);ri;};



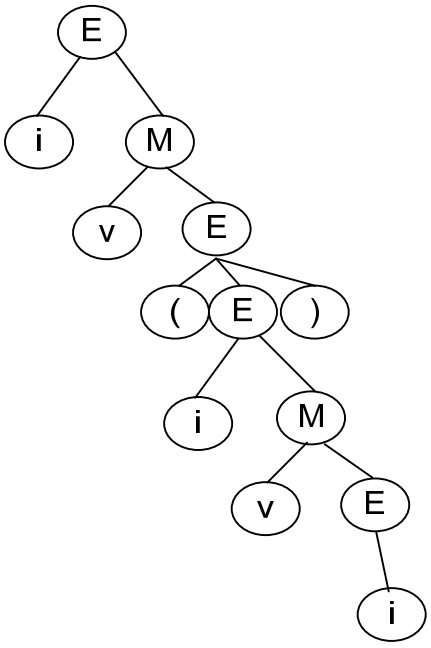
(5.2)
(5.1)
<=>

ti,ti){dti;i=iv(ivi);ri;};



(2.5)
(2.3)
<=>

dti;i=iv(ivi);ri;};



(3.5)
(4.1)
(3.3)
(3.5)
(4.1)
(3.1)
<=>

i=iv(ivi);ri;};

Статические семантические проверки – это проверки, которые могут быть проведены до выполнения кода**.**

Такие проверки могут быть реализованы путем обхода дерева в глубину и использования информации из таблицы идентификаторов.

Динамические семантические проверки производятся во время выполнения (или интерпретации) программы.

Например, проверка того, что нет деления на ноль или, что индекс массива не выходит за допустимые пределы и другое.

Основные функции семантического анализатора:

- ***дополнение таблиц идентификаторов.*** ТИ формируется на этапе лексического анализа, где в нее помещаются все уникальные имена, распознанные сканером. Для каждого имени заносятся все данные, полученные из текста программы (тип идентификатора, тип значений и т.д.). Во время семантического анализа ТИ может быть дополнена необходимой информацией;
- ***выделение неявно заданной информации.*** В представлении программ некоторые данные об элементах программы не указаны явно (например, тип переменной может определяться по первому символу имени);
- ***обнаружение ошибок.*** Синтаксический анализ определяет корректность отдельных конструкций и программы в целом с точки зрения формальных правил грамматики используемого языка, но здесь могут быть ошибки (не согласованы типы правой и левой частей оператора присваивания, несколько одинаковых меток и т.д.).
- ***проверка и выполнение некоторых операций программы:*** присваивание начальных значений; действия с константами.