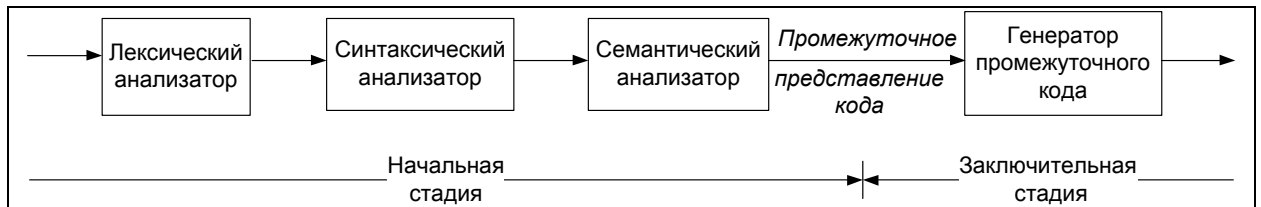


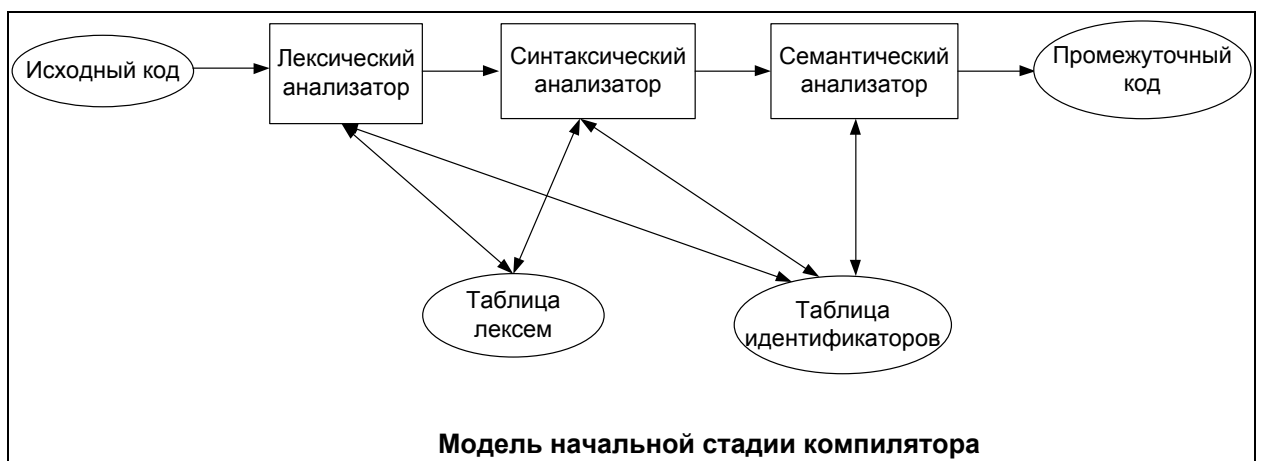
Семантический анализ и подготовка к генерации кода

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.



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 число и типы фактических параметров должны быть согласованы с числом и типами формальных параметров.

Пример. Оператор языка C++:

a = b + c;

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

```
{
    int c, a = 1;
    c = a + b;
    std::cout << "c = " << c << "\n";
}
```

1 error C2065: b: необъявленный идентификатор
2 IntelliSense: идентификатор "b" не определен

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

```
{
    int c, a = 1;
    char b[] = "это строка";
    c = a + b;
    std::cout << "c = " << c << "\n";
}
```

1 error C2440: =: невозможно преобразовать "char *" в "int"
2 IntelliSense: значение типа "char *" нельзя присвоить сущности типа "int"

Построение таблицы идентификаторов (ТИ)

Требования:

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

Необходимые действия:

- поиск имени в таблице идентификаторов, 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
<code>a = b + c;</code>	<ul style="list-style-type: none">– операция сложения;– операция присваивания результата сложения.
<code>int c = 1; float b = 2.5; double a; a = b + c;</code>	<ul style="list-style-type: none">– преобразование целочисленной переменной <code>c</code> в формат чисел с плавающей точкой;– сложение двух чисел с плавающей точкой;– преобразование результата в число с плавающей точкой удвоенной точности;– присвоение результата переменной <code>a</code>.

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:\Ade1\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:

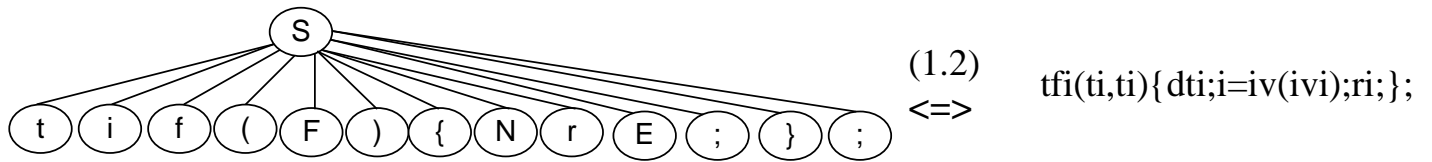
Правила КС грамматики	Эквивалентная грамматика в нормальной форме Грейбах
$S \rightarrow C; CS;$ $C \rightarrow tfi(F)\{B\}; m\{B\};$ $B \rightarrow NrE;$ $N \rightarrow O ON$ $O \rightarrow dti; rE; i=E; dtfi(F);$ $E \rightarrow i l (E) EvE i(W)$ $F \rightarrow ti ti, F$ $W \rightarrow i l i, W l, W$	\Rightarrow <ol style="list-style-type: none"> $S \rightarrow m\{NrE;\}; tfi(F)\{NrE;\};S m\{NrE;\};S$ $N \rightarrow dti; rE; i=E; dtfi(F); dti;N rE;N i=E;N dtfi(F);N$ $E \rightarrow i l (E) i(W) iM lM (E)M i(W)M$ $M \rightarrow vE vEM$ $F \rightarrow ti ti, F$ $W \rightarrow i l i, W l, W$

Последовательность правил грамматики

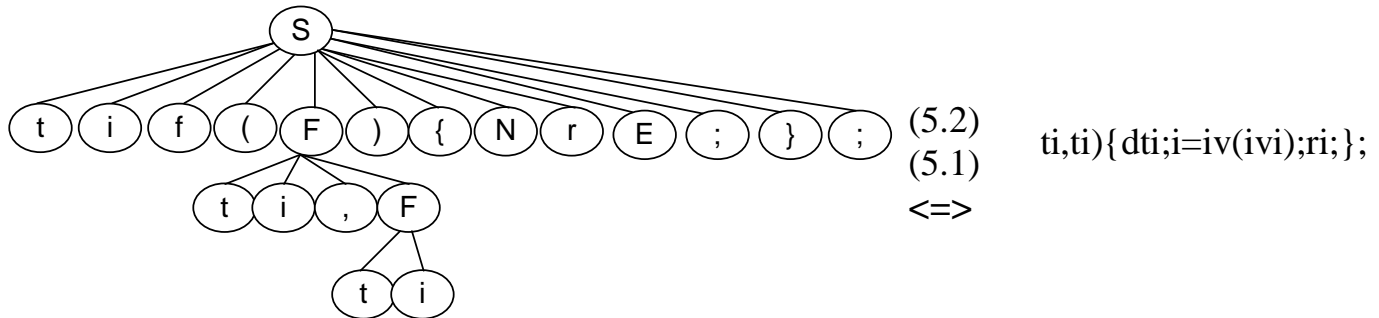
<pre> 553 : E->i 553 : SAVESTATE: 553 : 1;,>,\$ 554 : 1;,>,\$ 554 : 1;,>,\$ 555 : 1;,>,\$ 555 : 1;,>,\$ 556 : 1;,>,\$ 556 : 1;,>,\$ 557 : 1;,>,\$ 557 : 1;,>,\$ 558 : 1;,>,\$ 558 : 1;,>,\$ 559 : LENTA_END 560 : ----->LENTA_END </pre>	<pre> 0 : синтаксический анализ выполнен без ошибок, всего строк: 36 0-0 : S->tfi(F)<NrE;>;S 4-1 : F->ti,F 4-0 : F->ti 1-0 : N->dti;N 1-2 : N->i=E; 2-4 : E->iM 3-0 : M->vE 2-2 : E->(E) 2-4 : E->iM 3-0 : M->vE 2-0 : E->i 2-0 : E->i 0-0 : S->tfi(F)<NrE;>;S 4-1 : F->ti,F 4-0 : F->ti 1-0 : N->dti;N 1-5 : N->dtfi(F);N 4-1 : F->ti,F 4-1 : F->ti,F 4-0 : F->ti 1-2 : N->i=E; 2-7 : E->i(W)M 5-2 : W->i,W 5-3 : W->l,W 5-1 : W->l 3-0 : M->vE 2-0 : E->i 2-0 : E->i 0-2 : S->m(NrE;); 1-0 : N->dti;N 1-0 : N->dti;N 1-0 : N->dti;N 1-0 : N->dti;N 1-0 : N->dti;N 1-0 : N->dti;N </pre>
---	--

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

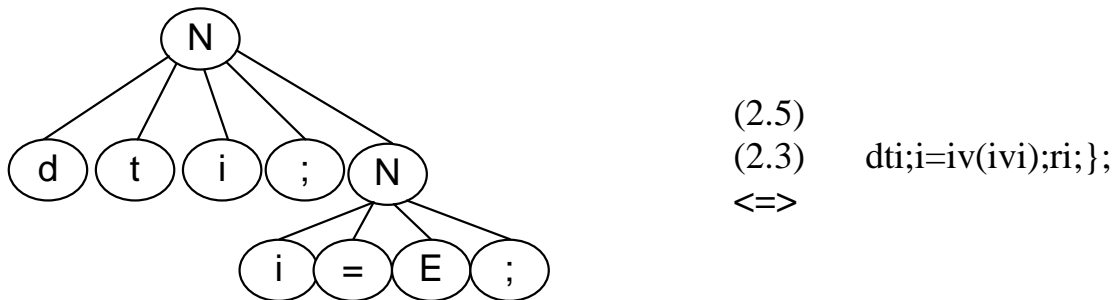
Дерево разбора:



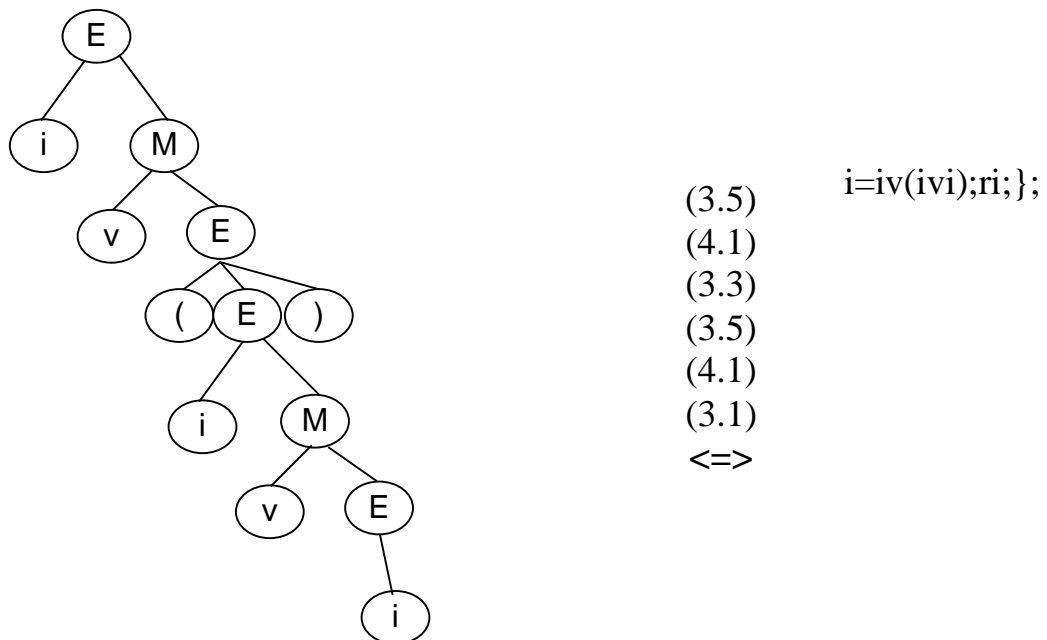
$t f i(t i, t i)\{d t i ; i=i v(i v i) ; r i ;\} ;$



$t i, t i\{d t i ; i=i v(i v i) ; r i ;\} ;$



$d t i ; i=i v(i v i) ; r i ;$



$i=i v(i v i) ; r i ;$

Статические семантические проверки – это проверки, которые могут быть выполнены до выполнением кода. Такие проверки могут быть реализованы путем обхода дерева в глубину и использования информации из таблицы идентификаторов.

Динамические семантические проверки – проверки 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.