**Включение действий в синтаксис**

Синтаксический анализатор отвечает на вопрос о принадлежности входной цепочки языку. В ходе распознавания выявляется структура входного текста. Эта структура может быть представлена явно, например в виде дерева, или неявно — последовательностью действий, совершенных распознавателем.

На основе распознавания структуры входного текста строится и его содержательная обработка, трансляция.

 Синтаксический анализатор служит основой, остовом транслятора, предоставляя возможность выполнить необходимые действия по смысловой (семантической) обработке в нужные моменты в соответствии со структурой входной цепочки.

**Семантические процедуры**

Встраиваемые в распознаватель действия, предназначенные для выполнения смысловой обработки входного текста, будем называть *семантическими процедурами*.

Слово «процедура» употребляется здесь в широком смысле, как определенная последовательность действий. В программе-распознавателе это может быть не обязательно процедура-подпрограмма, но и просто один или несколько операторов.

Рассмотрим использование семантических процедур на простом примере.

Пусть речь идет о распознавателе целых чисел без знака, который был использован в программе-анализаторе многочленов (см. лекция\_№7,рис.23).

Теперь в его задачу будет входить получение значения числа, представленного последовательностью цифр.

Обозначив семантические процедуры Р1 и Р2 разместим на синтаксической диаграмме «Целое» (рис. 29) соответствующие им треугольные значки в тех местах, при прохождении которых во время анализа эти процедуры должны выполняться.



*Рис. 29. Семантические процедуры на диаграмме целого без знака*

Процедура Р1 будет выполняться в начале обработки и присвоит искомому числу исходное нулевое значение. Переменная у — это формируемое значение числа.

Р1: у := 0;

Задача Р2 — добавлять к числу справа очередную цифру. Для этого «старое» значение у умножим на 10 (основание десятичной системы счисления) и добавим значение прочитанной цифры.

 Условимся, что семантическая процедура, значок которой размещен перед обозначением терминального символа, выполняется в тот момент, когда этот символ является текущим (содержится в переменной ch).

Надо побеспокоиться и о том, чтобы при вычислениях не произошло переполнения — то есть не получилось число, превышающее максимально допустимое целое.

Р2: d := ord(Ch) - ord('0') ;

 **if** у <= (maxint - d) **div** 10 **then**

 у := 10\*y + d

 **else**

 Error('Слишком большое число');

Понятно, что выполнять такую проверку

 **if** 10\*y + d <= maxint **then** ...

было бы, как минимум, наивно — переполнение случится еще до того, как дело дойдет до сравнения.

Теперь можно написать распознающую процедуру, выполняющую не только синтаксический анализ, но и получение числового значения целого, которое будет выходным параметром этой процедуры.

**procedure** Number(**var** у : integer); { Целое }

**var**

 d : integer;

**begin**

 y := 0; { P1 }

 **if** **not**( Ch **in** [' 0 ' . . ' 9 ' ] ) **then**

 Error('Число начинается не с цифры');

 **repeat**

 d := ord(Ch) - ord('0'); { P2 }

 **if** у <= (maxint - d) **div** 10 **then** { P2 }

 у := 10\*y + d { P2 }

 **else** { P2 }

 Error('Слишком большое число'); { P2 }

 NextCh;

 **until not**( Ch in [' 0 ' ..'9'] );

**end;**

Напомним, что процедура *Error* останавливает работу всего анализатора, поэтому не надо беспокоиться, что при возникновении ошибки в цикле работа этого цикла разладится. В дальнейшем мы обсудим другие варианты обработки ошибок, но пока такое соглашение позволяет не загромождать транслятор дополнительными проверками.

Можно еще отметить, что ошибка «Слишком большое число» уже не синтаксическая. Она связана с содержательной обработкой и вправе называться семантической.

**Пример: умножение многочленов**

Составить программу, вводящую в символьной форме два многочлена от х с целыми коэффициентами и выводящую их произведение в символьной форме в порядке убывания степеней. Суммарная степень многочленов не превышает 255.

Вот пример работы такой программы:

Перемножение многочленов

-------------------------------------------

1-й многочлен:

 3x^2+3x-5

2-й многочлен:

 х^З - 2х

Произведение равно:

 Зх^5 + Зх^4 - 11х^3 – 6х^2 + 10х

Программа печатает запрос, в ответ на который можно ввести запись первого, а затем второго многочлена в привычной, используемой в математике форме. Обработав полученные данные, программа печатает результат — произведение многочленов в таком же естественном виде.

**Проектирование**

Приступим к решению поставленной задачи. Сформулируем общий план:

1.Напечатать заголовок.

2.Ввести текст 1-го многочлена, проанализировать его запись и преобразовать в удобную для дальнейших вычислений форму. Для выполнения действий с многочленами (в том числе перемножения) удобно хранить их в программе в виде массива коэффициентов, количество которых соответствует степени многочлена, а индекс каждого коэффициента равен степени соответствующего слагаемого.

3. Ввести текст 2-го многочлена, проанализировать его запись и преобразовать в удобную для дальнейших вычислений форму.

4. Перемножить многочлены. Имея представление обоих многочленов - сомножителей во внутреннем формате и выполняя необходимые вычисления с их коэффициентами, получаем коэффициенты многочлена-произведения.

5. Напечатать результат.

Запишем начало программы, в котором в соответствии с уже принятыми решениями определим необходимые константы, типы данных и переменные.

**program** PolyMult;

{ Перемножение многочленов }

**const**

 nmax =255; { Максимальная степень }

**type**

 tPoly = **record** { Тип многочленов }

 n: integer; { Степень }

 a: **array** [O..Nmax] **of** integer; { Коэффициенты }

 **end;**

**var**

 P1, P2, Q: tPoly; { Сомножители и произведение }

Обратите внимание, что многочлены отнесены к типу tPoly, который представляет собой запись, содержащую, кроме упоминавшегося массива коэффициентов, величину n — фактическую степень. Теперь, пользуясь предварительно составленным планом, запишем основную программу.

**begin**

WriteLn('Перемножение многочленов');

WriteLn('-------------------------------------------------- ' ) ;

WriteLn;

WriteLn('1-й многочлен');

GetPoly(P1);

WriteLn('2-й многочлен1);

GetPoly(P2);

{ Перемножение }

 MultPoly(P1, P2, Q);

WriteLn;

WriteLn ('Произведение:');

{ Печать результата }

 WritePoly(Q);

WriteLn;

**end.**

**Умножение и вывод**

Детализацию начнем с процедуры, которая выполняет перемножение. Ее входными параметрами являются многочлены-сомножители, выходным — многочлен-произведение. И входные, и выходной параметры являются многочленами, представленными в виде совокупности массива коэффициентов и величины, задающей степень многочлена, то есть относятся к типу tPoly. Обозначив сомножители X и Y, а результат — Z, запишем заголовок процедуры.

**procedure** MultPoly(X, Y: tPoly; **var** Z: tPoly);

Основная идея вычисления коэффициентов многочлена-произведения состоит в том, что при попарном перемножении слагаемых первого и второго многочленов получающееся произведение участвует (в качестве одного из слагаемых) в

формировании того слагаемого многочлена-результата, степень которого равна сумме степеней сомножителей.

То есть при перемножении i-го члена многочлена X и j-гo члена многочлена Y получается величина, которая должна быть добавлена к i+j-му слагаемому Z:

Z.a[i+j] := Z.a[i+j] + X.a[i]\*Y.a[j];

Такое вычисление нужно выполнить для всех сочетаний i и j, не забыв присвоить нулевые начальные значения коэффициентам Z и побеспокоиться об определении степени многочлена Z.

Процедура ClearPoly, выполняющая «обнуление» многочлена, выглядит так:

**procedure** ClearPoly(**var** P : tPoly);

**var**

 i : integer;

**begin**

 **for** i := 0 **to** nmax **do**

 P.a[i] := 0;

 Р.n := 0;

**end;**

Теперь займемся печатью многочлена. Слагаемые должны выводиться в порядке убывания степеней. Слагаемому может предшествовать знак. Коэффициент (при ненулевой степени) печатается, если он не равен 0 или 1. Буква х выводится для ненулевых степеней, а значение показателя степени (и знак «^» перед ним), — если эта степень больше единицы.

**Транслятор многочленов**

Нам осталось реализовать транслятор (процедуру GetPoly), преобразующий введенную запись многочлена в массив коэффициентов. Его задача — считывать символы из входной строки, выполнять распознавание многочлена и вычислять его коэффициенты. Распознающие процедуры будут вложены внутрь GetPoly, а сама эта процедура лишь подготовит чтение входной строки, вызовет распознаватель и убедится, что за многочленом во входной строке ничего не содержится.

{ Ввод и трансляция многочлена }

**procedure** GetPoly(**var** P: tPoly);

**const**

 EOT = chr(0); { Конец текста }

**var**

 Pos: integer; { номер символа }

 Ch : char; { очередной символ }

{ Здесь разместятся процедуры распознавателя }

**begin**

 Pos := 0;

 NextCh;

 Poly(P);

 **if** Ch <> EOT **then**

 Error('Ожидается конец текста');

**end;**

Разместим на синтаксических диаграммах значки семантических процедур и определим содержание этих процедур. Вначале на диаграмме многочлена (рис. 30).

Перед началом обработки массив коэффициентов многочлена заполняется нулями, а его степень принимается равной нулю. Эту работу выполнит семантическая процедура Р3, вызвав уже написанную нами процедуру clearPoly.

Р3: ClearPoly(P) ;

Р4 и Р5 отвечают за запоминание знака перед слагаемым. Знак сохраняется в локальной переменной Ор.

Р4: Op := Ch;

Р5: Ор := ' + ';

Каждое слагаемое многочлена имеет вид аkxk. Транслятор слагаемого вычислит значения коэффициента а и степени к. Получив эти значения, семантическая процедура Р6 в зависимости от знака добавит или отнимет значение а из k-й ячейки массива коэффициентов:

Р6: **if** Ор = ' + ' **then**

 P.a[k] := P.a[k] + а

 **else**

 P.a[k] := P.a[k] - a;

Было бы неправильно просто записывать значение коэффициента в а [к], поскольку в записи многочлена могут быть несколько членов с одинаковой степенью х — синтаксис этого не запрещает. Складывая или вычитая каждый коэффициент с предыдущей суммой, транслятор как бы приводит подобные.

Определение степени n многочлена Р выполняет семантическая процедура Р7.

P7: P.n := nmах;

 **while** (P.n > 0) **and** (P.a[P.n] = 0) **do**

 P.n := P.n-1;

Теперь, пользуясь диаграммой и вставляя в текст анализатора в соответствующих местах определенные нами семантические процедуры, можно записать распознаватель многочлена.

{ Многочлен }

**procedure** Poly(var P: tPoly);

**var**

 a : integer; { Модуль коэффициента }

 k : integer; { Степень слагаемого }

 Op : char; { Знак операции }

**begin**

 ClearPoly(P); { P3 }

 **if** Ch **in** [' + ','-'] **then begin**

 Op := Ch; { P4 }

 NextCh;

 **end**

 **else**

 Op := '+'; { P5 }

 Addend(a, k);

 **if** Op = ' + ' **then** { P6 }

 P.a[k] := P.a[k] +a { }

 **else** { }

 P.a[k] := P.a[k] - a; { }

 **while** Ch **in** ['+','-'] **do begin**

 Op := Ch; { P4 }

 NextCh;

 Addend(a, k);

 **if** Op = '+' **then** { P6 }

 P.a[k] := P.a[k] + a { }

 **else** { }

 P.a[k] := P.a[k] - a; { }

 **end;**

 P.n := nmax; { P7 }

 **while** (P.n > 0) **and** (P.a[P.n] = 0) **do** { }

 P.n := P.n-1; { }

**end;**

Задача транслятора слагаемого, как уже говорилось, состоит в определении коэффициента (точнее, модуля коэффициента) а и степени k слагаемого.

В соответствии с этим предусмотрим и необходимые семантические процедуры (рис. 31).



*Рис.31. Синтаксическая диаграмма слагаемого с семантическими процедурами*

Процедура Р8 присваивает коэффициенту а значение, равное величине целого числа, которое вычисляется транслятором целого.

Р8: а := значение целого;

В программе это будет реализовано подстановкой переменной а в качестве параметра при вызове процедуры Number.

Если коэффициент отсутствует, он принимается равным единице.

Р9: а := 1;

Значение степени либо берется равным вычисленному распознавателем степени, либо нулевым, если в записи слагаемого отсутствует х.

Р10: k := значение степени;

Р11: k := 0;

Реализация распознавателя нетерминала «Степень» не вызывает затруднений и выполняется по соответствующей синтаксической диаграмме, на которой обозначены семантические процедуры Р12 и Р13 (рис. 32). Чтобы защитить программу от ошибки при обращении к массиву коэффициентов, в процедуре Р12 предусмотрен контроль величины показателя степени. Вычисляемая степень обозначена p.



*Рис.32. Синтаксическая диаграмма степени с семантическими процедурами*

Р12: р := значение целого;

 if p > nmax then

 Error('Слишком большая степень');

Р13: р := 1;

Обратите внимание, что в случае, когда семантическая процедура (Р13) расположена на ветви диаграммы, где не было терминальных и нетерминальных блоков, она сама играет роль блока, что несколько меняет структуру программы-распознавателя (в нашем случае появляется ветвь else).

**Обработка ошибок при трансляции**

В рассмотренных примерах обработка ошибок выполнялась с помощью процедуры *Error*, которая после выдачи сообщения прекращала работу всей программы.

Такой способ реакции на ошибку упрощает синтаксический анализатор, но далеко не идеален. Можно рассмотреть несколько вариантов его усовершенствования:

- завершение работы распознавателя (транслятора) после обнаружения первой ошибки без прерывания работы вызвавшей его программы;

- продолжение работы распознавателя после обнаружения первой ошибки с целью обнаружения других ошибок.

Второй вариант реакции на ошибки используется во многих трансляторах языков программирования. В литературе можно найти немало рекомендаций по способам восстановления распознавателя после обнаружения ошибки для продолжения анализа. Однако общего решения задачи не существует. Многое зависит от конкретного языка. Качественное восстановление, обеспечивающее выдачу осмысленных сообщений, предполагает, в том числе, использование эмпирических приемов.

Обсудим и реализуем более простой первый вариант. Он совсем неплох.

В условиях, когда скорость работы компьютеров неизмеримо возросла, время, необходимое для компиляции программы, невелико, и его потери, связанные с тем, что компилятор не сообщил программисту о нескольких ошибках сразу, исчезающе малы.

Итак, необходимо организовать реакцию на ошибку таким образом, чтобы после выдачи сообщения, анализатор не останавливал всю программу, а лишь завершил свою работу, давая возможность продолжить исполнение вызвавшей его программе.

Проблема при обработке ошибок состоит лишь в том, чтобы после обнаружения ошибки завершить все процедуры, цепочка вызова которых привела к процедуре, обнаружившей ошибку.

Это можно сделать, если при обнаружении ошибки процедура *Error* установит признак ошибки и прочитает текст до конца. Текущим символом станет «конец текста».

 Поскольку этот символ не может встретиться в анализируемом тексте, он будет отвергнут всеми частями анализатора, который завершит свою работу без прерывания программы в целом.

В качестве признака ошибки разумно использовать номер ошибочного символа (обозначим ErrPos), ненулевое значение которого соответствует наличию ошибки и несет информацию о ее местоположении.

Перед началом работы анализатора до первого вызова NextCh нужно выполнить ErrPos := 0.

Исправленные процедуры Error и Nextch теперь выглядят так:

--------------------------------------------------------------------

Получаем:

{ Умножение многочленов. Z = Х\*У }

**procedure** MultPoly(X, Y: tPoly; **var** Z: tPoly);

**var**

i, j: integer;

**begin**

 ClearPoly(Z); { "Обнуление" Z }

 **for** i := 0 to X.n **do**

 **for** j := 0 to Y.n **do**

 Z.a[i+j] := Z.a[i+j] + X.a[i]\*Y.a[j];

 { Определение степени многочлена Z }

 Z.n := nmax;

 **while** ( Z.n>0 ) **and** ( Z.a[Z.n]=0 ) **do**

 Z.n := Z.n - 1;

**end;**

{ Вывод многочлена }

**procedure** WritePoly(P : tPoly);

**var**

 i : integer;

**begin**

 **with** P **do**

 **for** i := n **downto** 0 **do begin**

 **if** ( a[i]>0 ) **and** ( i<>n ) **then**

 Writer ('+ ')

 **else if** ( a[i]<0 ) **and** ( i=n ) **then**

 Write (' - ')

 **else if** a[i]<0 **then**

 Write (' - ');

if ( abs(a[i])>1 ) or

( i=0 ) and ( a[i] <> 0 ) or

( n=0 )

then

Write(abs(a[i]));

if ( i>0 ) and ( a[i]<>0 ) then

Write('x');

if ( i>1 ) and ( a[i]<>0 ) then

Write (‘^', i)

end;

end;

**procedure** Power(**var** p : integer);

**begin**

 **if** Ch = '^' **then begin**

 NextCh;

 Number(p); {P12 }

 **if** p > nmax **then** { }

 Error('Слишком большая степень1); { }

 end

 **else**

 P := 1; { P13 }

**end;**