**Интерпретация выражений**

Рассмотрим, каким образом вычисляется выражение по его записи без явного преобразования в ПОЛИЗ.

Очевидно, что можно действовать, комбинируя действия, связанные с переводом в ПОЛИЗ, с действиями, которые выполняются при вычислении выражения по его польской записи.

Вместо вывода операндов и операций в выходную строку надо помещать операнды в стек, а операции выполнять, применяя их к верхним элементам стека и возвращая на стек результат.

Способ решения задачи определен с помощью синтаксических диаграмм (рис. 36)

и семантических процедур. В отличие от предыдущих примеров, добавлена возможность записывать знак перед первым слагаемым.

Р1: Запомнить (Ch) ;

Р2: Запомнить(' + ');

Р3: Вспомнить (Op);

 **if** Op = '-' **then begin**

 Pop(x); Push(-x);

 **end;**

P4: Вспомнить (Op) ;

 Pop(x2); Pop(x1);

 **if** Op = '+' **then** Push(x1 + x2)

 **else** {Op = '-'} Push(x1 - x2);

P5: Вспомнить (Op) ;

 Pop(x2); Pop(x1);

 **if** Op = '\*' **then** Push(x1\*x2)

 **else** {Op = '/'} Push(x1/x2);

P6: Push(Значение(Ch));

**Семантическое дерево выражения**

Как уже говорилось, задача разбора состоит в построении дерева вывода (синтаксического дерева). Однако ни один из рассмотренных нами алгоритмов не строит это дерево в явном виде.

Структура входной цепочки, которую представляет дерево, лишь отражается в последовательности действий, совершаемых синтаксическим анализатором.

На практике важную роль может играть семантическое дерево.

Его построение облегчает решение многих задач трансляции. Дерево может использоваться в качестве промежуточного представления программы.

Рассмотрим построение и использование семантического дерева на примере простых арифметических выражений, которые уже неоднократно нами использовались (см. диаграммы на рис. 27,28). На рис. 37 показаны деревья трех выражений.



Рис.37. Примеры семантических деревьев.

**Представление дерева**

Для представления семантического дерева выражения будем использовать динамические переменные и указатели.

Каждая вершина дерева содержит операнд или операцию. Поскольку в нашем примере используются однобуквенные операнды a, b, с и обозначения операций, состоящие из одного символа, достаточно предусмотреть в каждой записи, соответствующей вершине дерева, поле Ch символьного типа.

type

 tPtr = ^tNode; {Тип указателя на вершину}

 tNode = record {Тип вершины дерева }

 Ch : char; {Символ вершины }

 Counter: integer; {Счетчик временных

 переменных }

 Left : tPtr; {Указатель на левое

 поддерево }

 Right : tPtr; {Указатель на правое

 поддерево}

 end;

Поля Left и Right служат для установления связей в дереве.

В нашем примере дерево будет двоичным, поскольку все операции двуместные.

На поле Counter можно пока не обращать внимания, оно потребуется позже для решения одной из задач.

На рис.38 можно видеть дерево выражения a + Ь \* с, состоящее из динамических переменных типа tNode.

Рис 38. Двоичное дерево выражения а + b\*с

**Построение дерева**

Будем строить дерево с помощью программы, выполняющей синтаксический анализ выражения методом рекурсивного спуска. Потребуются следующие глобальные переменные:

var

 Ch : char; { Очередной символ }

 Tree : tPtr; { Дерево }

Counter : integer; { Счетчик временных переменных }

Задачей распознающих процедур *Expression* (выражение), *Term* (слагаемое) и *Multiplier* (множитель) кроме синтаксического анализа будет построение дерева того подвыражения, которое они распознали.

 Expression строит дерево выражения, Term — слагаемого, Multiplier — множителя.

 Построенное дерево будет выходным параметром каждой из этих процедур. Чтобы построить дерево Tree всего выражения, обращаемся к Expression:

Expression(Tree);

Expression строит дерево выражения, обращаясь к распознавателю слагаемого для получения ссылок на деревья слагаемых. Связи в дереве выстраиваются так, чтобы это соответствовало выполнению операций «+» и «-» в цепочке слагаемых в порядке слева направо.

**procedure** Expression(**var** Tree : tPtr);

**var**

 t : tPtr;

**begin**

 Term(Tree);

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

 New(t); {Новая вершина}

 t^.Ch := Ch; {содержит знак}

 t^.Left := Tree; {Слева - старое дерево }

 NextCh;

 Term(t^.Right); {Справа - новое слагаемое}

 Tree := t; {Указатель дерева - на новую

 вершину}

 **end;**

**end;**

Рисунок 39 иллюстрирует последовательность построения дерева для выражения a-b-с.

Анализатор слагаемого Term действует подобно процедуре Expression.

**procedure** Term(**var** Tree : tPtr);

**var**

 t : tPtr;

**begin**

 Multiplier(Tree);

 **while** Ch in [' \* ', ' /' ] **do begin**

 New(t);

 T^.Ch := Ch;

 T^.Left := Tree;

 NextCh;

 Multiplier(t^.Right);

 Tree := t;

**end;**

**end;**

Процедура-распознаватель множителя, встречая операнд, не заключенный в скобки, создает вершину, которая будет листом дерева.

**procedure** Multiplier(**var** Tree : tPtr);

**begin**

 **if** Ch **in** ['a', 'b', 'с'] **then begin**

 New(Tree);

 Tree^.Ch := Ch;

 Tree^.Left := **nil**;

 Tree^.Right := **nil**;

 NextCh

 **end**

 **else if** Ch = '(' **then begin**

 NextCh;

 Expression(Tree);

 **if** Ch = ') ' **then**

 NextCh

 **else**

 Error('Ожидается ")"');

 **end**

 **else**

 Error(' Ожидается a, b, с или "(" ');

**end;**

Дерево построено.

**Получение постфиксной записи**

Имея в распоряжении семантическое дерево, очень легко получить обратную польскую запись выражения. Для этого достаточно выполнить обход дерева (посещение всех его вершин) в порядке снизу вверх и слева направо, когда для каждой вершины вначале посещается ее левое поддерево, потом правое и в последнюю очередь — сама вершина (рис. 40, а).

Посещая каждую вершину, надо просто выводить ее содержимое (значение поля ch).



Рис.40. Различный порядок обхода дерева.

Рекурсивный обход дерева в указанном порядке выполняет процедура Postfix.

{ Обход дерева t снизу вверх }

**procedure** Postfix (t: tPtr);

**begin**

 **if** t <> **nil then begin**

 Postfix(t^.Left); { Обход левого поддерева }

 Postfix(t^.Right); {Обход правогоподдерева}

 Write(t^.Ch); { Посещение вершины t }

 **end;**

**end;**

Тривиальным случаем, когда не происходит рекурсивных вызовов (и не выполняется вообще никаких действий) здесь является ситуация, когда дерево пусто (t = nil).

Чтобы выполнить обход фактически и получить обратную польскую запись, нужно вызвать Postfix, указав фактическим параметром имеющееся дерево:

**Postfix(Tree);**