**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:

{ Ошибка }

**procedure** Error(Message : **string**);

**var**

e : **integer;**

**begin**

 **if** ErrPos = 0 **then begin**

 WriteLn(‘^': Pos);

 WriteLn('Синтаксическая ошибка: ', Message);

 e := Pos;

 **while** Ch <> EOT **do** NextCh;

 ErrPos := e;

 **end;**

**end;**

Процедура Error «самоблокируется», проверяя значение ErrPos и обходя выдачу сообщений при повторных вызовах. Пользоваться ею можно так же, как и раньше, как бы считая, что после вызова Error работа анализатора прерывается.

Дополнительные проверки на наличие ошибки после обращения к распознавателям нетерминалов не нужны.

Nextch после обнаружения ошибки всегда возвращает «конец текста».

**procedure** NextCh;

**begin**

**if** ErrPos <> 0 **then**

Ch := EOT

**else**

…………………

**end;**

Надо, однако, беспокоиться о том, чтобы при обнаружении ошибки не оставались неопределенными или недопустимыми данные, связанные с семантической обработкой. Они могут использоваться в вычислениях, которые будут выполняться после выдачи сообщения об ошибке при возврате из распознающих процедур.

При этом не должно возникнуть недопустимых ситуаций: деление на ноль, разыменование неопределенного или равного nil указателя, выход индекса за границы массива и т. п.

Например, распознаватель степени, в задачу которого входит определение величины р, при использовании нашей технологии следует записать так:

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

**begin**

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

 NextCh;

 Number(p);

 **if** p > nmax **then begin**

 Error('Слишком большая степень');

 p:=0;{Степень не должна остаться слишком большой}

 **end**

 **end**

 **else**

 Р := 1;

**end;**

Требуется еще одно уточнение. Поскольку по завершении работы распознавателя, обнаружившего ошибку, текущим символом станет EOT, его проверка уже не будет достаточным условием успешного окончания анализа. Использовавшееся завершение:

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

Error('Ожидается конец текста')

**else**

WriteLn('Правильно');

должно быть заменено следующим:

**if** (ErrPos = 0) **and** (Ch = EOT) **then**

WriteLn('Правильно')

**else**

Error('Ожидается конец текста');

**Трансляция выражений**

Выражение — одно из основных понятий языков программирования. Подсистема, ответственная за трансляцию выражений, представляет собой важную и одну из самых сложных частей компилятора.

**Польская запись**

Обычная форма выражений предусматривает запись знака операции между операндами. Например:

а + b

а + b\*с

(а + b)\*с

Такую запись называют инфиксной.

В некоторых случаях используется префиксная (прямая польская) запись,

когда обозначения операций записываются до операндов.

+ ab

+ а\*bс

\* + abc

Постфиксная (обратная польская) запись требует, чтобы знаки операций записывались после соответствующих операндов. Приведенные инфиксные выражения в обратной польской записи будут выглядеть так:

ab +

abc\* +

ab + c\*

Польской такая нотация названа в честь ее изобретателя — польского математика и логика Яна Лукасевича (Jan Lukasiewicz, 1878-1956).

Обратную польскую запись будем называть также польской инверсной записью (сокращенно ПОЛИЗ).

Польская запись не содержит скобок.

При записи в ПОЛИЗ операнды записываются в порядке следования в исходном выражении, операции — после своих операндов в порядке выполнения.

Интерес к ПОЛИЗ обусловлен тем, что она удобна для вычисления выражений и как промежуточная форма представления выражений в трансляторе.

Принцип обратной польской записи может быть применен не только к выражениям, но и операторам языков программирования.

**Алгоритм вычисления выражений в обратной польской записи**

Выражения в обратной польской записи вычисляются с помощью стека.

При чтении ПОЛИЗ слева направо операнды помещаются в стек, а операции применяются к верхним элементам стека. Результат выполнения операции возвращается в стек, заменяя собою операнды. По исчерпании выражения его значение находится на вершине стека.

В приведенном алгоритме (см.листинг) предполагается, что ПОЛИЗ состоит из отдельных элементов, и мы в состоянии отличать операнды от операций (с помощью множеств «Операнды», «Одноместные операции», «Двуместные операции»), а одноместные операции — от двуместных.

Предполагается также, что имеются только одноместные и двуместные операции.

*Листинг Алгоритм вычисления выражения в ПОЛИЗ*

Читать(элемент); { Первый элемент ПОЛИЗ }

**while** элемент <> конец **do begin**

 **if** элемент **in** Операнды **then**

 Push(Значение(элемент))

 **else if** элемент **in** Одноместные операции

**then begin**

 Pop (x) ;

 Push( Oп1(элемент, х) );

**end**

**else if** элемент **in** Двуместные операции **then begin**

 Pop(x2); Pop(x1); {Обратите внимание }

 Push( 0п2(элемент, x1, x2) ); {на порядок x1 и x2}

**end;**

 Читать(элемент);

**end;**

Pop(Значение выражения);

Операции со стеком в этом алгоритме выполняются с помощью процедур *Push* (в стек) и *Pop* (из стека), у которых параметр-стек для краткости не указан.

Функция *Значение* позволяет по записи элемента-операнда получить его значение. В зависимости от того, как кодируются операнды, является ли операнд константой или переменной, эта функция реализуется по-разному и здесь не детализируется.

Значением функций Oп1 и Оп2 является результат применения одноместной и двуместной операции, заданной значением первого параметра, к последующим параметрам. Например, если речь идет об операциях с вещественными числами, эти функции могут быть такими:

**function** 0п2(элемент: Элементы; x1, x2: r**eal**): **real**;

**begin**

 **case** элемент **of**

 сложение: 0п2 := x1 + х2;

 вычитание: 0п2 := x1 - х2;

 умножение: 0п2 := x1\*x2;

 деление: 0п2 := x1/x2;

 **end;**

**end;**

**function** On1(элемент: Элементы; х: **real)**: **real;**

**begin**

 **case** элемент **of**

 изменение знака: Oп1 := -x;

 синус: Oп1 := sin(x);

 косинус: Oп1 := cos (x);

 **end;**

**end;**

**Схема трансляции выражений**

Разные варианты использования обратной польской записи при трансляции выражений показаны на рис. 33.

Быстрый интерпретатор (рис. 33, а) вначале преобразует выражение в ПОЛИЗ, а затем вычисляет его по алгоритму, приведенному выше.

Компилятор (рис. 33,б) использует ПОЛИЗ в роли промежуточного представления, по которому генератор формирует машинный код.

В схеме «чистого интерпретатора» (рис. 33, в) польская запись в явном виде не формируется, но совершаемые интерпретатором действия включают в себя те, что нужны для получения польской записи и те, что служат для вычисления с помощью стека.

**Алгоритм Э. Дейкстры перевода выражений в обратную польскую запись (метод стека с приоритетами)**

 Рассмотрим простой и эффективный алгоритм Э. Дейкстры, основанный на использовании стека операций. Он состоит в следующем:

1. Каждому знаку операции присваивается числовой приоритет. Операции, выполняемые в первую очередь, имеют больший приоритет. Приоритеты операций начинаются с 2.

2. Открывающим скобкам присваивается приоритет 0, закрывающим — 1.

3. Входная строка считывается слева направо. Операнды по мере их считывания помещаются в выходную строку.

4. Знаки перед записью в выходную строку помещаются в стек (рис. 34) в соответствии со следующей дисциплиной:



Рис. 34. Использование стека в алгоритме Э. Дейкстры

- Открывающая скобка (знак с приоритетом 0) помещается в стек.

- Если приоритет знака операции больше приоритета знака на вершине стека или стек пуст, новый знак добавляется в стек.

- Если приоритет знака меньше или равен приоритету знака на вершине стека, из стека в выходную строку «выталкиваются» все знаки с приоритетами, большими или равными приоритету входного знака. После этого входной знак записывается в стек.

- Закрывающая скобка выталкивает в выходную строку все знаки до открывающей скобки. Открывающая скобка удаляется из стека. Закрывающая и открывающая скобки в выходную строку не записываются.

5. После просмотра всех символов входной строки из стека выталкиваются оставшиеся знаки.

Этот метод может быть распространен и на функции, переменные с индексами и др.