Трансляторы

Транслятор – переводит текст на ЯП в машинный код.

Трансляторы:

1. Компиляторы – переводят сразу весь текст в машинный код, а затем выполняют.
2. Интерпретаторы – переводят и выполняют программу построчно (перевёл строку – выполнил, перевёл следующую строку – выполнил и т.д.).

Трансляция:

1. Анализ:
   а) Лексический анализ – деление на лексемы (ключевые слова, идентификаторы, операции, разделители, строки, числа) и их запись в универсальном формате.
   б) Синтаксический анализ – проверка принадлежности грамматики к языку программирования (правильно ли записаны операторы, выражения). На этом этапе отлавливаются синтаксические ошибки.
   в) Семантический анализ – проверка на слысл. Отлавливание ошибок деления на ноль и т.д. [В рамках данного курса не разбираем]
2. Синтез:
   а) Перевод в промежуточный язык. [В рамках данного курса промежуточным языком является обратная польская запись]
   б) Перевод в код на нужном языке.

Разработка лексического анализатора

Лексемы (6 видов):

1. Ключевые слова
2. Идентификаторы
3. Разделители
4. Операторы
5. Строки
6. Числа

На вход лексического анализатора будет подаваться файл с программой на входном языке. Его задача – выделить лексемы и перевести их в единый язык. Для этого нужно собрать все лексемы в таблицы. Таблиц всего 3: служебные слова, операции, разделители. Слова, операции и разделители предопределены, не будем касаться ситуации введения программистом дополнительных операций. 3 таблицы заполняются ещё до лексического анализа. Ключевым словам ставятся в соответствие номера.

В таблицу служебных слов включаются: if, else, while и т.д.

Будем рассматривать описания процедур, арифметический выражения, логические выражения, начало и конец блоков.

Основные арифметические операции (<, >, <=, >=, +, - и т.д.), логические операции также заносятся в таблицу.

[Будем учитывать и рассматривать однострочные комментарии, без многострочных]

А таблицы идентификаторов, констант заполняются по ходу прохода по коду программы.

Лексический анализатор читает код:

1. Видит букву.
2. Читаем до тех пор, пока не встретим что-то отличное от буквы, цифры или знака подчёркивания (алфавитно-цифрового символа). Мы должны запоминать, что прочитали – должны организовать буфер.
3. Если встретили что-то отличное, то заглядываем в буфер, смотрим, что получилось, смотрим в таблицы: сначала в ключевые слова, если ничего не нашлось, идём в таблицу идентификаторов, если и тут ничего не нашлось, то идём в таблицу констант. Если мы нашли слово в какой-то таблице, то очищаем буфер и записываем вместо этого слова в коде лексему из промежуточного языка, например, W1.

[Дробные числа также рассматриваем (достаточно рассмотреть с фиксированной точкой)]

1. Видит цифру.
2. Читаем, пока не встретим что-то другое. Если видим букву сразу после цифр, то выдаём ошибку. Можем видеть только разделитель или точку (для дробных чисел).
3. Если закончили читать и не возникло ошибки (если не встретили второй точки и не встретили точку, после которой ничего нету), то заносим в таблицу и печатаем, например, N1.

1. Видит разделитель.
2. Печатаем лексему.

1. Видит операцию.
2. Печатаем лексему (если видим >, то проверяем, что это может быть >=, <= и т.д.).

Корректность построения выражений с помощью операций и чисел проверяется на этапе синтаксического анализа, а не на этапе лексического анализа.

На данном этапе нужно перевести лексемы из исходного языка в лексемы промежуточного языка.

[Мой вариант — 28: перевод из R в Python]

Обратная польская запись

Пример: `A + B * C`

Операнды просто переписываем в том же порядке, что и в выражении: `A B C`

Операции заносятся в стек: **`+ -> *+`**

Получаем запись в ОПЗ: `A B C * +`

При вычислении по ОПЗ наоборот: операнды заносим в стек, а операции берём в том же порядке.

Пример: `(A + B) * (C - D) - X ^ 3`

Стек: `_ -> ( -> +( -> )+( -> _ -> ...`

Операнды: `A B C D X 3`

ОПЗ: `A B + C D - * X 3 ^ -`

`A[i,j] == Aij3AЭМ`

`A[i+1,j*2]*(B+C[i-1]) == Ai1+j2*3AEMBCi1-2AEM+*`