ФЕДЕРАЛЬНОЕ АГЕНСТВО СВЯЗИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ТЕЛЕКОММУНИКАЦИЙ И ИНФОРМАТИКИ»

Кафедра вычислительных систем

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К курсовой работе по дисциплине

«Теория языков программирования и методы трансляции»

Выполнил:

Студент гр. ИВ-222

Гайдай А.В.

Проверил:

доцент к.т.н.

Молдованова О.В.

Новосибирск 2016 г.

СОДЕРЖАНИЕ

Введение

Цель работы: изучение составных частей, основных принципов построения и функционирования трансляторов, практическое освоение методов построения простейших трансляторов для некоторого входного языка, приобретение навыков командной работы.

Задание заключается в создании транслятора с некоторого входного языка на заданный выходной язык. Для выполнения работы студенты делятся на команды по 3 человека. Каждая команда разрабатывает свой входной язык программирования. В качестве выходного (результирующего) должен использоваться язык ассемблера процессоров типа Intel 80x86.

Учебное руководство по разработанному языку

1. Hello World

Для ознакомления с языком обычно предлагается написать первую программу, выводящую сообщение «Hello World».

hello.prog

```
1    PRINT "Hello World";
2    END
```

* PRINT – функция вывода строки или содержимого переменной на экран; END – структурная единица языка, указывающая на конец программы.

Для трансляции программы на язык ассемблер и после этого в исполняемый файл нужно выполнить следующий скрипт:

```
[user@name]$    ./agaidai hello.prog hello
```

В случае успеха на экране появится соответствующее сообщение. Теперь, помимо исходной программы, имеется объектный файл **hello.o**, файл на языке ассемблера **hello.asm** и, собственно, исполняемый файл **hello.out**. После запуска программы на экране появится сообщение «Hello World».

agaidai

```
1    #!/bin/bash
2    path="/home/anatolygayday/Documents/University/TPL/KR/v1_1_2/base/"
3
4    bison -dy $path"input.y"
5    flex $path"input.l"
6    gcc lex.yy.c y.tab.c
7    ./a.out $1 -lt 0
8    rm lex.yy.c y.tab.c y.tab.h a.out
9
10   $path"semantic_analyzer.out"
11
12   if [ $? -eq 0 ]
13   then
14       cp $path".tmp3" $2".asm"
15
16       nasm -g -f elf -o $2".o" $2".asm"
17       ld -m elf_i386 -o $2".out" $2".o"
18   fi
19
20   rm $path".tmp" $path".tmp1" $path".tmp2" $path".tmp3"
21   rm $path".tmp4" $path".tmp5" $path".tmp6" $path".tmp7"
```

2. Программа №2

Функционал разработанного языке не заканчивается только лишь на возможности вывода сообщений на экран. Для раскрытия большего потенциала к рассмотрению предлагается следующий учебный пример:

Example1.prog

```
1    INT a, FLOAT b, FLOAT s, INT wa
2
3    #=
4    PRINT "THIS IS COMMENT PART!\n";
5
6    PRINT "Input b: ";
7    SCAN b;
8    PRINT "b="; PRINT b; PRINT "\n";
9    =#
10
11   PRINT "Input a: ";
12   SCAN a;
13   PRINT "a="; PRINT a; PRITN "\n";
14
15   a := a + 0.2;
16
17   PRINT "Input b: ";
18   SCAN b;
19   PRINT "b="; PRINT b; PRINT "\n";
20
21   PRINT "Input s: ";
22   SCAN s;
23   PRINT "s="; PRINT s; PRINT "\n";
24
25   wa := 1 + s;
26
27   PRINT "\n\n_____\n\n";
28   IF s = b THEN
29      s := 0 – s;
30      IF  0 - wa – 1  = s THEN
31         PRINT "0 – wa – 1 = s\n";
32         PRINT "s="; PRINT s; PRINT "\n";
33         PRINT "b="; PRINT b; PRINT "\n";
34      FI
35
36      FOR a := 0; a < 10; a := a + 1; THEN
37
38         IF a > 5 THEN
39            s := s – 1;
40            b := b – 1;
41            PRINT "a > 5\n";
42         ELSE
43            a := a + 1;
44            b := b + 1;
45            PRINT "a <= 5\n";
46         FI
47
48         PRINT "s="; PRINT s; PRINT "\n";
49         PRINT "b="; PRINT b; PRINT "\n";
50      ROF
51   FI
52
52   IF s > b THEN
54      s := 0.00;
55      b := 0.00;
56   ELSE
57      s := 1.11;
```

```
58      b := 1.11;
59   FI
60
61   PRINT "RESULT\n";
62   PRINT "s="; PRINT s; PRINT "\n";
63   PRINT "b="; PRINT b; PRINT "\n";
64
65   END
66
```

Трансляция программы:

```
[user@name]$   ./agaidai Example1.prog Example1
Correct syntax     ;)
Correct semantic ;)
[WARNING]: implicit type (int = float) for variable 'a' in assigment
[WARNING]: implicit type (int = float) for variable 'wa' in assigment

[user@name]$
```

Анализ кода программы **Example1.prog**:

Структурные единицы разработанного языка **INT** и **FLOAT** должны находится перед телом программы, либо отсутствовать вовсе. С помощью них в таблицу идентификаторов добавляются целочисленные и вещественные переменные соответственно. На этом этапе происходит инициализация данных. Объявление переменных происходит через запятую, а после последнего объявления запятая не ставится.

Комментарий языка начинается с символов «**#=**» и заканчивается символами «**=#**». Комментарий не может содержать символ «**#**».

Функция **PRINT**, как уже отмечалось ранее, выводит либо строку, либо значение переменной на экран. Служебный символ «**\n**» означает перевод строки.

Функция **SCAN** предназначена для считывания значения с консоли и записи его в указанную переменную. Формат ввода: **[0-9]$^{+}$[.]*[0-9]***.

Последовательность символов «**:=**» есть оператор присваивание переменной слева от «**:=**» значения выражения справа от «**:=**».

Формат выражения: **expr → (expr)[+-*/](expr) | num | var .**

* expr – выражение; num – число; var- переменная.

Инструкция выбора **IF**.

1. При истинности условия стоящего после «**IF**» выполняется часть кода, идущая после служебного слова «**THEN**» и до «**ELSE**» или «**FI**», в случае если блок «**ELSE**» отсутствует.
2. В противном случае программа переходит к следующему действию, идущему после «**FI**», или же к выполнению блока «**ELSE**», если такой существует.

Циклическая инструкция **FOR**.

1. Сначала(единожды) выполняется выражение, которое следует непосредственно после служебного слова «**FOR**».
2. Определяется результат условия, идущего далее;
3. Если условие истинно, выполняется тело инструкции от служебного слова «**THEN**» до «**ROF**». В конце выполняется выражение стоящее перед «**THEN**» и алгоритм продолжает возвращается на шаг 2.
4. Если условие ложно, цикл завершает работу: выполняются действия, следующие после слова «**ROF**».

Инструкции **IF** и **FOR** могут быть вложенными. Максимальная глубина вложенности 31(независимо для каждого оператора).

При трансляции, на экран выводятся сообщения об имеющихся ошибках и предупреждениях. Так, в данном примере имеется два предупреждения: неявное приведение типов для переменных «**a**» и «**wa**». Вот те строки кода, где это происходит:

```
15  a := a + 0.2;
          …
25  wa := 1 + s;
```

Справочник по разработанному языку

Комментарий. Последовательность символов «**#=**» открывает комментарий, а «**=#**» закрывает его. Комментарий не должен включать в себя символ «**#**».

Идентификаторы. Идентификатор – это последовательность букв и цифр. Первым символом должна быть буква. Буквы нижнего и верхнего регистров различаются. Максимальная длина идентификатора 32 символа.

***** знак подчёркивания «**_**» считается буквой.

Ключевые слова. Следующие слова зарезервированы в качестве ключевых слов и в другом смысле использоваться не могут: *INT, FLOAT, PRINT, LEFT, IF, FI, ELSE, FOR, ROF, THEN, END, lExPrVaLuE, rExPrVaLuE*. Также ключевыми словами являются все идентификаторы, которые начинаются и заканчиваются последовательностью символов «**__**» (два нижних подчёркивания).

Строковые литералы. Строковый литерал, который также называют строковой константой – это последовательность символов, заключённых в кавычки «**”**».

***** символы «**#**» и «**”**» не должны входить в состав строкового литерала.

Выражения. Операторы имеют приоритет: «*****» и «**/**» – высший, «**+**» и «**-**» – низший). Гарантируется ассоциативность соответствующих операций. Выражение не должно начинаться и заканчиваться знаком операции («**+**», «**-**», «*****», «**/**»).

Формат выражения: **expr → (expr)[+-*/](expr) | num | var .**

***** expr – выражение; num – число; var- переменная.

****** максимальное количество операторов в выражении: 10.

Аддитивные операторы. Аддитивные операторы «**+**» и «**-**» выполняются слева направо. Результат выполнения оператора «**+**» есть сумма его операндов. Результат выполнения оператора «**-**» (минус) есть разность операндов.

Мультипликативные операторы. Мультипликативные операторы «*****» и «**/**» выполняются слева направо. Результат выполнения оператора «*****» есть произведение его операндов. Результат выполнения оператора «**/**» есть целая часть отношения
операндов.

Операторы присваивания. Оператор присваивания «**:=**» имеет левый операнд-идентификатор и правый операнд-выражение.

Преобразования. Оператор присваивания «**:=**», в зависимости от своих операндов, может вызывать преобразование их значений из одного типа(**FLOAT**) в другой(**INT**). О данных преобразованиях сообщается на этапе трансляции.

Операторы сравнения. Операторы сравнения выполняются слева направо.

Операторы: «**>**» (больше), «**<**» (меньше), «**=**» (равно), «**!**», (неравно). Результат сравнения: 0 – ложь, 1 – истина.

***** операнды операций сравнения приводятся к целочисленному виду.

Спецификаторы типа. Спецификатор **INT** заносит переменную в таблицу идентификаторов как целочисленную, а **FLOAT** – как вещественную.

Инструкции выбора.

IF условие **THEN** инструкции **FI**

IF условие **THEN** инструкции **ELSE** инструкции **FI**

Циклические инструкции.

FOR выражение условие; выражение **THEN** инструкции **ROF**

***** выражение включает знак «**;**» в конце.

****** под инструкциями понимаются инструкции выбора, циклические инструкции, операторы присваивания.

Описание грамматики входного языка

Litter → «A» | «B»| «C» | «D» | «E» | «F» | «G» | «H» | «I» | «J» | «K» | «L» | «M» | «N» | «O» | «P» | «Q» | «R» | «S» | «T» | «U» | «V» | «W» | «X» | «Y» | «Z» | «a» | «b» | «c» | «d» | «e» | «f» | «g» | «h» | «i» | «j» | «k» | «l» | «m» | «n» | «o» | «p» | «q» | «r» | «s» | «t» | «u» | «v» | «w» | «x» | «y» | «z»

Digit → «0» | «1» | «2» | «3» | «4» | «5» | «6» | «7» | «8» | «9»

Delim → «;»

Oper → «+»| «-»| «*»| «/»

Cmp → «>»| «<»| «=»| «!»

Описание лексем языка

END	**→**	**“END”**
DELIMITER2	**→**	**“,”**
COMMENT	**→**	**“#=”[^#]*”=#”**
PRINT	**→**	**“PRINT”**
SCAN	**→**	**“SCAN”**
FLOAT	**→**	**“FLOAT”**
INT	**→**	**“INT”**
FOR	**→**	**“FOR”**
ROF	**→**	**“ROF”**
IF	**→**	**“IF”**
FI	**→**	**“FI”**
THEN	**→**	**“THEN”**
ELSE	**→**	**“ELSE”**
STRING	**→**	**\”[^\”]*\”**
IDENTIFIER	**→**	**(“_” | {Letter}) (“_” | {Letter} | {Digit})***
NUMBER	**→**	**({Digit})+ | ({Digit})+ ((“.” | “”) ({Digit})+)**
ASSIGMENT	**→**	**“:=”**
COMPARE	**→**	**{Cmp}**
OPERATION	**→**	**{Oper}**
DELIMITER1	**→**	**{Delim}**
UNKNOWN	**→**	**.**

Описание правил языка

S	**→**	**INT IDETIFIER DELIMITER2 S**
		| FLOAT IDENTIFIER DELIMITER2 S
		| INT IDENTIFIER A END
		| FLOAT IDENTIFIER A END
		| A END
A	**→**	**B**
		| C
		| D

| F
| COMMENT
| B A
| C A
| D A
| F A
| COMMENT A

B → IF E THEN A FI
| IF E THEN A ELSE A FI

C → FOR F E DELIMITER1 F THEN A ROF

D → PRINT IDENTIFIER DELIMITER1
| PRINT STRING DELIMITER1
| SCAN IDENTIFIER DELIMITER1

E → G COMPARE G

F → IDENTIFIER ASSIGMENT IDENTIFIER

G → IDENTIFIER OPERATION G
| NUMBER OPERATION G
| IDENTIFIER
| NUMBER

Структура разработанного транслятора

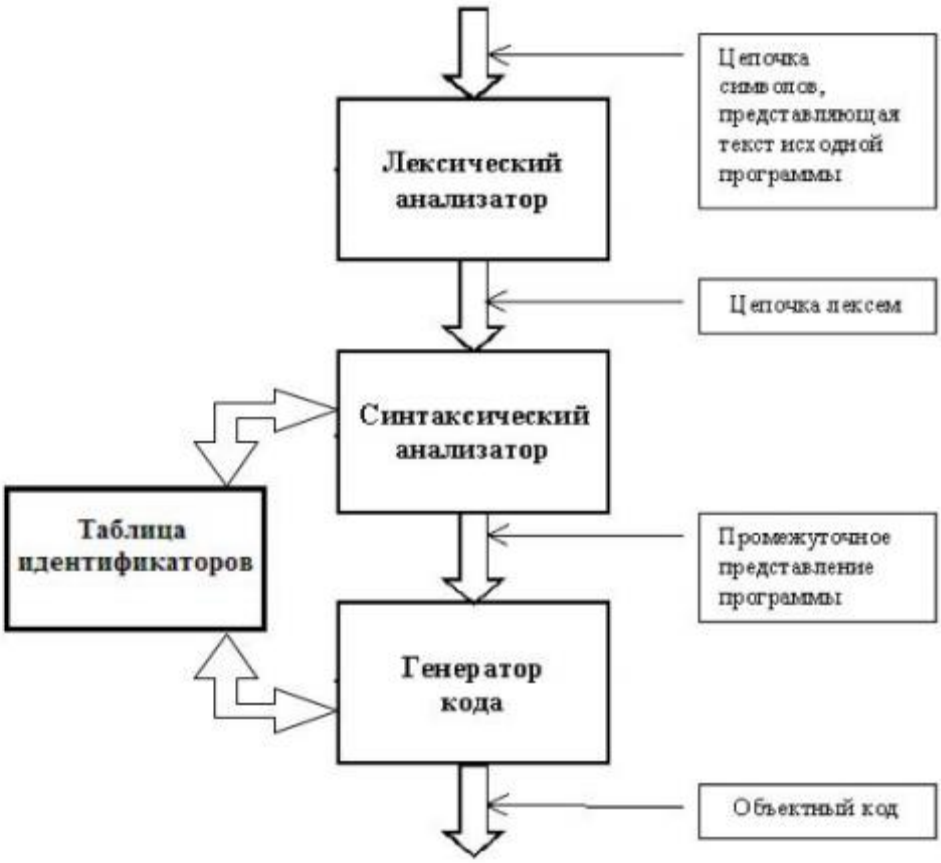

Рис. 1 - Структура разработанного транслятора

Описание средств разработки

Flex (Fast Lexical Analyzer) - это генератор программ, предназначенный для лексической обработки символьных входных данных. Он принимает проблемно-ориентированную спецификацию на высоком уровне и формирует программу на языке СИ, которая работает с регулярными выражениями. Регулярные выражения определяются пользователем в ключевой спецификации, выдаваемой программе lex. Система lex распознает эти выражения и разделяет входные данные на блоки в соответствии с ними. На границах между блоками исполняются фрагменты программ, разработанные пользователем. исходный файл lex объединяет регулярные выражения и фрагменты программ. По мере того, как выражения появляются на входе программы, составленной lex, соответствующий фрагмент подается на выполнение.

Пользователь вводит добавочные фрагменты программ, необходимые для комплектования его задания, включая коды, написанные другими генераторами. Программа, распознающая выражения, формируется из фрагментов программ пользователя на языке СИ. lex это не полный язык, а только генератор, представляющий новую возможность, которая дополняет возможности языка СИ.

flex превращает выражения и команды пользователя в программу на языке СИ, имеющую название yylex. Программа yylex распознает выражения во входном потоке и осуществляет специальные операции для каждого выражения, как только оно встретится.

Правила задаются в виде регулярных выражений слева и, обычно, кода на языке С справа. Правила содержат три секции, отделяющиеся строкой «%%»:

```
определения
%%
правила
%%
код пользователя
```

Определения содержат стартовые значения и определения, правила, непосредственно сами выражения и соответствующие им действия, пользовательский код просто включается в вывод flex. Некоторые секции могут отсутствовать.

GNU Bison – программный инструментарий, предназначенный для генерации синтаксических анализаторов на основе LALR(1)-грамматик. Bison совместим с другим генератором распознавателей – Yacc (Yet Another Compiler Compiler), разработанным в 1975 1978 гг. В большинстве случаев сканер, сгенерированный Flex, является сопрограммой для синтаксического анализатора, генерируемого с помощью Bison. Лексический анализатор возвращает синтаксическому поток токенов. Задача распознавателя выяснить взаимоотношения между этими токенами. Обычно такие взаимоотношения отображаются в виде дерева разбора. Программа, написанная на языке Bison, состоит из трех основных частей, как и программа на Flex:

```
объявления
%%
правила
%%
вспомогательные функции
```

Первые два раздела являются обязательными, хотя и могут быть пустыми. Третий раздел и предшествующие ему символы %% могут отсутствовать.

Первый раздел, раздел объявлений, – это управляющая информация для синтаксического анализатора, и обычно он настраивает среду исполнения для анализа. Этот раздел может включать в себя программный код на Си, который полностью копируется в начало генерируемого файла. Обычно это объявления переменных и директивы #include, заключенные в %{ и %}. В этом разделе также могут быть объявления вида %union, %start, %token, %type, %left, %right и %nonassoc. Раздел объявлений может содержать комментарии в /* и */.

Второй раздел содержит правила грамматики и действия, связанные с ними. Действие представляет собой программный код на Си, который выполняется, когда Bison применяет правило для входных данных.

Третий раздел – это программный код на Си, который полностью копируется в генерируемую программу анализатора.

Сложности при разработке

- Вывод чисел на экран с помощью языка ассемблер (с использованием только лишь инструкции **int 0x80**);
- Реализация функции **SCAN** на языке ассемблер (с использованием только лишь инструкции **int 0x80**);
- Работа с вещественными числами;
- Реализация наглядного вывода отчёта об имеющихся ошибках на этапе трансляции.

Примеры тестовых программ

Пример работы программы **Example1.prog** (код программы приведён выше).

Трансляция программы:

```
[user@name]$   ./agaidai Example1.prog Example1

Correct syntax     ;)
Correct semantic ;)
[WARNING]: implicit type (int = float) for variable 'a' in assigment
[WARNING]: implicit type (int = float) for variable 'wa' in assigment

[user@name]$
```

Результаты выполнения полученного исполняемого файла (**Example1.out**):

Входные данные: a-> 1; b->2; s->2.

```
[user@name]$   ./Example1.out

Input a: 1
a=1
Input b: 2
b=2.01
Input s: 2
s=2.01


_____

a <= 5
s=-2.01
b=3.01
a <= 5
s=-2.01
b=4.01
a <= 5
s=-2.01
b=5.01
a > 5
s=-3.01
b=4.01
a > 5
s=-4.01
b=3.01
a > 5
s=-5.01
b=2.01
a > 5
s=-6.01
b=1.01
RESULT
s=1.12
b=1.12
[user@name]$
```

Входные данные: a-> 1; b->0; s->2.

Трансляция программы:

```
[user@name]$   ./Example1.out

Input a: 1
a=1
Input b: 0
b=0.01
Input s: 2
s=2.01


_____

RESULT
s=0.01
b=0.01
[user@name]$
```

Входные данные: a-> 1; b->2; s->0.

Трансляция программы:

```
[user@name]$   ./Example1.out

Input a: 1
a=1
Input b: 2
b=2.01
Input s: 0
s=0.01


_____

RESULT
s=1.12
b=1.12
[user@name]$
```

Список литературы

1. http://e-learning.bmstu.ru/moodle/file.php/1/common_files/library/SPO/FPU/bmstu_iu6_Sysprogr_Floating_point.pdf
2. http://stackoverflow.com/questions/6903435/nasm-linux-assembly-printing-integers
3. http://easylab.net.ua/raznoe-c/assembler-operatsiya-logicheskogo-sdviga
4. http://www.kolasc.net.ru/cdo/programmes/assembler/cpumodel_1.html
5. http://www.codenet.ru/progr/asm/nasm/nasm_ru3.php#section-3.2.1
6. https://en.wikibooks.org/wiki/X86_Assembly/NASM_Syntax
7. https://en.wikibooks.org/wiki/X86_Assembly/Floating_Point
8. http://natalia.appmat.ru/c&c++/assembler.html
9. http://www.nasm.us/
10. http://easylab.net.ua/
11. http://asmworld.ru/