**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

курсовому проектированию

по курсу «Лингвистическое и программное обеспечение»

на тему «Разработка транслятора»

Пенза 2018

Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «САПР»

Выполнили:

студенты группы 16ВВ3

Принял:

Финогеев А. А.

Содержание

[**ВВЕДЕНИЕ 19**](#_Toc516003836)

[**ОПИСАНИЕ ЯЗЫКА В ФОРМЕ БЭКУСА-НАУРА 20**](#_Toc516003837)

[**ОПИСАНИЕ ГРАМАТИКИ 21**](#_Toc516003838)

[**ПРЕОБРАЗОВАНИЕ ГРАММАТИКИ В LL(1)-ФОРМУ 22**](#_Toc516003839)

[**СИНТАКТИЧЕСКИЙ ГРАФ 23**](#_Toc516003840)

[**ЛЕКСИЧЕСКИЙ АНАЛИЗАТОР 24**](#_Toc516003841)

[**СИНТАКТИЧЕСКИЙ АНАЛИЗАТОР 30**](#_Toc516003842)

[**ГЕНЕРАЦИЯ ПОСТИКСНОЙ ЗАПИСИ 31**](#_Toc516003843)

[**СПИСОК ЛИТЕРАТУРЫ 37**](#_Toc516003844)

[**ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ 38**](#_Toc516003845)

[**«Main.cpp» 38**](#_Toc516003846)

[**«Library.h» 40**](#_Toc516003847)

[**«Synt+Pfx.h» 42**](#_Toc516003848)

[**«ASM.h» 46**](#_Toc516003849)

### **ВВЕДЕНИЕ**

Целью курсового проекта является разработка транслятора, программы для преобразования исходного текста программы в текст, который будет понятен адресату. В качестве адресата может выступать как программное, так и техническое средство. В данной работе мы будем генерировать программу на языке ассемблера. Чтобы облегчить написание генератора кода и освободить его от посторонних соображений, связанных с конкретными особенностями какой-либо ЭВМ, мы будем использовать гипотетический процессор (виртуальную машину). Этот процессор не существует на самом деле (в аппаратном виде). При выборе его архитектуры требовалась максимальная простота и, в то же время, возможность легко выполнять на нем программы на языках, реализуемых в процессе выполнения курсового проектирования.

В качестве языка программирования для решения поставленной задачи была выбрана последняя версия С++ 2017 года.

В качестве среды разработки была выбрана Visual Studio 2017.

# ОПИСАНИЕ ЯЗЫКА В ФОРМЕ БЭКУСА-НАУРА

<Программа> ::= <Объявление переменных> <Описание вычислений> .   
  
<Описание вычислений> ::= Begin <Список присваиваний> End   
  
<Объявление переменных> ::= Var <Список переменных>   
  
<Список переменных> ::= <Идент> | <Идент> , <Список переменных>   
  
<Список присваиваний>::= <Присваивание> | <Присваивание> <Список присваиваний> | <Конструкция с CASE> | <Конструкция с CASE> <Список присваиваний>

<Конструкция с CASE>::= CASE <Идентификатор> OF <Константа>: <Список операторов> {<Список альтернатив>} ENDCASE  
  
<Присваивание> ::= <Идент> = <Выражение>   
  
<Выражение> ::= <Ун.оп.> <Подвыражение> | <Подвыражение>   
  
<Подвыражение> :: = ( <Выражение> ) | <Операнд> |   
  
< Подвыражение > <Бин.оп.> <Подвыражение>   
  
<Ун.оп.> ::= "-"   
  
<Бин.оп.> ::= "-" | "+" | "\*" | "/"   
  
<Операнд> ::= <Идент> | <Const>   
  
<Идент> ::= <Буква> <Идент> | <Буква>   
  
<Const> ::= <Цифра> <Const> | <Цифра>

# ОПИСАНИЕ ГРАМАТИКИ

|  |  |
| --- | --- |
| Алфавит терминальных символов | Символы:  “\n”, “=”, “(” , “)”, “,” , “:”, “/”,“ ” , “+”, “-”, “\*”, “`”.  Ключевые слова:  "Var","Begin", "End",  "OF","CASE","ENDCASE". |
| Алфавит нетерминальных символов | S, A, B, D, C, E , H, F, G, T, O, I, Q |
| Начальный символ | S |
| Набор порождающих правил | S-> А B.  A-> Var D  B-> Begin C End  D-> I | I , D  C-> E C | H C | ε  H-> CASE O R ENDCASE  R->OF O: C R | ε  E-> I = F  F-> ` G | G  G-> (F) | O | F T F  T-> + | - | \* | /  O-> I | Q  I-> “Буквенная последовательность”  Q-> “Числовая последовательность” |

## **ПРЕОБРАЗОВАНИЕ ГРАММАТИКИ В LL(1)-ФОРМУ**

|  |
| --- |
| D-> I | I , D |
| Факторизация:  D->I K  K->, I | ε |

|  |
| --- |
| Правило C-> E C | H C | ε преобразовывается в C-> E C | H C | ; C | ε для более удобной работы с пустым оператором ‘;’. |
| Факторизация:  C-> R C | ε  R-> E | H | ; |

|  |
| --- |
| F-> ` G | G |
| Факторизация:  F-> L G  L-> ` | ε |

|  |
| --- |
| G-> (F) | O | F T F |
| Устранение левосторонней рекурсии:  G-> (F) Z | O Z | F Z  Z-> T F | ε |
| Факторизация:  G-> X Z  X-> (F) | O | F  Z-> T F | ε |

|  |
| --- |
| I-> “a-z, A-Z” | “a-z, A-Z” I |
| Факторизация не требуется, так как идентификатор при разборе уже состоит из заданной буквенной последовательности |

## **СИНТАКТИЧЕСКИЙ ГРАФ**

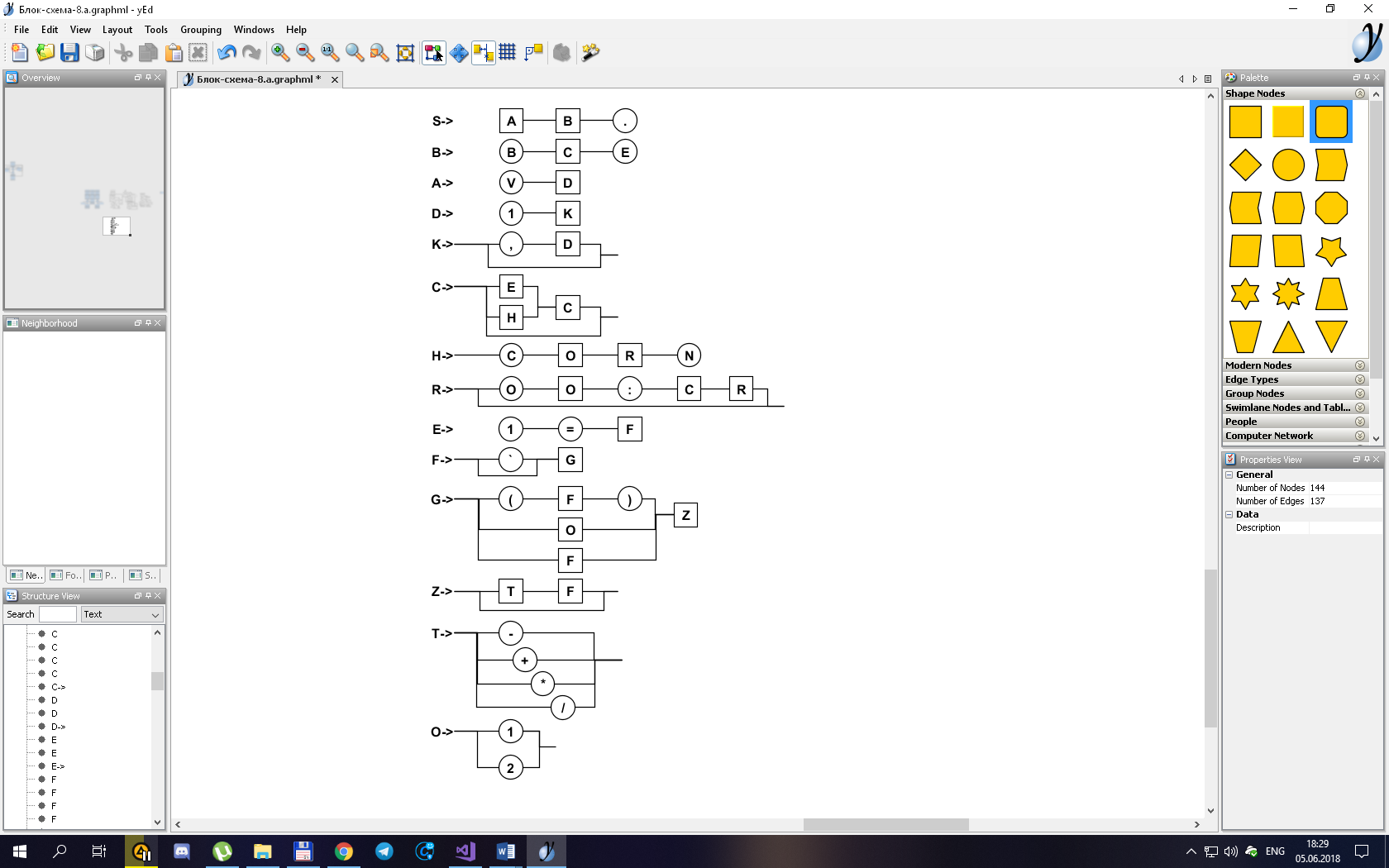


Рисунок 1‒ Синтактический граф

## **ЛЕКСИЧЕСКИЙ АНАЛИЗАТОР**

Задача лексического анализатора состоит в том, чтобы разбить исходный код на лексемы и присвоить этим лексемам определенный класс, к которому они принадлежат.

|  |  |
| --- | --- |
| Классы лексем | |
| Лексема: | Класс: |
| Переменная | 1 |
| Константа | 2 |
| Var | V |
| Begin | B |
| End | E |
| CASE | C |
| OF | O |
| ENDCASE | N |
| Недопустимая лексема | 0 |
| Данные лексемы являются сами для себя соответствующим классом: “\n”, “=”, “(” , “)”, “,” , “:”, “/”,“ ” , “+”, “-”, “\*”, “`”. | |

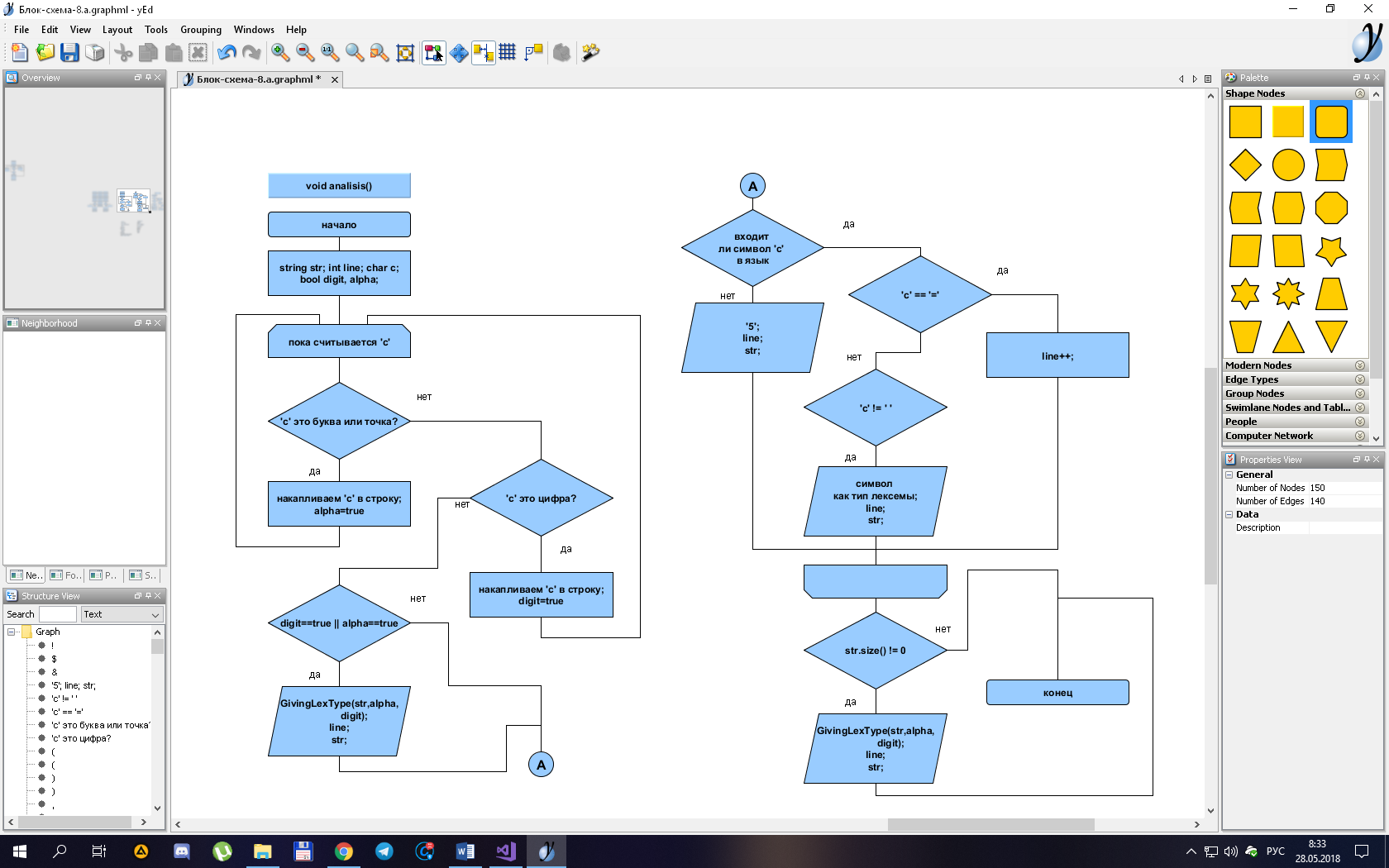
Схема лексического анализатора:

Рисунок 2‒ Схема лексического анализатора, часть 1

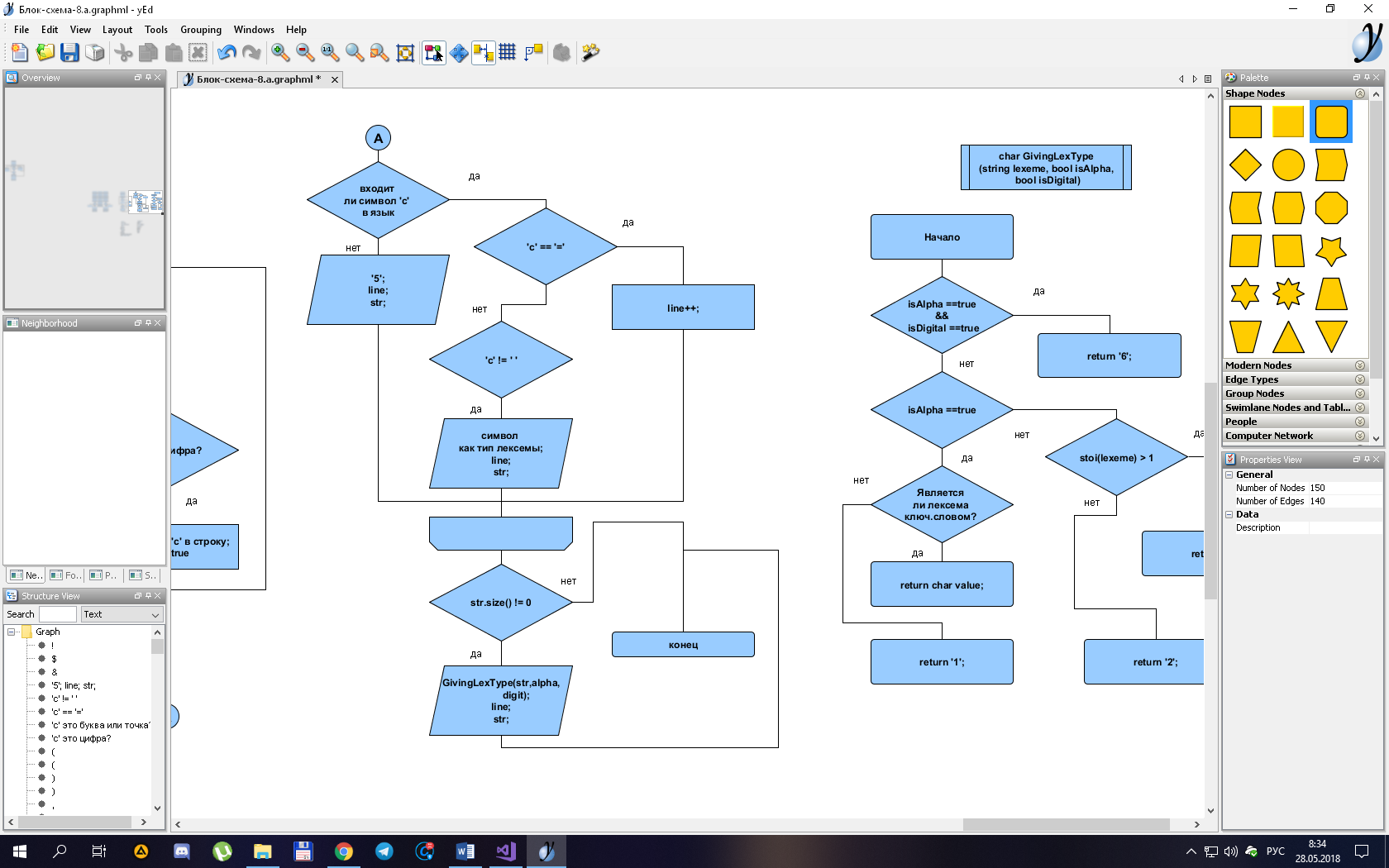


Рисунок 3‒ Схема лексического анализатора, часть 2

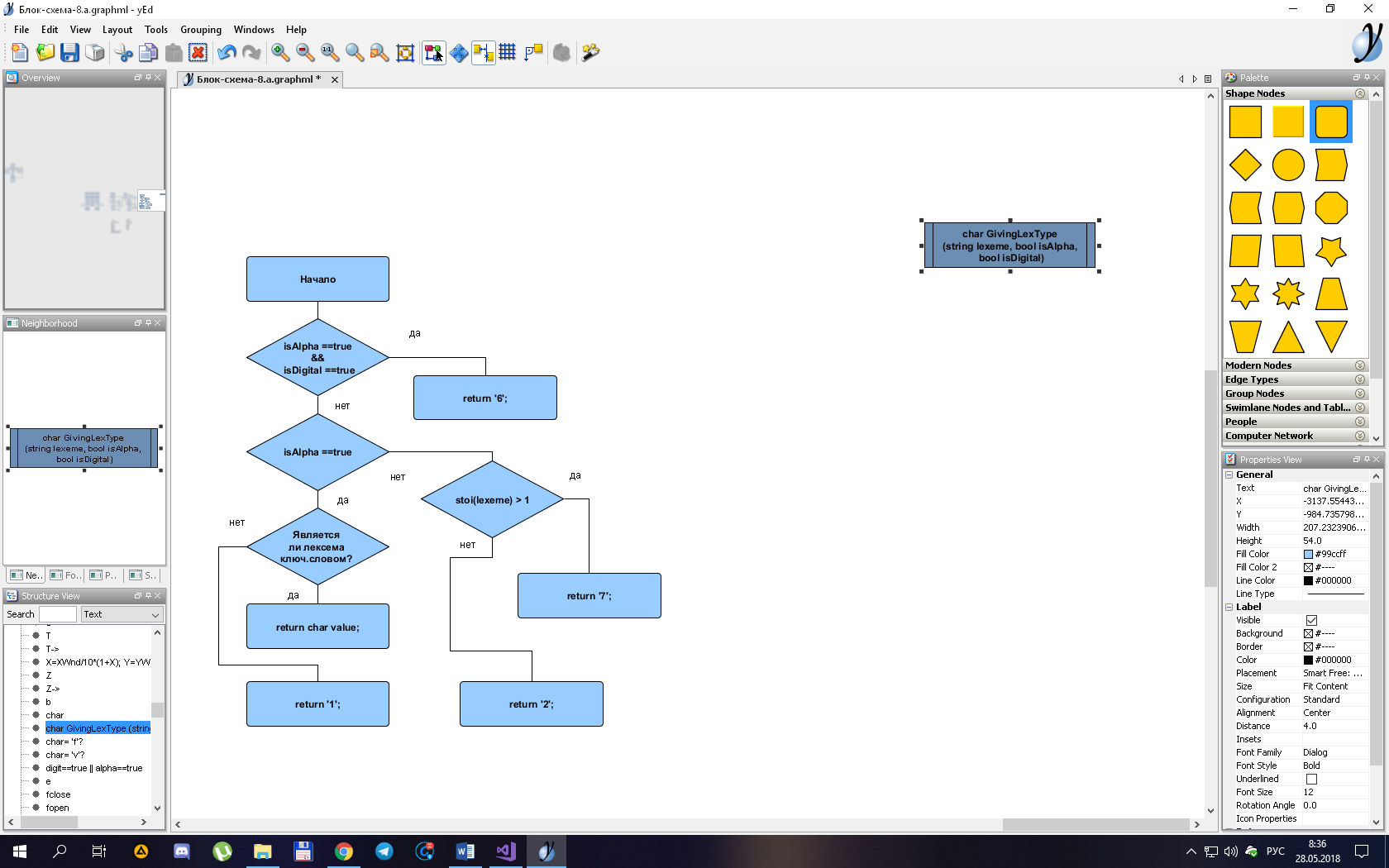
Схема функции, определяющей класс лексемы для буквенной и числовой последовательностей:

Рисунок 4‒ Схема функции   
char GivingLexType (string lexeme, bool isAlpha, bool isDigital)

При разборе входной последовательности информация о лексеме записывается в файл в следующем виде:

|  |  |  |
| --- | --- | --- |
| Класс лексемы | Номер строки | Лексема |

Пример разбора входной последовательности:

|  |  |
| --- | --- |
| Var a, b , c  Begin  a=1  b=0  c=0  CASE a  OF 5:  c=a\*(b+1)  OF 6:  c=a\*(b-1)  OF 7:  c=a/(b+1)  ENDCASE  End. | V 1 Var  1 1 a  , 1 ,  1 1 b  , 1 ,  1 1 c  B 2 Begin  1 4 a  = 4 =  2 4 1  1 5 b  = 5 =  2 5 0  1 6 c  = 6 =  2 6 0  C 8 CASE  1 8 a  O 9 OF  2 9 5  : 9 :  1 10 c  = 10 =  1 10 a  \* 10 \*  ( 10 (  1 10 b  + 10 +  2 10 1  ) 10 )  O 11 OF  2 11 6  : 11 :  1 12 c  = 12 =  1 12 a  \* 12 \*  ( 12 (  1 12 b  - 12 -  2 12 1  ) 12 )  O 13 OF  2 13 7  : 13 :  1 14 c  = 14 =  1 14 a  / 14 /  ( 14 (  1 14 b  + 14 +  2 14 1  ) 14 )  N 15 ENDCASE  E 17 End  . 17 . |

## **СИНТАКТИЧЕСКИЙ АНАЛИЗАТОР**

Синтактический анализатор основан на синтактическом графе, отображенном на рисунке 1.

Каждый символ входящей последовательности проверяется на соответствие с указанным на схеме символом. При несовпадении выводим соответствующее сообщение. По мимо этого транслятор способен отлавливать еще несколько видов ошибок:

|  |
| --- |
| Если переменная не объявлена |
| Неприемлемый символ |
| Смешанная лексема из цифр и букв |
| Несоответствие между ожидаемой и считанной лексемой |

Пример разбора анализатора некорректной последовательности:

|  |
| --- |
| a, b , c  Begin  a=1a  b=0  c=0  . |

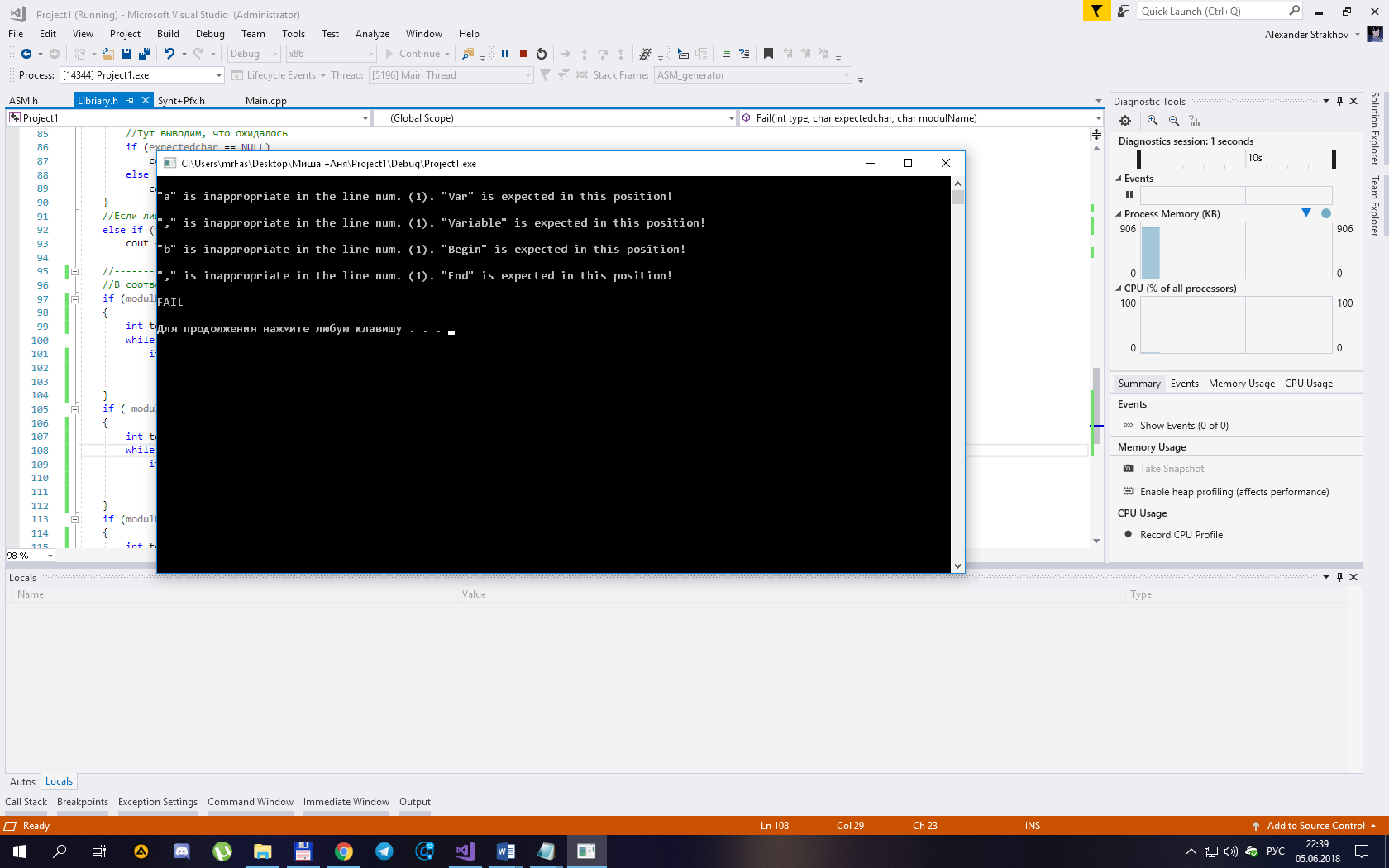
Результат:

Рисунок 5‒ Разбор некорректной последовательности

## **ГЕНЕРАЦИЯ ПОСТИКСНОЙ ЗАПИСИ**

Алгоритм перевода выражения в постфиксную запись следующий:

1. Константы и переменные кладутся в формируемую запись в порядке их появления в исходной последовательности.
2. При появлении операции в исходном массиве:
   1. если в стеке нет операций или верхним элементом стека является открывающая скобка, операции кладётся в стек;
   2. если новая операции имеет больший*[\*](http://natalia.appmat.ru/c&c++/postfisso.html" \l "nota)* приоритет, чем верхняя операции в стеке, то новая операции кладётся в стек;
   3. если новая операция имеет меньший или равный приоритет, чем верхняя операции в стеке, то операции, находящиеся в стеке, до ближайшей открывающей скобки или до операции с приоритетом меньшим, чем у новой операции, перекладываются в формируемую запись, а новая операции кладётся в стек.
3. Открывающая скобка кладётся в стек.
4. Закрывающая скобка выталкивает из стека в формируемую запись все операции до ближайшей открывающей скобки, открывающая скобка удаляется из стека.
5. При считывании операции сравнения ‘|’ генерируются две метки для последующего преобразования в машинный код.
6. После прохождения основной и альтернативной веток когда условие сравнения выполняется и когда нет, в запись добавляются маркер ‘&’ для обозначения окончания конструкции.
7. После того, как мы добрались до конца исходного выражения, операции, оставшиеся в стеке, перекладываются в формируемое выражение.

Пример генерации постфиксной записи.

|  |  |
| --- | --- |
| Var a, b , c  Begin  a=1  b=0  c=0  CASE a  OF 5:  c=a\*(b+1)  OF 6:  c=a\*(b-1)  OF 7:  c=a/(b+1)  ENDCASE  End. | a 1 =  b 0 =  c 0 =  5 a |  c a b 1 + \* =  #  6 a |  c a b 1 - \* =  #  7 a |  c a b 1 + / =  #  & |

**ГЕНЕРАЦИЯ МАШИННОГО КОДА**

Алгоритм перевода выражения в машинный код следующий.

Пока считывается 1 запись из файла с постфиксной записью:

1. Константы и переменные кладутся в стек.
2. При встрече операции «присваивания» верхнее значение стека присваивается следующему, второму элементу стека.
3. При встрече бинарных операции извлекаются два верхних элемента и записываются в регистры ax, bx. Затем производится операция, результат которой записывается в генерируемую переменную и кладется в вершину стека.
4. При встрече «логического отрицания» из стека извлекается верхний элемент и результат отрицания этого элемента записывается в стек в виде сгенерированной переменной.
5. При считывании «операции сравнения» два верхних элемента стека записываются в форме сравнения. Далее добавляются соответствующие метки переходов и происходит разбор содержимого результатов сравнения по схожему с исходным алгоритмом.

Список команд для реализаци генерации машинного кода на языке асемблер:

|  |  |
| --- | --- |
| XOR | Логическое исключающее «ИЛИ» |
| OR | Логическое «ИЛИ» |
| AND | Логическое «И» |
| NOT | Отрицание верхнего элемента стека |
| MOV | Присвавание переменной содержимого вершины стека |
| cmp | Сравнение двух верхних операций в стеке |
| JNE | При неравенстве операндов перейти к далее указанной метке |
| JMP | Безусловный переход к далее указанной метке |

Пример генерации машинного кода из исходной последовательности постфиксной записи

|  |  |
| --- | --- |
| Var a, b , c :Integer;  Begin  b=1;  c=`a;  c=`b;  b=10a-(7/2);  IF (a|1)  a = 6;  b = `11 / c;  {  ELSE  b=0;  }  End. | MOV ax, 1  MOV a, ax  MOV ax, 0  MOV b, ax  MOV ax, 0  MOV c, ax  Mark0  cmp a, 5  JNE Mark1  MOV ax, 1  MOV bx, b  ADD ax, bx  MOV t0, ax  MOV ax, t0  MOV bx, a  MUL ax, bx  MOV t1, ax  MOV ax, t1  MOV c, ax  JMP Markus  Mark1  cmp a, 6  JNE Mark2  MOV ax, 1  MOV bx, b  SUB ax, bx  MOV t0, ax  MOV ax, t0  MOV bx, a  MUL ax, bx  MOV t1, ax  MOV ax, t1  MOV c, ax  JMP Markus  Mark2  cmp a, 7  JNE Mark3  MOV ax, 1  MOV bx, b  ADD ax, bx  MOV t0, ax  MOV ax, t0  MOV bx, a  DIV ax, bx  MOV t1, ax  MOV ax, t1  MOV c, ax  JMP Markus  Markus: |

### **СПИСОК ЛИТЕРАТУРЫ**

1. Разработка трансляторов с языков САПР. Методические указания к выполнению лабораторных работ /Сост.: Кревский И.Г.; ПГТУ. - Пенза, 1993– 330 с.
2. Берри Р., Микиннз Б. Язык Си: введение для программистов: Пер. с англ. - М.: Финансы и статистика, 1988– 120 с.
3. Лэнгсам Й., Огенстайн М., Тененбаум А. Структуры данных для персональных ЭВМ: Пер. с англ. - М.: Мир, 1989 – 512 с.

### **ПРИЛОЖЕНИЕ А. ЛИСТИНГ ПРОГРАММЫ**

# «Main.cpp»

#include "Libriary.h"

#include "Synt+Pfx.h"

#include "ASM.h"

//Ф. назначает тип лексемам соостоящих из букв или цифр (лексема, флаг букв, флаг цифр)

char GivingLexType(string lexeme, bool isAlpha, bool isDigital) {

//Если в строке содержатся и цифры и буквы...

if (isAlpha && isDigital) return '0'; //возвращаем тип смешанной лексемы

//Если только буквы...

if (isAlpha)

{

//Проверка на соответствие строки с ключевыми словами записаными в хэш-массиве

for (auto going\_through : KeyWords)

if (str == going\_through.first)

return going\_through.second;

//Если соответствие не найдено, то возвращаем тип "переменная"

return '1';

}

//Если цифры...

return '2';

}

int main()

{

//Открытие файлов : с исходным кодом / для записи полученных лексем

Input.open("2.txt"); Output.open("Lexems.txt");

while (Input.get(c))

{

//--------------------------------------------------------------

if (isalpha(c))

{

alpha = true; str += c; continue;

}

else if (isdigit(c))

{

digit = true; str += c; continue;

}

//--------------------------------------------------------------

//Если флаги буквы или цифры активны

if (alpha || digit)

//Определение типа лексемы для строки

//с последующей записью в файл

Output << GivingLexType(str, alpha, digit) << ' ' << line << ' ' << str << '\n';

//Проверка на допустимость символа

if (strchr(Smbls, c))

{

//Если символ переном строки, то увеличиваем номер строки на 1

if (c == '\n')

line++;

//Так же если символ не пробел, то выводим его в файл

else if (c != ' ')

Output << c << ' ' << line << ' ' << c << '\n';

}

else

//Если символ не допустим

Output << '0' << ' ' << line << ' ' << str << '\n';

//Обнуляем флаги и строку

digit = false; alpha = false; str.clear();

}

//Отчистка i/o потоков

Input.close(); Input.clear(); Output.close(); Output.clear();

//----------------------------------------------------------------------------

//Открытие файлов : с лексемами / для постфиксной записи

Input.open("Lexems.txt");

//Синтактический анализ

S();

//Отчистка io потоков

Input.close(); Input.clear();

//----------------------------------------------------------------------------

//Если не найдены ошибки, то заносим запись в файл

if (numberOfErrors == 0){

//Открытие файла с постфиксной записью

Output.open("Postfix records.txt");

//Возвращаем значене строки с постфиксной записью

Output << pfxStr;

cout << pfxStr;

//Очищаем строки после окончания формирования постфиксной записи

pfxStr.clear();

//Отчистка io потоков

Output.close(); Output.clear();

//-----------------------------------------

Input.open("Postfix records.txt");

//Открытие файла для асм кода

Output.open("Output.txt");

//Генерация асм кода

ASM\_generator();

//Отчистка io потоков

Output.close(); Output.clear();

Input.close(); Input.clear();

cout << "\nSUCCESS\n\n";

}else

cout << "\nFAIL\n\n";

system("PAUSE"); exit(1);

}

# «Library.h»

#include <locale>

#include <string.h>

#include <iostream>

#include <stdio.h>

#include <fstream>

#include <stack>

#include <unordered\_map>

using namespace std;

//Файловые потоки

ifstream Input;

ofstream Output;

//Основные переменные для работы с записями

char c = NULL; //1 чар для работы с типами лексем

int line = 1; //Хранит номер строки

string str, //Строка для работы с содержимым лексемы

pfxStr; //Строка для накопления постфиксной записи для 1 выражения

unsigned int numberOfErrors = 0; //Cчетчик ошибок

string operatorForCMP;

//Слова, принадлежащие языку записанные в хэше

unordered\_map <string, char> KeyWords =

{

{ "Var" , 'V' },

{ "Begin" , 'B' },

{ "End" , 'E' },

{ "CASE" , 'C' },

{ "OF" , 'O' },

{ "ENDCASE", 'N' },

};

bool alpha = false, //Флаг буквы

digit = false; //Флаг цифры

//Допустимые символы в языке

char Smbls[] = "\n=(),.:/\*-+` ";

//Ф. переводит 1 "чар" в формат "строка"

string CharToString(char ConvertedChar){

string stroka;

return stroka += ConvertedChar;

}

//Ф. возвращает ожидаемою лексему определяя ее по типу, принимает (ожидаемый тип переменной)

string expectedLexeme(char expectType){

if (expectType == '1' || expectType == '2')

return "Variable";

//Если найдено соответствие, то возвращает тип лексемы соответствующей ключ. слову

for (auto going\_through : KeyWords)

if (expectType == going\_through.second)

return going\_through.first;

//Если тип не ключ. слово, то првоеряет не переменная или число ли это

//Если ничто не подошло, то тип ожидаемой лексемы это "символ"

//который предварительно преобразуется в строку

return CharToString(expectType);

}

//ф. выводит сообщение с ошибкой (тип ошибки, ожидаемый символ, название модуля из которго вызвана ошибка)

void Fail(int type, char expectedchar, char modulName){

//Итерируем в случае ошибки

numberOfErrors++;

cout << "\n";

//Если у переменной нет значения

if (type == 0)

cout << "Var \"" << str << "\" in the line num. (" << line << ") has no value!\n";

//Если переменная не объявленна

else if (type == 1)

cout << "Var \"" << str << "\" in the line num. (" << line << ") is not announced!\n";

//Если не соответсвие ожидания и реальности

else if (type == 2){

//Смотрим, если тип ошибки связан с некорректной лексемой

if (c == '0')

{

cout << "Unacceptable \"" << str << "\"";

Input >> c >> line >> str;

}

else //иначе просто мисспозиция при чтении

cout << "\"" << str << "\" is inappropriate";

//дописываем номер строки к ошибке

cout << " in the line num. (" << line << ").";

//Тут выводим, что ожидалось

if (expectedchar == NULL)

cout << '\n';

else

cout << " \"" << expectedLexeme(expectedchar) << "\" is expected in this position!\n";

}

//Если лишняя точка с запятой

else if (type == 3)

cout << "Error 03. Empty operator \";\" in the line num. (" << line << ").\n";

//----------------------------------------------------------------------------------

//В соответствии с местом вызова ошибки, пропускаем часть кода до определенного момента

if (modulName == 'A' || modulName == 'D' || modulName == 'B')

{

int temp = line;

while (Input >> c >> line >> str)

if ( c == ',' || c == '1' )

return;

}

if ( modulName == 'Z' || modulName == 'G' || modulName == 'E' || modulName == 'C' || modulName == 'H')

{

int temp = line;

while (Input >> c >> line >> str)

if (line > temp)

return;

}

if (modulName == 'H')

{

int temp = line;

while (Input >> c >> line >> str)

if (line > temp)

return;}}

# «Synt+Pfx.h»

//Здесь храним переменную для присваивания

string strForEqually;

//Mассив для хранения переменных и факт наличия у них значения

vector <string> AnnouncedVars;

//Контейнер типа "стек" для работы с постфиксной записью

stack <char> Stack;

//Для генерации меток

int mark = 0;

void G(); void C();

//Ф. определяет приоретет операций в выражении при формировании постфиксной записи

int Priority(char inch)

{

if (inch == '`') return 5;

if (inch == '|') return 4;

if (inch == '\*' || inch == '/') return 3;

if (inch == '+' || inch == '-') return 2;

if (inch == '=') return 1;

return 0;

}

//Ф. для работы со стеком в случае встречи с оператором

void forOperators()

{

//Если в стеке нет операций или верхним элементом стека является открывающая скобка или

//eсли новая операции имеет больший приоритет, чем верхняя операции в стеке, то новая операции кладётся в стек

if (Stack.empty() || Stack.top() == '(' || Priority(Stack.top()) < Priority(c))

Stack.push(c);

//Если новая операция имеет меньший или равный приоритет, чем верхняя операции в стеке, то...

else if (Priority(Stack.top()) >= Priority(c))

{

//Операции, находящиеся в стеке,

//до ближайшей открывающей скобки или до операции с приоритетом меньшим, чем у новой операции,

//перекладываются в формируемую запись,

while (!Stack.empty() && Stack.top() != '(' && !(Priority(Stack.top()) < Priority(c)))

{

pfxStr += CharToString(Stack.top()) += ' ';

Stack.pop();

}

//а новая операции кладётся в стек

Stack.push(c);

}

}

//Ф. для проверки переменной среди объявленных переменых

void checkVarAnnounced(string Var)

{

//если в "векторе" с переменными есть совпадение с переданной переменной, то возвращаемся

for (auto going\_through : AnnouncedVars)

if (Var == going\_through)

return;

//, если совпадения нет, выписываем ошибку

Fail(1, NULL, NULL);

}

//Ф. считывает лексему и применяет действии в соответствии с местом вызова

bool GetLex(char expect, char modulName)

{

if (c != expect)

return false;

if (modulName == 'D' && c == '1')

//Переменная, объявленная в Var заносится в хэш-массив как не имеющая значение

AnnouncedVars.push\_back(str);

else if (modulName == 'C' && c == '1')

{

//Проверка, объявленна ли переменная

checkVarAnnounced(str);

//Запись переменной в строку и в очередь с постф. записью

pfxStr += str += ' ';

}

else if (modulName == 'E' && c == '=')

forOperators();

else if (modulName == 'G')

switch (c)

{

case '`': forOperators(); break;

case '(': Stack.push(c); break;

case '2': pfxStr += str += ' '; break;

case '1':

//Проверка, объявленна ли переменная, проверка, имеет ли переменная значение

checkVarAnnounced(str);

pfxStr += str += ' ';

break;

case ')':

//Все операции выталкиваются в формируемую запись, до встречи с откр. скобкой

while (!Stack.empty() && Stack.top() != '(')

{

pfxStr += CharToString(Stack.top()) += ' ';

Stack.pop();

}

//Откр. скобка удаляется из стека

if (Stack.top() == '(')

Stack.pop();

break;

}

else if (modulName == 'Z' && (c == '+' || c == '-' || c == '\*' || c == '/'))

forOperators();

else if (modulName == 'H')

{

if (c == '1')

{

//Проверка, объявленна ли переменная, проверка, имеет ли переменная значение

checkVarAnnounced(str);

operatorForCMP = str;

}

else if (c == 'N')

{

pfxStr += "&\n";

}

}

else if (modulName == 'R')

{

if (c == '1' || c == '2')

{

if (c == '1') checkVarAnnounced(str);

pfxStr += str;

pfxStr += " ";

pfxStr += operatorForCMP;

pfxStr += " |\n";

}

}

//При совпадении ожидания/реальность считываем лексему

Input >> c >> line >> str;

}

//--------------------------[Синтактический разбор в соответсвии с синтактическим графом]--------------------------

void Z()

{

if (GetLex('+', 'Z') || GetLex('-', 'Z') || GetLex('\*', 'Z') || GetLex('/', 'Z')) G();

}

void G()

{

GetLex('`', 'G');

if (GetLex('(', 'G')) {

G();

if (!GetLex(')', 'G')) Fail(2, ')', 'G');

}

else if (!GetLex('2', 'G') && !GetLex('1', 'G')) Fail(2, '1', 'G');

Z();

}

void E()

{

if (!GetLex('=', 'E')) Fail(2, '=', 'E');

G();

//После того, как мы добрались до конца исходного выражения, операции, оставшиеся в стеке, перекладываются в формируемое выражение.

while (!Stack.empty())

{

pfxStr += CharToString(Stack.top()) += ' ';

Stack.pop();

}

pfxStr += '\n';

C();

}

void R()

{

if (GetLex('O', 'R'))

{

if (!GetLex('1', 'R') && !GetLex('2', 'R')) Fail(2, '1', 'R');

if (!GetLex(':', 'R')) Fail(2, ':', 'R');

C();

pfxStr += "#\n";

R();

}

}

void H()

{

if (!GetLex('1', 'H') && !GetLex('2', 'H')) Fail(2, '1', 'H');

R();

if (!GetLex('N', 'H')) Fail(2, 'N', 'H');

}

void C()

{

if (GetLex('C', 'C'))

{

H();

C();

}

else if (GetLex('1', 'C'))

{

E();

C();

}

}

void B()

{

if (!GetLex('B', 'B')) Fail(2, 'B', 'B');

C();

if (!GetLex('E', 'B')) Fail(2, 'E', 'B');

}

void D()

{

if (!GetLex('1', 'D')) Fail(2, '1', 'D');

//K

if (GetLex(',', 'D')) D();

}

void A()

{

if (!GetLex('V', 'A')) Fail(2, 'V', 'A');

D();

}

//S (Точка входа в синтактический анализатор)

void S()

{

Input >> c >> line >> str;

A();

B();

}

# «ASM.h»

//Для хранения переменных

stack <string> asmbl;

//Для генерации буферных переменных

int gen = 0;

int MARK = 0;

void ASM\_generator();

//При разборе конструкий генерация соответсвуещего асм-кода

void CASE()

{

ASM\_generator();

//Память для хранения названий меток переходов

string first\_mark, second\_mark;

//Считывание меток

Input >> first\_mark >> second\_mark;

//Запись асм\_конструкции

Output << "JNE " << first\_mark << '\n';

//Вызов генератор для разбора кода после проверки IF

ASM\_generator();

Output << "JMP " << second\_mark << '\n' << first\_mark << ":\n";

//Вызов генератор для разбора кода после проверки IF

ASM\_generator();

Output << second\_mark << ":\n";

}

void ASM\_generator()

{

Input >> str;

if (str == "=")

{

Output << "MOV ax, " << asmbl.top() << '\n';

asmbl.pop();

Output << "MOV " << asmbl.top() << ", ax\n";

asmbl.pop();

gen = 0;

}

else if (str == "+" || str == "-" || str == "/" || str == "\*")

{

string temp ="t";

temp += to\_string(gen++);

Output << "MOV ax, " << asmbl.top() << '\n';

asmbl.pop();

Output << "MOV bx, " << asmbl.top() << '\n';

asmbl.pop();

if (str == "+")

Output << "ADD";

else if (str == "-")

Output << "SUB";

else if (str == "/")

Output << "DIV";

else if (str == "\*")

Output << "MUL";

Output << " ax, bx\n";

Output << "MOV " << temp << ", ax\n";

asmbl.push(temp);

}

else if (str == "`")

{

string temp = "t";

temp += to\_string(gen++);

Output << "MOV " << temp << ',' << asmbl.top() << '\n';

asmbl.pop();

asmbl.push(temp);

Output << "NOT " << asmbl.top() << '\n';

}

else if (str == "|")

{

Output << "Mark"<< MARK << '\n';

Output << "cmp " << asmbl.top() << ", ";

asmbl.pop();

Output << asmbl.top() << '\n';

asmbl.pop();

MARK++;

Output << "JNE " <<"Mark"<< MARK << '\n';

ASM\_generator();

Output << "JMP " << "Markus" << '\n';

}

else if (str == "#")

return;

else if (str == "&")

{

Output << "Markus:\n";

return;

}

else

asmbl.push(str);

ASM\_generator();

}