**Федеральное агентство связи**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра МКиИТ

**Отчет по курсовой работе**

по дисциплине «Теория языков программирования»

Выполнили: студенты группы БВТ1801

Алмамма Башар

Кирилин Игорь Андреевич

Сидельников Никита Станиславович

Руководитель:

Кутейников Иван Алексеевич

Москва 2019

Содержание

[1. Цель работы 3](#_Toc29533915)

[2. Справочник по разработанному языку 4](#_Toc29533916)

[1) Лексика 4](#_Toc29533917)

[2) Синтаксис 5](#_Toc29533918)

[3) Классы лексем 6](#_Toc29533919)

[4) Описание грамматики входного языка расширенной формой Бэкуса-Haypa 7](#_Toc29533920)

[5) Структура разработанного транслятора 8](#_Toc29533921)

[3. Описание средств разработки 9](#_Toc29533922)

[4. Примеры тестовых программ на входном языке и результирующих программ 10](#_Toc29533923)

[5. Текст программы транслятора 14](#_Toc29533924)

[1) Лексический анализатор: 14](#_Toc29533925)

[2) Синтаксический анализатор: 18](#_Toc29533926)

[3) Оптимизатор: 22](#_Toc29533927)

[6. Вывод 29](#_Toc29533928)

# 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 80х86.

# Справочник по разработанному языку

## Лексика

1. **Элементы языка:**

1.1 Алфавит

* Строчные латинские буквы a .. z
* Арабские цифры 0 .. 9

1.2 Знаки пунктуации, арифметики и специальные символы

|  |  |  |  |
| --- | --- | --- | --- |
| **Символы** | **Наименование** | **Символы** | **Наименование** |
| **:** | равно | > | больше |
| ! | не равно | < | меньше |
| ; | точка с запятой | = | присваивание |
| ( | левая скобка | + | плюс |
| ) | правая скобка | - | минус |
| { | левая фигурная скобка | \* | умножение |
| } | правая фигурная скобка | / | деление |

1.3 Назначение символов

|  |  |
| --- | --- |
| **Символ** | **Назначение** |
| if | Условный оператор |
| while | Оператор цикла |
| print | Оператор вывода |
| = | Оператор присваивание |
| ; | Окончание выражения |
| +,-,\*,/ | Математические операторы |
| <,>,!,: | Операторы сравнения |
| (,),{,} | Разделители |
| a…z | Обозначения идентификаторов |
| 0…9 | Образование чисел |

## Синтаксис

1. **Операторы**

1.1 Условный оператор:

Имеется одна формы условного оператора:

* **if** (**выражение**) {**тело}**
* **if** (**выражение <, >, !, : выражение**) {**тело}**

**Тело** выполняется в случае, если **выражение** принимает ненулевое значение.

Пример: if (a > 5) {print a;}

1.2 Оператор цикла:

Оператор цикла с **предусловием** имеет вид:

* **while (выражение) {тело}**
* **while (выражение <, >, !, : выражение) {тело}**

При выполнении такого оператора сначала вычисляется значение **выражения**. Если оно равно 0, то оператор не выполняется и управление передается оператору, следующему за ним. Если значение выражения отлично от 0, то выполняется **тело**, затем снова вычисляется выражение и т.д.

Возможно, что тело цикла не выполнится ни разу, если выражение сразу будет равно 0.

Пример: while (a < 5) {a = a + 1;}

1.3 Оператор присваивания:

Оператор присваивания имеет вид:

* **id = выражение;**

Пример: a = 6 + 9 \* 7 – b / 2**;**

1.4 Операторы сложения | вычитания | умножения | деления

Математические операторы имеют вид:

* **id|num + ,- ,\* ,/ id|num**

Пример: a + d – 5 / 2

1.5 Операторы сравнения

Операторы сравнения имеют вид:

* **выражение <, >, !, : выражение**

Пример: a + 5 : b \* 3

2. **Правила грамматики**

2.1 Знак окончания выражения

В данном языке **;** является признаком конца выражения

2.2 Правила составления имен идентификаторов

Идентификаторы могут состоять из одного или нескольких символов **a – z**

2.3 Правила составления чисел (натуральные)

Числа могут состоять из одного или нескольких символов **0 – 9**

## Классы лексем

|  |  |
| --- | --- |
| **Описание лексемы** | **Лексема** |
| if, while, print, +, -, /, \*, <, >, !, : | KEY |
| (, ), {, }, ; | DLM |
| целые десятичные числа. последовательность цифр от 0 до 9 | NUM |
| последовательность букв a-z | ID |
| = | ASN |

## Описание грамматики входного языка расширенной формой Бэкуса-Haypa

Расширенная форма Бэкуса-Наура:

**G({ if, while, print, (, ), {, }, a-z, 0-9, =, +, -, \* , /, <, : , >, !, ;}, {S, A, B, C, F, V, N}, P, S)**

**P:**

**S-> if(A){S}S | while(A){S}S | print A;S | V = B;S| | ;**

**A -> B < B | B : B | B ! B | B > B | C | B**

**B-> C+B | C-B | C\*B | C/B | С**

**C-> V | N**

**V-> aD-zD | a – z**

**N->0N 9N | 0 - 9**

## Структура разработанного транслятора

На рисунке 1 представлена структура разработанного транслятора.

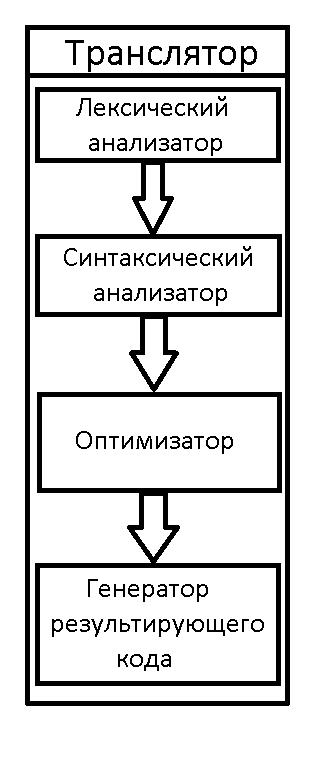
****

Рис 1. Структура разработанного транслятора.

# Описание средств разработки

1. Среда разработки – Visual Studio 2019;
2. Среда разработки для компилирования кода Assembler – flat\_assembler
3. Языки программирования:

* VS C++ (лексический);
* VS C++(синтаксический);
* Fasm [16 бит, процессор 8086, Assembler] (оптимизатор)

1. Программное обеспечение Visual Studio для компилирования языка VS C++;

**Проблемы, возникшие при разработке транслятора и способы их решения**

1. При создании синтаксического анализатора стояла задача создать правильную обработку входящего потока токенов. Чтобы это реализовать необходимо было графическое представление грамматики языка. С помощью операторов switch - case, поток водящих токенов анализировался аналогично графическому представлению.
2. Так как был использован 16-битный язык ассемблера FASM, то возникла проблема с воспроизведением программы. Файлы создавались не в .exe формате, а в более старом - .com. Чтобы их воспроизводить была установлена виртуальная машина dosBox, эмулирующая работу операционной системы Dos.

# Примеры тестовых программ на входном языке и результирующих программ

На рисунке 2 представлен пример работы лексического анализатора, обнаружение ошибочных действий.

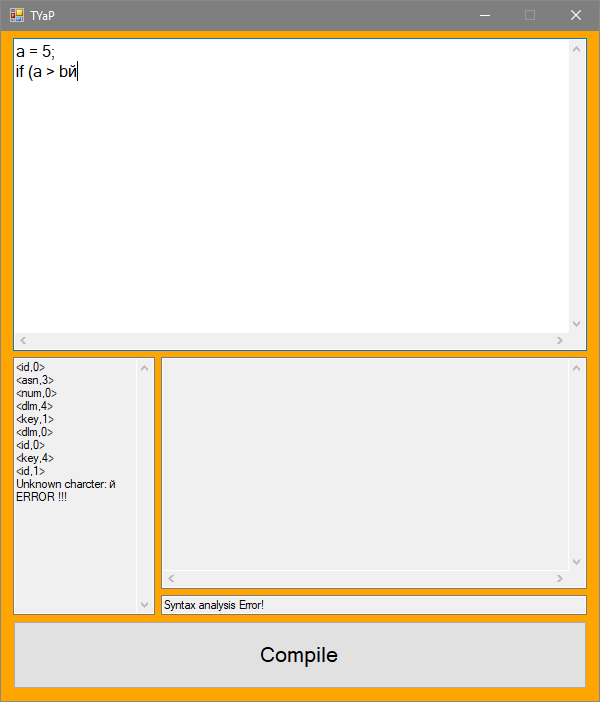


Рис 2. Пример работы лексического анализатора, обнаружение ошибочных действий.

На рисунке 3 представлен пример работы синтаксического анализатора, обнаружение ошибочных действий.

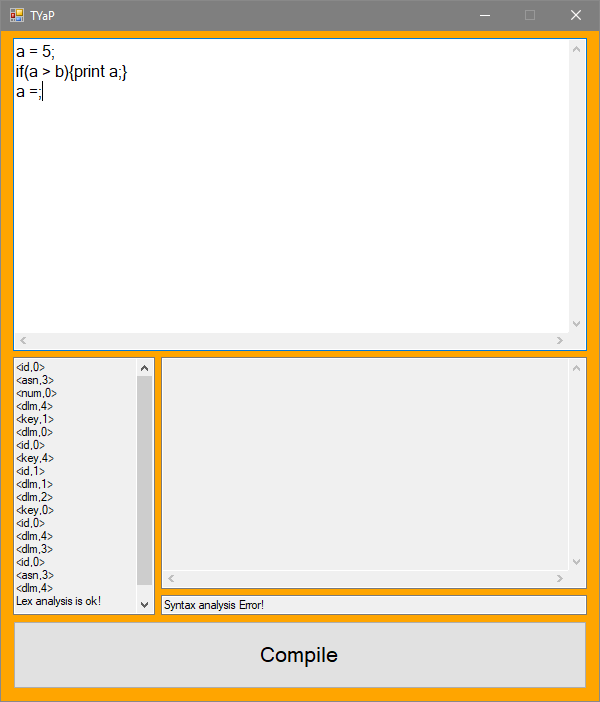


Рис 3. Пример работы синтаксического анализатора, обнаружение ошибочных действий.

На рисунке 4 представлен пример работы анализаторов и оптимизатора.

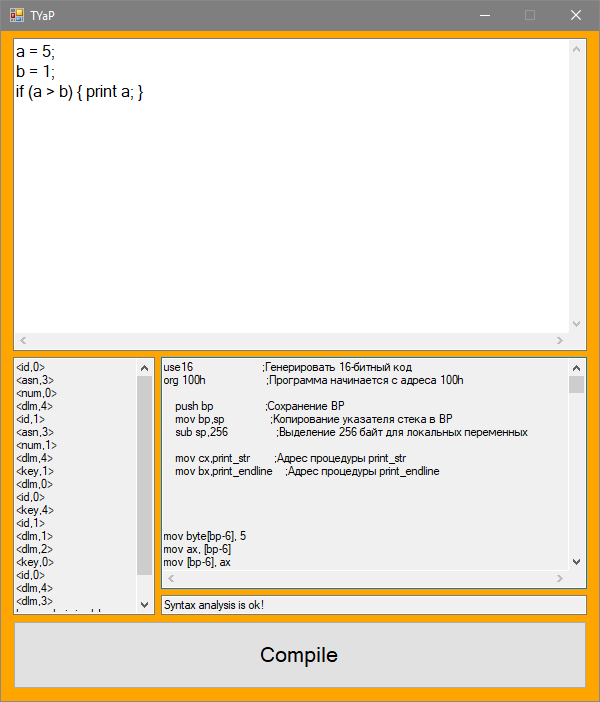


Рис 4. Пример работы анализаторов и оптимизатора.

На рисунке 5 представлен пример работы программы на созданном языке.

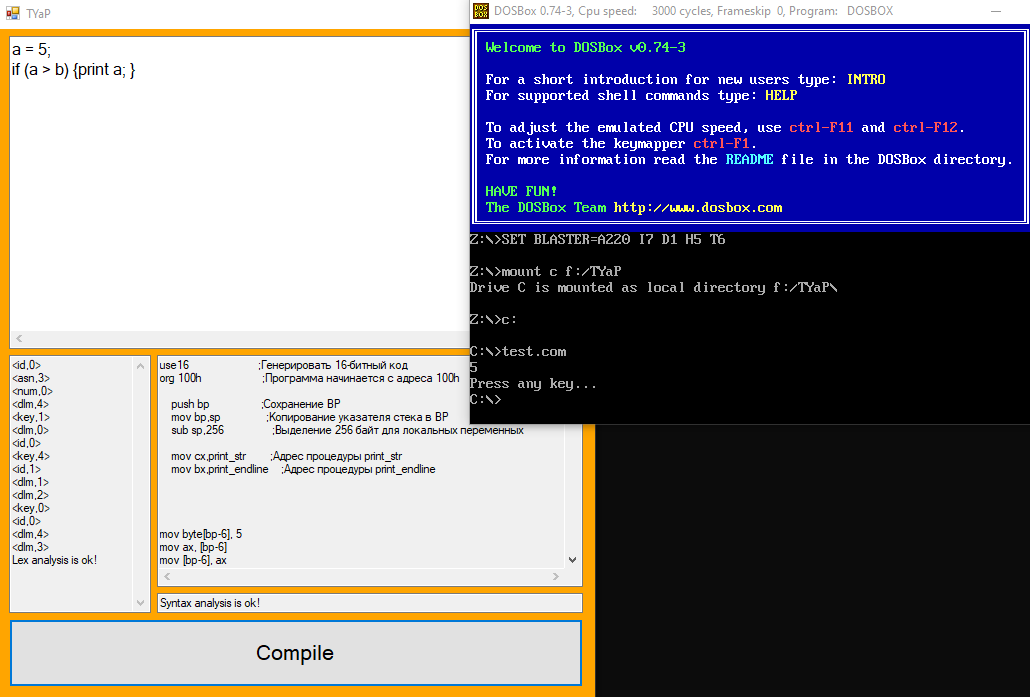
****

Рис 5. Пример работы программы на созданном языке.

# Текст программы транслятора

## Лексический анализатор:

String^ id\_or\_word(String^ buf, List<String^>^ TKey, List<String^>^ TID, List<String^>^ Name, List<int>^ Id, int &n) {

for (int i = 0; i < TKey->Count; i++) {

if (TKey[i] == buf) {

Name->Add("key"); Id->Add(i);

if (i >= 1 && i <= 2)n++;

return"<key," + i + ">\r\n";

}

}

for (int i = 0; i < TID->Count; i++) {

if (TID[i] == buf) {

Name->Add("id"); Id->Add(i);

return"<id," + i + ">\r\n";

}

}

TID->Add(buf);

Name->Add("id"); Id->Add(TID->Count - 1);

return "<id," + (TID->Count - 1) + ">\r\n";

}

bool scan(String^ Input, TextBox^ Output) {

enum state { ER, H, ID, NUM, ASN, DLM, END, IDAsn, MATH, DLMPhase, KEY, MATHKey, IDKey, NUMKey };

enum state TC; /\* текущее состояние \*/

String^ sOutput = "";

int key\_num = 0;

TID = gcnew List<String^>(0); // таблица идентификаторов анализируемой программы;

TNUM = gcnew List<String^>(0); // таблица чисел-констант, используемых в программе;

LexName = gcnew List<String^>(0); // таблица имен лексем

Id = gcnew List<int>(0); // таблица id лексем

TKey = gcnew List<String^>(0); // таблица ключевых слов входного языка;

TD = gcnew List<char>(0); // таблица разделителей входного языка;

String^ buf; /\* для накопления символов лексемы \*/

TKey->Add("print");

TKey->Add("if");

TKey->Add("while");

TKey->Add("=");

TKey->Add(">");

TKey->Add("<");

TKey->Add("!");

TKey->Add(":");

TKey->Add("+");

TKey->Add("-");

TKey->Add("\*");

TKey->Add("/");

TD->Add('(');

TD->Add(')');

TD->Add('{');

TD->Add('}');

TD->Add(';');

//л. анализатор

TC = H;

int j = 0;

while (j < Input->Length) {

switch (TC) {

case H:

buf = "";

if (Input[j] == ' ' || Input[j] == '\n' || Input[j] == '\r' || Input[j] == '\t')j++;

else if ((Input[j] >= 97 && Input[j] <= 122) || (Input[j] >= 65 && Input[j] <= 90)) {

buf += Convert::ToChar(Input[j]);

j++;

TC = ID;

}

else if (Input[j] == '>' || Input[j] == '<' ||

Input[j] == '!' || Input[j] == ':' ||

Input[j] == '-' || Input[j] == '\*' || Input[j] == '/' || Input[j] == '+') {

buf += Convert::ToChar(Input[j]);

j++;

sOutput += id\_or\_word(buf, TKey, TID, LexName, Id, key\_num);

TC = H;

}

else if (Input[j] >= 48 && Input[j] <= 57) {

buf += Convert::ToChar(Input[j]);

j++;

TC = NUM;

}

else if (Input[j] == '=') {

buf += Convert::ToChar(Input[j]);

j++;

TC = ASN;

}

else if (Input[j] == '|') {

TC = END;

}

else TC = DLM;

break;

case ID:

if ((Input[j] >= 97 && Input[j] <= 122) || (Input[j] >= 65 && Input[j] <= 90)) {

buf += Convert::ToChar(Input[j]);

j++;

}

else {

sOutput += id\_or\_word(buf, TKey, TID, LexName, Id, key\_num);

TC = H;

}

break;

case NUM:

if ((Input[j] >= 48 && Input[j] <= 57) || Input[j] == 46) {

buf += Convert::ToChar(Input[j]);

j++;

}

else {

bool flag = false;

for (int i = 0; i < TNUM->Count; i++) {

if (TNUM[i] == buf) {

sOutput += "<num," + i + ">\r\n";

LexName->Add("num"); Id->Add(i);

flag = true;

}

}

if (!flag) {

TNUM->Add(buf);

sOutput += "<num," + (TNUM->Count - 1) + ">\r\n";

LexName->Add("num"); Id->Add(TNUM->Count - 1);

}

TC = H;

}

break;

case ASN:

for (int i = 0; i < TKey->Count; i++) {

if (TKey[i] == buf) {

LexName->Add("asn"); Id->Add(i);

sOutput += "<asn," + i + ">\r\n";

}

}

TC = H;

break;

case DLM:

bool flag = false;

for (int i = 0; i < TD->Count; i++) {

if (TD[i] == Input[j]) {

sOutput += "<dlm," + i + ">\r\n";

LexName->Add("dlm"); Id->Add(i);

j++;

TC = H;

flag = true;

}

}

if (!flag) {

sOutput += "Unknown charcter: " + Input[j] + "\r\n";

TC = ER;

j++;

}

break;

}

if (TC == END || TC == ER)break;

}

if (TC == ER) {

sOutput += "ERROR !!!\r\n";

}

else sOutput += "Lex analysis is ok!\r\n";

Output->Text = sOutput;

if (TC == ER)return 0;

//////продолжения функции//////

}

## Синтаксический анализатор:

//////начало функции//////

//с. анализатор

TC = H;

int key\_sk = 0;

int key\_sf = 0;

int sk = 0;

int sf = 0;

{

bool Operflag = false;

for (int i = 0; i < LexName->Count; i++) {

switch (TC) {

case H:

if (LexName[i] == "id") {

TC = IDAsn;

}

else if (LexName[i] == "key" && Id[i] == 0) {

TC = MATH;

}

else if (LexName[i] == "key" && Id[i] >= 1 && Id[i] <= 2) {

TC = KEY;

}

else {

TC = ER;

}

break;

case IDAsn:

if (LexName[i] == "asn") {

TC = MATH;

}

else {

TC = ER;

}

break;

case DLM:

if (sk < 0 && sf < 0) {

TC = ER;

break;

}

if (LexName[i] == "id") {

TC = IDKey;

}

else if (LexName[i] == "num") {

TC = NUMKey;

}

else {

TC = ER;

}

break;

case NUM:

if (Id[i] >= 8 && Id[i] <= 11) {

TC = MATH;

}

else if (LexName[i] == "dlm" && TD[Id[i]] == ';') {

TC = DLMPhase;

}

else {

TC = ER;

}

break;

case ID:

if (Id[i] >= 8 && Id[i] <= 11) {

TC = MATH;

}

else if (LexName[i] == "dlm" && TD[Id[i]] == ';') {

TC = DLMPhase;

}

else {

TC = ER;

}

break;

case KEY:

if (LexName[i] == "dlm" && TD[Id[i]] == '(') {

sk++;

key\_sk++;

TC = DLM;

}

else {

TC = ER;

}

break;

case MATH:

if (LexName[i] == "id") {

TC = ID;

}

else if (LexName[i] == "num") {

TC = NUM;

}

else {

TC = ER;

}

break;

case DLMPhase:

if (sk == 0 && LexName[i] == "dlm" && TD[Id[i]] == '{') {

if (LexName[i - 1] != "dlm" || TD[Id[i - 1]] != ')') {

TC = ER;

break;

}

sf++;

key\_sf++;

TC = DLMPhase;

}

else if (sk == 0 && LexName[i] == "dlm" && TD[Id[i]] == '}') {

sf--;

TC = DLMPhase;

}

else if (sk == 0 && LexName[i] == "dlm" && TD[Id[i]] == ';') {

TC = DLMPhase;

}

else if (sk == 0 && LexName[i] == "id") {

TC = IDAsn;

}

else if (LexName[i] == "key" && Id[i] == 0) {

TC = MATH;

}

else if (sk == 0 && Id[i] >= 1 && Id[i] <= 2 && LexName[i] == "key") {

TC = KEY;

}

else {

TC = ER;

}

break;

case ER:

textBox2->Text = "ERROR: " + i + "\r\n";

i = LexName->Count;

break;

case NUMKey:

if (Id[i] >= 8 && Id[i] <= 11) {

TC = MATHKey;

}

else if (LexName[i] == "dlm" && TD[Id[i]] == ')') {

sk--;

Operflag = false;

TC = DLMPhase;

}

else if (LexName[i] == "key" && Id[i] >= 4 && Id[i] <= 7 && !Operflag) {

Operflag = true;

TC = MATHKey;

}

else {

TC = ER;

}

break;

case IDKey:

if (Id[i] >= 8 && Id[i] <= 11) {

TC = MATHKey;

}

else if (LexName[i] == "dlm" && TD[Id[i]] == ')') {

sk--;

Operflag = false;

TC = DLMPhase;

}

else if (LexName[i] == "key" && Id[i] >= 4 && Id[i] <= 7 && !Operflag) {

Operflag = true;

TC = MATHKey;

}

else {

TC = ER;

}

break;

case MATHKey:

if (LexName[i] == "id") {

TC = IDKey;

}

else if (LexName[i] == "num") {

TC = NUMKey;

}

else {

TC = ER;

}

break;

}

}

if (TC != DLMPhase)TC = ER;

}

if (sf != 0 || sk != 0 || key\_sk != key\_num || key\_sf != key\_num) {

textBox2->Text = "ERROR: with '{}' or '()'";

return 0;

}

return TC;

}

## Оптимизатор:

String^ MovAxID(int variable\_id) {

String^ command = "mov ax, [bp-" + 2 \* (variable\_id + 1) + "]\r\n";

return command;

}

String^ MovAxNUM(int num) {

String^ command = "mov byte[bp-" + 2 \* (TID->Count + 1) + "], " + num + "\r\n";

command += "mov ax, [bp-" + 2 \* (TID->Count + 1) + "]\r\n";

return command;

}

String^ OperAxIDNUM(char op, int variable, bool ID\_NUM) {

String^ oper = "";

String^ command = "";

switch (op)

{

case '+':

oper = "add";

break;

case '-':

oper = "sub";

break;

case '\*':

oper = "imul";

break;

case '/':

if (ID\_NUM) command = "mov ax, [bp-" + 2 \* (variable + 1) + "]\r\n";

else command = "mov ax, " + (variable)+"\r\n";

command += "idiv dl \r\n";

return command;

///////////////

break;

default:

break;

}

if (ID\_NUM) command = oper + " ax, [bp-" + 2 \* (variable + 1) + "]\r\n";

else command = oper + " ax, " + (variable) + "\r\n";

return command;

}

String^ MovIDAx(int variable\_id, bool AX\_ONLY) {

String^ command = "";

command += "mov [bp-" + 2 \* (TID->Count + 1) + "], ax\r\n";

command += "mov ax, [bp-" + 2 \* (TID->Count + 1) + "]\r\n";

if (!AX\_ONLY)command += "mov [bp-" + 2 \* (variable\_id + 1) + "], ax\r\n";

return command;

}

String^ OperMetka(char op, int metka, bool single) {

String^ oper = "";

String^ command = "";

if (!single) {

switch (op)

{

case '>':

//oper = "jg";

oper = "jle";

break;

case '<':

//oper = "jl";

oper = "jge";

break;

case ':':

//oper = "jz";

oper = "jnz";

break;

case '!':

//oper = "jnz";

oper = "jz";

break;

default:

break;

}

command += "cmp [bp-" + 2 \* (TID->Count + 2) + "], ax\r\n";

command += oper + " metka" + metka + "\r\n";

}

else {

//command += "cmp [bp-" + 2 \* (TID->Count + 2) + "], 0\r\n";

command += "cmp ax, 0\r\n";

command += "jz metka" + metka + "\r\n";

}

//////////////////

return command;

}

String^ PrintAx() {

String^ command = "";

command += "mov di, s\_sword\r\n";

command += "call cx\r\n";

command += "call print\_word\_sdec\r\n";

command += "call bx\r\n";

return command;

}

String^ Asm() {

List<int>^ LastKey = gcnew List<int>(0);

List<int>^ Metki = gcnew List<int>(0);

List<int>^ WhileMetki = gcnew List<int>(0);

int variable = 0;

int last\_oper = 0;

int last\_srav\_oper = 0;

int closed\_sf = 0;

int while\_closed\_sf = 0;

String^ AssemblerCode = "";

enum state { H, MATH, IF, WHILE, MATHKEY, PRINT };

enum state TC; /\* текущее состояние \*/

LastKey->Add(0);

TC = H;

for (int i = 0; i < LexName->Count; i++) {

switch (TC)

{

case H:

if (LexName[i] == "id") {

variable = Id[i];

TC = MATH;

}

else if (LexName[i] == "key" && TKey[Id[i]] == "if") {

LastKey->Add(2);

TC = IF;

}

else if (LexName[i] == "key" && TKey[Id[i]] == "while") {

AssemblerCode += "wmetka" + WhileMetki->Count + ":\r\n";

WhileMetki->Add(WhileMetki->Count);

LastKey->Add(3);

TC = WHILE;

}

else if (LexName[i] == "key" && TKey[Id[i]] == "print") {

TC = PRINT;

}

else if (LexName[i] == "dlm" && TD[Id[i]] == '}') {

LastKey->RemoveAt(LastKey->Count - 1);

if (LastKey[LastKey->Count - 1] == 3) {

AssemblerCode += "jmp wmetka" + (WhileMetki->Count - while\_closed\_sf - 1) + "\r\n";

while\_closed\_sf++;

if (LastKey->Count == 0) {

return "";

}

LastKey->RemoveAt(LastKey->Count - 1);

}

AssemblerCode += "metka" + (Metki->Count - closed\_sf - 1) + ":\r\n";

closed\_sf++;

AssemblerCode += "\r\n";

TC = H;

}

break;

case MATH:

if (LexName[i] == "id" && last\_oper != 0) {

AssemblerCode += OperAxIDNUM(Convert::ToChar(TKey[last\_oper]), Id[i], true);

TC = MATH;

}

else if (LexName[i] == "num" && last\_oper != 0) {

AssemblerCode += OperAxIDNUM(Convert::ToChar(TKey[last\_oper]), Convert::ToDouble(TNUM[Id[i]]), false);

TC = MATH;

}

else if (LexName[i] == "id") {

AssemblerCode += MovAxID(Id[i]);

TC = MATH;

}

else if (LexName[i] == "num") {

AssemblerCode += MovAxNUM(Convert::ToDouble(TNUM[Id[i]]));

TC = MATH;

}

else if (LexName[i] == "key" && Id[i] >= 8 && Id[i] <= 11) {

last\_oper = Id[i];

TC = MATH;

}

else if (LexName[i] == "dlm" && TD[Id[i]] == ';') {

AssemblerCode += MovIDAx(variable, false);

last\_oper = 0;

AssemblerCode += "\r\n";

TC = H;

}

break;

case IF:

if (LexName[i] == "dlm" && TD[Id[i]] == '(') {

TC = MATHKEY;

}

else if (LexName[i] == "dlm" && TD[Id[i]] == '{') {

if (last\_srav\_oper == 0)AssemblerCode += OperMetka(Convert::ToChar(TKey[10]), Metki->Count, true);

else AssemblerCode += OperMetka(Convert::ToChar(TKey[last\_srav\_oper]), Metki->Count, false);

Metki->Add(Metki->Count);

last\_srav\_oper = 0;

AssemblerCode += "\r\n";

TC = H;

}

break;

case MATHKEY:

if (LexName[i] == "id" && last\_oper != 0) {

AssemblerCode += OperAxIDNUM(Convert::ToChar(TKey[last\_oper]), Id[i], true);

TC = MATHKEY;

}

else if (LexName[i] == "num" && last\_oper != 0) {

AssemblerCode += OperAxIDNUM(Convert::ToChar(TKey[last\_oper]), Convert::ToDouble(TNUM[Id[i]]), false);

TC = MATHKEY;

}

else if (LexName[i] == "id") {

AssemblerCode += MovAxID(Id[i]);

TC = MATHKEY;

}

else if (LexName[i] == "num") {

AssemblerCode += MovAxNUM(Convert::ToDouble(TNUM[Id[i]]));

TC = MATHKEY;

}

else if (LexName[i] == "key" && Id[i] >= 8 && Id[i] <= 11) {

last\_oper = Id[i];

TC = MATHKEY;

}

else if (LexName[i] == "key" && Id[i] >= 4 && Id[i] <= 7) {

AssemblerCode += MovIDAx(TID->Count + 1, false);

last\_srav\_oper = Id[i];

last\_oper = 0;

TC = MATHKEY;

}

else if (LexName[i] == "dlm" && TD[Id[i]] == ')') {

AssemblerCode += MovIDAx(0, true);

last\_oper = 0;

TC = IF;

}

break;

case WHILE:

if (LexName[i] == "dlm" && TD[Id[i]] == '(') {

LastKey->Add(2);

TC = MATHKEY;

}

break;

case PRINT:

if (LexName[i] == "id" && last\_oper != 0) {

AssemblerCode += OperAxIDNUM(Convert::ToChar(TKey[last\_oper]), Id[i], true);

TC = PRINT;

}

else if (LexName[i] == "num" && last\_oper != 0) {

AssemblerCode += OperAxIDNUM(Convert::ToChar(TKey[last\_oper]), Convert::ToDouble(TNUM[Id[i]]), false);

TC = PRINT;

}

else if (LexName[i] == "id") {

AssemblerCode += MovAxID(Id[i]);

TC = PRINT;

}

else if (LexName[i] == "num") {

AssemblerCode += MovAxNUM(Convert::ToDouble(TNUM[Id[i]]));

TC = PRINT;

}

else if (LexName[i] == "key" && Id[i] >= 8 && Id[i] <= 11) {

last\_oper = Id[i];

TC = PRINT;

}

else if (LexName[i] == "dlm" && TD[Id[i]] == ';') {

AssemblerCode += MovIDAx(variable, true);

AssemblerCode += PrintAx();

AssemblerCode += "\r\n";

last\_oper = 0;

TC = H;

}

break;

default:

break;

}

}

return AssemblerCode;

}

# Вывод

В ходе курсовой работы были изучены основы работы транслятора, который состоит из лексического и синтаксического анализатора, оптимизатора кода программы в ассемблер и реализован при помощи языка С++ и программного обеспечения Microsoft Visual Studio. В процессе создания транслятора так же приобретен навык работе в команде и распределения работы между участниками проекта.