

РОСЖЕЛДОР
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Ростовский государственный университет путей сообщения»
(ФГБОУ ВО РГУПС)

Кафедра «ВТ и АСУ»

«Разработка компилятора для языка программирования»

Пояснительная записка
к курсовой работе по дисциплине
«Системное программное обеспечение вычислительных
систем»
СПО 01.11 ПЗ

Учебная группа АВБ-3-033

Выполнил студент Захаренко Н.С.


(подпись студента)

Руководитель курсовой работы


(подпись)

Жуков В. В.

Работа допущена к защите 12.05.22
(дата)

Работа защищена 14.05.22 с оценкой удов.
(дата) (подпись руководителя)

г. Ростов – на – Дону
2022 г.

РОСЖЕЛДОР
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Ростовский государственный университет путей сообщения»
(ФГБОУ ВО РГУПС)

УТВЕРЖДАЮ

Зав. кафедрой

О.В. Игнатьева

«07» февраля 20 22

Задание на курсовую работу (проект) № _____

Кафедра: Кафедра Вычислительная техника и автоматизированные системы управления
Специальность: 09.03.01 Информатика и вычислительная техника
Форма обучения: очная
Дисциплина: Системное программное обеспечение вычислительных систем
Вид работы: Курсовая работа
Группа: АВБ-033
Обучающийся: Захаренко Никита Сергеевич
Тема работы: Разработка простейшего компилятора с языка подобного Паскалю

Исходные данные: Операция size of. С помощью операции size of можно определить размер памяти который соответствует идентификатору или типу. Операция size of имеет возвращающий формат: size of (выражение). В качестве выражения может быть использован любой идентификатор, либо имя типа, заключенное в кавычки. Если в качестве выражения указано имя массива, то результатом является размер всего массива (т.е. произведение числа элементов на длину типа), а не размер указателя, соответствующего идентификатору массива.

Руководитель курсовой работы (проекта)

Жуков В.В.

Дата выдачи задания

«07» февраля 20 22

Задание получил

Захаренко Н.С.

«07» февраля 20 22

Содержание

ВВЕДЕНИЕ	4
1 ТЕХНИЧЕСКОЕ ЗАДАНИЕ	5
1.1 НАЗНАЧЕНИЕ РАЗРАБОТКИ.....	5
1.2 ТРЕБОВАНИЯ К ПРОГРАММЕ.....	5
1.3 ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ	5
1.4 ФУНКЦИОНАЛЬНОЕ НАЗНАЧЕНИЕ	6
1.5 СТАДИИ И ЭТАПЫ РАЗРАБОТКИ.....	6
1.6 ОПИСАНИЕ ЛОГИЧЕСКОЙ СТРУКТУРЫ.....	7
2 РАЗРАБОТКА ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА.....	10
2.1 ЛЕКСИЧЕСКИЙ АНАЛИЗАТОР.....	10
2.2 ТАБЛИЦЫ ИДЕНТИФИКАТОРОВ	12
2.3 СПЕЦИФИКАЦИИ ФУНКЦИЙ ЛЕКСИЧЕСКОГО АНАЛИЗАТОРА.....	15
3 РАЗРАБОТКА СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА	17
3.1 СИНТАКСИЧЕСКИЙ АНАЛИЗАТОР	17
3.2 СПЕЦИФИКАЦИИ ФУНКЦИЙ СИНТАКСИЧЕСКОГО АНАЛИЗАТОРА	18
3.3 ГЕНЕРАЦИЯ КОДА	19
3.4 ОБРАБОТКА ОШИБОК СИНТАКСИЧЕСКОГО АНАЛИЗА	20
3.5 ВЫЗОВ И ЗАГРУЗКА	21
3.6 ВХОДНЫЕ И ВЫХОДНЫЕ ДАННЫЕ	21
4 ТЕСТИРОВАНИЕ ПРОГРАММЫ КОМПИЛЯТОРА.....	22
4.1 КОНТРОЛЬНЫЙ ПРИМЕР	22
4.2 РЕЗУЛЬТАТ РАБОТЫ ПРОГРАММЫ.....	23
ЗАКЛЮЧЕНИЕ	24
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	25
ПРИЛОЖЕНИЕ 1 ЛИСТИНГ КОМПИЛЯТОРА.....	26
ПРИЛОЖЕНИЕ 2 ЛИСТИНГ ПРОГРАММЫ НА ЯЗЫКЕ АССЕМБЛЕРА.....	63

					<div style="text-align: center;"> <p><i>Разработка компилятора</i></p> </div>		
<i>Изм.</i>	<i>Лист</i>	<i>№ докум.</i>	<i>Подпис</i>	<i>Дат</i>			
Разработал	Захаренко				<div style="text-align: center;"> <p>РГУПС Кафедра ВТ и АСУ</p> <p>Группа АВБ-3-033</p> </div>		
Проверил	Жуков						
Н.контр.							
Утвердил							
					Стадия	Лист	Листов
						3	77

Введение

Компилятор – это не что иное, как переводчик исходного кода.

Большинство компиляторов переводят программу с некоторого высокоуровневого языка программирования в машинный код, который может быть непосредственно выполнен компьютером.

Целью курсовой работы является изучение составных частей, основных принципов построения и функционирования компиляторов, практическое освоение методов построения грамматики и простейших компиляторов.

В курсовой работе необходимо разработать компилятор и дополнительно реализовать возможности, предусмотренные индивидуальным вариантом.

Результатам курсовой работы является программная реализация заданного компилятора и пояснительная записка, оформленная в соответствии с требованиями стандартов и задания на курсовую работу.

1 Техническое задание

1.1 Назначение разработки

Компилятор представляет программу, которая осуществляет перевод исходной программы на входном языке в эквивалентную ей объектную программу на языке машинных команд или языке ассемблере.

Основные функции компилятора:

- 1 проверка исходной цепочки символов на принадлежность к входному языку;
- 2 генерация выходной цепочки символов на языке машинных команд или ассемблере.

1.2 Требования к программе

Код исходной программы должен соответствовать разработанной грамматике, которая включает данные целого, логического, символьного типов, одномерный массив элементов простого типа, литералы, а также операторы: присваивание, цикл while, условный, ввода-вывода, блок оператора, а также в соответствии с индивидуальным заданием необходимо реализовать операцию sizeof.

1.3 Общие сведения о программе

Компилятор на основе исходного текста программы, генерирует текст на машинном языке, который способен понимать компьютер.

Разработанная программа является однопроходным компилятором и применяется для компиляции на язык ассемблера.

Для работы данного приложения необходимо иметь Microsoft Visual Studio, для редактирования на языке высокого уровня

1.4 Функциональное назначение

Приложение используется для перевода с высокоуровневого языка на язык ассемблера.

Пользователь вводит код на языке предусмотренном грамматикой, а программа переводит его в ассемблерный.

Так же приложение сообщает об ошибках при наборе программы и её выполнении.

Входными данными для программы являются текстовый файл с кодом программы написанном на исходной грамматике.

Выходными данными для программы являются текстовый файл с объектной программой на языке ассемблера.

Данный программный продукт ограничен функционалом разработанной грамматики.

1.5 Стадии и этапы разработки

При разработке любой программы необходимо руководствоваться следующим алгоритмом работы:

1 Постановка задачи:

- сбор информации о задаче;
- формулировка условия задачи;
- определение конечных целей решения задачи;
- определение формы выдачи результатов;
- описание данных (их типов, диапазонов величин, структуры и т. п.).

2 Анализ и исследование задачи, модели:

- анализ существующих аналогов;
- анализ технических и программных средств;
- разработка математической модели;
- разработка структур данных.

3 Разработка алгоритма:

- выбор метода проектирования алгоритма;
- выбор формы записи алгоритма (блок-схемы, псевдокод и др.);
- выбор тестов и метода тестирования;
- проектирование алгоритма.

4 Программирование:

- выбор языка программирования;
- уточнение способов организации данных;
- запись алгоритма на выбранном языке программирования.

5 Тестирование и отладка:

- синтаксическая отладка;
- отладка семантики и логической структуры;
- тестовые расчеты и анализ результатов тестирования;
- совершенствование программы.

6 Анализ результатов решения задачи и уточнение в случае необходимости математической модели с повторным выполнением этапов 2-5.

7 Сопровождение программы:

- доработка программы для решения конкретных задач;
- составление документации к решенной задаче, к математической модели, к алгоритму, к программе, к набору тестов, к использованию.

Стадии разработки компилятора:

- 1 Разработка лексического анализатора.
- 2 Разработка процедур рекурсивного спуска синтаксического анализа.
- 3 Генерация кода и сборка компилятора.

1.6 Описание логической структуры

Рассмотрим структуру программы представленной на рисунке. 1.1 с описанием функций составных частей и используемые методы. Программа включает в себя лексический и синтаксический анализаторы, некоторые

семантические действия и генерацию кода. Рассмотрим по порядку каждый из них более подробно.

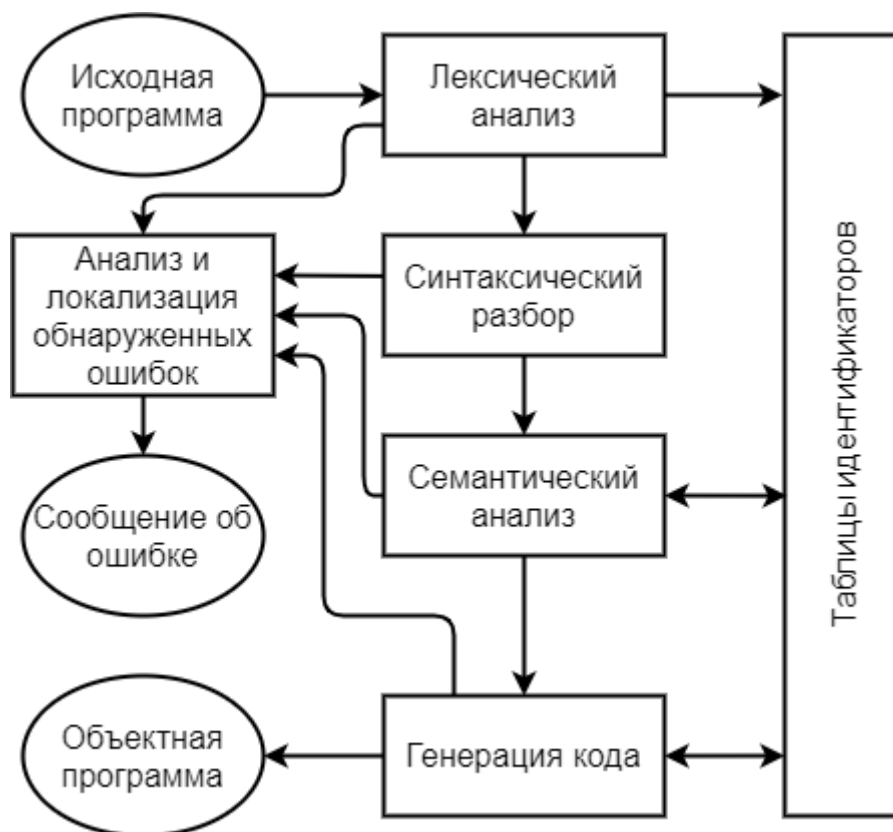


Рисунок 1.1 – Структура программы

Лексический анализ является основной частью компилятора, которая читает литеры программы на исходном языке и строит из них лексемы исходного языка, создает и заполняет таблицы идентификаторов. Таблицы идентификаторов – это специальным образом организованные наборы данных, служащие для хранения информации об элементах исходной программы, которые затем используются для порождения текста результирующей программы. На вход лексического анализатора поступает текст исходной программы, а выходная информация передаётся для дальнейшей обработки компилятором на этапе синтаксического разбора.

Синтаксический разбор — это основная часть компилятора, он выполняет выделение синтаксических конструкций в тексте исходной программы, обработанной лексическим анализатором. На этой же фазе компиляции проверяется синтаксическая правильность программы. Семантический анализ, проверяет правильность текста исходной программы с точки зрения семантики

входного языка. Также семантический анализ должен выполнять преобразования текста, требуемые семантикой входного языка. В реализации компилятора семантический анализ входит в синтаксический разбор. На основании внутреннего представления программы и информации, содержащейся в таблицах идентификаторов, порождается текст результирующей программы. Результатом этого этапа является объектный код.

Генерация кода непосредственно связана с порождением команд, составляющих предложения выходного языка и в целом текста результирующей программы.

Кроме того, в составе компилятора присутствует часть, ответственная за анализ и исправление ошибок, которая при наличии ошибки в тексте исходной программы должна максимально полно информировать пользователя о типе ошибки и месте её возникновения.

2 Разработка лексического анализатора

2.1 Лексический анализатор

Лексический анализ — процесс аналитического разбора входной последовательности символов (например, такой как исходный код на одном из языков программирования) с целью получения на выходе последовательности символов, называемых «токенами»

(подобно группировке букв в слова). Группа символов входной последовательности, идентифицируемая на выходе процесса как токен, называется лексемой. В процессе лексического анализа производится распознавание и выделение лексем из входной последовательности символов. Структура реализации токена в программе компилятора описана в фрагменте 2.1.

```
struct Token
{
    char table; //имя таблицы
    int num; //номер в таблице
};
```

Фрагмент 2.1 – Код структуры токена

С точки зрения теории токен образует лексемы, которые неразличимы при синтаксическом анализе.

Выделим следующие токены:

- идентификаторы – I;
- ключевые слова – K;
- литералы – L;
- числа – C;
- Однолитерные разделители – S.
- Двухлитерные разделители – D.
- Символы – O.

Лексический анализатор (или сканер) будет производить разбор исходного текста в соответствии со следующим графом рисунок 2.1. Сканер задан конечным детерминированным автоматом где:

0 – начальное состояние автомата;

END – конечное состояние автомата.

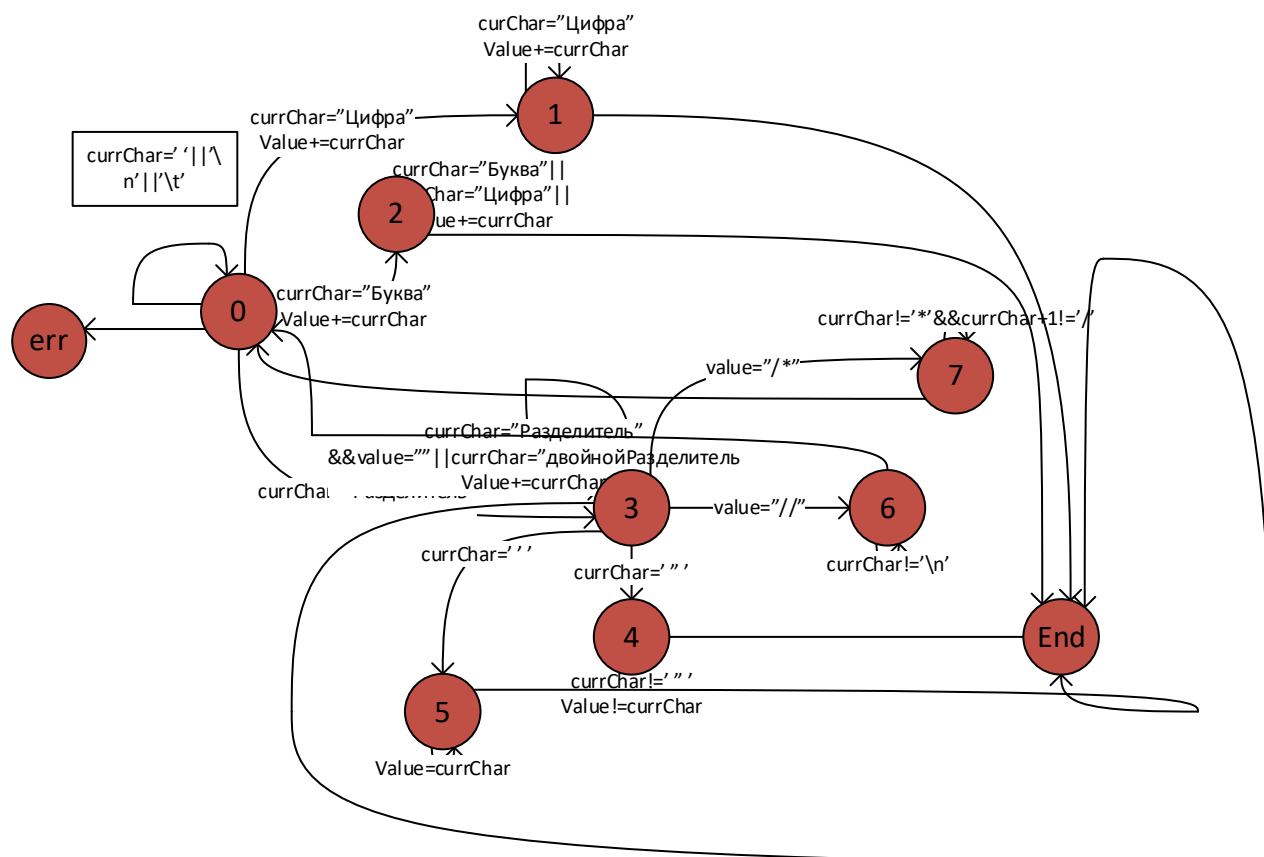


Рисунок 2.1 – Логическая структура лексического анализатора

По диаграмме состояний можно выделить следующие цепочки распознавания лексем, рассмотрим их более подробно:

Цепочка 0, 1, end – распознает числовые константы;

Цепочка 0, 2, end – распознает идентификаторы и ключевые слова;

Цепочка 0, 3, 4, end – распознает строковые литералы;

Цепочка 0, 3, end – распознает разделители;

Цепочка 0, 3, 6, 0 – распознает однострочные комментарии;

Цепочка 0, 3, 7, 0 – распознает многострочные комментарии.

Таким образом, алгоритм работы простейшего сканера можно описать следующим образом:

- просматривается входной поток символов программы на исходном языке до обнаружения очередного символа, ограничивающего лексему;
- для выбранной части входного потока выполняется функция распознавания лексемы;
- при успешном распознавании информация о выделенной лексеме заносится в таблицу лексем, и алгоритм возвращается к начальному этапу;
- при неуспешном распознавании выдается сообщение об ошибке и его выполнение прекращается.

Работа сканера продолжается до тех пор, пока не будут просмотрены все символы программы на исходном языке из входного потока.

2.2 Таблицы идентификаторов

Проверка правильности семантики и генерация кода требуют знания характеристик идентификаторов, используемых в программе на исходном языке. Эти характеристики выясняются из описаний и из того, как идентификаторы используются в программе и накапливаются в таблице символов или таблице идентификаторов. Любая таблица символов состоит из набора полей, количество которых равно числу идентификаторов программы. Каждое поле содержит в себе полную информацию о данном элементе таблицы. Под идентификаторами подразумеваются константы, переменные, имена процедур и функций, формальные и фактические параметры.

Ключевые слова «Таблица 2.1» являются предварительно определенными зарезервированными идентификаторами, которые имеют специальные значения для компилятора. Их нельзя использовать как идентификаторы в любой части программы.

Таблица 2.1 – Таблица ключевых слов

Номер	Лексема
0	and
1	array
2	begin
3	boolean
4	char
5	const
6	do
7	else
8	end
9	false
10	if
11	integer
12	let
13	not
14	of
15	or
16	program
17	read
18	sizeof
19	then
20	true
21	var
22	while
23	write

Лексический анализатор также распознает однолитерные и двухлитерные разделители, таблица 2.2 и таблица 2.3 соответственно. В случае если анализируемая литера не принадлежит ни одной из таблиц литер, с помощью которых записывается программа на входном языке, то вызывается ошибка.

Таблица 2.2 – Таблица однолитерных разделителей

Номер	Лексема
0	#
1	(
2)
3	+
4	-
5	*
6	/
7	:
8	;
9	=
10	<
11	>
12	'
13	.
14	,
15	[
16]
17	"
18	'
19	!
20	&
21	
22	\

Таблица 2.3 – Таблица двухлитерных разделителей

Номер	Лексема
0	>=
1	<=
2	==
3	!=
4	++
5	--
6	//
7	/*
8	*/
9	&&
10	

В компиляторе для каждого токена строятся таблицы «Фрагмент 2.2». Ключевые слова и разделители строятся до вызова процедуры лексического анализа и являются постоянными глобальными переменными, остальные же строятся в процессе «Фрагмент 2.3». Каждая лексема вносится в таблицу только один раз. Каждый токен должен иметь атрибут – номер строки, где хранится символ в таблице.

```
vector<string> keyWordsTable; // Таблица ключевых слов
```

```
vector<char> singleDelimetrTable; // Таблица однолитерных разделителей
```

```
vector<string> doubleDelimetrTable; // Таблица двухлитерных разделителей
```

Фрагмент 2.2 – Постоянные таблицы ключевых слов и токенов

```
vector<Id> identifierTable; // Таблица идентификаторов
```

```
vector<string> constTable; // Таблица чисел
```

```
vector<string> literalTable; // Таблица литералов
```

```
vector<char> charTable; // Таблица символов
```

Фрагмент 2.3 – Динамические таблицы идентификаторов, чисел и литералов

Программа, содержащая исходный код, написана в текстовом формате. Файл считывается в строку, с которой уже и работает программа.

2.3 Спецификации функций лексического анализатора

В лексическом анализаторе используются следующие функции:

Token Scan(bool step = false) – функция лексического анализатора, возвращает токен и может перемещаться на следующую позицию.

int IdFind(vector<Id> storage, string name) – функция ищет идентификатор внутри хранилища идентификаторов и возвращает его позицию.

Лексический анализатор необязательный этап компиляции, но желательный по следующим причинам:

- 1 Замена идентификаторов, констант, ограничителей и ключевых слов лексемами делает программу более удобной для дальнейшей обработки.

- 2 Лексический анализатор уменьшает длину программы, устраняя из ее исходного представления несущественный пробелы и комментарии.
- 3 Если будет изменена кодировка в исходном представлении программы, то это отразится только на лексическом анализаторе.

3 Разработка синтаксического анализатора

3.1 Синтаксический анализатор

Задача синтаксического анализатора – провести разбор текста программы, сопоставив его с эталоном разработанной грамматики. Для синтаксического разбора используются контекстно-свободные грамматики.

Один из эффективных методов синтаксического анализа – метод рекурсивного спуска. В основе метода рекурсивного спуска лежит левосторонний разбор строки языка. Исходной сентенциальной формой является начальный символ грамматики, а целевой – заданная строка языка. На каждом шаге разбора правило грамматики применяется к самому левому нетерминалу сентенции. Данный процесс соответствует построению дерева разбора цепочки сверху вниз (от корня к листьям).

Синтаксический анализатор проверяет следующую грамматику:

$\langle \text{программа} \rangle \rightarrow \text{'programm' } I \text{' ;' } \langle \text{блок объявлений} \rangle \langle \text{блок операторов} \rangle \text{' end' ' . '}$
 $\langle \text{блок объявлений} \rangle \rightarrow \text{' var' } \{ \langle \text{тип} \rangle I \text{' (' C ')' } | \varepsilon \} \{ \text{' , ' } I \text{' (' C ')' } | \varepsilon \} \text{' ; ' }$
 $\langle \text{тип} \rangle \rightarrow \text{' boolean' } | \text{' char' } | \text{' integer'}$
 $\langle \text{блок операторов} \rangle \rightarrow \text{' begin' } \langle \text{оператор} \rangle \text{' (; ' } | \varepsilon \text{' end'}$
 $\langle \text{оператор} \rangle \rightarrow \langle \text{блок операторов} \rangle | \langle \text{присваивание} \rangle | \langle \text{условие} \rangle | \langle \text{цикл} \rangle | \langle \text{ввод} \rangle | \langle \text{вывод} \rangle$
 $\langle \text{присваивание} \rangle \rightarrow \text{' (let ' } | \varepsilon \text{') I (' E ')' } | \text{' = ' E } | \text{' (' E ')' } | \varepsilon$
 $\langle \text{условие} \rangle \rightarrow \text{' if ' E } | \text{' then ' } \langle \text{оператор} \rangle \text{' (else ' } \langle \text{оператор} \rangle \text{' | } \varepsilon$
 $\langle \text{цикл} \rangle \rightarrow \text{' while ' E } | \text{' do ' } \langle \text{оператор} \rangle$
 $\langle \text{ввод} \rangle \rightarrow \text{' read ' (' I (' E ')' } | \varepsilon \{ \text{' , ' } I \text{' (' E ')' } | \varepsilon \} \text{')'}$
 $\langle \text{вывод} \rangle \rightarrow \text{' write ' (' E } | \langle \text{литерал} \rangle \{ \text{' , ' } E | \langle \text{литерал} \rangle \} \text{')'}$
 $\langle \text{sizeof} \rangle \rightarrow \text{' sizeof ' (' I } \langle \text{тип} \rangle \text{')'}$

E – арифметическое выражение

$E \rightarrow TE'$

$E' \rightarrow (+T | -T)(E' | \varepsilon)$

$T \rightarrow FT'$

$$T' \rightarrow (*F \mid /F)(T' \mid \varepsilon)$$
$$F \rightarrow I (\text{'[' E ']' } \mid \varepsilon) \mid \text{'(' E ')'} \mid C \mid \text{'<sizeof>'}$$

El – логическое выражение

$$El \rightarrow Tl \{ \text{'||' } Tl \}$$
$$Tl \rightarrow Fl \{ \text{'\&\&' } Fl \}$$
$$Fl \rightarrow I (\text{'[' E ']' } \mid \varepsilon) \mid \text{'(' El ')'} \mid [E \text{ Zn } E] \mid \text{'true' } \mid \text{'false' } \mid \text{'not' } Fl$$
$$Zn \rightarrow \text{'>' } \mid \text{'<' } \mid \text{'==' } \mid \text{'!=' } \mid \text{'<=' } \mid \text{'>=' }$$

3.2 Спецификации функций синтаксического анализатора

В синтаксическом анализаторе реализованы следующие функции соответствующие разработанной грамматике:

void CreateLabel ()– функция для генерации уникальных меток.

void Label(int n) – функция для генерации метки с заданным номером

void NextToken (bool isStep = false)– функция сканера, false – заглядывает в поток лексем, но не извлекает их, true –извлекает лексему.

void error(int n, string text)– функция обработки ошибок.

void Program () – функция рекурсивного спуска для <программа>.

void Var() – функция рекурсивного спуска для <блок объявлений>.

void Type()– функция рекурсивного спуска для <тип>.

void BDO()– функция рекурсивного спуска для <оператор>.

void Operators()– функция рекурсивного спуска для <блок операторов>.

void IF() – функция рекурсивного спуска для <условие>.

void WHILE() – функция рекурсивного спуска для <цикл>.

void LET() – функция рекурсивного спуска для <присваивание>.

void READ() – функция рекурсивного спуска для <ввод>.

void ReadBool() – функция ввода логического значения.

void ReadChar() – функция ввода символа.

void ReadInt() – функция ввода целочисленного числа.

void WRITE()– функция рекурсивного спуска для <вывод>.

void SIZEOF() – функция для <sizeof>

void WriteInt()– функция вывода целочисленного числа.

void WriteLiteral()– функция вывода текстовой строки.

void WriteChar()– функция вывода символа.

void WriteBool()– функция вывода логического значения.

void E(); void E_A(); void T();void T_A(); void F() – функции для разбора арифметического выражения.

void El(); void Tl(); void Fl(); void Z() – функции для разбора логического выражения.

void Code(string s) – функция записывающая ассемблерный код программы.

void Memory(string s) – функция записывающая область данных для программы.

3.3 Генерация кода

Генерация кода – это перевод транслятором внутреннего представления исходной программы в цепочку символов выходного языка. Поскольку выходным языком компилятора (в отличие от транслятора) может быть только язык ассемблера или машинный язык, то генерация кода порождает результирующую объектную программу на языке ассемблера или непосредственно на машинном языке (в машинных кодах). Эта результирующая программа всегда представляет собой линейную последовательность команд. Поэтому генерация объектного кода в любом случае должна выполнять действия, связанные с преобразованием сложных синтаксических структур в линейные цепочки. Обычно генерация объектного кода выполняется после того как выполнен синтаксический анализ программы и все необходимые действия по подготовке к генерации кода: распределено адресное пространство под переменные и подпрограммы, проверено соответствие имён и типов переменных констант и функций в синтаксических конструкциях исходной программы и т.д.

3.4 Обработка ошибок синтаксического анализа

В результате ввода кода программы, который не будет соответствовать грамматике, программа должна выводить сообщение об ошибке и завершаться. В таблице 3.1 приведена классификация событий/ошибок, которая соответствует функции `error(int errorCode, string oper)`.

Таблица 3.1 – Коды ошибок

№ ошибки	Описание
1	2
1	ожидание programm
2	ожидание имени программы
3	ожидание ;
4	ожидание .
5	ожидание типа
6	ожидание var
7	ожидание имени идентификатора
8	ожидание размера массива
9	ожидание]
10	ожидание begin
11	ожидание ; или end
12	ожидание then
13	ожидание do
14	ожидание (
15	Встречено [но идентификатор не массив
16	Идентификатор массив, но [не встречено
17	ожидание ,
18	ожидание =
19	ожидание ‘
20	ожидание символа
21	ожидание + или -
22	ожидание *,/ или &
23	ожидание)
24	ожидание числа
25	ожидание
26	ожидание &&
27	ожидание логического, но встречена константа
28	ожидание оператора сравнения
29	идентификатор не объявлен
30	Переобъявление идентификатора
99	Ожидался тип или идентификатор

3.5 Вызов и загрузка

Компилятор запускается с помощью программного обеспечения Visual Studio.

При запуске программы создается файл с расширением *.exe, который является исполняемым файлом. Далее происходит последующий запуск данного файла. Появляется консоль, содержащее текстовое сообщение о выполнении программы.

Программа преобразует входные данные в выходные.

3.6 Входные и выходные данные

Входными данными является текстовый документ, содержащий программу на языке, соответствующем заданной грамматике. Данный код программы должен храниться в папке с программой компилятора и иметь следующие обозначения:

Имя – text.

Расширение – *.txt.

Выходными данными является документ с расширением *.asm, в котором исходная программа записана на ассемблере. Далее с помощью `tasm.exe` и `tlink.exe` получаем файл с расширением *.exe.

Выводимая на экран текстовая информация (результаты компиляции) информирует об успешном выполнении или об обнаруженных ошибках.

4 Тестирование программы компилятора

4.1 Контрольный пример

Ниже приведён пример программы на разрабатываемом языке программирования «Фрагмент 4.1», соответствующем заданной грамматике. Описаны основные операторы присвоения, ввода, вывода, цикла, условный и операция sizeof.

```
program test;
var

integer a[4],b;
boolean flag;
char symbol;

begin
symbol='\n';
flag = true;
while flag do
begin

write("Enter size of array a[4] ",symbol);
read(b);

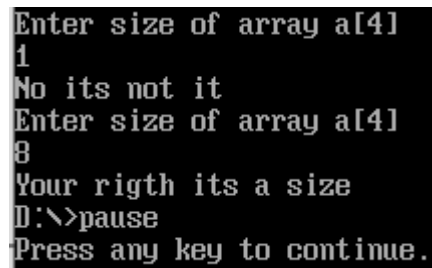
if [b==sizeof(a)] then
begin
write("Your rigth its a size",symbol);
flag = false;
end
else
write("No its not it ",symbol);
end;

end.
```

Фрагмент 4.1 – Программа на разрабатываемом языке

4.2 Результат работы программы

Результаты генерации компилятора представлены в приложение 2. На рисунке 4.1 представлен результат выполнения программы на языке ассемблера.



```
Enter size of array a[4]
1
No its not it
Enter size of array a[4]
8
Your rigth its a size
D:\>pause
Press any key to continue.
```

Рисунок 4.1 – Скриншот выполнения программы

Как можно видеть из рисунка 4.1, программа на языке ассемблера выполнена верно, без ошибок и полностью соответствует входной программе написанной на разработанном языке.

Заключение

В результате выполнения курсовой работы для заданного входного языка была разработана грамматика языка программирования, включающая следующие возможности:

- использование типов данных (целый, символьный, логический, одномерный массив; литералы);
- операторы (присваивания, цикла, выбора, ввода, вывода);
- возможность использования комментариев.

Разработан компилятор для указанного выше языка включающий следующие характеристики:

- лексический анализ;
- синтаксический анализ;
- генератор кода;
- организацию таблиц;
- реализацию индивидуального задания.

Отдельные части компилятора, разработанные в данной курсовой работе, дают представление о технике и методах, лежащих в основе построения компиляторов.

Список использованных источников

- 1 Семакин И.Г. Основы программирования: Учебник-Мастерство, 2012г. – 432 с.
- 2 Зубков С. В. Assembler для DOS, Windows и UNIX. 3-е изд., стер. –М. : ДМК Пресс ; СПб. : Питер, 2004. – 608 с.
- 3 Юров В. Assembler: специальный справочник– СПб.: Питер, 2000. – 496 с.
- 4 Ахо А., Сети Р., Ульман, Джеффри Д. Компиляторы: принципы, технологии и инструменты.: Пер. с англ – М.: “Вильямс”, 2003. – 768 с.
- 5 Огнева М. В. Программирование на языке с++: практический курс : учебное пособие для бакалавриата и специалитета / М. В. Огнева, Е. В. Кудрина. – М. : Издательство Юрайт, 2019. – 335 с.
- 6 Бондарев В.М. Программирование на С++. 2-е изд. –Харьков: “Компания СМИТ”,2005. – 284 с.

Приложение 1

Листинг компилятора

```
#include<fstream>
#include <iostream>
#include<string>
#include<algorithm>
#include<vector>
#include<iterator>
using namespace std;
string asmCode;
struct Token
{
    char table;
    int num;
};
template<typename T>
bool isFind(vector<T> storage, T value)
{
    if (find(storage.begin(), storage.end(), value) == storage.end())
        return false;
    else
        return true;
}
enum States { EMPTY, NUM, WORD, SEPARATOR, LITERAL, CHAR,
STRINGCOMMENT, BIGCOMMENT, END };
void Memmory(string s)
{
    asmCode += s + '\n';
}
struct Id {
    string name;
    int type;
    bool isArray;
    int size;
};

class Parcer {
private:
    vector<char> numbers;
    vector<char> alphabet;
    string programmCode;
    int currentPosition;
```

public:

```
vector<string> keyWordsTable;
vector<string> constTable;
vector<Id> identifierTable;
vector<char> singleSEPARATORTable;
vector<string> doubleSEPARATORTable;
vector<string> literalTable;
vector<char> charTable;
int currenStr;
Parcer(string filename)
{
    currenStr = 1;
    ifstream file;
    file.open(filename);
    while (!file.eof())
    {
        programmCode += file.get();

    }
    file.close();

    file.open("keyWords.txt");
    while (!file.eof()) {
        string word;
        file >> word;
        keyWordsTable.push_back(word);
    }
    file.close();

    file.open("Single.txt");
    while (!file.eof()) {
        char word;
        file >> word;
        singleSEPARATORTable.push_back(word);
    }
    file.close();

    file.open("Double.txt");
    while (!file.eof()) {
        string word;
        file >> word;
        doubleSEPARATORTable.push_back(word);
    }
}
```

```

    }
    file.close();

    for (int i = '0'; i <= '9'; i++)
    {
        numbers.push_back(i);
    }

    for (int i = 'a'; i <= 'z'; i++)
    {
        alphabet.push_back(i);
    }

    for (int i = 'A'; i <= 'Z'; i++)
    {
        alphabet.push_back(i);
    }

    currentPosition = 0;
}

Token NextToken(bool step = false) {
    Token result;
    string value = "";
    int state = 0;
    unsigned int imagePos = currentPosition;
    bool screen = false;
    while (currentPosition < programmCode.length() - 1)
    {
        char currChar = programmCode[imagePos];
        switch (state)
        {
            case EMPTY:

                if (currChar == ' ' || currChar == '\n' || currChar == '\t')
                {
                    imagePos++;
                    if (currChar == '\n' && step == true)currentStr++;
                    value = "";
                }
                else if (isFind(numbers, currChar))
                {
                    state = NUM;

```



```

    }
    else if (isFind(alphabet, currChar))
    {
        state = WORD;
    }
    else if (isFind(singleSEPARATORTable, currChar))
    {
        state = SEPARATOR;
    }
    else
    {
        cout << "Error undefined symbol in " << currenStr <<
" line" << endl;

        system("pause");
        exit(0);
    }
    break;
case NUM:
    if (isFind(numbers, currChar))
    {
        value += currChar;
        imagePos++;
    }
    else
    {
        result.table = 'C';
        auto i = find(constTable.begin(), constTable.end(),
value);

        if (i == constTable.end())
        {
            constTable.push_back(value);
            result.num = constTable.size() - 1;
        }
        else
        {
            result.num = distance(constTable.begin(), i);
        }
        state = END;
    }
    break;
case WORD:
    if (isFind(alphabet, currChar) || isFind(numbers, currChar))
    {
        value += currChar;
        imagePos++;
    }

```

```

        }
        else
        {
            if (isFind(keyWordsTable, value))
            {
                result.num = distance(keyWordsTable.begin(),
find(keyWordsTable.begin(), keyWordsTable.end(), value));
                result.table = 'K';
            }
            else
            {
                result.table = 'I';
                int i = IdFind(identiferTable, value);
                if (i == identiferTable.size())
                {
                    Id buff;
                    buff.name = value;
                    buff.isArray = 0;
                    buff.size = 0;
                    buff.type = 0;

                    identiferTable.push_back(buff);

                    result.num = identiferTable.size() - 1;
                }
                else
                {
                    result.num = i;
                }
            }
            state = END;
        }
        break;
    case SEPARATOR:
        if (isFind(singleSEPARATORTable, currChar) && value ==
"" || isFind(doubleSEPARATORTable, value + currChar))
        {
            if (currChar == "")
            {
                state = LITERAL;
                imagePos++;
            }
            else if (currChar == "\"" && programmCode[imagePos +
2] == "\")
            {

```

```

        state = CHAR;
        imagePos++;
    }
    else {
        value += currChar;
        imagePos++;
        if (value == "//")
            state = STRINGCOMMENT;
        if (value == "/*")
            state = BIGCOMMENT;
    }

}
else
{
    if (value.length() == 1)
    {
        result.table = 'S';
        result.num =
distance(singleSEPARATORTable.begin(), find(singleSEPARATORTable.begin(),
singleSEPARATORTable.end(), value[0]));

    }
    else if (value.length() == 2)
    {
        result.table = 'D';
        result.num =
distance(doubleSEPARATORTable.begin(), find(doubleSEPARATORTable.begin(),
doubleSEPARATORTable.end(), value));
    }
    state = END;
}

break;
case LITERAL:

    if (!screen)
    {
        if (currChar == "'")
        {
            result.table = 'L';
            auto i = find(literalTable.begin(),
literalTable.end(), value);

            if (i == literalTable.end())
            {

```

```

        literalTable.push_back(value);
        result.num = literalTable.size() - 1;

    }
    else
    {
        result.num = distance(literalTable.begin(),
i);

    }
    state = END;
}
else if (currChar != '\\')
{
    value += currChar;
}
else
{
    screen = true;
}
imagePos++;
}
else
{

    value += currChar;
    imagePos++;
    screen = false;
}

break;
case CHAR:

    if (currChar == "\"" || screen)
    {
        result.table = 'O';

        auto i = find(charTable.begin(), charTable.end(),
value[0]);

        if (i == charTable.end())
        {
            charTable.push_back(value[0]);
            result.num = charTable.size() - 1;

```

```

        }
        else
        {
            result.num = distance(charTable.begin(), i);
        }
        screen = false;
        state = END;
    }
    else
    {
        value += currChar;
        screen = true;
    }

    imagePos++;

    break;
case STRINGCOMMENT:
    if (currChar == '\n')
        state = EMPTY;
    else
        imagePos++;
    break;
case BIGCOMMENT:
    if (currChar == '*' && programmCode[imagePos + 1] == '/')
    {
        state = EMPTY;
        imagePos++;
    }
    imagePos++;
    break;
case END:
    if (step)currentPosition = imagePos;
    return result;
    break;
}

}
result.table = 'E';
result.num = '0';
return result;
}

```

private:

```
int IdFind(vector<Id> storage, string name)
{
    for (int i = 0; i < storage.size(); i++)
    {
        if (storage[i].name == name) return i;
    }
    return storage.size();
}
```

};

class Translator {

private:

```
Parcer* lex;
int currentLabel;
ofstream code;
vector<int> NotE;
vector<int> NotT;
vector<int> NotL;
int currentType;
char t_t;
int t_n;
int LastLetter;
```

public:

```
Translator(string filename)
{
```

```
    t_t = char();
```

```
    t_n = int();
```

```
    lex = new Parcer(filename);
```

```
    NotE.assign({ 1, 2, 10, 11, 15,16, 13, 14, 7, 8 });//символы не пускающие
```

в E

```
    NotT.assign({ 1, 2, 10, 11, 15,16, 13, 14, 7, 8, 3, 4 });//символы не
```

пускающие в T

```
    NotL.assign({ 1,2,15,16,13,7,8 });
```

```
    LastLetter = -1;
```

```
    currentLabel = 0;
```

```
}
```

```
void NextToken(bool isStep = false)
```

```
{
```

```

        Token t = lex->NextToken(isStep);
        t_t = t.table;
        t_n = t.num;
    }

void Program()
{
    NextToken(true);
    if (!(t_t == 'K' && t_n == 16))
        error(1, "programm");

    NextToken(true);
    if (!(t_t == 'I'))
        error(2, "programm");

    code.open(lex->identiferTable[t_n].name + ".asm");
    Code(".model small");
    Code(".stack 100h");
    Code(".code");
    Code("@start:");
    Code("mov ax, DGROUP");
    Code("mov ds, ax");

    NextToken(true);
    if (!(t_t == 'S' && t_n == 8))
        error(3, "programm");

    Var();
    Operators();

    NextToken(true);
    if (!(t_t == 'S' && t_n == 13))
        error(4, "programm");

    Code("mov ax, 4C00h");
    Code("int 21h");
    ifstream procedure("function.txt");
    string procedurs = "";
    while (!procedure.eof())
    {
        string u;
        getline(procedure, u);
        procedurs += u + "\n";
    }
}

```

```

    }
    Code(procedurs);
    Code(".data");
    Code("@buff db 255 dup(0)");
    Code("@TRUE db \"TRUE\",10,13,$");
    Code("@FALSE db \"FALSE\",10,13,$");
    Code("@maxlen db 20");
    Code("@actual db ?");
    Code("@outbuff db 20 dup (0)");
    Code("@InputError db \"out of range\",10,13,$");
    Code("@IntError db \"awaited int\",10,13,$");
    Code("@endl db 10,13,$");
    Code(asmCode);
    Code("end @start");
}

```

```

void error(int n, string text)
{
    cout << "Error ";
    switch (n)
    {
        case 1:cout << "awaited programm";
            break;
        case 2:cout << "awaited programm name";
            break;
        case 3:cout << "awaited ";
            break;
        case 4:cout << "awaited .";
            break;
        case 5:cout << "awaited name of type";
            break;
        case 6:cout << "awaited var";
            break;
        case 7:cout << "awaited identefier name";
            break;
        case 8:cout << "awaited size of array";
            break;
        case 9:cout << "awaited ]";
            break;
        case 10:cout << "awaited begin";
            break;
        case 11:cout << "awaited ; or end";
            break;
        case 12:cout << "awaited then";
            break;
    }
}

```



```

        case 13:cout << "awaited do";
            break;
        case 14:cout << "awaited (";
            break;
        case 15:cout << "met [ but identifier isn't Array";
            break;
        case 16:cout << "identifer is Array, but [ isn't met";
            break;
        case 17:cout << "awaited ,";
            break;
        case 18:cout << "awaited =";
            break;
        case 19:cout << "awaited \";
            break;
        case 20:cout << "awaited char";
            break;
        case 21:cout << "awaited + or -";
            break;
        case 22:cout << "awaited *,/ or &";
            break;
        case 23:cout << "awaited )";
            break;
        case 24:cout << "awaited number";
            break;
        case 25:cout << "awaited ||";
            break;
        case 26:cout << "awaited &&";
            break;
        case 27:cout << "awaited logical, but met const number";
            break;
        case 28:cout << "awaited comparison symbol ";
            break;
        case 29:cout << "identifier isn't defenition";
            break;
        case 30:cout << "redefinition identifier";
            break;
    }
    cout << " in block " << text << " in " << lex->currenStr << " line" << endl;
    system("pause");
    exit(0);
}

void Type() {
    NextToken(true);
}

```

```

        if (t_t == 'K' && (t_n == 3 || t_n == 4 || t_n == 11))
        {
            currentType = t_n;
        }
        else
        {
            error(5, "type");
        }
    }

void Var() {

    int currentIdentifer = 0;
    int varListBegin = 0;
    int varListEnd = 0;
    string arraySize;
    NextToken(true);
    if (!(t_t == 'K' && t_n == 21))//var
        error(6, "Var");
    NextToken();
    while (!(t_t == 'K' && t_n == 2))//begin
    {
        Type();
        NextToken(true);
        if (t_t != 'T')
        {
            error(7, "Var");
        }
        currentIdentifer++;
        if (t_n != (currentIdentifer))
        {
            error(30, "Var");
        }
        varListBegin = t_n;
        varListEnd = t_n;
        string variabl = "";
        NextToken();
        variabl += lex->identiferTable[currentIdentifer].name;
        variabl += " ";
        variabl += currentType == 11 ? "dw" : "db";

        if (t_t == 'S' && t_n == 15)//[
        {
            NextToken(true);//[

```

```

        NextToken(true);
        if (t_t != 'C')
        {
            error(8, "Var");
        }
        arraySize = lex->constTable[t_n];
        NextToken(true);
        if (!(t_t == 'S' && t_n == 16))
        {
            error(9, "Var");
        }
        lex->identiferTable[currentIdentifer].isArray = true;
        lex->identiferTable[currentIdentifer].size = stoi(arraySize);
        lex->identiferTable[currentIdentifer].type = currentType;

        variabl += " ";
        variabl += arraySize;
        variabl += " dup(";
        variabl += "0";
        variabl += ")";
    }
    else
    {
        lex->identiferTable[currentIdentifer].isArray = false;
        lex->identiferTable[currentIdentifer].type = currentType;
        variabl += " ";
        variabl += "0";
    }
    Memmory(variabl);
    NextToken();
    while (!(t_t == 'S' && t_n == 8))// ;
    {
        variabl = "";
        NextToken(true);// ,
        NextToken(true);
        if (t_t != 'I')
        {
            error(7, "Var");
        }

        currentIdentifer++;
        if (t_n != (currentIdentifer))
        {
            error(30, "Var");
        }
    }

```

```

    }
    variabl += lex->identiferTable[currentIdentifer].name;
    variabl += " ";
    variabl += currentType == 11 ? "dw" : "db";
    varListEnd = t_n;
    NextToken();
    if (t_t == 'S' && t_n == 15)//[
    {
        NextToken(true);
        NextToken(true);
        if (t_t != 'C')
        {
            error(8, "Var");
        }
        arraySize = lex->constTable[t_n];
        NextToken(true);
        if (!(t_t == 'S' && t_n == 16))//]
        {
            error(9, "Var");
        }
        lex->identiferTable[currentIdentifer].isArray = true;
        lex->identiferTable[currentIdentifer].size =

stoi(arraySize);

        lex->identiferTable[currentIdentifer].type =

        currentType;

        variabl += " ";
        variabl += arraySize;
        variabl += " dup(";
        variabl += "0";
        variabl += ")";

    }
    else {
        lex->identiferTable[currentIdentifer].isArray = false;
        lex->identiferTable[currentIdentifer].type =

        currentType;

        variabl += " ";
        variabl += "0";

    }

    NextToken();

```

```

        Memmory(variabl);

    }
    NextToken();
    if (!(t_t == 'K' && t_n == 2))//begin
    {
        NextToken(true);
        if (!(t_t == 'S' && t_n == 8)) //;
            error(3, "Var");
    }
    NextToken();

}

}

void BLO()
{
    NextToken();
    if (t_t == 'K' && t_n == 2)//begin
    {
        Operators();
    }
    else if (t_t == 'K' && t_n == 10)//if
    {
        IF();
    }
    else if (t_t == 'K' && t_n == 22)//while
    {
        WHILE();
    }
    else if (t_t == 'K' && t_n == 17)//read
    {
        READ();
    }
    else if (t_t == 'K' && t_n == 12 || t_t == 'T')
    {
        LET();
    }
    else if (t_t == 'K' && t_n == 23)
    {
        WRITE();
    }
}

```

```

else if (t_t == 'K' && t_n == 18)
{
    SIZEOF();
    NextToken(true);
    if (!(t_t == 'S' && t_n == 8))
    {
        error(3, "SIZEOF");
    }
}

}

void Operators() {
    NextToken(true);
    if (!(t_t == 'K' && t_n == 2))
    {
        error(10, "Operators");
    }
    NextToken(false);
    while (!(t_t == 'K' && t_n == 8))
    {
        BLO();
        NextToken();
        if (t_t == 'S' && t_n == 8)
        {
            NextToken(true);
        }
        else if (!(t_t == 'K' && t_n == 8))
        {
            error(11, "operators");
        }
    }
    NextToken(true);
}

void IF()
{
    int els, end;
    NextToken(true);
    EL();
    Code("cmp ax,0");
}

```

```

        Code("jnz @label" + to_string(currentLabel));
        Code("jmp @label" + to_string(currentLabel + 1));
        NextToken(true);
        CreateLable();
        els = currentLabel++;
        end = currentLabel++;
        if (!(t_t == 'K' && t_n == 19))
        {
            error(12, "If");

        }
        BLO();
        Code("jmp @label" + to_string(end));
        NextToken();
        Label(els);
        if (t_t == 'K' && t_n == 7)
        {
            NextToken(true);
            BLO();
            Code("jmp @label" + to_string(end));
        }
        Label(end);
    }

void WHILE() {
    NextToken(true);
    int end, loop;
    loop = currentLabel;
    CreateLable();
    EL();
    Code("cmp ax, 0");
    Code("jnz @label" + to_string(currentLabel));
    Code("jmp @label" + to_string(currentLabel + 1));
    CreateLable();
    end = currentLabel++;
    NextToken(true);
    if (!(t_t == 'K' && t_n == 6))
    {
        error(13, "WHILE");
    }
    BLO();
    Code("jmp @label" + to_string(loop));
    Label(end);
}

```

```

void READ() {
    int IdNum;
    NextToken(true);
    NextToken(true);
    if (!(t_t == 'S' && t_n == 1))
    {
        error(14, "READ");
    }
    NextToken();
    if (t_t == 'S' && t_n == 2)
    {
        NextToken(true);
    }
    else
    {
        if (t_t != 'I')
        {
            error(7, "READ");
        }
        NextToken(true);
        IdNum = t_n;
        NextToken();
        if (lex->identiferTable[IdNum].type == NULL)
        {
            error(29, "READ");
        }
        if (!lex->identiferTable[IdNum].isArray && t_t == 'S' && t_n ==
15)
        {
            error(15, "READ");
        }
        if (lex->identiferTable[IdNum].isArray && !(t_t == 'S' && t_n ==
15))
        {
            error(16, "READ");
        }
        if (t_t == 'S' && t_n == 15)//[
        {
            NextToken(true);
            E();
            NextToken(true);
            if (!(t_t == 'S' && t_n == 16))
            {
                error(9, "READ");
            }
        }
    }
}

```



```

    }
    else {
        Code("push 0");
    }

    if (lex->identiferTable[IdNum].type == 11)
    {
        Code("mov @maxlen,6");
    }
    else
    {
        Code("mov @maxlen,2");
    }

    Code("call @read");
    switch (lex->identiferTable[IdNum].type)
    {
    case 3:ReadBool();
        break;
    case 4:ReadChar();
        break;
    case 11:ReadInt();
        break;
    default:
        break;
    }
    Code("pop ax");
    Code("pop bx");
    if (lex->identiferTable[IdNum].type == 11)
    {
        Code("sal bx,1");
        Code("mov " + lex->identiferTable[IdNum].name +
"[bx],ax");
    }
    else
        Code("mov " + lex->identiferTable[IdNum].name + "[bx],al");
    NextToken();
    while (!(t_t == 'S' && t_n == 2))
    {
        NextToken(true);
        if (!(t_t == 'S' && t_n == 14))//,
        {
            error(17, "READ");
        }
    }

```

```

        NextToken();
        if (t_t != 'I')
        {
            error(7, "READ");
        }
        NextToken(true);
        IdNum = t_n;
        NextToken();
        if (!lex->identiferTable[IdNum].isArray && t_t == 'S' && t_n
== 15)
        {
            error(15, "LET");
        }
        if (lex->identiferTable[IdNum].isArray && !(t_t == 'S' &&
t_n == 15))
        {
            error(16, "LET");
        }
        if (t_t == 'S' && t_n == 15)//[
        {
            NextToken(true);
            E();
            NextToken(true);
            if (!(t_t == 'S' && t_n == 16))
            {
                error(9, "READ");
            }
        }

        if (lex->identiferTable[IdNum].type == 11)
        {
            Code("mov @maxlen,6");
        }
        else
        {
            Code("mov @maxlen,2");
        }

        Code("call @read");
        switch (lex->identiferTable[IdNum].type)
        {
        case 3:ReadBool();
            break;
        case 4:ReadChar();

```

```

        break;
    case 11:
        ReadInt();
        break;
    default:
        break;
    }
    Code("pop ax");
    Code("pop bx");
    if (lex->identiferTable[IdNum].type == 11)
    {
        Code("sal bx,1");
        Code("mov " + lex->identiferTable[IdNum].name +
"[bx],ax");
    }
    else
        Code("mov " + lex->identiferTable[IdNum].name +
"[bx],al");
    NextToken();
}

}
NextToken(true);
}

void SIZEOF()
{
    NextToken(true);
    NextToken(true);
    if (!(t_t == 'S' && t_n == 1))
    {
        error(14, "SIZEOF");
    }
    NextToken(true);
    if (t_t == 'I')
    {
        Id id = lex->identiferTable[t_n];
        switch (id.type)
        {
            case 3:
            {
                if (id.isArray)
                {
                    Code("push " + to_string(1 * id.size));
                }
            }
        }
    }
}

```

```

        else
        {
            Code("push 1");
        }
    }
    break;
case 4:
    if (id.isArray)
    {
        Code("push " + to_string(1 * id.size));
    }
    else
    {
        Code("push 1");
    }
    break;
case 11:
    if (id.isArray)
    {
        Code("push " + to_string(2 * id.size));
    }
    else
    {
        Code("push 2");
    }
    break;
default:
    break;
}

}
else if (t_t == 'K')
{
    switch (t_n)
    {
    case 3:
        Code("push 1");
        break;
    case 4:
        Code("push 1");
        break;
    case 11:
        Code("push 2");
        break;
    default:

```

```

        break;
    }
}
else {
    error(99, "SIZEOF");
}
NextToken(true);
if (!(t_t == 'S' && t_n == 2))
{
    error(23, "SIZEOF");
}

}

void WRITE()
{
    int IdNum;
    NextToken(true);
    NextToken(true);
    if (!(t_t == 'S' && t_n == 1))
    {
        error(14, "write");
    }
    NextToken();
    if (t_t == 'S' && t_n == 2)
    {
        NextToken(true);
    }
    else
    {
        if (t_t == 'L') {
            if (t_n == lex->literalTable.size() - 1 && t_n != LastLetter)
                Memmory("@L" + to_string(t_n) + " db \"" + lex-
>literalTable[t_n] + "\", '$");
            LastLetter = t_n;
            WriteLiteral();
            NextToken(true);
        }
        else
        {
            if (t_t == 'I')
            {
                Code("xor ax,ax");
                IdNum = t_n;
            }
        }
    }
}

```

```

        if (lex->identiferTable[IdNum].type == NULL)
        {
            error(29, "WRITE");
        }
        switch (lex->identiferTable[IdNum].type)
        {
        case 3:
            EL();
            WriteBool();
            break;
        case 4:
            E();
            WriteChar();
            break;
        case 11:
            E();
            WriteInt();
            break;
        default:
            break;
        }
    }
    else if (t_t == 'C')
    {
        E();
        WriteInt();
    }
}

}
NextToken();
while (!(t_t == 'S' && t_n == 2))
{
    NextToken(true);
    if (!(t_t == 'S' && t_n == 14))
    {
        error(17, "Write");
    }
    NextToken();
    if (t_t == 'L') {
        if (t_n == lex->literalTable.size() - 1 && t_n != LastLetter)
            Memmory("@L" + to_string(t_n) + " db \"" + lex-
>literalTable[t_n] + "\", '$");
        LastLetter = t_n;
        WriteLiteral();
    }
}

```

```

        NextToken(true);
    }
    else
    {
        if (t_t == 'I')
        {
            Code("xor ax,ax");
            IdNum = t_n;
            switch (lex->identiferTable[IdNum].type)
            {
                case 3:
                    EL();
                    WriteBool();
                    break;
                case 4:
                    E();
                    WriteChar();
                    break;
                case 11:
                    E();
                    WriteInt();
                    break;
                default:
                    break;
            }
        }
        else if (t_t == 'C')
        {
            E();
            WriteInt();
        }
    }
    NextToken();
}
NextToken(true);
}

void LET()
{
    NextToken();
    if (t_t == 'K' && t_n == 12)
    {
        NextToken(true);
    }
    NextToken(true);
}

```

```

if (t_t != 'T')
{
    error(7, "LET");
}

int IdNum = t_n;

Code("xor bx,bx");
if (lex->identiferTable[IdNum].type == NULL)
{
    error(29, "LET");
}
NextToken();
if (!lex->identiferTable[IdNum].isArray && t_t == 'S' && t_n == 15)
{
    error(15, "LET");
}
if (lex->identiferTable[IdNum].isArray && !(t_t == 'S' && t_n == 15))
{
    error(16, "LET");
}
if (t_t == 'S' && t_n == 15)
{
    NextToken(true);
    E();

    Code("pop bx");
    if (lex->identiferTable[IdNum].type == 11)
    {
        Code("sal bx,1");
    }

    NextToken(true);
    if (!(t_t == 'S' && t_n == 16))
    {
        error(9, "LET");
    }
}

Code("push bx");
NextToken(true);
if (!(t_t == 'S' && t_n == 9))//=
{
    error(18, "LET");
}

```



```

    }
    switch (lex->identiferTable[IdNum].type)
    {
    case 3:
        EL();
        Code("pop ax");
        break;
    case 4:
    {

        NextToken(true);
        {
            if ((t_t == 'O'))
            {
                Code(("push " + to_string(int(lex->charTable[t_n]))));
            }
            else if ((t_t == 'S' && t_n == 18))
            {
                NextToken(true);
                if (t_t == 'D' && t_n == 11)
                {
                    Code("push 10");
                }
                else
                {
                    error(20, "LET");
                }
                NextToken(true);
                if (!(t_t == 'S' && t_n == 18))
                {
                    error(19, "Let");
                }
            }
        }
        else {
            error(20, "Let");
        }
    }

    Code("pop ax");
}

```

```

        break;
    case 11:
        E();
        Code("pop ax");
        break;
    default:
        break;
    }
    Code("pop bx");
    if (lex->identiferTable[IdNum].type == 11)
        Code("mov " + lex->identiferTable[IdNum].name + "[bx],ax");
    else
        Code("mov " + lex->identiferTable[IdNum].name + "[bx],al");
}

```

void ReadBool()

```

{
    Code("xor dx,dx");
    Code("mov dl,@outbuff");
    Code("sub dx,'0'");
    Code("push dx");
}

```

void ReadChar()

```

{
    Code("xor dx,dx");
    Code("mov dl,@outbuff");
    Code("push dx");
}

```

void ReadInt()

```

{
    Code("call @string_to_int");
}

```

void WriteBool()

```

{
    Code("call @bool_to_string");
    Code("call @print");
}

```

void WriteChar()

```

{

```

```

        Code("call @print_char");
    }

void WriteInt()
{
    Code("call @int_to_string");
    Code("call @print");
}

void WriteLiteral()
{
    Code("lea dx,@L" + to_string(t_n));
    Code("push dx");
    Code("call @print");
}

void E()
{
    T();
    NextToken();
    if (!(t_t == 'D' || t_t == 'K' || t_t == 'T' || (t_t == 'S' && isFind(NotE, t_n))))
        E_A();
}

void E_A()
{
    NextToken(true);
    if (t_t == 'S' && t_n == 3)//+
    {
        T();
        Code("pop ax");
        Code("pop bx");
        Code("add ax,bx");
        Code("push ax");
    }
    else if (t_t == 'S' && t_n == 4)//-
    {
        T();
        Code("pop bx");
        Code("pop ax");
        Code("sub ax,bx");
        Code("push ax");
    }
    else
    {
        error(21, "E_A");
    }
}

```

```

    }
    NextToken();
    if (!(t_t == 'D' || t_t == 'K' || t_t == 'T' || (t_t == 'S' && isFind(NotE, t_n))))
        E_A();
}

void T() {
    F();
    NextToken();
    if (!(t_t == 'D' || t_t == 'K' || t_t == 'T' || (t_t == 'S' && isFind(NotT, t_n))))
        T_A();
}

void T_A()
{
    NextToken(true);
    if (t_t == 'S' && t_n == 5) // *
    {
        F();
        Code("pop ax");
        Code("pop bx");
        Code("mul bx");
        Code("push ax");
    }
    else if (t_t == 'S' && t_n == 6) // /
    {
        F();
        Code("xor dx,dx");
        Code("pop bx");
        Code("pop ax");
        Code("div bx");
        Code("push ax");
    }
    else
    {
        error(22, "T_A");
    }
    NextToken();
    if (!(t_t == 'D' || t_t == 'K' || t_t == 'T' || (t_t == 'S' && isFind(NotT, t_n))))
        T_A();
}

void F() {
    NextToken();
    if (t_t == 'K' && t_n == 18)

```

```

    {
        SIZEOF();
    }
else if (t_t == 'C')
{
    NextToken(true);
    Code("push " + lex->constTable[t_n]);
}
else if ((t_t == 'O'))
{
    NextToken(true);
    Code(("push " + to_string(int(lex->charTable[t_n]))));
}
else if ((t_t == 'S' && t_n == 18))
{
    NextToken(true);
    NextToken(true);
    if (t_t == 'D' && t_n == 11)
    {
        Code("push 10");
    }
    else
    {
        error(20, "F");
    }
    NextToken(true);
    if (!(t_t == 'S' && t_n == 18))
    {
        error(19, "F");
    }
}
else if (t_t == 'I')
{
    NextToken(true);

    int a = t_n;
    Code("xor bx,bx");
    if (lex->identiferTable[a].type == NULL)
    {
        error(29, "F");
    }

    NextToken();
}

```

```

        if (!lex->identiferTable[a].isArray && t_t == 'S' && t_n == 15)
        {
            error(15, "F");
        }
        if (lex->identiferTable[a].isArray && !(t_t == 'S' && t_n == 15))
        {
            error(16, "F");
        }

        if (t_t == 'S' && t_n == 15) //[
        {

            NextToken(true);
            E();
            Code("pop bx");
            if (lex->identiferTable[a].type == 11)
            {
                Code("sal bx,1");
            }
            NextToken(true);
            if (!(t_t == 'S' && t_n == 16))//[
                error(9, "F");
            }
            if (lex->identiferTable[a].type == 11)
                Code("mov ax," + lex->identiferTable[a].name + "[bx]");
            else
                Code("mov al," + lex->identiferTable[a].name + "[bx]");
            Code("push ax");
        }
        else if ((t_t == 'S' && t_n == 1))//(
        {
            NextToken(true);
            E();
            NextToken(true);
            if (!(t_t == 'S' && t_n == 2))//(
                error(23, "F");
            }
        else {
            error(24, "F");
        }
    }

    void EL()
    {

```

```

    TL();
    NextToken();
    while (!(t_t == 'S' && isFind(NotL, t_n) || (t_t == 'K' && (t_n == 7 || t_n
== 19 || t_n == 6))))//do else then
    {
        if (t_t == 'D' && t_n == 10)
        {
            NextToken(true);
            TL();
            Code("pop bx");
            Code("pop ax");
            Code("or ax,bx");
            Code("push ax");
        }
        else
        {
            error(25, "EL");
        }
        NextToken();
    }
}

void TL() {
    FL();
    NextToken();
    while (!(t_t == 'S' && isFind(NotL, t_n) || (t_t == 'K' && (t_n == 7 || t_n
== 19 || t_n == 6)) || (t_t == 'D' && t_n == 10))))//else then
    {
        if (t_t == 'D' && t_n == 9)
        {
            NextToken(true);
            FL();
            Code("pop bx");
            Code("pop ax");
            Code("and ax,bx");
            Code("push ax");
        }
        else {
            error(26, "TL");
        }
        NextToken();
    }
}

```

```

void FL() {

    NextToken(false);
    if (t_t == 'C')
    {
        error(27, "FL");
    }

    if (((t_t == 'K') && (t_n == 9 || t_n == 20)) || (t_t == 'I'))
    {
        NextToken(true);
        if (t_t == 'K')
        {
            if (t_n == 20)
            {
                Code("mov ax,1");
                Code("push ax");
            }
            else if (t_n == 9)
            {
                Code("xor ax,ax");
                Code("push ax");
            }
        }
        else if (t_t == 'I')//I(['E']|eps)
        {
            int a = t_n;
            Code("xor bx,bx");
            if (lex->identiferTable[a].type == NULL)
            {
                error(29, "FL");
            }
            NextToken(false);

            if (!lex->identiferTable[a].isArray && t_t == 'S' && t_n ==
15)
            {
                error(15, "FL");
            }
            if (lex->identiferTable[a].isArray && !(t_t == 'S' && t_n ==
15))
            {
                error(16, "FL");
            }
        }
    }
}

```



```

    }

    if (t_t == 'S' && t_n == 15)
    {
        NextToken(true);
        E();

        Code("pop bx");
        if (lex->identiferTable[a].type == 11)
        {
            Code("sal bx,1");
        }

        NextToken(true);
        if (!(t_t == 'S' && t_n == 16))
        {
            error(9, "FL");
        }
    }

    Code("mov al," + lex->identiferTable[a].name + "[bx]");
    Code("push ax");

}
else
{
    if (t_t == 'S' && t_n == 19)
    {
        NextToken(true);
        FL();
        Code("pop ax");
        Code("not ax");
        Code("push ax");
    }
    else
    {
        if (t_t == 'S' && t_n == 15)//[E Z E ]
        {
            NextToken(true);
            E();
            Z();
            E();
            if (!(t_t == 'S' && t_n == 16))
            {

```

```

        error(9, "FL");
    }
    Code("call @otnoshenie");
    Code("pop ax");
    NextToken(true);
}
else if (t_t == 'S' && t_n == 1) //(EL)
{
    NextToken(true);
    EL();
    NextToken(true);
    if (!(t_t == 'S' && t_n == 2))
    {
        error(24, "Fl");
    }
}
}

}

}

void Z()
{
    NextToken();
    if ((t_t == 'S' && (t_n == 10 || t_n == 11)) || (t_t == 'D' && (t_n == 0 || t_n
== 1 || t_n == 2 || t_n == 3)))
    {
        NextToken(true);
        Code("push " + to_string(t_n));
    }
    else
    {
        error(28, "Z");
    }
}

private:

void Code(string s)
{
    code << s << endl;
}

void CreateLable()

```

```

    {
        code << "@label" << currentLabel << ": " << endl;
        currentLabel++;
    }

void Label(int n)
{
    code << "@label" << n << ": " << endl;
}

};

int main()
{
    Translator a("text.txt");
    a.Program();
    cout << "File is created!\n";
    system("pause");
}

```

Приложение 2

Листинг программы на языке ассемблера

```

.model small
.stack 100h
.code
@start:
mov ax, DGROUP
mov ds, ax
xor bx,bx
push bx
push 10
pop ax

```

```
pop bx
mov symbol[bx],al
xor bx,bx
push bx
mov ax,1
push ax
pop ax
pop bx
mov flag[bx],al
@label0:
xor bx,bx
mov al,flag[bx]
push ax
cmp ax, 0
jnz @label1
jmp @label2
@label1:
lea dx,@L0
push dx
call @print
xor ax,ax
xor bx,bx
mov al,symbol[bx]
push ax
call @print_char
push 0
mov @maxlen,6
call @read
call @string_to_int
pop ax
```

```
pop bx
sal bx,1
mov b[bx],ax
xor bx,bx
mov ax,b[bx]
push ax
push 2
push 8
call @otnoshenie
pop ax
cmp ax,0
jnz @label3
jmp @label4
@label3:
lea dx,@L1
push dx
call @print
xor ax,ax
xor bx,bx
mov al,symbol[bx]
push ax
call @print_char
xor bx,bx
push bx
xor ax,ax
push ax
pop ax
pop bx
mov flag[bx],al
jmp @label5
```

```
@label4:
lea dx,@L2
push dx
call @print
xor ax,ax
xor bx,bx
mov al,symbol[bx]
push ax
call @print_char
jmp @label5
@label5:
jmp @label0
@label2:
mov ax, 4C00h
int 21h
@otnoshenie proc
pop si
pop cx
pop bx
pop ax
xor bp,bp
cmp bx,0
jz @M0
cmp bx,1
jz @M1
cmp bx,2
jz @M2
cmp bx,3
jz @M3
cmp bx,10
```

jz @M4
cmp bx,11
jz @M5

@M0:cmp ax,cx
jnge @EndOtnoshenie
not bp
jmp @EndOtnoshenie

@M1:cmp ax,cx
jnle @EndOtnoshenie
not bp
jmp @EndOtnoshenie

@M2:cmp ax,cx
jnz @EndOtnoshenie
not bp
jmp @EndOtnoshenie

@M3:cmp ax,cx
jz @EndOtnoshenie
not bp
jmp @EndOtnoshenie

@M4:cmp ax,cx
jnl @EndOtnoshenie
not bp
jmp @EndOtnoshenie

```
@M5:cmp ax,cx
jng @EndOtnoshenie
not bp
jmp @EndOtnoshenie
```

```
@EndOtnoshenie:
push bp
push si
ret
@otnoshenie endp
```

```
@read proc
@r3:pop si
pop cx
mov ah, 0AH
lea dx,@maxlen
INT 21h
```

```
push si
push cx
xor dx,dx
mov dx, offset @endl;
push dx;
call @print
pop cx
pop si
```

```
mov al,@maxlen
```



```
cmp al,2
jbe @r1;
mov bx,0

@r7:cmp bl,@actual
je @r4
cmp @outbuff[bx],'0'
jnb @r5
mov dx, offset @IntError;
push cx
push si
push dx
call @print
jmp @r3
@r5:
cmp @outbuff[bx],'9'
jna @r6
mov dx, offset @IntError;
push cx
push si
push dx
call @print
jmp @r3
@r6:
inc bx
jmp @r7

@r4:
mov al,@actual
mov ah,0
```

```
cmp ax,5
jnz @r1
cmp @outbuff[0],'6'
jb @r1
ja @r2
cmp @outbuff[1],'5'
jb @r1
ja @r2
cmp @outbuff[2],'5'
jb @r1
ja @r2
cmp @outbuff[3],'3'
jb @r1
ja @r2
cmp @outbuff[4],'5'
jbe @r1
    @r2:
    mov dx, offset @InputError;
    push cx
    push si
    push dx
    call @print
    jmp @r3
    @r1:
    mov di,ax
    mov @outbuff [di],'$'
    push cx
    push si
    ret
endp
```

```

@string_to_int proc
@sti4:
pop si
mov dx,10
xor ax,ax
mov al,@actual
mov di,ax
xor ax,ax
xor cx,cx
mov bx,1
@sti1:cmp di,0
jz @sti2
xor ax,ax
mov al,@outbuff[di-1]
cmp al,'0'
jae @sti3
push si
call @read
jmp @sti4
@sti3:
sub al,'0'
mul bx
add cx,ax
dec di
mov ax,bx
mov dx,10
mul dx
mov bx,ax
jmp @sti1

```

@sti2:

push cx

push si

ret

endp

@print proc

pop si

pop dx

mov ah,9

int 21h

push si

ret

endp

@int_to_string proc

pop si

pop ax

mov bx,10

xor cx,cx

@razbor:

inc cx

cmp ax,10

jb @endRazbor

xor dx,dx

div bx

```
push dx
cmp ax,0
jnz @razbor
@endRazbor:
push ax
```

```
xor dx,dx
lea bx,@buff
@toBuff:
cmp cx,dx
jz @endToBuff
pop ax
add ax,'0'
mov [bx],al
inc dx
inc bx
jmp @toBuff
@endToBuff:
mov [bx],36
lea dx,@buff
push dx
push si
ret
endp
```

```
@bool_to_string proc
pop si
pop ax
```

```
cmp ax,0
jnz @Mtrue
lea dx,@FALSE
jmp @EndBool
@Mtrue:
lea dx,@TRUE
@EndBool:
push dx
push si
ret
endp
```

```
@print_char proc
pop si
pop dx
mov ah,2
int 21h
push si
ret
endp
```

```
.data
@buff db 255 dup(0)
@TRUE db "TRUE",10,13,$'
@FALSE db "FALSE",10,13,$'
@maxlen db 20
@actual db ?
@outbuff db 20 dup (0)
@InputError db "out of range",10,13,$'
@IntError db "awaited int",10,13,$'
```

```
@endl db 10,13','$  
a dw 4 dup(0)  
b dw 0  
flag db 0  
symbol db 0  
@L0 db "Enter size of array a[4] ",'$'  
@L1 db "Your rigth its a size",'$'  
@L2 db "No its not it ",'$'  
  
end @start
```