**ОГЛАВЛЕНИЕ**

[ВВЕДЕНИЕ 3](#_Toc42623421)

[ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА 4](#_Toc42623422)

[Ввод 4](#_Toc42623423)

[Вывод 4](#_Toc42623424)

[Описание алгоритма 5](#_Toc42623425)

[Способы представления алгоритмов 5](#_Toc42623426)

[Обобщенный алгоритм конвертора 7](#_Toc42623427)

[Структура программы 8](#_Toc42623428)

[Описание констант 9](#_Toc42623429)

[Ввод тестов и вывод результатов 10](#_Toc42623430)

[Выводы 13](#_Toc42623431)

[Приложение 13](#_Toc42623432)

Изм.

Лист

№ докум.

Подпись

Дата

Лист

2

RU. 02068048.502900.01.01

Разраб.

Фомин А.Е.

Провер.

Монахов Ю.М.

Реценз.

Н. Контр.

Монахов Ю.М.

Утверд.

КОМПИЛЯТОР ПОДМНОЖЕСТВА ПРОЦЕДУРНОГО ЯЗЫКА В АССЕМБЛЕР

Лит.

Листов

42

ВлГУ

# ВВЕДЕНИЕ

Задачи курсовой работы заключаются:

Описание программы, предназначенной для компиляции программ на некотором исходном языке программирования под целевую платформу. Исходным языком, предназначенным для компиляции, является выбранное мной подмножество языка Pascal. Язык, выбранный для разработки C++.

Требования к приложению:

Требования к входному языку:

1. Должны присутствовать операторные скобки.

2. Должна игнорироваться индентация программы.

3. Должны поддерживаться комментарии любой длины.

4. Входная программа должна представлять собой единый модуль, но также должна быть поддержка вызова функций.

Операторы:

1. Оператор присваивания.

2. Арифметика (\*, /, +, -, >, <, =).

3. Логические операторы (И, ИЛИ, НЕ).

4. Условный оператор (ЕСЛИ).

5. Операторы цикла (while, break, continue).

6. Базовый вывод (строковый литерал, переменная).

7. Типы (целочисленный 32 бита, с плавающей запятой 32 бита).

Требования к выходному языку:

1. В ассемблере.

Интерфейс приложения:

* На вход поступает код программы на языке Pascal
* В процессе работы компилятора выполняется этот код на целевой платформе
* На выход получаем этот файл, очищенный от комментариев и пустых блоков

# ПРОЕКТИРОВАНИЕ КОМПИЛЯТОРА

# Ввод

Исходный текст программы пишется заранее с помощью текстового редактора и сохраняется в текстовом файле, имя которого вводится в ответ на запрос программы:

В качестве входного файла в данном курсовом проекте используется файл с кодом на языке Pascal: <infile.pas>.

# Вывод

Выходной текст создается конвертором и хранится в текстовом файле, имя которого вводится в ответ на запрос программы:

Выходной файл имеет имя <outfile.c>;

В консоли после ввода пути и названия файла, пользователю будет выведено одно из двух возможных сообщений:

- об успешности операции.

- сообщение об ошибке.

При успешной операции, запустится специальная подпрограмма обработки текста.

При ошибке, на экран будет выведено предложение повторного ввода названия файла.

Похожая реакция программы должна быть предусмотрена и для исходящего файла. Таким образом, при невозможности создания файла на экран поступит сообщение об ошибке. И что бы не потерять обработанную информацию, весь текст транслируемой программы должен храниться в оперативной памяти компьютера. В результате, можно предусмотреть множественный запрос имени выходного файла, без нанесения вреда данным.

# Описание алгоритма

Точного определения алгоритма не существует, так же, как не существует определения информации, множества и т.д. Однако можно дать достаточно полное представление о таком понятии как алгоритм.

Под алгоритмом понимают совокупность точных и однозначных инструкций для некоторого исполнителя данного алгоритма, предназначенных для решения какой-либо задачи (достижения какой-либо цели). При этом предполагается выполнение следующих свойств:

1. Дискретность – команды, инструкции алгоритма представляют собой разделимую последовательность действий.

2. Конечность – число шагов алгоритма должно быть конечно.

3. Определенность (однозначность, детерминированность) – каждая команда алгоритма должна быть однозначно воспринята исполнителем.

4. Массовость – алгоритм предназначен для решения множества задач заданного вида.

5. Эффективности – интерес представляют в первую очередь такие алгоритмы, которые решают поставленную задачу в пределах допустимого времени с желательно меньшим расходом ресурсов исполнителя.

# Способы представления алгоритмов

Любое представление алгоритма является некоторым информационным блоком, то есть представление алгоритма является информацией, и она расположена на некотором носителе информации.

Представление алгоритма можно понимать как отображение множества алгоритмов для фиксированного исполнителя во множество некоторых данных, например, символов или рисунков.

Существует три основных способа представления алгоритмов:

1) графический;

2) неформальная языковая (алгоритмическая) нотация (запись);

3) запись на алгоритмическом языке**.**

Любая форма записи (представления) алгоритма должна обеспечивать свойства алгоритма: дискретности, конечности, определенности, массовости.

В графической форме алгоритм представлен в виде геометрических фигур. Обычно они связываются линиями, которые показывают направление передачи информации при исполнении алгоритма. Существует несколько вариантов графического представления алгоритмов, но широкую известность получило (и стало фактическим стандартом графического представления) представление в виде блок-схем.

Алгоритм может быть представлен в виде записей литературного языка, например, русского. В этом случае последовательностью предложений описывается последовательность действий исполнителя, которым может быть в большинстве случаев только человек. Никаких специальных правил и требований к таким записям алгоритмов не предъявляется. Главное, что бы выполнялись требования, предъявляемые к алгоритмам.

Третий способ представления алгоритмов – это способ записи алгоритмов с использованием алгоритмических языков, либо языков программирования. Алгоритмический язык – это система правил и обозначений для точной и однозначной записи алгоритмов. Такая запись является формализованной. Это означает, что запись подчиняется строгим требованиям синтаксиса языка.

Алгоритм трансляции в данной программе заключается в следующем, весь входной файл представляется в виде одной символьной строки размером константы lenprog=10000. После чего вводится массив ключевых слов, которые программа и будет распознавать. Каждому из этих слов пишется подпрограмма, которая заменяет эти слова аналогичными в языке C. Помимо этого считается количество операторов в входном файле - koper1, и в выходном – koper2, и ошибочные - ker. Помимо этого программа переводит комментарии Begin, End, var, оператор присваивания.

Для реализации транслирования создаются переменные, которые помогают выполнять наиболее частые операции, типа: пропуск пробела, копирование символа, постановка или удаление отступа, добавление нового символа или целой фразы.

Программа распознает несколько типов данных, integer, Boolean и real. Что позволяет выводить в результате программы почти скомпилированный код.

# Обобщенный алгоритм конвертора

Анализ текста программы можно разделить на два хода. Первым ходом будет являться анализ отдельных символов, а вторым сопоставление с правилами обработки этих символов. Для этого существует специальная функция, которая перемещает указатель на начало слова (это так же могут быть символы и любые другие текстовые конструкции) и возвращает тип этого слова. В зависимости от типа слова программа переходит к выбору действий. К примеру, при встрече слова var, программа запускает специальную функцию анализа переменных.

Все переменные заносятся в динамический список в памяти, и затем используются в главной программе в качестве проверки правильности введенных арифметических и логических выражений. При анализе блока переменных, так же используется подфункция обработки типов. К примеру когда программа встречает ключевое слово integer в выходной буфер она записывает int и далее выводит созданный ранее список переменных. Примерно то же самое происходит, если программа встречает ключевое слово type, но в данном случае создается еще одна динамическая конструкция (стэк) в которую помещаются все пользовательские типы. Динамически создается и таблица идентификаторов этих типов и правила обработки этих типов при объявлении. Таким образом, основная программа обработки представляет собой цикл обрабатывающие различные разделы программы - function, var, type и главную программу. Внутри каждой подпрограммы существуют специальные функции обработки конструкций языка.

Таким образом повышается строгость проверки синтаксиса, ведь даже верно описанные синтаксические конструкции языка Pascal в неверном разделе будут обработаны программой как ошибки.

# Структура программы

Errore\_oper – Подпрограмма обработки ошибок, определяет места ошибки и заносит данные о количестве ошибок в специальную переменную.

find\_Comment – Находит в коде транслируемой программы комментарии

ReadWord – Подпрограмма анализа текста на наличие токенов. Определяет тип токена и делает его проверку. Возвращает позицию начала, количество символов в токене, а так же его тип. Заодно пропускает пробелы.

isBegin – Подпрограмма обработки блока операторов, запускается после того как во входном файле встречено слово begin.

isVarBlock – Подпрограмма обработки блока переменных – var. Проверяет конструкции на их тип: обычные переменные или метка. После чего передает поток управления в ту или иную функцию.

fin\_comment – Подпрограмма обработки окончания комментария.

ReadWords – Подпрограмма посимвольно просматривает файл и определяет к чему можно отнести данный символ.

Assign\_a\_var – Подпрограмма обработки операции присваивания. Замещает:= на = также проверяет соответствие переменных в выражении с объявленными ранее переменными.

funcfunction – подпрограмма обработки конструкции function в тексте.

funcelse – подпрограмма оставляющая оперaтор else.

funcif – подпрограмма обрабатывающая оператор if.

funcrecord – Подпрограмма обрабатывает блок после слова type, в котором, в соответствии со спецификой нашей программы, могут быть только конструкция record или ошибочные операторы.

isKeyWord – Подпрограмма обрабатывает массив с ключевыми словами, проверяет на наличие данного слова и возвращает его номер, если слово было найдено. Если же слово незнакомо, то подпрограмма возвращает 0, что означает, что встреченное слово не является ключевым словом.

funcread – подпрограмма распознающая конструкцию read и заменяющая ее на scanf.

funcbegin – подпрограмма находит операторы Begin и End и заменяет их на фигурные скобки, она включает все функции по обработке текста.

# Описание констант

Key\_Words\_num – число, обозначающее размер словаря ключевых слов. В нашей программе оно равно 16. Это такие слова как: "true", "var", "real", "boolean", "false", "type","record", "and", "or", "repeat", "do", “until”, "begin", "end", “procedure”, “read”

lenprog – Данная константа необходима, так как мы представляем всю программу в виде одной строки, поэтому она является ограничителем работы программы. Таблицы идентификаций:

Таблица 1

Таблица номеров идентификации для ключевых слов

|  |  |
| --- | --- |
| Ключевое слово | Идентификационный номер |
| KeyW\_begin | 1 |
| KeyW \_end | 2 |
| KeyW \_var | 3 |
| KeyW \_real | 4 |
| KeyW \_boolean | 5 |
| KeyW \_false | 6 |
| KeyW \_true | 7 |
| KeyW \_do | 8 |
| KeyW \_else | 9 |
| KeyW \_for | 10 |
| KeyW \_function | 11 |
| KeyW \_type | 12 |
| KeyW \_record | 13 |
| KeyW \_and | 14 |
| KeyW \_or | 15 |
| KeyW \_read | 16 |

Таблица 2

Таблица идентификаторов типов переменных

|  |  |
| --- | --- |
| Тип | Идентификационный номер |
| IntType | 1 |
| RealType | 2 |
| BoolType | 3 |
| ArrayType | 4 |

транслятор pascal спецификация текст

Таблица 3

Таблица типов слов

|  |  |
| --- | --- |
| Тип слова | Идентификационный номер |
| errore\_word | 0 |
| w\_spa | 1 |
| string\_word | 2 |
| number | 3 |
| w\_ch | 4 |

# Ввод тестов и вывод результатов

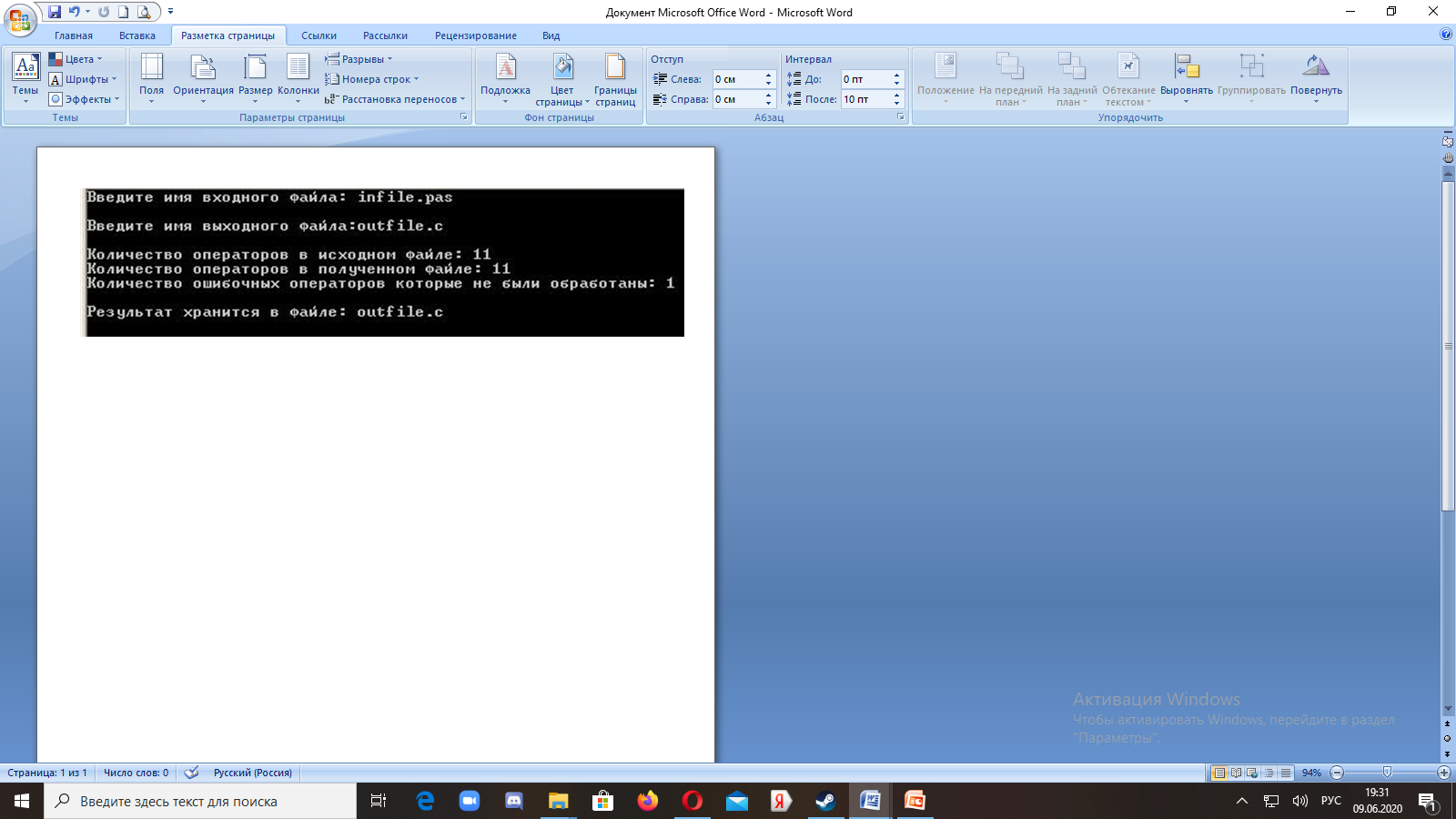


Рисунок 1 - Вид экрана после обработки входного файла

Код программы до обработки:

type

rec = record

x,y: real;

end;

var

t:rec;

a,b,k,s: real;

f: boolean;

{Vars}

{Begin pr}

procedure pr(k: integer; var d: real);

begin

repeat

read(a);

until f or f;

End; {End pr}

begin

f:= true;

repeat

read(a);

until f and f;

write(a);

end.

Код программы после обработки:

#include <stdio.h>

struct rec

{

float x,y;

};

rec t;

float a,b,k,s;

int f;

/\*Vars\*/

/\*Begin pr\*/

void pr(int k, float & d)

{

{

do {

scanf("%f",&a);

} while (f||f);

}

/\*End pr\*/

void main()

{

f=1;

do {

scanf("%f",&a);

} while (f&&f);

< Ошибка! >

write(a);

# Выводы

В курсовой работе разработана программа, позволяющая частично транслировать с языка Паскаль на язык Си некоторые функции, обрабатывать комментарии, целочисленный тип переменных, начало и конец цикла, а так же определять логические и арифметические выражения.

# Приложение

**Текст программы**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <conio.h>

#include <stdio.h>

#include <string.h>

#include <locale.h>

FILE \*file\_in, \*file\_out;

const int lenprog = 10000;//весь исходный текст представляется одной большой символьной строкой, а не массивом символов, это максимальная длина

const int Key\_Words\_num = 16;

const char \* KWs[Key\_Words\_num] =

{

"begin", "end", "var", "real",

"boolean", "false","true", "do",

"repeat", "until", "function","type",

"record", "and", "or", "read"

};// это массив ключевых слов, которые обрабатываются данной программой

//ключевыей слова

//каждому ключевому слову присвоен номер, это будет дальше испоьзоваться в проге

#define keyW\_begin 1

#define keyW\_end 2

#define keyW\_var 3

#define keyW\_real 4

#define keyW\_boolean 5

#define keyW\_false 6

#define keyW\_true 7

#define keyW\_do 8

#define keyW\_else 9

#define keyW\_for 10

#define keyW\_function 11

#define keyW\_type 12

#define keyW\_record 13

#define keyW\_and 14

#define keyW\_or 15

#define keyW\_read 16

#define keyW\_procedure 17

//типы

//тоже присвоены номера

#define IntType 1 //целый

#define RealType 2 //дробный

#define BoolType 3 //логический

#define ArrayType 4 //массив

const int cuservars = 6;

int uvars;

//тип "слова"

#define errore\_word 0 //ошибка

#define w\_spa 1 //пробелы

#define string\_word 2 //строка

#define number 3 //число

#define w\_ch 4 //символ (разделитель)

//собственно это наши входные и выходные большие строки, в эти большие

//строки записываются также символы конца строки '\n', что и используется в дальнейшем для формирования выходного теста построчно

char instr[lenprog],

outstr[lenprog];

int inlen, outlen, inpos;//текущие позиции (строки входные, выходные, позиция в строке входной)

int koper1, koper2, ker;//эти переменные считают количество операторов во входном, выходном файле и кол-во ошибок

int lenght\_world; //длина "слова"

int wtype; //тип "слова"

int wnumf; //0 - целое число, 1 - дробное

int dt; //размер отступа (от начала строки)

typedef struct var { //элемент списка переменных

char s[64]; //идент-р переменной

int tp; //тип переменной

var \* next;

};

typedef struct types { //элемент списка типов

char s[64]; //идент-р переменной

int tid;

types \* next;

};

var \*vars, \*lvars; //списки переменных

//(глобальных и локальных)

types \* typeslist;

// прототипы функций

int fle();

// Дальнейшие функции описывают работу с динамической памятью,

// В которой создается список с переменными, их типами и названием

// Используя этот список, затем идет построение раздела объявления переменных

// в c++

//добавить переменную в список

// где gl - тип переменной, глобальная или локальная.

int addvar(int gl) {

var \*np, \*p;

if (gl) p = vars; else p = lvars;

// int memcmp(const void \*buf1, const void \*buf2, size\_t count);

// Функция memcmp() сравнивает первые count символов массивов, адресуемых параметрами buf1 и buf2.

// buf1 меньше buf2 = меньше нуля

// buf1 равен buf2 = 0

// buf1 больше buf2 = больше нуля

while (p) {

if (strlen(p->s) == lenght\_world &&

!memcmp(p->s, &instr[inpos], lenght\_world)) return 0;

p = p->next;

}

np = new var;

// void \*memcpy(void \*to, const void \*from, size\_t count);

// Функция memcpy() копирует count символов из массива,

// адресуемого параметром from, в массив, адресуемый параметром to.

// Если заданные массивы перекрываются, поведение функции memcopy() не определено.

memcpy(np->s, &instr[inpos], lenght\_world);

// после имени переменной мы записываем 0 (метка

// что бы потом читать имя переменной до этого символа)

np->s[lenght\_world] = 0;

// тип переменной задаем как -1 (метка, что бы потом переопределить)

np->tp = -1;

// Если глобальная переменная, то добавляем этот элемент в начало

// (т.е. получается стек из переменных) списка глобальных переменных

// иначе в начало списка локальных

if (gl) {

np->next = vars;

vars = np;

}

else {

np->next = lvars;

lvars = np;

}

return 1;

}

int addnewtype()

{

types \*head = typeslist, \*ntype;

while (head) {

if (strlen(head->s) == lenght\_world &&

!memcmp(head->s, &instr[inpos], lenght\_world)) return 0;

head = head->next;

}

ntype = new types;

memcpy(ntype->s, &instr[inpos], lenght\_world);

ntype->s[lenght\_world] = 0;

ntype->tid = uvars;

uvars++;

ntype->next = typeslist;

typeslist = ntype;

return 1;

}

//установка типа переменной

// если gl - идентификатор глобальности = 1, то это глобальная переменная

// иначе локальная

// есть четыре типа переменных (описаны выше)

// конструкцию while можно расшифровать так:

// пока указатель на структуру p существует выполнять:

// если тип переменной -1 то присвоить ей заданный тип

// и далее шаг вперед

void settype(int gl, int type) {

var \*p;

if (gl) p = vars; else p = lvars;

while (p) {

if (p->tp == -1) p->tp = type;

p = p->next;

}

}

// получение типа переменной

// при этом тип уже должен был заранее задан

int vartype() {

var \* p;

// обработка локальных переменных

p = lvars;

while (p) {

if (strlen(p->s) == lenght\_world && // если слово во входном тексте равно названию переменной (в списке),

!memcmp(p->s, &instr[inpos], lenght\_world)) return p->tp; // то вернуть тип переменной

p = p->next;

}

// обработка глобальных переменных

p = vars;

while (p) {

if (strlen(p->s) == lenght\_world &&

!memcmp(p->s, &instr[inpos], lenght\_world)) return p->tp;

p = p->next;

}

return 0;

}

char\* usertype()

{

types \* p;

// обработка локальных переменных

p = typeslist;

while (p) {

if (strlen(p->s) == lenght\_world && // если слово во входном тексте равно названию переменной (в списке),

!memcmp(p->s, &instr[inpos], lenght\_world)) return p->s; // то вернуть тип переменной

p = p->next;

}

return 0;

}

// освобождение списка переменных

// просто удаление обоих списков

// как для локальных переменных, так и для глобальных

void freevars(int gl) {

var \*p, \*pp;

p = lvars;

while (p) {

pp = p;

p = p->next;

delete pp;

}

lvars = NULL;

if (gl) {

p = vars;

while (p) {

pp = p;

p = p->next;

delete pp;

}

vars = NULL;

}

}

// Одна из главных функций получение слова, здесь посимвольно анализируется текст

// Вкратце описание выглядит так

// прога анализирует текст посимвольно и каждый раз определяет тип символа

// (число, буква, пробел ит.д.),

// когда удается ясно определить значение этого символа (цифра, переменная и т.д.)

// или последовательности символов( перемнная, ключевое слово и т.д.)

// выполнение функции прерывается она возвращает тип этого слова

//Функция возвращает один из следующих кодов

////тип "слова"

// errore\_word 0 //ошибка

// w\_spa 1 //пробелы

// string\_word 2 //строка

// number 3 //число

// w\_ch 4 //символ (разделитель, ...)

// чтение слова

int ReadWord()

{

int st = 0;

char c; // символ который сравнивается

lenght\_world = 0;// длина анализуруемого слова

while (lenght\_world + inpos < inlen) {

c = instr[inpos + lenght\_world];

switch (st) { // получение символа и дальнейший выбор направления обработки

case 0:

if (c == ' ' || c == '\t' || c == '\n') st = 1;

else

if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) st = 2;

else

if (c >= '0' && c <= '9') st = 3;

else

if (

c == '.' || c <= ',' || c >= ':' || c <= ';' ||

c == '+' || c <= '-' || c >= '\*' || c <= '/' ||

c == '\''

)

{

lenght\_world = 1; return wtype = w\_ch;

}//возвращает, что это символ

else { lenght\_world = 0; return wtype = errore\_word; }

break;

case 1:

if (c == ' ' || c == '\t' || c == '\n') lenght\_world++;

else return wtype = w\_spa; ///возвращает, что это символ пробел

break;

case 2:

if (

(c >= 'A' && c <= 'Z') ||

(c >= 'a' && c <= 'z') ||

(c >= '0' && c <= '9') ||

c == '\_'

) lenght\_world++;

else return wtype = string\_word;

break;

case 3:

if (c >= '0' && c <= '9') lenght\_world++; else

if (c == '.'&& instr[inpos + lenght\_world + 1] != '.') { // говорит, что есть точка в числе, т.е оно дробное

lenght\_world++;

st = 5;

}

else {

wnumf = 0;

return wtype = number;//возвращает, что это число

}

break;

case 5:

if (c >= '0' && c <= '9') lenght\_world++; else {

wnumf = 1;

return wtype = number;//возвращает, что это число (дробную часть)

}

}

}

lenght\_world = 0;

return 0;

}

//запись строки в выходной буфер

void put\_str(char \* s) {

int l = strlen(s);

memcpy(&outstr[outlen], s, l);

outlen += l;

}

int scmp(char \* m, char \* s, int n) {

int l = strlen(s);

if (n > l) l = n;

return \_memicmp(m, s, l);

}

//запись символа в выходной буфер

void put\_char(char c) {

outstr[outlen] = c;

outlen++;

}

void wcopy() {

memcpy(&outstr[outlen], &instr[inpos], lenght\_world);

inpos += lenght\_world;

outlen += lenght\_world;

}

int wequ(char \* s) {

return (!scmp(&instr[inpos], s, lenght\_world));

}

//пропускает пробелы

void wskip() {

inpos += lenght\_world;

}

void wstr(char \* s) {

// char \*strncpy (char \*dst, const char \*src, size\_t len);

// dst — указатель на буфер назначения.

// src — указатель на исходную строку.

// len — максимальное количество копируемых символов (см. раздел Безопасность ниже).

// Функция копирует из строки src в буфер dst не более чем len символов

// (включая нулевой символ), не гарантируя завершения строки

// нулевым символом (если длина строки src больше или равна len).

// Если длина строки src меньше len, то буфер добивается до len нулями.

// Функция возвращает значение dst.

strncpy(s, &instr[inpos], lenght\_world);

}

//получение слов, и если тип слова пробелы, запускает процедуру, пропускающую пробелы

int ReadWordS() {

ReadWord();

if (wtype == w\_spa) {

wskip();

ReadWord();

}

return wtype;

}

//увеличение отступа

void inc\_dt()

{

dt += 2;

}

//уменьшение отступа

void dec\_dt()

{

dt -= 2;

}

//вывод отступа

void put\_dt()

{

for (int i = 0; i < dt; i++) put\_char(' ');

}

//вывод отступа

void put\_dt11()

{

char s[10];

for (int i = 0; i < dt; i++) {

sprintf(s, "%d", i / 2);

put\_str(s);

}

}

//открытие файла

int ReadText(char \* s1)

{

if ((file\_in = fopen(s1, "rt")) == NULL) {

return 0;

}

fseek(file\_in, 0, SEEK\_END);

inlen = ftell(file\_in);

printf("%i", inlen);

fseek(file\_in, 0, SEEK\_SET);

if (inlen > lenprog) inlen = lenprog;

inlen = fread(instr, 1, inlen, file\_in);

instr[inlen] = 0;

inpos = 0;

outlen = 0;

return 1;

}

//вывод обработанного текста

int PutText(char \* s2)

{

if ((file\_out = fopen(s2, "wt")) == NULL) {

return 0;

}

fwrite(outstr, outlen, 1, file\_out);

return 1;

}

//вывод ошибочного оператора

void Errore\_oper()

{

put\_char('\n');

put\_str((char \*)"< Ошибка! > \n");

int k;

while (1) {

ReadWordS();

if (instr[inpos] == ';' || inpos >= inlen) {

wcopy();

break;

};

wcopy();

}

ker++; // увеличение счетчика ошибок

}

//проверка на комментарий

int find\_Comment() {

return (instr[inpos] == '{' ||

instr[inpos] == '(' || instr[inpos + 1] == '\*');

}

//проверка на закрытие комментария

void fin\_Comment() {

if (instr[inpos] == '{') {

outstr[outlen] = '/';

outstr[outlen + 1] = '\*';

inpos++;

outlen += 2;

while (instr[inpos] != '}' && inpos < inlen) {

if (inpos >= inlen) return;

outstr[outlen] = instr[inpos];

inpos++;

outlen++;

}

outstr[outlen] = '\*';

outstr[outlen + 1] = '/';

inpos++;

outlen += 2;

}

else {

outstr[outlen] = '/';

outstr[outlen + 1] = '\*';

inpos += 2;

outlen += 2;

while (!(instr[inpos] == '\*' && instr[inpos + 1] == ')')

&& inpos < inlen) {

if (inpos >= inlen) return;

outstr[outlen] = instr[inpos];

inpos++;

outlen++;

}

outstr[outlen] = '\*';

outstr[outlen + 1] = '/';

inpos += 2;

outlen += 2;

}

put\_char('\n');

}

// здесь идет проверка на ключевое слово,

// сравниваются полученное нами слово со словом из списка

// и если да, то возвращается номер слова в списке

// проверка на ключевое слово

int funckeyW() {

for (int i = 0; i < Key\_Words\_num; i++) {// просмотр всего массива Key\_Words\_num

if (!scmp(&instr[inpos], (char \*)KWs[i], lenght\_world))

return i + 1;

}

return 0;

}

// вот это обработка переменных после слова var,

// здесь идут обращения к тем четырем функциям работающим со списком перемнных

// обработка описания переменных: x1,..,xn: тип;

int fIntRealBoolAr(int svar, int gl) {

char s[256];

int label;

int sp = 0;

ReadWordS();

while (1) {

if (wtype != string\_word || funckeyW() || gl > 0 && vartype()) return 0;

addvar(gl);

if (svar) {

s[sp] = '&';

s[sp + 1] = ' ';

sp += 2;

}

memcpy(&s[sp], &instr[inpos], lenght\_world);

inpos += lenght\_world;

sp += lenght\_world;

ReadWordS();

if (instr[inpos] == ',') {

s[sp] = ',';

inpos++;

sp++;

}

else break;

ReadWordS();

} // while(1)

if (instr[inpos] == ':') { //тип переменных

inpos++;

ReadWordS();

if (wtype != string\_word) return 0;

if (!scmp(&instr[inpos], (char \*)"boolean", lenght\_world)) {

settype(gl, BoolType);

put\_str((char \*)"int ");

wskip();

memcpy(&outstr[outlen], &s[0], sp);

outlen += sp;

}

else

if (!scmp(&instr[inpos], (char \*)"real", lenght\_world)) {

settype(gl, RealType);

put\_str((char \*)"float ");

wskip();

memcpy(&outstr[outlen], &s[0], sp);

outlen += sp;

}

else

if (!scmp(&instr[inpos], (char \*)"integer", lenght\_world)) {

settype(gl, IntType);

put\_str((char \*)"int ");

wskip();

memcpy(&outstr[outlen], &s[0], sp);

outlen += sp;

}

else

if (!scmp(&instr[inpos], usertype(), lenght\_world)) {

put\_str(usertype());

put\_str((char \*)" ");

wskip();

memcpy(&outstr[outlen], &s[0], sp);

outlen += sp;

}

else

if (!scmp(&instr[inpos], (char \*)"array", lenght\_world)) {

wskip();

settype(gl, ArrayType);

ReadWordS();

if (instr[inpos] != '[') return 0;

s[sp] = '[';

inpos++;

sp++;

while (1) {

ReadWordS();

if (instr[inpos] != '1') return 0;

wskip();

ReadWordS();

if (instr[inpos] != '.' || instr[inpos + 1] != '.')

return 0;

inpos += 2;

ReadWordS();

if (wtype != number) return 0;

memcpy(&s[sp], &instr[inpos], lenght\_world);

inpos += lenght\_world;

sp += lenght\_world;

ReadWordS();

if (instr[inpos] == ']') {

s[sp] = ']';

inpos++;

sp++;

break;

}

if (instr[inpos] == ',') {

inpos++;

s[sp] = ']';

sp++;

s[sp] = '[';

sp++;

}

}

ReadWordS();

if (wtype != string\_word) return 0;

if (scmp(&instr[inpos], (char \*)"of", lenght\_world)) return 0;

wskip();

ReadWordS();

if (wtype != string\_word) return 0;

if (scmp(&instr[inpos], (char \*)"real", lenght\_world)) return 0;

wskip();

put\_str((char \*)"float ");

memcpy(&outstr[outlen], &s[0], sp);

outlen += sp;

}

}

else return 0;

return 1;

}

//обработка описания блока переменных

int funcvar(int gl) {

inpos += lenght\_world;

ReadWordS();

do {

koper1++;

if (find\_Comment()) { //комментарии

fin\_Comment();

koper2++;

continue;

}

put\_dt();

if (!fIntRealBoolAr(0, gl)) Errore\_oper();

else koper2++;

ReadWordS();

if (instr[inpos] != ';')

return 0;

wskip();

put\_str((char \*)";\n");

ReadWordS();

if (wtype != string\_word || funckeyW())

return 1;

} while (1);

}

//обработка блока описания переменных

int funcvardescr() {

inpos += lenght\_world;

int k, svar;

ReadWordS();

do {

k = funckeyW();

svar = k == keyW\_var;

if (svar) {

wskip();

ReadWordS();

}

if (!fIntRealBoolAr(svar, 0)) return 0;

ReadWordS();

if (instr[inpos] != ';') return 1;

wskip();

put\_str((char \*)", ");

ReadWordS();

k = funckeyW();

if (wtype != string\_word || k && k != keyW\_var) return 0;

} while (1);

}

int funcbegin(int k);

//обработка процедуры

int funcprocedure() {

//чтение заголовка процедуры

wskip();

put\_str((char \*)"\nvoid ");

ReadWordS();

if (wtype != string\_word || vartype()) return 0;

addvar(1);

settype(1, 10); //резервируем идентификатор

wcopy();

ReadWordS();

if (instr[inpos] != '(') return 0;

put\_char('(');

//список параметров

if (!funcvardescr()) return 0;

ReadWordS();

if (instr[inpos] != ')') return 0;

wcopy();

ReadWordS();

if (instr[inpos] != ';') return 0;

wskip();

put\_str((char \*)"\n{\n");

inc\_dt();

//тело процедуры

int b;

do {

b = 1;

ReadWordS();

if (!scmp(&instr[inpos], (char \*)"var", lenght\_world)) {

koper1++;

if (!funcvar(0)) return 0;

}

else

if (!scmp(&instr[inpos], (char \*)"begin", lenght\_world)) {

if (!funcbegin(2)) return 0;

b = 0;

}

else

if (find\_Comment()) fin\_Comment();

else return 0;

} while (b == 1);

freevars(0);

inpos++;

return 1;

}

//обработка оператора record

int funcrecord() {

wskip();

put\_str((char \*)"struct ");

ReadWordS();

if (wtype != string\_word || funckeyW()) return 0;

addnewtype();

wcopy();

ReadWordS();

if (instr[inpos] != '=') return 0;

wskip();

ReadWordS();

if (!wequ((char \*)"record")) return 0;

put\_str((char \*)"\n{\n");

inc\_dt();

if (!funcvar(-1)) return 0;

dec\_dt();

ReadWordS();

if (!wequ((char \*)"end")) return 0;

wskip();

put\_char('}');

ReadWordS();

if (instr[inpos] != ';') return 0;

wcopy();

put\_str((char \*)"\n\n");

inpos += lenght\_world;

return 1;

}

//обработка оператора read

int funcread(int ln) {

char s[256];

int sp;

int t;

wskip();

put\_dt();

put\_str((char \*)"scanf");

ReadWordS();

if (instr[inpos] != '(') return 0;

inpos++;

put\_str((char \*)"(\"");

sp = 0;

while (1) {

ReadWordS();

if (wtype != string\_word) return 0;

t = vartype();

if (t == IntType) put\_str((char \*)"%d");

else if (t == RealType) put\_str((char \*)"%f");

else return 0;

s[sp] = '&';

sp += lenght\_world;

memcpy(&s[sp], &instr[inpos], lenght\_world);

inpos += lenght\_world;

sp += lenght\_world;

ReadWordS();

if (instr[inpos] != ',') break;

s[sp] = instr[inpos];

inpos++;

sp++;

}

put\_str((char \*)"\",");

memcpy(&outstr[outlen], s, sp);

outlen += sp;

ReadWordS();

if (instr[inpos] != ')') return 0;

inpos++;

put\_char(')');

ReadWordS();

if (instr[inpos] != ';') return 0;

inpos++;

if (ln) put\_str((char \*)"; printf(\"\\n\");\n");

else put\_str((char \*)";\n");

return 1;

}

//обработка арифметического выражения

int fae() {

ReadWordS();

if (instr[inpos] == '+') {

wcopy();

}

else

if (instr[inpos] == '-') {

wcopy();

}

while (1) {

ReadWordS();

if (wtype == number) wcopy(); else

if (wtype == string\_word && vartype() == IntType) wcopy(); else

if (wtype == string\_word && vartype() == RealType) wcopy(); else

if (wtype == string\_word && vartype() == ArrayType) {

wcopy();

ReadWordS();

if (instr[inpos] == '[') {

wcopy();

while (1) {

if (!fae()) return 0;

put\_str((char \*)"-1");

ReadWordS();

if (instr[inpos] == ']') {

wcopy();

break;

}

if (instr[inpos] == ',') {

wskip();

put\_str((char \*)"][");

}

}

}

}

else

if (instr[inpos] == '(') {

wcopy();

if (!fae()) return 0;

ReadWordS();

if (instr[inpos] != ')') return 0;

inpos++;

put\_char(')');

}

else return 0;

ReadWordS();

char c = instr[inpos];

if (c == '+' || c == '-' || c == '\*' || c == '/') wcopy();

else return 1;

}

}

//обработка арифметического выражения

int ae() {

char c, c1;

if (!fae()) return 0;

ReadWordS();

c = instr[inpos];

c1 = instr[inpos + 1];

if (c == '<'&&c1 == '>') {

inpos += 2;

put\_str((char \*)"!=");

}

else

if (c == '=') {

inpos++;

put\_str((char \*)"==");

}

else

if (c == '>' || c == '<') {

if (c1 == '=') {

inpos += 2;

}

else wcopy();

}

ReadWordS();

if (!fae()) return 0;

return 1;

}

//обработка логического выражения

int fle() {

int k;

char c, c1;

int arifm, ip, op;

while (1) {

ReadWordS();

k = funckeyW();

int ip, op;

ip = inpos;

op = outlen;

arifm = 0;

if (instr[inpos] == '+' ||

instr[inpos] == '(' ||

instr[inpos] == '-' ||

wtype == string\_word && !funckeyW() ||

wtype == number)

arifm = ae();

if (!arifm) {

inpos = ip;

outlen = op;

ReadWordS();

k = funckeyW();

//------

if (wtype == string\_word && k == keyW\_true) {

wskip();

put\_char('1');

}

else

if (wtype == string\_word && k == keyW\_false) {

wskip();

put\_char('0');

}

else

if (wtype == string\_word && vartype() == BoolType) wcopy(); else

if (instr[inpos] == '(') {

wcopy();

if (!fle()) return 0;

ReadWordS();

if (instr[inpos] != ')') return 0;

inpos++;

put\_char(')');

}

else return 0;

}

ReadWordS();

k = funckeyW();

if (k == keyW\_or) put\_str((char \*)"||"); else

if (k == keyW\_and) put\_str((char \*)"&&");

else return 1;

wskip();

}

}

//проверка на присваивание

int asign\_a\_var() {

int type = vartype();

if (!(type == BoolType || type == RealType || type == IntType)) return 0;

//put\_char('\n');

put\_dt();

wcopy();

ReadWordS();

if (instr[inpos] != ':' || instr[inpos + 1] != '=')

return 0;

put\_char('=');

inpos += 2;

if (type == BoolType) {

if (!fle()) return 0;

}

else

if (!fae()) return 0;

ReadWordS();

if (instr[inpos] != ';') return 0;

wcopy();

put\_char('\n');

return 1;

}

//обработка оператора repeat

int funcelse() {

wskip();

put\_dt();

put\_str((char \*)"do {\n");

inc\_dt();

return 1;

}

//обработка оператора repeat

int funcif() {

wskip();

dec\_dt();

put\_dt();

put\_str((char \*)"} if (");

if (!fle()) return 0;

put\_char(')');

ReadWordS();

if (instr[inpos] != ';') return 0;

inpos++;

put\_str((char \*)";\n");

return 1;

}

//обработка блока операторов

int funcbegin(int gl) {

int rep\_n = 0;

if (gl != 3) wskip();

if (gl == 1) put\_str((char \*)"\n\nvoid main()\n");

if ((gl != 2) || (gl != 3)) {

put\_dt();

put\_str((char \*)"{\n");

inc\_dt();

}

int b;

do {

b = 1;

ReadWordS();

if (find\_Comment()) {

fin\_Comment();

continue;

}

switch (funckeyW()) {

case keyW\_begin:

koper1++;

if (!funcbegin(0)) return 0;

koper2++;

break;

case keyW\_read:

koper1++;

if (!funcread(0)) return 0;

koper2++;

break;

case keyW\_procedure:

if (!funcprocedure()) return 0;

break;

case keyW\_end:

koper1++;

if (gl == 3) return 3;

wskip();

dec\_dt();

put\_dt();

put\_str((char \*)"}\n");

ReadWordS();

if (gl == 1 && instr[inpos] == '.' ||

gl != 1 && instr[inpos] == ';') {

wskip();

koper2++;

return 1;

}

else

{

wskip();

return 0;

}

case 0:

if (!asign\_a\_var()) return 0; //присваивание

break;

default:

return 0;

}

} while (b);

return 1;

}

//главная функция преобразования ттекста

//преобразование текста

int Translate()

{

int b;

int k;

koper1 = koper2 = 0;

put\_str((char \*)"#include <stdio.h>\n\n"); //для scanf и printf

do {

b = 1;

ReadWordS();

k = funckeyW(); //проверка на ключевое слово

if (k == keyW\_var) { //начало блока переменных

koper1++;

if (!funcvar(1)) {

Errore\_oper();

}

else koper2++;

}

else

if (k == keyW\_type) { //блок описания типов

koper1++;

if (!funcrecord()) {

Errore\_oper();

}

else koper2++;

}

else

if (k == keyW\_procedure) { //процедура

if (!funcprocedure()) {

Errore\_oper();

}

}

else

if (k == keyW\_begin) { //начало блока операторов

if (!funcbegin(1)) {

Errore\_oper();

}

b = 0;

}

else

if (find\_Comment()) fin\_Comment(); //комментарии

else {

koper1++;

Errore\_oper();

};

} while (b == 1);

if (instr[inpos] != '.') return 0; //точка в конце программы

inpos++;

return 1;

}

void main()

{

setlocale(LC\_ALL, "Russian");

char s[128];

printf("Введите имя файла(с расширением) с кодом на языке Pascal:");

scanf("%s", s);

if (!ReadText(s))

{

printf("\nОшибка открытия файла!");

}

vars = NULL;

lvars = NULL;

uvars = cuservars;

Translate();

freevars(1);

printf("\nВведите имя выходного файлас расширением .с:");

scanf("%s", s);

if (!PutText(s))

{

printf("\nОшибка создания файла!");

}

printf("\nКоличество операторов"

" в исходном файле: %d", koper1);

printf("\nКоличество операторов"

" в полученном файле: %d", koper2);

printf("\nКоличество ошибочных операторов"

" которые не были обработаны: %d", ker);

printf("\n\nРезультат хранится в файле: %s", s);

fclose(file\_in);

fclose(file\_out);

while (!\_kbhit());

}