Федеральное агентство по образованию

ОМСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Кафедра информатики и вычислительной техники

**Пояснительная записка**

**к курсовому проекту**

по дисциплине СПО

Выполнил:

студент гр. ИВТ-425

Елизов В. В.

Проверил:

к.т.н., доцент

Флоренсов А. Н.

Омск 2009

**Содержание**

[Задание 3](#_Toc199875837)

[Описание разработанного языка 4](#_Toc199875838)

[Описание модулей 7](#_Toc199875839)

[main.c 8](#_Toc199875840)

[init.c 9](#_Toc199875841)

[lexer.c 10](#_Toc199875842)

[parser.c 12](#_Toc199875843)

[symbol.c 19](#_Toc199875844)

[emitter.c 21](#_Toc199875845)

[errors.c 24](#_Toc199875846)

[Список использованной литературы 25](#_Toc199875847)

**Задание №4**

Разработать на основе предиктивного анализатора интерпретатор программ для языка программирования, который описательно задается следующими определениями:

1. язык обеспечивает вычисления на основе типов double и int языка Си (используемого компилятором компиляторов), предполагается использование директив определения типа;
2. арифметические выражения могут использоваться сколь угодно сложной структуры, но не должны содержать обращений к функциям;
3. язык содержит операторы цикла, задаваемые ключевым словом while, и условные операторы if - else; допускаются операторы break;
4. условия задаются в виде

*переменная операция\_сравнения число*

1. для выдачи результатов служат операторы

print *список имен переменных*

printn *список имен переменных*

причем оператор printn выдает округленные целые значения переменных, а для ввода оператор

input *список имен переменных*

где *список имен переменных* представляет собой перечисление через запятую;

1. остальные синтаксические элементы языка выбираются разработчиком самостоятельно (скобки, средства указания конца оператора, блока операторов, символ присваивания, обозначения операций сравнения и т.п.);
2. Имена программных объектов должны формироваться по общеупотребительному правилу: первым символом должна быть произвольная латинская буква, последующими символами могут быть цифры или латинские буквы.

Основным результатом работы компилятора должен быть исполняемый файл интерпретатора; в качестве дополнительного файла, выдаваемого компилятором при указании опции запроса для него, должен формироваться файл таблицы символических имен.

Представляемые результаты разработки должны включать пояснительную записку и исходный файл на языке Си, детально прокомментированный, с текстами подключаемых файлов, созданных разработчиком. Рекомендуется проводить разработку постепенно, реализую до получения результирующего файла сначала первые пункты его задания, затем добавляя к ним группу следующих и т.д.

**Описание разработанного языка**

Разработанный интерпретатор работает с особым языком программирования, который, для упрощения, далее в тексте будет называться NL.

Алфавит языка можно разделить на 6 групп:

1. *Символы пробела и табуляции*.
2. *Буквы* - это прописные и строчные буквы латинского алфавита.
3. *Арабские цифры* - это цифры от 0 до 9.
4. Разделители и ограничители: , ; ( ) { }
5. Знаки операций + - \* / := != == > < <= >=
6. Зарезервированные слова: if, else, while, input, print, printn

Для записи программы не требуется какого-либо специального формата файла. Можно использовать любое расширение.

Любая программа должна начинаться символа { и заканчиваться символом }. Они называются операторные скобки и указывают на начало и конец процедуры.

К тому же любая операция должна заканчиваться символом ; (точка с запятой).

Простейшая программа может выглядеть следующим образом:

{

print 5.3;

}

При выполнении этой программы на экране появится число 5.3. Можно заметить, что оператор print выводит на экран число (значение переменной), которое следует за ним. Для вывода последовательности аргументы необходимо указать через запятую.

Также существует оператор printn, который выполняет те же самые функции, что и print, только на экран выведется лишь целочисленное значение. Например, если выполнить программу:

{

printn 5.3;

}

То на экране появится округленный результат – число 5.

Для того чтобы использовать переменную необходимо либо ввести ее значение при помощи оператора input, либо использовать присваивание (оператор :=). Переменная описывается последовательностью латинских букв, цифр и знака подчеркивания, начинающейся с буквы.

{

a:=5.3;

input b;

print a,b;

printn a,b;

}

В вышеописанном примере используется две переменных, a и b. Первая принимает значение 5.3, значение же второй необходимо ввести вручную и нажать клавишу ввода. Результатом выполнения этой программы будет 4 числа: 5.3, значение переменной b, 5 и округленное до целого значение второй переменной. Причем, каждое число будет выведено в новой строке.

Для ввода нескольких переменных, как и для вывода можно использовать последовательность имен, перечисленных через запятую.

Язык NL позволяет выполнять простейшие операции (над числами и переменными) такие как: сложение, вычитание, умножение и деление. Которые записываются так же, как и в математике: +, -, \*, /. Порядок выполнения этих операций также соответствует привычному. Для изменения приоритета нужно использовать скобки.

{

input a,b,c;

d:=(a\*(a+b\*b)+c)\*c;

print d;

}

Для выполнения более сложных операций используется оператор условного перехода if-else – когда последовательность действий зависит от какого-либо условия. Структура условного оператора в полной форме имеет следующий вид:

if (условие) {действия1} else { действия 2};

Условие - это выражение логического типа, которое может принимать два значения: истина или ложь. Для записи условия используются операции сравнения:

* == - равно
* != - неравно
* > - больше
* < - меньше
* >= - меньше-равно
* <= - больше-равно

Рассмотрим на примере работу условного оператора:

{

input a,b;

if(a==b)

{

print 10;

}

else

{

print 0;

};

}

В начале программы вводятся два числа a и b. Затем они сравниваются на равенство. Если a равно b, то будут выполняться действия, находящиеся в операторных скобках непосредственно за словом if. То есть, на экран выведется 10. Затем программа «перепрыгнет» через операторные скобки после слова else, на чем выполнение закончится. Если же условие окажется ложным (a не равно b), то пропустится первая последовательность действий и выполнится вторая, находящаяся в операторных скобках после слова else. А именно, выведется 0.

Для выполнения повторяющихся действий используется оператор цикла while. Структура этого оператора имеет вид:  
while **(условие) {последовательность действий};,** где условие - это любое логическое выражение, истинность которого проверяется в начале каждой итерации (условие выполнения тела цикла). При истинности условия выполнится последовательность действий, находящаяся в операторных скобках и опять проверится условие. Так будет продолжаться до тех пор, пока результат логического выражения не окажется ложным. Тогда последовательность действий перестанет выполняться и программа перейдет к следующему за операторными скобками действию. Следует заметить, что если условие всегда будет выполняться, то программа зациклится – никогда не остановится. Ее нужно будет завершить принудительно.

{

i:=10;

while(i>=1)

{

i:=i-1;

print i;

};

}

Выполнение данной программы приведет к тому, что на экран выведется 10 цифр – от 9 до 0.

Следует отметить, что операторы цикла, как и условные могут быть вложенными:

{

i:=10;

while(i>=1)

{

i:=i-1;

print i;

j:=i;

while(j<10)

{

j:=j+1;

print j;

};

};

}

Таким образом, язык NL позволяет реализовать довольно объемные вычисления и алгоритмы с ветвящейся структурой.

**Описание модулей**

Разработанная программа состоит из семи модулей:

1. main.c – специальная функция, которая открывает файл и выполняет запуск процедур обработки.
2. init.c – инициализация, выполняет действия, которые нужно произвести до начала анализа (занесение в таблицу символов зарезервированных ключевых слов).
3. lexer.c – лексический анализатор. Выделяет из входной последовательности слово (токен).
4. parser.c – синтаксический анализатор. Обрабатывает слово, выделенное ранее лексическим анализатором.
5. symbol.c – модуль реализует взаимодействие с таблицей символов и кодов (добавить запись, найти номер строки)
6. emitter.c – выполнение программы, по таблице кодов, с использованием таблицы символов.
7. errors.c – обработка ошибок.

Для описания общих переменных и внешних функций используется заголовочный файл global.h

Далее будут приведены исходные коды описанных выше файлов.

**main.c**

#include "global.h"

FILE\* Open;

int main(int argc,char \*argv[])

{

char \*name=argv[1]; // Считать первый аргумент (после имени исполняемого файла) из командной строки

char \*a1=argv[2]; // Считать второй аргумент

char \*a2=argv[3]; // Считать третий аргумент

if((Open=fopen(name,"r"))==NULL) // Если не удалось открыть файл, то

{

printf("Can't open file test.txt\n"); // Вывод сообщения об ошибке

getch(); // Ждать нажатия на любую кнопку

exit(1); // Выход

}

init(); // Иницализация

parse(); // Разбор

emit(); // Выполнение

fclose(Open); //Закрыть файл

if (a1) // Если второй аргумент есть, то

{

printf("\nsymtable:\n"); // Вывод таблицы символов

get\_symtable();

}

if (a2) // Если третий аргумент есть, то

{

printf("\ncodetable:\n"); // Вывод таблицы кодов

get\_codetable();

}

printf("\nPress any key to exit.\n");

getch(); // Ждать нажатия на клавишу

return(0);

}

**init.c**

#include "global.h"

struct entry keywords[]=

{

"if",IF,0,

"else",ELSE,0,

"while",WHILE,0,

"input",INPUT,0,

"print",PRINT,0,

"printn",PRINTN,0,

0,0,0,

};

void init(void)/\* Загрузка ключевых слов в таблицу символов \*/

{

struct entry \*i;

for(i=keywords;i->token;i++)

insert(i->lexptr,i->token);

}

**lexer.c**

#include "global.h"

int CmpNextSym(int,int,int);

char lexbuf[BSIZE];

int lineno = 1;

double tokenval = NONE;

int lexan(void) /\* Лексический анализатор \*/

{

int t;

while (1)

{

t = fgetc(Open);// Считать в t символ из ранее открытого файла

if (t == ' ' || t == '\t'); /\* Отбрасываем разделители-пробелы \*/

else if (t == '\n')// Если символ перевода строки, то

lineno++;// Увеличить счетчик линий

else if (isdigit(t)) /\* t - цифра \*/

{

ungetc(t, Open);// Вернуть t во входной поток

fscanf(Open,"%lf",&tokenval);// Занести занчение числа (1 или > символов) в tokenval

return NUM;// Вернуть идентификатор числа

}

else if (isalpha(t)) /\* t - буква \*/

{

int p, b = 0;

while (isalnum(t)) /\* Пока t - буква или цифра \*/

{

lexbuf[b++] = t;// Добавить в буффер t

t=fgetc(Open);// Считать очередной символ

if (b >= BSIZE)// если превышен рзмер буффера

error("compiler error");// Вызов процедуры выхода с сообщением об ошибке

}

lexbuf[b] = EOS;// Добавить в буффер признак завершения последовательности символов (слова)

if (t != EOF)// Если t - не признак конца файла, то

ungetc(t, Open);// Вернуть t

p = lookup(lexbuf);// p = положение считанного слова в таблце символов

if (p == 0)// Если слово встретилось впервые

p = insert(lexbuf, ID);// Добавить в таблицу символов новую переменную

tokenval = p;

return symtable[p].token;// Вернуть соответствующий считанному слову идентификатор

}

else if (t == EOF) return DONE;// Если в t признак завершения файла - вернуть идентификатор конца программы

else switch(t)// Иначе, если t один из символов:

{

case '>':

return CmpNextSym('=',JG,JGE);// Если за t следует =, то вернуть инедтификатор условия JGE (больше-равно), иначе - JG (больше)

case '<':

return CmpNextSym('=',JL,JLE);// Меньше-равно, или меньше

case '!':

return CmpNextSym('=',JNE,'!');// Не равно, или символ !

case '=':

return CmpNextSym('=',JE,'=');// Равно (условие) или символ =

case ':':

return CmpNextSym('=',EQUAL,':');// Прсвоить или символ :

case '{':

return BEGIN;// Вернуть идентификатор BEGIN

case '}':

return END;// Вернуть идентификатор END

default:// Если что-либо другое, то

tokenval=NONE;

return t;// Вернуть t

}

}

}

int CmpNextSym(int ch,int good,int bad)

{

int nc=fgetc(Open);// считать следующий символ

if(nc==ch) return good;//если следующий символ = ch - вернуть good

ungetc(nc,Open);//иначе, возврат символа во входной поток

return bad;// вернуть bad

}

**parser.c**

#include "global.h"

int FillCode(void);

int FC(void);

int expr(void);

int term(void);

int factor(void);

int IdIn(void);

void LabelPush(int);

int LabelPop(void);

int match(int);

int lookahead;//текущий сканируемый входной токен

int LabelStack[100],LabelCnt=0;

double tv;

int parse(void)/\* Разбор и трансляция списка выражений \*/

{

lookahead=lexan();// Чтение слова

while(lookahead!=DONE)// До тех пор, пока не будет получен идентификатор завершения программы

{

FillCode();// Заполнение таблицы кодов

}

return 0;// Возврат

}

int FillCode()

{

int t;

FC();// Обработка слова

while(1)// Бесконечно повторять

{

switch(lookahead)

{

case ';'://Если текущее слово - символ ";", то

match(';');// Перейти к следующему слову

insertcode(0,";",0);// Добавить в таблицу клдлв ";"

FC();// Обработка слова

break;

default:

return;

}

}

}

int FC(void)

{

while(1)

{

switch(lookahead)

{

case ID:

match(ID);

tv=tokenval;

match(EQUAL);

expr();// Обработка выражения

insertcode(EQUAL,symtable[tv].lexptr,0);// Добавить в таблицу кодов присваивание переменной

break;

case PRINTN:

match(PRINTN);

expr();// Обработка выражения

insertcode(PRINTN,"printn",0);// Добавить строчку в таблицу кодов

while(1)

{

switch(lookahead)

{

case ',':// Обработкааргументов, введенных через запятую

match(',');

expr();

insertcode(PRINTN,"printn",0);

break;

default:

return;

}

}

case INPUT:

match(INPUT);

IdIn();// Добваить в таблицу кодов строчку получения заначения переменной

break;

case PRINT:

match(PRINT);

expr();

insertcode(PRINT,"print",0);

while(1)

{

switch(lookahead)

{

case ',':

match(',');

expr();

insertcode(PRINT,"print",0);

break;

default:

return;

}

}

case BEGIN:

match(BEGIN);

FillCode();// Вызов процедуры обработки слов (вначале проверится слово, затем ";")

match(END);

break;

case IF:

match(IF);

match('(');

expr();

insertcode(THEN,"then",0);

LabelPush(lastcode);// Добавить метку

match(')');

FC();

codetable[LabelPop()].value=lastcode+1;// Изменить занчение в строке метки, для указания на нужную строчку

insertcode(GOTO,"else",0);

LabelPush(lastcode);// Добавить метку

match(ELSE);

FC();

codetable[LabelPop()].value=lastcode;// Изменить занчение в строке метки, для указания на нужную строчку

break;

case WHILE:

insertcode(WHILE,"while",0);

LabelPush(lastcode);

match(WHILE);

match('(');

expr();

match(')');

insertcode(DO,"do",0);

LabelPush(lastcode);

FC();

codetable[LabelPop()].value=lastcode+1;

insertcode(GOTO,"goto",LabelPop());

break;

default:

return;

}

}

}

int expr(void)

{

term();// Ввызов вспомогательной процедуры разбора

while(1)

{

switch(lookahead)// Обработка операций + - и логических уловий

{

case '+':

match(lookahead);// Перейти к следующему слову

term();

insertcode('+',"+",0);

break;

case '-':

match(lookahead);

term();

insertcode('-',"-",0);

break;

case JNE:

match(lookahead);

term();

insertcode(JE,"!=",0);

break;

case JE:

match(lookahead);

term();

insertcode(JNE,"==",0);

break;

case JL:

match(lookahead);

term();

insertcode(JL,"<",0);

break;

case JLE:

match(lookahead);

term();

insertcode(JLE,"<=",0);

break;

case JG:

match(lookahead);

term();

insertcode(JG,">",0);

break;

case JGE:

match(lookahead);

term();

insertcode(JGE,">=",0);

break;

default:

return;

}

}

}

int term()

{

factor();// Ввызов вспомогательной процедуры разбора

while(1)

{

switch(lookahead)// Обработка матемтических операций типа \*, /

{

case '\*':

match(lookahead);

factor();

insertcode('\*',"\*",0);

break;

case '/':

match(lookahead);

factor();

insertcode('/',"/",0);

break;

default:

return;

}

}

}

int factor(void)

{

switch(lookahead)

{

case NUM:

insertcode(NUM,"num",tokenval);

match(NUM);

break;

case ID:

insertcode(ID,symtable[tokenval].lexptr,0);

match(ID);

break;

case '(':

match('(');

expr();

match(')');

break;

default:

error("syntax error");

}

return 0;

}

int IdIn()

{

insertcode(INPUT,symtable[tokenval].lexptr,0);

match(ID);

while(1)

{

switch(lookahead)

{

case ',':

match(',');

insertcode(INPUT,symtable[tokenval].lexptr,0);

match(ID);

break;

default:

return;

}

}

}

void LabelPush(int n)// Добавить метку в стек

{

LabelStack[LabelCnt++]=n;

}

int LabelPop(void)// Извлечь метку из стека

{

LabelCnt--;

return LabelStack[LabelCnt];

}

int match(int t)//процедура переходит к следующему входному токену, если ее аргумент t совпадает со сканируемым символом.

{

if(lookahead==t)// Если t совпадает со сканируемым словом, то

lookahead=lexan();// Считать следующее слово

else error("syntax error");// Иначе - вывод ошибки, выход

return 0;

}

**symbol.c**

#include "global.h"

#define STRMAX 1000 /\* Размер массива лексем \*/

#define SYMMAX 1000 /\* Размер таблицы символов \*/

#define CODEMAX 1000/\* Размер таблицы кодов \*/

char lexemes[STRMAX];

int lastchar = -1; /\* Последняя использованная позиция в lexemes \*/

struct entry symtable[SYMMAX];

int lastentry = 0; // Последняя использованная позиция в таблице символов

char lexgen[STRMAX];

int lastlexgen = -1;

struct code codetable[CODEMAX];

int lastcode = 0; /\* Последняя использованная позиция в таблице кодов \*/

int lookup(char s[]) /\* Возвращает положение в таблице символов для s \*/

{

int i;

for(i = lastentry; i > 0; i--)

if (strcmp(symtable[i].lexptr, s) == 0)

return i;

return 0;

}

int insert(char s[], int tok) /\* Добавить строчку в таблицу символов \*/

{

int len;

len = strlen(s); /\* strlen вычисляет длину строки \*/

if (lastentry + 1 >= SYMMAX)

error("symbol table full");

if (lastchar + len + 1 >= STRMAX)

error("lexemes array full");

lastentry++;

symtable[lastentry].token = tok;

symtable[lastentry].lexptr = &lexemes[lastchar + 1];

lastchar += len + 1;

strcpy(symtable[lastentry].lexptr, s);

return lastentry;

}

int insertcode(int tok,char s[],double value){ //Добавление в таблицу кодов

int len;

len=strlen(s);

if(lastcode+1>=CODEMAX)

error("code table full");

if(lastlexgen+len+1>=STRMAX)

error("lexemes array full");

lastcode++;

codetable[lastcode].value = value;

codetable[lastcode].token = tok;

codetable[lastcode].lexptr = &lexgen[lastlexgen+1];

lastlexgen+=len +1;

strcpy(codetable[lastcode].lexptr,s);

return lastcode;

}

int get\_codetable()// Вывести таблицу кодов

{

int i;

for(i=1;i<=lastcode;i++)

printf("%d %d %s %g\n",i,codetable[i].token,codetable[i].lexptr,codetable[i].value);

return 0;

}

int get\_symtable()// Вывести таблицу символов

{

int i;

for(i=1;i<=lastentry;i++)

printf("%d %d %s %g\n",i,symtable[i].token,symtable[i].lexptr,symtable[i].value);

return 0;

}

**emitter.c**

#include "global.h"

double stack[1000];

int j=0;

int a;

double pop(void);

void push(double n);

double x,y,z;

int emit(void) //Выполнение программы

{

int i=0;

while(i<lastcode)// Выполнять, пока не достигнут конец таблицы кодов

{

switch(codetable[i].token)// Выполнить действие, в соответствии с прочитанным словом из таблицы кодов

{

case NUM:

push(codetable[i].value);

break;

case ID:

push(symtable[lookup(codetable[i].lexptr)].value);

break;

case '+':

y=pop();

z=pop();

push(z+y);

break;

case '-':

y=pop();

z=pop();

push(z-y);

break;

case '\*':

y=pop();

z=pop();

push(z\*y);

break;

case '/':

y=pop();

z=pop();

push(z/y);

break;

case EQUAL:

symtable[lookup(codetable[i].lexptr)].value=pop();

break;

case JE:

y=pop();

z=pop();

push(y==z);

break;

case JNE:

y=pop();

z=pop();

push(y!=z);

break;

case JL:

y=pop();

z=pop();

push(y<z);

break;

case JLE:

y=pop();

z=pop();

push(y<=z);

break;

case JG:

y=pop();

z=pop();

push(y>z);

break;

case JGE:

y=pop();

z=pop();

push(y>=z);

break;

case DO:

if(pop()==1)i=codetable[i].value;// Если условие выполнено - перейти в соответствующую строку

break;

case GOTO:

i=codetable[i].value;

break;

case THEN:

if(pop()==1)i=codetable[i].value;

break;

case PRINT:

printf("%g \n",pop());

break;

case PRINTN:

a = pop();

printf("%d \n",a);

break;

case INPUT:

scanf("%lf,",&symtable[lookup(codetable[i].lexptr)].value);

break;

}

i++;

}

return 0;

}

void push(double n)//Положить в стек

{

stack[j++]=n;

}

double pop()//Извлечь из стека

{

j--;

return stack[j];

}

**errors.c**

#include"global.h"

int error(char \*s)

{

printf("%s in line %d\n",s,lineno); // вывод сообщения об ошибке в конкретной строке

getch();

exit(1); // выход

return 0;

}

**Список использованной литературы**

1. А. Ахо, Р. Сети, Д. Ульман Компиляторы: принципы, технологии и инструменты.: Пер. с англ. — М.: Издательский дом "Вильяме", 2003. — 768 с.
2. Липпман C++ для начинающих. – 1194 с.
3. Флоренсов А.Н. Операционные системы: Учеб. пос. - Омск: Издательство ОмГТУ, 2005. – 160 с.