Институт приборостроения, автоматизации и информационных технологий

Кафедра информационных систем и цифровых технологий

Дисциплина «Теория языков программирования и методы трансляции»

Отчет к лабораторной работе № 6

«Генерация кода и реализация машины»

Выполнил:

Василения Иван Валерьевич

Мельниченко Артём Олегович

Принял:

Гордиенко А. П.

Орёл, 2024г

Грамматика:

goal -> 'var' var\_decl\_list 'begin' comand\_list 'end' + '.'

var\_decl\_list -> var\_decl ';' var\_decl\_list

var\_decl\_list -> э

var\_decl -> 'identifier' ':' 'type'

comand\_list -> comand comand\_list

comand\_list -> э

comand -> assignment | read | write | if\_statement

assignment -> 'identifier' ':=' expr ';'

read -> 'read' '(' 'identifier' ')' ';'

write -> 'write' '(' out\_info ')' ';'

out\_info -> 'identifier'

t\_goal -> expr

expr -> term expr1

expr1 -> '+' term expr1

epxr1 -> '-' term expr1

expr1 -> э

term -> factor term1

term1 -> '\*' factor term1

term1 -> '/' factor term1

term1 -> э

factor -> 'number'

factor -> 'identifier'

factor -> '(' expr ')'

factor -> 'round' '(' expr ')'

if\_statement -> 'if' '(' condition ')' 'then' 'begin' comand\_list 'end' end\_of\_statement

end\_of\_statement -> 'else' 'begin' comand\_list 'end' | э

condition -> condition\_member condition\_sign condition\_member

condition\_member -> 'identifier' | 'number'

condition\_sign -> '>' | '<' | '='

Анализатор и генератор кода:  
  
#ifndef CODE\_GENERATOR\_H

#define CODE\_GENERATOR\_H

#include "stack.h"

#include <stdlib.h>

#include <stdio.h>

#include "variable\_table.h"

#include "comands.h"

void push\_constant(int constant);

stack\* stk;

variable\_table\* vt;

void generator\_init(){

    stk = stack\_alloc();

    vt = variable\_table\_alloc();

}

void run\_comands(){

    for(int i = 0; i < current\_size; i++){

        printf("cmd№ %d: ", i);

        int cmd = comand\_table[i].comand;

        char\* param1 = comand\_table[i].param1;

        char\* param2 = comand\_table[i].param2;

        switch(cmd){

            case PUSH\_CONST:

                push\_const(param1);

                break;

            case PUSH\_VAR:

                push\_var(param1);

                break;

            case ADD:

                comand\_add();

                break;

            case SUB:

                comand\_sub();

                break;

            case MUL:

                comand\_mul();

                break;

            case DIV:

                comand\_div();

                break;

            case ASSIGN:

                comand\_assign(param1);

                break;

            case READ:

                comand\_read(param1);

                break;

            case WRITE:

                comand\_write(param1);

                break;

            case COMPARISON:

                bool res;

                comand\_comparison(param1, &res);

                if (res) i++;

                break;

            case GOTO:

                i = atoi(param1) - 1;

                printf("GOTO %s\n", param1);

                break;

            default:

                printf("Unknown comand %d(%s, %s)\n", cmd, param1, param2);

                break;

        }

    }

}

void comand\_comparison(char\* sign, bool\* res){

    int p1 = pop\_const();

    int p2 = pop\_const();

    bool val;

    if(strcmp(sign, ">") == 0){

        val = p2>p1;

    }

    else if(strcmp(sign, "<") == 0){

        val = p2<p1;

    }

    else if(strcmp(sign, ">=") == 0){

        val = p2>=p1;

    }

    else if(strcmp(sign, "<=") == 0){

        val = p2<=p1;

    }

    else if(strcmp(sign, "=") == 0){

        val = p2==p1;

    }

    printf("comparison %d %s %d = %d;\n", p2, sign, p1, val);

    \*res = val;

}

void comand\_add(){

    int p1 = pop\_const();

    int p2 = pop\_const();

    stack\_push(stk, p1+p2);

    printf("push %d\n", p1+p2);

}

void comand\_mul(){

    int p1 = pop\_const();

    int p2 = pop\_const();

    stack\_push(stk, p1\*p2);

    printf("push %d\n", p1\*p2);

}

void comand\_div(){

    int p1 = pop\_const();

    int p2 = pop\_const();

    stack\_push(stk, p2/p1);

    printf("push %d\n", p2/p1);

}

void comand\_sub(){

    int p1 = pop\_const();

    int p2 = pop\_const();

    stack\_push(stk, p2-p1);

    printf("push %d\n", p2-p1);

}

void comand\_assign(char\* name){

    int val = stack\_pop(stk);

    update\_var(name, val);

    printf("assign %s = %d\n", name, val);

}

void comand\_read(char\* name){

    int val;

    printf("Enter integer: ");

    scanf("%d", &val);

    update\_var(name, val);

    printf("read %s = %d\n", name, val);

}

void comand\_write(char\* name){

    int val = get\_value\_i(vt, name);

    printf("write %s = %d\n", name, val);

}

void add\_var(char\* name){

    add\_value\_i(vt, name);

}

void update\_var(char\* name, int value){

    update\_value\_i(vt, name, value);

}

void push\_const(char\* constant){

    printf("push const %s\n", constant);

    int val = atoi(constant);

    stack\_push(stk, val);

}

void push\_var(char\* name){

    int val = get\_value\_i(vt, name);

    stack\_push(stk, val);

    printf("push var %d\n", val);

}

int pop\_const(){

    int val = stack\_pop(stk);

    printf("pop %d\n", val);

    return val;

}

void print\_st(){

    printf("\n========================\nSTK: ");

    print\_stack(stk);

    printf("\n");

    printf("COMANDS TABLE:\n");

    print\_comand\_table();

}

void print\_vt(){

    print\_variables(vt);

}

#endif

Реализация терминала if\_statement:

bool if\_statement() {

    if (curr\_token == IF) {

        eat(IF);

        eat(LBRACKET);

        condition();

        char\* toelse = (char\*)malloc(20);

        add\_comand\_goto(toelse);

        eat(RBRACKET);

        eat(THEN);

        eat(BEGIN\_P);

        comand\_list();

        char\* afterelse = (char\*)malloc(20);

        add\_comand\_goto(afterelse);

        eat(END);

        sprintf(toelse, "%d", get\_current\_table\_size());

        else\_statement();

        sprintf(afterelse, "%d", get\_current\_table\_size());

    }

    else {

        return false;

    }

    return true;

}