Министерство образования и науки Российской Федерации
Новосибирский государственный технический университет
Кафедра прикладной математики

Языки программирования и методы трансляции

Лабораторная работа №4

Факультет	прикладной математики и информатики
Группа	ПМ-01
Студенты	Александров М.Е.
	Жигалов П.С.
Преподаватели	Еланцева И.Л.
	Полетаева И.А.
Вариант	7

Новосибирск
2013

1. Цель работы

Изучить методы генерации кода с учетом различных промежуточных форм представления программы. Изучить методы управления памятью и особенности из использования на этапе генерации кода.

Научиться проектировать генератор кода.

2. Задание

Подмножество языка C++ включает:

- данные типа **int, float, массивы** из элементов указанных типов;
- инструкции описания переменных;
- операторы присваивания в любой последовательности;
- операции **+, – , *, = =, != , <, >** .

В соответствии с выбранным вариантом реализовать генератор кода. Исходными данными являются:

- синтаксическое дерево или постфиксная запись, построенные в лабораторной работе №3;
- таблицы лексем.

Результатом выполнения лабораторной работы является программа на языке Ассемблер, разработанная на основе знаний и практических навыков, полученных при изучении курса «Языки программирования и методы трансляции (часть I)».

В режиме отладки продемонстрировать работоспособность генератора кода и транслятора в целом.

3. Структура входных и выходных данных

Входные данные представляют собой имена файлов: файла для вывода ассемблерного кода, файла ошибок, а также полученные в результате работ №2-№3 таблицы и постфиксную запись. Результатом работы программы являются два файла – файл с ассемблерным кодом и файл ошибок.

4. Выражения языка С и их представление в Ассемблере (NASM)

Выражения С	Представление в Ассемблере	Примечание
int a = 0;	a: dd 0	Помещается в section .data
float b = 0.0;	b: dq 0.0	Помещается в section .data
int c[5];	c: resd 5	Помещается в section .bss
float d[3];	d: resq 3	Помещается в section .bss
a = b;	fld qword [b] fistp dword [a]	
b = a;	fild dword [a] fstp qword [b]	
c[2] = a;	const_50_0_42876T8GHP: dd 2 tmp_var_int_42876T8GHP: dd 0	Помещается в section .data
	fild dword [const_50_0_42876T8GHP] fistp dword [tmp_var_int_42876T8GHP] mov edx, [tmp_var_int_42876T8GHP] push edx fild dword [a] pop edx fistp dword [c+edx*4]	Здесь и далее 42876T8GHP и JIQATI3187– уникальные идентификаторы, формируется по постфиксной записи.
a = c[2];	//	Помещается в section .data
	fild dword [const_50_0_42876T8GHP] fistp dword [tmp_var_int_42876T8GHP] mov edx, [tmp_var_int_42876T8GHP] fild dword [c+edx*4] fistp dword [a]	
d[2] = b;	//	Помещается в section .data
	fild dword [const_50_0_42876T8GHP] fistp dword [tmp_var_int_42876T8GHP] mov edx, [tmp_var_int_42876T8GHP] push edx fld qword [b] pop edx fstp qword [d+edx*8]	
b = d[2];	//	Помещается в section .data
	fild dword [const_50_0_42876T8GHP] fistp dword [tmp_var_int_42876T8GHP] mov edx, [tmp_var_int_42876T8GHP] fld qword [d+edx*8] fstp qword [b]	

Выражения C	Представление в Ассемблере	Примечание
+	fadd	
-	fsub	В прямом порядке
	fsubr	В обратном порядке
*	fmul	
< > == !=	tmp_var_int_JIQATI3187: dd 0	Помещается в section .data
	fcomp fistp dword [tmp_var_int_JIQATI3187] fstsw ax sahf jXX lbl_YY_ZZ_JIQATI3187 fldz jmp lbl_YY_ex_JIQATI3187 lbl_YY_ZZ_JIQATI3187: fld1 lbl_YY_ex_JIQATI3187:	jXX: == je != jne < ja > jb YY – уникальный индекс для каждой операции ZZ: == eq != ne < lt > gt

5. Тесты

5.1. Целочисленные переменные и массивы, операции над ними

Код на C:

```
void main()
{
    int a=2, b=-3, c[3], d=9, e=7, f, g, h;
    c[0] = 3;
    c[1] = c[0] + a;
    c[2] = c[1] - c[0];
    f = a - b;
    f += 1;
    g = a * 8 - c[1];
    h = b * 2 - (a > b);
}
```

Код на Ассемблере:

```
section .data
        a:      dd      2
        b:      dd      -3
        d:      dd      9
        e:      dd      7
        const_48_0_CVCEHPMF6N:  dd      0
        const_51_0_CVCEHPMF6N:  dd      3
        const_49_0_CVCEHPMF6N:  dd      1
        const_50_0_CVCEHPMF6N:  dd      2
        f:      dd      0
        g:      dd      0
        const_56_0_CVCEHPMF6N:  dd      8
        h:      dd      0
        tmp_var_int_CVCEHPMF6N: dd      0
section .bss
        c:      resd    3
section .text
        global main
main:
        finit
        fild    dword [const_48_0_CVCEHPMF6N]
        fistp   dword [tmp_var_int_CVCEHPMF6N]
        mov     edx, [tmp_var_int_CVCEHPMF6N]
        push    edx
        fild    dword [const_51_0_CVCEHPMF6N]
        pop     edx
        fistp   dword [c+edx*4]
        finit
        fild    dword [const_49_0_CVCEHPMF6N]
        fistp   dword [tmp_var_int_CVCEHPMF6N]
        mov     edx, [tmp_var_int_CVCEHPMF6N]
        push    edx
        fild    dword [const_48_0_CVCEHPMF6N]
        fistp   dword [tmp_var_int_CVCEHPMF6N]
        mov     edx, [tmp_var_int_CVCEHPMF6N]
        fild    dword [c+edx*4]
        fild    dword [a]
        fadd
        pop     edx
        fistp   dword [c+edx*4]
        finit
        fild    dword [const_50_0_CVCEHPMF6N]
```

```
        fistp   dword [tmp_var_int_CVCEHPMF6N]
        mov     edx, [tmp_var_int_CVCEHPMF6N]
        push    edx
        fild    dword [const_49_0_CVCEHPMF6N]
        fistp   dword [tmp_var_int_CVCEHPMF6N]
        mov     edx, [tmp_var_int_CVCEHPMF6N]
        fild    dword [c+edx*4]
        fild    dword [const_48_0_CVCEHPMF6N]
        fistp   dword [tmp_var_int_CVCEHPMF6N]
        mov     edx, [tmp_var_int_CVCEHPMF6N]
        fild    dword [c+edx*4]
        fsub
        pop     edx
        fistp   dword [c+edx*4]
        finit
        fild    dword [a]
        fild    dword [b]
        fsub
        fistp   dword [f]
        finit
        fild    dword [f]
        fild    dword [const_49_0_CVCEHPMF6N]
        fadd
        fistp   dword [f]
        finit
        fild    dword [a]
        fild    dword [const_56_0_CVCEHPMF6N]
        fmul
        fild    dword [const_49_0_CVCEHPMF6N]
        fistp   dword [tmp_var_int_CVCEHPMF6N]
        mov     edx, [tmp_var_int_CVCEHPMF6N]
        fild    dword [c+edx*4]
        fsub
        fistp   dword [g]
        finit
        fild    dword [b]
        fild    dword [const_50_0_CVCEHPMF6N]
        fmul
        fild    dword [a]
        fild    dword [b]
        fcomp
        fistp   dword [tmp_var_int_CVCEHPMF6N]
        fstsw   ax
        sahf
        jb lbl_gt_70_CVCEHPMF6N
        fldz
        jmp lbl_ex_70_CVCEHPMF6N
lbl_gt_70_CVCEHPMF6N:
        fld1
lbl_ex_70_CVCEHPMF6N:
        fsub
        fistp   dword [h]
        mov     eax, 0
        ret
```

Значения переменных после выполнения программы:

```
a = 2
b = -3
d = 9
e = 7
c[0] = 3
c[1] = 5
c[2] = 2
f = 6
g = 11
h = -7
```

5.2. Целые и вещественные переменные и массивы, операции над ними, приведение типов

Код на C:

```
void main()
{
    int a = 5, q[1], g = 3.4;
    float b = 3.5, c[4], d[2];
    c[0] = g + 1;
    c[1] = b + 2;
    c[3] = c[0] == c[0];
    q[0] = b;
```

```
    d[1] = q[0];
    d[0] *= 3.0;
}
```

Код на Ассемблере:

```
section .data
        a:      dd      5
        g:      dd      3
        b:      dq      3.5
        const_48_0_4JK3QSI9Q1:  dd      0
        const_49_0_4JK3QSI9Q1:  dd      1
        const_50_1_4JK3QSI9Q1:  dd      2
        const_51_0_4JK3QSI9Q1:  dd      3
        const_45_0_4JK3QSI9Q1:  dq      3.0
        tmp_var_int_4JK3QSI9Q1: dd      0
section .bss
        c:      resq    4
        q:      resd    1
        d:      resq    2
section .text
        global main
main:
        finit
        fild    dword [const_48_0_4JK3QSI9Q1]
        fistp   dword [tmp_var_int_4JK3QSI9Q1]
        mov     edx, [tmp_var_int_4JK3QSI9Q1]
        push    edx
        fild    dword [g]
        fild    dword [const_49_0_4JK3QSI9Q1]
        fadd
        pop     edx
        fstp    qword [c+edx*8]
        finit
        fild    dword [const_49_0_4JK3QSI9Q1]
        fistp   dword [tmp_var_int_4JK3QSI9Q1]
        mov     edx, [tmp_var_int_4JK3QSI9Q1]
        push    edx
        fld     qword [b]
        fild    dword [const_50_1_4JK3QSI9Q1]
        fadd
        pop     edx
        fstp    qword [c+edx*8]
        finit
        fild    dword [const_51_0_4JK3QSI9Q1]
        fistp   dword [tmp_var_int_4JK3QSI9Q1]
        mov     edx, [tmp_var_int_4JK3QSI9Q1]
        push    edx
        fild    dword [const_48_0_4JK3QSI9Q1]
        fistp   dword [tmp_var_int_4JK3QSI9Q1]
        mov     edx, [tmp_var_int_4JK3QSI9Q1]
        fld     qword [c+edx*8]
        fild    dword [const_48_0_4JK3QSI9Q1]
        fistp   dword [tmp_var_int_4JK3QSI9Q1]
        mov     edx, [tmp_var_int_4JK3QSI9Q1]
        fld     qword [c+edx*8]
        fcomp
        fistp   dword [tmp_var_int_4JK3QSI9Q1]
        fstsw   ax
        sahf
        je lbl_eq_37_4JK3QSI9Q1
        fldz
        jmp lbl_ex_37_4JK3QSI9Q1
lbl_eq_37_4JK3QSI9Q1:
        fld1
lbl_ex_37_4JK3QSI9Q1:
        pop     edx
        fstp    qword [c+edx*8]
        finit
        fild    dword [const_48_0_4JK3QSI9Q1]
        fistp   dword [tmp_var_int_4JK3QSI9Q1]
        mov     edx, [tmp_var_int_4JK3QSI9Q1]
        push    edx
        fld     qword [b]
        pop     edx
        fistp   dword [q+edx*4]
        finit
        fild    dword [const_49_0_4JK3QSI9Q1]
        fistp   dword [tmp_var_int_4JK3QSI9Q1]
        mov     edx, [tmp_var_int_4JK3QSI9Q1]
        push    edx
```

```
        fild    dword [const_48_0_4JK3QSI9Q1]
        fistp   dword [tmp_var_int_4JK3QSI9Q1]
        mov     edx, [tmp_var_int_4JK3QSI9Q1]
        fild    dword [q+edx*4]
        pop     edx
        fstp    qword [d+edx*8]
        finit
        fild    dword [const_48_0_4JK3QSI9Q1]
        fistp   dword [tmp_var_int_4JK3QSI9Q1]
        mov     edx, [tmp_var_int_4JK3QSI9Q1]
        push    edx
        fld     qword [const_45_0_4JK3QSI9Q1]
        pop     edx
        fld     qword [d+edx*8]
        fmul
        fstp    qword [d+edx*8]
        mov     eax, 0
        ret
```

Значения переменных после выполнения программы:

```
a = 5
g = 3
b = 3.500000
c[0] = 4.000000
c[1] = 5.500000
c[2] = 0.000000
c[3] = 1.000000
q[0] = 4
d[0] = 0.000000
d[1] = 4.000000
```

5.3. Неинициализированная переменная

Код на C:

```
void main()
{
    int ar = 2, bk = 7, kbn;
    ar += bk;
    ar = bk + kbn;
}
```

Файл ошибок:

```
Code error: Variable "kbn" is not initialized!
```

5.4. Сложное выражение с множественными операторами сравнения

Код на C:

```
void main()
{
    int a = 5;
    float b = 3.7, c = 0;
    c -= (a > 4 < 2 == 1 != b) * 2 < b;
}
```

Код на Ассемблере:

```
section .data
        a:      dd      5
        b:      dq      3.7
        c:      dq      0
        const_52_1_4UHFE02VO7:  dd      4
        const_50_0_4UHFE02VO7:  dd      2
        const_49_0_4UHFE02VO7:  dd      1
        tmp_var_int_4UHFE02VO7: dd      0
section .text
        global main
main:
        finit
        fild    dword [a]
        fild    dword [const_52_1_4UHFE02VO7]
        fcomp
        fistp   dword [tmp_var_int_4UHFE02VO7]
        fstsw   ax
        sahf
        jb lbl_gt_15_4UHFE02VO7
```

```
        fldz
        jmp lbl_ex_15_4UHFE02VO7
lbl_gt_15_4UHFE02VO7:
        fld1
lbl_ex_15_4UHFE02VO7:
        fild    dword [const_50_0_4UHFE02VO7]
        fcomp
        fistp   dword [tmp_var_int_4UHFE02VO7]
        fstsw   ax
        sahf
        ja lbl_lt_17_4UHFE02VO7
        fldz
        jmp lbl_ex_17_4UHFE02VO7
lbl_lt_17_4UHFE02VO7:
        fld1
lbl_ex_17_4UHFE02VO7:
        fild    dword [const_49_0_4UHFE02VO7]
        fcomp
        fistp   dword [tmp_var_int_4UHFE02VO7]
        fstsw   ax
        sahf
        je lbl_eq_19_4UHFE02VO7
        fldz
        jmp lbl_ex_19_4UHFE02VO7
lbl_eq_19_4UHFE02VO7:
        fld1
lbl_ex_19_4UHFE02VO7:
        fld     qword [b]
        fcomp
        fistp   dword [tmp_var_int_4UHFE02VO7]
        fstsw   ax
        sahf
        jne lbl_ne_21_4UHFE02VO7
        fldz
        jmp lbl_ex_21_4UHFE02VO7
lbl_ne_21_4UHFE02VO7:
        fld1
lbl_ex_21_4UHFE02VO7:
        fild    dword [const_50_0_4UHFE02VO7]
        fmul
        fld     qword [b]
        fcomp
        fistp   dword [tmp_var_int_4UHFE02VO7]
        fstsw   ax
        sahf
        ja lbl_lt_25_4UHFE02VO7
        fldz
        jmp lbl_ex_25_4UHFE02VO7
lbl_lt_25_4UHFE02VO7:
        fld1
lbl_ex_25_4UHFE02VO7:
        fld     qword [c]
        fsubr
        fstp    qword [c]
        mov     eax, 0
        ret
```

Значения переменных после выполнения программы:

```
a = 5
b = 3.700000
c = -1.000000
```

6. Код программы

translator.h

```
#ifndef TRANSLATOR_H_INCLUDED
#define TRANSLATOR_H_INCLUDED
#include <iostream>
#include <sstream>
#include <fstream>
#include <string>
#include <stack>
#include <deque>
#include <vector>
#include <locale>
#include <cmath>
#include "table_const.h"
#include "table_var.h"
```

```
#include "lexeme.h"
#include "token.h"
using namespace std;

class translator
{
private:
    // Постоянные таблицы
    table_const<char> letters;      // 0
    table_const<char> numbers;      // 1
    table_const<string> operations; // 2
    table_const<string> keywords;   // 3
    table_const<char> separators;   // 4
    // Переменные таблицы
    table_var identifiers;          // 5
    table_var constants;            // 6
    // Файловые потоки
    ifstream in_source;
    ofstream out_token;
    ofstream out_error;
    // Анализ строки
    bool analyze_lexical_string(string str);
    // Удаление комментариев
    bool analyze_lexical_decomment(string& str, bool is_changed);
    // Счетчики для подробных сообщений об ошибке
    int analyze_lexical_strnum, analyze_lexical_strinc;
    // Удаление пробелов
    static inline void ltrim(string& out_)
    {
        int notwhite = out_.find_first_not_of(" \t\n");
        out_.erase(0, notwhite);
    }
    static inline void rtrim(string& out_)
    {
        int notwhite = out_.find_last_not_of(" \t\n");
        out_.erase(notwhite + 1);
    }
    static inline void trim(string& out_)
    {
        ltrim(out_);
        rtrim(out_);
    }
    /** Синтаксический анализ    */
    // Определяем какая строка содержится в токене
    string get_token_text(token get_t);
    // Структура элемент таблицы разбора
    struct table_parse_elem
    {
        vector<string> terminal; // Терминалы
        int jump;                // Переход
        bool accept;             // Принимать или нет
        bool stack_;             // Класть в стек или нет
        bool return_;            // Возвращать или нет
        bool error;              // Может ли быть ошибка
    };
    // Таблица разбора
    vector<table_parse_elem> table_parse;
    // Структура элемент постфиксной записи
    class postfix_elem
    {
    public:
        string id;
        short int type;
        short int table;
        postfix_elem()
        {
            id = "", type = 0, table = -1;
        }
        postfix_elem(string id_, int type_, int table_)
        {
            id = id_, type = type_, table = table_;
        }
        postfix_elem(string id_, int type_)
        {
            id = id_, type = type_, table = -1;
        }
        postfix_elem(string id_)
        {
            id = id_, type = 1, table = -1;
        }
        friend bool operator == (const postfix_elem& f, const postfix_elem& l)
        {
            if(f.type == l.type && f.table == l.table && f.id == l.id) return true;
            return false;
        }
        friend ostream& operator << (ostream& ostream_, const postfix_elem& pe_)
        {
```

```
            ostream_ << pe_.id;
            return ostream_;
        }
    };
    // Сравнение приоритетов операций
    bool priority_le(string what, string with_what);
    // Постфиксная запись
    vector<postfix_elem> postfix_record;
    // Построение постфиксной записи
    bool make_postfix(vector<token> t);
    // Построение постфиксной записи (локально)
    bool make_postfix(vector<token> t, vector<postfix_elem>& postfix_tmp);
    /** Упрощение выражений с константами, РГЗ */
    // Свертка констант
    void constant_folding(vector<postfix_elem>& postfix_tmp);
    // Частные случаи
    bool special_case(postfix_elem oper1p, postfix_elem oper2p, postfix_elem operation, postfix_elem& result);
    /** Генерация кода на Ассемблере */
    // Получения хэша для служебных переменных и констант
    string calc_salt(int length);
public:
    // Конструктор со вводом постоянных таблиц
    translator();
    // Отладочный вывод таблиц
    void debug_print(ostream& stream);
    // Лексический анализ
    bool analyze_lexical(string file_source, string file_tokens, string file_error);
    // Синтаксический анализ
    bool analyze_syntactical(string file_tokens, string file_error);
    // Печать постфиксной записи в файл и на экран
    void postfix_print(string file_tree);
    // Генерация кода
    bool generation_code(string file_asm, string file_error, bool need_printf, bool need_salt);
};

#endif // TRANSLATOR_H_INCLUDED
```

transl_codegen.cpp

```
#include "translator.h"
/** ================ Генерация кода на Ассемблере ================ */

// Получения хэша для служебных переменных и констант
string translator::calc_salt(int length)
{
    string postfix_str = "";
    for(int i = 0; i < (int)postfix_record.size(); i++)
        postfix_str.append(postfix_record[i].id);
    locale loc;
    const collate<char>& coll = use_facet<collate<char> >(loc);
    unsigned long salt_long = coll.hash(postfix_str.data(), postfix_str.data() + postfix_str.length());
    unsigned long salt_long_orig = salt_long;
    unsigned long salt_long_new = salt_long >> 5;
    stringstream a;
    while((int)a.str().length() < length)
    {
        unsigned char c = (salt_long - (salt_long_new << 5));
        if(c < 10) c += '0';
        else if(c < 36) c += 'A' - 10;
        else c = '_';
        a << c;
        salt_long = salt_long_new;
        salt_long_new = salt_long_new >> 5;
        if(salt_long_new == 0)
        {
            salt_long_orig = salt_long_orig >> 1;
            salt_long = salt_long_orig;
            salt_long_new = salt_long >> 5;
        }
    }
    return a.str();
}

// Генерация кода
bool translator::generation_code(string file_asm, string file_error, bool need_printf, bool need_salt)
{
    ofstream out_code(file_asm.c_str(), ios::trunc);
    out_error.open(file_error.c_str(), ios::trunc);
    string salt;
    if(need_salt) salt = calc_salt(10);
    else salt = "";
    bool need_adv_int = false;
    stack<postfix_elem> parse_stack;
    vector<postfix_elem> variables;
    vector<string> values;
    stringstream outcode;
```

```
    int index = 0;
    bool local_error = false;

    while(!local_error && index < (int)postfix_record.size())
    {
        stack<postfix_elem> array_stack;
        bool array_assign_here;
        lexeme lex_array_assign;
        identifiers.get_lexeme(postfix_record[index].id, lex_array_assign);
        if(lex_array_assign.dimension > 0)
            array_assign_here = true;
        else
            array_assign_here = false;
        bool array_assign_is_accepted = false;
        bool maybe_uninit_flag = false;
        string maybe_uninit_name = "";

        if(!(postfix_record[index+2].id == "=" && postfix_record[index+1].table == 6))
            outcode << "\tfinit\n";

        int i;
        for(i = index; !local_error && i < (int)postfix_record.size() && postfix_record[i].id != ";"; i++)
        {
            if(postfix_record[i].table == 5 || postfix_record[i].table == 6)
            {
                parse_stack.push(postfix_record[i]);
                bool added = false;
                for(int j = 0; !added && j < (int)variables.size(); j++)
                {
                    if(variables[j] == postfix_record[i])
                        added = true;
                }
                if(!added)
                {
                    variables.push_back(postfix_record[i]);
                    values.push_back("");
                }
                lexeme lex_array_check;
                identifiers.get_lexeme(postfix_record[i].id, lex_array_check);
                if(lex_array_check.dimension > 0)
                    array_stack.push(postfix_record[i]);

                if(postfix_record[i].table == 5 && lex_array_check.dimension == 0 && !lex_array_check.is_init[0])
                {
                    if(i != index)
                    {
                        cerr << "Code error: Variable \"" << postfix_record[i].id << "\" is not initialized!" << endl;
                        out_error << "Code error: Variable \"" << postfix_record[i].id << "\" is not initialized!" <<
endl;
                        out_code.close();
                        out_error.close();
                        return false;
                    }
                    else
                    {
                        maybe_uninit_flag = true;
                        maybe_uninit_name = postfix_record[i].id;
                    }
                }
            }
            else
            {
                postfix_elem oper1p, oper2p;
                int type1 = 0, type2 = 0;
                lexeme lex;

                oper2p = parse_stack.top();
                parse_stack.pop();
                oper1p = parse_stack.top();
                parse_stack.pop();

                if(oper1p.table == 5)
                {
                    identifiers.get_lexeme(oper1p.id, lex);
                    type1 = lex.type;
                    if(postfix_record[i].id != "=" && postfix_record[i].id != "[]")
                    {
                        if(type1 == 2)
                            outcode << "\tfld\t\tqword [" << oper1p.id << "]\n";
                        else
                            outcode << "\tfild\tdword [" << oper1p.id << "]\n";
                    }
                }
                else if(oper1p.table == 6)
                {
                    constants.get_lexeme(oper1p.id, lex);
                    type1 = lex.type;
```

```
    int hash, chain;
    constants.get_location(oper1p.id, hash, chain);
    if(postfix_record[i].id != "=")
    {
        if(type1 == 2)
            outcode << "\tfld\t\tqword [const_" << hash << "_" << chain << "_" << salt << "]\n";
        else
            outcode << "\tfild\tdword [const_" << hash << "_" << chain << "_" << salt << "]\n";
    }
}

if(oper2p.table == 5)
{
    identifiers.get_lexeme(oper2p.id, lex);
    type2 = lex.type;
    if(type2 == 2)
        outcode << "\tfld\t\tqword [" << oper2p.id << "]\n";
    else
        outcode << "\tfild\tdword [" << oper2p.id << "]\n";
}
else if(oper2p.table == 6)
{
    constants.get_lexeme(oper2p.id, lex);
    type2 = lex.type;
    int hash, chain;
    constants.get_location(oper2p.id, hash, chain);
    if(type2 == 2)
        outcode << "\tfld\t\tqword [const_" << hash << "_" << chain << "_" << salt << "]\n";
    else
        outcode << "\tfild\tdword [const_" << hash << "_" << chain << "_" << salt << "]\n";
}

if(postfix_record[i].id == "+")
    outcode << "\tfadd\n";
else if(postfix_record[i].id == "-")
{
    if(oper2p.id == "last" && oper1p.id != "last")
        outcode << "\tfsubr\n";
    else
        outcode << "\tfsub\n";
}
else if(postfix_record[i].id == "*")
    outcode << "\tfmul\n";
else if(postfix_record[i].id == "==")
{
    outcode << "\tfcomp\n";
    outcode << "\tfistp\tdword [tmp_var_int_" << salt << "]\n";
    outcode << "\tfstsw\tax\n\tsahf\n";
    outcode << "\tje lbl_eq_" << i << "_" << salt << "\n";
    outcode << "\tfldz\n\tjmp lbl_ex_" << i << "_" << salt << "\n";
    outcode << "lbl_eq_" << i << "_" << salt << ":\n\tfld1\n";
    outcode << "lbl_ex_" << i << "_" << salt << ":\n";
    need_adv_int = true;
}
else if(postfix_record[i].id == "!=")
{
    outcode << "\tfcomp\n";
    outcode << "\tfistp\tdword [tmp_var_int_" << salt << "]\n";
    outcode << "\tfstsw\tax\n\tsahf\n";
    outcode << "\tjne lbl_ne_" << i << "_" << salt << "\n";
    outcode << "\tfldz\n\tjmp lbl_ex_" << i << "_" << salt << "\n";
    outcode << "lbl_ne_" << i << "_" << salt << ":\n\tfld1\n";
    outcode << "lbl_ex_" << i << "_" << salt << ":\n";
    need_adv_int = true;
}
else if(postfix_record[i].id == ">")
{
    outcode << "\tfcomp\n";
    outcode << "\tfistp\tdword [tmp_var_int_" << salt << "]\n";
    outcode << "\tfstsw\tax\n\tsahf\n";
    if(oper2p.id == "last" && oper1p.id != "last")
        outcode << "\tja lbl_gt_" << i << "_" << salt << "\n";
    else
        outcode << "\tjb lbl_gt_" << i << "_" << salt << "\n";
    outcode << "\tfldz\n\tjmp lbl_ex_" << i << "_" << salt << "\n";
    outcode << "lbl_gt_" << i << "_" << salt << ":\n\tfld1\n";
    outcode << "lbl_ex_" << i << "_" << salt << ":\n";
    need_adv_int = true;
}
else if(postfix_record[i].id == "<")
{
    outcode << "\tfcomp\n";
    outcode << "\tfistp\tdword [tmp_var_int_" << salt << "]\n";
    outcode << "\tfstsw\tax\n\tsahf\n";
    if(oper2p.id == "last" && oper1p.id != "last")
        outcode << "\tjb lbl_lt_" << i << "_" << salt << "\n";
    else
```

```
            outcode << "\tja lbl_lt_" << i << "_" << salt << "\n";
        outcode << "\tfldz\n\tjmp lbl_ex_" << i << "_" << salt << "\n";
        outcode << "lbl_lt_" << i << "_" << salt << ":\n\tfld1\n";
        outcode << "lbl_ex_" << i << "_" << salt << ":\n";
        need_adv_int = true;
    }
    else if(postfix_record[i].id == "=")
    {
        if(!array_assign_is_accepted)
        {
            if(type1 == 2)
                outcode << "\tfstp\tqword [" << oper1p.id << "]\n";
            else
                outcode << "\tfistp\tdword [" << oper1p.id << "]\n";
            identifiers.set_is_init(postfix_record[index].id, true);
        }
        else
        {
            outcode << "\tpop\t\tedx\n";
            if(lex_array_assign.type == 2)
                outcode << "\tfstp\tqword [" << lex_array_assign.name << "+edx*8]\n";
            else
                outcode << "\tfistp\tdword [" << lex_array_assign.name << "+edx*4]\n";
        }
    }
    else if(postfix_record[i].id == "+=")
    {
        if(maybe_uninit_flag)
        {
            cerr << "Code error: Variable \"" << maybe_uninit_name << "\" is not initialized!" << endl;
            out_error << "Code error: Variable \"" << maybe_uninit_name << "\" is not initialized!" << endl;
            out_code.close();
            out_error.close();
            return false;
        }
        if(!array_assign_is_accepted)
        {
            outcode << "\tfadd\n";
            if(type1 == 2)
                outcode << "\tfstp\tqword [" << oper1p.id << "]\n";
            else
                outcode << "\tfistp\tdword [" << oper1p.id << "]\n";
        }
        else
        {
            outcode << "\tpop\t\tedx\n";
            if(lex_array_assign.type == 2)
            {
                outcode << "\tfld\t\tqword [" << lex_array_assign.name << "+edx*8]\n";
                outcode << "\tfadd\n";
                outcode << "\tfstp\tqword [" << lex_array_assign.name << "+edx*8]\n";
            }
            else
            {
                outcode << "\tfild\tdword [" << lex_array_assign.name << "+edx*4]\n";
                outcode << "\tfadd\n";
                outcode << "\tfistp\tdword [" << lex_array_assign.name << "+edx*4]\n";
            }
        }
    }
    else if(postfix_record[i].id == "*=")
    {
        if(maybe_uninit_flag)
        {
            cerr << "Code error: Variable \"" << maybe_uninit_name << "\" is not initialized!" << endl;
            out_error << "Code error: Variable \"" << maybe_uninit_name << "\" is not initialized!" << endl;
            out_code.close();
            out_error.close();
            return false;
        }
        if(!array_assign_is_accepted)
        {
            outcode << "\tfmul\n";
            if(type1 == 2)
                outcode << "\tfstp\tqword [" << oper1p.id << "]\n";
            else
                outcode << "\tfistp\tdword [" << oper1p.id << "]\n";
        }
        else
        {
            outcode << "\tpop\t\tedx\n";
            if(lex_array_assign.type == 2)
            {
                outcode << "\tfld\t\tqword [" << lex_array_assign.name << "+edx*8]\n";
                outcode << "\tfmul\n";
                outcode << "\tfstp\tqword [" << lex_array_assign.name << "+edx*8]\n";
            }
```

```
                        else
                        {
                            outcode << "\tfild\tdword [" << lex_array_assign.name << "+edx*4]\n";
                            outcode << "\tfmul\n";
                            outcode << "\tfistp\tdword [" << lex_array_assign.name << "+edx*4]\n";
                        }
                    }
                }
                else if(postfix_record[i].id == "-=")
                {
                    if(maybe_uninit_flag)
                    {
                        cerr << "Code error: Variable \"" << maybe_uninit_name << "\" is not initialized!" << endl;
                        out_error << "Code error: Variable \"" << maybe_uninit_name << "\" is not initialized!" << endl;
                        out_code.close();
                        out_error.close();
                        return false;
                    }
                    if(!array_assign_is_accepted)
                    {
                        if(oper2p.id == "last" && oper1p.id != "last")
                            outcode << "\tfsubr\n";
                        else
                            outcode << "\tfsub\n";
                        if(type1 == 2)
                            outcode << "\tfstp\tqword [" << oper1p.id << "]\n";
                        else
                            outcode << "\tfistp\tdword [" << oper1p.id << "]\n";
                    }
                    else
                    {
                        outcode << "\tpop\t\tedx\n";
                        if(lex_array_assign.type == 2)
                        {
                            outcode << "\tfld\t\tqword [" << lex_array_assign.name << "+edx*8]\n";
                            outcode << "\tfsubr\n";
                            outcode << "\tfstp\tqword [" << lex_array_assign.name << "+edx*8]\n";
                        }
                        else
                        {
                            outcode << "\tfild\tdword [" << lex_array_assign.name << "+edx*4]\n";
                            outcode << "\tfsubr\n";
                            outcode << "\tfistp\tdword [" << lex_array_assign.name << "+edx*4]\n";
                        }
                    }
                }
                else if(postfix_record[i].id == "[]")
                {
                    array_stack.pop();
                    bool is_assign = false;
                    if(array_assign_here && !array_assign_is_accepted && array_stack.size() == 0)
                    {
                        array_assign_is_accepted = true;
                        is_assign = true;
                    }
                    need_adv_int = true;
                    outcode << "\tfistp\tdword [tmp_var_int_" << salt << "]\n";
                    outcode << "\tmov\t\tedx, [tmp_var_int_" << salt << "]\n";
                    if(!is_assign)
                    {
                        if(type1 == 2)
                            outcode << "\tfld\t\tqword [" << oper1p.id << "+edx*8]\n";
                        else
                            outcode << "\tfild\tdword [" << oper1p.id << "+edx*4]\n";
                    }
                    else
                        outcode << "\tpush\tedx\n";
                }
                parse_stack.push(postfix_elem("last"));
            }

            if(i == index && postfix_record[i+2].id == "=" && postfix_record[i+1].table == 6)
            {
                lexeme lex_init;
                if(identifiers.get_lexeme(postfix_record[i].id, lex_init) && lex_init.dimension == 0 &&
!lex_init.is_init[0])
                {
                    bool found = false;
                    int j;
                    for(j = 0; !found && j < (int)variables.size(); j++)
                    {
                        if(variables[j].id == postfix_record[i].id)
                            found = true;
                    }
                    if(found)
                    {
                        identifiers.set_is_init(postfix_record[i].id, true);
```

```
                        values[j-1] = postfix_record[i+1].id;
                        i += 2;
                    }
                }
            }
        }

        while(parse_stack.size() > 0)
            parse_stack.pop();
        index = i + 1;
    }

    stringstream printf_out;
    if(need_printf)
        out_code << "extern printf\n";

    stringstream bss_out;
    bool need_bss = false;

    out_code << "section .data\n";
    for(int i = 0; i < (int)variables.size(); i++)
    {
        lexeme lex;
        if(variables[i].table == 5)
        {
            identifiers.get_lexeme(variables[i].id, lex);
            if(lex.type == 2)
            {
                if(lex.dimension == 0)
                {
                    out_code << "\t" << variables[i].id << ":\tdq\t";
                    if(values[i] == "")
                        out_code << "0.0\n";
                    else
                    {
                        if(values[i].find(".") == string::npos)
                        {
                            stringstream a_type_force;
                            a_type_force << values[i];
                            int f_type_force;
                            a_type_force >> f_type_force;
                            out_code << (float)f_type_force << "\n";
                        }
                        else
                            out_code << values[i] << "\n";
                    }
                    if(need_printf)
                    {
                        out_code << "\tmsg_" << variables[i].id << "_" << salt << ":\tdb\t\"" << variables[i].id <<
"\",0\n";
                        printf_out << "\tpush\tdword msg_" << variables[i].id << "_" << salt << "\n";
                        printf_out << "\tpush\tdword printf_str_" << salt << "\n";
                        printf_out << "\tcall\tprintf\n\tadd\t\tesp, 8\n";
                        printf_out << "\tpush\tdword [" << variables[i].id << "+4]\n";
                        printf_out << "\tpush\tdword [" << variables[i].id << "]\n";
                        printf_out << "\tpush\tdword printf_flt_" << salt << "\n";
                        printf_out << "\tcall\tprintf\n\tadd\t\tesp, 12\n";
                    }
                }
                else
                {
                    need_bss = true;
                    bss_out << "\t" << variables[i].id << ":\tresq\t" << lex.dimension << "\n";
                    if(need_printf)
                    {
                        out_code << "\tmsg_" << variables[i].id << "_" << salt << ":\tdb\t\"" << variables[i].id <<
"\",0\n";
                        for(int j = 0; j < lex.dimension; j++)
                        {
                            printf_out << "\tmov\t\tedx, " << j << "\n";
                            printf_out << "\tpush\tedx" << "\n";
                            printf_out << "\tpush\tdword msg_" << variables[i].id << "_" << salt << "\n";
                            printf_out << "\tpush\tdword printf_arr_" << salt << "\n";
                            printf_out << "\tcall\tprintf\n\tadd\t\tesp, 12\n";
                            printf_out << "\tpush\tdword [" << variables[i].id << "+" << j << "*8+4]\n";
                            printf_out << "\tpush\tdword [" << variables[i].id << "+" << j << "*8]\n";
                            printf_out << "\tpush\tdword printf_flt_" << salt << "\n";
                            printf_out << "\tcall\tprintf\n\tadd\t\tesp, 12\n";
                        }
                    }
                }
            }
            else
            {
                if(lex.dimension == 0)
                {
                    out_code << "\t" << variables[i].id << ":\tdd\t";
```

```
                    if(values[i] == "")
                        out_code << "0\n";
                    else
                    {
                        if(values[i].find(".") != string::npos)
                        {
                            stringstream a_type_force;
                            a_type_force << values[i];
                            float i_type_force;
                            a_type_force >> i_type_force;
                            out_code << (int)round(i_type_force) << "\n";
                        }
                        else
                            out_code << values[i] << "\n";
                    }
                    if(need_printf)
                    {
                        out_code << "\tmsg_" << variables[i].id << "_" << salt << ":\tdb\t\"" << variables[i].id <<
"\",0\n";
                        printf_out << "\tpush\tdword msg_" << variables[i].id << "_" << salt << "\n";
                        printf_out << "\tpush\tdword printf_str_" << salt << "\n";
                        printf_out << "\tcall\tprintf\n\tadd\t\tesp, 8\n";
                        printf_out << "\tpush\tdword [" << variables[i].id << "]\n";
                        printf_out << "\tpush\tdword printf_int_" << salt << "\n";
                        printf_out << "\tcall\tprintf\n\tadd\t\tesp, 8\n";
                    }
                }
                else
                {
                    need_bss = true;
                    bss_out << "\t" << variables[i].id << ":\tresd\t" << lex.dimension << "\n";
                    if(need_printf)
                    {
                        out_code << "\tmsg_" << variables[i].id << "_" << salt << ":\tdb\t\"" << variables[i].id <<
"\",0\n";
                        for(int j = 0; j < lex.dimension; j++)
                        {
                            printf_out << "\tmov\t\tedx, " << j << "\n";
                            printf_out << "\tpush\tedx" << "\n";
                            printf_out << "\tpush\tdword msg_" << variables[i].id << "_" << salt << "\n";
                            printf_out << "\tpush\tdword printf_arr_" << salt << "\n";
                            printf_out << "\tcall\tprintf\n\tadd\t\tesp, 12\n";
                            printf_out << "\tpush\tdword [" << variables[i].id << "+" << j << "*4]\n";
                            printf_out << "\tpush\tdword printf_int_" << salt << "\n";
                            printf_out << "\tcall\tprintf\n\tadd\t\tesp, 8\n";
                        }
                    }
                }
            }
        }
        else
        {
            int hash, chain;
            constants.get_location(variables[i].id, hash, chain);
            constants.get_lexeme(hash, chain, lex);
            if(lex.type == 2)
                out_code << "\tconst_" << hash << "_" << chain << "_" << salt << ":\tdq\t" << variables[i].id << "\n";
            else
                out_code << "\tconst_" << hash << "_" << chain << "_" << salt << ":\tdd\t" << variables[i].id << "\n";
        }
    }

    if(need_adv_int)
        out_code << "\ttmp_var_int_" << salt << ":\tdd\t0\n";
    if(need_printf)
    {
        out_code << "\tprintf_flt_" << salt << ":\tdb\t\"%f\",10,0\n";
        out_code << "\tprintf_int_" << salt << ":\tdb\t\"%i\",10,0\n";
        out_code << "\tprintf_str_" << salt << ":\tdb\t\"%s = \",0\n";
    }
    if(need_bss)
    {
        if(need_printf) out_code << "\tprintf_arr_" << salt << ":\tdb\t\"%s[%i] = \",0\n";
        out_code << "section .bss\n" << bss_out.str();
    }

    out_code << "section .text\n\tglobal main\nmain:\n";
    out_code << outcode.str();
    out_code << printf_out.str();
    out_code << "\tmov\t\teax, 0\n\tret\n";

    out_code.close();
    out_error.close();
    return true;
}
```