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 С++ 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 C 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 С** | **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;

}