МИНИСТЕРСТВО НАУКИ
И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования «Новосибирский государственный технический университет»

НГТУ НЭТИ

Кафедра прикладной математики

Лабораторная работа №4

по дисциплине «Языки программирования и методы трансляции»

ФПМИ

Группа	ПМ-92
Вариант	7
Студенты	Кутузов Иван
	Иванов Владислав
Преподаватель	Еланцева И. Л.
Дата	10.06.2022

Новосибирск

Цель:

Изучить методы генерации кода с учетом различных промежуточных форм представления программы. Изучить методы управления памятью и особенности их использования на этапе генерации кода.
Научиться проектировать генератора кода.

Задание:

Подмножество языка C++ включает:

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

Исходное выражение на C++	Выражение на языке ассмемблер	Комментарий
int a	a dd ?	типы
float b	b real8 ?	
int[] c	c dd 3 dup(?)	
int a = 1	mov eax 1 mov a eax	присваивание
a + 5	mov eax a add 5	сложение
a * 10	mov eax a mul 10	умножение
a[0] - 2	mov si 0 mov eax a[si] sub 2	дереференс массива
float f = 1.0	fld 1.0 fstp f	присваивание
f * 2.0	fld f fld 2.0 fmulp st(1) st(0)	умножение с плавающей точкой

Тесты:

```
void main() {
   int a = 2;
   1 - (a + 2);
}
```

Результат:

```
.386
.MODEL FLAT, STDCALL

EXTRN   ExitProcess@4:NEAR

.DATA
        a       real8   ?
        acc     dd      ?

.CODE

MAIN PROC
        finit

        fld     1.2
        fstp    a

        fld     a
        fld     5.0
        fmulp   st(1), st(0)

        CALL    ExitProcess@4
MAIN ENDP
END MAIN
```

```
void main() {
   float a = 1.2;
   a * 5.0;
}
```

Результат:

```
.386
.MODEL FLAT, STDCALL

EXTRN   ExitProcess@4:NEAR

.DATA
        a       dd      ?
        acc     dd      ?

.CODE

MAIN PROC
        finit

        mov     a       2

        mov     eax     a
        add     eax     2
        mov     acc     eax
        mov     eax     1
        sub     eax     acc

        CALL    ExitProcess@4
MAIN ENDP
END MAIN
```

```
void main() {
   int[] a = {1, 2, 3};
   a[0] + 5 * 15;
}
```

Результат:

```
.386
.MODEL FLAT, STDCALL

EXTRN   ExitProcess@4:NEAR

.DATA
        a       dd 3    dup (?)
        acc     dd      ?

.CODE

MAIN PROC
        finit

        mov     si      0
        mov     ebx     1
        mov     a[si]   ebx
        inc     si
        mov     ebx     2
        mov     a[si]   ebx
        inc     si
        mov     ebx     3
        mov     a[si]   ebx
        inc     si

        mov     eax     5
        mul     eax     15
        mov     acc     eax
        mov     si      0
        mov     eax     a[si]
        add     eax     acc

        CALL    ExitProcess@4
MAIN ENDP
END MAIN
```

Текст программы:

Syntax.h

```
class PriorityTable {
private:
	vector<pair<string, int>> _priority;

public:
	PriorityTable() {
		_priority.push_back(make_pair("(", 0));
		_priority.push_back(make_pair("[", 0));
		_priority.push_back(make_pair("=", 1));
		_priority.push_back(make_pair(")", 1));
		_priority.push_back(make_pair("]", 1));
		_priority.push_back(make_pair("!=", 4));
		_priority.push_back(make_pair("==", 4));
		_priority.push_back(make_pair("<", 4));
		_priority.push_back(make_pair(">", 4));
		_priority.push_back(make_pair("*", 3));
		_priority.push_back(make_pair("+", 2));
		_priority.push_back(make_pair("-", 2));
	}

	int priorityOf(string key) {
		for (auto pair : _priority) {
			if (pair.first == key) {
				return pair.second;
			}
		}
	}
};

class CompilationResult {
public:
	virtual string toString() = 0;
};

class Code : public CompilationResult {
private:
	HashTable<string, Type> _identiferTable;
	vector<Token> _tokens;
	HashTable<string, int> _arrays;

	string _header = ".386\n.MODEL FLAT,
STDCALL\n\nEXTRN\tExitProcess@4:NEAR\n\n";
	string _end = "\tCALL\tExitProcess@4\nMAIN ENDP\nEND MAIN";

	PriorityTable _priorityTable;
public:
	Code(HashTable<string, Type> identiferTable, HashTable<string, int>
arrays, vector<Token> tokens) {
		_identiferTable = identiferTable;
		_tokens = tokens;
		_arrays = arrays;
	}

	string toString() override {
		return _header + dataField() + mainField() + _end;
	}

private:
	string dataField() {
		string data = ".DATA\n";
		auto identifers = _identiferTable.asList();
```

```
            for (auto identifer : *identifers) {
                  auto type = identifer.second;
                  auto key = identifer.first;

                  if (type == Type::INTEGER) {
                        data += "\t" + key + "\tdd\t?\n";
                  }
                  if (type == Type::FLOAT) {
                        data += "\t" + key + "\treal8\t?\n";
                  }
                  if (type == Type::ARRAY) {
                        auto numberOfElements = _arrays.get(key);
                        data += "\t" + key + "\tdd " +
std::to_string(numberOfElements) + "\tdup (?)\n";
                  }
            }

            data += "\tacc\tdd\t?\n";

            return data + "\n";
      }

      string mainField() {
            string code = ".CODE\n\nMAIN PROC\n\tfinit\n\n";
            vector<Token> command;

            for (auto token : _tokens) {
                  if (token.type == Type::END_OF_COMMAND) {
                        code += commandAsString(command);
                        code += "\n";
                        command = vector<Token>();
                  }
                  else {
                        command.push_back(token);
                  }
            }

            return code;
      }

      string commandAsString(vector<Token> command) {
            string commandAsString = {};
            stack<Token> operandStack;

            Token accumulator(Type::INTEGER, "acc");

            bool isAaccumulated = false;

            auto commandWithPriority = sortByPriority(command);

            for (auto element : commandWithPriority) {
                  switch (element.type) {

                        case Type::OPERATION: {
                              if (operandStack.top().type == Type::INTEGER ||
operandStack.top().type == Type::INTEGER_CONSTANT) {
                                    auto right = operandStack.top();
                                    operandStack.pop();
                                    auto left = operandStack.top();
                                    operandStack.pop();

                                    if ((!operandStack.empty() &&
operandStack.top().type == Type::ARRAY)) {
                                          commandAsString += "\tmov\tsi\t" +
left.key + "\n";

                                          left.key = operandStack.top().key
```

```
+ "[si]";
                                                operandStack.pop();
                                        }
                                        if (left.type == Type::ARRAY) {
                                                commandAsString += "\tmov\tsi\t" +
right.key + "\n";

                                                right.key = left.key + "[si]";

                                                left = operandStack.top();
                                                operandStack.pop();
                                        }

                                        if (!isAaccumulated) {
                                                commandAsString += "\tmov\teax\t"
+ left.key + "\n";
                                        }

                                        if (right.key == accumulator.key) {
                                                commandAsString += "\tmov\teax\t"
+ left.key + "\n";
                                        }

                                        if (element.key == "+") {
                                                commandAsString += "\tadd\teax\t"
+ right.key + "\n";
                                        }

                                        if (element.key == "-") {
                                                commandAsString += "\tsub\teax\t"
+ right.key + "\n";
                                        }

                                        if (element.key == "*") {
                                                commandAsString += "\tmul\teax\t"
+ right.key + "\n";
                                        }


                                        if (!isAaccumulated) {
                                                commandAsString += "\tmov\t" +
accumulator.key + "\teax\n";
                                                isAaccumulated = true;
                                        }

                                        operandStack.push(accumulator);
                                }
                                else {
                                        if (!isAaccumulated) {
                                                auto right = operandStack.top();
                                                operandStack.pop();
                                                auto left = operandStack.top();
                                                operandStack.pop();

                                                if (element.key == "+") {
                                                        commandAsString += "\tfld\t"
+ left.key + "\n";

                                                        commandAsString += "\tfld\t"
+ right.key + "\n";

                                                        commandAsString +=
"\tfaddp\tst(1), st(0)\n";
                                                }

                                                if (element.key == "-") {
                                                        commandAsString += "\tfld\t"
+ left.key + "\n";

                                                        commandAsString += "\tfld\t"
```

```
+ right.key + "\n";
                                                commandAsString +=
"\tfsubp\tst(1), st(0)\n";
                                        }

                                        if (element.key == "*") {
                                                commandAsString += "\tfld\t"
+ left.key + "\n";
                                                commandAsString += "\tfld\t"
+ right.key + "\n";
                                                commandAsString +=
"\tfmulp\tst(1), st(0)\n";
                                        }

                                        isAaccumulated = true;
                                }
                                else {
                                        auto operand = operandStack.top();
                                        operandStack.pop();

                                        if (element.key == "+") {
                                                commandAsString += "\tfld\t"
+ operand.key + "\n";
                                                commandAsString +=
"\tfaddp\tst(1), st(0)\n";
                                        }

                                        if (element.key == "-") {
                                                commandAsString += "\tfld\t"
+ operand.key + "\n";
                                                commandAsString +=
"\tfsubp\tst(1), st(0)\n";
                                        }

                                        if (element.key == "*") {
                                                commandAsString += "\tfld\t"
+ operand.key + "\n";
                                                commandAsString +=
"\tfmulp\tst(1), st(0)\n";
                                        }
                                }
                        }

                        break;
                }

                case Type::BOOL_OPERATION: {
                        if (operandStack.top().type == Type::INTEGER ||
operandStack.top().type == Type::INTEGER_CONSTANT) {
                                auto right = operandStack.top();
                                operandStack.pop();
                                auto left = operandStack.top();
                                operandStack.pop();

                                if ((!operandStack.empty() &&
operandStack.top().type == Type::ARRAY)) {
                                        commandAsString += "\tmov\tsi\t" +
left.key + "\n";

                                        left.key = operandStack.top().key
+ "[si]";
                                        operandStack.pop();
                                }
                                if (left.type == Type::ARRAY) {
                                        commandAsString += "\tmov\tsi\t" +
right.key + "\n";
```

```
                                        right.key = left.key + "[si]";

                                        left = operandStack.top();
                                        operandStack.pop();
                                    }

                                    if (!isAaccumulated) {
                                        commandAsString += "\tmov\teax\t"
+ left.key + "\n";
                                    }

                                    if (right.key == accumulator.key) {
                                        right = left;
                                    }

                                    if (element.key == "!=") {
                                        commandAsString += "\t" + left.key
+ "\t.ne.\t" + right.key + "\n";
                                        commandAsString += "\tmov\t" +
accumulator.key + "\t" + right.key + "\n";
                                    }

                                    if (element.key == "==") {
                                        commandAsString += "\t" + left.key
+ "\t.eq.\t" + right.key + "\n";
                                        commandAsString += "\tmov\t" +
accumulator.key + "\t" + right.key + "\n";
                                    }

                                    if (element.key == ">") {
                                        commandAsString += "\t" + left.key
+ "\t.gt.\t" + right.key + "\n";
                                        commandAsString += "\tmov\t" +
accumulator.key + "\t" + right.key + "\n";
                                    }

                                    if (element.key == "<") {
                                        commandAsString += "\t" + left.key
+ "\t.lt.\t" + right.key + "\n";
                                        commandAsString += "\tmov\t" +
accumulator.key + "\t" + right.key + "\n";
                                    }


                                    if (!isAaccumulated) {
                                        commandAsString += "\tmov\t" +
accumulator.key + "\teax\n";
                                        isAaccumulated = true;
                                    }

                                    operandStack.push(accumulator);
                                }

                                break;
                            }

                            case Type::ASSIGMENT: {
                                auto right = operandStack.top();
                                operandStack.pop();
                                auto left = operandStack.top();
                                operandStack.pop();

                                if (left.type == Type::ARRAY) {
                                    commandAsString +=
"\tmov\tsi\t0\n\tmov\tebx\t" + right.key + "\n\tmov\t" + left.key +
"[si]\tebx\n";
                                }
```

```
                                if (operandStack.size() > 0) {
                                        vector<Token> elements = {right, left};

                                        for (int i = 0; i < operandStack.size() -
1; i++) {

elements.push_back(operandStack.top());
                                                operandStack.pop();
                                        }

                                        left = operandStack.top();
                                        operandStack.pop();

                                        commandAsString += "\tmov\tsi\t0\n";
                                        for (int i = elements.size() - 1; i >= 0;
i--) {
                                                commandAsString += "\tmov\tebx\t"
+ elements[i].key + "\n\tmov\t" + left.key + "[si]\tebx\n\tinc\tsi\n";
                                        }
                                }

                                if (left.type == Type::INTEGER) {
                                        commandAsString += "\tmov\t" + left.key +
"\t" + right.key + "\n";
                                }

                                if (left.type == Type::FLOAT) {
                                        commandAsString += "\tfld\t" + right.key
+ "\n\tfstp\t" + left.key + "\n";
                                }

                                break;
                        }

                        default: {
                                operandStack.push(element);

                                break;
                        }
                }
        }

        return commandAsString;
    }

    vector<Token> sortByPriority(vector<Token> command) {
        vector<Token> commandWithPriority;
        stack<Token> operatorStack;
        stack<Token> operandStack;

        for (int i = 0; i < command.size(); i++) {
            if (command[i].key == "(") {
                vector<Token> subCommand = {};

                while (command[i + 1].key != ")") {
                    i++;
                    subCommand.push_back(command[i]);
                }

                stack<Token> swap;
                while (!operandStack.empty())
                {
                    swap.push(operandStack.top());
                    operandStack.pop();
                }
```

```
                            while (!swap.empty())
                            {
                                    commandWithPriority.push_back(swap.top());
                                    swap.pop();
                            }

                            auto subWithPriority = sortByPriority(subCommand);
                            for (auto element : subWithPriority) {
                                    commandWithPriority.push_back(element);
                            }

                            while (!operatorStack.empty()) {

commandWithPriority.push_back(operatorStack.top());
                                    operatorStack.pop();
                            }

                            i += 2;
                    }

                    if (
                            command[i].type == Type::OPERATION ||
                            command[i].type == Type::BOOL_OPERATION ||
                            command[i].type == Type::ASSIGMENT ||
                            command[i].type == Type::BRACKETS_BEGIN ||
                            command[i].type == Type::BRACKETS_END
                            ) {
                            if (operatorStack.empty()) {
                                    operatorStack.push(command[i]);
                            }
                            else if
(_priorityTable.priorityOf(operatorStack.top().key) <
_priorityTable.priorityOf(command[i].key)) {
                                    operatorStack.push(command[i]);
                            }
                            else {
                                    while (!operatorStack.empty() &&
_priorityTable.priorityOf(operatorStack.top().key) >=
_priorityTable.priorityOf(command[i].key)) {
                                            if (operatorStack.top().type !=
Type::BRACKETS_BEGIN && operatorStack.top().type != Type::BRACKETS_END) {
                                                    stack<Token> swap;

                                                    if (command[command.size() -
1].key == "=") {
                                                            if
(commandWithPriority.size() == 0) {
                                                                    auto right =
operandStack.top();
                                                                    operandStack.pop();
                                                                    auto left =
operandStack.top();
                                                                    operandStack.pop();


commandWithPriority.push_back(left);

commandWithPriority.push_back(right);
                                                            }
                                                            else {

commandWithPriority.push_back(operandStack.top());
                                                                    operandStack.pop();
                                                            }
                                                    }
                                                    else if
(commandWithPriority.size() == 0) {
```

```
                                                        while
(!operandStack.empty())
                                                        {

swap.push(operandStack.top());
                                                                operandStack.pop();
                                                        }

                                                        auto left = swap.top();
                                                        swap.pop();
                                                        auto right = swap.top();
                                                        swap.pop();


commandWithPriority.push_back(left);

commandWithPriority.push_back(right);

                                                        while (!swap.empty())
                                                        {

operandStack.push(swap.top());
                                                                swap.pop();
                                                        }
                                                }
                                                else {
                                                        while
(!operandStack.empty())
                                                        {

swap.push(operandStack.top());
                                                                operandStack.pop();
                                                        }


commandWithPriority.push_back(swap.top());
                                                        swap.pop();

                                                        while (!swap.empty())
                                                        {

operandStack.push(swap.top());
                                                                swap.pop();
                                                        }
                                                }

                                                if (operatorStack.top().key != "["
&& operatorStack.top().key != "]") {

commandWithPriority.push_back(operatorStack.top());
                                                }

                                        }
                                        operatorStack.pop();
                                }

                                operatorStack.push(command[i]);
                        }
                }
                else {
                        operandStack.push(command[i]);
                }
        }

        stack<Token> swap;
        while (!operandStack.empty())
        {
```

```
                        swap.push(operandStack.top());
                        operandStack.pop();
                }

                while (!swap.empty())
                {
                        if (swap.top().key != "[" && swap.top().key != "]") {
                                commandWithPriority.push_back(swap.top());
                        }

                        swap.pop();
                }

                auto stackSize = operatorStack.size();

                for (int i = 0; i < stackSize; i++) {
                        if (operatorStack.top().type != Type::BRACKETS_BEGIN &&
operatorStack.top().type != Type::BRACKETS_END) {
                                commandWithPriority.push_back(operatorStack.top());
                        }

                        operatorStack.pop();
                }

                return commandWithPriority;
        }
};

class ErrorMessage : public CompilationResult {
private:
        string _message;

public:
        ErrorMessage(string message) {
                _message = message;
        }

        string toString() override {
                return _message;
        }
};
```