МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

“Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики ”

(НИУ ИТМО)

_____

Факультет программной инженерии и компьютерной техники

Направление подготовки 09.04.04 Программная инженерия

Лабораторная работа № 2

Вариант 3

По дисциплине “Системное программное обеспечение”

Студент группы P4114

Трофимова Полина Владимировна

Преподаватель

Кореньков Юрий Дмитриевич

Санкт-Петербург, 2023 г.

Задание на лабораторную

Реализовать построение графа потока управления посредством анализа дерева разбора для набора входных файлов. Выполнить анализ собранной информации и сформировать набор файлов с графическим представлением для результатов анализа.

Порядок выполнения:

1. Описать структуры данных, необходимые для представления информации о наборе файлов, наборе подпрограмм и графе потока управления, где:

a. Для каждой подпрограммы: имя и информация о сигнатуре, граф потока управления, имя исходного файла с текстом подпрограммы.

b. Для каждого узла в графе потока управления, представляющего собой базовый блок алгоритма подпрограммы: целевые узлы для безусловного и условного перехода (по мере необходимости), дерево операций, ассоциированных с данным местом в алгоритме, представленном в исходном тексте подпрограммы

2. Реализовать модуль, формирующий граф потока управления на основе синтаксической структуры текста подпрограмм для входных файлов

a. Программный интерфейс модуля принимает на вход коллекцию, описывающую набор анализируемых файлов, для каждого файла – имя и соответствующее дерево разбора в виде структуры данных, являющейся результатом работы модуля, созданного по заданию 1 (п. 3.b).

b. Результатом работы модуля является структура данных, разработанная в п. 1, содержащая информацию о проанализированных подпрограммах и коллекция с информацией об ошибках

c. Посредством обхода дерева разбора подпрограммы, сформировать для неё граф потока управления, порождая его узлы и формируя между ними дуги в зависимости от синтаксической конструкции, представленной данным узлом дерева разбора: выражение, ветвление, цикл, прерывание цикла, выход из подпрограммы – для всех синтаксических конструкций по варианту (п. 2.b)

d. С каждым узлом графа потока управления связать дерево операций, в котором каждая операция в составе текста программы представлена как совокупность вида операции и соответствующих операндов (см задание 1, пп. 2.d-g)

e. При возникновении логической ошибки в синтаксической структуре при обходе дерева разбора, сохранить в коллекции информацию об ошибке и её положении в исходном тексте

3. Реализовать тестовую программу для демонстрации работоспособности созданного модуля

a. Через аргументы командной строки программа должна принимать набор имён входных файлов, имя выходной директории

b. Использовать модуль, разработанный в задании 1 для синтаксического анализа каждого входного файла и формирования набора деревьев разбора

c. Использовать модуль, разработанный в п. 2 для формирования графов потока управления каждой подпрограммы, выявленной в синтаксической структуре текстов, содержащихся во входных файлах

d. Для каждой обнаруженной подпрограммы вывести представление графа потока управления в отдельный файл с именем “sourceName.functionName.ext” в выходной директории, поумолчанию размещать выходной файлы в той же директории, что соответствующий входной

e. Для деревьев операций в графах потока управления всей совокупности подпрограмм сформировать граф вызовов, описывающий отношения между ними в плане обращения их друг к другу по именам и вывести его представление в дополнительный файл, по-умолчанию размещаемый рядом с файлом, содержащим подпрограмму main.

4. Результаты тестирования представить в виде отчета, в который включить:

a. В части 3 привести описание разработанных структур данных

b. В части 4 описать программный интерфейс и особенности реализации разработанного модуля

c. В части 5 привести примеры исходных анализируемых текстов для всех синтаксических конструкций разбираемого языка и соответствующие результаты разбора

Узлы графа потока управления описываются, описывается структурой:

Переход к следующему узлу осуществляется без условия (nextByDefault) или по условию (nextByCondition), ops – различные операции, tag – текстовое представление узла.

```
typedef struct CFGNode {
    struct CFGNode* nextByDefault;
    struct CFGNode* nextByCondition;
    struct OpNode* ops;
    char *tag;
    myTreeNode* source;
} CFGNode;

typedef struct OpNode {
    OPkind kind;
    char* tag;
    int operandsCount;
    struct OpNode* operands[];
} OpNode;
```

Представление выражений и дерева операций:

```
static OpNode *processExpression(myTreeNode* node, ModelCollectionState* s, bool isNotTopLevelExpr)
{
    switch (node->kind)
    {
    case AST_EXPR_DIV: return makeBinaryOperation(node, OP_DIV, s);
    case AST_EXPR_MUL: return makeBinaryOperation(node, OP_MUL, s);
    case AST_EXPR_SUB: return makeBinaryOperation(node, OP_SUB, s);
    case AST_EXPR_SUM: return makeBinaryOperation(node, OP_SUM, s);
    case AST_EXPR_NOT_EQUAL: return makeBinaryOperation(node, OP_NOT_EQUAL, s);
    case AST_EXPR_LESS: return makeBinaryOperation(node, OP_LESS, s);
    case AST_EXPR_GREATER: return makeBinaryOperation(node, OP_GREATER, s);
    case AST_EXPR_LESS_EQUAL: return makeBinaryOperation(node, OP_LESS_EQUAL, s);
    case AST_EXPR_GREATER_EQUAL: return makeBinaryOperation(node, OP_GREATER_EQUAL, s);

    case AST_EXPR_INV: return makeBinaryOperation(node, OP_INV, s);
    case AST_EXPR_NEG: return makeBinaryOperation(node, OP_NEG, s);
    case AST_EXPR_NOT: return makeBinaryOperation(node, OP_NOT, s);

    case AST_CALL_OR_INDEXER_EXPR: {
        OpNode* o = myAllocWithArray(OpNode, OpNode*, 2);
        o->tag = node->children[0]->text;
        o->operandsCount = 2;
        o->operands[0] = processExpression(node->children[1], s, true);
        o->operands[1] = processExpression(node->children[1], s, true);
        o->kind = OP_ARR_GET_ITEM;
        return o;
    }
```

```
static CFGNode* processStatement(CFGNode* cn, myTreeNode* node, ModelCollectionState* s)
{
    switch (node->kind)
    {
    case IF_STMT: {
        CFGNode* exit = createNode(s->currProcedure, node, "if-leave", NULL);
        CFGNode* condition = createNode(s->currProcedure, node, "if-cond", processExpression(node->children[0], s, false ));
        cn->nextByDefault = condition;
        CFGNode* thenBody = createNode(s->currProcedure, node, "if-then", NULL);
        condition->nextByCondition = thenBody;
        CFGNode* thenExit = createNode(s->currProcedure, node, "if-leave-then", NULL);
        CFGNode* thenBodyTail = processStatement(thenBody, node->children[1], s);
        thenBodyTail->nextByDefault = thenExit;
        thenExit->nextByDefault = exit;
        if (node->childrenCount > 2)
        {
            CFGNode* elseBody = createNode(s->currProcedure, node, "if-else", NULL);
            condition->nextByDefault = elseBody;
            CFGNode* elseBodyTail = processStatement(elseBody, node->children[2], s);
            CFGNode* elseExit = createNode(s->currProcedure, node, "if-else-leave", NULL);
            elseBodyTail->nextByDefault = elseExit;
            elseExit->nextByDefault = exit;
        }
        else
        {
            condition->nextByDefault = exit;
        }
        return exit;
    }
```

Вывод полученного графа потока управления осуществляется в формате dgml.

Примеры входных и выходных данных:

Входные данные 1.

```
function f(x as string, y as long) as int
    dim x as int

    do
        dim y as int
        x = y ;
    loop while z < 3

    if x < k then
                x+2;
    end if

    while z > c

        if x > k then
            y + k;
            if x < k then
                x+2;
            else
                break
            end if
        else
            break
        end if

        x+2;

    wend
end function
```

Результат 1.

Вывод:

В ходе выполнения лабораторной работы, мною был изучен процесс построения графа потока управления для программы и реализовано построение графа потока управления посредством анализа дерева разбора из Лабораторной работы №1.