Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Национальный исследовательский университет ИТМО»
Факультет «Программной инженерии и компьютерной техники»

ОТЧЕТ

к Практическому заданию №2

по дисциплине

«Системное программное обеспечение»

Выполнил:студент гр. P4114
Попов А. И.
Проверил:к.т.н., преподаватель факультета «ПИиКТ»
_____ Ю.Д. Кореньков

Санкт-Петербург 2024

Содержание

Цели и задачи

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, по-

умолчанию 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.

f.

Сообщения об ошибке должны выводиться тестовой программной (не модулем, отвечающим

за анализ!) в стандартный поток вывода ошибок

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

Описание работы

Данная программа предназначена для анализа AST дерева с целью построения трех ключевых структур – графа потока управления, дерева операций и графа вызовов.

Описание разработанных структур данных

1. **CfgNode:**

Узел графа управления (CFG), представляющий базовый блок или операцию, включает:

- id: Уникальный идентификатор узла.
- nodeName: Название узла (например, If, LoopEntry, AssignmentOP).
- optree: Дерево операций (структура OpNode), связанное с узлом.
- nextNode: Ссылка на следующий узел в CFG.
- condNode: Ссылка на узел-условие (для ветвлений).

2. **CfgInstance**:

Контейнер для хранения всех узлов CFG, включает:

- nodes: Массив указателей на узлы CFG.

- count: Количество узлов.
- capacity: Емкость массива.

3. **CallEdge**:

Ребро в графе вызовов между функциями.

- caller: Имя функции-источника вызова.
- callee: Имя функции-цели вызова.
- next: Ссылка на следующее ребро в списке.

4. **CallGraph**:

Граф вызовов функций.

- functions: Список имен функций.
- edges: Список ребер вызовов.

5. **ProgramUnit**:

Модуль программы, содержащий метаданные и аналитические структуры.

- funcs: Список объявленных функций.
- callGraph: Граф вызовов.
- inputFiles: Список входных файлов с AST.

Интерфейсы модуля:

1. **Создание CFG**:
    - prepareControlFlowGraph(): Преобразует AST функции в CFG.
    - collectCfg(): Рекурсивно обходит AST и создает узлы CFG.
2. **Обработка узлов AST**:
    - handleIfStatement(), handleWhileStatement(), handleFunctionCall() и др.: Создают узлы CFG для условных операторов, циклов и вызовов функций.
    - buildOpTree(): Строит дерево операций для выражений.
3. **Граф вызовов**:

- initCallGraph(): Инициализирует граф.
- addFunctionToGraph(), addCallEdge(): Добавляют функции и ребра вызовов.

Особенности реализации:

- **Ветвления**: для if создаются узлы If, Then, Else, Merge. Условия связываются через condNode.
- **Циклы**: для while формируются
  узлы LoopEntry, LoopCond, LoopBody, LoopEnd. Управление потоком возвращается к условию.
- **Вызовы функций**: Узел Call связывается с графом вызовов через addCallEdge().
- **Операции**: Дерево операций (OpNode) строится для выражений, включая арифметику, сравнения, присваивания.

Пример

Входные данные:

```
int main(int argc, char[] argv)
{
    printf("1");

    while(1)
    {
        printf("2");

        while(2)
        {
            printf("3");

            if(3){
                printf("break\n");
                break;
            }
            else if (4)
                printf("continue\n");
            else
            {
                printf("labb:\n");
                printf("4");
                return 0;
            }
        }

        printf("5");

        if(5){
            printf("break\n");
            break;
        }
        else if (6)
            printf("continue\n");
        else
            printf("6");

        printf("7");

        do
        {
```

```
        printf("8");
        if (9) {
            printf("break\n");
            break;
        }
    } while(10);

    printf("11");

    if(11)
        printf("break\n");
    else if (12)
        printf("continue\n");
    else
    {
        printf("12");
        printf("goto labb\n");
    }

    printf("13");

}

printf("14");

if (a > 3)
{
    printf("1\n");
}
else
{
    printf("2\n");
    printf("goto lab\n");
}

while (i < 10)
{
    if (a > b)
         i -= 2;
    else if (2 * 3)
        printf("break\n");
    else
        printf("continue\n");
}
```

```
    if (1)
        if(2)
            if(3)
                x();
            else
                y();

        else
            z();



    if (a > 3)
    {
        printf("1\n");
    }
    else
    {
        printf("2\n");
        printf("22\n");
        printf("222\n");
    }

    while (a < 3)
    {
        printf("3\n");
    }

    printf("Hello!\n");

    count_digits(5);
}

int x()
{
y();
z();
}

int y()
{
x();
```

```
z();
}

int z()
{
x();
}

int count_digits(int num) {
    int count;
    count = 0

    while (num != 0) {
        if (a == 5) {
            num /= 10;
            count++;
        }
    }

    return(count);
}
```

Выходные данные:

Preparing CFG for file: example.txt

Processing node: Body

Processing node: Call

Processing node: Call

Processing node: Body

Processing node: Call

Processing node: Call

[DEBUG] Drawing CFG for x

Preparing CFG for file: example.txt

Processing node: Body

Processing node: Call

Processing node: Call

Processing node: Body

Processing node: Call

Processing node: Call

[DEBUG] Drawing CFG for y

Preparing CFG for file: example.txt

Processing node: Body

Processing node: Call

Processing node: Body

Processing node: Call

[DEBUG] Drawing CFG for z

Preparing CFG for file: example.txt

Processing node: Body

Processing node: VarDeclaration

Processing node: AssignmentOP

Processing node: LoopStatement

Processing node: Body

Processing node: ConditionStatement

Processing node: Body

Processing node: AssignmentOP

Processing node: ++

Processing node: Identifier

Processing node: count

Processing node: ReturnStatement

Processing node: Body

Processing node: VarDeclaration

Processing node: AssignmentOP

Processing node: LoopStatement

Processing node: Body

Processing node: ConditionStatement

Processing node: Body

Processing node: AssignmentOP

Processing node: ++

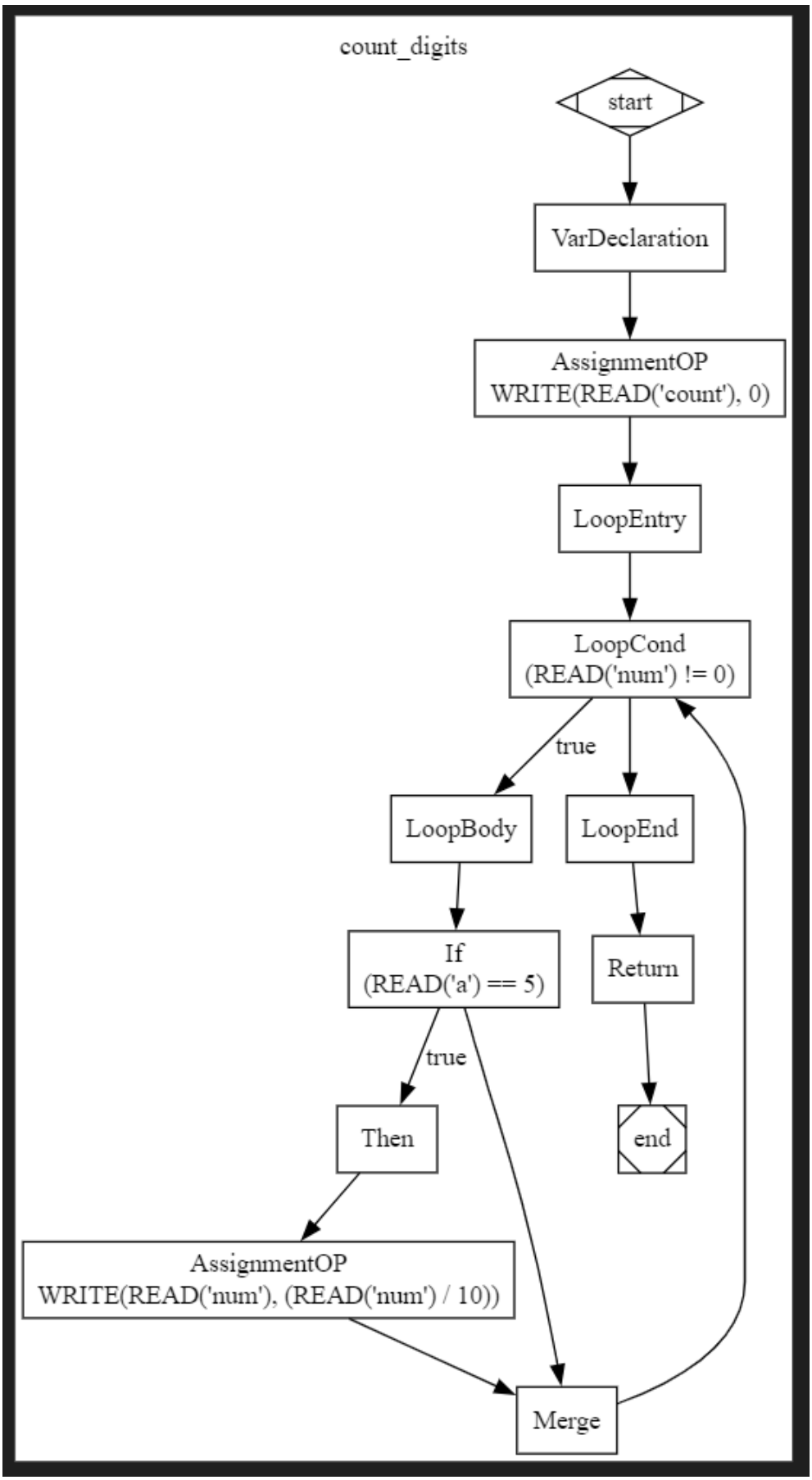
Processing node: Identifier

Processing node: count

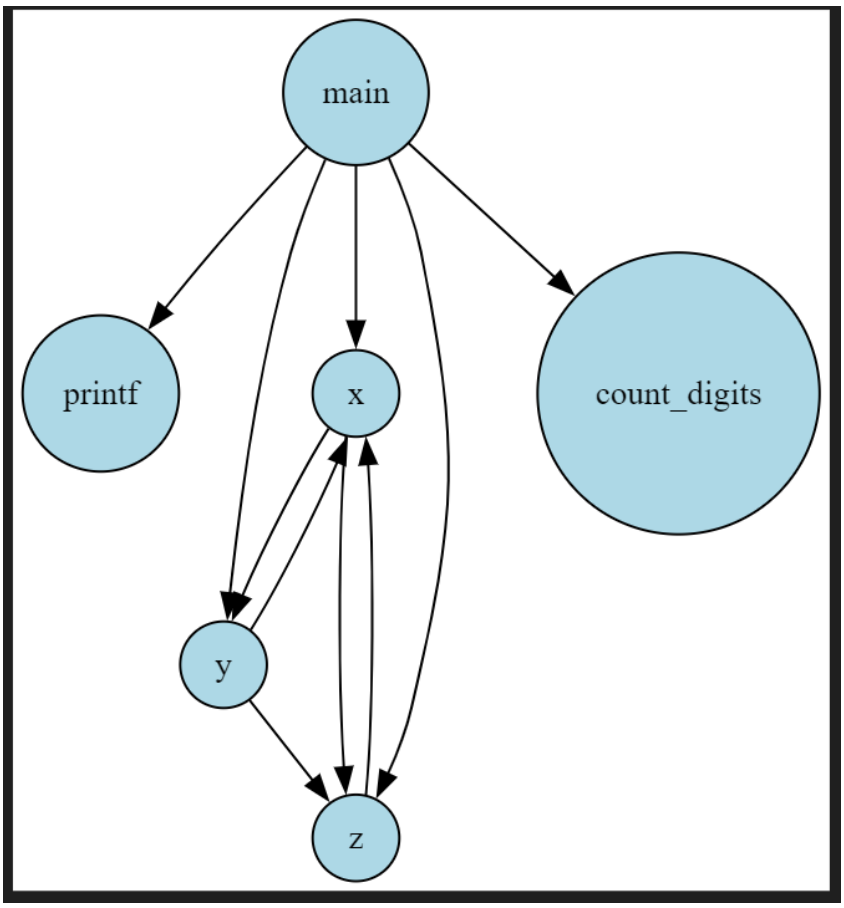
Processing node: ReturnStatement

[DEBUG] Drawing CFG for count_digits

Вывод обрабатывающихся нод AST в консоли.



Файл count_digits.dot для визуализации в Graphviz.



Файл call_graph.dot для визуализации в Graphviz.

Заключение

Программа анализирует исходный код, строит управляющие графы (CFG) для функций и граф вызовов. В результате выполнения практического задания были созданы, модули для генерации CFG, OperationTree, CallGraph. Пример демонстрирует корректную обработку циклов, условий, присваиваний и вызовов. Визуализация в формате DOT позволяет наглядно представить структуру программы. В результате, научился обходить AST дерево, и строить из него для каждой отдельный функции граф потока управления и дерево операций.