МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«Санкт Петербургский национальный-исследовательский университет Информационных технологий, механики и оптики»

Факультет программной инженерии и компьютерной техники

«Системное программное обеспечение»

Выполнил студент группы №P4116
Веденин Вадим Витальевич

Проверил
Кореньков Юрий Дмитриевич

САНКТ -ПЕТЕРБУРГ
2024

Цели

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.

Задачи

1. Описать структуры данных, необходимые для представления информации о наборе файлов, наборе
    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

Описание решения

При dfs-обходе AST формируется граф потока управления (CFG), представленный следующими структурами

```
struct CFG {
    char* procedureName;
    Block* entryblock;
    int id;
    CFGLinkList* cfgLinkList;
};


struct Block {
    int id;
    char* call;
    LinkList* predecessors;
    LinkList* exits;
    ASTNode* node;
    char* info;
};


struct Link {
    Block* source;
    Block* target;
    char* comment;
};
```

На вход программа получает список файлов с входными данными. Далее осуществляется поиск всех подпрограмм, после чего для каждой подпрограммы формируется граф потока управления и выводится в отдельный файл в формате dot. В дальнейшем его можно визуализировать при помощи любой подходящей программы.

Входные данные

Файл №1

```
//int commented_func (int arg1) {
//    while (i <= 10) {
//        if (i > 3) {
//            r = 5;
//        }
//}


int test_func2 () {
    //int a = 2;
    int j = 0;
    do {
        res = 10 + 2;
        j = j + 1;
    } while (j > 3);
}

int test_func3 (int arg1) {
    // test_func2();
    if (a == 5) {
        a = 4;
    }
}

int test_func4 (int arg1) {
    int a;
    if (a == 5) {
        a = arg1;
        test_func3(a);
    }else {
        test_func2();
    }
}

int test_func() {
    test_func4(2);
    if (f1() > 2) {
        b = 5;
        x = 4 + 2;
        do {
            r = x - 2;
            do {
                while (k > 10) {
                    test_func1(10);
                    a = b * 2;
                    test_func4();
                }
            } while (i < 2);
       } while (j > 3);
    }
}
```

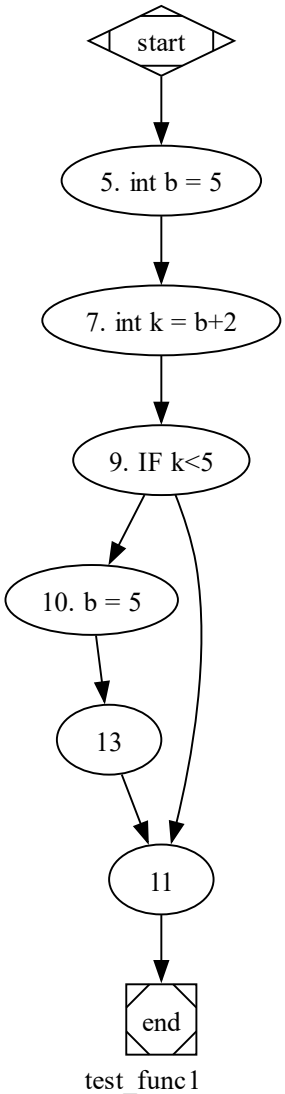
Файл №2

```
int test_func1() {
    int b = 5;
    int k = b + 2;
    if (k < 5) {
        b = 5;
    }
}
```

Визуализированный результат

test_func1:

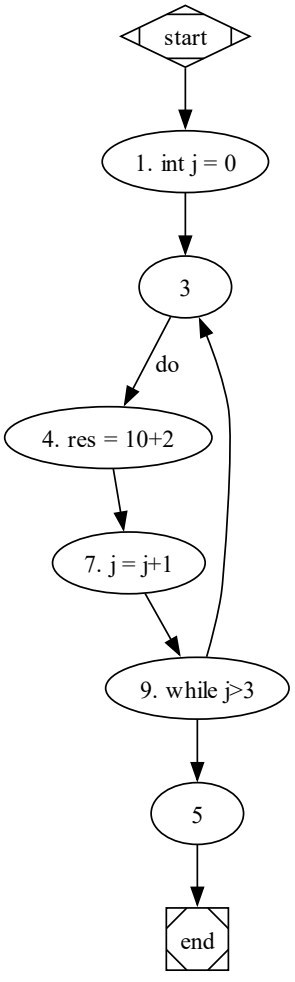
```
int test_func1() {
    int b = 5;
    int k = b + 2;
    if (k < 5) {
        b = 5;
    }
}
```



test_func1

test_func2:

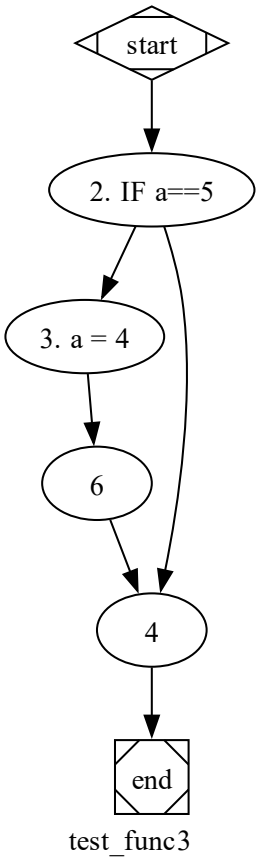
```
int test_func2 () {
    //int a = 2;
    int j = 0;
    do {
        res = 10 + 2;
        j = j + 1;
    } while (j > 3);
}
```



test_func2

test_func3:

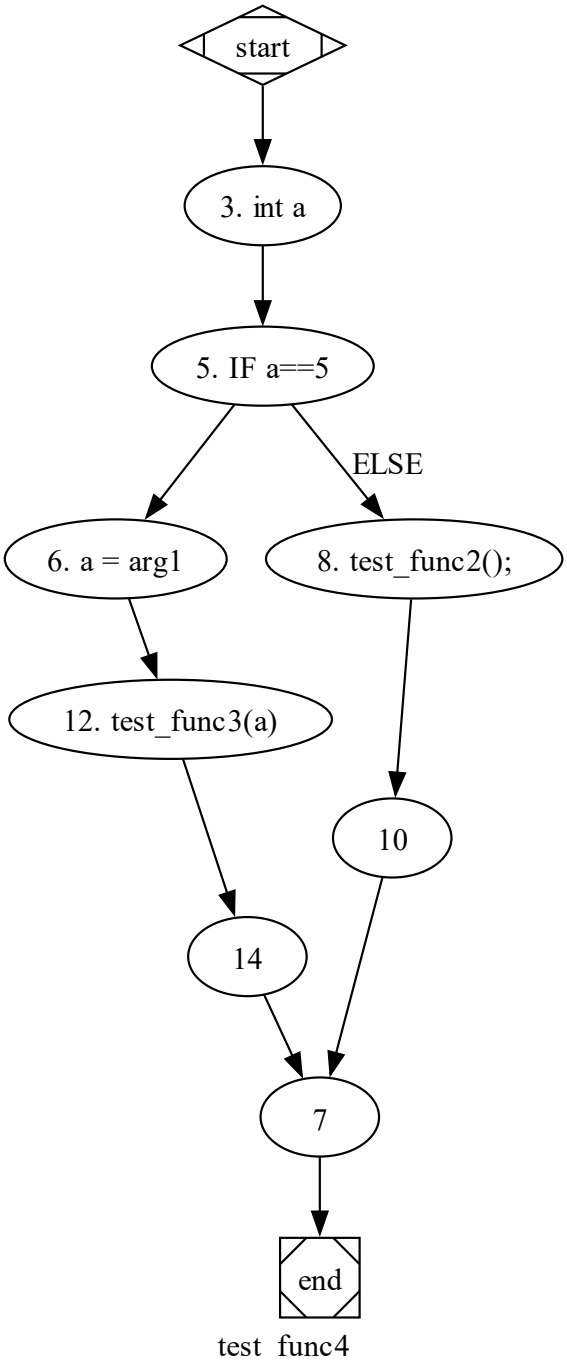
```
int test_func3 (int arg1) {
    // test_func2();
    if (a == 5) {
        a = 4;
    }
}
```



test_func3

test_func4:

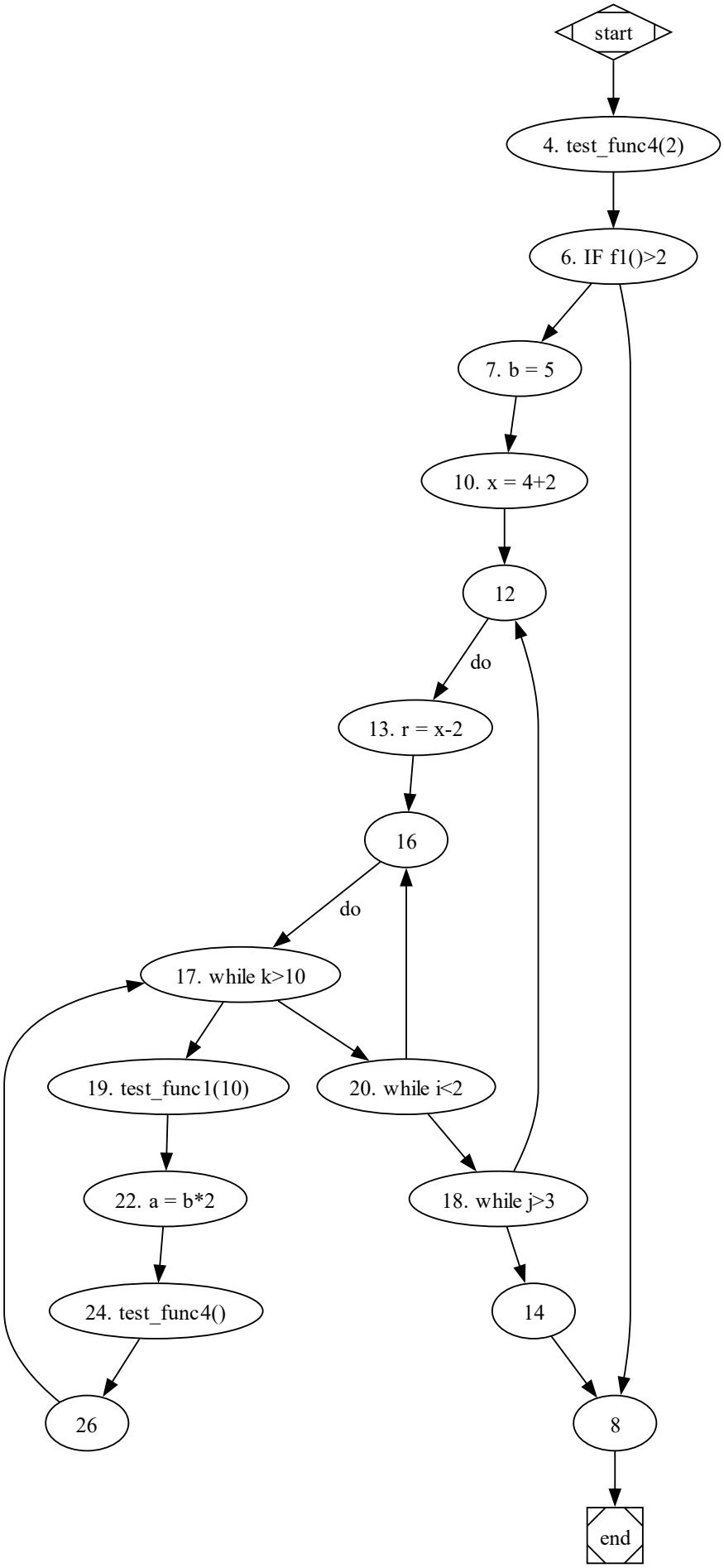
```
int test_func4 (int arg1) {
    int a;
    if (a == 5) {
        a = arg1;
        test_func3(a);
    }else {
        test_func2();
    }
}
```



test_func4

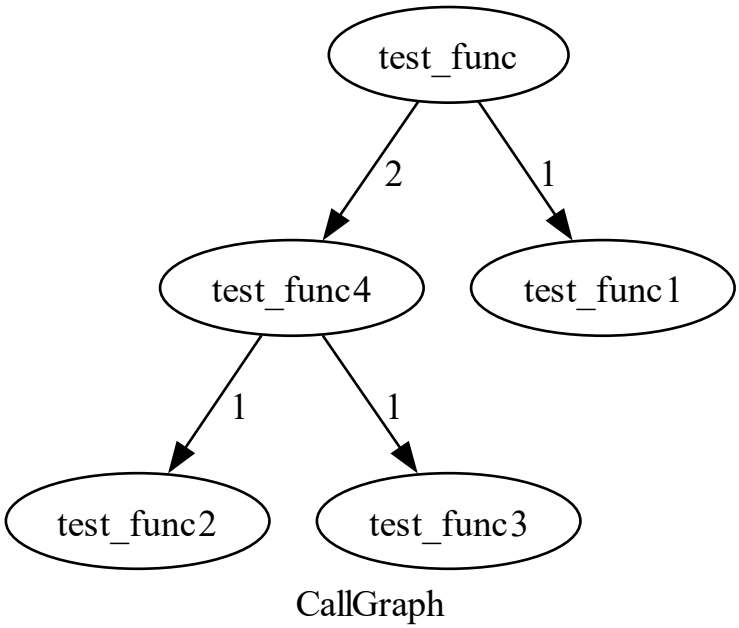
test_func:

```
int test_func() {
    test_func4(2);
    if (f1() > 2) {
        b = 5;
        x = 4 + 2;
        do {
            r = x - 2;
            do {
                while (k > 10) {
                    test_func1(10);
                    a = b * 2;
                    test_func4();
                }
            } while (i < 2);
       } while (j > 3);
    }
}
```


start
4. test_func4(2)
6. IF f1()>2
7. b = 5
10. x = 4+2
12
do
13. r = x-2
16
do
17. while k>10
19. test_func1(10)
20. while i<2
22. a = b*2
18. while j>3
24. test_func4()
14
26
8
end

test_func

Граф вызовов:



Вывод:

Таким образом, в результате выполнения работы были изучены граф потока управления и граф вызовов. Был разработан модуль, создающий их на основе дерева AST. Также было реализовано их графическое представление в формате dot.