Министерство науки и высшего образования Российской Федерации
федеральное государственное автономное образовательное учреждение высшего образования
«национальный исследовательский университет итмо»
(Университет ИТМО)

Факультет программной инженерии и компьютерной техники

Образовательная программа: 09.04.04 Программная инженерия

Направление подготовки: Системное и прикладное программное обеспечение

Системное программное обеспечение

Отчёт по лабораторной работе №2

Тема задания: Построение графа потока управления и графа вызовов программы

Обучающийся: Попов Павел Сергеевич, гр. P4116

Преподаватель: Кореньков Юрий Дмитриевич, доцент

Санкт-Петербург

21 ноября 2024

1 Постановка задачи

1.1 Описать структуры данных

Необходимо описать структуры данных, которые представляют информацию о:

- наборе файлов,
- наборе подпрограмм,
- графе потока управления.

Для каждой подпрограммы необходимо хранить:

- имя и информацию о сигнатуре,
- граф потока управления,
- имя исходного файла с текстом подпрограммы.

Для каждого узла в графе потока управления, представляющего базовый блок алгоритма подпрограммы, необходимо хранить:

- целевые узлы для безусловного и условного перехода (по мере необходимости),
- дерево операций, ассоциированных с данным местом в алгоритме, представленном в исходном тексте подпрограммы.

1.2 Реализация модуля формирования графа потока управления

Модуль принимает на вход коллекцию, описывающую набор анализируемых файлов. Для каждого файла передаются:

- имя файла,
- соответствующее дерево разбора в виде структуры данных.

Результатом работы модуля является:

- структура данных, содержащая информацию о проанализированных подпрограммах,
- коллекция с информацией об ошибках.

Обходя дерево разбора подпрограммы, модуль формирует для неё граф потока управления:

- порождает узлы,
- формирует между ними дуги в зависимости от синтаксической конструкции (выражение, ветвление, цикл, прерывание цикла, выход из подпрограммы и др.).

Каждый узел графа потока управления связан с деревом операций, в котором каждая операция представлена как совокупность:

- вида операции,
- соответствующих операндов.

При возникновении логической ошибки в синтаксической структуре при обходе дерева разбора информация об ошибке сохраняется в коллекции с указанием её положения в исходном тексте.

1.3 Реализация тестовой программы

Тестовая программа должна:

- Принимать через аргументы командной строки набор имён входных файлов и имя выходной директории.
- Использовать модуль синтаксического анализа для формирования деревьев разбора.
- Использовать модуль формирования графов потока управления для каждой подпрограммы.
- Выводить представление графа потока управления в отдельный файл с именем `sourceName.functionName.ext` в выходной директории.
- Для деревьев операций формировать граф вызовов и выводить его представление в дополнительный файл.
- Сообщения об ошибках выводить в стандартный поток вывода ошибок.

1.4 Результаты тестирования

Результаты тестирования представлены в виде отчёта, включающего:

- Описание разработанных структур данных.
- Описание программного интерфейса и особенностей реализации модуля.
- Примеры анализируемых исходных текстов и соответствующие результаты разбора.

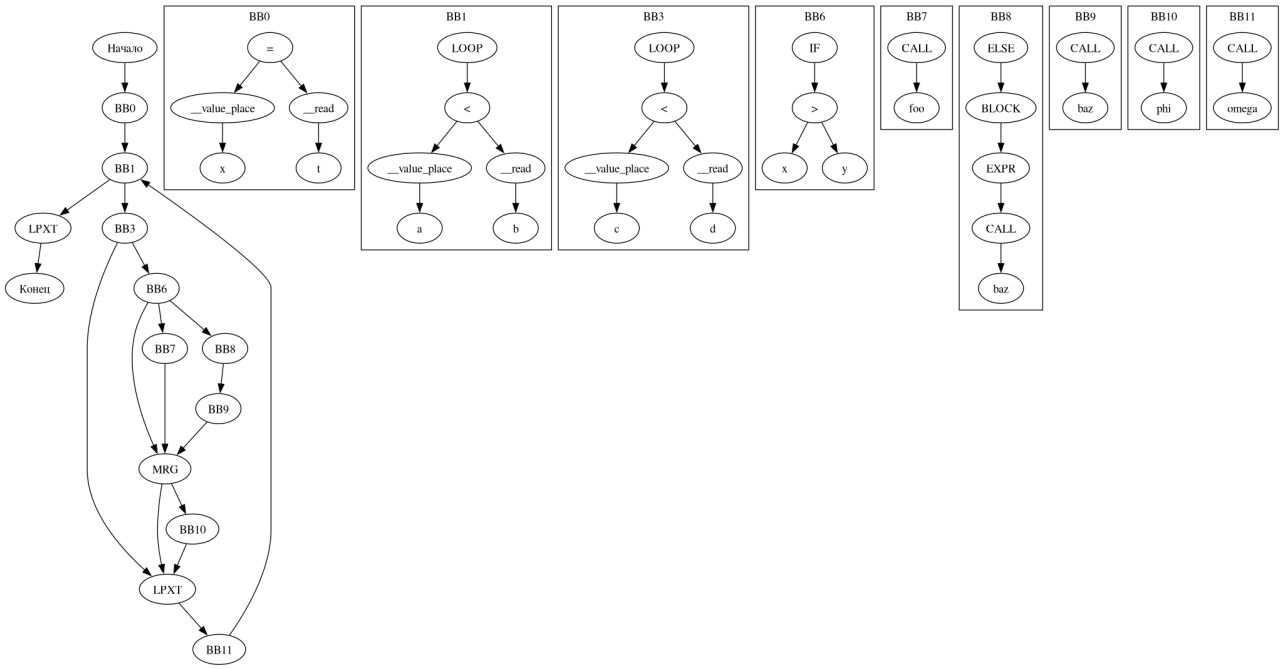
2 Примеры входных программ

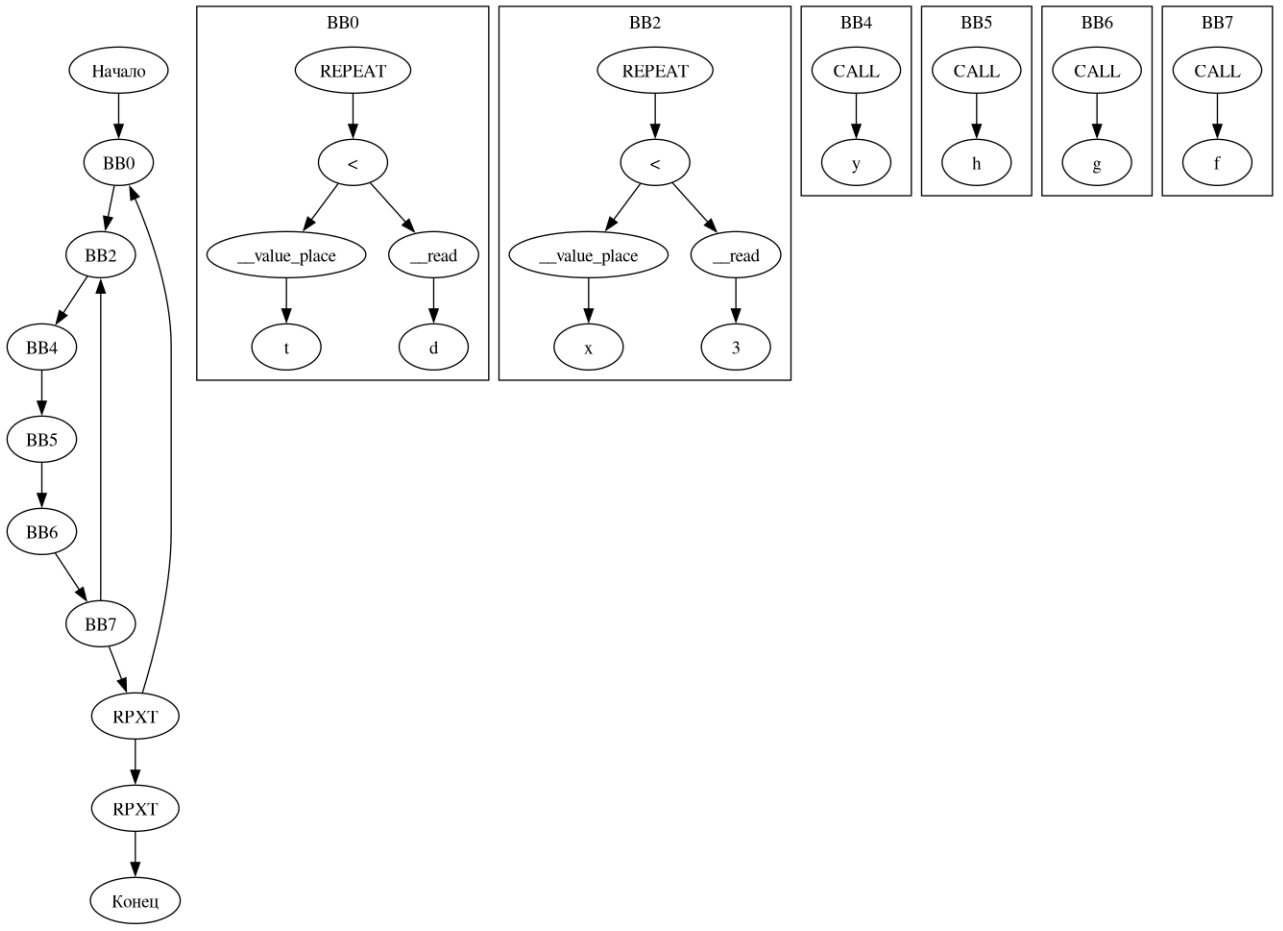
```
    def repeatCallExample() of int
    {
        {
            f(g(h(x,y())));
        } until x < 3;

    } while t < d;
    end

def main() of int
    x = t;
    while a < b
        while c < d
            if x > y then {
                foo();
                break;
                bar();
            }
            else {
                baz();
            }
            phi();
            break;
            psi();
    end
    omega();
    end
end
```

```
def baz() of int
    x = k;
end

def bar() of int
    t = d;
end

def foo() of int
    baz();
    baz();
    foo();
end

def main() of int
    foo();
    bar();
    baz();
end
```

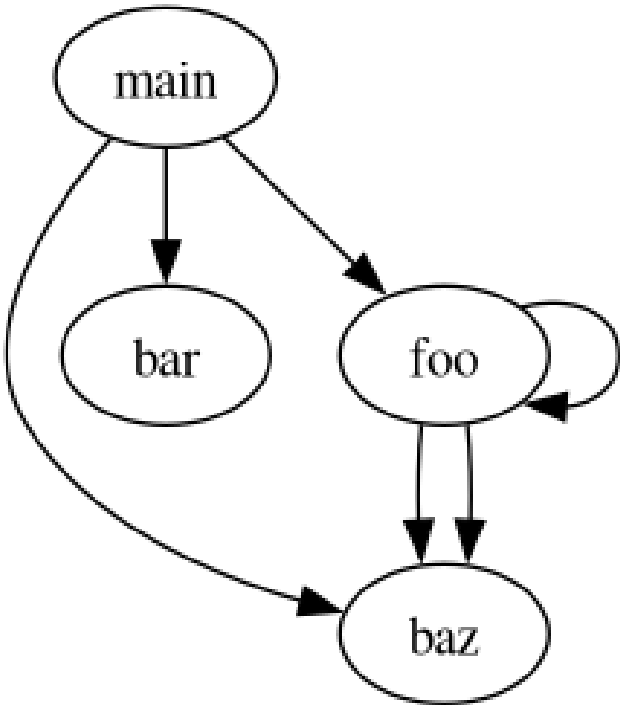
3 Результаты

Примеры построения графов потока управления.





Примеры построения графа вызовов.



4 Выводы

Получена тестовая программа, в результате которой порождаются cfg, cg.

5 Листинг кода

Ниже приведены примеры некоторые структуры, которые использовались в процессе построения cfg и cg.

```
    typedef enum BB_t {
  standard,
  merge,
  loop_exit,
  repeat_exit
} BB_t;

typedef struct BB {
  AST* node;
  OT* opTree;
  BB_t bt;
  int* successors; //sucessors` indices in BasicBlock** array
  int successorCount;
} BB;

typedef struct CFG {
  ST* symbolTable;
```

```
18    BB** blocks;
19    int blockCount;
20    int lastProcessedIndex;
21    LS* loopStack;
22    IS* ifStack;
23
24  } CFG;
25
26  typedef struct CGB { // call graph block
27      AST* node;
28      int* successors; //sucessors` indices in CGB2** array
29      int successorCount;
30  } CGB;
31
32  typedef struct CG
33  {
34      CGB** blocks;
35      int blockCount;
36  } CG;
37
```