# УНИВЕРСИТЕТ ИТМО

Факультет программной инженерии и компьютерной техники Направление подготовки 09.03.04 Программная инженерия Дисциплина «Системное программное обеспечение»

Лабораторная работа №2

## Студент

*Лещуков А. А. P4114*

## Преподаватель

*Кореньков Ю. Д.*

Санкт-Петербург, 2025 г.

# Цель

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 подпрограмм и графе потока управления:
2. Для каждой подпрограммы: имя и информация о сигнатуре, граф потока управления, имя исходного файла с текстом подпрограммы.
3. 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.
4. 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.

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

1. 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.
2. 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.
3. При возникновении логической ошибки в синтаксической структуре при обходе дерева разбора, сохранить в коллекции информацию об ошибке и её положении в исходном тексте.
4. 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.

1. Сообщения об ошибке должны выводиться тестовой программной (не модулем, отвечающим за анализ!) в стандартный поток вывода ошибок.
2. 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

Структура, которая хранит граф потока управления для всех функций данного файла:

typedef struct CfgFile {

char\* fileName;

CfgSubroutine\* headSubroutine;

int numberOfSubroutines;

} CfgFile;

Структура, которая хранит начало графа потока управления конкретной функции, а также имя функции, параметры и возвращаемый тип:

typedef struct CfgSubroutine {

char\* functionName;

ParameterList parameterList;

TypeRef returnType;

CfgNode\* cfgStart;

struct CfgSubroutine\* next;

} CfgSubroutine;

Структура, которая хранит список параметров:

typedef struct {

Parameter\* headParameter;

int numberOfParams;

} ParameterList;

Структура, которая хранит объект параметра – идентификатор, тип и ссылка на следующий:

typedef struct Parameter {

char\* identifier;

TypeRef typeRef;

struct Parameter\* next;

} Parameter;

Структура, которая объект типа – это встроенный тип, кастомный или массив:

typedef struct {

TypeRefTag tag;

union {

BuiltInType builtInType;

char\* identifier;

ArrayType arrayType;

};

} TypeRef;

Перечисление тегов для определения того, какой это тип:

typedef enum {

TYPE\_REF\_NO\_TYPE,

TYPE\_REF\_BUILT\_IN\_TYPE,

TYPE\_REF\_IDENTIFIER,

TYPE\_REF\_ARRAY\_TYPE

} TypeRefTag;

Структура, которая хранит тип массива:

typedef struct {

ArrayTypeTag tag;

union {

BuiltInType builtInType;

char\* identifier;

};

int dimensions[5];

int numberOfDimensions;

} ArrayType;

Перечисление тегов типа массива:

typedef enum {

ARRAY\_TYPE\_BUILT\_IN\_TYPE,

ARRAY\_TYPE\_IDENTIFIER

} ArrayTypeTag;

Перечисление тегов встроенных типов:

typedef enum BuiltInType {

BUILT\_IN\_TYPE\_BOOL,

BUILT\_IN\_TYPE\_BYTE,

BUILT\_IN\_TYPE\_INT,

BUILT\_IN\_TYPE\_UINT,

BUILT\_IN\_TYPE\_LONG,

BUILT\_IN\_TYPE\_ULONG,

BUILT\_IN\_TYPE\_CHAR,

BUILT\_IN\_TYPE\_STRING,

} BuiltInType;

Узел графа потока управления:

typedef struct CfgNode {

CfgNodeType type;

struct CfgNode\* next[2]; // there can be maximum 2 next nodes (true/false for conditions)

int numberOfNext;

OpTreeNode\* opTree; // operations tree

int id; /\* for printing out \*/

bool isVisited; /\* for prints as well \*/

} CfgNode;

Типы узлов графа потока управления:

typedef enum {

NODE\_TYPE\_IF\_STAT,

NODE\_TYPE\_WHILE\_STAT,

NODE\_TYPE\_REPEAT\_STAT,

NODE\_TYPE\_EXPR\_STAT,

NODE\_TYPE\_BREAK\_STAT,

NODE\_TYPE\_START, // as START

NODE\_TYPE\_END, // as END

NODE\_TYPE\_EMPTY // used as a helper node for cfg building

} CfgNodeType;

Узел дерева операций:

typedef struct OpTreeNode {

OpTreeNodeType type;

struct OpTreeNode\*\* next;

int numberOfNext;

int id;

union {

int number;

char\* string;

char\* identifier;

} data;

} OpTreeNode;

Типы узлов дерева операций:

typedef enum {

OP\_TREE\_NODE\_TYPE\_WRITE,

OP\_TREE\_NODE\_TYPE\_READ,

OP\_TREE\_NODE\_TYPE\_LOGICAL\_OR,

OP\_TREE\_NODE\_TYPE\_LOGICAL\_AND,

OP\_TREE\_NODE\_TYPE\_EQUAL,

OP\_TREE\_NODE\_TYPE\_NOT\_EQUAL,

OP\_TREE\_NODE\_TYPE\_LESS,

OP\_TREE\_NODE\_TYPE\_GREATER,

OP\_TREE\_NODE\_TYPE\_LESS\_EQUAL,

OP\_TREE\_NODE\_TYPE\_GREATER\_EQUAL,

OP\_TREE\_NODE\_TYPE\_ADDITION,

OP\_TREE\_NODE\_TYPE\_SUBTRACTION,

OP\_TREE\_NODE\_TYPE\_MULTIPLICATION,

OP\_TREE\_NODE\_TYPE\_DIVISION,

OP\_TREE\_NODE\_TYPE\_REMAINDER,

OP\_TREE\_NODE\_TYPE\_BIT\_OR,

OP\_TREE\_NODE\_TYPE\_BIT\_XOR,

OP\_TREE\_NODE\_TYPE\_BIT\_AND,

OP\_TREE\_NODE\_TYPE\_UNARY\_MINUS,

OP\_TREE\_NODE\_TYPE\_UNARY\_LOGICAL\_NOT,

OP\_TREE\_NODE\_TYPE\_UNARY\_BIT\_NOT,

OP\_TREE\_NODE\_TYPE\_FUNCTION\_CALL,

OP\_TREE\_NODE\_TYPE\_FUNC\_CALL\_ARGS,

OP\_TREE\_NODE\_TYPE\_ARRAY\_ACCESS,

OP\_TREE\_NODE\_TYPE\_ARRAY\_INDEX,

OP\_TREE\_NODE\_TYPE\_VALUE\_PLACE,

OP\_TREE\_NODE\_TYPE\_VALUE\_INT,

OP\_TREE\_NODE\_TYPE\_VALUE\_BOOL,

OP\_TREE\_NODE\_TYPE\_VALUE\_CHAR,

OP\_TREE\_NODE\_TYPE\_VALUE\_STRING,

OP\_TREE\_NODE\_TYPE\_VALUE /\* Like a general value (not operation) \*/

} OpTreeNodeType;

Структура, которая хранит потоки вызовов:

typedef struct {

int numberOfCalls;

CallerCalleePair \*head;

} CallGraph;

Структура, которая описывает какая функция какую вызывает:

typedef struct CallerCalleePair {

char\* caller;

char\* callee;

struct CallerCalleePair\* next;

} CallerCalleePair;

# Аспекты реализации

Функция, с которой начинается построение графа потока управления – *parseStatements()*. Далее рекурсивно определяется, что за утверждение, добавляется узел в граф потока управления и строится дерево операций для данного узла.

Требовало внимания утверждение repeat, так как оно отличается от других утверждений, поэтому при парсинге утверждений оно требует отдельного рассмотрения.

Для получения графа потока управления пользователь-программист может просто вызвать функцию – *getCfg()*, передав туда исходный файл и дерево синтаксического разбора. В результате он получит *CfgFile* структуру, из которой может получить/вытащить любую необходимую для себя информацию.

# Результаты

Исходный код, который проверяется:

def main(argc of int, argv of char array[5])

printf("1");

while(1)

printf("2");

while(2)

printf("3");

if(3) then

printf("break\n");

else

{

if (4) then

printf("continue\n");

else

{

printf("labb:\n");

printf("4");

}

}

end

printf("5");

if(5) then

printf("break\n");

else

{

if (6) then

printf("continue\n");

else

printf("6");

}

printf("7");

{

printf("8");

if (9) then

printf("break\n");

} while(10);

printf("11");

if(11) then

printf("break\n");

else

{

if (12) then

printf("continue\n");

else

{

printf("12");

printf("goto labb\n");

}

}

printf("13");

end

printf("14");

if (a > 3) then

{

printf("1\n");

}

else

{

printf("2\n");

printf("goto lab\n");

}

while (i < 10)

if (a > b) then

i = i + 2;

else

{

if (2 \* 3) then

printf("break\n");

else

printf("continue\n");

}

end

if (1) then

if(2) then

if(3) then

x();

else

y();

else

z();

if (a > 3) then

{

printf("1\n");

}

else

{

printf("2\n");

printf("22\n");

printf("222\n");

}

while (a < 3)

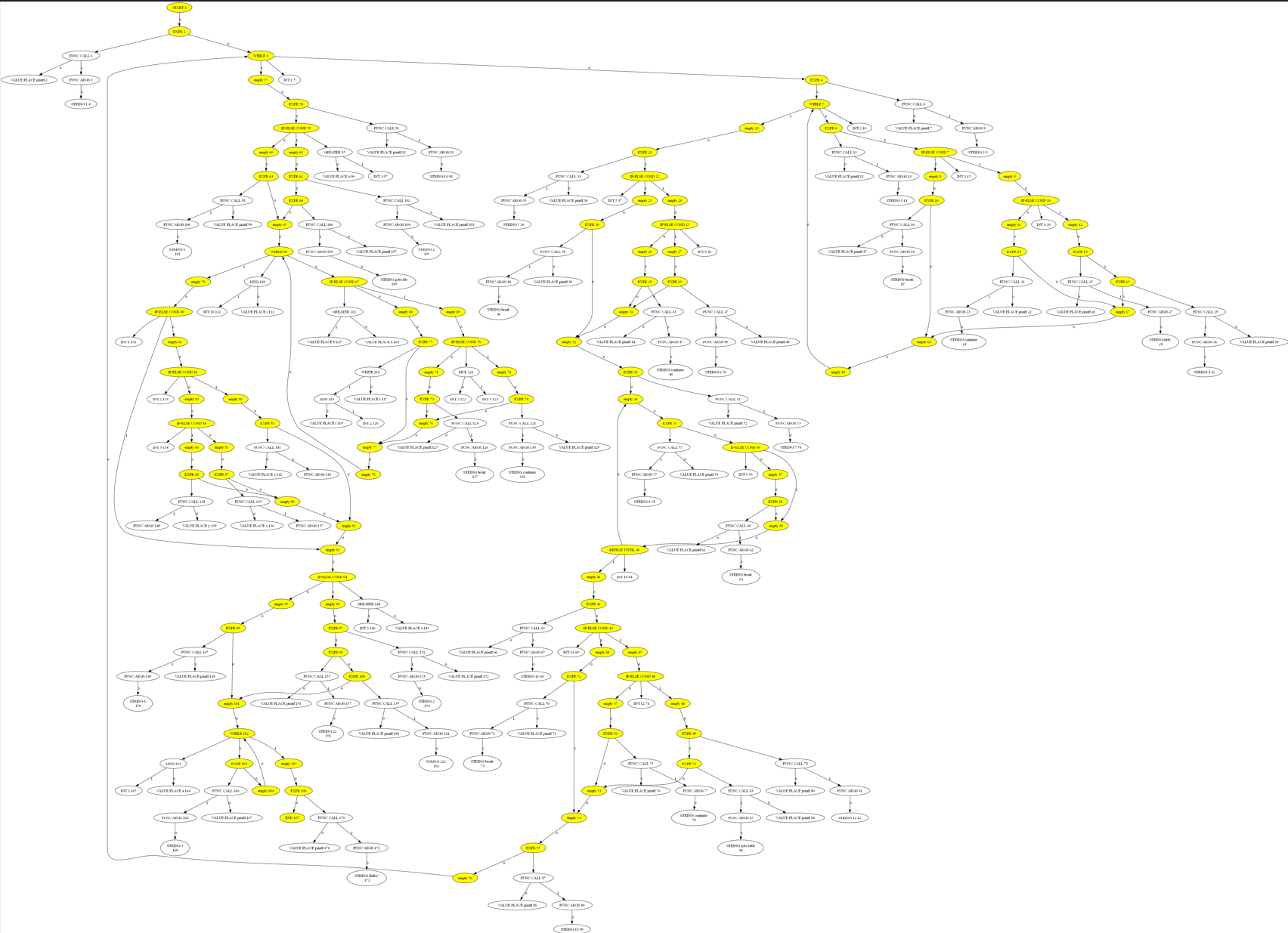
printf("3\n");

end

printf("Hello!\n");

end

В результате обработки данного исходного кода (AST и CFG модулями) получается следующий граф потока управления (<https://github.com/chetter14/my-language/blob/master/main.png>). Жёлтыми отображены узлы графа потока управления, белым – дерево операций для каждого узла.



# Выводы

В результате лабораторной работы я реализовал структуры данных и функции, которые оперируя этими структурами, постепенно строили граф потока управления и дерево операций. На выходе имеется возможность просмотреть граф потока управления и дерево операций, которые были получены. Граф потока управления и дерево операций охватывают все конструкции и инструкции, которые требуются заданием, и отображают их верным образом.