Министерство науки и образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Волгоградский государственный технический университет»

Утверждаю

Зав. кафедрой ПОАС, проф. д.т.н.

\_\_\_\_\_\_\_\_\_\_\_\_\_ Дворянкин А. М.

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г.

КАЧЕСТВО И НАДЕЖНОСТЬ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Определить интерфейс функции при ее выделении из кода программы

Описание программы

|  |  |
| --- | --- |
| СОГЛАСОВАНО:  Руководитель проекта:  Доцент кафедры ПОАС  \_\_\_\_\_\_\_\_\_\_\_\_\_ Сычев О. А.  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г. | Разработчик:  Студент ПрИн-266  \_\_\_\_\_\_\_\_\_\_\_\_ Катышев А. М.  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г. |
|  | Нормоконтролер:  Ассистент кафедры ПОАС  \_\_\_\_\_\_\_\_\_\_\_\_\_ Мамонтов Д. П.  «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2017 г. |

Волгоград, 2017 г.

Содержание

[1 Общие сведения 3](#_Toc480033870)

[2 Функциональное назначение 3](#_Toc480033871)

[3 Описание логической структуры 3](#_Toc480033872)

[4 Используемые технические средства 9](#_Toc480033873)

[5 Вызов и загрузка 9](#_Toc480033874)

[6 Входные данные 9](#_Toc480033875)

[7 Выходные данные 10](#_Toc480033876)

[Приложение А. Структуры данных, используемых в программе 11](#_Toc480033877)

[Приложение Б. Описание алгоритмов функций 18](#_Toc480033878)

[Приложение В. Дерево вызова функций 44](#_Toc480033879)

[Приложение Г. Диаграмма потоков данных 45](#_Toc480033880)

# 1 Общие сведения

Наименование программы: Function Refactor.

Для нормального функционирования программы необходима операционная система Windows XP или выше.

Программа написана на языке C++ с использованием библиотеки Qt.

# 2 Функциональное назначение

Программа Function Refactor предназначена для проведения рефакторинга «Выделение функции» данного участка кода внутри исходного кода программы. Программа должна выделять логически завершенный участок кода, расположенный внутри известных границ исходного кода программы, в отдельную функцию, при использовании которой вместо выделенного фрагмента не изменялось бы внешнее поведение всей программы.

Функциональные ограничения подробно описаны в пункте 3.1 Требования к функциональным характеристикам технического задания.

# 3 Описание логической структуры

Общий алгоритм программы:

- получение программой входных данных в виде xml-дерева разбора текста программы, границ выделяемого участка кода в отдельную функцию и имени новой выделяемой функции;

- определение наличия ошибок во входных данных (ошибки подробно описаны в приложении Г технического задания), если ошибки присутствуют, то переход к пункту 4;

- преобразование входных файлов в структуры данных, используемых в программе «Function Refactor» (см. приложение А);

- выполнение рефакторинга «Выделение функции»;

- создание файла xml-дерева разбора текста программы после проведения рефакторинга;

- вывод результирующего текста программы в новый файл формата xml с учетом проведенного рефакторинга или сообщения о невозможности проведения рефакторинга на экран.

Алгоритм основной задачи – выполнение рефакторинга «Выделение функции» (см. пункт 4 выше) представлен в приложении Б (описание функции bool performFunctionExtraction (const QVector<codeDeclaration\*> &source, QVector<codeDeclaration\*> &destination)):

Программа будет реализована на процедурном языке с использованием функций:

- bool findInputDataErrors (const QDomDocument &doc, const QStringList &data, QStringList &errorMessages) – определение наличия ошибок во входных данных, где const QDomDocument &doc – представление дерева разбора текста программы, const QStringList &data – список строк с входными данными (верхняя, нижняя границы и имя функции), QStringList &errorMessages – список сообщений об обнаруженных ошибках, возвращаемое значение – успешность обнаружения ошибки (true – обнаружена ошибка, false – ошибки не обнаружены);

- void extractProgramStructure (const QDomDocument &doc, QVector<function> &functions) – преобразование входных файлов в структуры данных, используемых в программе «Function Refactor», где doc – представление xml-дерева текста входной программы в виде QDomDocument, functions – вектор всех функций, составляющих основу входной программы;

- void buildVariable (const QDomElement &element, variable &getVariable) – формирование структурированного представления переменной, element – элемент xml-дерева, дочерние элементы которого содержат описание переменной, getVariable – структурированное представление переменной;

- void buildStruct (const QDomElement &element, structure &getStructure) – формирование структурированного представления переменной, element – элемент xml-дерева, дочерние элементы которого содержат описание переменной, getStructure – структурированное представление структуры;

- void buildFunction (const QDomElement &element, function &getFunction) – формирование структурированного представления переменной, element – элемент xml-дерева, дочерние элементы которого содержат описание переменной, getFunction – структурированное представление функции;

- void buildExpression (const QDomElement &element, expression \* expr) – формирование структуры выражения любого типа, где element – узел xml-дерева, содержащий описание выражения, expr – структурированное представление выражения;

- void buildIfElseStatement (const QDomElement &element, ifElseStmt &ifElse) – формирование структуры условного оператора if-else, где element – узел xml-дерева, содержащий описание конструкции if-else, ifElse – структурированное представление конструкции if-else;

- void buildForStatement (const QDomElement &element, forStmt &forLoop) – формирование структуры цикла for, где element – узел xml-дерева, содержащий описание конструкции for, forLoop – структурированное представление конструкции цикла for;

- void buildWhileStatement (const QDomElement &element, whileStmt &whileLoop) – формирование структуры цикла while/do-while, где element – узел xml-дерева, содержащий описание конструкции while/do-while, whileLoop – структурированное представление конструкции цикла while/do-while;

- void buildBodyStatement (const QDomElement &element, statement \* body) – формировать тело составного блока, где element – узел xml-дерева, описывающий узел тела какого-либо составного оператора, body – тело данного оператора;

- bool performFunctionExtraction (const QVector<codeDeclaration\*> &source, QVector<codeDeclaration\*> &destination) – выполнение главной задачи, где source – исходное представление разбора текста программы в удобном формате, destination – результирующий разбор текста программы после проведения рефакторинга, возвращаемое значение – успешность проведения рефакторинга (true – выделение функции произошло успешно, false – выделение функции невозможно);

- void createXmlProgramTree (const QVector<codeDeclaration\*> &source, QDomDocument &destination) – создание выходного xml-документа по итоговому дереву, где source – исходное структурированное представление дерева программы после проведения рефакторинга, destination – выходной xml-файл, содержащий дерево разбора текста программы после проведения рефакторинга;

- void variablesDefinitionInElement (const QDomElement &element, QVector<variable> &variables) – определение всех переменных, используемых внутри элемента дерева, где element – элемент xml-дерева, дочерние элементы которого содержат определение переменных; variables – вектор переменных, используемых внутри элемента дерева;

- bool findTxtDataErrors (const QStringList &txt, QStringList &errors) – определение наличия ошибок во входных данных текстового файла, где txt – список строк, содержащих данные о границах выделяемого фрагмента и имени новой выделяемой функции, errors – список ошибок, обнаруженных при анализе входных данных; возвращаемое значение – успешность обнаружения ошибок в текстовом файле (true – обнаружена хотя бы одна ошибка, false – ошибок не обнаружено);

- bool findXmlDataErrors (const QDomElement &element, QStringList &errors) – определение наличия ошибок во входных данных текстового файла, где element – элемент xml-дерева, содержащий данные о дереве разбора текста программы, errors – список ошибок, обнаруженных при анализе входных данных; возвращаемое значение – успешность обнаружения ошибок в текстовом файле (true – обнаружена хотя бы одна ошибка, false – ошибок не обнаружено);

- bool findSelectedFragment (сonst QVector<codeDeclaration\*> &code, const blockBounds bounds, QVector<statement\*> &selectedCodeFragment) – поиск выделяемого участка кода по заданным границам, где code – исходное представление программы в виде структурированного дерева, где производится поиск выделяемого фрагмента; bounds – границы искомого выделяемого фрагмента кода; selectedCodeFragment – вектор операторов исходного кода, которые входят в заданные границы; возвращаемое значение – успешность выделения фрагмента по заданным границам (true – фрагмент выделен успешно, false – по заданным границам выделить синтаксически корректный участок кода невозможно);

- void getUsedVariablesInsideCodeSection (const QVector<variable> &analyzedVariables, const QVector<statement\*> &codeSection, QVector<variable> &changingVariables, QVector<variable> &unchangingVariables) – определение использования заданных переменных в тексте программы, где analyzedVariables – переменные, подлежащие поиску внутри данного участка кода; codeSection – участок кода, где происходит поиск анализируемых переменных; changingVariables – список найденных переменных, значения которых изменяются внутри данного участка кода; unchangingVariables – список найденных переменных значения которых не изменяются внутри данного участка кода;

- void functionHeaderCreation (const function &func, QDomElement &parent) – формирование заголовка, где func – описание функции в структурированном формате; parent – элемент xml-дерева, относительно которого будет формироваться представление заголовка функции в формате xml;

- void functionBodyCreation (const QVector<statement\*> &statements, QDomElement &parent) – формирование тела новой выделяемой функции, где const statements – вектор операторов в структурированном формате; parent – элемент xml-дерева, относительно которого будет формироваться представление тело функции в формате xml;

- void functionCallCreation (const QString functionIdentifier, const QVector<variable> &argsIdentifiers, QDomElement &parent) – формирование вызова функции, где functionIdentifier – имя вызываемой функции; const QVector<variable> &argsIdentifiers – идентификаторы входных параметров; QDomElement &parent – элемент xml-дерева, относительно которого будет формироваться представление вызов функции в формате xml;

- void functionCallArgumentsCreation (QVector<QString> &identifiers, QDomElement &parent) – формирование входных параметров функции, где identifiers – идентификаторы переменных, которые будут являться входными параметрами функции с учетом порядка; parent – элемент xml-дерева, относительно которого будет формироваться представление аргументов вызова функции в формате xml;

- void formArgumentList (const QVector<variable> args, QDomElement &parent) – формирование тега списка аргументов, где args – вектор аргументов; parent – элемент дерева, относительно которого будет строиться список аргументов;

- void typeXmlCreation (type dataType, QDomElement &parent) – формирование типа данных, где dataType – тип данных в структурированном представлении; parent – элемент xml-дерева, относительно которого будет формироваться представление типа данных в формате xml.

Дерево вызовов функций приведено в приложении В.

# 4 Используемые технические средства

Для создания программы использовался компьютер с процессором Intel Core 2 Duo, тактовой частотой 2.53 ГГц, оперативной памятью 4 Гб, под управлением операционной системы Windows 7 Ultimate.

# 5 Вызов и загрузка

Для вызова данной программы необходимо:

- запустить командную строку, где вызов программы будет осуществляться из главного диска;

- ввести полный путь до запускаемой программы, имя файла с деревом разбора текста программы в формате xml и имя txt-файла с границами выделяемого фрагмента кода и именем новой выделяемой функции; все вводимые данные разделяются пробелом.

Пример вызова программы из командной строки:

FunctionRefactor.exe program.xml data.txt

# 6 Входные данные

Подробное описание входных данных приведено в пункте 3.2 Требования к надежности и приложении Б технического задания.

# 7 Выходные данные

Подробное описание входных данных приведено в пункте 3.2 Требования к надежности и приложении В технического задания.

# Приложение А

Структуры данных, используемых в программе

- структура, описывающая границы какого-либо блока:

struct blockBounds

{

QPair<int, int> start; // «координаты» начальной позиции блока

QPair<int, int> end; // «координаты» конечной позиции блока

};

- структура, описывающая тип каких-либо данных:

struct type

{

QString typeName; // имя типа

bool isConst; // признак константности

bool isPointer; // признак указателя

};

- перечисление типов возможных операций на верхнем уровне кода

enum codeDeclarationType

{

variableDeclaration, // тип объявления глобальной переменной

structDefinition, // тип описания структуры

functionDefinition // тип объявления функции

};

- класс описания объектов кода программы на верхнем уровне кода:

class codeDeclaration

{

public:

codeDeclarationType type; // тип описание кода программы на верхнем уровне

};

- структура, описывающая одну переменную:

class variable : public codeDeclaration

{

type variableType; // тип переменной

QString identifier; // имя (идентификатор) переменной

bool isInitialized; // признак наличия значения у переменной

QString value; // значение переменной

QPair<int, int> visibilityArea; // область видимости переменной в виде <начальная строка, конечная строка>

};

- структура, описывающая определение структуры:

class structure : public codeDeclaration

{

QString identifier; // имя (идентификатор) структуры

QVector<variable> member; // поля (члены) структуры

};

- перечисление, описывающие все возможные типы операций:

enum expressionType

{

identifier, // имя переменной

logic, // значение true или false

numeric, // значение цифры

character, // значение символа

string, // значение строки

assign, // присвоение

plus, // сложение

minus, // вычитание

multiply, // умножение

division, // деление

unaryPlus, // унарный плюс

unaryMinus, // унарный минус

moduloSign, // операция модуль

prefixIncrement, // префиксный инкремент

postfixIncrement, // постфиксный инкремент

prefixDecrement, // префиксный декремент

postfixDecrement, // постфиксный декремент

equal, // эквивалентность

notEqual, // неэквивалентность

lesser, // операция меньше

greater, // операция больше

lesserEqual, // операция меньше или равно

greaterEqual, // операция больше или равно

not, // логическое отрицание

and, // логическое И

or, // логическое ИЛИ

binaryNot, // побитовая инверсия

binaryAnd, // побитовое И

binaryOr, // побитовое ИЛИ

binaryXor, // побитовое исключающее ИЛИ

lestShift, // побитовый сдвиг влево

rightShift, // побитовый сдвиг вправо

plusAssign, // сложение, совмещенное с присваиванием

minusAssign, // вычитание, совмещенное с присваиванием

multiplyAssign, // умножение, совмещенное с присваиванием

divisionAssign, // деление, совмещенное с присваиванием

moduloAssign, // вычисление остатка от деления, совмещённое с присваиванием

binaryAndAssign, // побитовое И, совмещенное с присваиванием

binaryOrAssign, // побитовое ИЛИ, совмещенное с присваиванием

binaryXorAssign, // побитовое исключающее ИЛИ, совмещенное с присваиванием

leftShiftAssign, // побитовый сдвиг влево, совмещенное с присваиванием

rightShiftAssign, // побитовый сдвиг вправо, совмещенное с присваиванием

arrayAccess, // доступ к элементу массива

dereference, // непрямое обращение

takeAddress, // ссылка

propertyAccessValue, // обращение к члену структуры по значению

propertyAccessPointer, // обращение к члену структуры по указателю

functionCall, // вызов функции

variableDeclaration // инициализация объявленной переменной (для инициализации цикла for

};

- перечисление возможных составных операторов:

enum statementType

{

ifElseStatement, // составной оператор if-else

forStatement, // составной оператор for

whileStatement, // составной оператор while

expressionStatement, // выражение

variableDeclaration // объявление переменной

};

- класс представления порядка операторов:

class statement

{

public:

statementType typeStmt; // тип данного оператора

QVector<variable> externalVariables; // внешние переменные относительно данного оператора (глобальные переменные относительно описания функций)

QVector<structure> structDefinition; // вектор описания структур, приводящихся в данном составном операторе

statement \* parentStatement; // родитель данного оператора

blockBounds bounds; // границы данного оператора

};

- класс представления блока операторов:

class blockStatement : public statement

{

QVector<statement\*> blockBody; // тело данного блока операторов

}

- класс, описывающий выражение любого типа:

class expression : public statement

{

public:

expressionType type; // тип значения данного элемента выражения

QString value; // значение данного элемента выражения

expression \* leftOperand; // левый операнд выражения

expression \* rightOperand; // правый операнд выражения

};

- класс, описывающий один составной оператор if-else:

class ifElseStmt : public statement

{

public:

QVector<variable> localVariables; // локальные переменные относительно данного составного оператора

expression \* condition; // условное выражение

statement \* ifThenBlock; // блок операторов if-then части

statement \* elseBlock; // блок операторов else части

};

- класс объявления переменной, как выражения

class variableExpression : public expression

{

public:

type variableDeclarationType; // тип инициализируемой переменной

QVector<variable> declaredVariables; // объявленные переменные

};

- класс, описывающий один составной оператор цикла for:

class forStmt : public statement

{

public:

QVector<variable> localVariables; // локальные переменные относительно данного составного оператора

expression \* initialization; // инициализация цикла

expression \* condition; // условие выполнения цикла for

expression \* step; // шаг цикла for

statement \* forBlock; // блок операторов тела цикла for

};

- класс, описывающий один составной оператор цикла while/do-while:

class whileStmt : public statement

{

public:

QVector<variable> localVariables; // локальные переменные относительно данного составного оператора

bool isDoWhile; // признак цикла do-while

expression \* condition; // условие выполнения цикла while

statement \* whileBlock; // блок операторов тела цикла while

};

- класс, содержащий описание функции:

class function : public codeDeclaration

{

public:

type returnType; // тип возвращаемого значения

QString identifier; // имя (идентификатор) функции

QVector<variable> arguments; // аргументы функции

QVector<variable> localVariables; // локальные переменные функции

statement \* functionBody; // тело функции

};

# Приложение Б

Описание алгоритмов функций

Точка выполнения программы, функция main.

Входные параметры:

- int argc – количество параметров командной строки;

- char\* argv[] – параметры командной строки;

- program.xml – дерево разбора текста программы в формате xml;

- data.txt – текстовый файл, содержащий значения границ выделяемого фрагмента и имя новой выделяемой функции.

Выходные параметры:

- result.xml – дерево разбора текста программы в формате xml после проведения рефакторинга.

Алгоритм:

1. считать входные файлы (xml-дерево разбора текста программы и txt-файл с данными границ выделяемого фрагмента и именем выделяемой функции): если указано не все параметры командной строки, то вывести сообщение о некорректности ввода аргументов и корректно завершить работу; если искомых файлов с заданными именами не обнаружено, то вывести сообщение об отсутствии файлов и корректно завершить работу;
2. выполнить поиск ошибок, содержащихся во входных файлах (см. описание функции findInputDataErrors), при обнаружении ошибок необходимо выдать пользователю список обнаруженных ошибок на экран и корректно завершить работу;
3. преобразовать входное дерево из формата QDomDocument в структуры данных, используемые для анализа и проведения рефакторинга – функция extractProgramStructure;
4. провести рефакторинг «Выделение функции» заданного выделенного фрагмента кода – функция performFunctionExtraction;
5. преобразовать дерево, представленное структурами данных программы, которое получено в результате рефакторинга, в xml-формат – функция createXmlProgramTree;
6. записать полученное дерево разбора программы с учетом проведения рефакторинга в xml-файл.

Определение наличия ошибок во входных данных.

bool findInputDataErrors (const QDomDocument &doc, const QStringList &data, QStringList &errorMessages).

Входные параметры:

- const QDomDocument &doc – xml-дерево разбора программы в виде объекта QDomDocument;

- QStringList &data – список строк, считанных из txt-файла, содержащего значения границ выделяемого фрагмента кода и имя выделяемой функции.

Выходные параметры:

- QStringList &errorMessages – список ошибок, обнаруженных при анализе входных данных;

- возвращаемое значение – успешность обнаружения ошибок (true – обнаружена хотя бы одна ошибка, false – ошибок не обнаружено).

Алгоритм:

1) определить наличия ошибок в текстовом файле – функция findTxtDataErrors;

2) определить наличия ошибок в xml-дереве разбора программы – функция findXmlDataErrors;

3) вернуть признак обнаружения ошибок во входных данных.

Преобразование входных файлов в структуры данных, используемых в программе «Function Refactor».

void extractProgramStructure (const QDomDocument &doc, QVector<codeDeclaration\*> &codeStructure).

Входные параметры:

- const QDomDocument &doc – xml-дерево разбора программы в виде объекта QDomDocument.

Выходные параметры:

- QVector<codeDeclaration\*> &codeStructure – вектор описаний объявлений, составляющих внутреннее устройство входной программы;

Алгоритм:

1) выделить в xml-файле список узлов описания кода программы на верхнем уровне;

2) для каждого элемента списка полученных узлов…

2.1) если текущий элемент списка – объявление переменной, то сформировать структурированное представление объявления переменной и занести ее в результирующий вектор – функция buildVariable;

2.2) если текущий элемент списка – описание структуры, то сформировать структурированное представление описания структуры и занести ее в результирующий вектор – функция buildStruct;

2.3) если текущий элемент списка – объявление функции, то сформировать структурированное представление объявления функции и занести ее в результирующий вектор – buildFunction;

Формирование структурированного представления переменной.

void buildVariable (const QDomElement &element, variable &getVariable).

Входные параметры:

- const QDomElement &element – элемент xml-дерева, дочерние элементы которого содержат описание переменной.

Выходные параметры:

- variable &getVariable – структурированное представление переменной

Алгоритм:

1) выделить внутри текущего элемента тип переменной (тег <decl\_specifiers>);

2) выделить внутри текущего элемента имя переменной (тег <lvalue>);

3) проверить инициализацию переменной определенным значением (тег <initializer>);

4) определить область видимости переменной (текущий атрибут linestart и атрибут lineend у элемент-родителя);

Формирование структурированного представления описания структуры.

void buildStruct (const QDomElement &element, structure &getStruct).

Входные параметры:

- const QDomElement &element – элемент xml-дерева, дочерние элементы которого содержат описание структуры.

Выходные параметры:

- structure &getStruct – структурированное представление структуры.

Алгоритм:

1) выделить внутри текущего элемента имя структуры (тег <struct\_header>);

2) выделить список узлов-объявлений полей структуры (тег <structure\_components\_list>);

3) для каждого выделенного узла…

3.1) сформировать структурированное представление объявления переменной и занести ее в результирующий вектор полей структуры – функция buildVariable;

Формирование структурированного представления объявления функции.

void buildFunction (const QDomElement &element, function &getFunction).

Входные параметры:

- const QDomElement &element – элемент xml-дерева, дочерние элементы которого содержат описание функции.

Выходные параметры:

- function &getFunction – структурированное представление функции.

Алгоритм:

1) сформировать тип возвращаемого значения текущей функции (тег <decl\_specifiers>);

2) сформировать имя текущей функции (тег <lvalue>);

3) сформировать аргументы текущей функции – функция variablesDefinitionInElement;

4) сформировать локальные переменные относительно текущей функции – функция variablesDefinitionInElement;

5) для каждого оператора в теле текущей функции…

5.1) сформировать структуру тела функции – функция buildBodyStatement.

Формирование тела составного блока.

void buildBodyStatement (const QDomElement &element, statement \* body).

Входные параметры:

- const QDomElement &element – элемент xml-дерева, дочерние элементы которого содержат тело составного блока.

Выходные параметры:

- statement \* body – тело составного оператора в структурированном представлении.

Алгоритм:

1) сформировать внешние и локальные переменные (для блока описания функции внешние переменные будут представленны глобальными переменными) – функция variablesDefinitionInElement;

2) если данный оператор является выражением, то сформировать выражение в виде структуры expression – функция buildExpression;

3) если данный оператор является условным оператором, то сформировать блок операторов if-else в виде структуры ifElseStmt – функция buildIfElseStatement;

4) если данный оператор является циклом for, то сформировать блок цикла for в виде структуры forStmt – функция buildForStatement;

5) если данный оператор является циклом while или do-while, то сформировать блок данного цикла в виде структуры whileStmt – функция buildWhileStatement;

Формирование выражения:

void buildExpression (const QDomElement &element, expression \* expr).

Входные параметры:

- const QDomElement &element – элемент xml-дерева, дочерние элементы которого содержат описание выражения.

Выходные параметры:

- expression \* expr – описание выражения в структурированном представлении.

Алгоритм:

1) если левый операнд выражения в формате xml не является вложенным выражением (данный узел дерева является одним из тегов: <identifier>, <numeric>, <character>, <string>), то сформировать значение левого операнда текущего выражения значением из дерева (по атрибуту value);

2) иначе сформировать тип и значения операции для текущего узла выражения и сформировать выражение относительно левого операнда текущего выражения, а значение атрибута value тега второго ребенка присвоить типу текущей операции;

3) если правый операнд выражения в формате xml не является вложенным выражением (данный узел дерева является одним из тегов: <identifier>, <numeric>, <character>, <string>), то сформировать значение правого операнда текущего выражения значением из дерева (по атрибуту value);

4) иначе сформировать выражение относительно правого операнда текущего выражения, а значение атрибута value тега второго ребенка присвоить типу текущей операции.

Формирование структуры условного оператора if-else.

void buildIfElseStatement (const QDomElement &element, ifElseStmt &ifElse).

Входные параметры:

- const QDomElement &element – элемент xml-дерева, дочерние элементы которого содержат описание составного оператора if-else.

Выходные параметры:

- ifElseStmt &ifElse – описание составного оператора if-else в структурированном представлении.

Алгоритм:

1) сформировать условное выражение данного блока if-else – функция buildExpression;

2) для каждого оператора в теле блока if …

2.1) сформировать тело блока if – функция buildBodyStatement;

3) если встретилась конструкция else if, то сформировать вложенную структуру условного оператора if-else – функция buildIfElseStatement;

4) для каждого оператора в теле блока else …

4.1) сформировать тело блока else – функция buildBodyStatement.

Формирование структуры цикла for.

void buildForStatement (const QDomElement &element, forStmt &forLoop).

Входные параметры:

- const QDomElement &element – элемент xml-дерева, дочерние элементы которого содержат описание составного оператора цикла for.

Выходные параметры:

- forStmt &forLoop – описание составного оператора цикла for в структурированном представлении.

Алгоритм:

1) сформировать инициализацию цикла for;

2) сформировать выражение условия продолжения цикла for;

3) для каждого оператора в теле блока for …

3.1) сформировать тело блока for – функция buildBodyStatement;

Формирование структуры цикла while/do-while.

void buildWhileStatement (const QDomElement &element, whileStmt &whileLoop).

Входные параметры:

- const QDomElement &element – элемент xml-дерева, дочерние элементы которого содержат описание составного оператора цикла while.

Выходные параметры:

- whileStmt &whileLoop – описание составного оператора цикла while в структурированном представлении.

Алгоритм:

1) определить тип цикла;

2) сформировать выражение условия продолжения цикла while;

3) для каждого оператора в теле блока цикла …

3.1) сформировать тело блока цикла – функция buildBodyStatement;

Выполнение главной задачи.

bool performFunctionExtraction (const QVector<codeDeclaration\*> &source, QVector<codeDeclaration\*> &destination).

Входные параметры:

- QVector<codeDeclaration\*> &source – представление исходного дерева программы в структурированном виде;

Выходные параметры:

- QVector<codeDeclaration\*> &destination – представление результирующего дерева программы в структурированном виде после проведения рефакторинга.

Алгоритм:

1) провести поиск участка кода, который необходимо выделить в отдельную функцию, по заданным границам – функция findSelectedFragment;

2) если синтаксически корректно завершенный фрагмент кода выделить не удалось по заданным границам (заданный пользователем границы «разрывают» простой оператор), то вернуть неуспешный результат проведения рефакторинга;

3) определить переменные, участвующие в выделенном фрагменте кода – функция getUsedVariablesInsideCodeSection;

4) пока текущий родительский составной оператор не является функцией относительно текущего оператора…

4.1) если родительский оператор является циклом…

4.1.1) определить переменные, используемые после текущего оператора, но внутри тела цикла – функция getUsedVariablesInsideCodeSection;

4.1.2) определить переменные, используемые до текущего оператора, но внутри тела цикла – функция getUsedVariablesInsideCodeSection;

4.1.3) выходными параметрами функции будут являться переменные, которые:

- переписываются внутри выделенного фрагмента;

- используются (не изменяют значения) до выделенного фрагмента, но внутри тела цикла;

- используются (не изменяют значения) после выделенного фрагмента, но внутри тела цикла;

4.1.4) входными параметрами функции будут являться переменные, которые:

- переписываются до выделенного фрагмента, но внутри тела цикла;

- переписываются после выделенного фрагмента, но внутри тела цикла;

4.2) если родительский оператор является условным оператором, родителем которого является цикл…

4.2.1) определить переменные, используемые в смежных условных блоках if, else-if и/или else – функция getUsedVariablesInsideCodeSection;

4.3) перейти к родительскому оператору для продолжения анализа;

5) определить, какие из этих переменных используются до и после выделенного фрагмента в тексте программы – функция getUsedVariablesInsideCodeSection;

6) если переменная, используемая в выделенном фрагменте, не имела никакого значения до данного фрагмента (не была проинициализирована), то объявление переменной будет являться телом новой функции (входным параметром являться она не будет);

7) если выходной параметр функции только один, то тип данного параметра будет являться возвращаемым значением новой функции;

8) если выделенный фрагмент кода содержит локальные объявления типа, то скопировать их описание в новую функцию;

9) если выделенный фрагмент кода содержит оператор [return <выражение>], который является частью какого-либо условия, то тип возвращаемого значения выделяемой функции – bool, оператор [return <выражение>] заменяется оператором [return true] внутри тела выделяемой функции, а в тексте программы выделяемый фрагмент заменяется на условный оператор if(<вызов\_функции>), тело которого будет [return <выражение>];

10) в противном случае (если выделенный фрагмент кода содержит оператор [return <выражение>], который не является частью какого-либо условного выражения) тип возвращаемого значения выделяемой функции – void, тело функции состоит только из операторов, которые определены до вызова оператора [return <выражение>], а в тексте программы выделяемый фрагмент заменяется на вызов функции внутри исходного оператора return (return <вызов выделенной функции>);

11) построить структурированное представление новой выделяемой функции;

12) заменить выделенный фрагмент на вызов новой функции с необходимыми параметрами…

12.1) заменить выделенный фрагмент из кода программы структурированным представлением выражения;

12.2) если функция имеет возвращаемое значение, то левый операнд выражения будет являться изменяемой переменной, к которой приравнивается результат работы функции;

12.3) добавить тип к переменной, если она не была проинициализирована в коде ранее.

Создание выходного xml-документа по итоговому дереву.

void createXmlProgramTree (const QVector<codeDeclaration\*> &source, QDomDocument &destination).

Входные данные:

- const QVector<codeDeclaration\*> &source – дерево программы в структурированном виде после проведения рефакторинга, из которого будет строиться дерево программы в xml-формате;

Выходные данные:

- QDomDocument &destination – дерево представления программы в формате xml.

Алгоритм:

1) формировать заголовка новой выделяемой функции – функция functionHeaderCreation;

2) формировать тела новой выделяемой функции – функция functionBodyCreation;

3) формировать вызова новой выделенной функции в тексте программы вместо выделенного фрагмента (все теги строятся в соответствии с выделенным структурированным деревом) – functionCallCreation.

Формирование тега списка аргументов.

void formArgumentList (const QVector<variable> args, QDomElement &parent).

Входные параметры:

- const QVector<variable> args – вектор аргументов.

Обновляемые параметры:

- QDomElement &parent – элемент дерева, относительно которого будет строиться список аргументов.

Алгоритм:

1) формировать тега списка аргументов функции <arg\_list> (вложен в тег <formal\_args\_list>);

2) для всех аргументов функции:

2.1) формировать тег описания текущего аргумента <arg> (вложен в тег <arg\_list>);

2.2) формировать тега описания сложного типа данных <decl\_specifiers> (вложен в тег <arg>);

2.3) формировать тега типа текущего аргумента – функция typeDefinition:

2.4) формировать тега l-value типа выражения <lvalue> (вложен в тег <arg>);

2.5) если l-value выражение передается по указателю, то формировать тег тип возвращаемого значения <operator\_token…/> с атрибутом value равным "\*" (вложен в тег <lvalue>);

2.6) формировать тега идентификатора/имени аргумента <identifier> (вложен в тег <lvalue>);

2.7) если текущий аргумент функции не последний, то формировать тег разделения аргументов <comma … /> с атрибутом value равным "," (вложен в тег <arg\_list>).

Определение всех переменных, используемых внутри элемента дерева.

void variablesDefinitionInElement (const QDomElement &element, QVector<variable> &variables).

Входные параметры:

- const QDomElement &element – элемент xml-дерева, дочерние элементы которого содержат определение переменных.

Выходные параметры:

- QVector<variable> &variables – вектор переменных, используемых внутри элемента дерева.

Алгоритм:

1) получение списка всех потомков текущего узла;

2) для каждого элемента из списка выделенных объявлений переменных:

2.1) получить структурированное представление текущей переменной – функция buildVariable;

2.2) занести текущие значения в вектор variables.

Определение наличия ошибок во входных данных текстового файла.

bool findTxtDataErrors (const QStringList &txt, QStringList &errors).

Входные параметры:

- const QStringList &txt – список строк, содержащих данные о границах выделяемого фрагмента и имени новой выделяемой функции.

Выходные параметры:

- QStringList &errors – список ошибок, обнаруженных при анализе входных данных;

- возвращаемое значение – успешность обнаружения ошибок в текстовом файле (true – обнаружена хотя бы одна ошибка, false – ошибок не обнаружено).

Алгоритм:

1) если в первой строке некорректно составлен формат верхней границы выделяемого фрагмента (в соответствии с приложением Г технического задания), то занести в список ошибок описание ошибки нарушения формата верхней границы в виде строки QString (сообщения об ошибках приведены в приложении Г технического задания);

2) если во второй строке некорректно составлен формат нижней границы выделяемого фрагмента, то занести в список ошибок описание ошибки нарушения формата нижней границы в виде строки QString;

3) если в третьей строке некорректно описано имя функции, то занести в список ошибок описание ошибки нарушения составления имени функции в виде строки QString;

4) вернуть успешность нахождения ошибки в текстовом файле.

Определение наличия ошибок во xml-дереве разбора текста программы.

bool findXmlDataErrors (const QDomElement &element, QStringList &errors).

Входные параметры:

- const QDomElement &element – элемент xml-дерева, содержащий данные о дереве разбора текста программы.

Выходные параметры:

- QStringList &errors – список ошибок, обнаруженных при анализе входных данных;

- возвращаемое значение – успешность обнаружения ошибок в текстовом файле (true – обнаружена хотя бы одна ошибка, false – ошибок не обнаружено).

Алгоритм:

1) для текущего элемента дерева разбора текста программы проверить наличия ошибок в каждом дочернем элементе (корректность вложенности тегов в каждый определенный тег приведен в приложении Г технического задания: таблица 3);

2) если текущий элемент имеет некорректную вложенность тегов, то занести в список ошибок описание ошибки нарушения вложенности в текущий тег в виде строки QString;

3) перейти к элементу-брату относительно текущего элемента;

4) вернуть успешность нахождения ошибки в текстовом файле.

Поиск выделяемого участка кода по заданным границам.

bool findSelectedFragment (сonst QVector<codeDeclaration\*> &code, const blockBounds bounds, QVector<statement\*> &selectedCodeFragment).

Входные параметры:

- сonst QVector<codeDeclaration\*> &code – исходное представление программы в виде структурированного дерева, где производится поиск выделяемого фрагмента;

- const blockBounds bounds – границы искомого выделяемого фрагмента кода;

Выходные параметры:

- QVector<statement\*> &selectedCodeFragment – вектор операторов исходного кода, которые входят в заданные границы;

- возвращаемое значение – успешность выделения фрагмента по заданным границам (true – фрагмент выделен успешно, false – по заданным границам выделить синтаксически корректный участок кода невозможно).

Алгоритм:

1) выделить начало выделяемого фрагмента:

1.1) для каждого элемента дерева…

1.1.1) найти стартовую позицию строки текущего оператора дерева;

1.1.2) если заданная пользователем стартовая строка совпадает со стартовой строкой оператора и стартовый столбец совпадает со стартовым столбцом оператором, то…

1.1.2.1) проверить конечные границы начального оператора: если конечная строка начального оператора меньше заданной, то определить все последующие операторы, как часть выделяемого фрагмента кода, пока конечные границы какого-либо последующего оператора не совпадет с заданными конечными границами.

Определение использования заданных переменных в тексте программы

void getUsedVariablesInsideCodeSection (const QVector<variable> &analyzedVariables, const QVector<statement\*> &codeSection, QVector<variable> &changingVariables, QVector<variable> &unchangingVariables).

Входные параметры:

- const QVector<variable> &analyzedVariables – переменные, подлежащие поиску внутри данного участка кода;

- const QVector<statement\*> &codeSection – участок кода, где происходит поиск анализируемых переменных.

Выходные параметры:

- QVector<variable> &changingVariables – список найденных переменных, значения которых изменяются внутри данного участка кода;

- QVector<variable> &unchangingVariables – список найденных переменных значения которых не изменяются внутри данного участка кода.

Алгоритм:

1) для каждого оператора в участке кода…

1.1) найти использование любой переменной из списка analyzedVariables в текущем операторе (левые операнды типа identifier, к которым применены операции assign, prefixIncrement, postfixIncrement, prefixDecrement, postfixDecrement, plusAssign, minusAssign, multiplyAssign, divisionAssign, moduloAssign, binaryAndAssign, binaryOrAssign, binaryXorAssign, leftShiftAssign, rightShiftAssign);

1.2) если найдена изменяемая переменная, то занести ее в список переменных, изменяющих свое значение;

1.3) иначе занести ее в список переменных, не изменяющих свое значение.

Формирование заголовка.

void functionHeaderCreation (const function& func, QDomElement &parent).

Входные параметры:

- const function& func – описание функции в структурированном формате.

Обновляемые параметры:

- QDomElement &parent – элемент xml-дерева, относительно которого будет формироваться представление типа данных в формате xml.

Алгоритм:

1) формировать тег определения функции <function\_definition> (вложен в корневой тег <program>);

2) формировать тип возвращаемого значения (вложен в тег <function\_definition>) – функция typeDefinition;

3) формировать тег l-value типа выражения <lvalue> (вложен в тег <function\_definition>);

4) если l-value выражение передается по указателю, то формировать тег типа возвращаемого значения <operator\_token…/> с атрибутом value равным "\*" (вложен в тег <lvalue>);

5) формировать тег идентификатора/имени функции <identifier> (вложен в тег <lvalue>);

6) формировать тег блока аргументов функции <formal\_args\_list> (вложен в тег <function\_definition>);

7) формировать тег открывающей скобки <bracket … /> с атрибутом value равным "(" (вложен в тег <formal\_args\_list>);

8) формировать тег списка аргументов <arg\_list> (вложен в тег<formal\_args\_list>) – функция formArgumentList;

9) формировать тег закрывающей скобки <bracket … /> с атрибутом value равным ")" (вложен в тег <formal\_args\_list>);

Формирование тела составного оператора.

void functionBodyCreation (const QVector<statement\*> &statements, QDomElement &parent).

Входные параметры:

- const QVector<statement\*> &statements – вектор операторов в структурированном формате.

Обновляемые параметры:

- QDomElement &parent – элемент xml-дерева, относительно которого будет формироваться представление типа данных в формате xml.

Алгоритм:

1) формировать тег тела функции <function\_body> (вложен в тег <function\_definition>);

2) формировать тег открывающей скобки <bracket … /> с атрибутом value равным "{" (вложен в тег <function\_body>);

3) формировать тег списка операторов функции <stmt\_list> (вложен в тег <function\_body>);

4) формировать списка операторов тела функции (перечень тегов в соответствии с возможными выражениями, их атрибутов приведен в приложении Г технического задания), может вызываться рекурсивно;

5) формировать тег закрывающей скобки <bracket … /> с атрибутом value равным "}" (вложен в тег <function\_body>);

Формирование вызова функции

void functionCallCreation (const QString functionIdentifier, const QVector<variable> &argsIdentifiers, QDomElement &parent).

Входные параметры:

- const QString functionIdentifier – имя вызываемой функции;

- const QVector<variable> &argsIdentifiers – идентификаторы входных параметров.

Обновляемые параметры:

- QDomElement &parent – элемент xml-дерева, относительно которого будет формироваться представление типа данных в формате xml.

Алгоритм:

1) формировать тег имени функции <identifier> (вложен в тег <expr\_function\_call>);

2) формировать тег открывающей скобки <bracket … /> с атрибутом value равным "(" (вложен в тег <expr\_function\_call>);

3) формировать входные параметры функции, если параметров не менее двух – функция functionCallArgumentsCreation;

4) формировать тег идентификатора входного параметра <identifier>, если входной параметр только один (вложен в тег <expr\_function\_call>);

Формирование входных параметров функции.

void functionCallArgumentsCreation (QVector<variable> &identifiers, QDomElement &parent).

Входные параметры:

- const QVector<variable> identifiers – идентификаторы переменных, которые будут являться входными параметрами функции с учетом порядка.

Обновляемые параметры:

- QDomElement &parent – элемент xml-дерева, относительно которого будет формироваться представление типа данных в формате xml.

Алгоритм:

1) если входных параметров больше двух, то формировать входные параметры функции с данными identifiers без последнего идентификатора в списке;

2) формировать тег разделения аргументов <comma … /> с атрибутом value равным "," (вложен в тег <expr\_list>);

3) формировать тег идентификатора последнего входного параметра <identifier>, (вложен в тег <expr\_list>);

Формирование типа данных.

void typeXmlCreation (type dataType, QDomElement &parent).

Входные параметры:

- type dataType – тип данных в структурированном представлении.

Обновляемые параметры:

- QDomElement &parent – элемент xml-дерева, относительно которого будет формироваться представление типа данных в формате xml.

Алгоритм:

1) формировать тег описания сложного типа данных <decl\_specifiers> (вложен в тег элемента parent);

2) формировать тег возвращаемого значения:

2.1) если тип возвращаемого значения является константным, то формировать тег ключевого слова <keyword … /> с атрибутом равным "bool" (вложен в тег <decl\_specifiers>);

2.2) если тип возвращаемого значения новой функции – bool, то формировать тег идентификатора bool <identifier> с атрибутом value равным "bool" (вложен в тег <decl\_specifiers>);

2.3) иначе:

2.3.1) формировать тег общего типа <builtintype> (вложен в тег <decl\_specifiers>);

2.3.2) формировать тег типа возвращаемого значения <typename … /> с атрибутом value равным значению типа возвращаемого значения функцией (вложен в тег <builtintype>).

# Приложение В

Дерево вызова функций

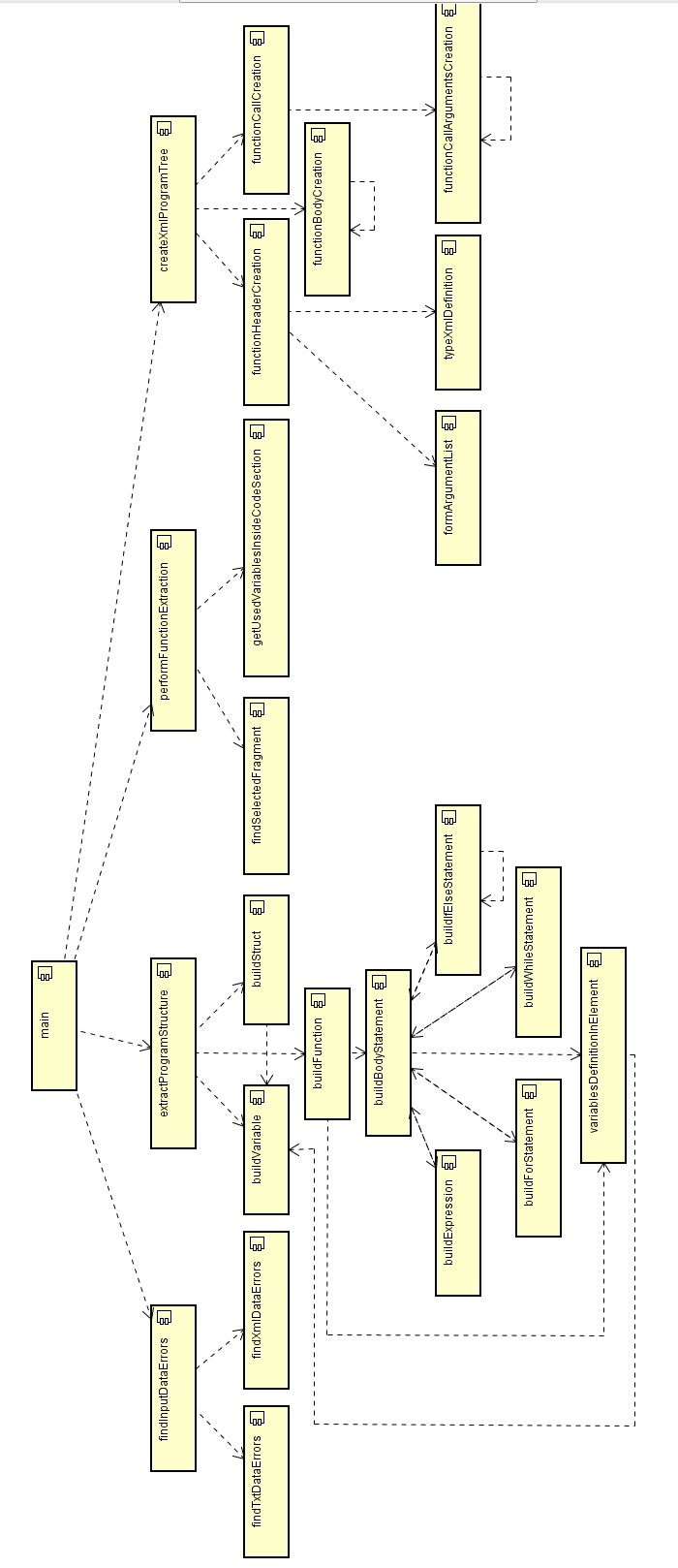


Рисунок 1. Дерево вызова функций

# Приложение Г

Диаграмма потоков данных

Выход

Файл дерева программы в xml-формате с проведенным рефакторингом

result.xml

Структурированное представление дерева программы с учетом проведенного рефакторинга

Рисунок 2. Диаграмма потоков данных

Структурированное представление дерева программы

(см. приложение А)

- файл дерева программы в xml-формате

program.xml

- файл данных границ и имени функции в формате txt

data.txt

Вход