Санкт-Петербургский государственный университет

Кафедра системного программирования

Группа 22.Б07-мм

Расширение рефакторинга средствами Clang

Десятников Павел Викторович

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ст. преп. каф. ИАС, К. К. Смирнов

Консультант:
старший разработчик ПО I категории, ООО «Софтком», П. А. Бабанов

Санкт-Петербург
2024

Оглавление

Введение **3**

1. Постановка задачи **4**

2. Обзор **5**

2.1. Clang Refactoring Engine 5

2.2. Eclipse IDE 6

3. Реализация **7**

3.1. Тестовое преобразование 7

3.2. Встраивание преобразования в IDE 9

Заключение **12**

Список литературы **13**

Введение

В современном мире рефакторинг является важным этапом разработки ПО, он позволяет улучшить читабельность и облегчить обслуживание кода, а в некоторых случаях также устранить ошибки в программе. Во многих современных IDE[1], таких как Visual Studio [5], CLion [1], Eclipse CDT [3] существуют встроенные инструменты, осуществляющие рефакторинг кода на языках C/C++, или расширения с такими инструментами.

В ООО «Софтком» ведётся разработка IDE на базе Eclipse. В этой среде используется инфраструктура LLVM [4] для работы с исходным кодом на языках C/C++. В процессе разработки был создан собственный статический анализатор, который используется для поиска ошибок и замечаний в коде. Также в IDE встроены сторонние анализаторы, такие как clang-check и clang static analyzer.

Заказчиком был составлен перечень типичных ошибок (замечаний), распознаваемых в том числе ранее упомянутыми встроенными инструментами. После выявления проблем генерируются соответствующие маркеры внутри IDE.

Требуется реализовать преобразования, с помощью которых можно эффективно разрешать полученные маркеры ошибок и замечаний. В данной работе будет реализовано тестовое преобразование с использованием Clang Refactoring Engine [2], далее оно будет встроено в IDE и будет вызываться через графический интерфейс приложения.

[1]Интегрированная среда разработки (англ. Integrated Development Environment).

1. Постановка задачи

Целью работы является интеграция в IDE тестового рефакторинга для типичной ошибки из предоставленного перечня.

Для её выполнения были поставлены следующие задачи:

1. сделать обзор используемых инструментов;
2. выбрать одну ошибку из списка типичных и реализовать её исправление;
3. интегрировать полученное преобразование в IDE.

2. Обзор

2.1. Clang Refactoring Engine

В данной работе для расширения рефакторинга кода используется Clang Refactoring Engine — фреймворк, позволяющий разработчику добавлять собственные преобразования, которые можно исполнять из консоли с помощью команды из листинга 1, а также интегрировать в IDE.

Листинг 1: Вызов команды рефакторинга из консоли.

```
clang-refactor <subcommand> <filepath> -p <build path> <optional
    arguments>
```

В Clang Refactoring Engine сущестувуют две основные абстракции (классы для наследования), отвечающие за добавление команд рефакторинга:

1. RefactoringAction — высокоуровневое описание команды рефакторинга;

2. RefactoringActionRule — её более низкоуровневая реализация.

Одной высокоуровневой команде может соответствовать несколько правил двух типов: SourceChangeRefactoringRule для изменений исходного кода внутри одной единицы трансляции и FindSymbolOccurrencesRefactoringRule для поиска вхождений определённых символов. Правила должны быть реализованы так, чтобы пользователь мог понять, какое действие было исполнено, несмотря на то, какое правило было применено.

Для определения того, может ли конкретное правило быть применено в данной ситуации, должны быть удовлетворены требования, задающиеся с помощью класса RefactoringActionRuleRequirement. Существуют также и опциональные требования — класс OptionRequirement.

В случае успешного исполнения изменённый код выводится в stdout, иначе выводится ошибка.

2.2. Eclipse IDE

IDE, разрабатываемое в «Софткоме», основано на Eclipse и предназначено для работы с кодом на языках C/C++. Основная часть исходного кода написана на языке Java.

Внутри Eclipse есть необходимые инструменты для генерации и разрешения собственных маркеров. Эти маркеры обладают набором базовых атрибутов, таких как:

- ресурс (файл);
- номер строки;
- уровень серьёзности проблемы (ошибка, замечание, информация).

Также есть возможность добавления любого количества собственных атрибутов. Этот факт открывает большие возможности для реализации разрешения маркеров.

Для вызовов долгих процессов, работающих в ядре IDE (независимо от графического интерфейса), используется внутренний фреймворк Jobs. Через него будет реализован вызов clang-refactor с нужными аргументами, а также удаление маркера и перезапись файла в случае успешного исполнения.

3. Реализация

3.1. Тестовое преобразование

В данном подразделе будет описано тестовое преобразование, реализованное с помощью Clang Refactoring Engine.

3.1.1. Выбор ошибки для исправления

Заказчиком был предоставлен список из более чем 50 типичных ошибок. Автору этой работы наиболее привлекательной и нетривиальной для реализации исправления показалась ошибка типа «Использование неинициализированной примитивной переменной». Несмотря на кажущуюся простоту и очевидность разрешения этой проблемы (а именно, инициализации такой переменной стандартным значением — нулём), на деле присутствует множество тонкостей.

3.1.2. Реализация преобразования

Так как требуется, чтобы данное преобразование позже встраивалось в IDE, очень важно на начальном этапе понять, какие данные подаются на вход. В данном случае, передаются файл и конкретное место[2] в исходном коде, где должно присутствовать обращение к неинициализированной переменной, а также необходимо получить папку сборки, в которой обязательно должен присутствовать файл compile_commands.json[3]. В случае его отсутствия следует внести в корневой CMakeLists.txt содержимое листинга 2.

Листинг 2: Включение генерации compile_commands.json

```
set(CMAKE_EXPORT_COMPILE_COMMANDS ON)
```

Подкоманда для исполнения данного преобразования была названа «init-primitives», она содержится в высокоуровневом описании преобразования (класс для наследования — RefactoringAction). В нём — единствен-

[2]Его формат — <файл>:<строка>:<столбец>, далее — SourceLocation.

[3]Его наличие позволяет корректно компилировать переданный файл и строить абстрактное синтаксическое дерево Clang (далее — ClangAST).

ное правило (RefactoringActionRule), принимающее в качестве параметра указатель на VarDecl — узел ClangAST, отвечающий за определение проблемной неинициализированной переменной, и выполняющее нужное преобразование исходного кода.

На листингах ниже представлены пример простого кода и построенное абстрактное синтаксическое дерево Clang[4].

Листинг 3: Пример кода с использованием неинициализированной переменной.

```
int main() {
    int a;
    a++;
    return 0;
}
```

Листинг 4: Построенное ClangAST для примера выше.

```
TranslationUnitDecl
`-FunctionDecl <line:2:1, line:7:1> line:2:5 main 'int ()'
  `-CompoundStmt <line:3:1, line:7:1>
    |-DeclStmt <line:4:5, col:10>
    | `-VarDecl <col:5, col:9> col:9 used a 'int'
    |-UnaryOperator <line:5:5, col:6> 'int' postfix '++'
    | `-DeclRefExpr <col:5> 'int' lvalue Var 0xcc99460 'a' 'int'
    `-ReturnStmt <line:6:5, col:12>
      `-IntegerLiteral <col:12> 'int' 0
```

Самая содержательная часть работы состоит в реализации нахождения требуемого VarDecl из переданной строки кода и проверки его на соответствие требованиям, таким как отсутствие инициализации, примитивность типа.

С этой целью был реализован PrimitiveVarDeclRequirement (собственный класс, наследуемый от SourceLocationRequirement). Это требование передаётся в качестве параметра при конструировании правила. Логика требования проста: запускается поиск по построенному ClangAST, происходит поиск DeclRefExpr (см. листинг 4) — узел, представляющий собой выражение обращения объявлению (в данном случае к нужной

[4]Для генерации и анализа ClangAST использовался интернет-источник Compiler Explorer.

переменной). Далее выполняется поиск соответствующего VarDecl по полученному выражению. В конце выполняются проверки полученного узла на отсутствие инициализации и примитивность типа. Если какой-либо этап не был выполнен успешно, то возвращается ошибка, и преобразование не будет исполнено.

3.1.3. Расширение Clang Refactoring Engine

Как оказалось, в самом Clang Refactoring Engine отсутствует важная функциональность для данной работы. Так, в нём есть SourceRangeRequirement[5], а также способ передачи SourceRange как аргумента при вызове clang-refactor, но отсутствуют аналогичные возможности для SourceLocation.

В связи с этим для корректной работы полученного преобразования потребовалось добавить недостающую функциональность. Данную часть работы практически полностью выполнил Сергеев Игнатий, коллега автора, он занимался реализацией большой части преобразований, требуемых заказчиком. Автор лишь внёс несколько небольших изменений в эту часть кода.

3.2. Встраивание преобразования в IDE

Нужный в работе маркер генерируется при любом использовании неинициализированной переменной, будь то инкремент, проверка в условии или что-то иное. По умолчанию маркер имеет базовый набор атрибутов: ресурс (файл), номер строки, уровень серьёзности проблемы (ошибка, замечание, информация). Этого недостаточно для вызова преобразования: хоть по ресурсу легко узнать папку сборки проекта, SourceLocation отсутствует.

Чтобы понять, откуда взять этот недостающий атрибут, достаточно заметить, что все они находятся из строки вывода clang-check (именно он генерирует этот маркер). Его вывод в stdout можно видеть на листинге

[5]SourceRange — тривиальный кортеж их двух SourceLocation, является представлением выделенной области в файле.

5.

Листинг 5: Вывод clang-check.

```
C:\Users\desya\projects\lib\lib1.c:3:2: warning: The expression is an
   uninitialized value. The computed value will also be garbage [core.
   uninitialized.Assign]
    3 | a++;
      | ^~~
1 warning generated.
```

Обработчик разбивает строку вывода на части по двоеточию, первые четыре в точности составляют SourceLocation в нужном формате, поэтому их конкатенация была добавлена в качестве атрибута маркера.

В Eclipse существует интерфейс IMarkerResolutionGenerator, классы, его реализующие должны содержать метод, возвращающий нужные исправления (IMarkerResolution).

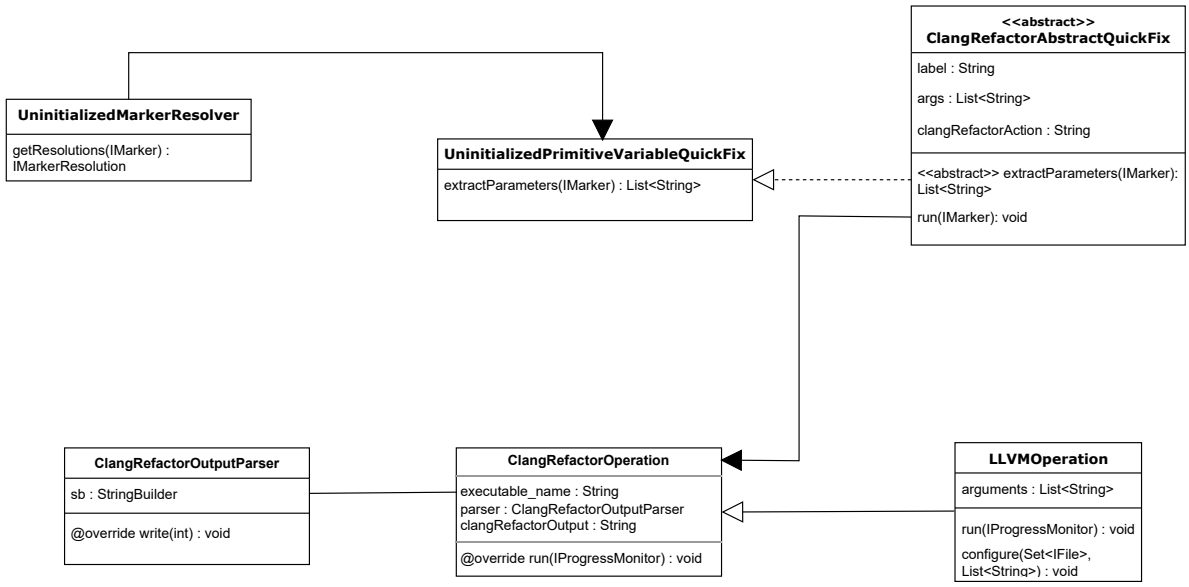

Рис. 1: Архитектура решения.

Была придумана такая архитектура, представленная на рисунке 1: абстрактный класс ClangRefactorQuickFix реализует интерфейс IMarkerResolution, его метод run конструирует ClangRefactorOperation, передавая ему список нужных аргументов[6]. Для того, чтобы пользо-

[6]Классы-наследники реализуют метод extractParameters, который выделяет из маркера нужные атрибуты.

вательский интерфейс не зависал во время работы clang-refactor, этот процесс выполняется в ядре IDE с использованием фреймворка Job. Если clang-refactor завершил свою работу успешно, то файл перезаписывается, а маркер удаляется.

Важным моментом является то, что были также реализованы правила, блокирующие чтение и запись файла другими потоками. Сделано это для исключения ошибок вида «потерянная запись», когда два потока считали файл, преобразовали, и одно из двух изменений было потеряно.

Ниже представлено окно выбора нужного преобразования, применимого к определённому типу маркера.

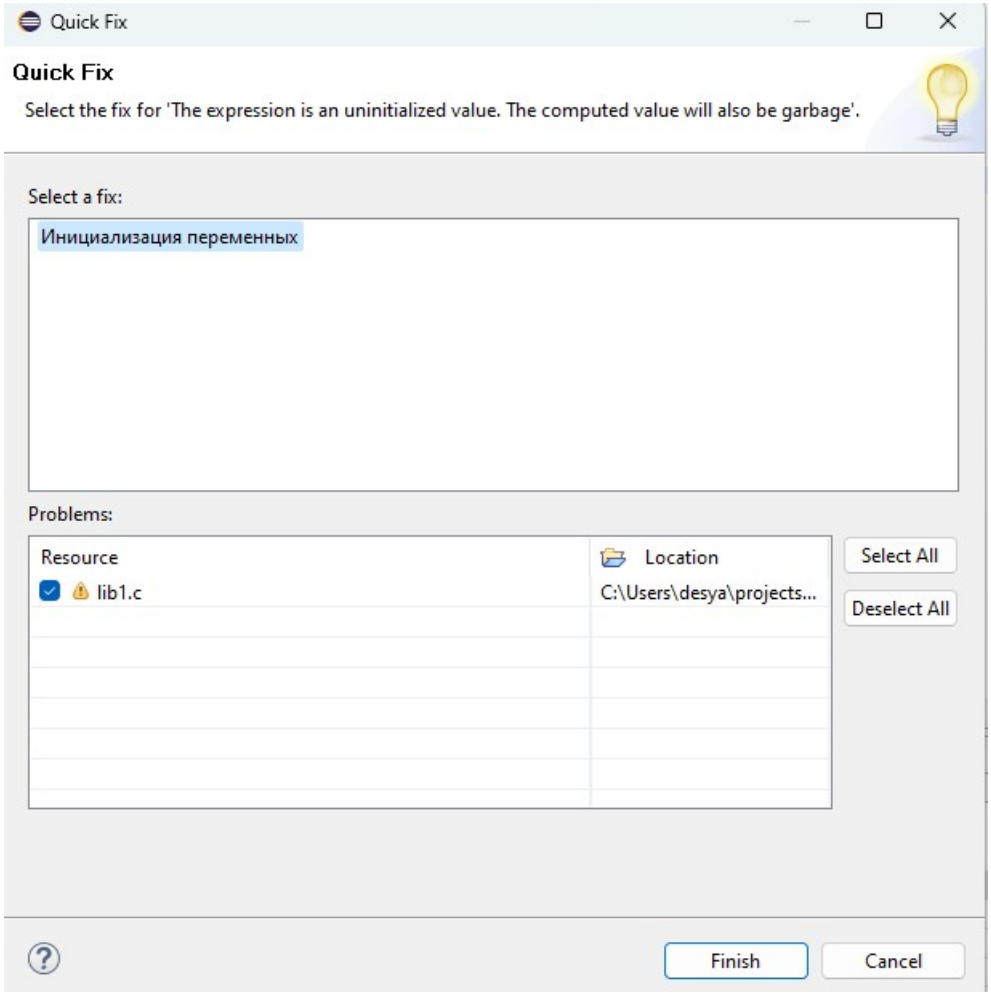

Рис. 2: Окно выбора нужного разрешения маркера.

Заключение

IDE является проектом с закрытым исходным кодом.

Репозиторий с реализованным рефакторингом: https://github.com/desyatok/llvm-project.

В ходе выполнения данной работы были достигнуты все поставленные цели, а именно:

1. сделан обзор используемых инструментов;
2. была выбрана одна ошибка из списка типичных и реализовано её исправление;
3. полученное преобразование было интегрировано в IDE.

Список литературы

[1] CLion IDE. — URL: https://www.jetbrains.com/clion/ (дата обращения: 25 марта 2024 г.).

[2] Clang Refactoring Engine. — URL: https://clang.llvm.org/docs/RefactoringEngine.html (дата обращения: 25 марта 2024 г.).

[3] Eclipse CDT. — URL: https://projects.eclipse.org/projects/tools.cdt (дата обращения: 25 марта 2024 г.).

[4] LLVM. — URL: https://llvm.org (дата обращения: 25 марта 2024 г.).

[5] Visual Studio. — URL: https://visualstudio.microsoft.com/ru/ (дата обращения: 25 марта 2024 г.).