

Утилита для генерации методов конвертации значений enum в строки и наоборот

Постановка задачи

Реализовать утилиту etosc для генерации методов конвертации значений enum'a в строки и наоборот.

Функциональные требования

1. Распознавать определения enum'ов в переданном в качестве аргумента командной строки файле исходного кода на языке C++ 11 стандарта и выше
2. Для каждого встреченного определения enum'a генерировать методы конвертации его значений в std::string и наоборот. Объявления и определения методов конвертации записывать в файлы etosc.h и etosc.cpp соответственно.

Используемое средство построения трансляторов

В процессе своей работы утилита etosc должна разбирать файл исходного кода на языке C++ и находить в нем определения пространств имен, классов, и enum'ов. При этом необходимо пропускать все остальные конструкции языка. Для выполнения этой задачи было решено использовать antlr версии 4, так как он позволяет построить парсер, используя технологию нечеткого парсинга («fuzzy parsing» <https://github.com/antlr/antlr4/blob/master/doc/wildcard.md>), не указывая при этом полную грамматику исходного языка.

Для использования в проекте была разработана следующая грамматика:

parseFile : (expr)* EOF;

expr : namespace_

 | enum_

 | LEFT_BRACE (expr)* RIGHT_BRACE

 | ~(LEFT_BRACE | RIGHT_BRACE)

;

namespace_ : namespaceKey WORD LEFT_BRACE (expr)+ RIGHT_BRACE;

enum_ : ENUM (classKey)? WORD (WORD)? LEFT_BRACE enumList RIGHT_BRACE ;

enumList : enumerator COMMA enumList | enumerator;

enumerator : WORD (EQUAL ENUMERATOR_VALUE)?

namespaceKey : classKey | NAME_SPACE;
classKey : CLASS | STRUCT;

Описание архитектуры

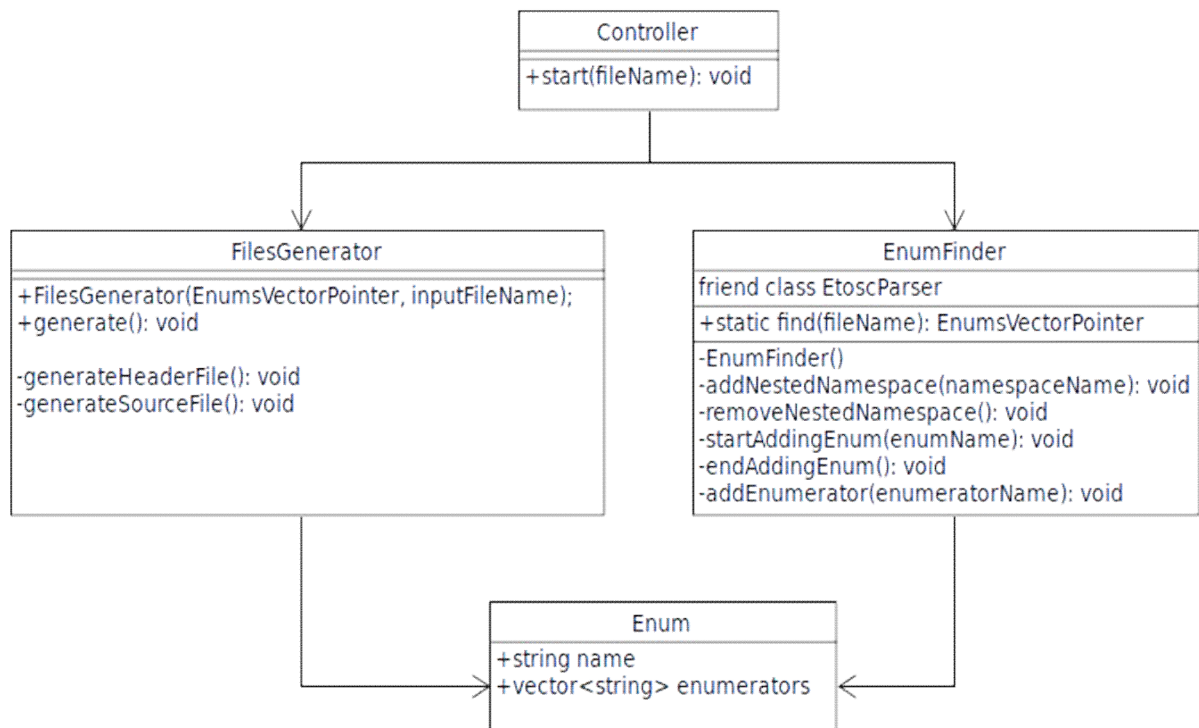
Классы программы можно разделить на два пакета (рис. 1).



Рис. 1 Диаграмма пакетов

Пакет antlr отвечает за разбор входного файла и содержит построенные утилитой antlr лексический и синтаксический анализатор, представленные классами EtoscLexer EtoscParser соответственно, а также библиотеку (antlr runtime), необходимую для их работы.

Пакет main содержит классы, реализующие основную логику приложения (рис. 2).



Класс Controller.

Является управляющим классом. Его метод start(...) вызывается из функции main(...) и содержит основные шаги алгоритма работы программы, в виде вызовов методов классов EnumFinder и FilesGenerator.

Класс EnumFinder.

Отвечает за поиск определений enum'ов в исходном файле и сохранение информации о найденных enum'ах и их значениях в памяти программы. Для этого EnumFinder инициализирует классы пакета antlr и предоставляет классу EtoscParser методы для сохранения информации об enum'ах, которые парсер вызывает в процессе разбора входного файла. К этим методам относятся:

- addNestedNamespace и removeNestedNamespace, вызываются, соответственно, в начале и в конце определения пространства имен или класса. Служат для корректного определения пространства имен, в котором объявлен enum.
- startAddingEnum и endAddingEnum, вызываются, соответственно, в начале и в конце определения enum'a. Служат для сохранения имени enum'a и определения текущего enum'a, к которому будут относиться найденные в ходе дальнейшего разбора значения.
- addEnumerator, вызывается после разбора значения enum'a. Служит для добавления имени значения к текущему enum'у.

Конструктор класса является закрытым. Для взаимодействия с классом Controller служит статический метод find(fileName), который создает объект класса EnumFinder, инициализирует классы пакета antlr, запускает поиск enum'ов в указанном файле вызовом метода parseFile() класса EtoscParser и возвращает информацию о найденных enum'ах в виде вектора объектов структуры Enum.

Класс FilesGenerator.

Служит для генерации выходных файлов, на основе переданных в конструктор вектора объектов структуры Enum и имени входного файла.

Для взаимодействия с классом Controller служит метод generate(), который вызывает закрытые методы generateHeaderFile() и generateSourceFile(), служащие для создания файлов etosc.h и etosc.cpp, содержащих объявления и определения методов конвертации значений enum'ов в std::string и наоборот.

Структура Enum.

Служит для представления enum'a в памяти программы. Содержит имя enum'a и вектор имен его значений.

Алгоритм работы программы

Алгоритм работы программы продемонстрирован на диаграмме последовательности (рис. 3).

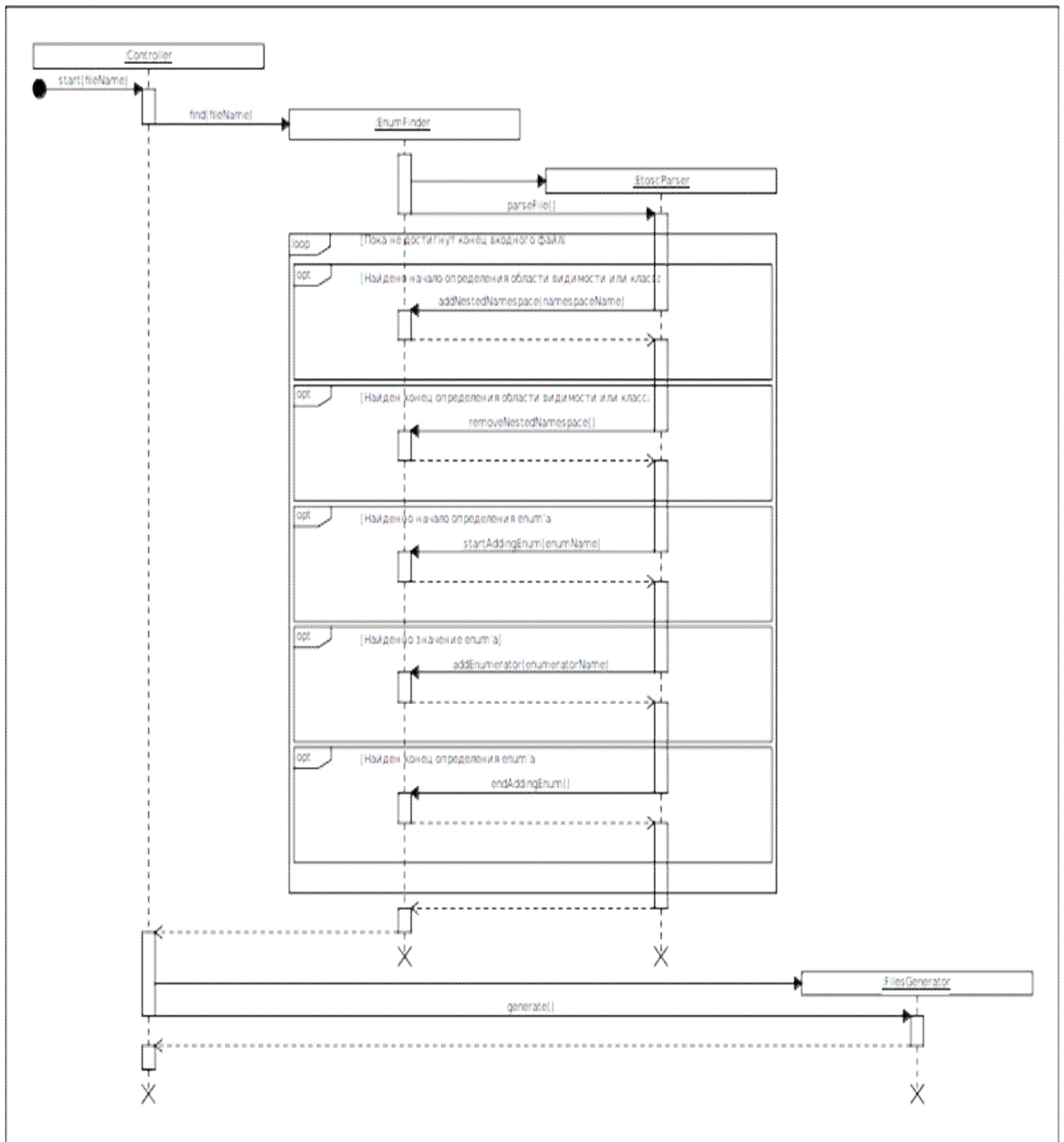


Рис. 3 Диаграмма последовательности

В функции main создается объект класса Controller и вызывается его метод start(fileName), в который передается имя входного файла. В методе start происходит вызов статического метода find(fileName) класса EnumFinder. В методе find происходит создание объекта класса EnumFinder и инициализация классов пакета antrl, в частности создание объекта класса EtoscParser. Затем вызывается метод parseFile() класса EtoscParser, в котором происходит разбор входного файла. В процессе разбора парсер вызывает методы класса EnumFinder, если встречается соответствующие конструкции языка C++ (см. Описание Архитектуры). После окончания разбора файла происходит возврат из метода find в метод start и уничтожение объектов классов EnumFinder и EtoscParser. Затем происходит создание объекта класса

FileGenerator и вызов его метода generate(). После завершения генерации файлов программа завершается.

Пример использования

Пример входного файла:

```
// example.h

#ifndef EXAMPLE_H
#define EXAMPLE_H

namespace A {

    class C {
    public:
        enum class Color {red, green, blue = 0x56FFC};
    };

    enum Fruits {banana, apple};
}

#endif
```

Пример выходных файлов:

```
// etosc.h

#ifndef ETOSC
#define ETOSC

#include <string>

template <typename EnumType>
std::string enumToString(EnumType val);

template <typename EnumType>
EnumType stringToEnum(const std::string &str);

#endif

-----
// etosc.cpp

#include <stdexcept>

#include "etosc.h"
#include "example.h"
```

```

template<>
std::string enumToString(A::C::Color val)
{
    switch (val)
    {
        case A::C::Color::red:
            return "red";

        case A::C::Color::green:
            return "green";

        case A::C::Color::blue:
            return "blue";
    }
}

template<>
A::C::Color stringToEnum(const std::string &str)
{
    if (str == "red")
        return A::C::Color::red;

    if (str == "green")
        return A::C::Color::green;

    if (str == "blue")
        return A::C::Color::blue;

    throw std::runtime_error(std::string("Enum A::C::Color has no "
                                         "enumerator named ") + str);
}

template<>
std::string enumToString(A::Fruits val)
{
    switch (val)
    {
        case A::Fruits::banana:
            return "banana";

        case A::Fruits::apple:
            return "apple";
    }
}

template<>
A::Fruits stringToEnum(const std::string &str)
{
    if (str == "banana")

```

```

        return A::Fruits::banana;

    if (str == "apple")
        return A::Fruits::apple;

    throw std::runtime_error(std::string("Enum A::Fruits has no "
                                         "enumerator named ") + str);
}

```

Пример использования:

```

// etosc.cpp

#include <iostream>
#include <string>
#include <exception>

#include "example.h"
#include "etosc.h"

using namespace std;

int main()
{
    cout << "enumToString: \n";
    cout << enumToString(A::C::Color::red) << endl;
    cout << enumToString(A::Fruits::apple) << endl;

    cout << "stringToEnum: \n";

    try {
        cout << ((stringToEnum<A::C::Color>("blue") == A::C::Color::blue) ? "true\n" : "false\n");
        cout << ((stringToEnum<A::Fruits>("banana") == A::banana) ? "true\n" : "false\n");

        stringToEnum<A::Fruits>("red");
    }
    catch (exception &e)
    {
        cout << e.what() << endl;
    }

    return 0;
}

```

Вывод:

```

enumToString:
red
apple
stringToEnum:

```

```
true  
true  
Enum A::Fruits has no enumerator named red
```

Вывод

Реализована утилита etosc для генерации методов конвертации значений enum'a в строки и наоборот. Улучшены навыки программирования и проектирования приложений на языке C++, а также построения трансляторов с помощью утилиты antlr версии 4.