ФГБОУ ВО «Чувашский государственный университет

им. И. Н. Ульянова»

Факультет Информатики и вычислительной техники

Объектно-ориентированное программирование

Курсовая работа

на тему “Модель лексического анализатора языка C++”

Выполнил: студент гр. ИВТ-41-22

Иванов Владимир Сергеевич

Проверил: кандидат технических наук

Обломов Игорь Александрович

Чебоксары 2024

Оглавление

Введение

В современном программировании языки компиляции, такие как C++, занимают важное место благодаря своей производительности и близости к аппаратному обеспечению. Разработка надежных и эффективных компиляторов для таких языков имеет решающее значение для создания высокопроизводительных и стабильных программ. Лексический анализатор является одним из первых и ключевых этапов в процессе компиляции, от которого зависит успешное выполнение последующих этапов, таких как синтаксический анализ и генерация кода.

Целью курсовой работы является создание программы на языке C++ с использованием объектно-ориентированного программирования, реализующую модель лексического анализатора. Программа классифицирует различные лексемы и выводит их тип.

Таким образом, данная курсовая работа не только углубляет понимание важного аспекта компиляторостроения, но и способствует развитию навыков объектно-ориентированного программирования.

1.Обзор предметной области

Лексический анализатор является одним из ключевых компонентов компилятора, играющим критическую роль в процессе преобразования исходного кода программы в машинный код. Чтобы глубже понять значимость и механизм работы лексического анализатора, рассмотрим основные концепции и подходы в данной предметной области.

Основные концепции лексического анализа

1. **Лексемы и токены**:

* **Лексема**: последовательность символов, объединенных в единицу, которая распознается лексическим анализатором как значимая. Примеры лексем включают ключевые слова, идентификаторы, операторы, литералы и разделители.
* **Токен**: абстрактное представление лексемы, которое включает тип и возможное значение. Токены передаются на следующую стадию компиляции — синтаксический анализ. В коде токены представлены объектами классов, такие как Identifier, Keyword, Operator, Literal, и Delimiter.

1. **Цели лексического анализа**:

* Преобразование последовательности символов исходного кода в последовательность токенов.
* Удаление пробельных символов и комментариев, которые не влияют на смысл программы.
* Обработка ошибок на уровне лексем, таких как недопустимые символы или слишком длинные идентификаторы.

Лексические анализаторы находят применение в различных областях:

* **Компиляторы**: основной компонент, обеспечивающий корректное разбиение исходного кода на токены.
* **Интерпретаторы**: используется для разбора и исполнения кода на лету.
* **Инструменты статического анализа кода**: проверяют исходный код на наличие ошибок, уязвимостей и несоответствий стилю кодирования.
* **Средства разработки**: обеспечивают функции подсветки синтаксиса, автодополнения и рефакторинга кода.

Лексический анализатор — это фундаментальный компонент компилятора, обеспечивающий преобразование исходного кода в токены.

2.Возможные варианты и способы решения задачи

Разработка лексического анализатора для языка программирования C++ включает несколько ключевых этапов, которые могут быть реализованы различными способами. Рассмотрим основные варианты и способы решения этой задачи:

1. Использование готовых инструментов для генерации лексических анализаторов

1. Flex (Fast Lexical Analyzer Generator):

* **Описание**: Flex — это мощный инструмент для создания лексических анализаторов. Он принимает на вход спецификацию, написанную на языке регулярных выражений, и генерирует исходный код на C/C++.
* **Преимущества**:
  + Быстрая разработка благодаря использованию шаблонов.
  + Хорошо подходит для сложных проектов с многочисленными лексемами.
* **Недостатки**:
  + Требуется изучение синтаксиса Flex и его интеграции с остальным кодом.
* **Пример использования**: Написание спецификации для Flex, которая описывает правила распознавания идентификаторов, ключевых слов, операторов и других лексем, и генерация соответствующего кода.

1. ANTLR (Another Tool for Language Recognition):\

* **Описание**: ANTLR — это инструмент для создания лексических и синтаксических анализаторов. Он использует язык описания грамматик, из которых генерирует код на различных языках программирования, включая C++.
* **Преимущества**:
  + Мощные возможности для обработки сложных грамматик.
  + Поддержка различных целевых языков программирования.
* **Недостатки**:
  + Более сложный синтаксис по сравнению с Flex.
  + Может требовать дополнительной настройки для интеграции
* **Пример использования**: Определение грамматики для языка C++, включающей правила для лексем и синтаксиса, и генерация кода лексического анализатора с помощью ANTLR.

2. Ручная реализация лексического анализатора

1. Использование регулярных выражений:

* **Описание**: Регулярные выражения могут быть использованы для определения шаблонов, по которым распознаются лексемы.
* **Преимущества**:
  + Гибкость и контроль над процессом анализа.
  + Легкость в использовании для простых грамматик.
* **Недостатки**:
* Может стать сложным и трудоемким для больших и сложных языков.
* Возможны проблемы с производительностью при обработке больших объемов кода.
* **Пример использования**: Написание кода на C++ с использованием стандартной библиотеки регулярных выражений для поиска и классификации лексем.

1. Использование конечных автоматов:

* **Описание**: Конечные автоматы (детерминированные или недетерминированные) могут быть использованы для построения лексического анализатора.
* **Преимущества**:
  + Высокая производительность.
  + Хорошая основа для оптимизации и расширения.
* **Недостатки**:
  + Требует знаний теории автоматов и ручного создания автоматов для каждой лексемы.
* **Пример использования**: Реализация конечного автомата на C++ для распознавания лексем, где каждый состояние автомата представляет различную часть процесса распознавания.

3. Объектно-ориентированное программирование

1. Создание иерархии классов для лексем:

* **Описание**: Определение базового класса Lexeme и производных классов для различных типов лексем, таких как Identifier, Keyword, Operator, Literal и Delimiter.
* **Преимущества**:
  + Легкость в расширении и модификации.
  + Ясная структура и поддерживаемость кода.
* **Недостатки**:
  + Может потребовать больше времени на начальную настройку и проектирование.
* **Пример использования**: Создание классов лексем и их атрибутов, реализация метода для анализа и классификации строк входного кода.

Выбор способа реализации лексического анализатора зависит от требований проекта, доступных ресурсов и знаний разработчика. Готовые инструменты, такие как Flex и ANTLR, могут значительно ускорить процесс разработки, особенно для сложных языков. Ручная реализация на основе регулярных выражений или конечных автоматов предоставляет больше контроля и гибкости, но требует большего объема работы и знаний. Объектно-ориентированный подход помогает создать легко расширяемую и поддерживаемую систему. В данной курсовой работе выбран объектно-ориентированный подход, который демонстрирует основные принципы лексического анализа и позволяет легко расширять функциональность анализатора в будущем.

3.Блок-схема алгоритма