ОТЧЕТ

ПОТАПОВА НАТАЛЬЯ, ПШЕНИЧНИКОВ ГЛЕБ, 212 группа

Реализация модельного языка программирования

1. Введение

Данный отчет описывает процесс разработки и реализации интерпретатора модельного языка программирования на языке C++. Целью проекта является создание функционального интерпретатора, способного выполнять программы, написанные на модельном языке, и демонстрирующего основные этапы обработки исходного кода.

Модельный язык программирования включает в себя базовые конструкции императивного программирования, такие как переменные, операторы, условные выражения и циклы. Он предоставляет возможность объявления переменных различных типов данных (целые числа, вещественные числа, логические значения, строки и массивы), выполнения арифметических и логических операций, а также организации управления потоком выполнения программы с помощью условных операторов и циклов.

2. Архитектура интерпретатора

Интерпретатор модельного языка программирования состоит из трех основных компонентов:

- Лексический анализатор (scanner.cpp)

- Синтаксический, семантический анализатор (parser.cpp)

- Модуль выполнения (execute.cpp)

Каждый компонент выполняет определенную роль в процессе интерпретации исходного кода.

Грамматика языка следующая.

Синтаксис:

P → **program** D1 B

D1 → **var** D {; D} {D_A}

D → id {, id} **:** (**int** | **real** | **bool** | **string** | **array**);

D_A → **array [**num**] of** (**int** | **real** | **bool** | **string**);

B → **begin** S {; S} **end**

S → **if** E **then** S [**else** S] | **while** E **do** S | **for** id **:=** E **to** E **do** S | **repeat** S {; S} **until** E | **case** E **of** C {; C} **end** | id **:=** E | id [E] **:=** E | **read (**id**)** | **write (**E {, E}**)** | B

C → num {, num} **:** S | **else** S

E → E1 {(= | < | > | <= | >= | **!**=) E1}

E1 → T {(**+** | **-** | **or**) T}

T → F {(***** | **/** | **and** | **%**) F}

F → id | id [E] | num | real | true | false | string | not F | (E)

Лексика:
id → letter {letter | digit}

num → digit {digit}

real → digit {digit} . digit {digit}

string → " {char} " letter → **a | b | ... | z | A | B | ... | Z**

digit → **0 | 1 | ... | 9**

char → **любой_символ_кроме_"**

В этой грамматике:

- Нетерминалы обозначены большими буквами (P, D1, D, D_A и т.д.)
- Терминалы обозначены жирным шрифтом (program, var, int и т.д.) или просто символами (:=, (,) и т.д.)
- Правила вывода используют символ →
- Квадратные скобки [ ] обозначают необязательные элементы
- Фигурные скобки { } обозначают повторение элемента 0 или более раз
- Вертикальная черта | обозначает альтернативу

В разделе "синтаксис" определена структура программы и операторов. В разделе "лексика" определено, как формируются идентификаторы (id), числа (num), вещественные числа (real) и строки (string) из букв (letter), цифр (digit) и символов (char).

2.1. **Лексический анализатор (scanner.cpp)**

Лексический анализатор отвечает за разбиение исходного кода на лексемы (токены). Он получает входной поток символов и выделяет из него лексемы, такие как ключевые слова, идентификаторы, числовые литералы, строковые литералы и разделители.

Реализация лексического анализатора включает в себя:

- Таблицы ключевых слов (TW) и разделителей (TD) для быстрого распознавания лексем

- Метод get_lex() для получения следующей лексемы из входного потока

- Обработку различных типов лексем (идентификаторы, числа, строки, операторы и т.д.)

- Обработку комментариев и пропуск пробельных символов

- Обработку ошибок, связанных с неверными символами или незавершенными лексемами

Лексический анализатор возвращает объект типа Lex, который содержит информацию о типе лексемы, ее значении и позиции в исходном коде.

2.2. **Синтаксический анализатор (parser.cpp)**

Синтаксический анализатор получает поток лексем от лексического анализатора и проверяет, соответствует ли структура программы заданной грамматике языка. Он строит дерево разбора и генерирует польскую инверсную запись (ПОЛИЗ) для последующего выполнения кода.

Реализация синтаксического анализатора включает в себя:

- Грамматику языка, описывающую структуру допустимых конструкций

- Методы для разбора различных конструкций языка (выражения, операторы, объявления и т.д.)

- Обработку приоритетов операторов и скобок

- Генерацию ПОЛИЗа для последующего выполнения кода

- Обработку ошибок синтаксиса и несоответствия типов данных

Синтаксический анализатор использует рекурсивный спуск для разбора различных конструкций языка и заполняет таблицу идентификаторов (TID) информацией о переменных и их типах данных.

2.3. **Модуль выполнения (execute.cpp)**

Модуль выполнения интерпретирует сгенерированный ПОЛИЗ и выполняет соответствующие действия. Он использует стек значений (args) для хранения промежуточных результатов вычислений и таблицу идентификаторов (TID) для доступа к значениям переменных.

Реализация модуля выполнения включает в себя:

- Обработку различных типов лексем в ПОЛИЗе (числа, идентификаторы, операторы и т.д.)

- Реализацию арифметических и логических операций

- Поддержку условных переходов и циклов

- Обработку ввода/вывода данных

- Обработку ошибок времени выполнения, таких как деление на ноль или выход за границы массива

Модуль выполнения интерпретирует ПОЛИЗ, выполняя соответствующие операции и обновляя значения переменных в таблице идентификаторов.

3. **Поддерживаемые возможности языка**

Модельный язык программирования поддерживает следующие возможности:

- Объявление переменных различных типов данных (целые числа, вещественные числа, логические значения, строки и массивы)

- Арифметические операции: сложение, вычитание, умножение, деление и остаток от деления

- Логические операции: и, или, не

- Операции сравнения: равно, не равно, меньше, меньше или равно, больше, больше или равно

- Условные операторы: if, if-else, case

- Циклы: while, for, repeat-until

-Унарный минус

- Ввод/вывод данных: read, write

- Объявление и использование одномерных массивов

- Обработка исключений (с указанием места ошибки)

4. **Тестирование и отладка**

В процессе разработки интерпретатора проводилось тестирование различных аспектов его функциональности. Были созданы тестовые программы, охватывающие различные конструкции языка и граничные случаи. Тестирование включало в себя:

- Проверку корректности лексического и синтаксического, семантического анализа
- Проверку правильности выполнения арифметических и логических операций
- Тестирование условных операторов и циклов
- Проверку обработки ошибок и исключительных ситуаций
- Тестирование ввода/вывода данных

Для отладки использовались отладочные сообщения и точки останова в коде интерпретатора. Это позволило выявить и исправить ошибки и неточности в реализации.

5. **Направления дальнейшего развития**

Реализованный интерпретатор модельного языка программирования предоставляет базовую функциональность для выполнения простых программ. Однако есть несколько направлений для дальнейшего развития и улучшения:

- Расширение поддерживаемых типов данных (например, добавление поддержки многомерных массивов, записей, перечислений)
- Реализация поддержки функций и процедур
- Оптимизация производительности интерпретатора
- Улучшение обработки ошибок и предоставление более информативных сообщений об ошибках
- Реализация статической проверки типов данных
- Добавление поддержки модульности и возможности подключения внешних библиотек
- Разработка интегрированной среды разработки (IDE) для удобства написания и отладки программ на модельном языке

6. **Заключение**

В данном отчете была представлена реализация интерпретатора модельного языка программирования на языке C++. Были описаны основные компоненты интерпретатора: лексический анализатор, синтаксический и семантический анализатор и модуль выполнения.

Реализованный интерпретатор демонстрирует основные принципы разработки языков программирования и может служить отправной точкой для дальнейшего изучения и развития в области компиляторов и интерпретаторов. Проект предоставляет базовую функциональность и поддерживает основные конструкции императивного программирования, позволяя писать и выполнять простые программы на модельном языке.

Дальнейшее развитие проекта может включать расширение возможностей языка, оптимизацию производительности и улучшение пользовательского опыта. Также можно рассмотреть возможность компиляции исходного кода в машинный код для повышения скорости выполнения программ.

Данный проект послужил ценным опытом в понимании принципов разработки языков программирования и реализации интерпретаторов. Он демонстрирует применение теоретических концепций на практике и закладывает основу для дальнейшего изучения и исследований в этой области.