Отчет по заданию «Интерпретатор модельного языка программирования»

Содержимое файлов

main.cpp: создание объекта типа Interpretator, запуск interpretation, обработка ошибок.
Ident.cpp, Ident.h: класс для идентификаторов.
Lex.cpp, Lex.h: класс для лексем.
Scanner.cpp, Scanner.h: лексический анализ.
Parser.cpp, Parser.h: синтаксический, семантический анализ, перевод в ПОЛИЗ.
Executer.cpp, Executer.h: этап выполнения.

Вариант задания

Реализуются if-else, циклы for и while, continue, read, write.
Логические операции: and, or, not.
Типы данных: int, string, real.
Добавлены операции: унарные + и -.

Лексический этап

Метод get_lex возвращает лексему и обрабатывает лексические ошибки:

- некорректные символы

- идентификатор >80 символов

- незакрытый комментарий

- строка незаконченная кавычкой

- незаконченное !=

Лексемы, которые обрабатываются:

Служебные слова:

LEX_NULL, LEX_PROGRAM, LEX_INT, LEX_STRING, LEX_REAL, LEX_READ, LEX_WRITE, LEX_IF, LEX_ELSE, LEX_WHILE, LEX_FOR, LEX_CONTINUE, LEX_AND, LEX_OR, LEX_NOT.

Разделители:

LEX_FIN, LEX_EQ, LEX_DEQ (double =), LEX_NEQ, LEX_STAR, LEX_SLASH, LEX_PLUS, LEX_MINUS, LEX_GTS (>), LEX_GTSEQ (>=), LEX_LTS (<), LEX_LTSEQ (<=), LEX_SEMICOLON, LEX_COMMA, LEX_DOT, LEX_LPAR (левая скобка), LEX_RPAR, LEX_LBR ({), LEX_RBR.

Дополнительные лексемы:

LEX_NUM – целое число, LEX_RNUM – дробное число, LEX_ID – идентификатор, LEX_STR_CONST – строковая константа.

POLIZ_LABEL, POLIZ_ADDRESS, POLIZ_GO, POLIZ_FGO, POLIZ_WRITE – лексемы для ПОЛИЗа.

Синтаксический этап

Используемая грамматика:

{} - повторение конструкций.

~{}~ - фигурные скобки - терминалы.

P -> program ~{D O @

D -> {int D1;| real D1; | string D1;} | eps

D1 -> D2 {,D2}

D2 -> id = "str_const" | id = +num | id = -num | id

O -> O1 O| }~

O1 -> if (E) O1 else O1 | while (E) O1 | continue; |

for (id E| + E| - E| not E| num E| str_const E ;

id E| + E| - E| not E| num E| str_const E ;

id E| + E| - E| not E| num E| str_const E) O1 |

read(id); | write(E {,E}); | id E; | num E; | str_const E; | ~{ O | ;

E -> E_OR { = E}

E_OR -> E_AND { or E_AND}

E_AND -> E_R { and E_R}

E_R -> E_AS {> E_AS | >= E_AS | < E_AS | <= E_AS | != E_AS | == E_AS}

E_AS -> E_MD {+ E_MD | - E_MD}

E_MD -> F {* F | / F}

F -> id | num | str_const | not F | + F | - F | (E)

Семантический этап

В классе Parser предусмотрены методы:

- void dec (type_of_lex) – пометить, что переменная объявлена, проверить, что это имя еще не объявлено.

- void check_id () – проверка, было ли объявление переменной перед использованием, занесение типа идентификатора в семантический стек.

- void check_op () – проверка корректности типов операндов бинарной операции, занесение типа результата в семантический стек.

- void check_not () – проверка корректности операнда для not.

- void check_unar (Lex) – проверка корректности операнда для унарной операции.

- void eq_type () – проверка корректности операндов для присваивания.

- void eq_bool () – проверка корректности операнда для логического выражения.

- void check_id_in_read () – проверка, объявлена ли переменная перед чтением.

ПОЛИЗ и этап выполнения

ПОЛИЗ строится в процессе синтаксического анализа.

В классе Executer метод execute разбирает вектор лексем poliz и осуществляет выполнение программы.

Обрабатывает ошибки:

- использование неинициализированной переменной

- недопустимый тип переменной

- недопустимый тип логического выражения

- деление на 0