

## К заданию практикума №2.

Описание задания смотрите в методичке «**Интерпретатор модельного языка программирования**».

### Требования к реализации:

Кроме типов **int** и **string** использовать еще один тип (**real** или **boolean**) на выбор.

Набор операторов, который должен быть реализован в модельном языке, следующий:

<оператор> →

**if** (<выражение>) <оператор> **else** <оператор>

|

**switch**(<выражение>){

**case** <константа>: { <оператор> }

{ **case** <константа>: { <оператор> }

[ **default** : { <оператор> }

}

| **for** ([<выражение>; [<выражение>; [<выражение>]) <оператор>

| **while** (<выражение>) <оператор>

| **break**;

| **goto** <идентификатор> ;

| **read** (<идентификатор>);

| **write** (<выражение> { ,<выражение> } );

| <составной оператор>

| <помеченный оператор>

| <оператор-выражение>

<помеченный оператор> → <идентификатор> : <оператор>

Оператор **read** - оператор ввода значения переменной <идентификатор>; **write** – оператор вывода значений списка выражений, указанных в круглых скобках. Форматы ввода и вывода данных определяются реализацией. Числовые константы записываются в десятичной системе счисления.

Семантика остальных операторов как в Си.

К общему набору арифметических операций добавить **унарный минус**.

Граматику для выражений пишете самостоятельно.

### Контекстные условия.

1. Любой идентификатор, используемый в программе, должен быть описан и только один раз.
2. При инициализации переменных типы констант должны совпадать с типами переменных.
3. С помощью оператора **read** можно вводить данные любых типов, определенных в языке, кроме логического.
4. С помощью оператора **write** можно выводить значения любых типов, определенных в языке.
5. В операторе цикла **for** типы 1-ого и 3-его выражений произвольные.
6. Если в языке есть логический тип данных (**boolean**), то только логическое выражение может использоваться в условном операторе **if**, в операторах цикла **while** и **for** в качестве условия завершения цикла. Если в языке нет логического типа данных, то используется целочисленное выражение (0 == false; любое значение, отличное от 0, == true). Вещественное выражение не может использоваться в качестве условия завершения цикла.
7. Тип выражения и констант вариантов в операторе **switch** должен быть целочисленным.
8. Описанием идентификатора-метки считается ее использование в помеченном операторе. Разные операторы не могут быть помечены одинаковыми метками. Одна и та же метка не может помечать один и тот же оператор более одного раза.

Выполнение задания практикума в рамках описанного выше минимального варианта (можно без оператора switch) при соблюдении требований к

синтаксическому и семантическому разбору, указанных в методическом пособии «Интерпретатор модельного языка», соответствует оценке 3.

Для получения 5 необходимо расширить программу следующим образом:

1) Добавить возможность задания составного типа

**struct <идентификатор>{<описание>;{<описание>;}};**

доступ к полям структуры должен осуществляться с помощью оператора ‘.’

#### Контекстные условия:

- Определения структур находятся в начале программы до описаний переменных.

Выделять ли для имен структур отдельное пространство имен ( в этом случае использование перед именем ключевого слова struct будет обязательно) или вводить их в общее пространство имен, решаете сами.

- Допустимые типы полей внутри структуры – только простые (**int, bool, real, string**).

- Использование значений полей структуры в выражениях такое же, как и у переменных того же типа.

- Возможно присваивание структур, причем только для структур одного типа.

Эквивалентность типов здесь именная, т.е. определяется именем структуры, а не набором ее полей.

2) Добавить этап препроцессорирования, на котором должны обрабатываться директивы условной компиляции:

**< if –строка> <текст> <else – часть> #endif <перевод строки>**

**<if-строка> -> #ifdef <идентификатор> <перевод строки> |**

**#ifndef <идентификатор> <перевод строки>**

**<else-часть>-> <else-строка> <текст> | ε**

**<else-строка> -> #else <перевод строки>**

**<текст> -> <любая последовательность строк программы, включая строки препроцессора, которые не являются частью данной условной структуры> | ε**

**#define <идентификатор> <целочисленная константа> <перевод строки>**

**#undef <идентификатор><перевод строки>**

Смысл команд условной компиляции обычный для C/C++.

Директивы условной компиляции могут быть вложенными.

Символ '#', начинающий директиву препроцессора, должен быть первым символом—не разделителем в строке.

Препроцессирование выполняется после лексического анализа.

Сдача задания делится на 3 этапа:

1. Лексический анализ + препроцессор;
2. Синтаксический анализ + ПОЛИЗ;
3. Интерпретация.