# Интерпретатор модельного языка программирования

Требуется разработать и реализовать интерпретатор модельного языка программирования. Инструментальный язык — C++.

# Синтаксис модельного языка:

Синтаксис описан с помощью расширенной БНФ:

a) запись вида {α} означает итерацию цепочки α, т.е. в порождаемой цепочке в этом месте может находиться либо ε, либо α, либо αα, либо ααα и т.д.

b) запись вида [α] означает, что в порождаемой цепочке в этом месте может находиться либо α, либо ε.

c) чтобы отличать фигурные скобки — служебные символы БНФ от фигурных скобок — символов модельного языка, первые выделены жирным шрифтом и подчеркнуты.

d) жирным шрифтом выделены служебные слова модельного языка.

## Общая часть (одинаковая для всех вариантов):

⟨*программа*⟩ → ***program*** { ⟨*описания*⟩ ⟨*операторы*⟩ }

⟨*описания*⟩ → **{** ⟨*описание*⟩; **}**

⟨*описание*⟩ → ⟨*тип*⟩ ⟨*переменная*⟩ **{ ,** ⟨*переменная*⟩ **}**

⟨*тип*⟩ → ***int*** | ***string***

⟨*переменная*⟩ → ⟨*идентификатор*⟩ |
⟨*идентификатор*⟩ = ⟨*константа*⟩

⟨*константа*⟩ → ⟨*целочисленная*⟩ | ⟨*строковая*⟩

⟨*целочисленная*⟩ → [⟨*знак*⟩] ⟨*цифра*⟩ **{** ⟨*цифра*⟩ **}**

⟨*знак*⟩ → + | −

⟨*строковая*⟩ → " **{** ⟨*литера*⟩ **}** "

⟨*операторы*⟩ → **{** ⟨*оператор*⟩ **}**

⟨*оператор*⟩ → ***if*** (⟨*выражение*⟩) ⟨*оператор*⟩ ***else*** ⟨*оператор*⟩ |
***while*** (⟨*выражение*⟩) ⟨*оператор*⟩|
***read*** (⟨*идентификатор*⟩); |
***write*** (⟨*выражение*⟩ **{ ,** ⟨*выражение*⟩ **}** ); |
⟨*составной оператор*⟩ | ⟨*оператор-выражение*⟩

⟨*составной оператор*⟩ → { ⟨*операторы*⟩ }

⟨*оператор-выражение*⟩ → ⟨*выражение*⟩;

Семантика операторов ***if*** и ***while*** общепринятая; 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.

# О выражении

Набор операций и их старшинство:

| | |
|---|---|
| ***not*** | (логическое отрицание) |
| * / | (умножение и деление) |
| + − | (сложение и вычитание) |
| < > <= >= == != | (операции отношения) |
| ***and*** | (логическое умножение) |
| ***or*** | (логическое сложение) |
| = | (присваивание) |

Семантика операций общепринятая (в частности, семантика операции присваивания эквивалентна семантике соответствующего оператора языка C).

Синтаксис выражений описывается самостоятельно; настоятельно рекомендуется с помощью синтаксических правил задать старшинство операций.

Для данных типа ***string*** определены следующие операции:

- инициализация и присваивание
- + операция конкатенации строк; группировка слева направо

  Например, если имеется определение *string a* = "*one*", *b* = "*two*", *c*; то в результате выполнения оператора-выражения *c* = *a* + " *plus* " + *b*; в *c* будет получена строка "*one plus two*".
- операции отношения < > == !=

  Например, если *string a* = "*one*", *b* = "*only*", *c* = "*one*"; то истинными являются выражения *a* < *b*, *a* == *c*, *b* != *c*; ложными — *c* > *b*, *a* != *c*, *a* == *b*.

## Вариантная часть (расширяет язык, определенный в общей части):

В каждый вариант входит (как минимум) один из подпунктов каждого пункта, отмеченного римской цифрой.

### I. Условные операторы

1. ⟨*оператор*⟩ → ***if*** (⟨*выражение*⟩) ⟨*оператор*⟩
2. ⟨*оператор*⟩ → ***case*** (⟨*выражение*⟩) ***of*** ⟨*список_вариантов*⟩ ***end***;
   ⟨*список вариантов*⟩ → ⟨*вариант*⟩ { ⟨*вариант*⟩ }

⟨*вариант*⟩ → ⟨*константа*⟩ { , ⟨*константа*⟩ } : ⟨*оператор*⟩

Семантика оператора **if** общеизвестна; семантика оператора **case –** как в Паскале (в частности, если значение выражения не совпадает со значением какой-либо константы в списке вариантов, это считается ошибкой; константы в разных вариантах должны быть различными, даже в одном варианте не должно быть одинаковых констант).

## II. Операторы цикла

1. ⟨*оператор*⟩ → ***do*** ⟨*оператор*⟩ ***while*** (⟨*выражение*⟩) **;**
2. ⟨*оператор*⟩ → ***for*** ( [⟨*выражение*⟩]**;**[⟨*выражение*⟩]**;**[⟨*выражение*⟩])
   ⟨*оператор*⟩

Семантика операторов цикла **do-while** и **for** совпадает с семантикой соответствующих операторов языка C.

3. ⟨*оператор*⟩→ ***for*** ⟨*параметр цикла*⟩ = ⟨*выражение*⟩ ***step*** ⟨*выражение*⟩
   ***until*** ⟨*выражение*⟩ ***do*** ⟨*оператор*⟩
   ⟨*параметр цикла*⟩ → ⟨*идентификатор*⟩

Если записать оператор цикла с параметром в виде ***for*** $I = E_1$ ***step*** $E_2$ ***until*** $E_3$ ***do*** $S$, где $E_1$, $E_2$, $E_3$ — выражения, $S$ — оператор, то семантика этого оператора такова: параметр цикла $I$ принимает последовательные значения от заданного начального значения $E_1$ до заданного конечного значения $E_3$ с шагом $E_2$; при каждом значении параметра цикла выполняется оператор $S$. Если начальное значение больше конечного, то оператор не выполняется ни разу. Значения выражений $E_1$, $E_2$, $E_3$ вычисляются один раз перед входом в цикл. Значение параметра цикла после его нормального (не по ***goto***) завершения считается неопределенным.

## III. Операторы перехода

1. ⟨*оператор*⟩ → ⟨*помеченный оператор*⟩ | ***goto*** ⟨*идентификатор*⟩ ;
   ⟨*помеченный оператор*⟩ → ⟨*идентификатор*⟩ **:** ⟨*оператор*⟩
2. ⟨*оператор*⟩ → ***continue***;
3. ⟨*оператор*⟩ → ***break***;

В соответствии с контекстными условиями, любой идентификатор, используемый в программе, должен быть описан и только один раз. Описанием идентификатора-метки считается ее использование в помеченном операторе. Разные операторы не могут быть помечены одинаковыми метками (это эквивалентно повторному описанию этой метки); более того, будем считать, что одна и та же метка не может помечать один и тот же оператор более одного раза.

Семантика операторов ***continue*** и ***break*** совпадает с семантикой соответствующих операторов языка C и относится к оператору цикла любого вида, входящему в выбранный вариант модельного языка.

## IV. Типы данных

1. ⟨*тип*⟩ → ***boolean***

⟨*константа*⟩ → ⟨*логическая*⟩

⟨*логическая*⟩ → ***true*** | ***false***

2. ⟨*тип*⟩ → ***real***

⟨*константа*⟩ → ⟨*вещественная*⟩

⟨*вещественная*⟩ → [⟨*знак*⟩] ⟨*целая часть*⟩**.**⟨*дробная часть*⟩

⟨*целая часть*⟩ → ⟨*цифра*⟩ { ⟨*цифра*⟩ }

⟨*дробная часть*⟩ → ⟨*цифра*⟩ { ⟨*цифра*⟩ }

Числовые константы записываются в десятичной системе счисления.

### V. Дополнительные операции

1. унарный минус,
2. унарные минус и плюс,
3. % — остаток от деления.

### VI. Правило вычисления логических выражений

1. «Ленивые» вычисления логических выражений (слева направо; до тех пор, пока не станет известно значение выражения).
2. Вычисления по обычной схеме (слева направо; вычисление всего выражения).

## 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 с параметром ***for*** $I = E_1$ ***step*** $E_2$ ***until*** $E_3$ ***do*** $S$ тип выражений и параметра цикла должен быть целочисленным.
6. Если в языке есть логический тип данных (***boolean***), то только логическое выражение может использоваться в условном операторе ***if*****,** в операторах цикла ***while*****, *for*** и ***do-while*** 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 ***case*** должен быть целочисленным.

8. Если в языке есть логический тип данных (***boolean***), то тип выражения и совместимость типов операндов в выражении определяются по правилам, приведенным в Таблице № 1, иначе — в Таблице № 2.

**Замечание:**

- если в языке отсутствует какая-либо операция либо тип, указанный в качестве типа операнда операции, то строки таблиц, их содержащие, не принимать во внимание.
- если в языке есть логический тип данных, то результат выполнения операций отношения — логическое значение ***true*** или ***false***, иначе — целочисленные 0 (***false***) либо 1 (***true***).
- *X* — первый операнд, *Y* — второй операнд двуместной операции; если операция одноместная, то вместо типа второго операнда стоит «–».

| Операция | тип X | тип Y | тип результата |
|---|---|---|---|
| + – * / % | *int* | *int* | *int* |
| + – * / | *real* | *real* | *real* |
| + – * / | *real* | *int* | *real* |
| + – * / | *int* | *real* | *real* |
| + | *string* | *string* | *string* |
| унарные +– | *int* | – | *int* |
| унарные +– | *real* | – | *real* |
| < > <= >= == != | *int* | *int* | *boolean* |
| < > <= >= == != | *real* | *real* | *boolean* |
| < > <= >= == != | *real* | *int* | *boolean* |
| < > <= >= == != | *int* | *real* | *boolean* |
| < > == != | *string* | *string* | *boolean* |
| ***and or*** | *boolean* | *boolean* | *boolean* |
| ***not*** | *boolean* | – | *boolean* |
| = | *int* | *int* | *int* |
| = | *real* | *real* | *real* |
| = | *int* | *real* | *int* |
| = | *real* | *int* | *real* |
| = | *string* | *string* | *string* |
| = | *boolean* | *boolean* | *boolean* |

Таблица №1

| Операция | тип X | тип Y | тип результата |
|---|---|---|---|
| + – * / % | *int* | *int* | *int* |
| + – * / | *real* | *real* | *real* |
| + – * / | *real* | *int* | *real* |
| + – * / | *int* | *real* | *real* |
| + | *string* | *string* | *string* |
| унарные +– | *int* | – | *int* |
| унарные +– | *real* | – | *real* |
| < > <= >= == != | *int* | *int* | *int* |
| < > <= >= == != | *real* | *real* | *int* |
| < > <= >= == != | *real* | *int* | *int* |
| < > <= >= == != | *int* | *real* | *int* |
| < > == != | *string* | *string* | *int* |
| ***and or*** | *int* | *int* | *int* |
| ***not*** | *int* | – | *int* |
| = | *int* | *int* | *int* |
| = | *real* | *real* | *real* |
| = | *int* | *real* | *int* |
| = | *real* | *int* | *real* |
| = | *string* | *string* | *string* |

Таблица №2

# Правила записи текста программы на модельном языке

Эти правила характерны для большинства языков программирования:

- в любом месте программы, кроме идентификаторов, служебных слов и числовых констант, может находиться произвольное число пробельных литер и комментариев вида /* ⟨любые символы, кроме */ ⟩ */.
- пробел в строковой константе считается значащим символом строки.
- внутри идентификаторов, служебных слов, числовых констант и разделителей, состоящих из нескольких символов, пробельные литеры недопустимы.
- между идентификаторами, числами и служебными словами должен находиться хотя бы один разделитель текста. Разделитель текста — это пробельная литера, комментарий либо разделитель, определенный в алфавите языка ( * / % + − < > <= >= == != , ; : ( ) ).

# Фазы работы интерпретатора модельного языка

Концептуально интерпретатором выполняются следующие фазы:

- лексический анализ,
- синтаксический анализ,
- семантический анализ (контроль контекстных условий),
- генерация программы на внутреннем языке (в качестве внутреннего языка предлагается использовать **польскую инверсную запись** — ПОЛИЗ),
- интерпретация программы на внутреннем языке.

На практике некоторые фазы можно сгруппировать, т.е. реализовать их функции на одном проходе.

### VII. Варианты таких группировок:

1. Первый проход — лексический анализ; результат — последовательность лексем + таблицы (идентификаторов и констант; таблицы служебных слов и разделителей формируются заранее);

   Второй проход — синтаксический анализ + семантический анализ + генерация: анализируется последовательность лексем и генерируется программа на внутреннем языке, эквивалентная исходной. На этом проходе активно используются и заполняются таблицы;

   Третий проход — интерпретация программы на внутреннем языке при заданных входных данных.

2. Первый проход — лексический анализ + синтаксический анализ + семантический анализ + генерация. При этом ведущую роль играет синтаксический анализатор –

по его запросу лексический анализатор выдает очередную лексему; после того, как синтаксический анализатор выделил некоторую синтаксическую единицу, осуществляется контроль контекстных условий и генерируется соответствующий фрагмент внутреннего представления;

Второй проход — интерпретация программы на внутреннем языке при заданных входных данных.

### VIII. Какой

бы вариант группировки фаз не был выбран, интерпретация программы на внутреннем языке может выполняться для различных наборов входных данных.

Следовательно, нужно позаботиться о хранении программы на внутреннем языке, чтобы можно было обеспечить следующие возможности:

1. неоднократная интерпретация программы на внутреннем языке при различных входных данных (в рамках одного запуска программы-интерпретатора),
2. неоднократная интерпретация программы на внутреннем языке при различных входных данных (при различных запусках программы-интерпретатора).

# Тестирование и отладка интерпретатора

Прежде всего, необходимо осознать, что **тестирование** — это процесс выявления дефектов в программных продуктах. После того как дефект обнаружен, начинается **отладка** — обнаружение причины появления дефекта и ее устранение.

Поскольку «полное» (исчерпывающее) тестирование невозможно, необходима некоторая методика, которая позволила бы разработать компактный, но достаточно продуктивный комплект тестов, который позволил бы обнаружить как можно больше дефектов.

Для того, чтобы можно было сравнивать разные комплекты тестов, необходимы явно сформулированные критерии полноты тестирования, которые обеспечиваются этими комплектами. Эти критерии различны для разных стратегий тестирования.

Обычно различают две стратегии тестирования: тестирование методами **«белого ящика»** и тестирование методами **«черного ящика»**.

Стратегии методов «белого ящика» опираются на знание логики программы, ее внутренней структуры. К методам «белого ящика» относятся методы покрытия операторов, покрытия решений, покрытия условий и метод комбинаторного покрытия условий. Все эти методы базируются на том, что при прогоне тестов должны выполняться те или иные конструкции в тексте программы. Например, комплект тестов, полный относительно критерия покрытия операторов, подразумевает выполнение каждого оператора программы хотя бы один раз при прогоне всех тестов комплекта.

При использовании методов «черного ящика» комплект тестов создается только в соответствии со спецификацией программы. Основная цель – выяснение ситуаций, в которых поведение программы не соответствует ее спецификации. К методам «черного ящика» относятся метод эквивалентных разбиений, анализ граничных условий, метод функциональных диаграмм.