Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт Космических и Информационных технологий
институт
Кафедра Информатики
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.С.Кузнецов
подпись инициалы, фамилия
« 08 » 06 2017г.

# БАКАЛАВРСКАЯ РАБОТА

09.03.04 «Программная инженерия»
код – наименование направления

Предметно-ориентированный язык для интеллектуальной системы генерации сюжетов
тема

Руководитель: _____ 08.06.17 доцент, к.т.н. А.С.Кузнецов
подпись, дата должность, ученая степень инициалы, фамилия

Выпускник: Головин, 08.06.17 Р.А.Головин
подпись, дата инициалы, фамилия

Нормоконтроль: _____ 08.06.17 доцент, к.т.н. О.А.Антамошкин
подпись, дата должность, ученая степень инициалы, фамилия

Красноярск 2017

# РЕФЕРАТ

Бакалаврская работа 30 стр., 1 рисунок, 1 источник.

Объект исследования – защита персональных данных.

Цель работы – разработать предметно-ориентированный язык сверхвысокого уровня для интеллектуальной системы генерации сюжетов.

Метод исследования – практический эксперимент.

Результат – изучена предметная область по данной теме, а также разработан язык сверхвысокого уровня.

**Оглавление**

## Введение

В наш век электроники, когда компьютеры проникли буквально всюду, почти не осталось сфер, где бы компьютер не смог заменить человека. Искусственный интеллект автоматически доказывает теоремы, переводит тексты с одного языка на другой, распознаёт изображения, управляет компьютерными противниками в игре…

В общем, область распространения ИИ велика, и если он ещё не достиг совершенства, то вплотную приблизился к нему. Однако ещё существует одна область, где компьютер человеку не конкурент. Эта область – творчество. В написании книг, стихов и музыки человек всё ещё безусловный фаворит. Но время не стоит на месте, и когда-нибудь монополия человека на творчество будет нарушена.

Цель данной работы – приблизить наступление тех времён, когда компьютерная программа сможет писать за человека книги. Оставим в стороне вопрос об этичности или не этичности этих намерений – он лежит за рамками рассматриваемой проблемы. Автор данного отчёт задался целью создать такую интеллектуальную систему, которая бы помогала начинающим авторам в их нелёгком ремесле.

Однако разработка подобной ИС подводит нас к другому вопросу. Если бы наша экспертная система имела бы вид простого логического аппарата, который, имея правила и факты, выводил бы из имеющегося искомое, то весь труд нашего эксперта сводился бы к придумыванию новых фактов и введению новых правил. Однако чем сложней будет система, тем больше труд эксперта будет походить на труд программиста-разработчика, что, безусловно, противоречит самой идее экспертных систем. Именно поэтому, желая не только облегчить труд эксперта, но и дать ему в руки мощный инструмент разработки, и была затеяна эта работа.

Чтобы обуздать наратающую сложность, нужно научить программу общаться с экспертом без посредничества программиста, создав для этого

простой, но функциональный язык, оперируя элементами которого, эксперт будет вводить в процесс генерации сюжета новые функциональные сущности.

Решено, что будущий язык будет интерпретироваться средствами языка C++ и представлять собой набор простых команд, комбинируя которые, можно контролировать процесс генерации сюжета. Данный язык будет обладать сверхвысоким уровнем абстракции и максимально простым синтаксисом.

# 1. Анализ и обзор языков программирования сверхвысокого уровня

## 1.1 Особенности предметной области и решаемых задач

На данный момент существует ряд программ-ассистентов, призванный выполнять примерно те же функции, что IDE в работе программиста или память переводов в работе переводчика. ПО такого типа позволяют работать с набором текстов будущего произведения художественной литературы как с проектом, поддерживая связь фрагментов-сцен в хронологическом порядке или по сюжетным линиям, а также поддерживают ведение вспомогательных баз данных, например для персонажей. К таким novel-writing программам относятся YWriter, oStorybook, Writer's Cafe и Scrivener.

Однако все эти программы лишь ассистируют, весь писательский труд всё равно выполняет человек. Разрабатываемая экспертная система предусматривает исключение человека из процесса создания литературного произведения.

## 1.2 Анализ существующих языков сверхвысокого уровня

Сверхвысокоуровневым языком программирования (VHLL) называют те языки, в которых достигается максимальный уровень абстракции. Программист описывает не «как нужно сделать», а «что нужно сделать».

Именно поэтому каждый подобный язык разрабатывается под решение ряда узкоспециализированных задач, и для решения других непригоден.

Из этого следует, что аналогов разрабатываемого языка не существует. Чтобы убедиться в этом, ниже будет приведён ознакомительный обзор двух языков сверхвысокого уровня.

### 1.2.1 Smalltalk

Smalltalk — объектно- ориентированный язык программирования с динамической типизацией, основанный на идее посылки сообщений, разработанный в Xerox PARC Аланом Кэйем, Дэном Ингаллсом, Тедом Кэглером, Адель Голдберг, и другими в 1970-х годах. Представляет собой интегрированную среду разработки и исполнения, объекты которой доступны для модификации через неё саму, и программирование в которой в итоге сводится к модификации её собственного поведения. Язык был представлен как Smalltalk-80.[1]

Smalltalk является одним из многих объектно-ориентированных языков, основанных на языке Симула, который сам оказал большое влияние на развитие многих объектно-ориентированных языков. Многие идеи 1980-х и 1990-х по написанию программ появились в сообществе Smalltalk. К ним можно отнести рефакторинг, шаблоны проектирования (применительно к ПО), карты «класс — обязанности — взаимодействие» и экстремальное

программирование в целом. Основатель концепции вики Уорд Каннингем также входит в сообщество Smalltalk.

Основными идеями Smalltalk являются:

- Всё — объекты, и всё их взаимодействие — через посылку сообщений. Строки, целые числа, логические значения, определения классов, блоки кода, стеки, память, составляющие самой интегрированной среды разработки и исполнения — всё представляется в виде объектов. У объектов есть методы и состояние. Любому объекту может быть послано любое сообщение. При отправке сообщения, среда исполнения всегда ищет у объекта-получателя подходящий метод и выполняет его, а если не находит — выполняет у объекта-получателя специальный метод для неопознанных сообщений. Объект-получатель сам определяет, является ли полученное сообщение правильным, и что надо сделать, чтобы его обработать.

- Всё доступно для изменения. Если вы хотите изменить саму интегрированную среду разработки и исполнения, вы можете сделать это в работающей системе, без остановки, перекомпиляции и перезапуска. Если вам необходима в языке новая управляющая конструкция языка, вы можете добавить её. В некоторых реализациях вы можете также изменить синтаксис языка или способ работы сборщика мусора.

- Динамическая типизация — это означает, что вы не указываете типы переменных в программе, что делает язык гораздо лаконичней. (Как объяснено выше, является ли операция правильной, определяет объект-получатель, а не компилятор).

Smalltalk также использует другие современные идеи:

- Сборка мусора встроена в язык и незаметна разработчику.

- Dynamic translation: современные коммерческие виртуальные машины компилируют байткоды в машинные коды для быстрого выполнения.

- Выполнение кода в виртуальной машине. Программы Smalltalk обычно компилируются в байткоды и выполняются виртуальной машиной, что позволяет выполнять их на любом оборудовании, для которого существует виртуальная машина.

Одной из особенностей Smalltalk является то, что даже такие традиционные конструкции, как if-then-else, for, while, и т. д. не являются частью языка. Все они реализованы с помощью объектов. Например, решение принимается с помощью посылки сообщения ifTrue: логическому объекту, и передаёт управление фрагменту текста, если логическое значение истинно.

Собственно встроенных синтаксических конструкций в языке не много:

- посылка сообщения объекту с возможной передачей ему других объектов;

- присваивание объекта переменной;

- возвращение объекта из метода;

- и несколько синтаксических конструкций для определения объектов-литералов и временных переменных.

Аналогом механизма обмена сообщениями Smalltalk является сеть интернет: можно представить каждый объект как веб-сервер, отвечающий на запросы. При этом, сервер на запросы может просто выдавать заранее предопределённый ответ, например веб-страницу, расположенную по определённому пути; может перенаправить запрос-сообщение другому объекту, аналог — прокси-сервер; может изменить запрос по определённым правилам, аналог — техника url rewriting, и конечно же может сформировать абсолютно новую страницу, соответствующую данным, переданным с сообщением. Если для реакции на сообщение у объекта нет предопределённого

метода, то среда вызывает у получателя метод #doesNotUnderstand:, так же, как веб-сервер возвращает страницу с сообщением об ошибке, если задан несуществующий путь к веб-странице.

### 1.2.2 Icon

Icon — язык программирования, унаследовавший идеологию более раннего языка того же автора Снобол. Название языка не имеет ничего общего с «иконками», а является сокращением от слова англ. iconoclastic (иконоборческий), используемом в смысле борьбы с конформизмом в разработке языков программирования. [2]

Это сверхвысокоуровневый язык программирования, в который интегрированы механизмы сопоставления с образцом и бэктрекинга, что сближает его с языками логического программирования.

Язык является динамически типизированным, имеет встроенные мощные типы данных. Процедуры в Icon’е относятся к величинам первого класса, что означает возможность присваивания переменным значений самих процедур, а не результатов их выполнения. Существует механизм со-выражений, позволяющий создавать сопрограммы.

Любое предложение (оператор) языка Icon на самом деле является выражением и может возвращать значения. Выражения в Icon, помимо собственно возвращаемых ими значений, которых может быть любое количество, производит одно из двух состояний — успех англ. success или неудачу англ. failure. Успех или неудача выражений используются вместо булевых значений управляющими структурами Icon. Благодаря этому мы можем писать конструкции типа:

```
if a := read() then write(a)
```

Более того, поскольку состояние «неудачи» имеет свойство «всплывать» из вложенных вызовов функций, становясь результатом внешней функции, можно использовать ещё более краткие идиомы, вроде:

```
while write(read())
```

для чтения потока ввода и дублирования его в поток вывода (echo).

Операции сравнения, такие > и <, могут быть успешны и производить результат, равный значению их второго аргумента, или неудачны, и не производить никакого результата. Благодаря этому в Icon можно писать выражения типа if a < b < c then ….

Связанная с успехом и неудачей концепция называется в Icon целенаправленным выполнением. Это способ, благодаря которому вычисление может продолжаться, пока не будет достигнута некая цель. В вышеприведённом примере с echo цель — чтение всего содержимого файла на входе, пока не будет встречен конец файла. Цель задаётся непосредственно, а не с помощью дополнительных проверок кодов возврата или чего то подобного. Целенаправленное вычисление реализуется с помощью бэктрэкинга, это очень мощный механизм.

## 1.3 Выделение существенных особенностей разрабатываемого языка сверхвысокого уровня

На основе анализа существующих VHLL был составлен общий список требований к разрабатываемому языку:

- За каждой из существующих инструкций должно стоять полностью завершённое действие;
- Каждая команда своим названием должна максимально точно передавать своё назначение;
- Названия всех конструкций, используемых в языке, должны быть простыми и легко запоминаться далёкими от программирования людьми.

Стоит, однако, заметить, что сформулированные требования слишком абстрактны, чтобы их формализировать. Не исключено, что найдутся пользователи, для которых синтаксис языка окажется слишком сложным.

# 2. Используемые технологии

## 2.1 Выбор средства разработки. Язык программирования

Для решения поставленной задачи был выбран язык программирования C++, поскольку интеллектуальная система, для общения с которой и разрабатывается данный язык, была реализована средствами этого языка.

## 2.2 Среда разработки

Поскольку автор данной работы имеет большой опыт в работе со средой разработки Microsoft Visual Studio, то для решения поставленной задачи была выбрана именно она, а конкретно Visual Studio 2012.

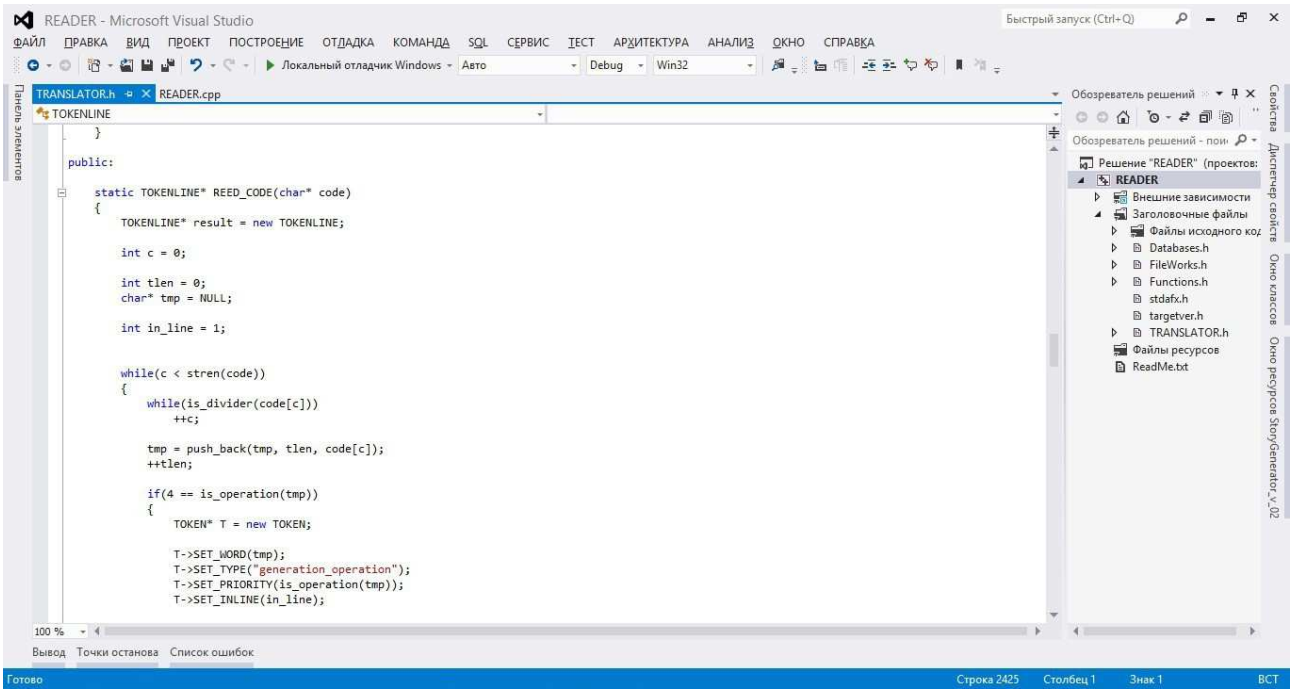


Рисунок 1 – Интерфейс Mircosoft Visual Studio 2012

Одним из достоинств данной среды является возможность разрабатывать как консольные приложения, так и приложения с графическим

интерфейсом, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server). [3]

## 3. Реализация и описание

### 3.1 Описание языка

#### 3.1.1 Лексические соглашения

##### 3.1.1.1 Алфавит

Множество символов языка READER содержит заглавные буквы латинского алфавита, цифры и знаки пунктуации, которые имеют определенный смысл для компилятора языка. Программы строятся путем комбинирования символов из множества символов языка READER.

Языковые конструкции могут содержать только символы из множества символов языка READER, однако внутри символьных строк может быть использован любой представимый символ. При обнаружении неверно использованных символов компилятор выдает сообщение об ошибке.

Ниже описывается символы из множества символов языка READER и объясняются правила их использования.

##### 3.1.1.2 Буквы и цифры

Множество символов языка READER включает прописные буквы латинского алфавита и арабские цифры:

- прописные латинские буквы: ABCDEFGHIJKLMNOPQRSTUVWXYZ;
- десятичные цифры: 0123456789.

Буквы и цифры используются при формировании констант, переменных и ключевых слов (эти конструкции описаны ниже).

##### 3.1.1.3 Пробельные символы

В языке READER лексемы разделяются знаком пробела. Символа перевода на новую строку нет, т.к. каждая команда и так пишется в отдельной строке.

Лексемы обязаны отделяться друг от друга как минимум одним знаком пробела.

### 3.1.1.4 Разделители

В языке READER разделительные символы несут особый смысл, т.к. каждый из них выполняет различные функции.

Таблица 1 – Разделители языка READER

| Символ | Наименование |
|---|---|
| , | Запятая |
| . | Точка |
| ; | Точка с запятой |
| : | Двоеточие |
| ( | Левая круглая скобка |
| ) | Правая круглая скобка |
| { | Левая фигурная скобка |
| } | Правая фигурная скобка |
| < | Знак “меньше” |
| > | Знак “больше” |
| [ | Левая квадратная скобка |
| ] | Правая квадратная скобка |
| ! | Восклицательный знак |
| + | Знак плюс |
| ? | Знак вопроса |
| = | Знак равенства |
| \| | Вертикальная черта |

Более подробно о функциях этих символов в языке READER будет сказано ниже.

Элементы множества представимых символов, которые не представлены в данном списке (в частности, русские буквы), могут быть использованы только в символьных константах.

### 3.1.1.5 Операции

Операции — это определённая комбинация символов, влекущая за собой определённые действия в программе. С точки зрения компилятора, каждая из этих конструкций – самостоятельная лексема.

В таблице 2 представлен список операций. Операции должны использоваться точно так, как они представлены в таблице, добавление

пробельных символов в многосимвольные операции недопустимо.

Таблица 2 – Операции языка READER

| Операция | Наименование |
|---|---|
| CHARACTER | Порождение переменной типа CHARACTER |
| ROLE | Порождение переменной типа ROLE |
| LOCATION | Порождение переменной типа LOCATION |
| SETTING | Порождение переменной типа SETTING |
| + | Логическое «И» |
| \| | Логическое «ИЛИ» |
| = | Присваивание |
| ?= | Проверка равенства |
| != | Проверка неравенства |

#### 3.1.1.6 Константы

В языке READER присутствует два вида констант: символьные и логические. Более подробно о них изложено ниже.

1) Символьные константы

Символьная строка — это последовательность символов, заключенная в знаки меньше-больше. В языке символьная константа представляется следующим образом:

<*любая последовательность символов*>

Разбивать строки на части недопустимо.

Примеры символьных строк:

1) <Hello, world! >
2) <Тестовая строка>
3) <”Здесь могла быть ваша реклама”>

2) Логические константы

В языке READER присутствуют две логические константы – {DA} и {NO}. Эти значения нельзя присваивать переменным, зато можно использовать для проверки успешности выполнения некоторых функций.

#### 3.1.1.7 Идентификаторы

Идентификатор — это последовательность из одной или более латинских букв и/или цифр. Идентификаторы служат именами переменных и фактов, используемых в программе. Идентификаторы вводятся вручную пользователем или генерируются программой по мере необходимости. После этого его можно использовать в последующих операторах программы.

Примеры идентификаторов:

1) ACT
2) CH1
3) EV12

Использование идентификаторов, совпадающих с ключевыми словами, приведёт к применению этих ключевых слов. Например, использование конструкции CHARACTER CHARACTER приведёт к порождению двух переменных типа CHARACTER со случайно сгенерированными идентификаторами.

#### 3.1.1.8 Ключевые слова

Ключевые слова — это предопределенные идентификаторы, которые имеют специальное значение для компилятора языка READER.

Список ключевых слов:

1) CHARACTER
2) ROLE
3) LOCATION
4) SETTING
5) {DA}
6) {NO}
7) GENERATE
8) NEAR
9) LIMIT

### 3.1.2 Класс программ

Поскольку READER является языком сверхвысокого уровня, то спектр задач, решаемых им, узок. Язык READER используется для настройки процесса генерации сюжета и обеспечения функционирования «материнской» интеллектуальной системы. Для решения задач иного плана он не предназначен.

### 3.1.3 Синтаксис языка

#### 3.1.3.1 Структура программы

На подготовительном этапе программа представляет из себя набор инструкций, которые вводит эксперт, чтобы обеспечить программу необходимыми данными для генерации сюжета.

Эксперт может устанавливать рамки, за которые программа не имеет права выходить при генерации, объявляя переменные и присваивая им значения, вводить обязательные сущности и так далее.

После того, как эксперт закончил составлять краткий план генерации сюжета, запускается сам процесс. В ходе работы интеллектуальной системы будут применяться правила, определяющие дальнейший процесс генерации. Правила представляю себя список условий – выражений на языке READER – при истинности которых будет выполнен список инструкций – также выражений на языке READER.

Если эксперт не написал предварительный план генерации, то он будет состоять из одной команды – GENERATE.

#### 3.1.3.2 Объявления переменных

Синтаксис:

[тип] [идентификатор] (= <значение для поля NAME>)

Объявление переменной определяет её имя и тип. Часть, заключённая в скобки, необязательна – это присвоение значения полю [идентификатор]: NAME.

Примеры:

- ROLE RL1 = <Дракон>;
- CHARACTER CH1, CH2;
- CHARACTER ACT = <Добряк>;

### 3.1.3.3 Выражения

Выражение — это комбинация операндов и операций, задающая порядок вычисления некоторого значения. Операции определяют действия, выполняемые над операндами. В языке READER операндами могут быть константы или переменные.

Результат вычисления выражения зависит от приоритета операций. Приоритет операций определяет группирование операндов в выражении и последовательность выполнения операций.

### 3.1.3.4 Операнды

Операндом выражения может быть константа или переменная. Эти операнды могут посредством так называемых первичных операций комбинироваться в первичные выражения — вызов функции, индексное выражение, выражение выбора структурного элемента. Эти первичные выражения, в свою очередь, являются операндами содержащего их выражения. Комбинация их с другими операциями приводит к образованию новых, более сложных выражений, также являющихся операндами содержащего их выражения, и т.д.

1) Константы

Операнду-константе соответствует значение и тип представляющей его константы. Символьная константа имеет тип SYMBOLIC. Логическая константа имеет тип LOGIC.

2) Идентификаторы

Идентификаторы – это имена переменных и фактов. Идентификатор должен быть уникальным в простанстве имён как переменных, так и фактов, По умолчанию значение у объекта отсутствует, но оно может быть задано экспертом позже, либо сгенерировано программой.

3) Вызовы фактов

Синтаксис:

<имя факта>[список аргументов]

<имя факта>(список аргументов)

Разница между этими двумя вызовами заключается в том, что первый вызов обеспечивает факту с заданными параметрами попадание в рабочую память, а второй проверяет, существует ли факт с подобным именем и параметрами в памяти программы.

4) Выбор поля переменной

Синтаксис:

<имя переменной>:<название поля>

Выражение выбора поля позволяет получить доступ к элементу переменной.

Примеры:

- RL1:NAME
- CH1:LOCATION

1) Операции

В зависимости от используемых операций выражения подразделяются на унарные, бинарные и выражения присваивания.

а) Унарная операция

В состав унарного выражения входят унарная операция и следующий после неё операнд.

Синтаксис:

<унарная-операция> <операнд>

Унарные операции будут рассмотрены далее в разделе “Операторы”.

б) В состав бинарного выражения входят два операнда, и бинарной операцией между ними.

Синтаксис:

<операнд1> <бинарная-операция> <операнд2>

Бинарные операции рассмотрены в разделе “Операторы”.

в) Выражения присваивания

Синтаксис выражений присваивания:

<операнд1> = <операнд2>

В качестве <операнда1> могут выступать только идентификаторы или поля переменных.

### 3.1.3.6 Операции

В языке READER операции имеют либо один операнд (унарные операции), либо два операнда (бинарные операции). Опе рация присваивания может быть только бинарной.

1) Унарные операции

Операции порождения, такие как CHARACTER, ROLE, LOCATION и SETTING, создают в программе переменную указанного типа. Если присутствует присваивание, то полю NAME этой переменной будет присвоено значение.

Пример:

CHARACTER CH1, CH2 = <Прохожий>

Результатом выполнения этой операции будет создание двух переменных типа CHARACTER с полем NAME, равным «Прохожий».

2) Бинарные операции

а) Логическое И (+)

Операция И объединяет между собой два условия (случае, если условий больше, результат объединения первых двух считается за одно). Если все операнды имеют значение {DA}, то и всё выражение целиком имеет значение {DA}. Значение {NO} вырабатывается при ложности хотя бы одного операнда.

б) Логическое ИЛИ (|)

Операция ИЛИ объединяет между собой два условия (случае, если условий больше, результат объединения первых двух считается за одно). Если хотя бы один операнд имеет значение {DA}, то и всё выражение целиком имеет значение {DA}. Значение {NO} вырабатывается при ложности всех операндов.

3) Операции отношения

В языке READER с помощью операций отношений выясняется, являются ли операнды равными или неравными. Операции возвращают {DA}, если предположение верно, и {NO}, если неверно.

Имеются следующие операции отношения:

Таблица 3 – Операции отношения

| Операция | Проверяемое отношение |
|---|---|
| ?= | Первый операнд равен второму операнду |
| != | Первый операнд не равен второму операнду |

Примеры:

CHARACTER CH1, CH2;

1) CH1:NAME ?= CH2:NAME
2) CH1:ROLE != <Дракон>

Если CH1:NAME и CH2:NAME равны, то выражение 1 значение {DA}, иначе –{NO}.

1) Присваивание

Операция присваивания обозначается знаком =. Значение правого операнда присваивается левому операнду. Левый операнд должен быть переменной того же типа, что и правый.

Операция имеет значение {DA}, если присваивание было произведено, иначе операция имеет значение {NO}.

Пример:

CHARACTER ACT = <Исполнитель>; значение <Исполнитель> присваивается переменной ACT:NAME, операция присваивания возвращает значение {DA}.

### 3.1.3.7 Прочие функции

1) NEAR.

Синтаксис:

NEAR[LOC1, LOC2];

NEAR(LOC1, LOC2);

Действие:

1). В первом случае функция NEAR установит связь между локациями LOC1 и LOC2 и вернёт {DA}, если удалось установить связь, {NO} в противном случае.

2). Во втором случае будет выполнена проверка существования связи между локациями LOC1 и LOC2. Операция вернёт {DA}, если связь существует, или {NO}, если связи нет.

2) LIMIT.

Синтаксис:

LIMIT[<тип переменной>, <предел>];

Действие:

Функция LIMIT устанавливает для типа переменной <тип переменной> устанавливает максимально допустимое количество <предел>. Сверх этого количества программа не будет порождать переменные данного типа. Если эксперт сначала установил предел, а затем сам превысил его, то предел будет увеличен.

Пример:

LIMIT[CHARACTER, 12];

3) GENERATE.

Синтаксис:

GENERATE;

Действие:

Функция запускает свободную генерацию сюжета. По умолчанию весь предварительный план состоит только из этой функции. Если эту функцию вписал эксперт, то программа будет игнорировать все команды, которые идут ниже этой.

### 3.2 Примеры программ

#### 3.2.1 Пример 1

```
LIMIT[CHARACTER, 8];
LIMIT[LOCATION, 1];
LIMIT[ROLE, 8];
GENERATE.
```

#### 3.2.2 Пример 2

```
CHARACTER CH1, CH2;
CH1:NAME = <>;
CH1:ROLE = <>;
CH2:NAME = <>;
CH2:ROLE = <>;
CHARACTER CH3 = <>;
CH3:ROLE = <>;
F1[CH1, CH2];
F2[CH3];
GENERATE.
```

## Заключение

Цель, поставленная вначале ВКР, была выполнена – разработан предметно-ориентированный язык для интеллектуальной системы генерации

сюжетов, а также изучена предметная область. Получены теоретические знания, а так же практические навыки по разработке трансляторов. Язык разработан для личного использования в исследовательских целях.

## Список использованных источников

1. Salltalk [Электронный ресурс]:// Свободная энциклопедия «Wikipedia» – Режим доступа: https://ru.wikipedia.org/wiki/Smalltalk Дата обращения: 14.05.17

2. Icon [Электронный ресурс]:// Свободная энциклопедия «Wikipedia» – Режим доступа: https://ru.wikipedia.org/wiki/Icon_(язык_программирования) Дата обращения: 14.05.17

3. Microsoft Visual Studio [Электронный ресурс]:// Свободная энциклопедия «Wikipedia» – Режим доступа: https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio#Visual_Studio_2012 Дата обращения: 17.05.17

Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт Космических и Информационных технологий
институт
Кафедра Информатики
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.С.Кузнецов
подпись инициалы, фамилия
« 08 » 06 2017г.

# БАКАЛАВРСКАЯ РАБОТА

09.03.04 «Программная инженерия»
код – наименование направления

Предметно-ориентированный язык для интеллектуальной системы генерации сюжетов
тема

Руководитель: _____ 08.06.17 доцент, к.т.н. А.С.Кузнецов
подпись, дата должность, ученая степень инициалы, фамилия

Выпускник: Головин, 08.06.17 Р.А.Головин
подпись, дата инициалы, фамилия

Нормоконтроль: _____ 08.06.17 доцент, к.т.н. О.А.Антамошкин
подпись, дата должность, ученая степень инициалы, фамилия

Красноярск 2017

# РЕФЕРАТ

Бакалаврская работа 30 стр., 1 рисунок, 1 источник.

Объект исследования – защита персональных данных.

Цель работы – разработать предметно-ориентированный язык сверхвысокого уровня для интеллектуальной системы генерации сюжетов.

Метод исследования – практический эксперимент.

Результат – изучена предметная область по данной теме, а также разработан язык сверхвысокого уровня.

**Оглавление**

## Введение

В наш век электроники, когда компьютеры проникли буквально всюду, почти не осталось сфер, где бы компьютер не смог заменить человека. Искусственный интеллект автоматически доказывает теоремы, переводит тексты с одного языка на другой, распознаёт изображения, управляет компьютерными противниками в игре…

В общем, область распространения ИИ велика, и если он ещё не достиг совершенства, то вплотную приблизился к нему. Однако ещё существует одна область, где компьютер человеку не конкурент. Эта область – творчество. В написании книг, стихов и музыки человек всё ещё безусловный фаворит. Но время не стоит на месте, и когда-нибудь монополия человека на творчество будет нарушена.

Цель данной работы – приблизить наступление тех времён, когда компьютерная программа сможет писать за человека книги. Оставим в стороне вопрос об этичности или не этичности этих намерений – он лежит за рамками рассматриваемой проблемы. Автор данного отчёт задался целью создать такую интеллектуальную систему, которая бы помогала начинающим авторам в их нелёгком ремесле.

Однако разработка подобной ИС подводит нас к другому вопросу. Если бы наша экспертная система имела бы вид простого логического аппарата, который, имея правила и факты, выводил бы из имеющегося искомое, то весь труд нашего эксперта сводился бы к придумыванию новых фактов и введению новых правил. Однако чем сложней будет система, тем больше труд эксперта будет походить на труд программиста-разработчика, что, безусловно, противоречит самой идее экспертных систем. Именно поэтому, желая не только облегчить труд эксперта, но и дать ему в руки мощный инструмент разработки, и была затеяна эта работа.

Чтобы обуздать наратающую сложность, нужно научить программу общаться с экспертом без посредничества программиста, создав для этого

простой, но функциональный язык, оперируя элементами которого, эксперт будет вводить в процесс генерации сюжета новые функциональные сущности.

Решено, что будущий язык будет интерпретироваться средствами языка C++ и представлять собой набор простых команд, комбинируя которые, можно контролировать процесс генерации сюжета. Данный язык будет обладать сверхвысоким уровнем абстракции и максимально простым синтаксисом.

# 1. Анализ и обзор языков программирования сверхвысокого уровня

## 1.1 Особенности предметной области и решаемых задач

На данный момент существует ряд программ-ассистентов, призванный выполнять примерно те же функции, что IDE в работе программиста или память переводов в работе переводчика. ПО такого типа позволяют работать с набором текстов будущего произведения художественной литературы как с проектом, поддерживая связь фрагментов-сцен в хронологическом порядке или по сюжетным линиям, а также поддерживают ведение вспомогательных баз данных, например для персонажей. К таким novel-writing программам относятся YWriter, oStorybook, Writer's Cafe и Scrivener.

Однако все эти программы лишь ассистируют, весь писательский труд всё равно выполняет человек. Разрабатываемая экспертная система предусматривает исключение человека из процесса создания литературного произведения.

## 1.2 Анализ существующих языков сверхвысокого уровня

Сверхвысокоуровневым языком программирования (VHLL) называют те языки, в которых достигается максимальный уровень абстракции. Программист описывает не «как нужно сделать», а «что нужно сделать».

Именно поэтому каждый подобный язык разрабатывается под решение ряда узкоспециализированных задач, и для решения других непригоден.

Из этого следует, что аналогов разрабатываемого языка не существует. Чтобы убедиться в этом, ниже будет приведён ознакомительный обзор двух языков сверхвысокого уровня.

### 1.2.1 Smalltalk

Smalltalk — объектно- ориентированный язык программирования с динамической типизацией, основанный на идее посылки сообщений, разработанный в Xerox PARC Аланом Кэйем, Дэном Ингаллсом, Тедом Кэглером, Адель Голдберг, и другими в 1970-х годах. Представляет собой интегрированную среду разработки и исполнения, объекты которой доступны для модификации через неё саму, и программирование в которой в итоге сводится к модификации её собственного поведения. Язык был представлен как Smalltalk-80.[1]

Smalltalk является одним из многих объектно-ориентированных языков, основанных на языке Симула, который сам оказал большое влияние на развитие многих объектно-ориентированных языков. Многие идеи 1980-х и 1990-х по написанию программ появились в сообществе Smalltalk. К ним можно отнести рефакторинг, шаблоны проектирования (применительно к ПО), карты «класс — обязанности — взаимодействие» и экстремальное

программирование в целом. Основатель концепции вики Уорд Каннингем также входит в сообщество Smalltalk.

Основными идеями Smalltalk являются:

- Всё — объекты, и всё их взаимодействие — через посылку сообщений. Строки, целые числа, логические значения, определения классов, блоки кода, стеки, память, составляющие самой интегрированной среды разработки и исполнения — всё представляется в виде объектов. У объектов есть методы и состояние. Любому объекту может быть послано любое сообщение. При отправке сообщения, среда исполнения всегда ищет у объекта-получателя подходящий метод и выполняет его, а если не находит — выполняет у объекта-получателя специальный метод для неопознанных сообщений. Объект-получатель сам определяет, является ли полученное сообщение правильным, и что надо сделать, чтобы его обработать.

- Всё доступно для изменения. Если вы хотите изменить саму интегрированную среду разработки и исполнения, вы можете сделать это в работающей системе, без остановки, перекомпиляции и перезапуска. Если вам необходима в языке новая управляющая конструкция языка, вы можете добавить её. В некоторых реализациях вы можете также изменить синтаксис языка или способ работы сборщика мусора.

- Динамическая типизация — это означает, что вы не указываете типы переменных в программе, что делает язык гораздо лаконичней. (Как объяснено выше, является ли операция правильной, определяет объект-получатель, а не компилятор).

Smalltalk также использует другие современные идеи:

- Сборка мусора встроена в язык и незаметна разработчику.

- Dynamic translation: современные коммерческие виртуальные машины компилируют байткоды в машинные коды для быстрого выполнения.

- Выполнение кода в виртуальной машине. Программы Smalltalk обычно компилируются в байткоды и выполняются виртуальной машиной, что позволяет выполнять их на любом оборудовании, для которого существует виртуальная машина.

Одной из особенностей Smalltalk является то, что даже такие традиционные конструкции, как if-then-else, for, while, и т. д. не являются частью языка. Все они реализованы с помощью объектов. Например, решение принимается с помощью посылки сообщения ifTrue: логическому объекту, и передаёт управление фрагменту текста, если логическое значение истинно.

Собственно встроенных синтаксических конструкций в языке не много:

- посылка сообщения объекту с возможной передачей ему других объектов;

- присваивание объекта переменной;

- возвращение объекта из метода;

- и несколько синтаксических конструкций для определения объектов-литералов и временных переменных.

Аналогом механизма обмена сообщениями Smalltalk является сеть интернет: можно представить каждый объект как веб-сервер, отвечающий на запросы. При этом, сервер на запросы может просто выдавать заранее предопределённый ответ, например веб-страницу, расположенную по определённому пути; может перенаправить запрос-сообщение другому объекту, аналог — прокси-сервер; может изменить запрос по определённым правилам, аналог — техника url rewriting, и конечно же может сформировать абсолютно новую страницу, соответствующую данным, переданным с сообщением. Если для реакции на сообщение у объекта нет предопределённого

метода, то среда вызывает у получателя метод #doesNotUnderstand:, так же, как веб-сервер возвращает страницу с сообщением об ошибке, если задан несуществующий путь к веб-странице.

### 1.2.2 Icon

Icon — язык программирования, унаследовавший идеологию более раннего языка того же автора Снобол. Название языка не имеет ничего общего с «иконками», а является сокращением от слова англ. iconoclastic (иконоборческий), используемом в смысле борьбы с конформизмом в разработке языков программирования. [2]

Это сверхвысокоуровневый язык программирования, в который интегрированы механизмы сопоставления с образцом и бэктрекинга, что сближает его с языками логического программирования.

Язык является динамически типизированным, имеет встроенные мощные типы данных. Процедуры в Icon’е относятся к величинам первого класса, что означает возможность присваивания переменным значений самих процедур, а не результатов их выполнения. Существует механизм со-выражений, позволяющий создавать сопрограммы.

Любое предложение (оператор) языка Icon на самом деле является выражением и может возвращать значения. Выражения в Icon, помимо собственно возвращаемых ими значений, которых может быть любое количество, производит одно из двух состояний — успех англ. success или неудачу англ. failure. Успех или неудача выражений используются вместо булевых значений управляющими структурами Icon. Благодаря этому мы можем писать конструкции типа:

```
if a := read() then write(a)
```

Более того, поскольку состояние «неудачи» имеет свойство «всплывать» из вложенных вызовов функций, становясь результатом внешней функции, можно использовать ещё более краткие идиомы, вроде:

```
while write(read())
```

для чтения потока ввода и дублирования его в поток вывода (echo).

Операции сравнения, такие > и <, могут быть успешны и производить результат, равный значению их второго аргумента, или неудачны, и не производить никакого результата. Благодаря этому в Icon можно писать выражения типа if a < b < c then ….

Связанная с успехом и неудачей концепция называется в Icon целенаправленным выполнением. Это способ, благодаря которому вычисление может продолжаться, пока не будет достигнута некая цель. В вышеприведённом примере с echo цель — чтение всего содержимого файла на входе, пока не будет встречен конец файла. Цель задаётся непосредственно, а не с помощью дополнительных проверок кодов возврата или чего то подобного. Целенаправленное вычисление реализуется с помощью бэктрэкинга, это очень мощный механизм.

## 1.3 Выделение существенных особенностей разрабатываемого языка сверхвысокого уровня

На основе анализа существующих VHLL был составлен общий список требований к разрабатываемому языку:

- За каждой из существующих инструкций должно стоять полностью завершённое действие;
- Каждая команда своим названием должна максимально точно передавать своё назначение;
- Названия всех конструкций, используемых в языке, должны быть простыми и легко запоминаться далёкими от программирования людьми.

Стоит, однако, заметить, что сформулированные требования слишком абстрактны, чтобы их формализировать. Не исключено, что найдутся пользователи, для которых синтаксис языка окажется слишком сложным.

# 2. Используемые технологии

## 2.1 Выбор средства разработки. Язык программирования

Для решения поставленной задачи был выбран язык программирования C++, поскольку интеллектуальная система, для общения с которой и разрабатывается данный язык, была реализована средствами этого языка.

## 2.2 Среда разработки

Поскольку автор данной работы имеет большой опыт в работе со средой разработки Microsoft Visual Studio, то для решения поставленной задачи была выбрана именно она, а конкретно Visual Studio 2012.

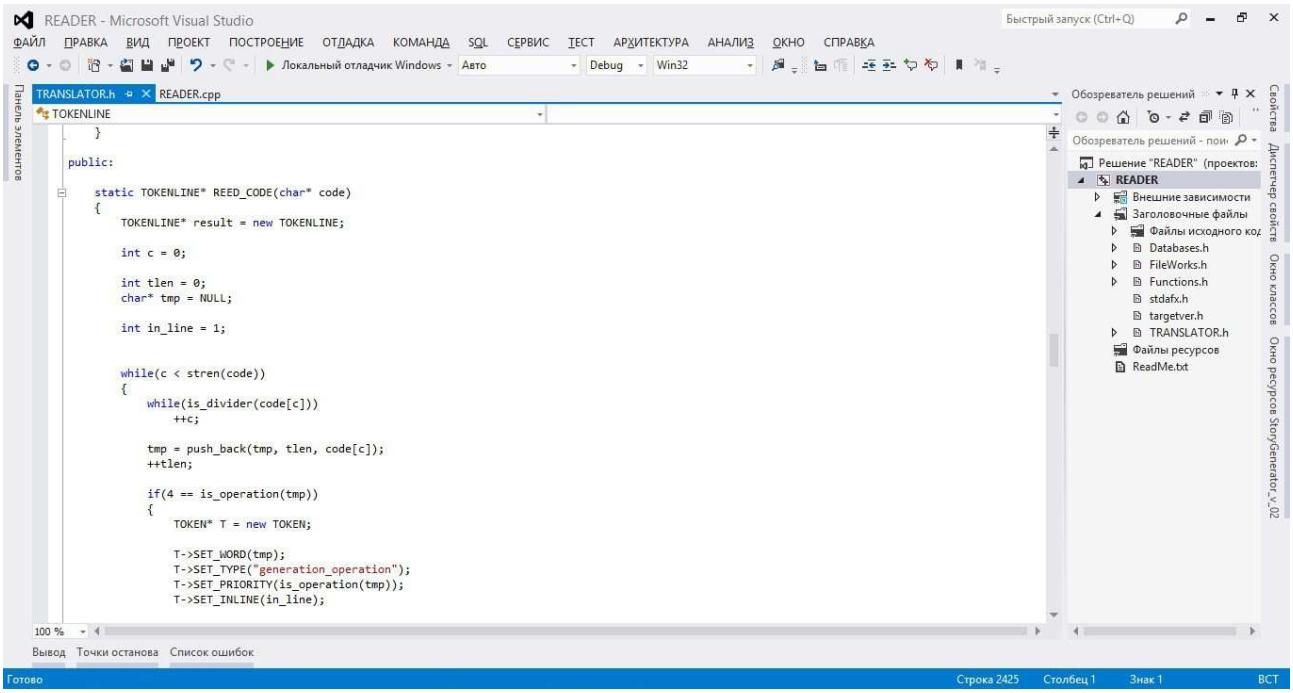


Рисунок 1 – Интерфейс Mircosoft Visual Studio 2012

Одним из достоинств данной среды является возможность разрабатывать как консольные приложения, так и приложения с графическим

интерфейсом, а также веб-сайты, веб-приложения, веб-службы как в родном, так и в управляемом кодах для всех платформ, поддерживаемых Windows, Windows Mobile, Windows CE, .NET Framework, Xbox, Windows Phone .NET Compact Framework и Silverlight.

Visual Studio включает в себя редактор исходного кода с поддержкой технологии IntelliSense и возможностью простейшего рефакторинга кода. Встроенный отладчик может работать как отладчик уровня исходного кода, так и отладчик машинного уровня. Остальные встраиваемые инструменты включают в себя редактор форм для упрощения создания графического интерфейса приложения, веб-редактор, дизайнер классов и дизайнер схемы базы данных. Visual Studio позволяет создавать и подключать сторонние дополнения (плагины) для расширения функциональности практически на каждом уровне, включая добавление поддержки систем контроля версий исходного кода (как, например, Subversion и Visual SourceSafe), добавление новых наборов инструментов (например, для редактирования и визуального проектирования кода на предметно-ориентированных языках программирования) или инструментов для прочих аспектов процесса разработки программного обеспечения (например, клиент Team Explorer для работы с Team Foundation Server). [3]

## 3. Реализация и описание

### 3.1 Описание языка

#### 3.1.1 Лексические соглашения

##### 3.1.1.1 Алфавит

Множество символов языка READER содержит заглавные буквы латинского алфавита, цифры и знаки пунктуации, которые имеют определенный смысл для компилятора языка. Программы строятся путем комбинирования символов из множества символов языка READER.

Языковые конструкции могут содержать только символы из множества символов языка READER, однако внутри символьных строк может быть использован любой представимый символ. При обнаружении неверно использованных символов компилятор выдает сообщение об ошибке.

Ниже описывается символы из множества символов языка READER и объясняются правила их использования.

##### 3.1.1.2 Буквы и цифры

Множество символов языка READER включает прописные буквы латинского алфавита и арабские цифры:

- прописные латинские буквы: ABCDEFGHIJKLMNOPQRSTUVWXYZ;
- десятичные цифры: 0123456789.

Буквы и цифры используются при формировании констант, переменных и ключевых слов (эти конструкции описаны ниже).

##### 3.1.1.3 Пробельные символы

В языке READER лексемы разделяются знаком пробела. Символа перевода на новую строку нет, т.к. каждая команда и так пишется в отдельной строке.

Лексемы обязаны отделяться друг от друга как минимум одним знаком пробела.

### 3.1.1.4 Разделители

В языке READER разделительные символы несут особый смысл, т.к. каждый из них выполняет различные функции.

Таблица 1 – Разделители языка READER

| Символ | Наименование |
|---|---|
| , | Запятая |
| . | Точка |
| ; | Точка с запятой |
| : | Двоеточие |
| ( | Левая круглая скобка |
| ) | Правая круглая скобка |
| { | Левая фигурная скобка |
| } | Правая фигурная скобка |
| < | Знак “меньше” |
| > | Знак “больше” |
| [ | Левая квадратная скобка |
| ] | Правая квадратная скобка |
| ! | Восклицательный знак |
| + | Знак плюс |
| ? | Знак вопроса |
| = | Знак равенства |
| \| | Вертикальная черта |

Более подробно о функциях этих символов в языке READER будет сказано ниже.

Элементы множества представимых символов, которые не представлены в данном списке (в частности, русские буквы), могут быть использованы только в символьных константах.

### 3.1.1.5 Операции

Операции — это определённая комбинация символов, влекущая за собой определённые действия в программе. С точки зрения компилятора, каждая из этих конструкций – самостоятельная лексема.

В таблице 2 представлен список операций. Операции должны использоваться точно так, как они представлены в таблице, добавление

пробельных символов в многосимвольные операции недопустимо.

Таблица 2 – Операции языка READER

| Операция | Наименование |
|---|---|
| CHARACTER | Порождение переменной типа CHARACTER |
| ROLE | Порождение переменной типа ROLE |
| LOCATION | Порождение переменной типа LOCATION |
| SETTING | Порождение переменной типа SETTING |
| + | Логическое «И» |
| \| | Логическое «ИЛИ» |
| = | Присваивание |
| ?= | Проверка равенства |
| != | Проверка неравенства |

#### 3.1.1.6 Константы

В языке READER присутствует два вида констант: символьные и логические. Более подробно о них изложено ниже.

1) Символьные константы

Символьная строка — это последовательность символов, заключенная в знаки меньше-больше. В языке символьная константа представляется следующим образом:

<*любая последовательность символов*>

Разбивать строки на части недопустимо.

Примеры символьных строк:

1) <Hello, world! >
2) <Тестовая строка>
3) <”Здесь могла быть ваша реклама”>

2) Логические константы

В языке READER присутствуют две логические константы – {DA} и {NO}. Эти значения нельзя присваивать переменным, зато можно использовать для проверки успешности выполнения некоторых функций.

#### 3.1.1.7 Идентификаторы

Идентификатор — это последовательность из одной или более латинских букв и/или цифр. Идентификаторы служат именами переменных и фактов, используемых в программе. Идентификаторы вводятся вручную пользователем или генерируются программой по мере необходимости. После этого его можно использовать в последующих операторах программы.

Примеры идентификаторов:

1) ACT
2) CH1
3) EV12

Использование идентификаторов, совпадающих с ключевыми словами, приведёт к применению этих ключевых слов. Например, использование конструкции CHARACTER CHARACTER приведёт к порождению двух переменных типа CHARACTER со случайно сгенерированными идентификаторами.

#### 3.1.1.8 Ключевые слова

Ключевые слова — это предопределенные идентификаторы, которые имеют специальное значение для компилятора языка READER.

Список ключевых слов:

1) CHARACTER
2) ROLE
3) LOCATION
4) SETTING
5) {DA}
6) {NO}
7) GENERATE
8) NEAR
9) LIMIT

### 3.1.2 Класс программ

Поскольку READER является языком сверхвысокого уровня, то спектр задач, решаемых им, узок. Язык READER используется для настройки процесса генерации сюжета и обеспечения функционирования «материнской» интеллектуальной системы. Для решения задач иного плана он не предназначен.

### 3.1.3 Синтаксис языка

#### 3.1.3.1 Структура программы

На подготовительном этапе программа представляет из себя набор инструкций, которые вводит эксперт, чтобы обеспечить программу необходимыми данными для генерации сюжета.

Эксперт может устанавливать рамки, за которые программа не имеет права выходить при генерации, объявляя переменные и присваивая им значения, вводить обязательные сущности и так далее.

После того, как эксперт закончил составлять краткий план генерации сюжета, запускается сам процесс. В ходе работы интеллектуальной системы будут применяться правила, определяющие дальнейший процесс генерации. Правила представляю себя список условий – выражений на языке READER – при истинности которых будет выполнен список инструкций – также выражений на языке READER.

Если эксперт не написал предварительный план генерации, то он будет состоять из одной команды – GENERATE.

#### 3.1.3.2 Объявления переменных

Синтаксис:

[тип] [идентификатор] (= <значение для поля NAME>)

Объявление переменной определяет её имя и тип. Часть, заключённая в скобки, необязательна – это присвоение значения полю [идентификатор]: NAME.

Примеры:

- ROLE RL1 = <Дракон>;
- CHARACTER CH1, CH2;
- CHARACTER ACT = <Добряк>;

### 3.1.3.3 Выражения

Выражение — это комбинация операндов и операций, задающая порядок вычисления некоторого значения. Операции определяют действия, выполняемые над операндами. В языке READER операндами могут быть константы или переменные.

Результат вычисления выражения зависит от приоритета операций. Приоритет операций определяет группирование операндов в выражении и последовательность выполнения операций.

### 3.1.3.4 Операнды

Операндом выражения может быть константа или переменная. Эти операнды могут посредством так называемых первичных операций комбинироваться в первичные выражения — вызов функции, индексное выражение, выражение выбора структурного элемента. Эти первичные выражения, в свою очередь, являются операндами содержащего их выражения. Комбинация их с другими операциями приводит к образованию новых, более сложных выражений, также являющихся операндами содержащего их выражения, и т.д.

1) Константы

Операнду-константе соответствует значение и тип представляющей его константы. Символьная константа имеет тип SYMBOLIC. Логическая константа имеет тип LOGIC.

2) Идентификаторы

Идентификаторы – это имена переменных и фактов. Идентификатор должен быть уникальным в простанстве имён как переменных, так и фактов, По умолчанию значение у объекта отсутствует, но оно может быть задано экспертом позже, либо сгенерировано программой.

3) Вызовы фактов

Синтаксис:

<имя факта>[список аргументов]

<имя факта>(список аргументов)

Разница между этими двумя вызовами заключается в том, что первый вызов обеспечивает факту с заданными параметрами попадание в рабочую память, а второй проверяет, существует ли факт с подобным именем и параметрами в памяти программы.

4) Выбор поля переменной

Синтаксис:

<имя переменной>:<название поля>

Выражение выбора поля позволяет получить доступ к элементу переменной.

Примеры:

- RL1:NAME
- CH1:LOCATION

1) Операции

В зависимости от используемых операций выражения подразделяются на унарные, бинарные и выражения присваивания.

а) Унарная операция

В состав унарного выражения входят унарная операция и следующий после неё операнд.

Синтаксис:

<унарная-операция> <операнд>

Унарные операции будут рассмотрены далее в разделе “Операторы”.

б) В состав бинарного выражения входят два операнда, и бинарной операцией между ними.

Синтаксис:

<операнд1> <бинарная-операция> <операнд2>

Бинарные операции рассмотрены в разделе “Операторы”.

в) Выражения присваивания

Синтаксис выражений присваивания:

<операнд1> = <операнд2>

В качестве <операнда1> могут выступать только идентификаторы или поля переменных.

### 3.1.3.6 Операции

В языке READER операции имеют либо один операнд (унарные операции), либо два операнда (бинарные операции). Опе рация присваивания может быть только бинарной.

1) Унарные операции

Операции порождения, такие как CHARACTER, ROLE, LOCATION и SETTING, создают в программе переменную указанного типа. Если присутствует присваивание, то полю NAME этой переменной будет присвоено значение.

Пример:

CHARACTER CH1, CH2 = <Прохожий>

Результатом выполнения этой операции будет создание двух переменных типа CHARACTER с полем NAME, равным «Прохожий».

2) Бинарные операции

а) Логическое И (+)

Операция И объединяет между собой два условия (случае, если условий больше, результат объединения первых двух считается за одно). Если все операнды имеют значение {DA}, то и всё выражение целиком имеет значение {DA}. Значение {NO} вырабатывается при ложности хотя бы одного операнда.

б) Логическое ИЛИ (|)

Операция ИЛИ объединяет между собой два условия (случае, если условий больше, результат объединения первых двух считается за одно). Если хотя бы один операнд имеет значение {DA}, то и всё выражение целиком имеет значение {DA}. Значение {NO} вырабатывается при ложности всех операндов.

3) Операции отношения

В языке READER с помощью операций отношений выясняется, являются ли операнды равными или неравными. Операции возвращают {DA}, если предположение верно, и {NO}, если неверно.

Имеются следующие операции отношения:

Таблица 3 – Операции отношения

| Операция | Проверяемое отношение |
|---|---|
| ?= | Первый операнд равен второму операнду |
| != | Первый операнд не равен второму операнду |

Примеры:

CHARACTER CH1, CH2;

1) CH1:NAME ?= CH2:NAME
2) CH1:ROLE != <Дракон>

Если CH1:NAME и CH2:NAME равны, то выражение 1 значение {DA}, иначе –{NO}.

1) Присваивание

Операция присваивания обозначается знаком =. Значение правого операнда присваивается левому операнду. Левый операнд должен быть переменной того же типа, что и правый.

Операция имеет значение {DA}, если присваивание было произведено, иначе операция имеет значение {NO}.

Пример:

CHARACTER ACT = <Исполнитель>; значение <Исполнитель> присваивается переменной ACT:NAME, операция присваивания возвращает значение {DA}.

### 3.1.3.7 Прочие функции

1) NEAR.

Синтаксис:

NEAR[LOC1, LOC2];

NEAR(LOC1, LOC2);

Действие:

1). В первом случае функция NEAR установит связь между локациями LOC1 и LOC2 и вернёт {DA}, если удалось установить связь, {NO} в противном случае.

2). Во втором случае будет выполнена проверка существования связи между локациями LOC1 и LOC2. Операция вернёт {DA}, если связь существует, или {NO}, если связи нет.

2) LIMIT.

Синтаксис:

LIMIT[<тип переменной>, <предел>];

Действие:

Функция LIMIT устанавливает для типа переменной <тип переменной> устанавливает максимально допустимое количество <предел>. Сверх этого количества программа не будет порождать переменные данного типа. Если эксперт сначала установил предел, а затем сам превысил его, то предел будет увеличен.

Пример:

LIMIT[CHARACTER, 12];

3) GENERATE.

Синтаксис:

GENERATE;

Действие:

Функция запускает свободную генерацию сюжета. По умолчанию весь предварительный план состоит только из этой функции. Если эту функцию вписал эксперт, то программа будет игнорировать все команды, которые идут ниже этой.

### 3.2 Примеры программ

#### 3.2.1 Пример 1

```
LIMIT[CHARACTER, 8];
LIMIT[LOCATION, 1];
LIMIT[ROLE, 8];
GENERATE.
```

#### 3.2.2 Пример 2

```
CHARACTER CH1, CH2;
CH1:NAME = <>;
CH1:ROLE = <>;
CH2:NAME = <>;
CH2:ROLE = <>;
CHARACTER CH3 = <>;
CH3:ROLE = <>;
F1[CH1, CH2];
F2[CH3];
GENERATE.
```

## Заключение

Цель, поставленная вначале ВКР, была выполнена – разработан предметно-ориентированный язык для интеллектуальной системы генерации

сюжетов, а также изучена предметная область. Получены теоретические знания, а так же практические навыки по разработке трансляторов. Язык разработан для личного использования в исследовательских целях.

## Список использованных источников

1. Salltalk [Электронный ресурс]:// Свободная энциклопедия «Wikipedia» – Режим доступа: https://ru.wikipedia.org/wiki/Smalltalk Дата обращения: 14.05.17

2. Icon [Электронный ресурс]:// Свободная энциклопедия «Wikipedia» – Режим доступа: https://ru.wikipedia.org/wiki/Icon_(язык_программирования) Дата обращения: 14.05.17

3. Microsoft Visual Studio [Электронный ресурс]:// Свободная энциклопедия «Wikipedia» – Режим доступа: https://ru.wikipedia.org/wiki/Microsoft_Visual_Studio#Visual_Studio_2012 Дата обращения: 17.05.17