**Аннотация**

В рамках данного дипломного проекта разработан интерпретатор языка Пролог с визуальным вводом программы и возможностью работы с универсальными базами данных.

Настоящая пояснительная записка включает в себя описание механизма вывода в языке Пролог и роли логического программирования в развитии вычислительной техники.

Приводится описание разработанного интерпретатора, а также необходимые для работы с ним документа: требования по эксплуатации, руководство по установке, руководство пользователя, тексты программ.

Приводится расчет затрат на разработку программного продукта.

Приводится анализ опасных и вредных факторов, возникающих при эксплуатации программы интерпретатора.

**Содержание**

Введение. 4

1. Исследовательская часть. 8

1.1. Роль реляционных языков в развитии вычислительной техники. 8

1.2. Основные механизмы дедукции. 10

1.3. Исчисление предикатов как язык для решения задач. 12

1.3.1 Унификация и принцип резольвенции в исчислении предикатов. 13

1.3.2. Методы поиска доказательства в исчислении предикатов. 15

1.3.2.1. Исчисление предикатов при решении задач. 15

1.3.2.2. Стратегии перебора. 16

1.3.2.3. Стратегии упрощения. 17

1.3.2.4. Стратегии очищения. 18

1.3.2.5. Формы доказательства с отфильтровыванием предшествующих вершин. 4

1.3.2.6. Стратегии поддерживающего множества. 4

1.3.2.7. Стратегии упорядочения. 5

1.4. Анализ характеристик существующих интерпретаторов. 6

1.5. Необходимость разработки интерпретатора языка Пролог. 8

1.6. Выбор языка программирования. 8

2. Конструкторская часть. 10

2.1. Синтаксис программ на Прологе в нотации Бэкуса-Наура. 10

2.2. Общая структура интерпретатора. 10

2.2.1. Принцип работы предкомпилятора. 11

2.2.1.1. Работа лексического анализатора. 11

2.2.1.2. Синтаксический анализатор. 12

2.2.1.3. Анализ арифметического выражения. 13

2.2.1.4. Анализ параметров предикатов. 15

2.2.1.5. Проверка типов параметров. 15

2.3. Работа интерпретатора. 16

2.3.1. Выполнение обращений к базам данных. 18

2.3.2. Вычисление арифметических выражений. 19

2.4. Объекты, используемые компилятором и интерпретатором. 19

2.4.1. Объекты переменных TPrologVariable, TPrologVariables, TPrologVariableList, TPrologVariableStruct. 19

2.5.2. Стандартные функции и предикаты. 21

2.6. Представление Пролог-программы в виде объектов. 21

2.7. Основные модули. 23

2.8. Демонстрационная программа по выбору конфигурации компьютера. 25

3. Технологическая часть. 28

3.1. Требования по эксплуатации интерпретатора языка Пролог. 28

3.2. Установка системы. 28

3.3. Руководство пользователя программы интерпретатора языка Пролог. 29

3.3.1. Запуск программы. 29

3.3.2. Перечень функций, реализуемых системой. 29

3.3.3. Редактирование Пролог-программы. 29

3.3.4. Запуск программы на Прологе и ее отладка. 34

3.3.5. Работа с меню. 35

3.4. Описание процесса выполнения программы, написанной на языке Пролог. 38

3.5. Общие сведенья об интерпретаторе. 39

3.6. Особенности работы Пролог-программы с базами данных. 39

3.7. Описание стандартных предикатов. 40

3.8. Описание функций. 43

4. Организационно-экономическая часть. 45

4.1. Расчет затрат на разработку интерпретатора Пролог. 45

5. Промышленная экология и безопасность. 47

5.1. Введение. 47

5.2. Анализ характера загрязнения окружающей среды при производстве вычислительной техники. 48

5.2.1. Источники загрязнения. 48

5.2.2. Очистка воздуха от вредных примесей. 50

5.3. Анализ влияния опасных и вредных факторов, при эксплуатации программы интерпретатора Пролог. 52

5.3.1. Повышенный уровень шума на рабочем месте. 53

5.3.2. Опасный уровень напряжения электрической цепи, замыкание которой может произойти через человека. 54

5.3.3. Пожарная опасность. 55

5.3.4. Повышенный уровень электромагнитных излучений. 56

5.3.5 Повышенная яркость света. 57

5.3.6. Прямая и отраженная блеклость. 58

5.3.7. Нарушение микроклимата рабочих помещений. 58

5.3.8. Защита от психофизиологических факторов. 59

5.4. Анализ использования защитных экранов для снижения влияния опасных и вредных факторов, во время работы на автоматизированном рабочем месте. 60

5.4.1 Основные функции защитных экранов, необходимые для снижения влияния вредных и опасных факторов, во время работы с программным средством. 60

5.4.1.1. Защита от электростатического и электромагнитного воздействий. 61

5.4.1.2. Защита от рентгеновского излучения. 61

5.4.1.3. Защита от ультрафиолетового излучения. 61

5.4.2. анализ основных типов защитных экранов, которые приемлемы для снижения влияния вредных и опасных факторов, во время с программным средством. 62

5.4.2.1. Сетчатый (частичная защита зрения). 62

5.4.2.2. Стеклянный двухслойный с заземлением (частичная защита зрения, частичная защита здоровья). 62

5.4.2.3. Стеклянный многослойный с заземлением (полная защита зрения, полная защита здоровья). 63

5.5. Расчет необходимого звукопоглощения, при работе с АРМ. 63

Заключение 66

Литература 67

Приложения 68

Приложение 1 68

Приложение 2 73

Приложение 3 81

Приложение 4. 87

# Введение

Разрабатываемый программный продукт предназначен для визуального создания, редактирования и интерпретации программ, написанных на языке Пролог с возможностью работы с универсальными базами данных.

Постоянно возрастающий объем информации, которую необходимо обрабатывать современным компьютерам предъявляет более широкие требования к современным базам данных. Если на заре развития компьютерной техники база данных была обычным файлом, который представлял собой типизированный файл, к которому можно было обращаться по абсолютному номеру записи, то сейчас база данных представляет собой интеллектуальную среду, которая включает в себя подчас несколько таблиц с данными, связанными между собой. Причем конечный пользователь из-за сложности структуры базы не знает, в каком месте файла хранятся данным, с которыми он работает. Современные базы данных обладают встроенными возможностями защиты прав доступа, а также способами поддержки целостности данных и их непротиворечивости. Это достигается за счет включения в сами базы данных отдельный частей программы, которые действуют независимо от пользовательской программы как программы-серверы. Доступ к таблицам стал значительно проще за счет использования языка SQL, который помогает быстро выбирать нужный пользователю сегмент информации из общего объема, также удалять ненужную информацию и добавлять новую.

Базы данных сейчас используются не только как обычные хранилища информации, но и как хранилища знаний. Поэтому появляется новое требование к базам данных, которое пришло от баз знаний, - это возможность логического вывода новых знаний из уже известных, а также работа в режиме экспертной системы.

В самом узком смысле термин *экспертная система* используется для описания одной из небольшого числа программ, разработанных общепризнанными специалистами в *инженерии знаний*. Назначение этих программ состоит в воспроизведении возможности решения задач, которыми обладает эксперт. Большинство экспертных систем не может полностью заменить человека. Такие системы используются для повышения эффективности работы и расширения знаний персонала средней квалификации.

В широком смысле экспертная система - это любая программа, применяемая для экспертных консультаций. Данное определение охватывает все программы, используемые в качестве экспертных систем, не учитывая того момента, что истинные эксперты могли и не участвовать в создании этих программ.

В любой системе экспертных консультаций обязательно должны иметься следующие три компоненты:

1. *язык представления знаний*, с помощью которого можно интуитивно представить знания о сложной области;
2. *стратегия решения задач*, позволяющая выполнять действия с представленными знаниями столь же компетентно, как это делают эксперты-люди;
3. *интерфейс с пользователем*, обеспечивающий естественность и удобство доступа к знаниям, которыми обладает программа, и способный объяснять свои ответы, как неопытным пользователям, так и пользователям-экспертам[1].

Традиционным языком в создании экспертным систем является язык Пролог. Это классический язык логического программирования. Он имеет встроенный механизм вывода, основанный на принципе резолюций, помогающий формально обращаться со знаниями. Кроме того, язык Пролог является реляционным языком программирования, то есть оптимально приспособлен для работы с реляционными базами данных. Так как Пролог оперирует правилами, программисту не нужно задумываться над программированием последовательности действий для машины, как это делается при программировании на процедурных языках. Программист просто составляет совокупность правил, описывающую данную предметную область, а Пролог выполняет составленную программу, используя алгоритм бэктрекинга.

Как указывалось ранее, немаловажным в экспертной системе является интерфейс с пользователем как профессионалам в данной предметной области, так и непрофессионалом. Вследствие этого, интерфейс с человеком должен осуществляться на естественном языке. Так как Пролог является декларативным языком и основан на исчислении высказываний, то на нем достаточно несложно можно написать обработку естественного языка.

Актуальным вопросом всегда был вид представления знаний. Изначально в Прологе база знаний хранилась в текстовом файле в формате языка Пролог, который загружался в память, и компилировался во время исполнения программы. Это было явным недостатком Пролога, так как ведение такой базы знаний возможно только человеком и только вручную, то есть, используя только текстовый редактор, а не какую-то специализированную программу. Это ограничивало возможности применения Пролога. Кроме этого, еще одним ограничением было то, что Прологу приходилось загружать в память всю базу знаний. То есть, если оперативной памяти компьютера не хватало, то программа не могла работать.

Подключение к Прологу универсальных баз данных позволяет снять эти два ограничения. Универсальными базами данных могут пользоваться другие программы, которые специализированы для ввода того или иного формата представления знаний. Таким образом, увеличивается скорость и качество ввода знаний. Более того, знания могут добавляться прямо по ходу выполнения Пролог-программы. Становится возможным поступление данных сразу с разных точек (с разных компьютеров).

Также использование баз данных позволяет снять ограничение на объем оперативной памяти компьютера, так как менеджер баз данных грузит в память только те данные, которые требуются в настоящий момент, а не все сразу. Также не нужна компиляция знаний во время выполнения программы, так как они уже находятся в нужном формате. Использование баз данных замедляет скорость работы программы, так как ей приходится обращаться к диску за данными, но менеджер баз данных позволяет уменьшить это замедление за счет кэширования данных и опережающего чтения.

Разрабатываемая система позволяет снять высокие требования к объему памяти компьютера, так как использует универсальные базы данных. Теперь база знаний может храниться в файле базы данных и загружаться в память компьютера по необходимости. Использование баз данных позволяет работать с одной базой знаний нескольким программам, а также предоставляется возможность удобного редактирования базы с помощью других программ. Загрузка и поиск записей в БД возложена на операционную систему, которая централизовано и эффективно распределяет доступ к базам данных для нескольких программ, а также за счет встроенного кэширования позволяет снизить зависимость скорости выполнения программы от скорости работы диска.

Система содержит интегрированную среду разработчика, которая предоставляет широкие возможности по визуальному вводу, редактированию и отладке программы на Прологе.

# 1 Исследовательская часть

## 1.1 Роль реляционных языков в развитии вычислительной

## техники

В настоящее время растет круг практических систем, использующих достижения искусственного интеллекта на современных ЭВМ, появились престижные проекты создания ЭВМ новых поколений, в которых интеллектуальный интерфейс с конечным пользователем (непрофессионалом в информатике) является центральным элементом. В японском проекте создания ЭВМ пятого поколения язык Пролог прямо называется базовым языком программирования[5].

Близость Пролога к конечному пользователю объясняется тем, что он является декларативным языком. Чтобы задать определенную последовательность действий, приводящих к решению задачи, в программе на Прологе необходимо описать ее содержание в терминах объектов и отношений между ними. Таким образом, вместо алгоритма решения задачи, программист составляет ее логическую спецификацию.

Что же касается построения алгоритма, то это автоматически выполняется самой Пролог-системой с помощью встроенного механизма вывода. При этом цель решения задачи представляется в виде запроса к базе знаний, в которой содержится описание предметной области задачи. Для поиска в базе данных значений, требуемых в запросе, Пролог-система инициирует механизм вывода. Таким образом, вычисления в Прологе представляют собой процесс дедукции, направленный на построение доказательства целевого утверждения задачи.[1]

Семантика языка Пролог значительно отличается от семантики других языков программирования. Вообще, языки программирования можно разбить на три широкие категории в соответствии с природой семантики этих языков:

* Процедурные языки;
* Функциональные языки;
* Реляционные языки.

Смысл конструкции процедурного языка определяется в терминах поведения компьютера при выполнении этой конструкции. В функциональном языке смысл конструкции (например, вызов функции) определяется в терминах значения, которое она вырабатывает. А в реляционном языке – отношение между отдельными сущностями или классами сущностей. Таким образом, процедурные языки можно назвать языками низкого уровня, так как они дают картину мира, близкую к взгляду на мир с позиций компьютера. Языки же высокого уровня обеспечивают взгляд на мир, приближающийся к картине мира, представленной в спецификации задачи. При использовании идеального реляционного языка становится возможным написание программы, структурно изоморфной по отношению к своей спецификации, то есть для каждой вариации формы спецификации будет существовать соответствующая вариация формы программы.

Хотя Пролог и далек от идеального реляционного языка, он в то же время достаточно близок к такому языку. Это позволяет программисту воспользоваться упомянутыми выше преимуществами идеальных реляционных языков. Программист может мыслить в терминах структуры отношений, не заботясь о точности их трансляции в программу. То есть данный язык позволяет работать специалисту на высоком концептуальном уровне.[1]

Возможны три точки зрения программиста на Пролог-программу.

1. *Реляционный подход*. При этом программа рассматривается как множество взаимоопределенных, возможно очень сложных, взаимоотношений. Реляционный подход пригоден в том случае, когда хорошо известна структура предметной области. Процесс программирования при этом сводится к аксиоматическому определению каждого отношения. Входной и выходной потоки, а также поведение программы являются результатами действия запросов к отношению. Если отношение реализовано корректно, то будут правильными также входной и выходной потоки.

2. *Подход к программе с позиций потока данных.* Такой взгляд на программу уместен, когда известна природа выходного потока (то есть множество ответов). При программировании реализуется такая внутренняя структура программы, которая создает желаемый выходной поток. Если важен порядок следования ответов в выходном потоке, то при построении программы следует в явной форме учитывать процедурные факторы.

3. *Поведенческий подход к программе.* Поведенческий подход пригоден тогда, когда известно лишь желаемое поведение программы. Процесс программирования связан с построением такой внутренней структуры программы, которая обеспечит заданное поведение. При разработке такой программы следует обязательно учитывать процедурные факторы и влияние побочных эффектов.

Эти три подхода не являются взаимоисключающими, они представляют собой разные способы мышления в процессе программирования. С точки зрения стиля программирования рекомендуется применять либо реляционный подход, либо подход к программе с позиций потока данных, а к поведенческому следует прибегать лишь в случае крайней необходимости. Причина заключается в том, что программы, при составлении которых применялся поведенческий подход, почти всегда трудно читать, сопровождать и переводить с одной версии Пролог на другую.

## 1.2 Основные механизмы дедукции

Существуют два метода логического вывода, называемые *прямой цепочкой* рассуждений и *обратной цепочкой*.

Прямая цепочка рассуждений предполагает использование правил для логического вывода новых фактов, а также фактов, которые имплицитно существовали ранее, но могут быть сделаны явными посредством применения правил. То есть механизм вывода работает для пополнения изначального запаса истинных фактов фактами, которые подразумеваются посредством набора правил. Подобная прямая цепочка рассуждений на практике неприменима: в системе с реальным числом правил имеется столь много подразумеваемых (скрытых) фактов, что, будучи обнаружены, они затмят те несколько фактов, которые действительно представляют интерес и являются полезными.

Однако системы, основанные на прямой цепочке рассуждений, существуют. Любая такая система имеет дополнительный механизм (иногда весьма специфичный в зависимости от проблемы) для того, чтобы определить, с каким следующим правил ей предстоит работать. Вместо простого цикла, который механически выбирает правила, механизм выбора четко устанавливает приоритет выбора тех или иных фактов и правил.

Рассмотрим теперь иной способ получения логического вывода на основе фактов и правил. Начнем с заключения, которое представляет для нас интерес и не является явным истинным фактом. Оно не находится среди хранимых фактов, когда мы запускаем систему.

Механизм вывода просматривает все правила, которые приводят к данному факту как к заключению. Затем механизм просматривает посылки этих правил. Возможно, они уже хранятся среди истинных фактов. Тогда можно считать, что изучаемых факт является истинным и должен быть добавлен в хранилище истинных фактов. Если ни одно из правил не может быть использовано для непосредственного определения рассматриваемого значения из-за того, что необходимо установить истинность их посылок, то в таком случае необходимо идти в обратном направлении и попытаться установить достоверность всех посылок в тех правилах, которые могут применяться для установления истинности конечного вывода. Перемещение на много уровней назад в древовидной структуре даст нам факты, которые являются истинными. Это и есть обратная цепочка рассуждений.

Механизм вывода на Прологе основан на обратной цепочке рассуждений. Процесс выполнения программы сводится к установлению истинности определенного предложения в Прологе (и обычно в определении величин определенных переменных в процессе) посредством обратной цепочки рассуждений и продолжается до тех пор, пока не будут найдены некоторые базовые истинные факты, известные системе.[3]

## 1.3 Исчисление предикатов как язык для решения задач

Для автоматического анализа рассуждений необходим некоторый формальный язык, на котором можно формировать посылки и делать верные выводы. Все, что для этого требуется, - это возможность описать интересующую нас задачу и средства поиска соответствующих шагов в процессе логического вывода.

Исчисление предикатов первого порядка – это такая система в логике, в которой можно выразить большую часть того, что относится к математике, а также многое из разговорного языка. Эта система содержит правила логического вывода, позволяющие делать верные логические построения новых утверждений. Благодаря своей общности и логической силе исчисление предикатов может всерьез претендовать на использование его для машинного построения умозаключений.

Язык, подобный языку в исчислении предикатов, определяется его синтаксисом. Чтобы задать синтаксис, надо задать алфавит символов, которые будут использоваться в этом языке. Один из важных классов выражений в исчислении предикатов – это класс правильно построенных формул.

Мы обычно пользуемся языком для того, чтобы делать утверждения, касающиеся интересующей нас области. Отношения между языком и описываемой им областью определяется семантикой этого языка. Правильно построенные формулы исчисления предикатов как раз являются теми выражениями, которые мы будем использовать в качестве утверждений, касающихся интересующей нас области. Говорят, что правильно построенные формулы принимают значения T или F в зависимости от того, являются эти утверждения в этой области истинными или ложными. Приемы обращения с правильно построенными формулами позволяют строить умозаключения, относящиеся к некоторой области, и, следовательно, могут представить интерес при создании принятия решения, требующего такого умозаключения.[2]

### 1.3.1 Унификация и принцип резольвенции в исчислении

### предикатов

Унификация – процесс, являющийся основным в формальных преобразованиях, выполняемых при нахождении резольвент.

Термы литерала могут быть переменными буквами, константными буквами и выражениями, состоящими из функциональных букв и термов. Подстановочный частный случай литерала получается при подстановке в литералы термов вместо переменных. Например, для литерала частными случаями будут , , , .

Первый частный случай называется алфавитным вариантом исходного литерала, поскольку здесь вместо переменных, входящих в , подставлены лишь частные переменные. Последний из четырех частных случаев называется константным частным случаем, или атомом, так как ни в одном из термов этого литерала нет переменных.

В общем случае любую подстановку можно представить в виде множества упорядоченных пар Пара означает, что повсюду переменная заменяется термом . Существенно, что переменная в каждом ее вхождении заменяется одним и тем же термом. Для получения частных случаев литерала были использованы четыре подстановки

Обозначим через частный случай литерала P, получающийся при использовании подстановки . Например, . Композицией двух подстановок и называется результат применения к термам подстановки с последующим добавлением пар из , содержащие переменные, не входящие в число переменных из . Можно показать, что применение к литералу P последовательно подстановок и дает тот же результат, что и применение подстановки , то есть . Можно также показать, что композиция подстановок ассоциативна: . Если подстановка применяется к каждому элементу множества литералов, то множество соответствующих ей частных случаев обозначается через . Множество литералов называется унифицируемым, если существует такая подстановка , что . В этом случае подстановку называют унификатором , поскольку ее применение сжимает множество до одного элемента. Наиболее общим (или простейшим) унификатором для будет такой унификатор , что если - какой-нибудь унификатор для , дающий , то найдется подстановка , для которой .

Существует алгоритм, называемый алгоритмом унификации, который приводит к наиболее общему унификатору для унифицируемого множества литералов и сообщает о неудаче, если множество неунифицируемо. Алгоритм начинает работу с пустой подстановки и шаг за шагом строит наиболее общий унификатор, если такой существует.

Пусть исходные предложения задаются в виде и и переменные, входящие в , не встречаются в и обратно. Пусть и - такие два подмножества и , что для объединения существует наиболее общий унификатор . Тогда говорят, что два предложения и разрешаются, а новое предложение является их резольвентой. Резольвента представляет выведенное предложение, и процесс образования резольвенты из двух "родительских" предложений называется резольвенцией.

Иными словами мы хотим иметь возможность находить доказательство того, что некоторая правильно построенная формула W в исчислении предикатов логически следует из некоторого множества S правильно построенных формул. Это задача эквивалентна задаче доказательства того, что множество неудовлетворимо. Процессы выявления неудовлетворимости некоторого множества предложений называются процессами опровержения.

Принцип резольвенции непротиворечив и полон. Непротиворечивость означает, что если когда-нибудь мы придем к пустому предложению, то исходное множество обязано быть неудовлетворимым. Полнота означает, что если исходное множество неудовлетворимо, то, в конце концов, мы придем к пустому предложению.[2]

###

### 1.3.2 Методы поиска доказательства в исчислении предикатов

### 1.3.2.1 Исчисление предикатов при решении задач

Иногда достаточно только знать, следует ли логически правильно построенная формула W из некоторого множества S правильно построенных формул. Если W не следует из S, то, возможно, мы захотим знать, следует ~W из S. Конечно, в силу неразрешимости исчисления предикатов не всегда можно установить, следует ли W из S.

В других приложениях нужно знать значение элемента x (если он существует), при котором данная правильно построенная формула W (содержащая x в качестве переменной) логически следует из некоторого множества S правильно построенных формул. Иными словами, мы хотели бы знать, следует ли логически правильно построенная формула , и если да, то каков тот частный случай переменной x. Проблема поиска доказательства правильно построенной формулы , исходя из S, является обычной проблемой доказательства в исчислении предикатов, но для построения удовлетворяющего частного случая требуется, чтобы метод доказательства был "конструктивным".

Часто утверждения, относящиеся к задаче, делаются в форме фраз на разговорном языке, например английском. Поэтому естественно возникает вопрос, в каких случаях можно осуществить автоматический перевод с английского языка на язык исчисления предикатов. Написано несколько программ, позволяющих в ограниченных рамках перевод с естественного языка на язык предикатов, но способность работать с естественным языком пока находится в весьма неудовлетворительном состоянии.[1]

### 1.3.2.2 Стратегии перебора

Непосредственное применение принципа резольвенции соответствует простой процедуре полного перебора при построении опровержения. Такой перебор мы начинается множества S, к которому добавляется резольвенты всех пар предложений в S с тем, чтобы образовать множество R. Затем добавляются резольвенты всех пар предложений в R с тем, чтобы образовать множество R(R(S))=R2(S), и т.д. Этот метод перебора как правило непригоден для практики, так как множества R(S), R2(S),… слишком быстро разрастаются. Практические процедуры доказательства определяются стратегиями перебора, применяемыми для его ускорения. Такие стратегии бывают трех типов: стратегии упрощения, стратегии очищения и стратегии упорядочения.[2]

### 1.3.2.3 Стратегии упрощения

Иногда множество предложений удается упростить, исключив из него некоторые предложения или исключив из предложений определенные литералы. Эти упрощения таковы, что упрощенное множество предложений выполнимо тогда и только тогда, когда выполнимо исходное множество предложений. Таким образом, применение стратегий упрощения позволяет снизить скорость роста новых предложений.

Исключение тавтологий.

Любое предложение, содержащее литерал и его дополнение (такое предложение называется тавтологией), можно отбросить, так как любое невыполнимое множество, содержащее тавтологию, остается невыполнимым и после ее удаления.

Исключение путем означивания предикатов.

Иногда появляется возможность означить (выяснить значение истинности) литералы, и это оказывается удобнее, чем включать соответствующие предложения в S. Такое означивание легко провести для константных частных случаев. Например, если предикатная буква E обозначает отношение равенства, то означивание константных частных случаев типа E(7,3), когда они появляются, провести легко, хотя нам бы не хотелось добавлять к S полную таблицу, содержащих много константных частных случаев литералов E(x,y) и ~E(x,y).

Если какой-нибудь литерал предложения получает значение истинности T, то все предложения можно отбросить, не нарушая при этом свойства невыполнимости оставшегося множества. Если же какой-нибудь литерал при означивании получает значение истинности F, то из этого предложения можно исключить данное вхождение литерала.[2]

Исключение подслучаев.

Предложение называется подслучаем предложения , если существует такая подстановка , что . Например,

 - подслучай предложения ,

 - подслучай предложения ,

 - подслучай предложения

 - подслучай предложения .

Предложение в S, являющиеся подслучаем другого предложения в S, можно исключить из S, не нарушая свойства невыполнимости оставшегося множества. Отбрасывание предложений, являющихся подслучаями других, часто ведет к значительному уменьшению числа резольвенций, необходимых для нахождения доказательства.[2]

### 1.3.2.4 Стратегии очищения

Стратегии очищения основаны на тех теоретических результатах в теории доказательства с помощью резольвенций, в которых утверждается, что для нахождения опровержения не нужны все резольвенции. Иными словами, достаточно выполнить резольвенции только для предложений, удовлетворяющих определенным требованиям. Обозначим через объединение множества S и множества всех резольвент всех пар предложений из S , удовлетворяющих критерию C. Ясно, что .

Про стратегию очищения, использующую критерий C, говорят, что в ней используется "резольвенция по отношению к C". Для применения такой стратегии сначала вычисляется , затем и т.д. до тех пор, пока при некотором n в не окажется пустого предложения обозначемого nil.

Потенциальное достоинство стратегии очищения, в том, что на каждом уровне требуется меньше резольвенций. Однако уровень, на котором появляется пустое предложение, обычно возрастает, так что стратегия очищения приводит обычно к узконаправленному, но более глубокому перебору. Стратегия очищения полезна лишь в том случае, если она уменьшает все затраты усилий на перебор, включая усилия, необходимые для проверки критерия C.[2]

### 1.3.2.5 Формы доказательства с отфильтровыванием

### предшествующих вершин

Доказательство с отфильтровыванием предшествующих вершин производится с использованием AF-графа. Граф опровержения имеет AF-форму, если каждая из его вершин соответствует одному из следующих предложений:

1. базовому предложению;
2. предложению, непосредственно следующему за базовым;
3. предложению, непосредственно следующему за двумя небазовыми предложениями A и B, из которых B предшествует A (отсюда термин отфильтровывание предшествующих вершин).

Граф в виде лозы представляет собой частный случай графа в AF-форме: каждая из его вершин соответствует либо предложению 1, либо предложению 2. Но граф типа лозы существует не для всех неудовлетворимых множеств. Ниже приведен пример такого графа.[2]

Рис 1.1. Вид графа в AF-форме.

###

### 1.3.2.6 Стратегии поддерживающего множества

Стратегией поддерживающего множества называют стратегию, в которой выбирается такое непустое множество K исходного множества предложений S, что множество S-K удовлетворимо. Например, в качестве K можно взять множество предложений, возникающих из отрицания доказываемой теоремы. Говорят, что предложения в K имеют поддержку. При поиске опровержения допустимыми считаются резольвенты лишь тех пар предложений, в которых, по крайней мере, одно имеет поддержку, каждому предложению, построенному в результате резольвенции, также придается поддержка.

Так как множество S-K удовлетворимо, существует граф опровержения, имеющий AF-форму, у которого верхней вершиной служит один из элементов множества K. Таким образом, стратегия поддерживающего множества полна, поскольку она допускает все резольвенции, допускаемые AF-стратегией.[2]

### 1.3.2.7 Стратегии упорядочения

На основе резольвенций, обеспечиваемых различными стратегиями очищения, иногда можно искать опровержение, упорядочив выполняемые резольвенции. В стратегиях упорядочения не запрещаются никакие типы резольвенций, а лишь даются указания на то, какие из них надо выполнять в первую очередь. Стратегии упорядочения соответствуют эвристическим стратегиям перебора для поиска на графах. При хорошем упорядочении не обязательно вычислять все элементы множеств R(S), R2(S) и т.д. Если пустое предложение появляется впервые на уровне n, что хочется думать, можно прямо направить на этот уровень, не заполняя нижние уровни.

Две довольно эффективные стратегии упорядочения – это стратегия предпочтения одночленам и стратегия наименьшего числа компонент. В стратегии предпочтения одночленам делается попытка сначала построить резольвенты между одночленами, т.е. предложениями, содержащими один литерал. Если это удается, то сразу же получается опровержение. Если же не могут найти пару одночленов, у которых есть резольвента, то пытаются найти резольвенту для пар одночлен-двучлен и т.д. Как только какая-нибудь пара предложений разрешается, полученную резольвенту сразу сопоставляют с одночленами с тем, чтобы найти возможные резольвенты. Во избежание совершения невыгодной цепочки одночленных резольвенций обычно устанавливается граничный уровень.

Стратегия предпочтения одночленам оправдана гарантированным укорочением длины предложений, вызываемым одночленными резольвентами. Так как цель построения резольвент состоит в образовании пустого предложения, то стратегия предпочтения одночленам напрашивается сама собой. При введении граничных уровней для возможности использования и других резольвенций такая стратегия не препятствует нахождению опровержения и, как правило, сильно ускоряет процесс перебора.

Стратегия наименьшего числа компонент упорядочивает резольвенции согласно длине получаемых резольвент. Так, два предложения, дающие наиболее короткую резольвенту, разрешаются в первую очередь. Эта стратегия в некотором смысле дороже, поскольку до выполнения резольвенции надо подсчитать длины потенциальных резольвент и упорядочить их.[2]

## 1.4 Анализ характеристик существующих интерпретаторов

В настоящее время существует несколько интерпретаторов и компиляторов языка Пролог.

СиПролог (CProlog). Поставщиком является отдел архитектуры Университета Эдинбурга. Эта версия переносится практически на любой 32-разрядный компьютер с операционной системой UNIX. Синтаксис СиПролога совпадает с синтаксисом DEC-10 Пролога. Встроенного редактора не существует. У отладчика имеются всего четыре команды:

Call - вызов первой фразы предиката

Back To - вызов второй и последующих фраз

Exit - процедура выполнена успешно

Fail - система достигла конца множества фраз.[2]

Квинтус Пролог (Quintus Prolog). Поставляется фирмой Quintus Computer Systems Inc. Он Предназначен для ЭВМ под управлением UNIX и VMS. Квинтус Пролог можно запускать либо как самостоятельный процесс, либо через специальный интерфейс с редактором EMACS. Отладчик аналогичен отладчику СиПролога, но обладает большим количеством команд.[2]

Пролог-2. Поставляется фирмой Expert Systems Int. Работает под управлением MS-DOS. Поддерживает свой механизм виртуальной памяти, что позволяет писать программы, работающие с большим количеством данных. Из-за особого механизма виртуальной памяти программу необходимо разбивать на модули и указывать явно, должен ли находится модуль в реальной памяти или возможно его размещение в виртуальной. Имеется встроенный редактор. Отладчик аналогичен отладчику СиПролога.[2]

Эрити Пролог (Arity Prolog). Поставщик Arity Corp. Работает под управлением MS-DOS или совместимой операционной системы. Имеет механизм виртуальной памяти со страничной организацией. Встроенного редактора нет, отладчик аналогичен предыдущим.[2]

Турбо Пролог (Turbo Prolog). Поставщик Borland Int. Работает под управлением MS-DOS. Является полноценным компилятором, вследствие чего обладает строгим контролем типов. Создан собственный оконный интерфейс с четырьмя окнами: редактор, отладчик, консоль и окно сообщений об ошибках. Сообщения отладчика аналогичны сообщениям отладчика СиПролога.[6]

Visual Prolog. Поставщик Prolog Development Center. Работает под управлением Windows 3.1 и выше. Обладает развитым оконным интерфейсом. Позволяет создавать полноценные приложения для Windows с использованием окон. Сообщения отладчика аналогичны СиПролог.

Основным недостатком всех рассмотренных диалектов языка Пролог является то, что база данных у них должна храниться в оперативной памяти, либо в виртуальной, когда реальные базы данных могут занимать десятки и сотни мегабайт. Вторым недостатком является то что, ни один из диалектов не поддерживает современные базы данных, а подключает базы данных, содержащиеся в текстовых файлах и представляющих собой, по сути, часть Пролог-программы.

## 1.5 Необходимость разработки интерпретатора языка Пролог

Отсутствие возможности работы с универсальными базами данных у вышеперечисленных версий языка Пролог заставляют разработчиков писать программы на универсальных языках программирования. Вследствие этого, увеличивается время разработки программ для баз данных, так как универсальные языки программирования, такие как С, Паскаль и проч., не имеют удобных встроенных средств работы с базами данных.

Также все перечисленные выше версии Пролога имеют недостаточно удобный интерфейс и средства отладки, либо совсем его не имеют, что уменьшает скорость разработки программ на Прологе.

## 1.6 Выбор языка программирования

На выбор языка программирования влияют следующие факторы:

1. характер решаемой задачи;
2. имеющиеся в наличии системные библиотеки;
3. поддреживаемые компилятором платформы.

По характеру решаемой задачи, для программирования интерпретатора требуется язык программирования, позволяющий:

1. гибко работать с динамически выделяемой памятью;
2. иметь объектно-ориентированное расширение;
3. иметь средства обработки исключительных ситуаций;
4. получать высокоскоростной код.

Перечисленным требованиям удовлетворяют С++ и Паскаль. До недавнего времени Паскаль (и его диалект Delphi) значительно уступал С++ по возможности формирования высокоскоростного кода. Но теперь в компилятор Delphi 4 был встроен оптимизатор, который позволяет формировать высокоскоростной код.

Также в пользу Delphi 4 говорит и то, что он теперь может оперировать с динамическими массивами, то есть с такими массивами, количество элементов которых может меняться в процессе выполнения программы.

Borland Delphi 4 генерирует код для операционных систем Windows 95, 98 и NT. Имеет средства визуального построения приложений.

# 2 Конструкторская часть

## 2.1 Синтаксис программ на Прологе в нотации Бэкуса-Наура

Программа::=предложение <предложение>

Предложение::=утверждение, управляющая команда

Утверждение::=голова.проб\_символ

Голова :- хвост.проб\_символ

Голова if хвост.проб\_символ

Управляющая команда::=

целевое утверждение

<,целевое утверждение>.проб\_символ

Голова::=целевое утверждение

Хвост::=целевое утверждение

<,целевое утверждение>

Целевое утверждение::=атом|структура

Проб\_символ::=пробел, возврат каретки[6]

## 2.2 Общая структура интерпретатора

Интерпретатор языка Пролог состоит из следующих частей:

* Предкомпилятор;
* Интерпретатор.

Предкомпилятор выполняет перевод исходных данных в объекты интерпретатора. Исходными данными для предкомпилятора являются:

* Текст программы;
* Типы пользователя;
* Описания внешних данных (структур баз данных);
* Описания предикатов программы.

Интерпретатор на основе выполненных предкомпилятором действий и созданных им объектов выполняет программу с помощью алгоритма бэктрекинга.

### 2.2.1 Принцип работы предкомпилятора

Предкомпилятор состоит из двух основных частей:

* Лексический анализатор
* Синтаксический анализатор.

Лексический анализатор выполняет разбор текста программы на лексемы. В ходе работы лексического анализатора формируется массив лексем, соответствующих программе.

Синтаксический анализатор на основе массива лексем, полученных от лексического анализатора, формирует объект программы.

### 2.2.1.1 Работа лексического анализатора

Для удобства работы лексический анализатор склеивает весь текст программы в одну длинную строку. Такое склеивание можно проводить, так как максимальная длина строки в Delphi 4 равна 2 гигабайтам. При склеивании строк, в конце каждой строки ставится пара символов Enter и пробел. Это делается для того, чтобы удобно можно было вычислить положение лексемы в тексте программы.

Лексический анализатор просматривает текст программы символ за символом, пропуская символы, заключенные между фигурными скобками. В ходе просмотра анализатор выделяет цепочку символов, которая похожа на лексему, после чего передает управление анализатору лексем.

Анализатор лексем, получив строку с предполагаемой лексемой, пытается сначала сопоставить ее со стандартными лексемами (арифметические знаки, точка, запятая и т.п.). Если строка не является стандартной лексемой, то далее анализатор лексем пытается найти ее среди предикатов, функций и баз данных. В случае неудачи анализатор проверяется строку, является ли она правильным идентификатором. Если да, то это переменная. Если лексема начинается и кончается кавычками, то это строка. На заключительном этапе проверяется, может ли лексема быть числом.

Если ни одно из условий не было выполнено, то выдается сообщение об ошибке.

На выходе анализатора лексем формируется объект лексемы, в котором хранится тип лексемы, ее строковый вид, а также положение лексемы в тексте программы.

Потом из полученных лексем создается массив.

### 2.2.1.2 Синтаксический анализатор

На начальном этапе массив лексем разбивается на несколько подмассивов по одному для каждого предиката. При разбиении массива проверяется следующее правило: каждое предложение начинается с имени предиката и заканчивается точкой. Таким образом, выделяются лексемы соответствующие каждому предикату.

Затем работа продолжается с каждым из подмассивов отдельно. Пользуясь вышеописанным правилом, выделяется отдельное предложение и отправляется на синтаксический анализ.

Предложение в Прологе имеет следующий формат:

ИмяПредиката (Параметр1, Параметр2, …) if

Условие1(Параметр11, Параметр12, …),

Условие2(Пераметр21, Параметр22, …),

УсловиеN(ПараметрN1, ПараметрN2, …).

При синтаксическом анализе, во-первых, проверяется заголовок предложения. Проверяется имя предиката и параметры (их количество и тип) и наличие слова “if”. Если в качестве параметра стоит переменная, то считается, что переменная может быть любого типа, а константы подвергаются жесткому контролю.

Из массива лексем предложения выделяются отдельные условия. В этом случае должны быть выполнены следующие требования:

* Все условия разделены запятыми друг от друга;
* Цепочка условий заканчивается точкой;
* Внутри условия все скобки (круглые и квадратные) должны быть закрыты.

Проверка условий делится на три части в зависимости от типа первой лексемы:

* Вызов предиката, если первая лексема - имя предиката;
* Вызов базы данных, если первая лексема - имя базы данных;
* Вычисление арифметического выражения - во всех остальных случаях.

При синтаксическом анализе вызовов предикатов и баз данных выполняется разбор параметров примерно такой же, как при анализе заголовка предложения.

При анализе арифметического выражения строится дерево, соответствующее выражению.

### 2.2.1.3 Анализ арифметического выражения

Если на вход поступает массив из одного элемента, то немедленно формируется лист арифметического дерева, и программа выходит из функции.

Среди всего массива лексем находится оператор с наинизшим приоритетом, причем операции, которые находятся внутри скобок в рассмотрение не берутся. В Прологе приоритеты операций распределены следующим образом:

1. \*,/

2. +,-

3. >,<.>=,<=,<>

4. and,or

5. =

Если операция не найдена и первая и последняя лексема – парные круглые скобки, то необходимо их снять и вызвать опять функцию построения арифметического дерева. Возможен другой вариант при отсутствии найденной операции: первая лексема – функция, вторая – открывающая круглая скобка, последняя – закрывающая круглая скобка. В этом случае необходимо запустить процедуру нахождения параметров функции.

После того, как нашли нужную операцию, массив делится на две части – левую и правую. Если левый или правый массив пустой, то сообщить об ошибке.

В ходе выполнения следующих действий мы из арифметического выражения получаем дерево.

Например: A=5+func(6+C,D,E)/E-4

Рис 2.1. Дерево арифметического выражения.

### 2.2.1.4 Анализ параметров предикатов

Параметры на анализатор параметров поступают в скобках. Начиная с первой значимой лексемы, ищется полная запись параметра с таким условием, что запись параметра должна заканчиваться запятой или закрывающейся скобкой, и внутри параметра все круглые и квадратные скобки должны быть закрыты. Таким образом, формируется массив лексем параметра.

По первой лексеме массива можно определить, что это за параметр:

Если это название структуры – то структура,

Если левая квадратная скобка – то список,

Если число или строка – то константа,

Если идентификатор – то переменная.

Если выяснили, что параметром является список или структура, то отправляемся на специальные функции анализа списков или структур, где выделяются отдельные элементы списка или структуры и выясняется их тип.

###

### 2.2.1.5 Проверка типов параметров

На вход поступает объект с параметром и имя типа, с которым сравнивается параметр. На выходе мы должны выдать логическое значение, говорящее может ли параметр хотя бы теоретически относиться к сравниваемому типу.

Если параметром является переменная, то считается, что она может быть любого типа. Числовые, строковые и логические константы могут быть опознаны сразу.

Сложнее дело обстоит со структурами и списками, а также анализом составных типов.

При анализе составного типа необходимо выяснить, относится ли параметр к одному из типов составного типа. Если да, значит необходимо возвратить истину.

Рассматривая структуру, мы должны проверить тип каждого из элементов, составляющих структуру. Если все элементы имеют правильные типы, то возвратить истину.

Список может быть записан двумя способами:

1. [Элемент1, Элемент2, … , ЭлементN]
2. [Голова|Хвост]

В первом случае мы должны проверить тип каждого из элементов.

При рассмотрении второго случая необходимо учитывать то, что Голова имеет тип элемента списка, а Хвост – тип списка.

## 2.3 Работа интерпретатора

Функция работы интерпретатора представляет собой рекурсивную функцию, выполняющую алгоритм бэктрекинга.

Алгоритм бэктрекинга заключается в следующем. Для первого оператора Пролог-программы интерпретатор находит решение, удовлетворяющее этому оператору. Если решение было найдено, то переходим к следующему оператору. На втором операторе, с учетом результатов на предыдущем шаге, программа пытается решение для второго оператора. Если решение было найдено, то программа идет дальше. В противном случае, программа должна вернуться на шаг назад и подобрать другое решение для первого условия, а затем опять попытаться выполнить второе условие. Такой процесс идет до тех пор, пока не будет выполнено последнее условие, и предложение будет объявлено истинным. Или, если программа не сможет больше подобрать решения для первого условия, то все предложение будет объявлено ложным.

Принцип действия интерпретатора основан на рекурсивном вызове функции TPrologProgram.ExecutePredicate, которая выполняет предикат. На вход функции поступает объект TStackNode, в котором содержатся входные параметры, а также номер предложения, с которого необходимо начинать выполнять предикат. Функция ExecutePredicate возвращает логическое значение, указывающее на то, было ли найдено решение для предиката или нет. Входные и выходные параметры предиката хранятся в поле InputParameters.

Последовательность действий, которые выполняет функция ExecutePredicate, выглядит следующим образом:

1. В каждое предложение программа пытается подставить входные параметры. Если подстановка прошла успешно (это определяется функцией FindNamedAreas), то интерпретатор пытается выполнить это предложение. В противном случае просмотр продолжается.
2. Необходимо найти решение для каждого из условий предложения. Интерпретатор проходит по каждому из условий предложения последовательно.
3. Перед выполнением условия проверяется, запускается на оно на прямом пути или на обратном. Если на прямом пути, то в дополнительный стек заносится еще один элемент TSubStackNode, в котором содержатся следующие данные: само условие, список имен созданных на данном шаге переменных и список имен переменных свободных до текущего шага. Если условие запускается на обратном пути, то объект TSubStackNode не создается, так как был создан ранее.
4. Если текущее условие предикат или база данных, то для них необходимо создать новый объект TStackNode и сформировать пакет входных параметров. Затем, если текущее условие база данных, то вызывается функция обработки баз данных ExecuteExtDataPredicate, если условие стандартный предикат, то - ExecuteStandardPredicate, и, если это предикат пользователя то рекурсивно вызывается ExecutePredicate.
 Если текущее условие - выражение, то выполняется функция ExecuteArithmeticTerm.
5. Если после своего выполнения условие вернуло значение False, то запускается механизм обратного прохода. Уничтожается последний элемент TSubStackNode, и программа возвращается к предыдущему условию и пытается найти новое решение для него.
6. Если все условия были выполнены, то формируются выходные параметры и функция возвращает истину. В противном случае, программа возвращается к пункту 1 и пытается найти еще одно предложение, соответствующее входным данным.
7. Если были исчерпаны все предложения и ни для одно не было найдено решения, то необходимо возвратить False.

### 2.3.1 Выполнение обращений к базам данных

Обращение к базе данных происходит с помощью SQL-запросов.

При первом обращении к базе данных создается объект SQL-запроса и сам запрос.

Формат SQL-запроса

**SELECT** <поле1>,…,<полеN>

**FROM** <имя базы данных>

**WHERE**

<поле1>=<значение1> **and**

<поле2>=<значение2> **and**

…

<полеN>=<значениеN>

В запросе используются только условия-равенства, так как в Прологе при сопоставлении значении на входе в предикат используется только сравнение.

В список условий помещаются только те поля, значения которых на входе определены.

При первом обращении к базе берется первая запись из запроса.

При последующих вызовах базы данных (в случае обратного пути в бэктрекинге) объект SQL-запроса не создается, а используется ранее созданный, и применяется функция Next для доступа к следующей записи.

Таким образом, доступ к базе данных осуществляется как обычным предикатам, которые состоят только из предложений-фактов.

### 2.3.2 Вычисление арифметических выражений

На основе построенного при компиляции дерева арифметического выражения достаточно несложно, используя рекурсию, вычислить выражение.

На вход функции поступает объект с деревом арифметического выражения и объект со списком переменных, созданных ранее программой.

Программа смотрит, если в корне дерева находится арифметическая операция, то рекурсивно вызывается функция вычисления выражений для левой и правой частей выражения, которые в свою очередь являются деревьями арифметического выражения.

Дерево может состоять из одного элемента, если это константа или переменная. На этом шаге рекурсия заканчивается и возвращается значение константы или переменной (если до этого в списке переменных ее не значилось, то создается новая переменная).

## 2.4 Объекты, используемые компилятором и интерпретатором

### 2.4.1 Объекты переменных TPrologVariable, TPrologVariables,

### TPrologVariableList, TPrologVariableStruct

Для преставления переменных всех типов (целые, дробные, строки, логические, списки, структуры) служит класс TPrologVariable. Для определения типа переменной служит поле iType. В поле Data хранится указатель на данные, соответствующие типу. Для простых типов (целые, дробные, строки, логические) – это указатель на соответствующий тип. Конструктор создает переменную без типа. Деструктор автоматически освобождает память для переменной любого типа. Для этого класса опеределены следующие методы:

1. procedure CreateVariable(DomainType:string; vName:string) - создание переменной типа DomainType с именем vName;
2. procedure DestroyVariable - уничтожение переменной (эта процедура не является деструктором, так как не уничтожает сам объект, а только освобождает память)
3. function CreateCopy:TPrologVariable - создание точной копии переменной;
4. procedure AssignVariable (v:TPrologVariable) - присваивание значения переменной.

Для представления списков введен класс TPrologVariableList, в котором хранится имя типа списка, имя типа элемента списка, тип представления списка (в виде перечисления элементов или «голова-хвост»), а также динамический массив, содержащий объекты TPrologVariable, являющиеся элементами списка.

В структурах (TPrologVariableStruct) аналогично хранится имя типа структуры, массив с типами элементов структуры и динамический массив с элементами структуры.

Следует отметить, что в классе TPrologVariable, могут храниться как переменные, так и константы. У константы поле Name - пустая строка.

Класс TPrologVariables позволяет организовать к пакету переменных:

1. procedure AddVariable(v:TPrologVariable) - добавление переменной в список переменных;
2. function AddNewVariable:TPrologVariable - создание и добавление новой переменной в список;
3. procedure DeleteVariable(n:integer) procedure DeleteVariable(vName:string) procedure DeleteLastVariable - удаление переменной по ее индексу или по имени, либо последнюю переменную в списке;
4. function VariableByName(vName:string):TPrologVariable - доступ к конкретной переменной по ее имени;
5. property Variables[ind:integer]:TPrologVariable - свойство, оргназиующее доступ к переменным как к массиву переменных.

### 2.5.2 Стандартные функции и предикаты

Все стандартные функции и предикаты имеют одинаковые параметры.

Объявление функции выглядит следующим образом:

TStdFunction=function (Args:TPrologVariables):TPrologVariable;

Args – пакет с входными параметрами.

Функция возвращает значение в виде переменной TPrologVariable.

Объявление функции, выполняющая работу предиката:

TStdPredicate=function (VarPacket:TPrologVariables; BackTracking:Boolean):Boolean;

VarPacket – пакет с параметрами к предикату

BackTracking – логическая переменная, которая установлена в True, если предикат был вызван на обратном пути алгоритма бэктрекинга.

Функция возвращает логическое значение, которое должен возвратить предикат после своего выполнения.

## 2.6 Представление Пролог-программы в виде объектов

Для хранения откомпилированной Пролог-программы используются три класса TPrologTerm, TPredicateClause и TProgramPredicate.

TPrologTerm служит для хранения одного условия внутри предложения. В нем хранится тип условия (выражение, предикат, база данных), позиция условия в тексте Пролог-программы, а также указатель (Data) на объект представляющий данное условие (TSubTermPredicate, TSubTermExtData, TSubTermExpression).

TSubTermPredicate служит для хранения условий-предикатов. В данном классе есть следующие поля:

1. Name :String - имя предиката;
2. StandardPredicate :Boolean - True - если предикат стандартный;
3. Params :TVariablesArray - параметры для вызова предиката;
4. NoInverse :Boolean - признак наличия или отсутствия инверсии перед вызовом предиката.

TSubTermExtData аналогичен классу TSubTermPredicate за исключением того, что в нем нет поля StandardPredicate.

TSubTermExtData предназначен для представления арифметических выражений. Он представляет собой рекурсивный объект с полями LeftHand и RightHand для представления выражений левой и правой частей выражения. Данный класс содержит следующие поля:

1. Operation :TExpressionOperation - тип операции;
2. FuncName :String - имя функции, если Operation=eoFunction;
3. Operand :TPrologVariable - переменная или константа, если Operation=eoVariable;
4. LeftHand :TSubTermExpression - левая часть выражения;
5. RightHand :TSubTermExpression - правая часть выражения
6. FuncParams :array of TSubTermExpression - параметры функции, если Operation=eoFunction.

TPredicateClause хранит в себе одно предложение. Параметры предложения хранятся в массиве Params, а условия - в Terms.

TProgramPredicate хранит полную информацию о предикате. Объект этого класса используется как на этапе компиляции, так и на этапе интерпретации. Он хранит имя предиката (Name), типы параметров, текст предиката, все лексемы предиката, а также массив с предложениями предиката.

Класс TPrologProgram хранит всю информацию о Пролог-программе:

* Описания типов (Domains);
* Описания баз данных (ExtData);
* Предикаты (TProgramPredicate);
* Массив со стеком (Stack);
* Массив с контрольными точками (BreakPoints);
* Имя запускаемого предиката (StartPredicate);
* Флаг работы программы (OnTheRun).

## 2.7 Основные модули

Модуль CompileUnit. В данном модуле определяется класс пролог-программы - TPrologProgram, а также два класса, использующихся при запуске интерпретатора: TStackNode и TSubStackNode. Модуль CompileUnit является ядром интерпретатора.

Модуль ProgramClasses. В данном модуле описываются основные классы, использующиеся в объектах TPrologProgram, TStackNode и TSubStackNode как контейнерные:

* TLexemRecord - представляет одну лексему, выделенную из текста программы;
* TPrologVariable - представляет переменную во время выполнения пролог-программы.
* TPrologVariableList - контейнерный класс, использующийся в TPrologVariable, для представления списков;
* TPrologVariableStruct - контейнерный класс, использующийся в классе TPrologVariable, для представления структур;
* TPrologVariables - класс, представляющий собой список переменных TPrologVariable с возможностью доступа к переменной по имени;
* TSubTermPredicate - класс использующийся в TPrologTerm для представления обычного предикатного условия в предложении;
* TSubTermExtData - класс использующийся в TPrologTerm для представления предикатного условия вызова базы данных в предложении;
* TSubTermExpression - класс использующийся в TPrologTerm для представления арифметических выражений в предложении;
* TPrologTerm - класс использует TSubTermPredicate, TSubTermExtData, TSubTermExpression как контейнерные для представления одного условия в предложении;
* TPredicateClause - использует массив объекстов TPrologTerm и TPrologVariable для представления одного предложения в программе;
* TProgramDomain - класс использующийся в TPrologProgram как контейнерный для описания типов;
* TProgramExtData - класс использующийся в TPrologProgram как контейнерный для описания баз данных;
* TProgramPredicate - класс использующийся в TPrologProgram как контейнерный для описания предикатов;
* TPrologFile - класс использующийся в TPrologProgram как контейнерный для описания файлов открытых во время выполнения программы;

Модуль PrologRunTime описывает класс ошибок TRunTimeError, а также все функции и встроенные предикаты Пролога.

MainFormUnit - модуль описания главной формы TMainForm;

ProgFormUnit - модуль описания окна инспектора TProgForm;

DomConstrFormUnit - модуль описания конструктора типов TExtDataForm;

ExtDataFormUnit - модуль описания конструктора баз данных TExtDataForm;

PredicateConstrFormUnit - модуль описания конструктора предикатов TPredicateConstrForm;

EditorFormUnit - модуль описания окна редактора TEditorForm

ConsoleUnit - модуль описания окна консоли TConsole;

ErrorsFormUnit - модуль описания окна ошибок компиляции TErrorsForm;

StackFormUnit - модуль описания окна стека программы TStackForm;

RunTimeDebugFormUnit - модуль описания окна отладки программы TRunTimeDebugForm;

BreakPointsFormUnit - модуль описания окна контрольных точек TBreakPointsForm;

ProjectOptionsFormUnit - модуль описания окна опций проекта.

## 2.8 Демонстрационная программа по выбору конфигурации компьютера

На вход программы с клавиатуры поступают задачи, которые должен будет решать выбираемый компьютер. По полученным данным программа выбирает оптимальную конфигурацию для заданного набора задач (тип процессора, частоту процессора, объем оперативной памяти, размер видеопамяти, минимальный размер винчестера, диагональ монитора, а также список дополнительный устройств, таких как звуковые карты, CD-ROM приводы и прочее).

В программе используются две базы данных.

База данных базовых конфигураций ("Configs.db"), в которой содержатся оптимальные конфигурации для каждой из элементарных задач.

Таблица 2.1.

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип поля | Комментарий |
| Task | String[33] | Название задачи |
| Frequency | Integer[4] | Частота процессора |
| Processor | String[17] | Тип процессора |
| Memory | Integer[4] | Размер ОЗУ |
| VideoMemory | Integer[4] | Размер видеопамяти |
| HDD | Integer[4] | Размер винчестера |
| MonitorSize | Integer[4] | Диагональ монитора |
| Addition1Name | String[17] | Название 1-го дополнительного устройства |
| Addition1Value | String[9] | Характеристика 1-го дополнительного устройства |
| Addition2Name | String[17] | Название 2-го дополнительного устройства |
| Addition2Value | String[9] | Характеристика 2-го дополнительного устройства |
| Addition3Name | String[17] | Название 3-го дополнительного устройства |
| Addition3Value | String[9] | Характеристика 3-го дополнительного устройства |
| Addition4Name | String[17] | Название 4-го дополнительного устройства |
| Addition4Value | String[9] | Характеристика 4-го дополнительного устройства |

База приоритетов устройств ("DeviceClass.db"). Используется для разделения устройств по их характеристикам, в том числе и процессоров.

Таблица 2.2.

|  |  |  |
| --- | --- | --- |
| Имя поля | Тип поля | Комментарий |
| TypeName | String[17] | Имя класса устройств |
| SubType | String[33] | Имя отдельного устройства |
| SubTypeIndex | Real | Характеристика устройства |

Внутреннее представление конфигурации в программе выглядит в виде списка, в котором перечисляются: частота процессора, тип процессора, объем памяти, объем видеопамяти, размер винчестера, диагональ монитора, а также в двух списках одинаковой длины перечисляются виды и типы дополнительных устройств.

Пара предикатов ReadParameters и \_ReadParameters предназначены для ввода списка элементарных задач с клавиатуры в список строк StringList.

Предикат SelectProcessor выбирает из двух поступающих процессоров лучший. Если процессоры одного класса, то сравниваются частоты процессоров. Если процессоры разных классов, то выбирается процессор с более высоким классом, даже, если частота у него и ниже.

AddNewAddition добавляет в списки дополнительных устройств новое устройство. Сначала предикат пытается найти устройство среди уже известных. Если оно не найдено, то добавляется новое устройство. Если среди уже известных устройств имеется устройство аналогичного типа, то из двух устройств выбирается устройство с наивысшим классом по базе DeviceClass.

Предикат ChooseConfig проходит по всему списку элементарных задач, поступившему с клавиатуры, постепенно выбирая лучшую конфигурацию. Объемы оперативной памяти, видеопамяти и диагональ монитора выбираются по максимальному значению. Размер винчестера получается сложением размеров винчестеров для каждой элементарной задачи.

Текст демонстрационной программы приведен в приложении.

# 3 Технологическая часть

## 3.1 Требования по эксплуатации интерпретатора языка Пролог

Программа интерпретатора языка Пролог предназначена для эксплуатации на персональных вычислительных машинах на базе процессоров семейства Pentium и выше в минимальной конфигурации с 16 Мб оперативной памяти. Для установки программы необходимо:

* в минимальной конфигурации - 1.5 Мб дискового пространства;
* в полной конфигурации (с установкой Borland Database Engine) - 10 Мб дискового пространства.

В качестве операционной системы может использоваться Microsoft Windows 95, Windows 98, Windows NT.

## 3.2 Установка системы

Система поставляется на четырех дискетах размером 3.5" и объемом 1.44 Мб.

Установка программы производится с дискет, путем запуска из операционной системы Windows файла "Setup.exe", находящегося на первой дискете. Далее необходимо действовать согласно указаниям.

При установке программы необходимо указать директорию, в которую будет производиться инсталляция.

В процессе установки программа спросит, устанавливать ли Borland Database Engine на машину или нет. Если BDE уже установлена на компьютер ранее, то повторная установка не требуется.

По завершении процесса установки программа-инталлятор создаст в системном меню раздел с названием "Prolog", а внутри него ярлык на файл "Prolog.exe" с названием "Prolog with databases", ярлык на help-файл и на файл "readme.txt".

## 3.3 Руководство пользователя программы интерпретатора языка

## Пролог

### 3.3.1 Запуск программы

Запуск программы можно произвести несколькими способами.

Нажать кнопку "Пуск", выбрать в меню пункт "Программы", выбрать пункт "Prolog". После того, как раскроется подменю, нажать на строку "Prolog with databases".

Дважды щелкнуть по иконке Пролога на рабочем столе

Воспользоваться "проводником" или другим менеджером файлов.

### 3.3.2 Перечень функций, реализуемых системой

Программа интерпретатора языка Пролог выполняет следующие функции:

1. ввод, сохранение и редактирование программы на языке Пролог.
2. осуществление отладочного режима работы с просмотром вызовов предикатов и значений переменных.
3. запуск программы на исполнение
4. возможность приостановки работающей программы на контрольной точке, установленной в тексте программы, или при нажатии кнопки "Пауза".

### 3.3.3 Редактирование Пролог-программы

После запуска программы вся работа по написанию программы может производиться в интегрированной среде разработчика.

Интегрированная среда разработчика предоставляет разработчику следующие возможности:

1. создание нового файла программы;
2. открытие существующего файла программы;
3. сохранение файла;
4. сохранение файла под новым именем;
5. выход из системы;
6. редактирование файла с использованием буфера обмена;
7. настройку параметров программы Пролог;
8. настройку редактора;
9. запуск программы на исполнение;
10. временную остановку работы программы;
11. доступ к файлу справки.

Все эти возможности доступны через пункты меню или через кнопки быстрого доступа.

После запуска программы появляется главное окно, на котором размещены главное меню и основные панели с кнопками быстрого доступа.

Рис 3.1. Вид главного окна.

Программу необходимо в двух окнах: в окне редактора и в окне инспектора.

Инспектор содержит в себе основную информацию о программе: описания типов данных, таблиц баз данных и предикатов.

Рис 3.2. Вид окна инспектора.

Окно инспектора разбито на три части: "Типы", "Внешние данные" и "Предикаты". В случае надобности разделы "Типы" и "Внешние данные" можно временно свернуть, нажав на верхнюю кнопку соответствующей панели.

Для добавления элемента в список типов необходимо нажать на кнопку с изображением плюса. Тогда справа от кнопки появится меню для выбора типа: "Список", "Структура", "Составной тип". Для каждого типа открывается окно конструктора типов, в котором можно редактировать составные элементы типов. При нажатии на кнопку "Ок" тип будет занесен в список типов. Для внесения нового элемента в тип необходимо выбрать его из списка, ввести комментарий (если необходимо) и нажать на кнопку плюс на нижней панели

а) б)

Рис. 3.3. Вид окна конструктора типов: а) конструктор списков;

б) конструктор структур

При добавлении описания базы данных появляется окно для ввода структуры базы данных. В этом окне необходимо ввести имя базы данных в поле "Имя". Это имя будет использоваться в программе. Также необходимо ввести, либо выбрать из списка имя файла базы данных. Имя базы данных, которое будет использоваться в программе, и имя файла могут отличаться. Для внесения поля в базу данных необходимо выбрать тип поля из списка, ввести его длину, а также его имя в базе данных и нажать на кнопку плюс на нижней панели. Если база данных уже существует, то ее структуру можно ввести, нажав на кнопку "Поля таблицы" на нижней панели (вторая справа). Тогда появится окно, в котором будет список из полей базы данных. Можно будет выбрать те поля, с которыми будет работать программа. При нажатии на кнопку "Ок" выбранные поля будут внесены в список.

Рис. 3.4. Вид окна конструктора баз данных

Ввод предиката производится аналогично вводу нового типа.

Для редактирования элемента любого из списков можно дважды щелкнуть по нужному элементу. Для удаления элемента из списка нужно щелкнуть по нему, а затем нажать на кнопку минус.

При однократном нажатии на элемент с именем предиката текстовый курсор в окне редактора автоматически перейдет к тому месту в программе, где описывается данный предикат.

Текст программы вводится в окне редактора. Это окно состоит из панели инструментов и редактора. На панели инструментов находится кнопки для копирования, вырезания и вставки фрагмента текста, а также кнопка по установки контрольной точки в тексте. Из окна редактора можно мгновенно получить справку о любом стандартном предикате или функции, набрав имя предиката или функции в тексте и нажав кнопку F1.

Рис. 3.5. Вид окна редактора.

В процессе написания программы используются стандартные средства редактора операционной системы Windows. Окно редактора поддерживает стандартные операции с буфером обмена Windows, выполняемые по нажатию горячих клавиш, а также выделение областей текста. Поддерживаются следующие горячие клавиши:

Shift - стрелки вверх, вниз, вправо, влево, Home, End - выделение области текста для операций;

Ctrl-Insert или Ctrl-C - копировать выделенную область в буфер обмена;

Shift-Insert или Ctrl-V - вставить фрагмент из буфера обмена;

Shift-Del или Ctrl-X - вырезать фрагмент в буфер обмена.

Перед запуском программы в окне опций проекта необходимо установить имя базы алиаса (если используются базы данных) и имя запускаемого предиката. Если не задано имя запускаемого предиката, программа не запустится.

Рис. 3.6. Вид окна опций проекта.

**3.3.4 Запуск программы на Прологе и ее отладка**

Для запуска программы на главном окне надо нажать на кнопку "Запуск" или в меню "Проект" нажать на пункт с таким же названием. После нажатия на кнопку происходит компиляция программы. В статусной строке в главном окне появляется надпись "Компиляция". Если во время были найдены ошибки, то на экран появляется окно со списком найденных ошибок. При двойном щелчке по строке с названием ошибки текстовый курсор в окне редактора перейдет к месту в тексте, где была найдена ошибка.

Если процесс компиляции прошел нормально, то программа запускается на исполнение. Процесс выполнения программы может быть прерван нажатием на кнопки "Стоп" или "Пауза" в главном окне, а также, если программа приступила к обработке оператора, на котором установлена контрольная точка. При временной остановке на экран появляются два окна: окно трассировки и окно стека.

В окно "Трассировка" выводится протокол работы программы, то каждый вызов предиката, базы данных или арифметического выражения, а также возвращаемые ими значения.

Рис. 3.7. Вид окна трассировки.

В окне "Стек" можно просмотреть стек программы. Стек представлен в виде древовидной структуры. В корне дерева находятся вызовы предикатов. Развернув корневой элемент дерева можно увидеть параметры, с которыми был вызван предикат и переменные, которые были созданы в процессе выполнения предиката.

Общий вид корневого элемента дерева выглядит следующим образом: "Имя предиката *n1* : *n2*", где *n1* - номер предложения, *n2* - номер условия в предложении.

Рис.3.8. Вид окна стека.

### 3.3.5 Работа с меню

Для удобства пользователя при выборе команд в программу введена система меню, которая содержит следующие пункты:

* **Файл**
* **Окна**
* **Проект**
* **Настройка**
* **Помощь**

Меню **Файл** содержит следующие пункты.

**Создать** - позволяет создать новую программу на Прологе.

**Открыть** - позволяет открыть имеющийся файл и выводит стандартное окно Windows для открытия файла:

Рис.3.9. Вид окна для открытия файла.

При открытии файла в инспектор загружается информация о типах, базах данных и предикатах, а в редактор - текст программы. Если формат файла был нарушен, то он не будет загружен.

**Вспомнить** - за этим пунктом скрывается подменю, состоящее из имен последних восьми файлов, которые открывались Пролог-системой. Нажав на название нужного файла в этом подменю можно быстро загрузить его.

**Сохранить** - позволяет сохранить файл на диск. Если файл создавался с помощью пункта меню **Создать**, то выводится диалоговое окно, позволяющее присвоить имя сохраняемому файлу.

**Сохранить как** - позволяет присвоить новое имя файлу. При выборе этого пункта меню будет выведено на экран диалоговое окно для ввода имени файла.

**Печать** - формирует печатную форму из данных, содержащихся в инспекторе, окне редактора и окне опций проекта. После окончания процесса формирования печатной формы на экране появится окно, в котором можно будет просмотреть вид печатной формы. В этом окне можно выделить область текста и скопировать в буфер обмена. При нажатии в окне печати на кнопку "Печать" будет начат вывод на принтер. Общий вид печатной формы можно представить следующим образом:

*<Имя печатаемого файла>*

**DOMAINS**

*<Объявления типов>*

**ALIAS**

*<Имя алиаса в Borland Database Engine, которое использует данная программа>*

**DATABASES**

*<Описание структур баз данных>*

**PREDICATES**

*<Описание параметров предикатов>*

**GOAL**

*<Имя запускаемого предиката>*

**CLAUSES**

*<Текст программы>*

**Выход** - позволяет завершить работу программы интерпретатора Пролог.

Меню **Окна** позволяет открыть следующие окна:

**Инспектор;**

**Консоль;**

**Редактор;**

**Контрольные точки;**

**Стек;**

**Трассировка.**

Меню **Проект** содержит следующие пункты:

**Опции** - выводит на экран диалоговое окна с опциями проекта, в котором можно указать имя алиаса, которым будет пользоваться программа написанная на Прологе, а также имя запускаемого предиката.

**Запуск** - позволяет запустить Пролог-программу на исполнение.

Меню **Настройка** содержит пункт **Параметры редактора**. При выборе этого пункта появляется диалоговое окно, в котором можно выбрать тип и размер шрифта для окна редактора.

Меню **Помощь** содержит следующие пункты:

**Предметный указатель** - загружает help-файл и показывает главную страницу.

**О программе** - выводит окно с информацией о программе.

## 3.4 Описание процесса выполнения программы, написанной на

## языке Пролог

Выполнение программы, написанной на языке Пролог, ведется с использованием алгоритма бэктрекинга.

Предложение просматривает все свои условия последовательно и пытается найти решение для каждого из них. Найдя решение для первого условия, интерпретатор переходит к поиску решения для второго условия с учетом значений переменных, которые были изменены первыми условием. Если решение было найдено, то программа переходит к следующему условию. В противном случае интерпретатор должен вернуться к предыдущему условию и попытаться найти для него другое решение отличное от ранее найденного, а затем опять попытаться выполнить второе условие. Перед началом обратного пути уничтожаются все переменные, которые были созданы перед этим шагом и очищаются те переменные, которые были свободны. Такой процесс происходит до тех пор, пока не будет найдено общее решение верное для всех условий предложения. В этом случае предложение объявляется истинным и выполнение его прекращается. Если такого общего решения найдено не было, то предложение объявляется ложным.

Арифметические выражения и стандартные предикаты процесс бэктрекинга игнорируют и выполняются, так как выполняются однозначно.

**3.5 Общие сведенья об интерпретаторе**

Данная версия языка Пролог совместима по синтаксису с языком Turbo Prolog с некоторыми отличиями.

Поддерживаются следующие простые типы данных:

*Integer* - целочисленный тип;

*Real* - число с плавающей запятой;

*String* - строка;

*Boolean* - логический тип (имеет два значение *True* - истина и *False* - ложь).

Составные типы:

*Список* - используется для представления массивов данных. Количество элементов в списке не ограничено.

*Структура* - используется для представления массивов неоднородных данных. Количество полей в структуре не ограничено.

*Составной тип* - используется для совмещения нескольких типов под одним именем.

Комментарии внутри программы записываются внутри фигурных скобок.

## 3.6 Особенности работы Пролог-программы с базами данных

Данная версия Пролога работает со следующими видами баз данных:

Paradox 7 (или предыдущие версии);

DBase IV и DBase for Windows;

Fox Pro;

MS Access.

Для использования других видов баз данных (например, InterBase или Oracle) необходима установка драйверов этих баз данных.

Работа с базами данных производится под управлением Borland Database Engine, которая должны быть установлена на компьютер либо при установке Пролога, либо с какой-либо другой программой.

При выполнении программы база данных идентифицируется с помощью своего псевдонима, которое при вводе структуры базы данных средствами инспектора заносится в поле "Имя". Имя файла базы данных в тексте программы не используется, хотя псевдоним и имя файла могут совпадать.

Обращение к базе данных происходит так же, как к обычным предикатам, состоящим только из предложений-фактов.

При запуске Пролог-программы интерпретатор определяет, существует ли файл базы данных на диске. Если файл базы данных существует, то он открывается и проверяется соответствие структуры файла базы данных и описания этой базы данных в программе. При описании базы данных в программе некоторые поля можно не использовать. Если файл базы данных отсутствует на диске, то он создается по описанию структуры. При завершении выполнения Пролог-программы все базы данных автоматически закрываются.

## 3.7 Описание стандартных предикатов

Все стандартные предикаты выполняются только прямом пути алгоритма бэктрекинга. На обратном пути они ничего не делают.

**Предикаты ввода с консоли.**

**ReadString**(<свободная переменная>) - читает с консоли строку. В качестве параметра обязательно должна передаваться свободная переменная. В противном случае будет выдано сообщение об ошибке и программа закончит свою работу.

**ReadInteger**(<свободная переменная>) - читает с консоли целое число.

**ReadReal**(<свободная переменная>) - читает с консоли реальное число.

**Предикаты вывода на консоль.**

**Write** (<переменная или константа любого типа>, ...) - печать на консоль содержание переменной. Количество аргументов у предиката произвольно. Количество аргументов у предиката произвольно. Стиль вывода на печать устанавливается в опциях проекта. Например, в процессе отладки можно включить печать имен переменных вместе с их содержимым или кавычки у строковых переменных.

**WriteLn** (<переменная или константа любого типа>, ...) - печать на консоль содержание переменной или константы и перевод курсора на следующую строку. Количество аргументов у предиката произвольно.

**Nl** - перевод курсора консоли на следующую строку.

**Предикаты определения типов переменных.**

**IsInteger**(<переменная>) - предикат является истинным, если поступающая на вход переменная - целочисленная.

**IsReal**(<переменная>) - проверяет, является ли переменная реальным числом.

**IsNumeric**(<переменная>) - проверяет, является ли переменная числовой, то есть либо реальной, либо целой. Использование функции IsNumeric предпочтительнее, чем IsInteger и IsReal, так как в ходе некоторых арифметических операций над целыми числами результат получается дробным (например, при делении).

**IsString**(<переменная>) - проверяет, является ли переменная строкой.

**IsBoolean**(<переменная>) - проверяет, является ли переменная логической.

**IsList**(<переменная>) - проверяет, является ли переменная списком.

**IsStruct**(<переменная>) - проверяет, является ли переменная структурой.

**IsFree**(<переменная>) - проверяет, является ли переменная свободной.

**Предикаты для работы с базами данных.**

Предикаты открытия, закрытия и создания баз данных отсутствуют, так как объявленные базы данных открываются (и создаются, если не были созданы ранее) при запуске программы.

**DBAppendZ**(<имя базы>:String, <поле1>, ... , <полеN>) - добавляет запись в конец базы данных. Количество полей и их типы должны в точности совпадать с полями базы данных. Имя базы данных записывается в кавычках.

**DBAppendA**(<имя базы>:String, <поле1>, ... , <полеN>) - добавляет запись в начало базы данных. Количество полей и их типы должны в точности совпадать с полями базы данных. Имя базы данных записывается в кавычках.

**DBDelete**(<имя базы>:String, <поле1>, ... , <полеN>) - удаляет из базы данных запись. Количество полей и их типы должны в точности совпадать с полями базы данных. Возможно, удалять сразу группу записей, если какие-либо из полей будут обозначены пустой константой nil. Тогда будут удалены все записи базы данных, в которых совпадают значения остальных полей (которые имеют значения отличные от nil).

**DBClear**(<имя базы>:String) - очищает базу данных.

**Предикаты работы с файлами.**

Данная версия Пролога работает только с текстовыми файлами.

Для идентификации файла при работе программы используется целочисленный номер обработчика файла, по которому можно обращаться к файлу.

**FileOpenRead**(<имя файла>:String, <номер обработчика>:Integer) - открыть файл для чтения. Предикат возвращает во втором параметре номер обработчика файла. Во втором параметре предиката должна стоять свободная переменная.

**FileOpenWrite**(<имя файла>:String, <номер обработчика>:Integer) - открыть файл для записи. Предикат возвращает во втором параметре номер обработчика файла. Во втором параметре предиката должна стоять свободная переменная.

**FileRead**(<номер обработчика>:Integer,<значение>) - чтение из файла. Второй параметр должен быть свободной переменной, иначе интерпретатор выдаст ошибку "Неверные параметры при вызове предиката".

**FileWrite**(<номер обработчика>:Integer,<значение>:String) - запись в файл.

**FileClose**(<номер обработчика>:Integer) - закрытие файла. Данный предикат применять не обязательно, так как при завершении программы интерпретатор сам закрывает все открытые файлы.

**EOF**(<номер обработчика>:Integer) - проверка конца файла. Предикат истинен, если конец файла достигнут.

**Разное.**

**StringToList** (String, <список>) - превращает строку в список, состоящий из символов этой строки, и возвращает его через параметр <список>.

**Fail** - предикат всегда возвращает ложь.

## 3.8 Описание функций

**Арифметические функции.**

**Sin** (<Integer, Real>):Real - операция синуса. Аргументом может быть как реальное, так и целое число. Функция возращает реальное число.

**Cos** (<Integer, Real>) :Real - операция косинуса.

**Tan** (<Integer, Real>) :Real - операция тангенса.

**Exp** (<Integer, Real>) :Real - экспонента.

**Ln** (<Integer, Real>) :Real - натуральный логарифм.

**Int** (<Integer, Real>) :Integer - выделение целой части числа. Функция используется также для явного преобразования реального числа в целое.

**Frac** (<Integer, Real>) :Real - выделение дробной части числа.

**Abs**(<Integer, Real>):<Integer, Real> - взятие модуля числа.

**Функции работы со строками.**

**SubStr**(String, N1:Integer, N2:Integer):String - выделение подстроки, начиная с элемента с индексом N1 длиной N2 символов.

**FindStr**(S1:String, S2:String):Integer - находит позицию подстроки S2 в строке S1 и возвращает индекс первого символа подстроки в строке S1. Если в строке S1 не было найдено подстроки S2, то функция возвращает 0.

**Chr**(Integer):String - возвращает символ, соответствующий числу по таблице ASCII.

**Asc**(String):Integer - возвращает ASCII-код первого символа строки.

**NumbToStr**(<Integer или Real>):String - превращает число в строку

**StrToNumb**(String):<Integer или Real> - превращает строку в целое или реальное число, в зависимости от содержания строки. Интерпретатор сначала пытается преобразовать строку в целое число. Если не получается, то пытается превратить строку в реальное число. Если преобразование не удалось, то выводится сообщение об ошибке.

**Логические функции.**

**Not**(Boolean):Boolean - инвертирует значение логического выражения. (ВНИМАНИЕ: чтобы инвертировать значение, возвращаемое предикатом или базой данных необходимо перед именем предиката или базы данных поставить символ "~")

# 4 Организационно-экономическая часть

## 4.1 Расчет затрат на разработку интерпретатора Пролог

Трудоемкость разработки программного обеспечения рассчитывается по формуле:

 ч/м, где (4.1)

*L* - объем программы в операторах, шт.,

*nn* - норма производительности труда программиста, представляющая собой объем полезной работы, выполняемой в единицу времени,

*Ф* - среднемесячный фонд рабочего времени,

*Кн* - коэффициент новизны,

*Кун* - коэффициент использования в разработке типовых программ.

*L* = 10000

*nn* = 50

*Ф* = 25

*Кн* = 0.8

*Кун* = 0.7

Таким образом, трудоемкость разработки ПО составляет:

*Tn* = (10000·0.8·0.7)/(25·50) = 4.5 ч/м

Расчет стоимости ч/м программиста:

* заработная плата программиста составляет 460 руб.
* дополнительная заработная плата составляет 20% от основной, то есть 92 руб.
* Начисления на заработную плату ,в целом, составляют 38.5% от основной и дополнительной заработной платы, 212.52 руб.

Итого стоимость ч/м:

460+92+212.52=764.52 руб.

Накладные расходы составляют:

Знакл=0.25·460=115 руб.

Стоимость одного машинного часа с учетом первоначальной стоимости ЭВМ, стоимости потребляемой электроэнергии и износа машины определяетсяпо формуле:

Sмч=Sмаш1+Sэл+Sрем.

Срок службы ЭВМ - 8 лет. Первоначальная стоимость - 20000 руб. Потребляемая мощность 0.2 кВт/ч.

При эксплуатации машины 8 часов в день имеем:

Sмаш1=20000/(8·12·22·8)=1.18 руб/ч.

Принимая стоимость электроэнергии 0.1 руб/КВт, стоимость потребляемой энергии в час равна:

Sэл=0.1·0.2=0.02 руб.

Затраты на профилактическое обслуживание и ремон составляют 20% от стоимости машины:

Sрем=0.2·1.18=0.17руб.

Тогда стоимость одного машинного часа равна

Sмч=1.18+0.2+0.17=1.37 руб/ч.

Себестоимость программного продукта определяется:

, где

*Зод* - сумма основной и дополнительной заработной платы разработчика программы, начисления на нее и накладные расходы, руб./мес.,

*Тап* - время затрачиваемое разработчиком на разработку, составление алгоритма и написание программы, мес.,

*См* - себестоимость одного часа машинного времени, руб.,

*Кисп* = 0.5 - коэффициент использования ЭВМ.

764.52·4.5+8·4.5·1.37·25·0.5= 4056.84 руб.

**5 Промышленная экология и безопасность**

## 5.1 Введение

Охрана труда – система законодательных актов и норм, направленных на обеспечение безопасности труда, и соответствующих им социально-экономических, технических, организационных и санитарно-гигиенических мероприятий.

Полностью безопасных и безвредных производств не бывает. Задача охраны труда – свести к минимуму вероятность поражения или заболевания работающего с одновременным обеспечением комфорта для плодотворного труда.

Современное промышленное производство связано с использованием сложных технологических процессов и разнообразного оборудования, являющих источниками физических, химических и других факторов, оказывающих прямое или косвенное влияние на безопасность, здоровье и работоспособность человека.

Конструктивное несовершенство технических устройств и неправильная организация труда может приводить к действию на человека неблагоприятных психофизических факторов.

Нормализация и оптимизация производственной среды и предупреждение вредных выбросов в окружающую среду является необходимым условием жизнедеятельности человека. Улучшение условий труда, повышение его безопасности и безвредности имеет большее экономическое значение. Оно влияет на производительность труда, качество и себестоимость продукции.

В разделе «Промышленная экология и безопасность» выявляются наиболее существенные факторы, и производится выбор, обоснование и расчеты средств защиты и систем нормализации труда и предотвращения вредных выбросов в окружающую среду.

## 5.2 анализ характера загрязнения окружающей среды при производстве вычислительной техники

### 5.2.1 Источники загрязнения

В общем случае вычислительная техника представляет собой некоторую конструкцию, то есть совокупность деталей, находящихся в определенных пространственных, механических, электрических, магнитных и энергетических взаимосвязях. Поэтому в процессе производства вычислительной техники используется целый комплекс технологических приемов, связанных с переработкой различных по своей природе исходных материалов, последующей обработкой и сборкой деталей для получения функционально завершенного изделия.

В технологиях производства ЭВМ используются процессы, отрицательно воздействующие на окружающую среду, такие как литье, термическая, гальваническая и механическая обработка, резка, пайка, сварка и окраска. Источники, объекты первичного отрицательного воздействия и применяемые способы защиты среды обобщены в таблице 5.4.

Литейное производство связано с загрязнением атмосферы пылью, окисью углерода, сернистым ангидридом, а сточных вод механическими взвесями, в виде пыли, флюсов, окалины.

При термической обработке в атмосферу через систему вентиляции могут выбрасываться пары масла, окиси углерода, аммиака, цианистого водорода, а также пыли. Электротермическое оборудование потребляет воду для охлаждения, и в сточных водах могут находиться вредные вещества.

Гальванические работы сопряжены с использованием больших объемов воды для приготовления растворов электролитов и промывных операций. Поэтому сточные воды в этих случаях значительно загрязнены ядовитыми химическими веществами. Кроме того, воздух, удаляемый от технологического гальванического оборудования, содержит большое количество вредных веществ в различных агрегатных состояниях:

* капельножидким (брызги),
* тонкодисперсионном аэрозоле,
* паро- и газообразном.

При механической обработке материалов для охлаждения оборудования и инструмента, промывки деталей, санитарно-гигенической обработки помещений широко используется вода. сточные воды в этих случаях могут быть загрязнены минеральными маслами, мылами, металлической и абразивной пылью, эмульгаторами. Кроме того, при механической обработке металлов в атмосферу через систему вентиляции могут выбрасываться пыль, стружка, туманы масел и эмульсий, а при обработке неметаллических материалов – вредные пары связующих смол и пыль.

Таблица 5.1.

|  |  |  |  |
| --- | --- | --- | --- |
| Технологический процесс | Объект отрицательного воздействия | Источник загрязнения | Способ защиты |
| Литье | Атмосфера | Пыле- газовыделение | Пылеулавливание, фильтрация |
| Гидросфера | Сточные воды | Фильтрование, отстаивание, реагентная обработка |
| Термическая обработка | Атмосфера | Пыле- газовыделение | Пылеулавливание, фильтрация |
| Гидросфера | Сточные воды | Фильтрование, отстаивание, реагентная обработка |
| Гальваническая обработка | Атмосфера | Выделение вредных веществ в различном агрегатном состоянии | Очистка |
| Гидросфера | Сточные воды | То же |
| Механическая обработка | Атмосфера | Пыле- газовыделение | Пылеулавливание, фильтрация |
| Гидросфера | Сточные воды | То же |
| Резка, сварка, пайка | Атмосфера | Пыле- газовыделение | Пылеулавливание, фильтрация |
| Гидросфера | Сточные воды | То же |
| Лакокрасочные работы | Атмосфера | Газовыделение, лакокрасочные туманы | Фильтрация |
| Гидросфера | Сточные воды | То же |
| Сборка | Гидросфера | Сточные воды | То же |

Газовая и плазменная резка металлов, технологические процессы сварки и пайки сопровождаются выделением пыли и токсичных газов, а сточные воды могут загрязняться механическими примесями, кислотами.

Лакокрасочные работы связаны с выделением в атмосферу вредных веществ в вид паромов растворителей и лакокрасочных аэрозолей в процессе нанесения покрытия и при высыхании изделий. При уборке такого рода помещений сточные воды могут загрязняться примесями растворителей лаков и красок.

Процесс получения функционально завершенного изделия заканчивается сборочными операциями. Отрицательное воздействие на окружающую среду процессов сборки менее ощутимо. Однако и в этих случаях при проведении санитарно-гигенической обработки производственных помещений в сточные воды могут попадать различные нежелательные примеси.

### 5.2.2 Очистка воздуха от вредных примесей

Участок сверления и рубки печатных плат производит выброс стеклопластиковой пыли – вещества 3 класса опасности по ГОСТ 12.1.007-76 («Вредные вещества. Классификация и общие требования безопасности»).

Для защиты рабочего персонала от воздействия примеси в помещении участка применена вентиляционная система со скоростью движения воздуха в вытяжных столах 1.5 м/сек и общим расходом очищаемого воздуха на участке сверления и рубки печатных плат L=18000 м3/ч.

Для предотвращения выброса стеклопластиковой пыли в атмосферу применяем в качестве пылеуловителя циклон.

Исходные данные:

объем очищаемого вентиляционного воздуха Q=5 м3/с,

плотность газа при рабочих условиях r=1.3 кг/м3,

вязкость газа m=42.2⋅10-6 Па⋅с,

дисперсионный состав пыли d50 и lgsч=0.77,

rч=1300 кг/м – плотность частиц пыли.

h=0.9 – требуемая эффективность очистки газа.

Расчет циклона ведется методом последовательных приближений в следующем порядке:

1. Выбираем тип циклона – ЦН-24 и определяем оптимальную скорость газа wоп в сечении циклона с диаметром D. Из таблицы 1: wоп=4.5 м/с.

2. Вычисляем диаметр циклона D(м) по формуле =0.3015 м.

Полученное значение округляем до ближайшего типового значения. Внутренний диаметр циклона выбираем D=0.3 м.

3. По выбранному диаметру циклона находим действительную скорость движения газа в циклоне:

=4.76 м/с,

где n=1 – количество циклонов.

4. Определяем коэффициент гидравлического сопротивления одиночного циклона:

=67,

где

*k1*=0.93 – поправочный коэффициент на диаметр циклона;

*k2*=0.9 – поправочный коэффициент на запыленность газа;

=80 – коэффициент гидравлического сопротивления одиночного циклона диаметром 500 мм, при Cвх=80 г/м.

5. Гидравлическое сопротивление циклона вычисляется по формуле:

=971 Па/м.

6. Эффективность очистки газа в циклоне вычисляется по формуле:

,

где

 - табличная функция от параметра x, равного:

.

Для выбранного циклона значение =8.5 мкм, =0.308.

Значение определяем по формуле:

=87,

где значение определено по условиям работы типового циклона: *DT*=0.6 м, *rчт*=1930 кг/м3, *mT*=22.2\*10-6 Па\*с, *wT*=3.5 м/с.

Определив значение *x*, по данным таблицы 6 находим параметр =0.8413.

=0.0921.

Полученный коэффициент очистки больше требуемого, следовательно, тип циклона выбран верно.

## 5.3 Анализ влияния опасных и вредных факторов, при

## эксплуатации программы интерпретатора Пролог

Разрабатываемая в данном дипломном проекте модель внешней среды, представляет собой программный комплекс, функционирующий на средствах вычислительной техники, при эксплуатации которых возникают следующие опасные и вредные факторы:

Физические:

* повышенный уровень шума на рабочем месте;
* опасный уровень напряжения в электрической цепи, замыкание которой может произойти через человека;
* пожарная опасность;
* повышенный уровень электромагнитных излучений;
* повышенная яркость света;
* прямая и отраженная блеклость;
* нарушение микроклимата рабочих помещений.

Психофизиологические:

* гиподинамия;
* умственное перенапряжение;
* перенапряжение зрительных анализаторов;

### 5.3.1 Повышенный уровень шума на рабочем месте

Шум при работе средств вычислительной техники возникает при работе вентиляторов блоков питания аппаратуры, устройств вывода информации – принтеров, графопостроителей.

При длительном воздействии шума на человека происходят нежелательные явления: снижается острота слуха, повышается кровяное давление. Кроме того, шум оказывает воздействие на общее состояние человека, вызывая чувства неуверенности, стесненности, тревоги, плохого самочувствия, что приводит к снижению производительности труда, возникновению ошибок, может стать причиной травматизма.

Ослабления шума можно достигнуть следующими способами:

* уменьшение шума в источнике;
* изменение направленности излучения;
* рациональная планировка рабочего помещения (звукоизоляция стен, окон, дверей, потолка; установка штучных звукопоглощателей; размещение более тихих помещений вдали от шумных);
* борьба с шумом на пути его распространения (звукоизолирующие ограждения, кожухи, экраны, кабины).

Уровень шума на рабочем месте должен соответствовать требованиям ГОСТ 12.1.003-83. Согласно этому ГОСТу уровень звукового давления на постоянных рабочих местах в производственных помещениях при продолжительности шума более четырех часов должен соответствовать данным в таблице 5.2

|  |  |  |
| --- | --- | --- |
| Помещения | Среднегеометрические частоты октавных полос, Гц | Уровень звука, дБА |
| 31.5 | 63 | 125 | 250 | 500 | 1000 | 2000 | 4000 | 8000 |
| Уровни звукового давления, дБ |
| постоянние рабочие места в офисных помещениях | 86 | 71 | 61 | 54 | 49 | 45 | 42 | 40 | 38 | 50 |

### 5.3.2 Опасный уровень напряжения электрической цепи, замыкание которой может произойти через человека

Все используемые в данной разработке средства вычислительной техники в качестве источника питания используют переменное напряжение сети 220В, что может повлечь за собой поражение человека электрическим током. В соответствии с классификацией помещений по степени электрической опасности используемое рабочее помещение можно отнести к особо опасным.

Причинами поражения человека электрическим токов являются:

* случайное прикосновение или приближение на опасное расстояние к токоведущим частям, находящимся под напряжением;
* появление напряжения на металлических конструктивных частях электрооборудования – корпусах, кожухах и т.д. в результате повреждения изоляции и других причин;
* появление напряжения на отключенных токоведущих частях, на которых работают люди, вследствие ошибочного включения устройства;
* возникновение шагового напряжения на поверхности земли в результате замыкания провода на землю.

При работе аппаратуры запрещается:

1. проверять на ощупь наличие напряжения токоведущих частей аппаратуры;
2. применять для соединения блоков и приборов провода с поврежденной изоляцией;
3. проводить работу и монтаж в аппаратуре, находящейся под напряжением;
4. подключать блоки и прибора к работающей аппаратуре.

Чтобы избежать поражения электрическим током используются следующие технические средства, обеспечивающие безопасность работ в электроустановках:

* защитное заземление;
* защитное отключение;
* зануление;
* выравнивание потенциалов.

### 5.3.3 Пожарная опасность

Пожары в ВЦ представляют особую опасность, так как сопряжены с огромными материальными потерями.

В качестве горючего компонента на ВЦ могут служить строительные материалы для акустической и эстетической отделки помещений, перегородки, окна, двери, полы, мебель, стеллажи, магнитные ленты и диски, изоляция силовых кабелей, а также радиотехнические детали и соединительные провода электронной схемы.

Окислитель в виде кислорода воздуха имеется в любой точке помещения ВЦ.

Источниками воспламенения на ВЦ могут быть электрические искры, дуги и перегретые участки. Источники воспламенения возникают в электронных схемах, кабельных линиях, вспомогательных электрических и электронных приборах, а также в устройствах, применяемых для технического обслуживания элементов ЭВМ.

Таким образом, на ВЦ могут присутствовать все три основные фактора, способствующих возникновению пожара.

Кабельные линии электропитания состоят из горючего изоляционного материала, а также содержат вероятные источники открытого огня. Они являются –наиболее опасным элементом в конструкции ЭВМ и вычислительного центра с точки зрения возникновения и развития пожара.

Другим местом, где может возникнуть пожар, является хранилище информации. Ущерб от пожара определяется не только стоимостью сгоревших магнитных лент и дисков, но и потерей информации, записанной на ней.

Для обеспечения своевременных мер по обнаружению и локализации пожара, эвакуации рабочего персонала, а также для уменьшения материальных потерь необходимо выполнять следующие условия:

* наличие системы автоматической пожарной сигнализации;
* наличие эвакуационных путей и выходов;
* наличие первичных средств тушения пожаров: пожарные стволы, внутренние пожарные водопроводы, сухой песок, огнетушители.

Следует обратить особое внимание на то, что применение воды в машинных залах ЭВМ, ввиду опасности повреждения дорогостоящего электронного оборудования возможно только в исключительных случаях, когда пожар угрожает принять крупные размеры.

### 5.3.4 Повышенный уровень электромагнитных излучений

Электромагнитные излучения влияют на нервную систему человека, изменяют ориентацию клеток и цепей молекул в соответствии с направлением силовых линий электрического поля, биохимическую активность молекул и состав крови.

Действующие нормы СНиП 848-70 предусматривают следующие предельно допустимые величины:

1. напряженность электромагнитных полей радиочастот на рабочих местах на должна превышать по электрической составляющей 20 В/м в диапазоне частот 100 кГц-30МГц и 5 В/м в диапазоне 30-300 МГц;
2. по магнитной составляющей предельная величина равна 5 А/м в диапазоне частот 100 кГц-1.5МГц.

Ослабления мощности электромагнитного поля можно достичь следующими способами:

1. увеличить расстояние между источником и рабочим местом;
2. установить поглощающий или отражающий экран между источником и рабочим местом.

### 5.3.5 Повышенная яркость света

Свет является важным стимулятором не только зрительного анализатора, но и организма в целом, а также общей работоспособности человека. Положительное влияние его на производительность труда и качество работы в настоящее время не вызывает сомнений.

Обеспечение гигиенически рациональных условий освещения способствует длительному сохранению работоспособности, что приводит к росту производительности труда и к снижению ошибок в процессе труда.

Поскольку при работе с программным комплексом главным источником визуального отображения информации является монитор, который представляет собой самосветящийся прибор, то общая освещенность может вызвать перегрузку зрительных органов, что приводит к повышенному утомлению зрения в процессе выполнения работ и повышает опасность травматизма зрительных органов.

Освещенность рабочей зоны должна соответствовать нормам СНиП 11-4-79 «Искусственное освещение для зрительной работы малой степени точности (разряд V) и работа с самосветящимися материалами (разряд VII).

Яркость в поле зрения работающего должна быть распределена равномерно. Поскольку в поле зрения работающего постоянно находятся поверхности, значительно отличающие по яркости (например: экран монитора – текстовый документ и т.д.) то при переводе взгляда в яркоосвещенной на слабоосвещенную поверхность глаз должен переадаптироваться. Частая переадаптация ведет развитию утомления зрения. Степень неравномерности определяется коэффициентом неравномерности, который согласно требованиям СНиП 11-4-79 для данного вида работ должен быть не менее 0.3 в пределах рабочей области.

### 5.3.6 Прямая и отраженная блеклость

Прямая блеклость создается в основном источниками света и осветительными приборами. Находящиеся в поле зрения открытые лампы приводят к быстрому утомлению зрения и снижению производительности труда. Отраженная блеклость создается рабочими поверхностями, обладающими коэффициентом зеркального отражения по направлению к глазу работающего. Отраженная блеклость вызывает ослепленность и ведет к увеличению утомления зрения. Наиболее всего наличие блеклости присуще мониторам, которые функционально являются источником света и частью рабочей области.

Блеклость монитора устраняется следующими способами:

1. должна быть возможность изменения его положения: поверхность экрана монитора устанавливается относительно вертикальной плоскости на +6…-15 градусов; относительно горизонтальной на 0…45 градусов;
2. применение специальной арматуры – экранных фильтров;
3. высота подвеса светильников должна соответствовать нормам СНиП 11-4-79.

### 5.3.7 Нарушение микроклимата рабочих помещений

Метеорологические условия или микроклимат в производственных помещениях определяется следующими параметрами:

1. температура воздуха, °С;
2. относительная влажность, %;
3. скорость движения воздуха на рабочем месте, м/с.

Работа с комплексом программно-аппаратных средств относится к работам легкой тяжести (I категория), т.к. выполняется сидя или стоя, может быть связана с ходьбой, но не требует систематического физического напряжения и переноски тяжестей. Согласно категории проводимых работ и ГОСТ 12.1.005-76 оптимальные параметры микроклимата при проведении данной работы должны соответствовать значениям, указанным в таблице 5.3.

Таблица 5.3.

|  |  |  |  |
| --- | --- | --- | --- |
| Период года | Температура, °С | Относительная влажность воздуха, % | Скорость воздуха, м/с |
| Холодный и переходный | 20-23 | 60-40 | 0.1 |
| Теплый | 22-25 | 60-40 | 0.1 |

Запыленность воздуха в машинном зале не должна превышать 0.2 мг/м2 при размере частиц не более 2 мкм.

Для обеспечения установленных параметров микроклимата и чистоты воздуха в машинном зале применяют вентиляцию. В ВЦ применяют общеобменную искусственную вентиляцию в сочетании с местной, как искусственной, так и естественной. Общеобменная вентиляция используется для обеспечения в помещениях ВЦ соответствующих микроклиматических параметров; местная – для охлаждения собственно ЭВМ и вспомогательного оборудования.

В помещениях ВЦ предусматривают систему отопления, которая должна обеспечивать достаточное, постоянное и равномерное нагревание воздуха в холодный период года.

### 5.3.8 Защита от психофизиологических факторов

Защита от психофизиологических факторов осуществляется за счет реализации эргономических требований, предъявляемых к аппаратуре и рабочему месту в целом.

Эргономические требования в комплексе выражают три стороны деятельности человека – эффективность работы, сохранение здоровья и развитие личности в процессе труда. Эти требования определяются характеристиками человека. В перечне общих требований эргономики различают требования, учитываемые в процессе разработки ПЭВМ (требования к информации, предъявляемой человеку-оператору и к техническим средствам) и требования, учитываемые при эксплуатации изделий (требования к рабочим местам и рабочей среде).

## 5.4 Анализ использования защитных экранов для снижения

## влияния опасных и вредных факторов, во время работы на

## автоматизированном рабочем месте

Основным источником вредных факторов, оказываемых наиболее существенное влияние на здоровье оператора ЭВМ, является электронно-лучевой монитор.

Мероприятия по устранению данной проблемы могут осуществляться двумя путями:

* Отказ от электронно-лучевой технологии. Основной проблемой данного способа является чрезмерная дороговизна мониторов, изготовленных по другим технологиям.
* Применение защитных экранов. Применение защитных экранов является наиболее приемлемым (в плане затраты / результат) способом борьбы с воздействиями электронно-лучевых трубок.

### 5.4.1 Основные функции защитных экранов, необходимые для

### снижения влияния вредных и опасных факторов, во время работы

### с программным средством

Защита зрения оператора является основной функцией экрана. Зрение больше всего страдает от –повышенной яркости экрана электронно-лучевого монитора и недостаточного контраста изображения. Защитный экран уменьшает общую яркость изображения, вместе с тем темные участки изображения остаются хорошо различимыми, поскольку сильно увеличивается общий контраст и устраняются блики.

Экраны выполняют следующие защитные функции:

### 5.4.1.1 Защита от электростатического и электромагнитного

### воздействий

Основным источником вредного воздействия на организм человека являются электромагнитные колебания низкой частоты, связанные с работой схем развертки электронного луча. Они воздействуют на обмен веществ в организме и могут приводить к патологическим изменениям в клетках мягких тканей.

Другим источником вредного воздействия является электростатический заряд, скапливающийся на лицевой поверхности монитора. Вызываемая им деионизация атмосферы вокруг оператора угнетающе действует на нервную систему, способствуя развитию депрессии у оператора.

### 5.4.1.2 Защита от рентгеновского излучения

Рентгеновское излучение, возникающее при торможении электронного луча на внутренней поверхности кинескопа, является еще одним источником вредных воздействий на человека, приводящим к серьезным нарушениям функций организма на атомарном уровне.

### 5.4.1.3 Защита от ультрафиолетового излучения

Синий люминофор экрана имеет частичное излучение ультрафиолетовой области спектра. Это воздействие существенно при длительной работе с компьютером, приводящее к заболеваниям сетчатки глаза.

**5.4.2 Анализ основных типов защитных экранов, которые**

**приемлемы для снижения влияния вредных и опасных факторов,**

**во время с программным средством**

На сегодняшний день, на рынке имеются различные типы защитных экранов. Рассмотрим описания некоторых типов защитных экранов, которые получили наибольшее распространение:

### 5.4.2.1 Сетчатый (частичная защита зрения)

Экран представляет собой тонкую нейлоновую сетку, натянутую на пластмассовую рамку. Экран обеспечивает некоторое увеличение контрастности изображения и уменьшение общей яркости экрана.

Основные защитные свойства:

* улучшение контраста 400-500 %;
* уменьшение общей яркости 50%.

Недостатки: данный тип экрана не рекомендуется для использования, поскольку возникающие побочные явления (муар, ореол) вызывают неприятные ощущения у оператора.

### 5.4.2.2 Стеклянный двухслойный с заземлением (частичная защита

### зрения, частичная защита здоровья)

Экран изготовлен из специальных сортов свинцового стекла. Стекло заключено в пластмассовую рамку с устройством крепления к монитору и гнездом подключения к заземлению.

Основные защитные свойства:

* электростатическое поле 70-80%
* рентгеновское излучение 99.6%
* подавление бликов 97.3%
* улучшение контраста 450-550%
* уменьшение общей яркости 55%

Недостатки: коэффициент подавления бликов 97.2% (коэффициент зеркального отражения 2.8%) недостаточен для эффективного подавления бликов. Подключение заземления является фикцией, поскольку отсутствуют проводящие слои, нанесенные на стекло.

### 5.4.2.3 Стеклянный многослойный с заземлением (полная защита

### зрения, полная защита здоровья)

Экран изготовлен из специальных сортов стекла сильно легированных атомами тяжелых металлов, на стороне обращенной к пользователю нанесено полиэфирное и пятислойное диэлектрические покрытия, на обратной стороне вакуумным напылением нанесен слой металлического серебра.

Стекло заключено в пластмассовую рамку с устройством крепления к монитору и гнездом подключения заземления.

Основные защитные свойства:

* электростатическое поле 99.9%;
* НЧ поле 99.9%;
* ультрафиолетовое излучение 100%;
* рентгеновское излучение 99.6%;
* подавление бликов 99.3%;
* улучшение контраста 950-990%;
* уменьшение общей яркости 68%.

## 5.5 Расчет необходимого звукопоглощения, при работе с АРМ

Шум при работе с компьютером возникает при работе вентиляторов охлаждения блоков питания ЭВМ и при работе печатающих устройств. Необходимо отметить, что разница уровня шума между этими устройствами превышает 10 Дб, поэтому шум работы вентилятора охлаждения блока питания ЭВМ можно в расчет не принимать.

Рассмотрим требуемое снижение уровня звукового давления для помещения 5×10×4 метров.

dLтр = L - Lдоп,

где

L – расчетные или измеренные уровни звукового давления

Lдоп – допустимые уровни уровня звукового давления по ГОСТ 12.1003-83.

L=Lпринтера + 10Lg(Σ(X/Si) + 4/B),

где

i измеряется от 1 до m, m – количество источников шума, в нашем случае 2;

X – коэффициент, зависящий от отношения расстояний между источниками шума r и r1, причем r=1.5м; r1=3.4м и X=1.

Si – площадь полусферы, проходящая через расчетную точку i и окружающий источник, S1=14.1м2 и S2=72.6м2;

B – постоянная помещения, причем B=B1000\*M, где B1000 – постоянная помещения на частоте 1000 Гц (для помещения с жесткой мебелью и объемом 2000 м3 имеем B1000=10м2) и M – частотный множитель.

Расчет представлен в таблице 5.3:

Таблица 5.3.

|  |  |
| --- | --- |
| Показатели | Среднегеометрические показатели октавных полос, Гц |
| 63 | 125 | 250 | 500 | 1000 | 2000 | 4000 | 8000 |
| Lпринетра, дБ | 65 | 67 | 68 | 70 | 73 | 73 | 74 | 75 |
| M | 0.8 | 0.75 | 0.7 | 0.8 | 1 | 1.4 | 1.8 | 2.5 |
| B, м2 | 8 | 7.5 | 7 |  |  |  |  |  |
| 10Lg(Σ(X/Si) + 4/B) | -2.3 | -2 | -1.8 | -2.3 | -3.1 | -4.3 | -5.1 | -6.1 |
| L, дБ | 63 | 65 | 66 | 68 | 70 | 69 | 69 | 69 |
| Lдоп, дБ | 83 | 74 | 68 | 63 | 60 | 57 | 55 | 54 |

Рассчитаем площадь, которую необходимо покрыть звукопоглощающим материалом, чтобы получить снижение шума на 15дБ. В качестве звукопоглощающего материала будем использовать маты из базальтового волокна.

Sобл.=dA/a

где

dA – добавочное звукопоглощение, вносимое облицовкой;

A – ревибрационный коэффициент поглощения.

Для выбранного материала при частоте 8кГц имеем a=0.7. площадь ограничивающих поверхностей равна:

Sогр=Sпотолка+Sпола+2\*Sп.стены+2\*Sл.стены-2\*Sокна-Sдвери= 50+50+80+40-4-2=214 м2.

По номограмме определяем dA для a=0.7 и Sогр=214м2, получаем dA=130м2. Отсюда имеем Sобл=dA/a=130/0.7=163м2.

Если облицевать потолок и стены, то получим Sобл=164м2, что приведет снижение уровня до допустимой нормы.

# Заключение

В результате проектирования был разработан интерпретатор языка Пролог с возможностью работы с универсальными базами данных.

Для удобства работы с программы была создана интегрированная среда разработчика, имеющая средства для визуального ввода программы, систему меню, экранных форм и оперативной подсказки, правила работы с которыми изложены в руководстве пользователя.

Был проведен расчет затрат на разработку программы интерпретатора и интегрированной среды разработчика.

Рассмотрены вопросы, связанные с промышленной экологией и безопасностью при работе с программой интерпретатора.

Предлагаемая программа интерпретатора языка Пролог создана в соответствии с требованиями технического задания.

# Литература

1. Дж. Малпас Реляционный язык Пролог и его применение - М.:Наука,1990
2. Логический подход к искусственному интеллекту под ред. Г.П.Гаврилова - М.:Мир,1990
3. Д. Марселлус Программирование экспертных систем на Турбо Прологе - М.:Финансы и статистика, 1994
4. У. Клоксин, К.Меллиш Программирование на языке Пролог - М.:Мир, 1987
5. Язык Пролог в пятом поколении ЭВМ под ред. Н.И. Ильинского - М.:Мир, 1988
6. Дж. Доорс, А.Р.Рейблейн, С.Вандера Пролог-язык программирования будущего - М.:Финансы и статистика, 1990
7. И. Братко Программирование на языке Пролог для искусственного интеллекта - М.:Мир, 1990
8. Ахо А. Теория синтаксического анализа, перевода и компиляции - М.:Мир, 1979

# Приложения

##

## Приложение 1

Интерфейсная часть модуля интерпретатора

unit CompileUnit;

interface

uses Windows,Classes, SysUtils, ProgFormUnit, ProgramClasses, SyncObjs,

Forms, dbTables, ProjectOptionsFormUnit;

{DEFINE PRED\_COMPILER\_DEBUG}

const

StandardLexemsCount=16;

type

TSubStackNodeType=(ssntPredicate, ssntStdPredicate,

ssntArithmetic, ssntExtData);

TProgramSubStackNode=class(TObject)

ClauseIndex :integer; //индекс предложения, на котором

//была достигнута истина

CreatedVariables :TStrings; //созданные на этом шаге переменные

FreeBefore :TStrings; //Переменные, которые перед этим шагом были свободны

TempVariables :TPrologVariables;

//Arithmetic :Boolean; //True-если арифметический терм

iType :TSubStackNodeType; //Тип элемента

Term :TPrologTerm; //Терм в программе, которому

//соответсвует данный элемент

Belong :TObject; //объект TProgramStackNode,

//которому принадлежит данный объект

ExtDataQuery :TQuery;

TheEndWasReached :Boolean;

//VarPacket :TPrologVariables; //пакет с переменными,

//который отправляется дальше

procedure ClearCreatedVariables;

procedure StepBack;

constructor Create;

destructor Destroy; override;

end;

TTraceMode=(tmNoTrace, tmTrace, tmStep);

//tmNoTrace - нет трассировки

//tmTrace - трассировка

//tmStep - трассировка, не заходя внутрь вызываемых предикатов

TProgramStackNode=class(TObject)

//полностью описывает текущее состояние программы

PredicateName :string; //имя предиката

Predicate :TProgramPredicate;

InputParameters :TPrologVariables;

Variables :TPrologVariables; //переменные

SubStack :array of TProgramSubStackNode;//integer; //массив, в котором хранятся индексы,

//на которых было достигнуто истинное значение терма

CreatedVariables :array of array of string; //Массив,

//в котором хранятся имена созданных на i-м шаге переменных

TermNumb :integer; //Номер терма, на котором стоит программа

ClauseNumb :integer; //Номер предложения в предикате

ClausesCount :integer;

TraceMode :TTraceMode;

OnBreakPoint :Boolean; //True-если в данный момент стоим на контрольной точке

constructor Create;

destructor Destroy; override;

end;

TPrologProgram=class(TObject)

Domains :array of TProgramDomain;

ExtData :array of TProgramExtData;

Predicates :array of TProgramPredicate;

Stack :array of TProgramStackNode;

BreakPoints :array of Integer;

StartPredicate :string;

OnTheRun :Boolean; //True-если программа запущена

Files :array of TPrologFile;

function CompileProgram:Boolean;

procedure RecieveStructurData;

procedure CutLexemsToPredicates(Lexems:TLexemsArray);

function CheckingBeforeCompile:Boolean;

procedure RunProgram(TraceMode:TTraceMode);

procedure EraseProgramStack;

function AddNodeToStack:TProgramStackNode;

function FindPredicate(PredicateName:string):TProgramPredicate;

function ExecutePredicate(var StackNode:TProgramStackNode):Boolean;

procedure DeleteLastNodeFromStack;

function DebugProcess(StackNode:TProgramStackNode):Boolean;

function PreDebugProcess(StackNode:TProgramStackNode):Boolean;

procedure CreateBreakPointsArray;

function CheckPredicateClausesCount(

Term:TPrologTerm;

SubStackNode:TProgramSubStackNode):Boolean;

constructor Create;

destructor Destroy; override;

function TranslateLexems (ProgPart:TStrings; //var Lexems:TStringList;

var LexemsRecs:TLexemsArray;

Comments:Boolean=False):TPoint;

function AnalizeLexem(st:string):TLexemRecord;

procedure AddLexem(var LexemsRecs:TLexemsArray;

st:string; x,y,APos:integer);

procedure CheckBreakPoints(StackNode:TProgramStackNode);

end;

var

PrologProgram :TPrologProgram;

function TranslateSintax (var Predicate:TProgramPredicate):TPoint;

function AnalizeListElements(LRecs:TLexemsArray;

var DstVar:TPrologVariable):Boolean;

function AnalizeStructElements(LRecs:TLexemsArray;

var DstVar:TPrologVariable):Boolean;

function AnalizeArguments(Predicate:TProgramPredicate;

Lexems:TLexemsArray; var VArr:TVariablesArray;

CheckFlag:Boolean=True):boolean;

function CheckConstantType(

Param:TPrologVariable; Domain:string):boolean;

function CheckPredicateParameters(SubTerm

:TSubTermPredicate):Boolean;

function GetHelpContext (Lexem: TLexemRecord):LongInt;

function TestReservedFunction(st:String):Boolean;

function TestReservedPredicate(st:String):Boolean;

## Приложение 2

Интерфейсная часть модуля с дополнительными классами интерпретатора.

unit ProgramClasses;

interface

uses Classes, SysUtils, ProgFormUnit, dbtables, db, ProjectOptionsFormUnit;

type

TLexemErrors=(leString,leComment, leOk);

//leString Не найден конец строки

//leComment не найден конец комментария

//Виды лексем

TLexemType=(ltPredicate, ltDomain, ltExtData,

ltVariable, ltFunction, ltUnknown,

ltPlus,ltMinus,ltMultiply,ltDivide,

ltLeftBracket,ltRightBracket,

ltLeftSquareBracket,ltRightSquareBracket,

ltListDivider,ltComma,ltPoint,

ltPointAndComma,ltEqual,ltIf,ltExclamation,

ltString,ltReal,ltInteger,ltTrue,ltFalse,

ltAnonimous, ltGT, ltLT, ltGE, ltLE,

ltReservedPredicate, ltNil, ltAnd, ltOr, ltNotEqual,

ltComment);

TLexemRecord=record

iType :TLexemType;

//Для LPredicate и LDomain обязателен идентификатор в st

st :string;

x,y :integer; //координаты лексемы

AbsPos :integer; //абсолюная позиция в тексте

NoInverse :Boolean; //Для ltPredicate - признак отсутствия или

//наличия знака инверсии

end;

TLexemsArray=array of TLexemRecord;

TPrologVariablesTypes=(vtString,vtInteger,vtBoolean,

vtReal,vtList,vtStruct,vtUnknown,vtAnonimous);

TPrologVariable=class(TObject)

iType :TPrologVariablesTypes; //Тип переменной

Name :string; //Имя переменной

Data :Pointer; //Объект с хранимой информацией

procedure CreateVariable(DomainType:string; vName:string);

procedure DestroyVariable;

constructor Create;

destructor Destroy; override;

procedure ClearVariable;

function CreateCopy:TPrologVariable; //создает точную копию переменной

//Имя переменной переносится и в копию

procedure AssignVariable (v:TPrologVariable); //присваивает значение

//переменной. Имя переменной не меняется

end;

TVariablesArray=array of TPrologVariable;

TPrologListType=(pltStdList,pltHeadTail);

//pltStdList - Элементы списка хранятся в Elements

//pltHeadTail - представлен в виде головы и хвоста(используются только

//два элемента массива elements

TPrologVariableList=class(TObject)

ListName :string; //имя типа списка

ElemName :string; //Имена типа элементов

ListType :TPrologListType; //Тип списка.

Elements :TVariablesArray; //Элементы находятся в Data

DividerPos :integer; //позиция разделителя в случае ListType=pltHeadTail

//указывает, после какого элемента стоит разделитель

procedure ConvertList;

constructor Create;

destructor Destroy; override;

end;

TPrologVariableStruct=class(TObject)

StructName :string; //имя типа структуры

ElemTypes :array of string; //имена типов элементов

Elements :array of TPrologVariable; //Элементы находятся в Data

constructor Create;

destructor Destroy; override;

end;

TPrologVariables=class(TObject)

VarArr :array of TPrologVariable;

constructor Create;

destructor Destroy; override;

function Count:integer;

function High:integer;

procedure AddVariable(v:TPrologVariable);

procedure DeleteVariable(n:integer); overload;

procedure DeleteVariable(vName:string); overload;

procedure DeleteLastVariable;

function AddNewVariable:TPrologVariable;

function VariableByName(vName:string):TPrologVariable;

function GetVariable(ind:integer):TPrologVariable;

procedure SetVariable(ind:integer; v:TPrologVariable);

procedure ClearAndDestroyVariables;

property Variables[ind:integer]:TPrologVariable read GetVariable

write SetVariable; default;

end;

TSubTermPredicate=class(TObject)

Name :string;

StandardPredicate :Boolean; //True - если предикат стандартный

Params :TVariablesArray;

NoInverse :Boolean; //признак отсутствия инверсии

constructor Create;

destructor Destroy; override;

end;

TSubTermExtData=class(TObject)

Name :string;

Params :TVariablesArray;

NoInverse :Boolean; //признак отсутствия инверсии

constructor Create;

destructor Destroy; override;

end;

TExpressionOperation=

(eoPlus, eoMinus, eoMultiply, eoDivide, eoEqual,

eoGT, eoLT, eoGE, eoLE, eoVariable, eoFunction,

eoAnd, eoOR, eoNotEqual);

TSubTermExpression=class(TObject)

Operation :TExpressionOperation;

VarName :String; //имя перменной

FuncName :string; //имя функции

StringStr :string; //строка для строки константы

NumberInt :integer; //число для целой контанты

NumberReal :Extended; //число для реальной контанты

BooleanVal :Boolean; //Число для логической константы

Operand :TPrologVariable;

LeftHand :TSubTermExpression; //Указатель на данные тип

//TSubTermExpression, TPrologVariable

RightHand :TSubTermExpression;

FuncParams :array of TSubTermExpression;

constructor Create;

destructor Destroy; override;

end;

TPrologTermType=(pttExpression,pttPredicate,pttExtData,pttCutting);

TPrologTerm=class(TObject)

TermType :TPrologTermType; //тип терма

Data :pointer; //указатель на объектом с содержимым терма

x,y :integer;

constructor Create;

destructor Destroy; override;

end;

TTermsArray=array of TPrologTerm;

TProgramDomain=class(TObject)

//Внутренний формат доменов

iType :TDomainTypes;

Name :string;

Params :TStrings;

ListParam :string;

constructor Create;

destructor Destroy; override;

end;

TProgramExtData=class(TObject)

Name :string;

Fields :TStrings;

FieldsTypes :TStrings;

FieldsLengths :TStrings;

FileName :string;

Table :TTable;

DataSource :TDataSource;

constructor Create;

destructor Destroy; override;

function Open:Boolean;

procedure Close;

end;

TPredicateClause=class(TObject)

Params :TVariablesArray;

Terms :TTermsArray;

x,y, Endx, EndY :integer;

constructor Create;

destructor Destroy; override;

end;

TProgramPredicate=class(TObject)

Name :string;

Params :TStrings;

Text :TStrings;

Lexems :TLexemsArray;

Clauses :array of TPredicateClause;

constructor Create;

destructor Destroy; override;

end;

TFileOpenMode=(fomRead,fomWrite,fomNotOpened);

TPrologFile=class(TObject)

Name :string;

HandlerNumb :integer;

OpenMode :TFileOpenMode;

FileVar :TextFile;

constructor Create;

destructor Destroy; override;

procedure OpenRead(FileName:string);

procedure OpenWrite(FileName:string);

procedure Close;

function EndOfFile:Boolean;

procedure ReadFile(var st:string);

procedure WriteFile(st:string);

end;

TVariableSize=record

Name :string;

iType :TPrologVariablesTypes;

size :integer;

end;

const

SimplePrologTypesCount=4;

SimplePrologTypes:set of TPrologVariablesTypes=

[vtString, vtInteger, vtBoolean, vtReal];

PrologVariablesSizes:array [0..5] of TVariableSize=

((Name:'STRING'; iType:vtString; Size:SizeOf(string)),

(Name:'INTEGER'; iType:vtInteger; Size:SizeOf(integer)),

(Name:'BOOLEAN'; iType:vtBoolean; Size:SizeOf(Boolean)),

(Name:'REAL'; iType:vtReal; Size:SizeOf(Extended)),

(Name:'LIST'; iType:vtList; Size:SizeOf(TPrologVariableList)),

(Name:'STRUCT'; iType:vtStruct; Size:SizeOf(TPrologVariableStruct)));

## Приложение 3

Интерфейсная часть модуля с функциями и предикатами интерпретатора.

unit PrologRunTime;

interface

Uses SysUtils,CompileUnit, ProgramClasses, CommonFunctions;

type

TErrorCode=(ecType, ecNo, ecOverflow, ecDivideZero, ecExpressionValue,

ecArgsCount, ecArgType, ecTan, ecRealAsInteger, ecTypeInExtData,

ecListTail,ecPredicateParams,ecExtDataAbsent, ecExtDataDelete,

ecRangeError,ecConvertError,ecFileOpenError,ecFileCloseError,

ecFileAccessError,

ecCloseProgram,

ecStopPrologProgram);

//ecType - ошибка типа

//ecNo - нет ошибок

//ecOverflow - переполнение

//ecDivideZero - деление на ноль

//ecExpressionValue - выражение возвращает нелогическое значение

//ecArgsCount - неверное количество аргументов у функции

//ecArgType - ошибка типа аргумента функции

//ecTan - ошибка выполнения операции тангенса

//ecRealAsInteger - ошибка конвертирования реального числа в целое

// возникает, когда функции требуется целое число, а у предлагаемого

// аргумента функции есть ненулевая дробная часть

//ecTypeInExtData - ошибка типа при вызове базы данных

//ecListTail - разделение списка на голову и хвост оказалось неуспешным

//ecCloseProgram - закрытие программы

//ecStopPrologProgram - остановка программы

//ecPredicateParams - неверные параметры предиката

TRunTimeError=class(TObject)

Code :TErrorCode;

PredicateName :string;

x,y :integer;

constructor Create;

procedure SetError(err:TErrorCode);

procedure ShowOnScreen;

end;

TOperatorFunction=function (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

TStdFunction=function (Args:TPrologVariables):TPrologVariable;

TStdPredicate=function (VarPacket:TPrologVariables; BackTracking:Boolean):Boolean;

function VariableToStr(v:TPrologVariable;

PrintName:Boolean=True; PrintCommas:Boolean=True;

SquareBrackets:boolean=True):string;

function EqualOperator (Dst:TPrologVariable;

Src:TPrologVariable):Boolean;

function OperatorEQ (

Oper1:TPrologVariable; Oper2:TPrologVariable;

Variables:TPrologVariables):TPrologVariable;

function OperatorPlus (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function OperatorMinus (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function OperatorMultiply (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function OperatorDivide (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function OperatorGT (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function OperatorLT (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function OperatorGE (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function OperatorLE (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function OperatorNotEQ (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function OperatorAND (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function OperatorOR (

Oper1:TPrologVariable; Oper2:TPrologVariable):TPrologVariable;

function StdFunctionNot(Args:TPrologVariables):TPrologVariable;

function StdFunctionSin(Args:TPrologVariables):TPrologVariable;

function StdFunctionCos(Args:TPrologVariables):TPrologVariable;

function StdFunctionTan(Args:TPrologVariables):TPrologVariable;

function StdFunctionInt(Args:TPrologVariables):TPrologVariable;

function StdFunctionFrac(Args:TPrologVariables):TPrologVariable;

function StdFunctionSubStr(Args:TPrologVariables):TPrologVariable;

function StdFunctionFindStr(Args:TPrologVariables):TPrologVariable;

function StdFunctionChr(Args:TPrologVariables):TPrologVariable;

function StdFunctionAsc(Args:TPrologVariables):TPrologVariable;

function StdFunctionExp(Args:TPrologVariables):TPrologVariable;

function StdFunctionLn(Args:TPrologVariables):TPrologVariable;

function StdFunctionNumbToStr(Args:TPrologVariables):TPrologVariable;

function StdFunctionStrToNumb(Args:TPrologVariables):TPrologVariable;

function StdFunctionAbs(Args:TPrologVariables):TPrologVariable;

function StdPWrite(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPWriteLn(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPnl(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPIsInteger(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPIsReal(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPIsNumeric(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPIsString(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPIsBoolean(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPIsList(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPIsStruct(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPIsFree(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPReadInt(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPReadString(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPReadReal(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPDBAppendZ(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPDBAppendA(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPDBDelete(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPDBClear(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPFileOpenRead(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPFileOpenWrite(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPFileClose(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPFileRead(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPFileWrite(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPEOF(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPStringToList(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

function StdPFail(VarPacket:TPrologVariables;

BackTracking:Boolean):Boolean;

var

RunTimeError :TRuntimeError;

## Приложение 4.

Текст демонтрационной программы.

DOMAINS

ListElement:Complex {}

Integer {}

Real {}

String {}

StringList {}

ParamsList:List of ListElement {Список параметров конфигурации}

StringList:List of String {Список с запросами}

ALIAS

DBDEMOS

DATABASES

Configs:'Configs.DB' {}

Task:String[33] {}

Frequency:Integer[4] {}

Processor:String[17] {}

Memory:Integer[4] {}

VideoMemory:Integer[4] {}

HDD:Integer[4] {}

MonitorSize:Integer[4] {}

Addition1Name:String[17] {}

Addition1Value:String[9] {}

Addition2Name:String[17] {}

Addition2Value:String[9] {}

Addition3Name:String[17] {}

Addition3Value:String[9] {}

Addition4Name:String[17] {}

Addition4Value:String[9] {}

DeviceClass:'DeviceClass.db' {}

TypeName:String[17] {}

SubType:String[33] {}

SubTypeIndex:Real{}

PREDICATES

\_ReadParameters {Вспомогательный к ReadParameters}

String {Элемент списка}

StringList {Входной список}

StringList {Выходной список}

AddElementToStringList {Добавляет элемент к списку}

String {Элемент}

StringList {Входной список}

StringList {Выходной список}

AddNewAddition {Добавить в список дополнительных устройств}

StringList {список имен доп. уст-в}

StringList {список типов доп. уст-в}

String {новое имя доп. уст-ва}

String {новый тип доп. уст-ва}

StringList {вых. список имен доп. уст-в}

StringList {вых. список типов доп. уст-в}

ChooseConfig {выбор конфигурации}

StringList {список с запросами}

ParamsList {Входной список с параметрами}

ParamsList {Выходной список с параметрами}

GetListElement {Выдает по номеру элемент списка}

ParamsList {Список, в котором ищется элемент}

Integer {Номер искомого элемента}

Integer {Текущий счетчик}

ListElement {Возвращаемое значение}

Max {Выбирает максимальное значение}

ListElement {Значение 1}

ListElement {Значение 2}

ListElement {возвращаемое значение }

PrintAdditions {Печать дополнительных устройств}

StringList {Имена устройств}

StringList {Типы устройств}

ReadParameters {Читает параметры в список}

StringList {входной список}

StringList {выходной список}

run {Запускаемый предикат}

SelectProcessor {выбор процессора}

String {Процессор 1}

Integer {Частота 1}

String {Процессор 2}

Integer {Частота 2}

String {Выбранный процессор}

Integer {Выбранная частота}

GOAL

run

CLAUSES

ReadParameters(InList, OutList) if

ReadString(St),nl,

\_ReadParameters(St, InList, OutList).

\_ReadParameters("", InList, InList).

\_ReadParameters(St, InList, OutList) if

AddElementToStringList(St, InList, InList2),

ReadParameters(InList2, OutList).

AddElementToStringList(A,T,[A|T]).

GetListElement([H|\_],N,N,H).

GetListElement([\_|T],N,N1,K) if

N2=N1+1,

GetListElement(T,N,N2,K).

Max (Value1, Value2, Value1) if Value1>=Value2.

Max (Value1, Value2, Value2) if Value1<Value2.

SelectProcessor(OldProc,OldFreq,Proc1,Freq1,Proc1,OldFreq) if

DeviceClass("Processor",OldProc,OldProcNumb),

DeviceClass("Processor",Proc1,Proc1Numb),

OldProcNumb=Proc1Numb,

OldFreq>=Freq1.

SelectProcessor(OldProc,OldFreq1,Proc1,Freq1,Proc1,Freq1) if

DeviceClass("Processor",OldProc,OldProcNumb),

DeviceClass("Processor",Proc1,Proc1Numb),

OldProcNumb=Proc1Numb,

OldFreq<Freq1.

SelectProcessor(OldProc,OldFreq,Proc1,Freq1,OldProc,OldFreq) if

DeviceClass("Processor",OldProc,OldProcNumb),

DeviceClass("Processor",Proc1,Proc1Numb),

OldProcNumb>Proc1Numb.

SelectProcessor(OldProc,OldFreq,Proc1,Freq1,Proc1,Freq1) if

DeviceClass("Processor",OldProc,OldProcNumb),

DeviceClass("Processor",Proc1,Proc1Numb),

OldProcNumb<Proc1Numb.

{CreateParamsList(Freq,Proc,Mem,VMem,HDD,Monitor,Names,Vals,

[Freq,Proc,Mem,VMem,HDD,Monitor,Names,Vals]).}

AddNewAddition(N,V,"","",N,V).

AddNewAddition([],[],An,Av,[An],[Av]).

AddNewAddition([Hn|Tn],[Hv|Tv],Hn,Av,[Hn|Tn],[Hv|Tv]) if

DeviceClass(Hn,Hv,OldNumb),

DeviceClass(Hn,Av,NewNumb),

OldNumb>=NewNumb.

AddNewAddition([Hn|Tn],[Hv|Tv],Hn,Av,[Hn|Tn],[Av|Tv]) if

DeviceClass(Hn,Hv,OldNumb),

DeviceClass(Hn,Av,NewNumb),

OldNumb<NewNumb.

AddNewAddition([Hn|Tn],[Hv|Tv],An,Av,[Hn|NewN],[Hv|NewV]) if

AddNewAddition(Tn,Tv,An,Av,NewN,NewV).

ChooseConfig([],InParams,InParams).

ChooseConfig([H|T], InParams, OutParams) if

Configs(H,Freq1,Proc1,Mem1,VMem1,HDD1,Monitor1,an1,av1,an2,av2,an3,av3,an4,av4),

GetListElement(InParams,6,0,OldAddsNames),

GetListElement(InParams,7,0,OldAddsVals),

AddNewAddition(OldAddsNames,OldAddsVals,an1,av1,AddsNames1,AddsVals1),

AddNewAddition(AddsNames1,AddsVals1,an2,av2,AddsNames2,AddsVals2),

AddNewAddition(AddsNames2,AddsVals2,an3,av3,AddsNames3,AddsVals3),

AddNewAddition(AddsNames3,AddsVals3,an4,av4,AddsNames4,AddsVals4),

GetListElement(InParams,5,0,OldMonitor),

Max(Monitor1,OldMonitor,NewMonitor),

GetListElement(InParams,4,0,OldHDD),

{Max(HDD1,OldHDD,NewHDD),}

NewHDD=OldHDD+HDD1,

GetListElement(InParams,3,0,OldVMem),

Max(VMem1,OldVMem,NewVMem),

GetListElement(InParams,2,0,OldMem),

Max(Mem1,OldMem,NewMem),

GetListElement(InParams,1,0,OldProc),

GetListElement(InParams,0,0,OldFreq),

SelectProcessor(OldProc,OldFreq,Proc1,Freq1,NewProc,NewFreq),

{CreateParamsList(NewFreq,NewProc,NewMem,NewVMem,NewHDD,NewMonitor,AddsNames4,AddsVals4,InParams1),

ChooseConfig(T,InParams1,OutParams)}

ChooseConfig(T,[NewFreq,NewProc,NewMem,NewVMem,NewHDD,NewMonitor,AddsNames4,AddsVals4],OutParams).

PrintAdditions([],[]).

PrintAdditions([Hn|Tn],[Hv|Tv]) if

Write(Hn), Write(" "), WriteLn(Hv),

PrintAdditions(Tn,Tv).

run if

{ReadParameters([],A),

WriteLn(A),}

ChooseConfig(["Internet","сочинение музыки","Delphi 3"],[0,"86",0,0,0,0,[],[]],B),

{WriteLn(B),}

GetListElement(B,0,0,Freq),

GetListElement(B,1,0,Proc),

WriteLn("Процессор: ",Proc," ",Freq," MHz"),

GetListElement(B,2,0,Mem),

WriteLn("Память: ",Mem," МБайт"),

GetListElement(B,3,0,VMem),

WriteLn("Видео память: ",VMem," МБайт"),

GetListElement(B,4,0,HDD),

WriteLn("Винчестер: ",HDD," МБайт"),

GetListElement(B,5,0,Monitor),

WriteLn("Монитор: ",Monitor,""""),

GetListElement(B,6,0,Names),

GetListElement(B,7,0,Vals),

PrintAdditions(Names,Vals).