

Лабораторный практикум в средах Visual Prolog и DrRacket

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	2
1. ПРОЛОГ – язык логического программирования	4
1.1. Создание баз знаний в ПРОЛОГе	4
1.2 Поиск с возвратом. Управление поиском	16
1.3 Решение логических задач в ПРОЛОГе	27
1.4 Арифметические вычисления и рекурсия в ПРОЛОГе	31
1.5 Списки	35
1.6 Создание экспертных систем средствами ПРОЛОГа	40
2. ОСНОВЫ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ.....	50
2.1. Основы работы в среде DrRacket	50
2.2. Управляющие конструкции в Racket.....	57
2.3.Реализация рекурсии в Racket	64
2.4. Списки и атомы	66
Литература	75

ВВЕДЕНИЕ

Языки функционального и логического программирования являются представителями декларативной парадигмы программирования. В декларативных языках, в отличие от процедурных, описывается не алгоритм решения задачи, а знания об объектах и процессах соответствующей предметной области. Поиск решения задачи вырабатывается и реализуется в ходе логического вывода с помощью специального механизма. Знание общих концепций и методов современного декларативного программирования и, в частности, таких его разновидностей, как функциональное и логическое программирование, позволяет эффективно решать задачи, связанные с обработкой символьной информации, построением систем поддержки принятия решений, экспертных систем.

Основоположником логического направления программирования является язык ПРОЛОГ, а функционального - ЛИСП. В настоящее время эти и другие языки, поддерживающие и развивающие декларативную парадигму программирования, развиваются и становятся все более универсальными.

Предлагаемое методическое пособие представляет собой описание лабораторных работ по изучению функционального и логического программирования. Пособие состоит из двух частей: в первой рассматривается решение типовых задач на ПРОЛОГе в среде Visual Prolog, во второй части приведены примеры разработки программ на языке DrRacket, который является одним из современных диалектов ЛИСП.

Основная цель пособия – помощь студентам в подготовке к выполнению лабораторных работ.

Целью выполнения данных лабораторных работ является приобретение студентами навыков работы в средах Visual Prolog и DrRacket (формальная постановка задачи, преобразование в форму, пригодную для программирования, трассировка и оценка результатов), а также закрепление теоретических знаний, полученных при прослушивании курса лекций.

Пособие содержит необходимый теоретический материал, примеры решения задач и задания по программированию на языках Пролог и DrRacket. К каждому разделу приводятся несколько типовых задач с решениями, что

позволяет студентам самостоятельно подготовиться к выполнению лабораторной работы.

В качестве результата выполнения лабораторной работы студентом должен быть подготовлен отчет о проделанной работе, который включает:

1. Постановку задачи
2. Входные/Выходные данные (при необходимости)
3. Текст программы (с необходимыми комментариями)
4. Результаты тестирования (наборы входных и выходных данных).

В настоящем пособии приводится список литературы, которая может быть использована при подготовке к лабораторным работам.

Данное пособие может использоваться студентами при изучении дисциплин: Функциональное и логическое программирование, Интеллектуальные информационные системы, Основы логического программирования.

1. ПРОЛОГ – язык логического программирования

1.1. Создание баз знаний в ПРОЛОГе

Краткие теоретические сведения

ПРОЛОГ (ПРОграммирование в ЛОГике) - язык логического программирования, предназначен для решения задач из области искусственного интеллекта. Он используется для обработки естественного языка и разработке систем, основанных на знаниях. Методы поиска, используемые в нем, принципиально отличаются от традиционных. Вместо детальных инструкций, предписывающих как решать ту или иную задачу, программист на языке Prolog уделяет основное внимание описанию задачи.

Основные конструкции ПРОЛОГа заимствованы из логики. ПРОЛОГ относится не к процедурным, а к декларативным языкам программирования. Он ориентирован не на разработку решений, а на систематизированное и формализованное описание задачи с тем, чтобы решение следовало из составленного описания.

Современной средой, в основе которой лежит язык Пролог, является Visual Prolog. В среде Visual Prolog используется подход, получивший название «визуальное программирование», при котором внешний вид и поведение программ определяются с помощью специальных графических средств проектирования без традиционного программирования на алгоритмическом языке.

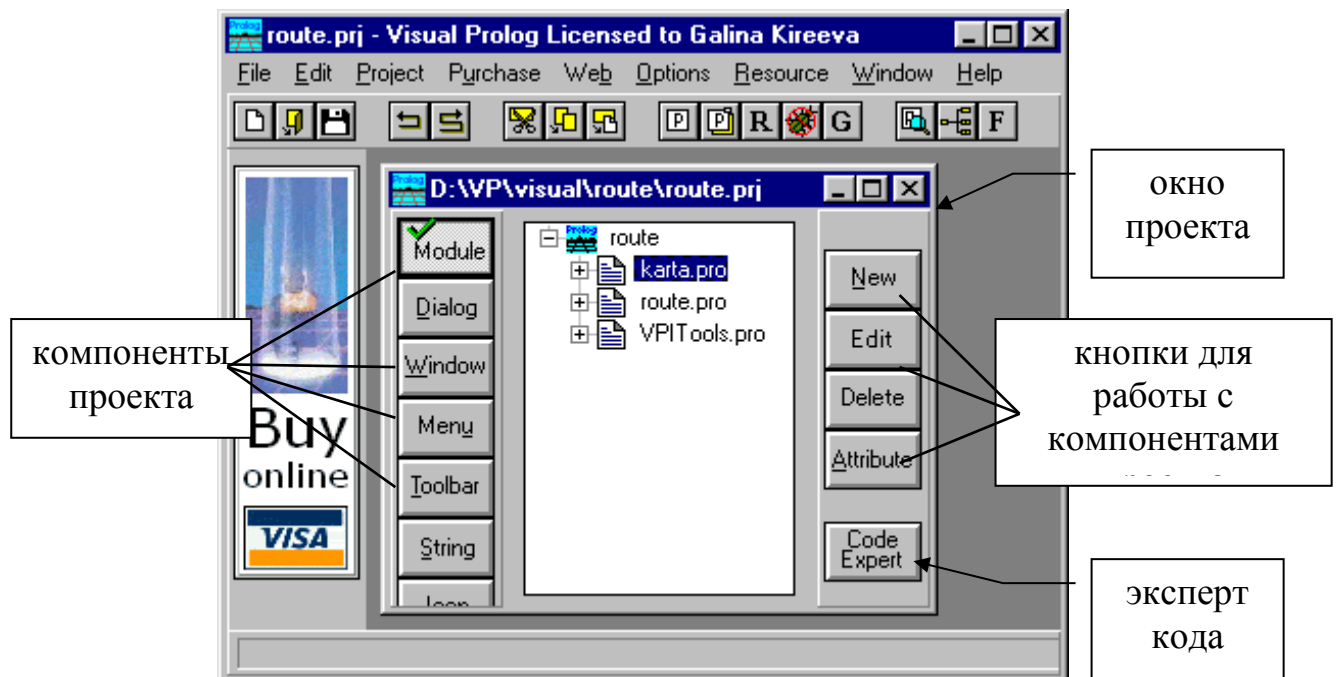


рис.1.1 Среда разработки Visual Prolog 6.1

Интерфейс Visual Prolog включает: главное меню, панель инструментов, окно проекта. Если во время последнего использования системы Visual Prolog там был открытый проект, то система автоматически вновь откроет этот проект.

На рис.1.1 изображен внешний вид среды Visual Prolog после запуска. В окне проекта отображаются модули открытого проекта route.prj: karta.pro, route.pro, VPITools.pro.

Левая панель кнопок в окне проекта позволяет выбирать нужный компонент проекта: модуль, окно, меню и т.д. С помощью кнопок правой панели выбранный компонент можно редактировать(кнопка Edit), удалять(кнопка Delete), а также добавлять новый(кнопка New).

Пункт меню File содержит команды для работы с файлами. Чтобы создавать новое окно редактирования, можно использовать команду File | New. Эта команда создаст новое окно редактора с заголовком "NONAME".

В меню Edit представлены команды, позволяющие редактировать текст программы. Встроенный редактор системы по интерфейсу похож на обычный текстовый редактор. Можно производить вырезку, копирование и вставку текста, операции Отмена/Восстановление, которые можно активизировать из меню Edit. Также меню Edit показывает "горячие клавиши", связанные для этих действий.

Пункт меню **Project** содержит команды для работы с проектом: создать новый, открыть, запустить и т.д. Запуск проекта на исполнение выполняется нажатием кнопки <R> на панели инструментов (или F9, или с помощью команд меню Project | Run).

Команды меню **Options** позволяют выполнять настройку проекта, устанавливать необходимые параметры.

Среда Visual Prolog позволяет протестировать программу без создания проекта. Для этого используется утилита Test Goal. Достаточно создать новый файл, набрать текст программы и активизировать Test Goal нажатием кнопки <G> на панели инструментов. Автономно исполняемый файл при этом не создается. Утилита Test Goal компилирует только тот код, который определен в активном окне редактора (код в других открытых окнах или модулях проектов, если они есть, игнорируются). Test Goal находит *все* возможные решения задачи и автоматически выводит значения *всех* переменных.

Программа на ПРОЛОГе состоит из предложений, которые могут быть фактами, правилами или запросами. Факт – это утверждение о том, что соблюдается некоторое конкретное соотношение между объектами. Факт используется для того, чтобы показать простую взаимосвязь между данными.

Структура факта:

<имя_отношения>(t1,t2,...,tn)), t1,t2,...,tn– объекты

Примеры фактов:

учится (ира, университет).

% Ира учится в университете

родитель(иван, алексей).

% Иван является родителем Алексея

язык_программирования (пролог).

% Пролог – это язык программирования

Набор фактов составляет базу данных. В виде факта в программе записываются данные, которые принимаются за истину и не требуют доказательства.

Правила используются для того, чтобы установить отношения между объектами на основе имеющихся фактов.

Структура правила:

<имя_правила> :- <тело правила> или

<имя_правила> if <тело правила>

Левая часть правила вывода называется головой правила, а правая часть - телом. Тело может состоять из нескольких условий, перечисленных через запятую или точку с запятой. Запятая означает операцию «логическое И», точка с запятой – операцию «логическое ИЛИ». В предложениях используются переменные для обобщенной формулировки правил вывода. Переменные действуют только в одном предложении. Имя в разных предложениях указывает на разные объекты. Все предложения обязательно заканчиваются точкой.

Примеры правил:

мать (X, Y) :- родитель (X, Y), женщина(X).

студент (X) :- учится (X, институт); учится (X, университет).

Правило отличается от факта тем, что факт - всегда истина, а правило является истинным, если выполняются все утверждения, составляющие тело правила. Факты и правила образуют базу знаний.

Переменные служат для обозначения объектов, значения которых меняются в ходе выполнения программы. Имена переменных начинаются с заглавных букв или знака «_» Область действия переменной – предложение. Одноименные переменные в разных предложениях могут иметь разные значения.

Специальным знаком «_» обозначается анонимная переменная, которая используется тогда, когда конкретное значение переменной не существенно для данного предложения. Значение анонимной переменной не выводится на печать.

Если имеется база данных, то можно написать запрос (цель) к ней. Запрос — это формулировка задачи, которую программа должна решить. Его структура такая же, как у правила или факта. Существуют запросы с константами и запросы с переменными.

Запросы с константами позволяют получить один из двух ответов: “да” или “нет”. Если в запрос входит переменная, то интерпретатор пытается найти такие ее значения, при которых запрос будет истинным. Запросы могут быть составными, т.е. состоять из нескольких простых запросов. Они могут объединяться знаком ‘,’ , который понимается как логическая связка И или знаком ‘;’(логическое ИЛИ).

Простые запросы называются подцелью, составной запрос принимает истинное значение тогда, когда истинна каждая подцель.

Как правило, программа состоит из четырех разделов.

DOMAINS – секция описания доменов(типов). Секция применяется, если в программе используются нестандартные домены.

PREDICATES – секция описания предикатов. Секция применяется, если в программе используются нестандартные предикаты.

CLAUSES – секция предложений. Именно в этой секции записываются предложения: факты и правила вывода.

GOAL – секция цели. В этой секции записывается запрос.

Пример 1.

Имеется база данных, содержащая следующие факты:

родитель(илья, марина).
родитель(марина, ира).
родитель(елена, иван).
родитель(николай, ира).
родитель(ольга, алексей).
родитель(марина, саша).
родитель(сергей, иван).

Определить:

- 1) верно ли, что Марина является родителем Саши;
- 2) верно ли, что Алексей является родителем Ольги;
- 3) кто является ребенком Николая;
- 4) кто родители Ивана;
- 5) всех родителей и их детей.

Решение.

1. Запустите среду Visual Prolog. Закройте окно проекта (если оно открыто) и откройте новый файл (**File|New**) (рис.2)

В появившемся окне наберите текст программы, содержащий разделы: PREDICATES (описание предиката *родитель*), CLAUSES (перечисляются имеющиеся факты) и GOAL (запрос).

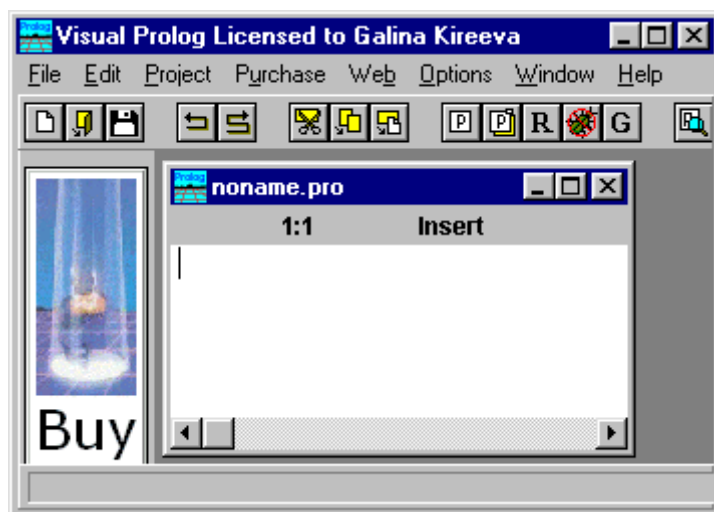


рис.2. Рабочее окно редактора

```

DOMAINS
    имя=symbol
PREDICATES
    nondeterm    родитель (имя, имя)
CLAUSES
    родитель (илья, марина) .
    родитель (марина, ира) .
    родитель (елена, иван) .
    родитель (николай, ира) .
    родитель (ольга, алексей) .
    родитель (марина, саша) .
    родитель (сергей, иван) .
GOAL
    родитель (марина, саша) .

```

Запустите и протестируйте программу с помощью команды **Project | Test Goal** (можно использовать кнопку на панели инструментов <G> или сочетание клавиш <Ctrl>+<G>). Результат выполнения программы будет выведен в отдельном окне



рис3. Окно вывода результата

Указание: перед следующим запуском программы следует закрыть это окно.

2. Для ответа на вопрос: верно ли, что Алексей является родителем Ольги, измените запрос:

```

GOAL
    родитель (алексей, ольга) .

```

После запуска программы (Project | Test Goal) будет получен ответ:

no

3. Для ответа на вопрос: кто является ребенком Николая, запишите цель:

```

GOAL
    родитель (николай, X) .

```

Результат:

```

X=ира
1 Solution

```

4. Для ответа на вопрос: кто родители Ивана, укажите запрос:

```

GOAL

```


родитель (X, иван) , родитель (Y, иван) , $X \neq Y$.

Результат:

X=елена, Y=сергей

X=сергей, Y=елена

2 Solutions

5. Для определения всех родителей и их детей, запишите:

GOAL

родитель (X, Y) .

Результат:

X=илья, Y=марина

X=марина, Y=ира

X=елена, Y=иван

X=николай, Y=ира

X=ольга, Y=алексей

X=марина, Y=саша

X=сергей, Y=иван

7 Solutions

Пример 2

Имеются факты вида: *родитель(имя, имя)* и *женщина(имя)*.

а) составить правило *мать* и определить, кто мать Маши.

б) составить правило *бабушка* и определить, кто бабушка Ирины.

в) составить правило *внучка* и определить, сколько внучек у Ольги и

как их зовут

Решение:

DOMAINS

имя= symbol

PREDICATES

nondeterm родитель (имя, имя)

женщина (имя)

nondeterm мать (имя, имя)

nondeterm бабушка (имя, имя)

nondeterm внучка (имя, имя)

CLAUSES

родитель (марина, ирина) .

родитель (елена, анна) .

родитель (ольга, марина) .

родитель (ольга, татьяна) .

родитель (татьяна, катя) .

родитель (анна, маша) .

женщина (ольга) .
женщина (маша) .
женщина (ирина) .
женщина (елена) .
женщина (анна) .
женщина (марина) .
женщина (татьяна) .
женщина (катя) .
мать (X, Y) : -родитель (X, Y) , женщина (X) .
бабушка (X, Z) : -мать (X, Y) , родитель (Y, Z) .
внучка (X, Y) : -бабушка (Y, X) , женщина (X) .

А) GOAL

мать (Кто, маша) .

Результат:

Кто=анна
1 Solution

Б) GOAL

бабушка (Кто, ирина) .

Результат:

Кто=ольга
1 Solution

В) GOAL

внучка (Кто, ольга) .

Результат:

Кто=ирина
Кто=катя
2 Solutions

Замечание: ключевое слово *nondeterm* определяет недетерминированные предикаты, которые могут совершать откат назад и генерировать множественные решения. Таким образом, если задача предполагает возможность получения несколько решений, следует объявлять предикаты как недетерминированные.

Пример 3

Записать по правилам Пролога следующие факты:

Билл играет в теннис, баскетбол и футбол

Майк играет в футбол и хоккей.

Сформулировать запросы, выясняющие:

а) кто играет в футбол

- б) во что играет Майк
в) во что играют и Билл, и Майк

Решение:

DOMAINS

имя, спорт=symbol

PREDICATES

play(имя, спорт)

CLAUSES

play(билл, теннис) .

play(билл, баскетбол) .

play(билл, футбол) .

play(майк, хоккей) .

play(майк, футбол) .

А) GOAL

play(X, футбол) .

Результат:

X=билл

X=майк

2 Solutions

Б) GOAL

play(майк, X) .

Результат:

X=хоккей

X=футбол

2 Solutions

В) GOAL

play(майк, X) , play(билл, X) .

Результат:

X=футбол

1 Solution

Создание проекта позволяет протестировать пример как автономную исполняемую программу. После запуска проекта на исполнение создается ехе-файл, работа которого завершается после *первого* решения, удовлетворяющего решению задачи. Запуск программы в этом режиме не обеспечивает автоматический вывод значений переменных, поэтому необходимо использовать стандартный предикат вывода **write**.

Пример 4.

Заданы отношения-факты:

родитель("Иван","Катя").
родитель("Анна","Олег").
родитель("Олег","Дима").
родитель("Игорь","Ольга").
родитель("Олег","Виктор").
родитель("Игорь","Иван").
мужчина("Дима").
мужчина("Иван").
мужчина("Игорь").
мужчина("Олег").
мужчина("Виктор").
женщина("Катя").
женщина("Ольга").
женщина("Анна").

Составить новое отношение-правило $ded(X,Y)$ и определить, кто является дедушкой Кати. Создать проект и протестировать пример как автономную исполняемую программу.

Решение

1. Запустите среду Visual Prolog и создайте новый проект (Project | New Project), активизируется окно **Application Expert** (эксперт приложения).
2. Определите имя проекта (Primer) и базовый каталог, куда будет сохранен проект (например, D:\VP\Primer)

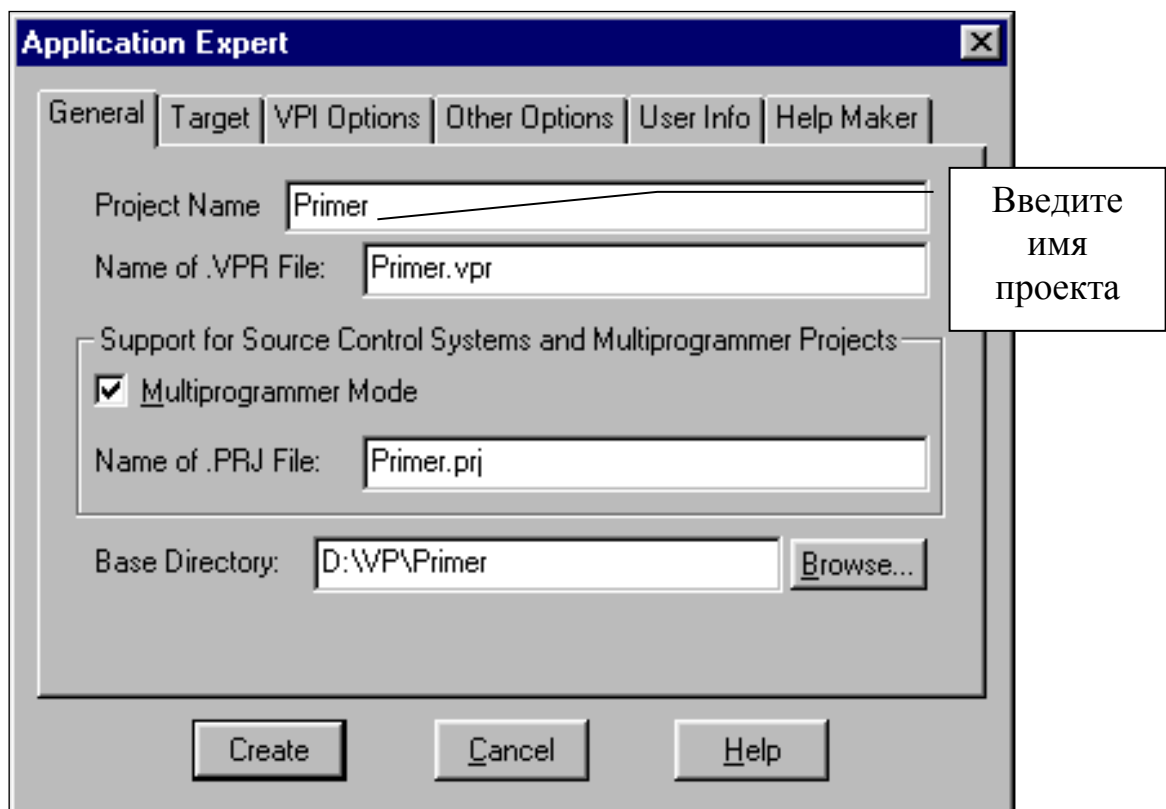


рис.4. Окно **Application Expert**

На вкладке **Target** установите параметры и нажмите кнопку **Create** для создания проекта (рис. 5):

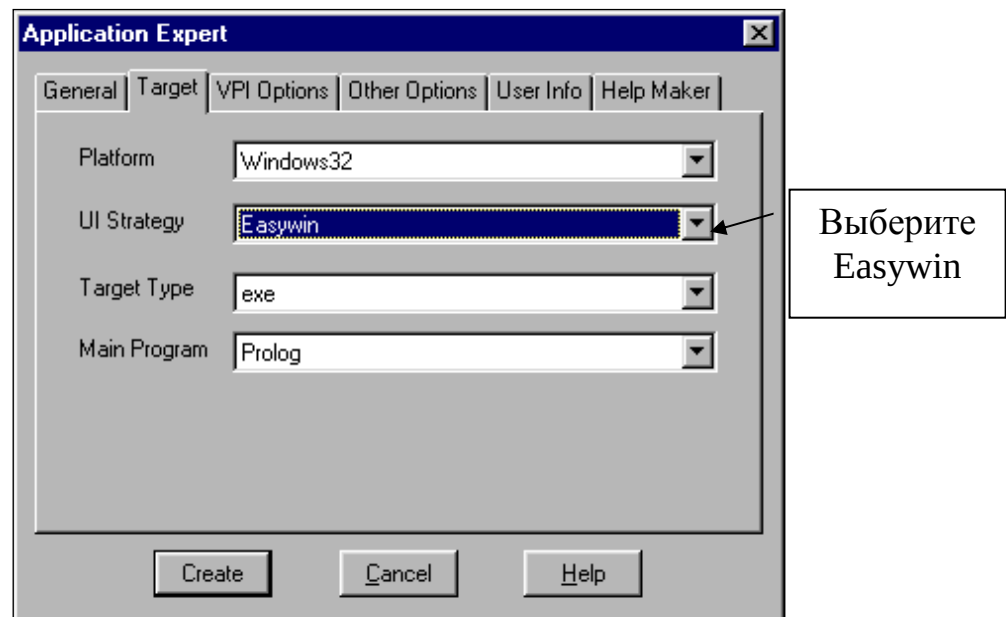


рис.5. Установки на вкладке **Target** окна **Application Expert**

3. Откройте окно **Compiler Options** (Options | Project | Compiler Options), откройте вкладку **Warnings** и установите опции компилятора для созданного проекта (рис.6):

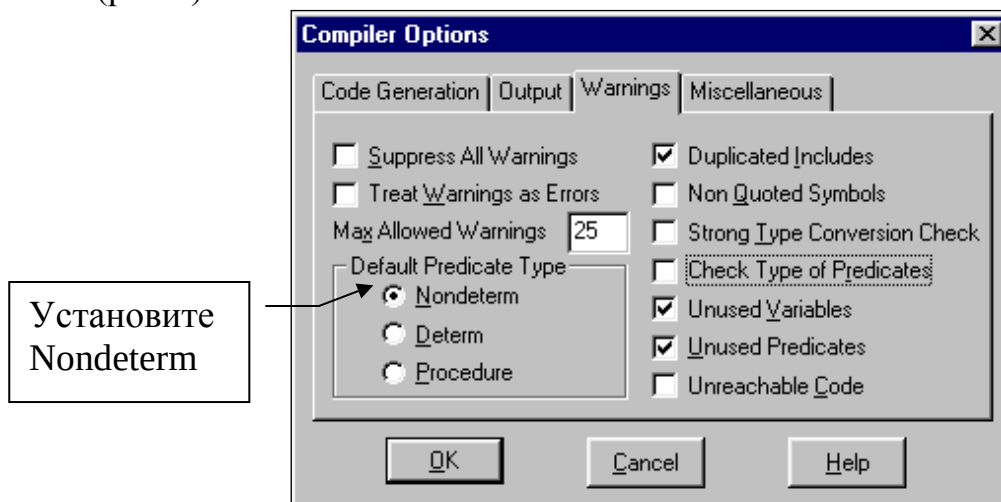


рис.6. Установки опций компилятора

Нажмите OK.

4. В окне проекта выделите файл **Primer.pro** и откройте его для редактирования (двойной щелчок или кнопка **Edit**)



рис.7. Окно проекта

Файл с расширением .pro содержит секции PREDICATES, GOAL, CLAUSES. Допишите необходимые определения так, чтобы получилась программа:

DOMAINS

имя=string

PREDICATES

родитель (имя, имя)

женщина (имя)

мужчина (имя)

дед (имя, имя)

CLAUSES

родитель ("Иван", "Катя") .

родитель ("Анна", "Олег") .

родитель ("Олег", "Дима") .

родитель ("Игорь", "Ольга") .

родитель ("Олег", "Виктор") .

родитель ("Игорь", "Иван") .

мужчина ("Дима") .

мужчина ("Иван") .

мужчина ("Игорь") .

мужчина ("Олег") .

мужчина ("Виктор") .

женщина ("Катя") .

женщина ("Ольга") .

женщина ("Анна") .

дед (X, Z) :-родитель (X, Y) , родитель (Y, Z) ,
мужчина (X) .

GOAL

дед (X, "Катя") , write (X) .

5. Откомпилируйте исходный код примера и запустите его как автономную исполняемую программу. (Project | Run, или клавиша <F9>, или кнопка <R>). Результат выполнения программы должен отобразиться в окне:

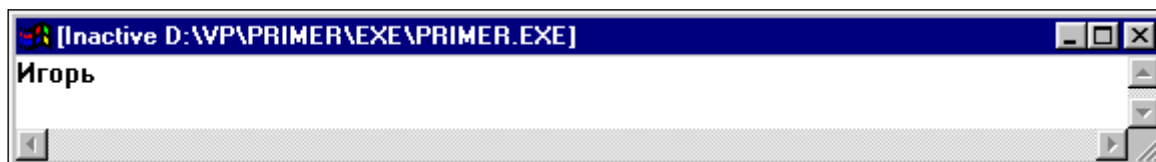


рис.8. Окно вывода результата

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Запишите по правилам Пролога следующие факты:

Ник подарил Тому книгу.

Мэри подарила Тому ручку.

Рик подарил Мэри игрушку.

Боб подарил Пэт игрушку.

Сформулируйте запросы, выясняющие:

- Правда ли, что Рик подарил Мэри игрушку?
- Правда ли, что Ник подарил Тому игрушку?
- Что подарила на день рождения Тому Мэри?
- Что подарили Тому на день рождения?
- Кто подарил Пэт игрушку?

2. Задано дерево родственных отношений:



Запишите данные факты по правилам ПРОЛОГа и сформулируйте запросы, выясняющие:

- Является ли Федор родителем Ольги?
- Кто является родителем Татьяны?
- Кто дети Ивана?
- Кто является родителем родителя Ольги?
- Кто внуки Ивана?
- Есть ли у Федора и Степана общий родитель?
- Кто чей родитель?

3. Имеется база данных, содержащая факты вида: любит(имя, продукт), фрукты(продукт), конфеты(продукт).

Составьте программу, определяющую:

- a) всех, кто любит бананы;
- b) кто любит и шоколад, и яблоки;
- c) что любит Вова;
- d) что любят и Света, и Вова.

Используя имеющиеся факты, составить новое правило **люб_фрукты(X)** и определить всех, кто любит фрукты;

4. Имеется база данных, содержащая факты вида: играет(имя, спорт), мужчина(имя), женщина(имя).

Составьте программу, определяющую:

- a) каким видом спорта увлекается Андрей;
- b) всех, кто играет в волейбол;
- c) каким видом спорта увлекаются и Ольга, и Саша;
- d) кто увлекается и футболом, и волейболом.

Используя имеющиеся факты, составить новое правило **волейбол_жен(X)** и определить всех женщин, играющих в волейбол.

5. Реализуйте проект примера 4 и доработайте исходный код следующим образом:

- 1) добавьте новое правило **бабушка** и определите, кто является бабушкой;
- 2) добавьте новое правило **внук** и определите, кто внук Анны;
- 3) добавьте новое правило **брат** и определите, кто брат Димы;
- 4) добавьте новое правило **сестра** и определите, кто сестра Ивана.

1.2 Поиск с возвратом. Управление поиском

Краткие теоретические сведения

В отличие от процедурных языков, в которых программист должен точно описать процесс решения задачи, Пролог является описательным (декларативным) языком. Программа на Прологе содержит описание проблемы, для этого используются правила и факты, и цель. Выполнение программы заключается в попытке доказать целевое утверждение, используя предположения, заданные в программе. Механизм поиска решений “встроен” в Пролог, он называется “поиск с возвратом”.

Поиск с возвратом (backtracking) – это один из основных приемов поиска решений поставленной задачи в ПРОЛОГ’е. Выполняя поиск, ПРОЛОГ может столкнуться с необходимостью выбора между альтернативными путями. Тогда он ставит маркер у места развилки (точка отката) и выбирает первую подцель. Если она не выполняется, то ПРОЛОГ возвращается в точку отката и переходит к следующей подцели.

Пример 1.

Рассмотрим программу, которая содержит сведения об именах и возрасте нескольких игроков в теннисном клубе. Цель состоит в подборе пары игроков, возраст которых 9 лет.

Решение:

```
DOMAINS
    child = symbol
    age   = integer
PREDICATES
    player(child, age) /*игрок(ребенок, возраст)*/
CLAUSES
    player(peter, 9).
    player(paul, 10).
    player(chris, 9).
    player(susan, 9).
Goal
    player(Person1, 9), player(Person2, 9), Person1<>Person2,
    write(Person1, "- ", Person2).
```

На первом шаге ПРОЛОГ пытается найти решение для первой подцели `player(Person1,9)`. Эта подцель согласуется путем сопоставления `Person1` с `peter`, так как Пролог просматривает утверждения для согласования сверху вниз.

После достижения первой подцели ПРОЛОГ переходит ко второй подцели: `player(Person2,9)`. Эта подцель опять согласуется при сопоставлении `Person2` с `peter`.

На следующем шаге ПРОЛОГ переходит к согласованию третьей подцели: `Person1<>Person2`. Так как `Person1` и `Person2` имеют значения `peter`, то данная подцель не согласуется, и Пролог осуществляет откат к предыдущей подцели `player(Person2,9)`. Эта подцель сопоставляется с фактом `player(chris,9)`, и `Person2` согласуется с `chris`. Теперь Пролог опять в качестве текущей подцели выполняет цель `Person1<>Person2`. Эта подцель успешна, так как `peter` и `chris` отличны. Таким образом, ПРОЛОГ согласовал все подцели целевого утверждения, и цель считается согласованной. Получена пара игроков: Peter-Chris.

Результат выполнения программы:

Peter - chris

Среда Visual Prolog позволяет использовать отладчик для пошагового выполнения программы. Отладчик работает с откомпилированным кодом. В исходном коде можно ставить точки останова и выполнять программу по шагам. В режиме пошагового выполнения программы можно просматривать текущие значения переменных и содержимое утвержденных фактов.

Пример 2.

Имеется база данных, содержащая факты вида *отдыхает(имя, город), украина(город), россия(город), прибалтика(город)*. Составить правило, позволяющее определить, кто отдыхал в России.

Проследить поиск решения задачи с помощью отладчика Visual Prolog и построить целевое дерево поиска с возвратом.

Решение:

1. Создайте новый проект (Project | New Project) и наберите текст программы:

```
DOMAINS
```

```
имя, город= symbol
```

```
PREDICATES
```

```
отдыхает(имя, город)
```

```
украина(город) .
```

```
россия(город) .
```

```
прибалтика(город) .
```

```
отдых_Россия(имя)
```

```
CLAUSES
```

```
отдыхает(sasha, antalia) .
```

```
отдыхает(anna, sochi) .
```

```
отдыхает(dima, urmala) .
```

```
отдыхает(oleg, kiev) .
```

```
украина(kiev) .
```

```
россия(sochi) .
```

```
прибалтика(urmala) .
```

```
отдых_Россия(X) :-
```

```
отдыхает(X, Y) ,
```

```
россия(Y) .
```

```
GOAL
```

```
отдых_Россия(X) ,
```

```
write(X) , nl.
```

3. Сохраните проект (**Project | Save Project**)

4. Запустите его на исполнение (**Project | Run**, или клавиша <F9>, или кнопка <R>). Результат выполнения программы:

```
anna
```

5. Проследите поиск этого решения с помощью отладчика(Debugger). Для этого:

а) запустите отладчик (**Project | Debug**);

б) в окне отладчика выберите команду **View | Local Variables** (для просмотра текущих значений переменных);

в) нажимайте клавишу <F7> (или **Run | Trace Into**) для пошагового выполнения программы, текущие значения переменных отображаются в окне Variables For Current Clause



окно, отображающее
текущие значения
переменных

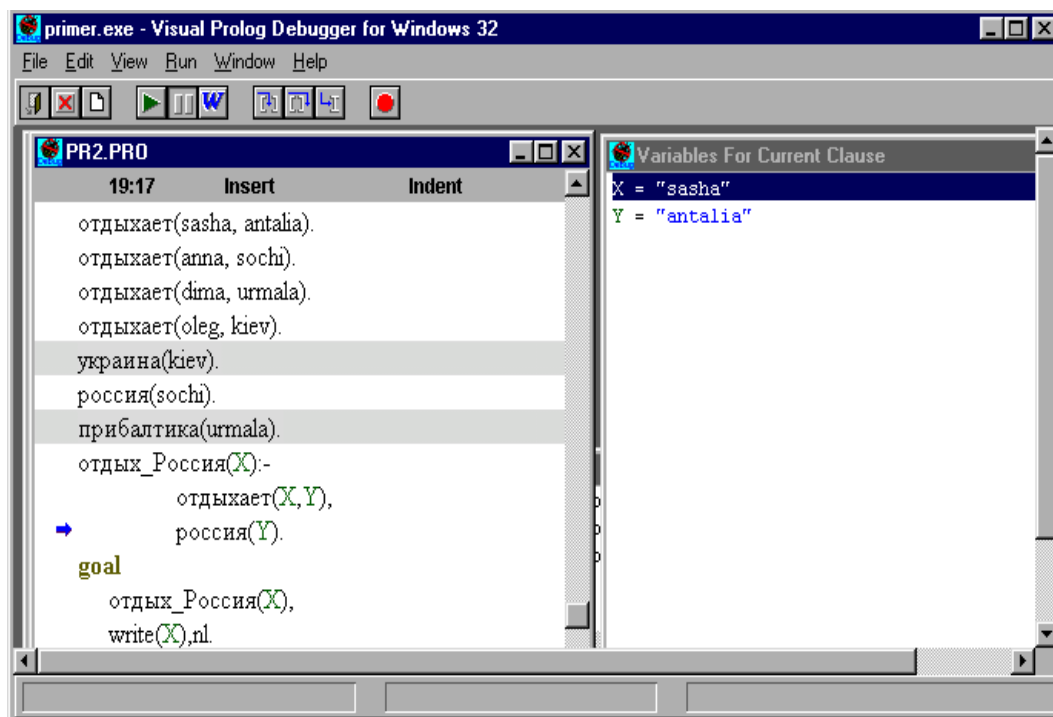


рис.9. Окно отладчика

Поиск решения можно представить следующим образом:

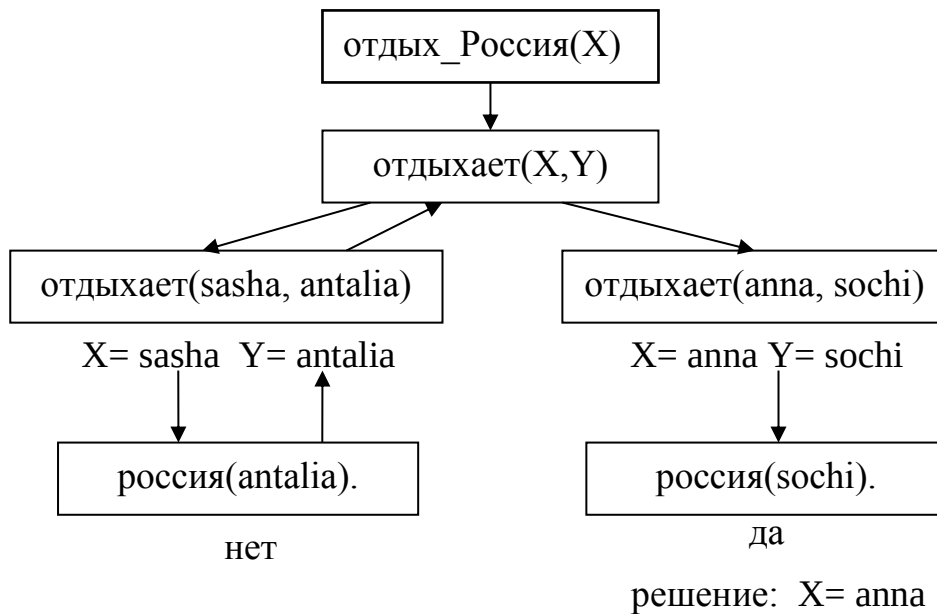


рис.10. Целевое дерево поиска решения

Управление поиском с возвратом заключается в решении двух задач: включении поиска с возвратом при необходимости найти все возможные решения задачи и отключении поиска с возвратом для ограничения пространства поиска.

Для решения этих задач используются два стандартных предиката: fail и отсечение.

fail – это тождественно-ложный предикат, искусственно создающий ситуацию неуспеха. После выполнения этого предиката управление передается в точку отката и поиск продолжается.

Чтобы прервать поиск решений при выполнении какого-либо условия, используется предикат **отсечения** (обозначается !), Однажды пройдя через отсечение, невозможно вернуться назад, т.к. этот предикат является тождественно-истинным. Процесс может только перейти к следующей подцели, если такая имеется.

Например, $p :- p1, p2, !, p3$.

Если достигнуты цели $p1$ и $p2$, то возврат к ним для поиска новых решений невозможен.

Пример 3

База данных содержит факты вида: **student(имя, курс)**. Создать проект, позволяющий сформировать список студентов 1-го курса.

Решение:

```

PREDICATES
    student(symbol, integer)
    spisok
  
```

```

CLAUSES
    student(vova,3).
    student(lena,1).
    student(dima,1).
    student(ira,2).
    student(marina,1).
    spisok:-student(X,1),write(X),nl,fail.
GOAL
    write("Список студентов 1-курса"),nl,spisok.

```

Результат выполнения программы:

```

Список студентов 1-курса
lena
dima
marina

```

Пример 4

База данных содержит факты вида **father(name, name)**. Создать проект, позволяющий определить кто чей отец.

Решение:

```

DOMAINS
    name=symbol
PREDICATES
    father (name, name)
    everybody
CLAUSES
    father ("Павел", "Петр").
    father ("Петр", "Михаил").
    father ("Петр", "Иван").
    everybody:- father (X, Y),
                write(X, " - это отец",Y,"a"),nl,
                fail.
GOAL
    everybody.

```

Результат выполнения программы:

```

Павел - это отец Петра
Петр - это отец Михаила
Петр - это отец Ивана

```

Пример 5

Создать проект, реализующий железнодорожный справочник. В справочнике содержится следующая информация о каждом поезде: номер поезда, пункт назначения и время отправления.

а) вывести всю информацию из справочника.

Решение:

```
DOMAINS
    nom=integer
    p, t=string
PREDICATES
    poezd(nom,p,t)
CLAUSES
    poezd(233,moskva,"12-30").
    poezd(257,moskva,"22-40").
    poezd(133,armavir,"10-20").
    poezd(353,armavir,"20-40").
    poezd(353,adler,"02-30").
    poezd(413,adler,"11-10").
    poezd(256,piter,"21-30").
GOAL
    write("      Расписание поездов"), nl,
    write("Номер Пункт прибытия Время отправления"),
    nl, poezd(N,P,T), write(N," ",P," ",T),nl,fail.
```

Результат выполнения программы

Номер	Пункт прибытия	Время отправления
233	moskva	12-30
257	moskva	22-40
133	armavir	10-20
353	armavir	20-40
353	adler	02-30
413	adler	11-10
256	piter	21-30

б) организовать поиск поезда по пункту назначения.

Решение:

```
GOAL
    write (" Пункт назначения:"), Readln(P), nl,
    write ("Номер      Время отправления"), nl,
    poezd(N,P,T), write(N,"      ",T), nl,
fail.
```

Комментарий: Readln –стандартный предикат ввода строкового значения

Результат выполнения программы

Пункт назначения:armavir

Номер	Время отправления
133	10-20
353	20-40

в) вывести информацию о поездах, отправляющихся в заданный временной промежуток

Решение:

GOAL

```

write(" Время отправления:"),nl,
write("с..."), Readln(T1),
write("до..."), Readln(T2), nl,
write("Номер Пункт назначения Время отправления"),
nl,poezd(N,P,T),T>=T1,T<=T2,write(N," ",P," ", T),
nl, fail.

```

Результат выполнения программы

```

Время отправления:
с...10-00
до...15-00

```

Номер	Пункт назначения	Время отправления
233	moskva	12-30
133	armavir	10-20
413	adler	11-10

Пример 6

Имеется база данных, содержащая данные о спортсменах: имя и вид спорта. Определить возможные пары одного из спортсменов-теннисистов с другими теннисистами.

Решение:

DOMAINS

```

имя, вид_сп=string

```

PREDICATES

```

играет(имя, вид_сп)
спис_спортс

```

CLAUSES

```

играет("Саша", теннис) .
играет("Аня", волейбол) .
играет("Олег", футбол) .
играет("Коля", теннис) .
играет("Саша", футбол) .
играет("Андрей", теннис) .
спис_спортс:- играет(X, теннис) , !, играет(Y, теннис) ,
X<>Y, write(X, "-", Y), nl, fail.

```

GOAL

```
write("Пары теннисистов"),nl,  
спис_спортс.
```

Результат выполнения программы:

```
Пары теннисистов  
Саша-Коля  
Саша-Андрей
```

Пример 7

Студенту в зависимости от набранной в процессе обучения суммы баллов **Z** присваивается квалификация:

```
магистр (M),    если  $80 \leq Z \leq 100$   
специалист (S), если  $60 \leq Z < 80$   
бакалавр (B),   если  $40 \leq Z < 60$   
неудачник (N),  если  $0 \leq Z < 40$ 
```

Составить программу, которая определит квалификацию в зависимости от введенного значения **Z**

Решение:

Для решения задачи составим правило **grade**, устанавливающее связь между количеством баллов (*z*) и квалификацией (*r*). Правило состоит из нескольких частей. Первые две части обеспечивают проверку недопустимых значений **Z** с выводом соответствующего сообщения. Остальные части правила определяют квалификацию в зависимости от значения **Z**.

```
DOMAINS  
    z=integer  
    r=string  
PREDICATES  
    grade(z,r)  
CLAUSES  
    grade(Z,""):-Z<0,! , write("Неверный ввод данных!") .  
    grade(Z,""):-Z>100,! ,write("Неверный ввод данных!") .  
    grade(Z,"M"):-Z>=80,! .  
    grade(Z,"S"):-Z>=60,! .  
    grade(Z,"B"):-Z>=40,! .  
    grade(Z,"N") .  
GOAL  
    write("Z="), readint(Z), grade(Z,R),write(R) .
```

Комментарий: readint – стандартный предикат ввода целочисленного значения

Результат выполнения программы:

1-й случай:

```
Z=88  
M
```


2-й случай:

Z=65

S

3-й случай:

Z=39

N

4-й случай:

Z=110

Неверный ввод данных!

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. База данных содержит следующие факты:

увлекается(“Коля”, гитара).

увлекается(“Оля”, скрипка).

увлекается(“Дима”, плавание).

увлекается(“Таня”, теннис).

спорт(плавание).

спорт(теннис).

муз_инстр(скрипка).

муз_инстр(гитара).

а) составить правило **спортсмен** и определить, кто увлекается спортом;

б) проследить за поиском решения с помощью отладчика;

в) построить целевое дерево поиска с возвратом.

2. База данных содержит следующие факты:

увлекается(“Дима”, плавание).

увлекается(“Таня”, теннис).

увлекается(“Коля”, гитара).

увлекается(“Оля”, скрипка).

спорт(плавание).

спорт(теннис).

муз_инстр(скрипка).

муз_инстр(гитара).

а) составить правило **музыкант** и определить, кто увлекается музыкой;

б) проследить за поиском решения с помощью отладчика;

в) построить целевое дерево поиска с возвратом.

3. Составить программу, реализующую справочник товаров. В справочнике содержится следующая информация о каждой товаре: вид товара (промышленный или продуктовый), название, цена, количество, страна-производитель. Вывести:

а) всю информацию из справочника;

- б) информацию о товарах, количество которых не превышает заданное значение;
 - в) информацию о промышленных товарах, цена которых превышает заданное значение.
4. База данных содержит факты вида: книга(автор, название, издательство, год_издания), украина(город).
- а) вывести весь список книг;
 - б) вывести список книг авторов Пушкина и Чехова;
 - в) вывести список книг, изданных в издательстве «Питер» не ранее 2000 года.
5. Составить программу, реализующую авиасправочник. В справочнике содержится следующая информация о каждом рейсе: номер рейса, пункт назначения, время вылета, дни(ежедн., чет, нечет). Вывести:
- а) всю информацию из справочника;
 - б) информацию о самолетах, вылетающих в заданный пункт по четным дням;
 - в) информацию о самолетах, вылетающих ежедневно не позже указанного времени.
6. Составить программу, реализующую географический справочник. В справочнике содержится следующая информация о каждой стране: название страны, название столицы, численность населения, географическое положение (Европа или Азия). Вывести:
- а) всю информацию из справочника;
 - б) информацию о странах, численность населения которых превышает заданное значение;
 - в) информацию о европейских странах, численность населения которых не превышает заданное значение.
7. Составить программу, реализующую словарь. В словаре содержится следующая информация: слово и его перевод (русские и английские слова). Реализовать вывод всего словаря, перевод с русского на английский, с английского на русский (с несколькими значениями).
8. Составить программу, реализующую телефонный справочник. В справочнике содержится следующая информация о каждом абоненте: имя и телефон. Реализовать вывод всей информации из справочника, поиск телефона по имени, поиск имени по телефону
9. База данных содержит факты вида: **ученик(имя, класс) и увлекается(имя, хобби)**. Составить программу, которая выводит:
- а) список всех учеников и их увлечения;
 - б) подбирает одному из учеников указанного класса, увлекающемуся кино, пару из других классов. Вывести все возможные пары.
10. База данных содержит факты вида: **ученик(имя, класс) и играет(имя, вид_спорта)**. Составить программу, которая:
- а) выводит список всех учеников заданного класса и вид спорта, которым они увлекаются;

б) подбирает одному из учеников указанного класса, играющему в бадминтон, пару из других классов. Вывести все возможные пары.

1.3 Решение логических задач в ПРОЛОГе

Краткие теоретические сведения

ПРОЛОГ позволяет наиболее естественным образом решать логические задачи, моделируя процесс размышления человека с помощью правил.

Многие логические задачи связаны с рассмотрением нескольких конечных множеств с одинаковым количеством элементов, между которыми устанавливается взаимно-однозначное соответствие. В ПРОЛОГе эти множества можно описывать как базы данных, а зависимости между объектами устанавливать с помощью правил.

Пример 1

В автомобильных гонках три первых места заняли Алеша, Петя и Коля. Какое место занял каждый из них, если Петя занял не второе и не третье место, а Коля - не третье?

Решение

Традиционным способом задача решается заполнением таблицы. По условию задачи Петя занял не второе и не третье место, а Коля - не третье. Это позволяет поставить символ '-' в соответствующих клетках.

Имя	I место	II место	III место
Алеша			
Петя		-	-
Коля			-

Между множеством имен участников гонки и множеством мест должно быть установлено взаимнооднозначное соответствие. Поэтому определяем занятое место сначала у Пети, затем у Коли и, наконец, у Алеши. В соответствующих клетках проставляем знак '+'. В каждой строке и каждом столбце должен быть только один такой знак.

Имя	I место	II место	III место
Алеша	-	-	+
Петя	+	-	-
Коля	-	+	-

Из последней таблицы следует, что Алеша занял третье место, Петя - первое, Коля - второе.

Программа на ПРОЛОГе будет выглядеть следующим образом:
PREDICATES

```

имя(string)
место(string)
соответствие(string,string)
решение(string,string,string,string,string,string)
CLAUSES
    имя(алеша) .
    имя(петя) .
    имя(коля) .
    место(первое) .
    место(второе) .
    место(третье) .
    /* Устанавливаем взаимнооднозначное соответствие
    между множеством имен и множеством мест, X - имя, Y - место */
    /* Петя занял не второе и не третье место */
    соответствие(X, Y) :- имя(X), X=петя,
                           место(Y), not(Y=второе),
                           not(Y=третье) .

    /* Коля занял не третье место */
    соответствие(X, Y) :- имя(X), X=коля,
                           место(Y), not(Y=третье) .

    соответствие(X, Y) :- имя(X), X=алеша, место(Y) .
    /* У всех ребят разные места */
    решение(X1,Y1,X2,Y2,X3,Y3) :-
        X1=петя, соответствие(X1,Y1),
        X2=коля, соответствие(X2,Y2),
        X3=алеша, соответствие(X3,Y3),
        Y1<>Y2, Y2<>Y3, Y1<>Y3.

GOAL
    решение(X1,Y1,X2,Y2,X3,Y3), write(X1," - ",Y1),nl,
    write(X2," - ",Y2),nl,write(X3," - ",Y3),nl.

```

Результат выполнения программы

```

петя   -   первое
коля   -   второе
алеша  -   третье

```

Пример 2

Наташа, Валя и Аня вышли на прогулку, причем туфли и платье каждой были или белого, или синего, или зеленого цвета. У Наташи были зеленые туфли, а Валя не любит белый цвет. Только у Ани платье и туфли были одного цвета. Определить цвет туфель и платья каждой из девочек, если у всех туфли и платья были разного цвета.

Решение

PREDICATES

```

имя(string)
туфли(string)
платье(string)
соот(string,string,string)
решение(string,string,string,string,string,string,
          string,string,string)

```

CLAUSES

```

имя(наташа) .
имя(валя) .
имя(аня) .
туфли(белый) .
туфли(синий) .
туфли(зеленый) .
платье(белый) .
платье(синий) .
платье(зеленый) .
% X – имя, Y – цвет туфель, Z – цвет платья
соот(X,Y,Z) :- имя(X) , туфли(Y) , платье(Z) ,
                X=наташа, Y=зеленый, Y<>Z .
соот(X,Y,Z) :- имя(X) , туфли(Y) , платье(Z) ,
                X=валя, not (Y=белый) ,
                not (Z=белый) , Y<>Z .
соот(X,Y,Z) :- имя(X) , туфли(Y) , платье(Z) , X=аня, Y=Z .
решение(X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3) :-
                X1=наташа, соот(X1,Y1,Z1) ,
                X2=валя, соот(X2,Y2,Z2) ,
                X3=аня, соот(X3,Y3,Z3) ,
                Y1<>Y2, Y2<>Y3, Y1<>Y3,
                Z1<>Z2, Z2<>Z3, Z1<>Z3 .

```

GOAL

```

решение(X1,Y1,Z1,X2,Y2,Z2,X3,Y3,Z3) ,
write(X1," туфли- ",Y1," платье- ",Z1),nl,
write(X2," туфли- ",Y2," платье- ",Z2),nl,
write(X3," туфли- ",Y3," платье- ",Z3),nl.

```

Результат выполнения программы

наташа	туфли -	зеленый	платье -	синий
валя	туфли -	синий	платье -	зеленый
аня	туфли -	белый	платье -	белый

Пример 3

Витя, Юра и Миша сидели на скамейке. В каком порядке они сидели, если известно, что Миша сидел слева от Юры, а Витя слева от Миши.

Решение

```

PREDICATES
    слева(string,string)
    ряд(string,string,string)
CLAUSES
    /* Миша сидел слева от Юры */
    слева(миша, юра) .
    /* Витя сидел слева от Миши */
    слева(витя, миша) .
    /* Объекты X, Y и Z образуют ряд,
    если X слева от Y и Y слева от Z */
    ряд(X, Y, Z) :- слева(X,Y) , слева(Y, Z) .
GOAL
    ряд(X, Y, Z) , write(X, "-", Y, "-", Z) , nl.

```

Результат выполнения программы

витя-миша-юра

Пример 4

Известно, что тополь выше березы, которая выше липы. Клен ниже липы, а сосна выше тополя и ниже ели. Определить самое высокое и самое низкое дерево.

Решение

```

DOMAINS
    name=string
PREDICATES
    выше(name,name)
    ряд(name,name,name,name,name,name)
CLAUSES
    выше(тополь, береза) .
    выше(липа, клен) .
    выше(ель, сосна) .
    выше(береза, липа) .
    выше(сосна, тополь) .
    ряд(X1, X2, X3, X4, X5, X6) :- выше(X1, X2) , выше(X2, X3) ,
                                   выше(X3, X4) , выше(X4, X5) ,
                                   выше(X5, X6) .
GOAL
    ряд(X, _, _, _, _, Y) , write("Самое высокое - ", X) , nl,
    write("Самое низкое - ", Y) , nl.

```

Результат выполнения программы

Самое высокое - ель
Самое низкое - клен

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Трое ребят вышли гулять с собакой, кошкой и хомячком. Известно, что Петя не любит кошек и живет в одном подъезде с хозяйкой хомячка. Лена дружит с Таней, гуляющей с кошкой. Определить, с каким животным гулял каждый из детей.
2. Беседуют трое друзей: Белокуров, Рыжов и Чернов. Брюнет сказал Белокурову: «Любопытно, что один из нас блондин, другой – брюнет, а третий – рыжий, но ни у кого цвет волос не соответствует фамилии». Какой цвет волос у каждого из друзей?
3. Витя, Юра, Миша и Дима сидели на скамейке. В каком порядке они сидели, если известно, что Юра сидел справа от Димы, Миша справа от Вити, а Витя справа от Юры.
4. Известно, что Волга длиннее Амударьи, а Днепр короче Амударьи. Лена длиннее Волги. Определить вторую по протяженности реку.

1.4 Арифметические вычисления и рекурсия в ПРОЛОГе

Краткие теоретические сведения

Хотя Пролог не предназначен для решения вычислительных задач, его возможности вычислений аналогичны соответствующим возможностям таких языков программирования как Basic, C, Pascal.

В языке Пролог имеется ряд встроенных функций для вычисления арифметических выражений, некоторые из которых перечислены в таблице 1.

Таблица 1. Математические операции и функции в Прологе

$X + Y$	Сумма X и Y
$X - Y$	Разность X и Y
$X * Y$	Произведение X и Y
X / Y	Деление X на Y
$X \bmod Y$	Остаток от деления X на Y
$X \operatorname{div} Y$	Целочисленное деление X на Y
$\operatorname{abs}(X)$	Абсолютная величина числа X
$\operatorname{sqrt}(X)$	Квадратный корень из X
$\operatorname{random}(X)$	Случайное число в диапазоне от 0 до 1
$\operatorname{random}(\operatorname{Int}, X)$	Случайное целое число в диапазоне от 0 до Int
$\operatorname{round}(X)$	Округление X
$\operatorname{trunc}(X)$	Целая часть X
$\sin(X)$	Синус X

$\cos(X)$	Косинус X
$\arctan(X)$	Арктангенс X
$\tan(X)$	Тангенс X
$\ln(X)$	Натуральный логарифм X
$\log(X)$	Логарифм X по основанию 10

Пример 1.

Вычислить значение выражения $Z=(2*X+Y)/(X-Y)$ для введенных X и Y.

Решение:

```
PREDICATES
    знач_выраж(real, real)
CLAUSES
    знач_выраж(X, Y) :- X <> Y, Z = (2*X+Y) / (X-Y),
                        write("Z=", Z);
                        write("Делить на 0 нельзя!").
GOAL
    Write("X="), readreal(X),
    Write("Y="), readreal(Y), знач_выраж(X, Y), nl.
```

Комментарий: readreal – предикат для ввода действительных чисел

Результат выполнения программы:

1-й случай:

```
X=4
Y=4
Делить на 0 нельзя!
```

2-й случай:

```
X=5
Y=2
Z=4
```

Пример 2.

Найти минимальное из двух введенных A и B.

Решение:

```
PREDICATES
    min(integer, integer, integer)
CLAUSES
    min(A, B, A) :- A <= B, !.
    min(A, B, B).
GOAL
    Write("A="), readreal(A), Write("B="), readreal(B),
    min(A, B, Min), write("min=", Min), nl.
```


Результат выполнения программы:

1-й случай:

A=5
B=17
min=5

2-й случай:

A=35
B=18
min=18

3-й случай:

A=8
B=8
min=8

Пример 3.

Определить, является четным или нечетным случайным образом выбранное число от 0 до 20.

Решение:

```
PREDICATES
    chet
CLAUSES
    chet:-random(20,X),write(X),X mod 2=0,
        write(" - четное"),!.
    chet:-write(" - нечетное").
GOAL
    chet.
```

Результат выполнения программы:

1-й случай:

6 - четное

2-й случай:

19 - нечетное

Рекурсивная процедура – это процедура, вызывающая сама себя до тех пор, пока не будет соблюдено некоторое условие, которое остановит рекурсию. Такое условие называют *граничным*. Рекурсия в ПРОЛОГе используется для организации повторяющихся действий.

Рекурсивное правило всегда состоит по крайней мере из двух частей, одна из которых является *нерекурсивной*. Она и определяет граничное условие.

В рекурсивной процедуре нет проблемы запоминания результатов ее выполнения, потому что любые вычисленные значения можно передавать из одного вызова в другой как аргументы рекурсивно вызываемого предиката.

Рекурсия является эффективным способом для решения задач, содержащих в себе подзадачу такого же типа.

Пример 4. Вычисление факториала.

Решение:

```
PREDICATES
    fact(integer, integer)
CLAUSES
    fact(0, 1) :- !.                % Факториал нуля равен единице
    fact(N, F) :- N1=N-1,           % уменьшаем N на единицу,
                                fact(N1, F1), % вычисляем факториал нового числа,
                                F=N*F1.      % а затем умножает его на N
GOAL
    write("N="), readint(N), fact(N, F), write("F=", F), nl.
```

Результат выполнения программы:

1-й случай:

N=0

F=1

2-й случай:

N=1

F=1

3-й случай:

N=4

F=24

Пример 5

Составить программу для вычисления $Y=X^n$, X, n – целые числа

Решение:

Составим правило **stepen**, состоящее из 3-х частей.

1-я часть правила (нерекурсивная) определяет, что $X^0=1$.

2-я часть правила (рекурсивная) вычисляет X^n для положительного n.

3-я часть (рекурсивная) - вычисляет X^n для отрицательного n
(добавляется необходимое условие $X \neq 0$)

```
PREDICATES
    stepen(real, real, real)
CLAUSES
    stepen(X, 0, 1) :- !.
    stepen(X, N, Y) :- N>0, N1=N-1, stepen(X, N1, Y1), Y=Y1*X, !.
    stepen(X, N, Y) :- X<>0, K=-N, stepen(X, K, Z), Y=1/Z.
GOAL
    write("X="), readreal(X),
```

```
write("N="), readreal(N),  
stepen(X,N,Y), write("Y=",Y), nl.
```

Результат выполнения программы:

1-й случай:

X=3

N=2

Y=9

2-й случай:

X=2

N=-2

Y=0.25

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Составить программу для вычисления значения выражения $Y=(X^2+1)/(X-2)$ для введенного X.
2. Составить программу для вычисления значения выражения $S=2(X^2+Y^2)/(X+Y)$ для введенных X и Y.
3. Составить программу для вычисления значения выражения $z=e^x \sin x + 3 \ln x$ для введенного X.
4. Составить программу для вычисления значения выражения $y=\ln(\lg(\sin x + e^x))$ для введенного X.
5. Составить программу для проверки введенного натурального числа на четность.
6. Составить программу для проверки попадает ли введенное число X в заданный промежуток [a,b].
7. Составить программу для выбора наименьшего из трех введенных чисел.
8. Составить программу для выбора наибольшего из трех введенных чисел.
9. Вычислить сумму $1+2+3+\dots+N$.
10. Подсчитать сумму ряда целых четных чисел от 2 до N.
11. Вычислить сумму ряда целых нечетных чисел от 1 до n.
12. Найти значение произведения: $2*4*6*\dots*12$
13. Найти значение произведения: $1*3*5*\dots*11$
14. Вычислить значение n-го члена ряда Фибоначчи: $f(0)=0, f(1)=1, f(n)=f(n-1)+f(n-2)$.

1.5 Списки

Краткие теоретические сведения

Список – это объект, который содержит конечное число других объектов. Список в ПРОЛОГе заключается в квадратные скобки и элементы списка разделяются запятыми. Список, который не содержит ни одного элемента, называется *пустым* списком.

Список является рекурсивным объектом. Он состоит из *головы* (первого элемента списка) и *хвоста* (все последующие элемента). Хвост также является списком. В ПРОЛОГе имеется операция “|”, которая позволяет делить список на голову и хвост. Пустой список нельзя разделить на голову и хвост.

Тип данных "список" объявляется в программе на Прологе следующим образом:

DOMAINS

списковый_тип = тип*

где "тип" - тип элементов списка; это может быть как стандартный тип, так и нестандартный, заданный пользователем и объявленный в разделе DOMAINS ранее.

Основными операциями на списками являются:

- формирование списка;
- объединение списков;
- поиск элемента в списке;
- вставка элемента в список и удаление из списка.
-

Пример 1

Сформировать список вида [7,6,5,4,3,2,1]

Решение

DOMAINS

list = integer*

PREDICATES

genl(integer, list)

CLAUSES

genl(0, []) :- !.

genl(N, [N|L]) :- N1=N-1, genl(N1, L).

GOAL

genl(7, L), write(L), nl.

Результат выполнения программы:

[7, 6, 5, 4, 3, 2, 1]

Пример 2

Сформировать список из N элементов, начиная с 2. Каждый следующий на 4 больше предыдущего.

Решение

DOMAINS

list = integer*

```

PREDICATES
    genl( integer, integer, list )
CLAUSES
    genl(N2,N2,[]) :-!.
    genl(N1,N2,[N1|L]) :-N1<N2, N=N1+4,
                           genl(N,N2,L) .
GOAL
    write("N="),readint(N),K=4*(N+1)-2,
    genl(2,K,L),write(L),nl.

```

Результат выполнения программы:

```

N=5
[2,6,10,14,18]

```

Пример 3

Сформировать список последовательных натуральных чисел от 4 до 20 и найти количество его элементов.

Решение:

```

DOMAINS
    list = integer*
PREDICATES
    genl1(integer, integer, list )
    len(integer, list )
CLAUSES
    genl1(N2,N2,[]) :-!.
    genl1(N1,N2,[N1|L]) :-N1<N2, N=N1+1, genl1(N,N2,L) .
    len(0,[]) .
    len(X,[_|L]) :-len(X1,L), X=X1+1.
GOAL
    genl1 (4,21,L ),write(L),nl,
    len(X, L),write("Количество элементов=",X),nl.

```

Результат выполнения программы:

```

[4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20]
Количество элементов=17

```

Пример 4

Определить, содержится ли введенное число X в заданном списке L.

Решение:

```

DOMAINS
    list = integer*
PREDICATES
    member(integer, list)
CLAUSES

```

```

member(X, [X|_]) :- write("yes"), !.
member(X, []) :- write("no"), !.
member(X, [_|L]) :- member(X, L).

GOAL
L=[1,2,3,4], write(L), nl, write("X="), readint(X),
member(X, L), nl.

```

Результат выполнения программы:

1-й случай:

```

[1,2,3,4]
X=3
yes

```

2-й случай:

```

[1,2,3,4]
X=5
no

```

Пример 5

Сформировать списки L1=[1,2,3], L2=[10,11,12,13,14,15] и объединить их в список L3.

Решение:

```

DOMAINS
    list = integer*
PREDICATES
    genl1(integer, integer, list)
    append(list, list, list)
CLAUSES
    genl1(N2, N2, []) :- !.
    genl1(N1, N2, [N1|L]) :- N1 < N2, N=N1+1, genl1(N, N2, L).
    append([], L, L).
    append([X|L1], L2, [X|L3]) :- append(L1, L2, L3).
GOAL
    genl1(1, 4, L1), write("L1=", L1), nl,
    genl1(10, 16, L2), write("L2=", L2), nl,
    append(L1, L2, L3), write("L3=", L3), nl.

```

Результат выполнения программы:

```

L1=[1,2,3]
L2=[10,11,12,13,14,15]
L3=[1,2,3,10,11,12,13,14,15]

```

Пример 6

Удалить из списка, элементами которого являются названия дней недели, указанный элемент.

Решение:

```

DOMAINS
    list = symbol*
PREDICATES
    del(symbol,list,list)
CLAUSES
    del(X,[X|L],L) .
    del(X,[Y|L],[Y|L1]):-del(X,L,L1) .
GOAL
    L=[пн, вт, ср, чт, пт, сб, вс],write("L=",L),nl,
    write("X="),readln(X),
    del(X,L,L1),write("L1=",L1),!;
    write("Элемент отсутствует в списке"),nl.

```

Результат выполнения программы:

1-й случай:

```

L=["пн", "вт", "ср", "чт", "пт", "сб", "вс"]
X=ср
L1=["пн", "вт", "чт", "пт", "сб", "вс"]

```

2-й случай:

```

L=["пн", "вт", "ср", "чт", "пт", "сб", "вс"]
X=пр
Элемент отсутствует в списке

```

Пример 7

Вставить в список имен новый элемент, значение которого вводится с клавиатуры. Вывести все возможные варианты вставок.

Решение:

```

DOMAINS
    list = symbol*
PREDICATES
    del(symbol,list,list)
    ins(symbol,list,list)
CLAUSES
    del(X,[X|L],L) .
    del(X,[Y|L],[Y|L1]):-del(X,L,L1) .
    ins(X,L1,L):-del(X,L,L1) .
GOAL
    L=[olga, oksana, toma, dima],write("L=",L),nl,
    write("X="),readln(X),
    ins(X,L,L1),write("L1=",L1),nl, fail.

```

Результат выполнения программы:

```

L=["olga", "oksana", "toma", "dima"]

```

```

X=vera
L1=["vera", "olga", "oksana", "toma", "dima"]
L1=["olga", "vera", "oksana", "toma", "dima"]
L1=["olga", "oksana", "vera", "toma", "dima"]
L1=["olga", "oksana", "toma", "vera", "dima"]
L1=["olga", "oksana", "toma", "dima", "vera"]

```

Пример 8

Найти сумму элементов списка целых чисел.

Решение:

```

DOMAINS
    list=integer*
PREDICATES
    sum_list(list, integer)
CLAUSES
    sum_list([], 0).
    sum_list([X|L], S):-sum_list(L, S1), S=S1+X.
GOAL
    L=[1, 2, 3, 4, 5], sum_list(L, S), write("S=", S).

```

Результат выполнения программы:

S=15

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Сформировать список [2, 4, 6, 8, 10] и удалить из него введенное число.
2. Сформировать списки [1, 3, 5, 7, 9] и [2, 4, 6, 8, 10] и объединить их в один.
3. Сформировать список [3, 6, 9, 12, 15, 18] и вставить в него введенное число.
4. Сформировать список из N натуральных чисел, начиная с 10. Каждое следующее на 5 больше предыдущего.
5. Сформировать список [3, 6, 9, 12, 15] и найти сумму его элементов
6. Сформировать список [6, 5, 4, 3, 2] и найти сумму его элементов
7. Сформировать список [7, 5, 3, 1] и найти произведение его элементов
8. Сформировать список из N последовательных натуральных чисел, начиная с 10. Найти сумму его элементов

1.6 Создание экспертных систем средствами ПРОЛОГа

Краткие теоретические сведения

Экспертные системы (ЭС) - это сложные программные комплексы, аккумулирующие знания специалистов в конкретных предметных областях и тиражирующие этот эмпирический опыт для консультаций менее квалифицированных пользователей.

При разработке экспертной системы на ПРОЛОГе база знаний записывается в виде предикатов, которые описывают обобщённые и конкретные сведения об объектах данной предметной области, выражают правила определения понятий, а формулируют запросы.

В программе на ПРОЛОГе в разделе clauses перечисляются факты, содержащие сведения, которые являются постоянными для данной предметной области. Такая база данных является статичной, т.к. факты невозможно удалить или добавить новые в ходе выполнения программы. Для работы с данными, которые ЭС получает от пользователя в процессе работы и сохраняет их в рабочей памяти, используется внутренняя (динамическая) база данных.

Динамическая база данных состоит из фактов, которые в ходе выполнения программы можно добавлять и удалять из базы, кроме того в ПРОЛОГе есть средства для сохранения их в файл и загрузки из файла в БД. Во внутренней базе данных могут храниться только факты, а не правила.

Таблица 1. Предикаты и их назначение [5].

Предикат	Назначение
asserta (<the fact>) asserta (<the fact>, facts_sectionName)	Факт добавляется в начало БД
assertz (<the fact>) assertz (<the fact>, facts_sectionName)	Факт добавляется в конец БД
assert (<the fact>) assert (<the fact>, facts_sectionName)	Добавление факта в конец БД, аналогичен assertz.
retract (<the fact>[databaseName])	Удаление факта из БД.
retractall (<the fact>[databaseName])	Удаление всех фактов из БД.

Построим небольшую экспертную систему, которая будет определять одну из нескольких рыб по признакам, указанным пользователем. Система будет задавать вопросы и строить логические выводы на основе полученных ответов.

Типичный диалог экспертной системы с пользователем может выглядеть следующим образом (рис.27):

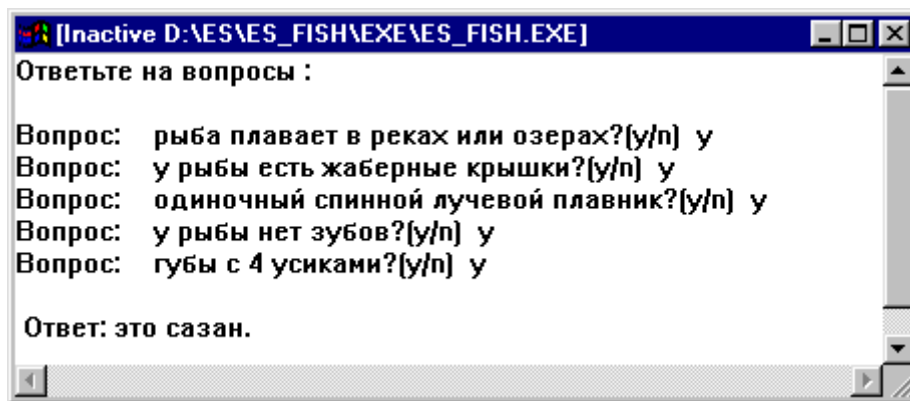


рис.27. Диалог экспертной системы с пользователем

Первым шагом построения такой системы является обеспечение ее знаниями, необходимыми для выполнения рассуждений. Программа должна во время консультаций выводить заключения из информации, имеющейся в базе знаний, а также использовать новую информацию, полученную от пользователя. Поэтому минимальная ЭС должна включать:

- базу знаний;
- механизм вывода;
- пользовательский интерфейс.

Разработку любой ЭС следует начать с исследования предметной области. Пусть на основе бесед с экспертом были получены следующие эмпирические правила:

1) ЕСЛИ

это отряд карпообразные и
И
у рыбы желто-золотистый окрас
И
губы с 4 усиками
ТО
это сазан

2) ЕСЛИ

это отряд карпообразные
И
у рыбы плавники с розовыми перьями
ТО
это плотва

3) ЕСЛИ

спинной плавник узкий
И
у рыбы желто-золотистый окрас
И
это отряд карпообразные
ТО

это лещ

4) ЕСЛИ

у рыбы нет зубов

И

одиночный спинной лучевой плавник

И

это костная рыба

И

это пресноводная рыба

ТО

это отряд карпообразные

5) ЕСЛИ

у рыбы есть костный скелет

ИЛИ

у рыбы есть жаберные крышки

ТО

это костная рыба

6) ЕСЛИ

рыба плавает в озерах

ИЛИ

рыба плавает в реках

ТО

это пресноводная рыба

Для создания базы знаний используем предикаты:

`fish(symbol)`

`otrajd(symbol)`

`vid(symbol)`

`priznak(symbol)`

Базу знаний будут составлять следующие правила:

`fish("это сазан") :-`

`otrajd("отряд карпообразные"),`

`priznak("губы с 4 усиками").`

`fish("это плотва") :-`

`otrajd("отряд карпообразные"),`

`priznak("плавники с розовыми перьями").`

`fish("это лещ") :-`

`otrajd("отряд карпообразные"),`

`priznak("у рыбы желто-золотистый окрас"),`

`priznak("у рыбы спинной плавник узкий").`

Необходимо предусмотреть, что искомой рыбы в базе знаний нет:

```

fish("Данной рыбы в базе знаний не обнаружено").
otrajd("отряд карпообразные):-
    vid("пресноводная рыба"),
    vid("костная рыба"),
    priznak("одиночный спинной лучевой плавник"),
    priznak("у рыбы нет зубов").
vid("костная рыба):-
    priznak("у рыбы есть жаберные крышки");
    priznak("у рыбы есть костный скелет").
vid("пресноводная рыба):-
    priznak("рыба плавает в реках или озерах").

```

Для хранения информации, полученной от пользователя, используются предикаты **yes** и **no**, составляющие внутреннюю базу фактов. Предикат **yes** служит для хранения фактов, соответствующих положительному ответу, а предикат **no** – для хранения отрицательных ответов. Т.е. предикат **yes** утверждает наличие какого-либо признака у рыбы, а **no** – отсутствие указанного признака. Эти предикаты объявляются в разделе внутренней базы фактов:

```

global facts
    yes (symbol)
    no  (symbol)

```

Добавить новые факты во внутреннюю базу можно с помощью правила `add_to_database`, состоящего из двух частей. Первая часть добавляет факты, соответствующие положительному ответу (с клавиатуры вводится 'y'). Вторая часть правила добавляет факты, указывающие на отсутствие данного признака у рыбы.

```

add_to_database (Y, 'y') :- assertz (yes (Y)).
add_to_database (Y, 'n') :- assertz (no (Y)), fail.

```

Необходимо предусмотреть очистку внутренней базы фактов. Для этого создадим правило:

```

clear_from_database :- retract(yes(_)), fail.
clear_from_database :- retract(no(_)), fail.

```

Для проверки наличия у рыбы определенного признака создадим правило `priznak (Y)`:

```

priznak (Y) :- yes (Y), !.
priznak (Y) :- not(no (Y)),
                question (Y).

```

Формулировка вопроса, ввод ответа и сохранение соответствующего правила осуществляется с помощью правил:

```
answer :- fish(X),!,nl,
         save("BF1.dbf"),
         write (" Ответ: ",X,"."),nl.
question(Y) :-
         write ("Вопрос:      ",Y,"?(y/n)  "),
         otvet(X),
         write(X),nl,
         add_to_database (Y,X).
otvet(C) :-readchar(C).
```

И, наконец, правило begin, запускающее сеанс консультации:

```
begin :- write ("Ответьте на вопросы :"),nl,nl,
         answer,
         clear_from_database,
         nl,nl,nl,nl,
         exit.
```

Полный листинг программы выглядит следующим образом:

```
GLOBAL FACTS
yes (symbol)
no  (symbol)

PREDICATES
fish(symbol)
otrajd(symbol)
vid(symbol)
begin
answer
question(symbol)
add_to_database(symbol,char)
otvet(char)
clear_from_database
priznak(symbol)
```

GOAL

begin.

CLAUSES

begin :-

write ("Ответьте на вопросы :"),nl,nl,
answer,
clear_from_database,
nl,nl,nl,nl,
exit.

answer :-

fish(X),!,nl,
save("BF1.dbf"),
write (" Ответ: ",X,"."),nl.

question(Y) :-

write ("Вопрос: ",Y,"? "),
otvet(X),
write(X),nl,
add_to_database (Y,X).

otvet(C) :-

readchar(C).

priznak (Y) :-

yes (Y),!.

priznak (Y) :-

not(no (Y)),
question (Y).

add_to_database (Y,'y') :-

assertz (yes (Y)).

add_to_database (Y,'n') :-

assertz (no (Y)),fail.

clear_from_database :- retract (yes(_)),fail.

clear_from_database :- retract (no(_)),fail.

fish("это сазан"):-

```

    otrajd("отряд карпообразные"),
    признак("губы с 4 усиками").
fish("это плотва"):-
    otrajd("отряд карпообразные"),
    признак("плавники с розовыми перьями").
fish("это лещ"):-
    otrajd("отряд карпообразные"),
    признак("у рыбы желто-золотистый окрас"),
    признак("у рыбы спинной плавник узкий").
fish("Данной рыбы в базе знаний не обнаружено").
otrajd("отряд карпообразные"):-
    vid("пресноводная рыба"),
    vid("костная рыба"),
    признак("одиночный спинной лучевой плавник"),
    признак("у рыбы нет зубов").
vid("костная рыба"):-
    признак("у рыбы есть жаберные крышки");
    признак("у рыбы есть костный скелет").
vid("пресноводная рыба"):-
    признак("рыба плавает в реках или озерах").

```

ЗАДАНИЕ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Реализуйте данную программу в среде Visual Prolog и протестируйте ее.
2. Расширьте базу знаний экспертной системы, добавив следующие правила:

- 1) ЕСЛИ
 - у рыбы есть электрические органы
 - И
 - это отряд скаты
 - ТО
 - это электрический скат
- 2) ЕСЛИ
 - у рыбы на хвосте ядовитый шип
 - И
 - это отряд скаты

ТО

это скат-хвостокол

3) ЕСЛИ

у рыбы серо-коричневый окрас

И

у рыбы коническая морда

И

это отряд акулы

ТО

это гиганская акула

4) ЕСЛИ

это отряд акулы

И

рыба нападает на людей

И

у рыбы молотообразная морда

ТО

это рыба молот

5) ЕСЛИ

у рыбы нет хвостового плавника

И

у рыбы тонкий длинный хвост

И

это хрящевая рыба

И

это морская рыба

ТО

это отряд скаты

6) ЕСЛИ

это морская рыба

И

это хрящевая рыба

И

плавники не гибкие

И

хвост ассиметричный

ТО

это отряд акулы

7) ЕСЛИ

у рыбы нет плавательного пузыря

ИЛИ

у рыбы есть хрящевый скелет

ТО

это хрящевая рыба

8) ЕСЛИ
рыба плавает в морях
ТО
это морская рыба

3. Протестируйте полученную экспертную систему.

2. ОСНОВЫ ФУНКЦИОНАЛЬНОГО ПРОГРАММИРОВАНИЯ

2.1. Основы работы в среде DrRacket

Краткие теоретические сведения

Функциональным называется программирование при помощи функций в математическом их понимании. Функциональное программирование основано на следующей идее: в результате каждого действия возникает значение, которое может быть аргументом следующего действия. Программы строятся из логически расчлененных определений функций. Каждое определение функции состоит из организующих вычисления управляющих структур и из вложенных, в том числе вызывающих самих себя (рекурсивных) вызовов функций.

Вызов функций является единственной разновидностью действий, выполняемых в функциональной программе. В алгоритмических языках программа является последовательностью операторов, вызовов процедур в соответствии с алгоритмом. В функциональном программировании программа состоит из вызовов функций и описывает то, что нужно делать и что собой представляет результат решения, а не как нужно действовать для получения результата. В языках функционального программирования программа и обрабатываемые ею данные имеют единую списочную форму представления

Язык LISP (LISt Processing) – язык программирования высокого уровня, разработан в 1961 году Дж. Маккарти. В основе Лиспа лежит функциональная модель вычислений, ориентированная прежде всего на решение задач нечислового характера.

Racket (ранее назывался 'PLTScheme') является мультипарадигмальным языком программирования в семействе Lisp. Библиотеки Racket поддерживают разработку веб-серверов и баз данных, GUI и диаграмм.

Язык Racket является интерпретируемым языком. Программа на Racket состоит из выражений, имеющих значения. Вычисляя выражения, интерпретатор выполняет один и тот же цикл: считывает выражение, вычисляет выражение, возвращает результат.

Инструментальным средством реализации программ на Racket является среда DrRacket.

DrRacket может работать в нескольких режимах:

1. интерактивно, с использованием простого текстового интерфейса командной строки;
2. в режиме программирования

После запуска DrRacket открывается окно, которое разделено на две части (рис.1). Верхняя представляет собой текстовый редактор для набора

программы. Нижняя часть предназначена для работы в режиме командной строки, здесь также выводится результат выполнения программы.

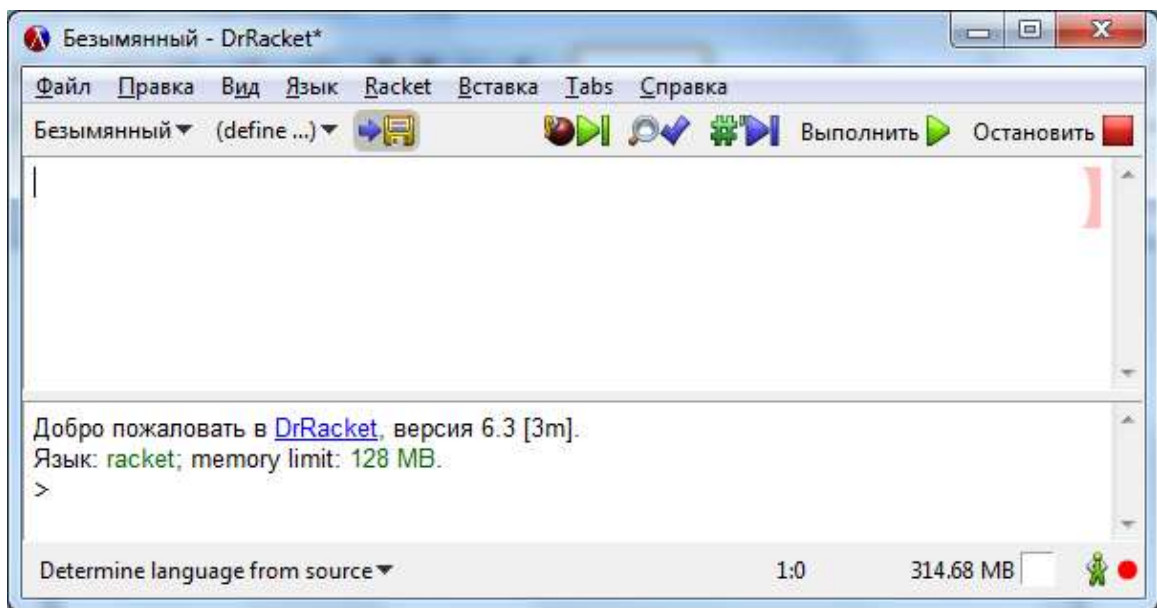


Рис.1. Окно DrRacket при первом запуске программы

При первом запуске программы необходимо выбрать язык. Для этого выполнить команды меню **Язык -> Выбрать язык...** и в открывшемся окне установить опцию **The Racket Language** (рис.2).

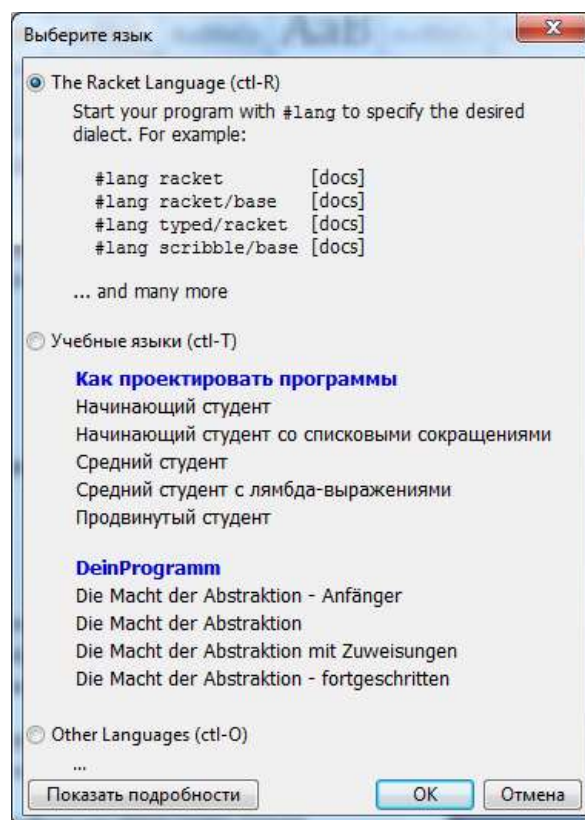


Рис.2. Выбор языка программирования

В режиме интерпретатора пользователь может использовать множество команд. Основным методом взаимодействия пользователя с DrRacket является ввод команд с командной строки. После появления на экране подсказки `>` пользователь может ввести команду (рис.1). Командами могут быть вызовы функций, глобальные переменные или константы. Если ввести вызов функции, вычисляется значение этой функции и на экран выводится результат.

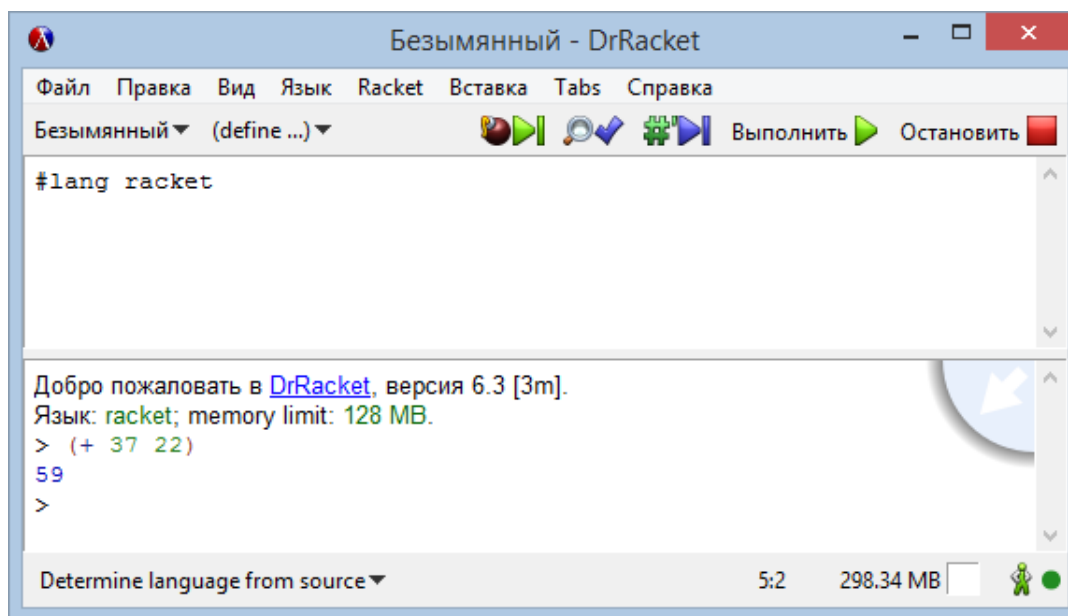


Рис.3 Вычисление суммы чисел в режиме командной строки

Вызовы функций в Racket имеют префиксную форму: аргументы функции могут стоять только после ее названия. Вызов функции начинается с открывающейся скобки, за которой следует имя функции, затем идут аргументы, каждый из которых отделен одним или несколькими пробелами. Аргументами функции могут быть данные простых типов, переменные или вызовы других функций. В конце вызова ставится закрывающаяся скобка. Возможно вкладывать одни вызовы функций в другие. Например, выражение $3 + 8 \cdot 9 + 4$ записывается следующим образом:

(+ 3 (* 8 9) 4)

Таблица 1. Запись математических функций в Racket

Функция	Обозначение функции в Racket
Сложение	+
Вычитание	-
Умножение	*
Деление	/
Целочисленное деление	quotient
Остаток от деления	remainder

Определение абсолютного значения	abs
Вычисление квадратного корня	sqrt
Вычисление квадрата	Sqr
Возведение в степень	expt
Нахождение минимума	min
Нахождение максимума	max
Синус	sin
Косинус	cos
Тангенс	tan
Арксинус	asin
Арккосинус	acos
Арктангенс	atan
Натуральный логарифм	log
Экспонента e^x	exp
Округление до ближайшего целого	round
Выбор целого случайного числа из интервала [0, n-1]	Random n

Пример.

В режиме командной строки вычислить значения выражений:

а) $2 + 5 \cdot 2 - |-8|$ б) $\max(3^2, \sqrt{|\sin 3.2|} \cdot \sqrt{e^3} \cdot 2^3)$

Решение

а) $(+ 2 (* 5 2) (- (abs -8)))$

б) $(\max (\text{expt } 2 \ 3) (\text{expt } 3 \ 2))$

в) $(* (\text{sqrt } (abs (\sin 3.2))) (\text{sqrt } (exp 3)))$

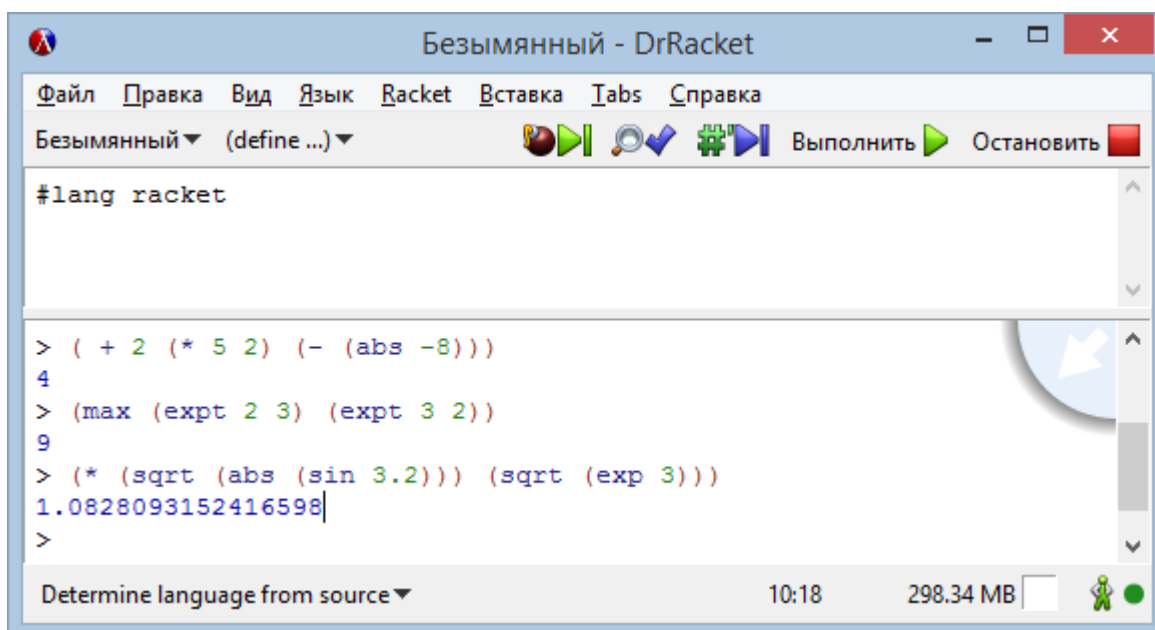


Рис.4. Вычисление выражений в режиме командной строки

Переменная в Racket задаётся следующей конструкцией языка:

(define имя <первоначальное значение>)

Пример:

(define width 3)

(define height 7)

(* 2 (+ width height))

Процедура в Racket определяется с помощью конструкции:

(define (<имя> <формальные-параметры> ...)

(<тело функции>))

*функция возвращает значение **последнего** вычислительного выражения в теле функции*

Например, функцию для вычисления среднего арифметического двух чисел можно определить следующим образом:

> (define (srednee x y)

(/ (+ x y) 2))

> (srednee 14 20)

17

Ввод-вывод данных в Racket можно организовать с помощью конструкций:

(read) ввод данных

(newline) переход на новую строку

(display выражение) вывод данных

(displayln выражение) вывод с переходом на новую строку

(write выражение) вывод данных

(writeln выражение) вывод с переходом на новую строку

(printf "<параметры формата>" выражение) форматированный вывод

Пример 1.

Составить программу для вычисления периметра прямоугольного треугольника по его катетам

Решение

; функция, определяющая значение гипотенузы прямоугольного треугольника по катетам x и y

(define (gipotenuza x y) ;

(sqrt (+ (sqr x) (sqr y))))

; функция, определяющая значение периметра

(define (perimetr x y)

(display "Периметр=")

(+ x y (gipotenuza x y)))

; функции для организации ввода данных

(define (a)

(display "Введите значение 1-го катета:") (read))

(define (b)

(display "Введите значение 2-го катета:") (read))

; вызов функции
(perimetr (a) (b))

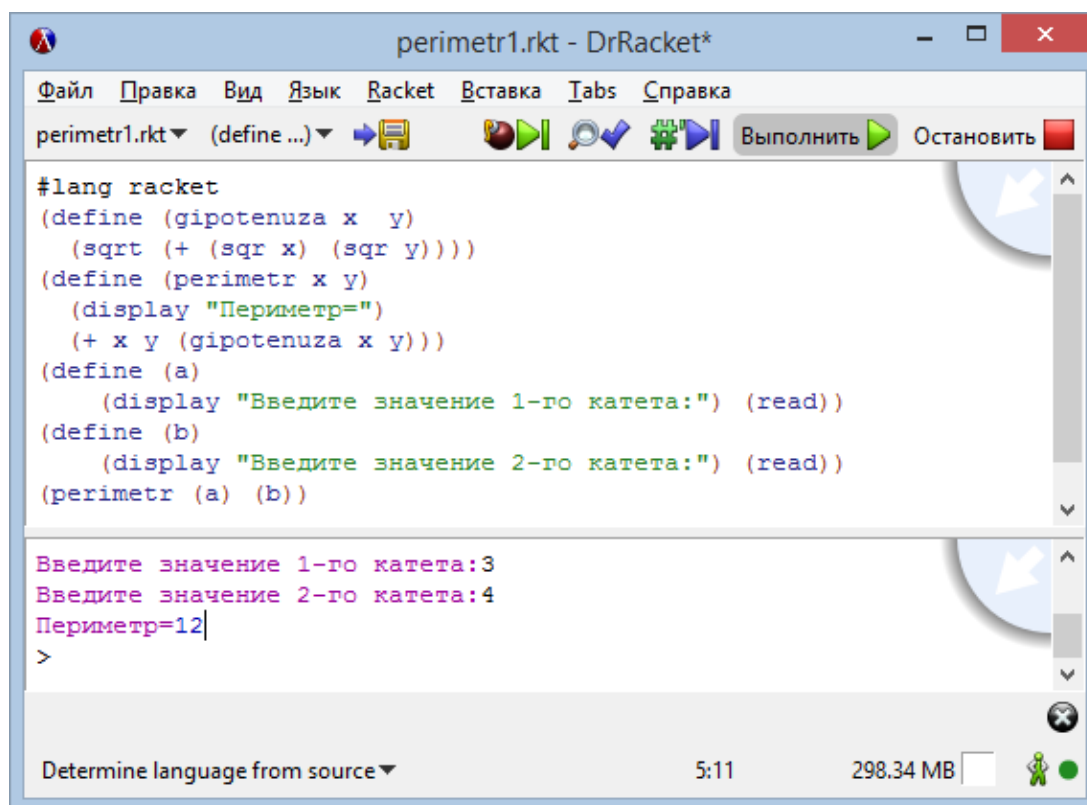


Рис.5. Результат выполнения программы

Пример 2.

Составить программу, записывающую введенное двузначное число в зеркальном отображении

Например, число 62 необходимо преобразовать в 26

Решение

Для решения необходимо выделить первую и вторую цифры число, а затем число единиц умножить на 10 и прибавить число десятков (меняем цифры местами). $26 = 2 \cdot 10 + 6$

Первую цифру введенного числа n можно выделить, используя функцию *quotient*:

(quotient n 10).

Вторую – с помощью функции *remainder*:

(remainder n 10))

А затем «собрать» число, переставив цифры:

(+ (* 10 (remainder n 10)) (quotient n 10)))

Текст программы

;функция, которая переставляет цифры заданного числа

(define (invers n)

(display "n=")

; функции для ввода числа

(define (a)

```
(display "Введите число:") (read))
```

; вызов функции

(invers (a))

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. В режиме командной строки вычислить значения выражений:

a) $(4^2 - 5) * (3 + 4)$ b) $4^2 + 28 / (5 + 2)$ c) $\sqrt{5^4 + \sqrt{7^2 + 1} + \ln 20.5}$

d) $|3e^3 - 2\ln 34|$ e) $\max(2^3, 3^2, 2^5)$ f) $3^3 - e^{5+\sin 2}$

$$1.1e^3 + |\cos \sqrt{\pi}| - \frac{4}{9}$$

g) $\sin 1 + 1/(\ln 5 - 5^3)$ h) $2e^4 - 4 - |\sin 6^2|$ i)

2. Вычислить периметр и площадь произвольного треугольника по длинам его сторон (для вычисления площади используйте формулу Герона

$S = \sqrt{p \cdot (p-a) \cdot (p-b) \cdot (p-c)}$, где а, b, с – стороны треугольника, р – полупериметр)

3. Вычислить длину окружности и площадь круга по заданному значению радиуса. ($L = 2\pi R$, $S = \pi R^2$)

4. Составить программу, которая переводит значение температуры из шкалы Цельсия в шкалу Фаренгейта по формуле $TF = 1.8TC + 32$

5. Составить программу, которая переводит значение веса из фунтов в килограммы (1 фунт=409,5 г).

6. Составить программу для перевода суммы из долларов в рубли. Вводится текущий курс доллара и сумма в долларах.

7. Написать программу для вычисления стоимости поездки на дачу (туда и обратно). Исходные данные: расстояние до дачи (в км), количество бензина, которое потребляет автомобиль на каждые 100 км, цена 1 л бензина.

8. Найти сумму цифр введенного двузначного числа.

9. Определить сумму квадратов цифр введенного двузначного числа.

10. Введено 3-значное число. Вывести число в зеркальном отображении

11. Определить сумму квадратов цифр введенного 3-значного числа.

12. Определить сумму цифр введенного 3-значного числа.

13. Вводится 4-значное число. Из его цифр получить два двузначных числа.

Первое состоит из 1 и 3 цифр исходного числа, второе – из 2-й и 4-й.

Например, 3765 $\rightarrow$ 36 и 75

14. Вводится 3-значное число. Из его цифр получить два двузначных числа. Первое состоит из 1 и 3 цифр исходного числа, второе – из 2-й и 3-й. Например, 765 -> 75 и 65

2.2. Управляющие конструкции в Racket

Краткие теоретические сведения

Для определения локальных переменных используется конструкция `let`.

```
(let ((<пер1> <выр1>)
      (<пер2> <выр2>)
      ...
      (<перn> <вырn>))
  <тело>)
```

Область определения переменной, введенной в `let`-выражении — тело `let`.

Пример 1. Создать функцию, которая вычисляет сумму цифр заданного двузначного числа.

Решение

```
(define (sum_zifr n)
  (let ((a (quotient n 10)) ; определяем значение переменной a (число
    десятков)
        (b (remainder n 10))) ; определяем значение переменной b (число
    единиц)
    (+ a b))) ; тело - сумма цифр (возвращаемое значение функции)
```

Для организации выбора из нескольких альтернативных действий используется конструкция `cond` (от английского слова *conditional*, «условный»).
Общий вид:

```
(cond (<p1> <e1>)
      (<p2> <e2>)
      ...
      (<pn> <en>)
      Else <e>)
)
```

$p_1, p_2, \dots, p_n$ — предикаты, определяющие условия выбора

$e_1, e_2, \dots, e_n$ — действия

Для выбора из двух возможных вариантов можно использовать конструкцию if:

(if <предикат> <следствие> <альтернатива>)

Предикат - это процедура или выражение, которые имеют значением истину или ложь.

Если в ветви необходимо выполнить несколько команд подряд, то их следует заключить в блок «begin». Например,

```
(if (< 1 3)
  (begin
    (display "one line")
    (newline)
    (display "two line")
    (- 1 3)
  )
  (+ 1 3)
)
```

Для записи логических выражений в Racket используются следующие логические отношения:

1) Равно: (= t1 t2 ...tn)

Принимает значение true (#t), если для всех параметров выполняются отношения $t1 = t2 = \dots = tn$. В противном случае принимает значение false (#f).

Например,

```
(= 3 3 3)
```

Результат: #t

```
(= 2 2 3)
```

Результат: #f

2) Меньше: (<t1 t2 ... tn)

Принимает значение true (#t), если для всех параметров выполняются отношения $t1 < t2 < \dots < tn$. В противном случае принимает значение false (#f).

Например,

```
(< 1 2 3 4)
```

Результат: #t

```
(< 1 2 2 3)
```

Результат: #f

3) Меньше либо равно: (<=t1 t2 ... tn)

Принимает значение true (#t), если для всех параметров выполняются отношения $t1 \leq t2 \leq \dots \leq tn$. В противном случае принимает значение false (#f).

Например,

```
(<= 1 2 2 3)
```

Результат: #t

(<= 2 1 3)

Результат: #f

4) Больше: (> t1 t2 ... tn)

Принимает значение true (#t), если для всех параметров выполняются отношения $t1 > t2 > \dots > tn$. В противном случае принимает значение false (#f).

Например,

(> 7 3 2 0)

Результат: #t

(> 2 1 3)

Результат: #f

5) Больше либо равно: (>= t1 t2 ... tn)

Принимает значение true (#t), если для всех параметров выполняются отношения $t1 \geq t2 \geq \dots \geq tn$. В противном случае принимает значение false (#f).

Например,

(>= 7 3 3 0)

Результат: #t

(> 2 1 3)

Результат: #f

Для записи сложных логических выражений используются логические связки not, and, or.

Например,

> (not (= 3 4))

Результат: #t

> (and (> 2 4) (< 1 5))

Результат: #f

> (or (> 3 1) (< 5 4))

Результат: #t

Пример 2. Создать функцию, которая увеличивает заданное двузначное число на 10, если сумма его цифр четна. Если нечетна, то увеличить число вдвое.

Решение

Для решения используем функцию, созданную в примере 1.

; процедура, вычисляющая сумму цифр

(define (sum_zifr n)

(let ((a (quotient n 10))

(b (remainder n 10))))

(+ a b)))

; процедура, преобразующая заданное число a.

(define (chet a)

(if (= (remainder (sum_zifr a) 2) 0) ; если сумма цифр числа четна

$(+ a 10)$; то увеличиваем число на 10
 $(* a 2))$; иначе увеличиваем число вдвое

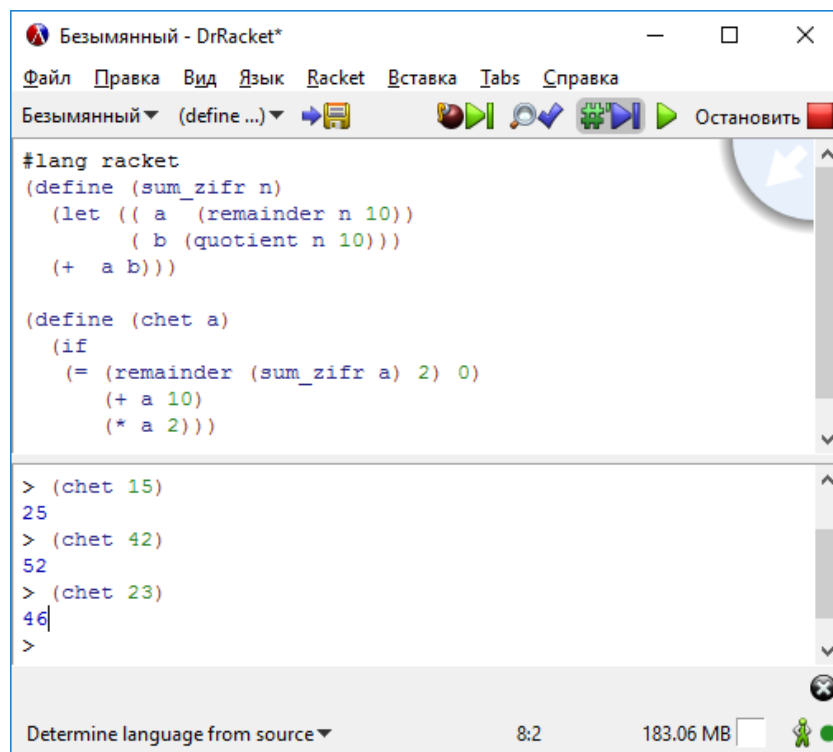


Рис. 1 Результат выполнения программы примера 2

Пример 3. Вычисление корней квадратного уравнения

Решение

```
(define (sq-roots a b c)
  (let ((D (- (* b b) (* 4 a c))))
    (if (< D 0)
        (display "Нет корней")
        (let ((sqrtD (sqrt D)))
          (let ((x1 (/ (- (- b) sqrtD) (* 2.0 a)))
                (x2 (/ (+ (- b) sqrtD) (* 2.0 a))))
            (list x1 x2))))))
```

Здесь вычисленные значения x_1 и x_2 помещаются в список с помощью `list`.

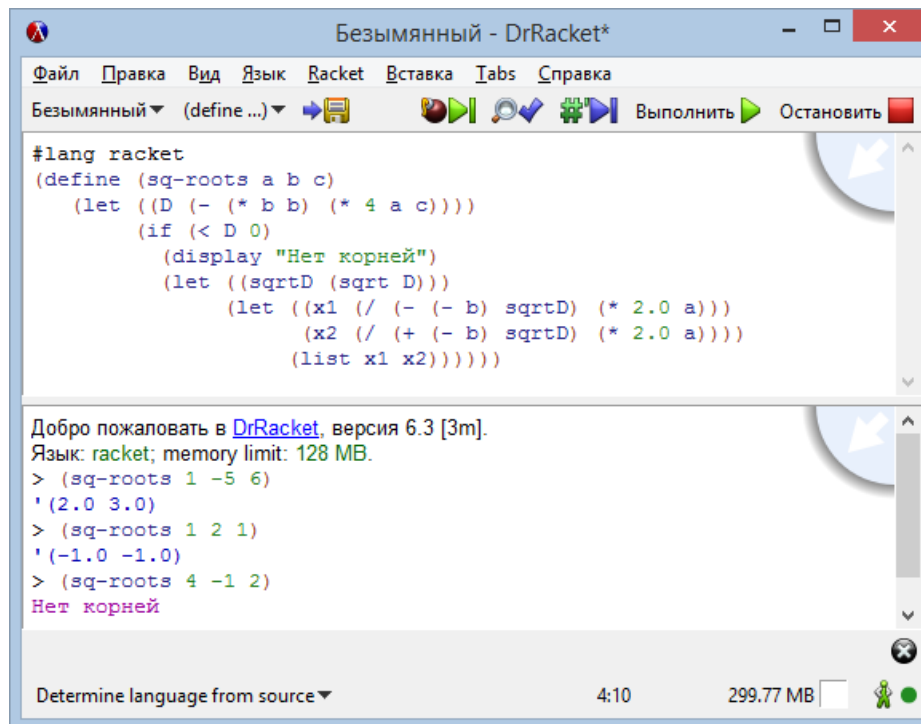


Рис. 2 Результат выполнения программы примера 3

Если код необходимо выполнить только в случае истины, можно использовать конструкцию `when`

(when <условие> <действие>)

Например,

(when (< 1 3) "true")

Для организации циклов используется конструкция `for`

**(for ([<параметр> (in-range <нач.зн> <кон.зн> <шаг>))
(тело цикла))**

Примеры использования:

- **(for ([i (in-range 10)])
(display i))**
Результат:
0 1 2 3 4 5 6 7 8 9
- **(for ([i (in-range 1 10 3)])
(display i))**
Результат:
1 4 7
- **(for ([i (in-range 1 10)])
(display i))**
Результат:
1 2 3 4 5 6 7 8 9

- **(for ([i '(1 2 3)])**
(displayln i))
 Результат:
 1
 2
 3

Пример 4. Вывести последовательность двузначных чисел, в записи которых хотя бы одна цифра четная

Решение

```
(define (posl)
  (for ([n (in-range 10 99)]) ; перебираем целые числа от 10 до 99
    (let ((a (quotient n 10)) ; определяем значение переменной a (число десятков)
          (b (remainder n 10)) ; определяем значение переменной b (число единиц)
        )
      (when (or (= (remainder a 2) 0) (= (remainder b 2) 0)) ; если хотя бы одна из
цифр четная
        (display (format "~a " n)) ; то выводим число
        )))
(posl) ; вызов функции
```

Здесь функция `format` задает формат вывода. Параметр `"~a "` определяет, что после вывода `n` будет выводиться пробел

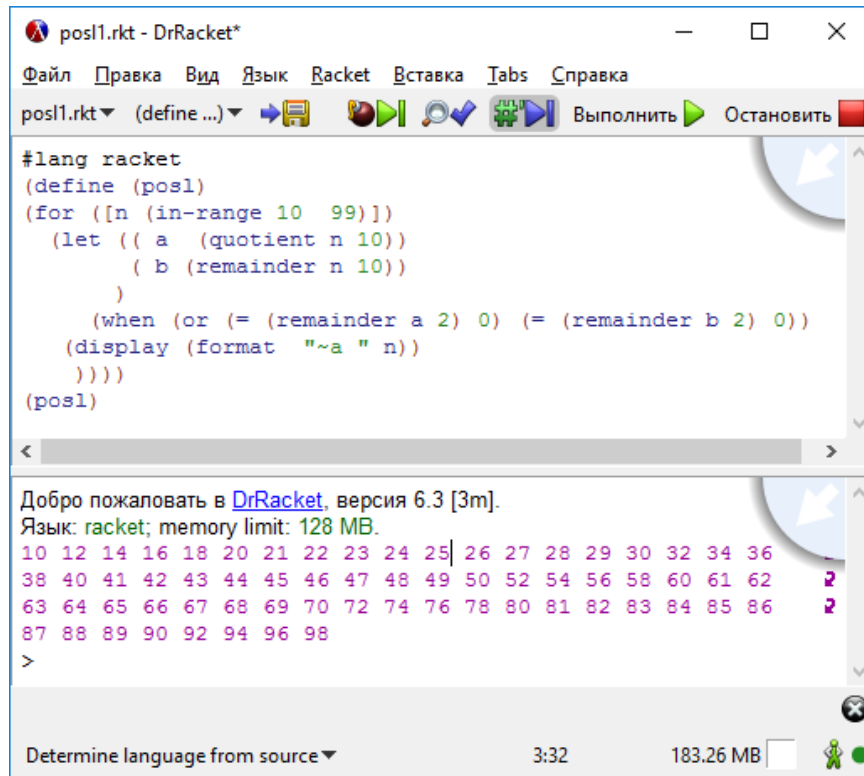


Рис. 3 Результат выполнения программы примера 4

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Создать функцию, которая увеличивает заданное число вдвое, если число нечетное. Если четное, то число уменьшить вдвое.
2. Создать функцию, которая увеличивает заданное двузначное число втрое, если сумма его цифр нечетна. Если сумма цифр четна, то число оставить без изменения.
3. Создать функцию, которая увеличивает заданное число на 100, если число принадлежит интервалу $[0, 20]$.
4. Создать функцию, которая увеличивает заданное 3-значное число на 1, если в его записи есть цифра 5. Если цифры 5 нет, то число уменьшить на 1.
5. Создать функцию, которая увеличивает заданное 3-значное число на 100, если сумма его цифр нечетна. Если сумма цифр четна, то число уменьшить на 100.
6. Вывести последовательность двузначных чисел, в записи которых обе цифры нечетные
7. Вывести последовательность двузначных чисел, у которых сумма цифр четна
8. Вывести последовательность двузначных чисел, у которых сумма цифр принадлежит интервалу $[10, 15]$
9. Вывести последовательность 3-значных чисел, в записи которых хотя бы одна цифра четная

10. Вывести последовательность 3-значных чисел, кратных 11.

2.3.Реализация рекурсии в Racket

Краткие теоретические сведения

Организовать цикл можно с помощью рекурсии, которая является одним из основных способов решения задач в языках функционального программирования.

Пример 5. Процедура, вычисляющая факториал заданного натурального n

Решение

(define (fact n) ; объявляем функцию

(if (= n 1) ; если $n=1$, то функция принимает значение 1

1

(n (fact (- n 1)))) ; иначе вызываем функцию с параметром $n-1$, а*

затем ее значение умножаем на n .



Рис. 4 Результат выполнения программы примера 5

Пример 6. Вычислить сумму квадратов целых чисел от a до b .

Решение

(define (sum-sqr a b) ; объявляем функцию sum-sqr с параметрами a и b

(if (> a b) ; если a больше b , то функция принимает значение 0

0

$(+ (\text{sqr } a) (\text{sum-sqr } (+ a 1) b))))$; иначе вычисляется сумма квадратов от $a+1$ до b и к ней прибавляется a^2 .

The screenshot shows the DrRacket IDE window titled "Безымянный - DrRacket*". The menu bar includes "Файл", "Правка", "Вид", "Язык", "Racket", "Вставка", "Tabs", and "Справка". The toolbar contains icons for saving, running, stepping through code, and other IDE functions. The main text area contains the following Racket code:

```
#lang racket
(define (sum-sqr a b)
  (if (> a b)
      0
      (+ (sqr a) (sum-sqr (+ a 1) b))))
```

The bottom pane shows the execution results:

```
> (sum-sqr 1 3)
14
> (sum-sqr 2 2)
4
> (sum-sqr 4 3)
0
> |
```

The status bar at the bottom indicates "Determine language from source", "9:2", and "204.37 MB".

Рис. 5 Результат выполнения программы примера 6

ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Создать рекурсивную функцию для вычисления произведения целых чисел от a до b .
2. Создать рекурсивную функцию для вычисления произведения целых чисел от a до b .
3. Создать рекурсивную функцию для вычисления суммы целых чисел от a до b .
4. Создать рекурсивную функцию для вычисления произведения N целых чисел, начиная с a .
5. Создать рекурсивную функцию для вычисления суммы N целых чисел, начиная с a .
6. Вычислить сумму $1+2+3+\dots+N$.
7. Подсчитать сумму ряда целых четных чисел от 2 до N .
8. Вычислить сумму ряда целых нечетных чисел от 1 до n .
9. Найти значение произведения: $1*3*5*\dots*11$
10. Вычислить значение n -го члена ряда Фибоначчи: $f(0)=0$, $f(1)=1$, $f(n)=f(n-1)+f(n-2)$.

2.4. Списки и атомы

Краткие теоретические сведения

Любые структуры данных строятся из более простых составляющих, простейшие из которых – элементарные данные. В Лиспе элементарные данные называют *атомами*. **Атом** – это последовательность из букв и цифр, начинающаяся с буквы.

Примеры атомов:

A

Nil

ATOM

LISP

Занятие2

Новый_год

Типичная форма записи символьных выражений называется списочной записью (list-notation). Элементы списка могут быть любой природы.

Список – это перечень произвольного числа элементов, разделенных пробелами, заключенный в круглые скобки. Список – это основной тип данных в Lisp. Пустой список обозначается парой скобок - (). Для обозначения пустого списка используется также специальная константа null.

Примеры списков:

(1 2 3 4 5) ;пятиэлементный список

(1 2 ((3) 4) 5) ;четыреэлементный список

((1 2 3 4 5)) ;одноэлементный список

В Racket есть также специальная процедура для построения списков (**list <a1> <a2> <a3> ...<an>**)

Например,

(list 1 2 3 4) -> '(1 2 3 4)

(list) -> '()

(list a b) -> ошибка

list строит список из значений символов a и b, а не из них самих, поэтому, чтобы создать список символов, нужно их записать с использованием символа ‘

(list 'a 'b) -> '(a b)

Задать список можно также с помощью символа ‘.

Например,

'(a b c) -> '(a b c)

Разницу в описании списков с помощью list и символа ‘ можно проиллюстрировать следующими примерами:

'((+ 1 2) (* 2 3)) -> '((+ 1 2) (* 2 3))

```
(list (+ 1 2) (* 2 3)) -> '(3 6)
```

```
('('a (+ 3 4) 'c) -> '(a (+ 3 4) 'c)
```

```
(list 'a (+ 3 4) 'c) -> '(a 7 c)
```

Первый элемент списка называется головой списка, все прочие элементы, кроме первого, представленные как список, называются хвостом списка.

Функции для работы со списками

- **(car list)** - отделяет голову списка

```
(car '(пн вт ср чт пт сб вс)) -> 'пн
```

```
(car '((1 2) 3 4)) -> '(1 2)
```

```
(* 10 (car '(1 2 3))) -> 10
```

- **(cdr list)** - отделяет хвост списка

Например,

```
(cdr '(пн вт ср чт пт сб вс)) -> '(вт ср чт пт сб вс)
```

```
(cdr '(1 (2 3) 4)) -> '((2 3) 4)
```

При помощи процедур car и cdr можно получить любой элемент списка

Например, третий элемент списка x задается выражением:

```
(car (cdr (cdr x)))
```

Пусть x есть список(A B C D). Тогда(cdr x) есть(B C D) , а (cdr(cdr x)) есть(C D). Таким образом, (car cdr (cdr x))) есть элемент с, который является третьим в исходном списке.

- **(cons head tail)** - соединяет элемент и список в новый список, где присоединенный элемент становится головой нового списка;

Например,

```
(cons 5 '(1 2 3 4)) -> '(5 1 2 3 4)
```

```
(cons '(1 (2 3)) '(4)) -> '((1 (2 3)) 4)
```

```
(cons '() '(1 2 3)) -> '(() 1 2 3)
```

Для выполнения математических операций (сложения, умножения,..) над всеми элементами списка можно применить процедуру **apply**:

(apply <функция> <список>)

Например,

```
(apply + '(1 2 3)) -> 6
```

```
(apply + 1 2 '(3)) -> 6
```

```
(apply + '())-> 0
```

Упорядочить элементы списка можно с помощью процедуры sort:

(sort <список> <направление сортировки>)

Направление сортировки - по возрастанию <, по убыванию >

Например,

```
(sort '(2 5 1 3 7) >) -> '(7 5 3 2 1)
```

```
(sort '(2 5 1 3 7) <) -> '(1 2 3 5 7)
```

```
(sort '(2 5 2 3 7) <) -> '(2 2 3 5 7)
```

(append list ...) – объединение списков, при этом происходит перераспределение их внутренней структуры (таким образом, чтобы результат удовлетворял определению списка).

```
(append '(x) '(y)) -> '(x y)
```

```
(append '(a) '(b c d)) -> '(a b c d)
```

```
(append '(a (b)) '((c))) -> '(a (b) (c))
```

```
(append '(a b) '(c . d)) -> '(a b c . d)
```

(reverse list) – возвращает список, перераспределенный в обратном порядке.

```
(reverse '(a b c)) -> (c b a)
```

```
(reverse '(a (b c) d (e (f)))) -> ((e (f)) d (b c) a)
```

(list-tail list k) – возвращает список на основе списка list, но без k первых элементов. Входящий список должен содержать не менее k элементов.

Например,

```
(list-tail '(1 2 3 4) 2) -> '(3 4)
```

```
(list-tail '(1 2 3 4) 4) -> '()
```

(list-ref list k) – возвращает k-й элемент списка. В списке должно быть не менее k элементов.

Нумерация элементов списка осуществляется от нуля.

Например,

```
(list-ref '(1 2 3 4) 1) -> 2
```

```
(list-ref '(1 2 3 4) 3) -> 4
```

Для превращения одного или нескольких списков в другой путем последовательного применения заданной операции к каждому элементу используется функция **map**. Например,

```
(map (lambda (list)
```

```
  (+ 1 list))
```

```
  '(1 2 3 4))
```

Результат: '(2 3 4 5) – список, полученный увеличением каждого элемента исходного списка на 1

```
(map (lambda (list)
```

```
  (* 2 list))
```

```
  '(1 2 3 4))
```

Результат: '(2 4 6 8) – список, полученный увеличением каждого элемента исходного списка вдвое

```
(map (lambda (list1 list2)
      (+ list1 list2))
      '(1 2 3 4)
      '(3 4 5 6))
```

Результат: '(4 6 8 10) – список, полученный поэлементным суммированием двух заданных списков

```
(map (lambda (list1 list2)
      (- list1 list2))
      '(10 20 30 40)
      '(3 4 5 6))
```

Результат: '(7 16 25 34) - список, полученный вычитанием соответствующих элементов двух заданных списков

Основной метод обработки списков – рекурсия.

Пример 1.

Создать функцию, которая выводит каждый элемент списка с новой строки

Решение

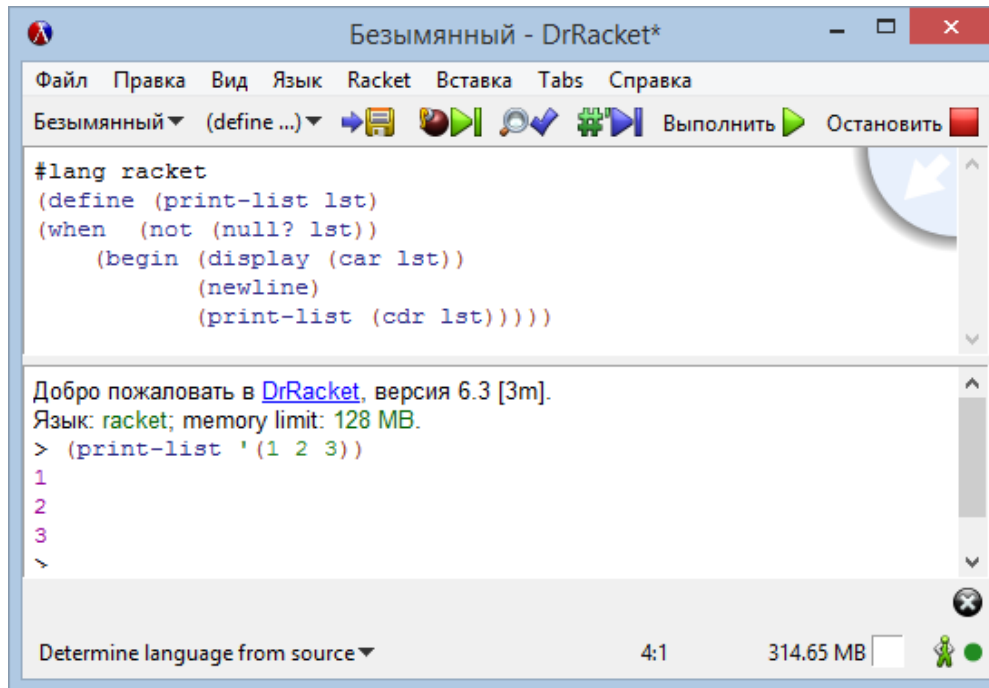
Имя функции print-list. Единственный параметр — исходный список.

Алгоритм работы функции следующий:

Если список не пуст, то:

1. выводим голову списка.
2. вызываем print-list для хвоста списка

```
(define (print-list lst)
  (when (not (null? lst))
    (begin (display (car lst))
            (newline)
            (print-list (cdr lst)))))
```



Пример 2.

Создать функцию, которая определяет количество элементов списка .

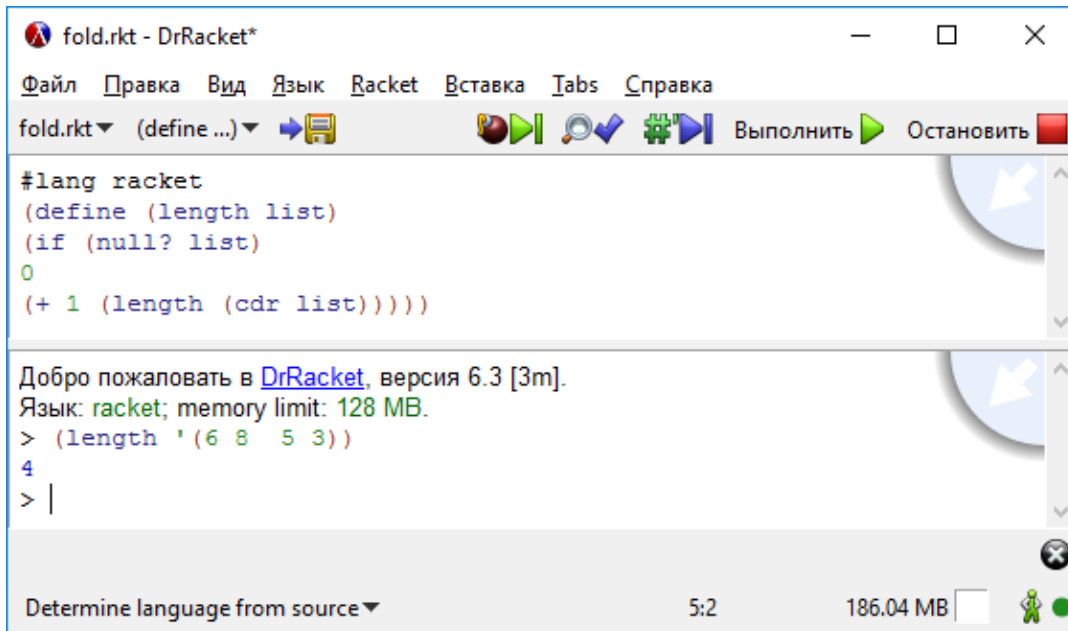
Решение

Длина любого списка равняется 1 плюс длина cdr этого списка.

Этот шаг последовательно применяется, пока мы не достигнем базового случая:

Длина пустого списка равна 0.

```
(define (length list)
  (if (null? list)
      0
      (+ 1 (length (cdr list)))))
```



Пример 3.

Создать функцию, которая берет в качестве аргументов список и число n и возвращает n -й элемент списка.

Решение

Обычно элементы списка нумеруют, начиная с 0. Метод вычисления `list-ref` следующий:

- Если $n = 0$, `list-ref` должна вернуть `car` списка.
- В остальных случаях `list-ref` должна вернуть $(n - 1)$ -й элемент от `cdr` списка.

```
(define (list-ref list n)
  (if (= n 0)
      (car list)
      (list-ref (cdr list) (- n 1))))
```

```
#lang racket
(define (list-ref list n)
  (if (= n 0)
      (car list)
      (list-ref (cdr list) (- n 1))))
```

Добро пожаловать в [DrRacket](#), версия 6.3 [3m].
Язык: racket; memory limit: 128 MB.

```
> (list-ref '(4 6 8 2) 2)
8
>
```

Determine language from source ▼ 5:2 186.04 MB

Пример 4.

Создать функцию для вычисления суммы элементов списка.

Решение

- Если список пустой, то сумма равна 0.
- В остальных случаях вычисляется сумма элементов хвоста, а затем к ней прибавляется голова списка.

```
(define (sum list)
  (if (null? list)
      0
      (+ (car list) (sum (cdr list)))))
```

```
#lang racket
(define (sum list)
  (if (null? list)
      0
      (+ (car list) (sum (cdr list)))))
```

Язык: racket; memory limit: 128 MB.

```
> (sum '(1 2 3 4))
10
> (sum '(1 2 3 4 5))
15
> |
```

Determine language from source ▼ 7:2 186.04 MB

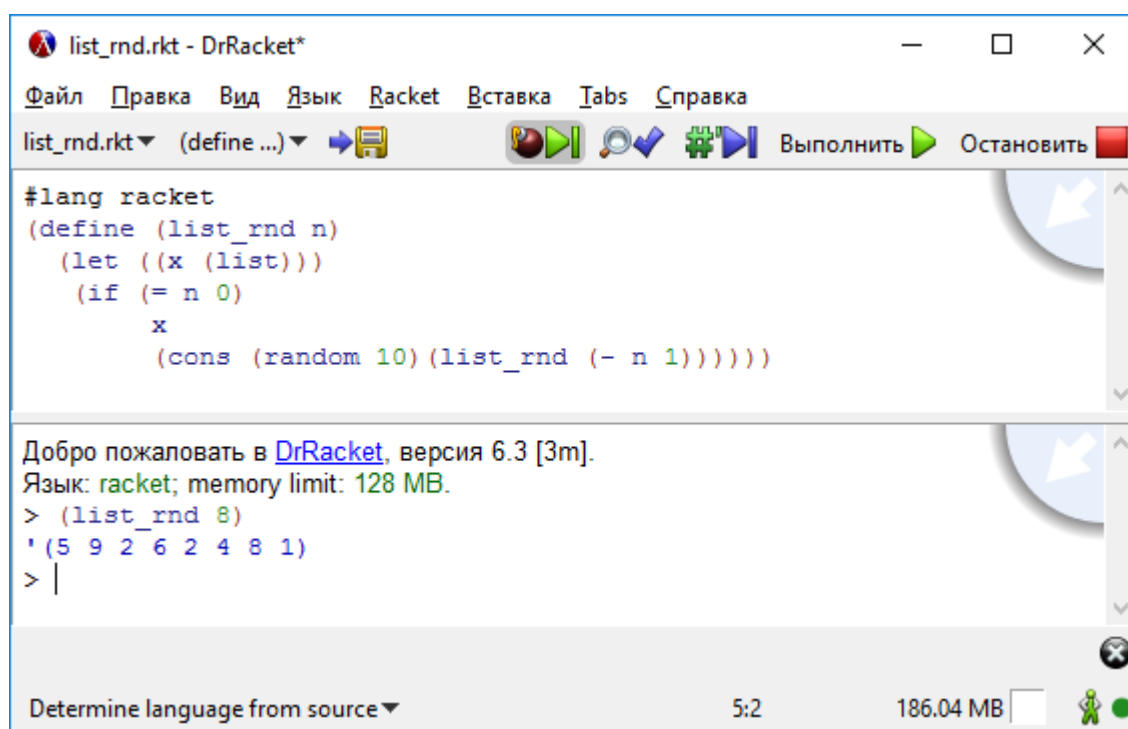
Пример 5.

Создать функцию, формирующую список из n случайных чисел, которые выбираются из интервала от 0 до 10.

Решение

- Если $n=0$, то список пустой.
- В остальных случаях формируем список из $(n-1)$ элемента и объединяем его со случайным числом

```
(define (list_rnd n)
  (let ((x (list)))
    (if (= n 0)
        x
        (cons (random 10) (list_rnd (- n 1))))))
```



ЗАДАНИЯ ДЛЯ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Создать функцию, формирующую список из n случайных чисел, которые выбираются из интервала от 0 до 10 и удалить из него введенное число.
2. Сформировать списки $[1, 3, 5, 7, 9]$ и $[2, 4, 6, 8, 10]$ и объединить их в один.
3. Создать функцию, формирующую список из n случайных чисел, которые выбираются из интервала от 0 до 100 и вставить в него введенное число.
4. Сформировать список из N натуральных чисел, начиная с 10. Каждое следующее на 5 больше предыдущего.
5. Создать функцию, формирующую список из n случайных чисел, которые выбираются из интервала от 0 до 60 и найти сумму его элементов

6. Сформировать список [6, 5, 4, 3, 2] и найти сумму его элементов
7. Сформировать список [7, 5, 3, 1] и найти произведение его элементов
8. Сформировать список из N последовательных натуральных чисел, начиная с 10. Найти сумму его элементов

Литература

1. Адаменко А., Кучуков А. Логическое программирование и Visual Prolog.- СПб, 2003
2. Братко И. Программирование на языке ПРОЛОГ для искусственного интеллекта.- М., 1990.
3. Козырева Г.Ф. Практикум решения задач в среде Visual Prolog. Учебно-методическое пособие для студентов, обучающихся по специальности «информатика». – Армавир, АГПУ – 2005, 65с.
4. Козырева Г.Ф. Учебно-методическое пособие «Контрольные работы по курсу «Интеллектуальные информационные системы»» для студентов заочной формы обучения. – Армавир, АГПУ. – 2009. – 34 с.
5. Сергиевский Г. М. Функциональное и логическое программирование : учеб. пособие / Г. М. Сергиевский, Н. Г. Волченков. – М. : Академия, 2010. – 317 с.
6. Сошников Д. В. Парадигма логического программирования / Д. В. Сошников. – М. : Вузовская книга, 2006. – 220 с.