СРЕДА РАЗРАБОТКИ VISUAL PROLOG

На сегодняшний день Visual Prolog, несомненно, является одной из наиболее мощных сред реализаций языка Prolog. Будучи наследником системы Turbo Prolog, он является компилирующей средой и основан на строгой статической типизации данных. В этом его принципиальное отличие от стандартной реализации языка Prolog. Кроме этого в его составе присутствуют средства визуального и объектно–ориентированного программирования, императивные конструкции, коллекторы списков и многое другое.

***Основные разделы Visual Prolog – программ***

Обычно программа на Visual Prolog состоит из четырех основных программных разделов. К ним относятся:

- **раздел clauses (предложений);**

- **раздел predicates (предикатов);**

- **раздел domains (доменов);**

- **раздел goal (целей).**

***Раздел предложений (clauses).***

В раздел clauses (предложений) помещаются все факты и правила, составляющие программу.

*•Факты* — это отношения или свойства, о которых известно, что они имеют значение "истина".

*Факт* представляет либо свойство объекта, либо отношение между объектами.

Факт самодостаточен. Для подтверждения факта не требуется дополнительных сведений, и факт может быть использован как основа для логического вывода.

Факт в Visual Prolog состоит из имени отношения и объекта или объектов, заключенных в круглые скобки. Факт завершается точкой (.).

Т.е. предложение на естественном языке Билл любит собак. (Bill likes

dogs) на синтаксисе Visual Prolog будет выглядеть likes (bill, dogs).

Факты помимо отношений, могут выражать и свойства. Так, например,

предложение естественного языка "Grass is green" (Трава зеленая) на Visual Prolog, выражая те же свойства, выглядит следующим образом: green (grass)

• *Правила —* это связанные отношения; они позволяют логически выводить одну порцию информации из другой.

Правило принимает значение "истина", если доказано, что заданный набор условий является истинным.

*Правило —* это свойство или отношение, которое достоверно, когда известно, что ряд других отношений достоверен. Синтаксически эти отношения разделены запятыми.

Все правила имеют *2* части: заголовок и тело, разделенные специальным знаком :-.

• *Заголовок —* это факт, который был бы истинным, если бы были истинными несколько условий. Это называется *выводом* или зависимым отношением.

• *Тело* — это ряд условий, которые должны быть истинными, чтобы можно было доказать, что заголовок правила истинен.

Ниже представлен обобщенный синтаксис правила в Visual Prolog:

заголовок: - <Подцель>, <Подцель>,..., <Подцель>.

Тело правила состоит из одной или более подцелей. Подцели разделяются запятыми, определяя конъюнкцию, а за последней подцелью правила следует точка.

Каждая подцель выполняет вызов другого предиката Visual Prolog, который может быть истинным или ложным. После того, как программа осуществила этот вызов, Visual Prolog проверяет истинность вызванного предиката, и если это так, то работа продолжается, но уже со следующей подцелью. Если же в процессе такой работы была достигнута точка, то все правило считается истинным; если хоть одна из подцелей ложна, то все правило ложно.

Ниже представлены правила, соответствующие связи "любить" (likes):

Синди любит все, что любит Билл. (Cindy likes everything that Bill likes)

Кейтлин любит все зеленое. (Caitlin likes everything that is green)

Используя эти правила, можно из предыдущих фактов найти некоторые вещи, которые любят Синди и Кейтлин.

Чтобы перевести эти правила на Пролог, нужно немного изменить синтаксис, например:

likes (cindy, Something):- likes (bill, Something).

likes (caitlin, Something):- green (Something).

Символ **:–** имеет смысл "если" (if). Однако if Пролога отличается от if, написанного в других языках, например в Pascal, где условие, содержащееся в операторе if, должно быть указано перед телом оператора, который может быть выполнен. Данный тип оператора известен как *условный оператор если/тогда* (if/then).

Visual Prolog использует другую форму логики в таких правилах. Вывод об истинности заголовка правила Пролога делается, если (после того, как) тело этого правила истинно, т.е *правило Пролога соответствует условной форме тогда, если (then/if).*

Все предложения для каждого конкретного предиката в разделе clauses должны располагаться вместе. Последовательность предложений, описывающих один предикат, называется *процедурой.*

***Раздел предикатов (predicates).***

Если в разделе clauses программы на Visual Prolog вы описали собственный предикат, то вы обязаны объявить его в разделе predicates (предикатов); в противном случае Visual Prolog не поймет, о чем вы ему "говорите". В результате объявления предиката вы сообщаете, к каким доменам (типам) принадлежат аргументы этого предиката.

Visual Prolog поставляется с большим набором встроенных предикатов (их не нужно объявлять*),* а интерактивное справочное руководство предоставляет полное их описание.

Предикаты задают факты и правила. В разделе же predicates все предикаты просто перечисляются с указанием типов (доменов) их аргументов.

Объявление предиката начинается с имени этого предиката, за которым идет открывающая (левая) круглая скобка, после чего следует ноль или больше доменов (типов) аргументов предиката:

predicatesName (argument\_typel OptionalNamel,

argument\_type2 OptionalName2,...,

argument\_typeN OptionalName3)

После каждого домена (типа) аргумента следует запятая, а после последнего типа аргумента - закрывающая (правая) скобка. Отметим, что, в отличие от предложений в разделе clauses, декларация предиката не завершается точкой. Доменами (типами) аргументов предиката могут быть либо стандартные домены, либо домены объявленные вами в разделе domains.

Имя предиката должно начинаться с буквы, за которой может располагаться последовательность букв, цифр и символов подчеркивания. Регистр букв не имеет значения, однако не следует использовать заглавные буквы в качестве первой буквы имени предиката. Имя предиката может иметь длину до 250 символов.

В именах предикатов запрещается использовать пробел, символ минус, звездочку и другие алфавитно-цифровые символы.

Аргументы предикатов должны принадлежать доменам, известным Visual Prolog. Эти домены могут быть либо стандартными доменами, либо некоторыми из тех, что вы объявили в разделе доменов.

Т.е. если предикат my\_predicate (symbol, integer) объявлен в разделе predicates следующим образом:

predicates

my\_predicate (symbol, integer)

то не нужно в разделе domains декларировать домены его аргументов, т. к. symbol и integer — стандартные домены. Однако если этот же предикат вы объявляете так:

predicates

my\_predicate(name, number),

то необходимо объявить, что name (символический тип) и number (целый тип) принадлежат к стандартным доменам symbol и integer:

domains

name= symbol

number = integer

predicates

my\_predicate(name, number)

***Раздел доменов (domains)***

В традиционном Прологе есть только один тип — *терм.* В Visual Prolog мы объявляем домены всех аргументов предикатов.

Домены позволяют задавать разные имена различным видам данных, которые, в противном случае, будут выглядеть абсолютно одинаково. В программах Visual Prolog объекты в отношениях (аргументы предикатов) принадлежат доменам, причем это могут быть как стандартные, так и описанные вами специальные домены.

Раздел domains служит двум полезным целям. Во-первых, вы можете задать доменам осмысленные имена, даже если внутренне эти домены аналогичны уже имеющимся стандартным. Во-вторых, объявление специальных доменов используется для описания структур данных, отсутствующих в стандартных доменах.

Иногда очень полезно описать новый домен — особенно, когда вы хотите прояснить отдельные части раздела predicates. Объявление собственных доменов, благодаря присваиванию осмысленных имен типам аргументов, помогает документировать описываемые вами предикаты.

Покажем, как объявление доменов помогает документировать предикаты:

Франк — мужчина, которому 45 лет.

Используя следующие домены, можно так объявить соответствующий предикат:

domains

name, gender = symbol

age = integer

predicates

person(name, gender, age)

***Раздел цели (goal)***

Этот раздел аналогичен телу правила: это просто список подцелей. Цель отличается от правила лишь следующим:

- за ключевым словом goal не следует **:-**;

- при запуске программы Visual Prolog автоматически выполняет цель.

Это происходит так, как будто Visual Prolog вызывает goal, запуская тем самым программу, которая пытается разрешить тело правила goal. Если все подцели в разделе goal истинны, — программа завершается успешно. Если же какая-то подцель из раздела goal ложна, то считается, что программа завершается неуспешно (хотя чисто внешне никакой разницы в этих случаях нет, — программа просто завершит свою работу).

Пример: Однократно дав языку Visual Prolog несколько фактов, мы можем задавать вопросы, касающиеся отношений между ними. На естественном языке мы спрашиваем:

Does Bill like dogs? (Билл любит собак?).

По правилам Пролога мы спрашиваем:

likes (bill, dogs).

Получив такой запрос, Visual Prolog ответит: yes (да), потому что Visual Prolog имеет факт, подтверждающий, что это так.

***Другие разделы программ***

Теперь, когда вы ознакомились с такими разделами программ Visual Prolog, как clauses, predicates, domains и goal, поговорим о некоторых других, часто используемых разделах программ: facts, constants и различных глобальных (global) разделах.

***Раздел фактов (facts)***

Программа на Visual Prolog представляет собой набор фактов и правил. Иногда в процессе работы программы бывает необходимо модифицировать (изменить, удалить или добавить) некоторые из фактов, с которыми она работает. В этом случае факты

рассматриваются как *динамическая* или *внутренняя* база данных, которая при выполнении программы может изменяться. Для объявления фактов программы, рассматривающихся как части динамической (или изменяющейся) базы данных, Visual Prolog включает специальный раздел — facts.

Следует отметить два ограничения, объявленные в разделе facts :

• в динамической базе данных Пролога могут содержаться только факты;

• факты базы данных не могут содержать свободные переменные.

Допускается наличие нескольких разделов facts , тогда в описании каждого раздела facts нужно явно указать его имя, например facts mydatabase. В двух различных разделах facts нельзя использовать одинаковые имена предикатов. Также нельзя использовать одинаковые имена предикатов в разделах facts и predicates. Если имя базы данных не указывается, то ей присваивается стандартное имя dbasedom. Программа может содержать локальные безымянные разделы фактов, если она состоит из единственного модуля, который не объявлен как часть проекта. Среда разработки компилирует программный файл как единственный модуль только при использовании утилиты TestGoal. Иначе безымянный раздел фактов должен быть объявлен глобальным, то есть как global facts.

В Прологе есть специальные встроенные предикаты для работы с динамической базой данных:

• assert;

• asserta;

• assertz;

• retract;

• retractall;

• save;

• consult.

Предикаты assert, asserta, assertz, - позволяют занести факт в БД, а предикаты retract, retractall - удалить из нее уже имеющийся факт.

Предикат assert заносит новый факт в БД в произвольное место, предикат asserta добавляет новый факт перед всеми уже внесенными фактами данного предиката, assertz добавляет новый факт после всех фактов данного предиката.

Предикат retract удаляет из БД первый факт, который сопоставляется с заданным фактом, предикат retractall удаляет из БД все факты, которые сопоставляются с заданным фактом.

Предикат save записывает все факты динамической БД в текстовый файл на диск, причем в каждую строку файла заносится один факт. Если файл с заданным именем уже существует, то старый файл будет затерт.

Предикат consult записывает в динамическую БД факты, считанные из текстового файла, при этом факты из файла дописываются в имеющуюся БД. Факты, содержащиеся в текстовом файле должны быть описаны в разделе domains.

Пример программы, генерирующей множество 4-разрядных двоичных чисел и записывающей их в динамическую БД имеет вид:

facts

dbin (byte, byte, byte, byte)

predicates

cifra (byte)

bin (byte, byte, byte, byte)

clauses

cifra (0).

cifra (1).

bin (A, B, C, D):- cifra (A), cifra (B), cifra (C), cifra (D),

assert (bin (A, B, C, D)).

goal

bin (A, B, C, D).

***Раздел констант (constants)***

В программах на Visual Prolog можно объявлять и использовать символические константы. Раздел для объявления констант обозначается ключевым словом constants, за которым следуют сами объявления, использующие следующий синтаксис:

<Id> = <Макроопределение>

<Id> – имя символической константы, а <Макроопределение> — это то, что вы присваиваете этой константе. Каждое <Макроопределение> завершается символом новой строки и, следовательно, на одной строке может быть только одно описание константы. Объявленные таким образом константы могут позже использоваться в программах.

Рассмотрим следующий фрагмент программы:

constants

zero = 0

pi = 3.141592653

Перед компиляцией программы Visual Prolog заменит каждую константу на соответствующую ей строку.

На использование символических констант накладываются следующие ограничения:

- описание константы не может ссылаться само на себя:

my\_number = 2\*my\_number/2 % не допускается

это приведет к сообщению об ошибке "Recursion in constant definition" (Peкурсия в описании константы);

- в описаниях констант система не различает верхний и нижний регистры.

Следовательно, при использовании в разделе программы clauses идентификатора типа constants, его первая буква должна быть строчной для того, чтобы избежать путаницы между константами и переменными.

- в программе может быть несколько разделов constants, однако объявление константы должно производиться перед ее использованием;

- идентификаторы констант являются глобальными и могут объявляться только один раз. Множественное объявление одного и того же идентификатора приведет к сообщению об ошибке "Constant identifier can only be declared once" (Идентификатор

константы может объявляться только один раз).

***Глобальные разделы (global)***

Visual Prolog позволяет объявлять некоторые разделы domains, predicates, clauses глобальными (а не локальными); сделать это вы можете, объявив в своей программе специальные разделы global domains, global predicates и global facts.

***Переменные***

В Visual Prolog *переменные* позволяют записывать общие факты и правила и задавать общие вопросы.

Имена переменных в Visual Prolog должны начинаться с заглавной буквы (или с символа подчеркивания), после которой может стоять любое количество букв (заглавных или строчных), цифр или символов подчеркивания. Удобно использовать в названии переменной буквы разного регистра: IncomeAndExpenditureAccount.

В простом запросе, чтобы найти того, кто любит теннис, можно использовать переменные. Например: likes(X, tennis).

В этом запросе буква X используется как переменная для нахождения неизвестного человека.

Осмысленный выбор имен переменных делает программу более удобной для чтения Например:

likes(Person, tennis).

лучше, чем

likes (X, tennis).

потому что Person имеет больше смысла, чем X.

Предложение на английском языке Bill likes the same thing as Kim. (Билл любит то же, что и Ким), может быть записано на Visual Prolog следующим образом:

likes (bill, Thing) :- likes (kim, Thing).

Thing *—* это переменная.

Если вам нужна только определенная информация запроса, можно использовать анонимные переменные для игнорирования ненужных значений. В Visual Prolog анонимные переменные обозначаются символом подчеркивания (\_).

Анонимная переменная может быть использована на месте любой другой переменной и ей никогда не присваивается значение.

Анонимные переменные также можно использовать в фактах.

Следующие факты Пролога:

ows (\_,shoes) .

eats(\_).

могли быть использованы для выражения утверждений на естественном языке:

У каждого есть туфли. (Everyone owns shoes)

Каждый ест. (Everyone eats)

Анонимные переменные сопоставляются с любыми данными.

***Комментарии***

Хорошим стилем программирования является включение в программу комментариев, объясняющих все то, что может быть непонятно кому-то другому (или даже вам, спустя полгода). Многострочные комментарии должны начинаться с символов /\* и завершаться символами \*/. Для установки однострочных комментариев можно использовать либо эти же символы, либо начинать комментарий символом процента (%)*.*

***Сопоставление (matching)***

В Visual Prolog имеется несколько примеров сопоставления одной вещи с другой.

Ясно, что *идентичные структуры сопоставимы* (сравнимы) друг с другом: parent (joe, tammy) сопоставимо с parent (joe, tammy).

Однако *сопоставление (сравнение)* обычно использует одну или несколько свободных переменных. Например, если X свободна, то parent (joe, X) сопоставимо с parent (joe, tammy) и X принимает значение (связывается с) tammy.

Если же X уже связана, то она действует так же, как обычная константа. Таким образом, если X связана со значением tammy, то parent (joe, X) сопоставимо с parent (joe, tammy), но parent (joe, X) не сопоставимо с parent (joe, millie) .

Две свободные переменные могут сопоставляться друг с другом. Например, parent (joe, X) сопоставляется с parent (joe, Y), связывая при этом переменные X и Y между собой. С момента связывания X и Y трактуются как одна переменная, и в любое изменение значения одной из них приводит к немедленному соответствующему изменению другой. В случае подобного "связывания" между собой нескольких свободных переменных все они называются *совмещенными свободными переменными.*

В Прологе связывание переменных (со значениями) производится двумя способами: на входе и выходе. Направление, в котором передаются значения, указываете в *шаблоне потока параметров* (flow pattern). В дальнейшем (для краткости) будем опускать слово "шаблон" и говорить просто "поток параметров". Когда переменная передается в предложение, она считается *входным аргументом* и обозначается символом (i). Когда же переменная возвращается из предложения, она является *выходньм аргументом* и обозначается символом (о).

**Интегрированная среда разработки**

Интегрированная среда разработки (Integrated Development Environment – IDE) используется для создания, разработки и поддержки проектов Visual Prolog. Ее используют на всех этапах жизни проекта для следующих задач:

• Создание – если проект создается средствами IDE, то в момент его создания выбираются основные свойства проекта, такие как: будет ли это исполняемое приложением или DLL, используется текстовое взаимодействие или пользовательский графический интерфейс и т.д.

• Построение – проект компилируется и связывается, используя IDE.

• Просмотр – компилятор и IDE собирают информацию о проекте, которая затем используется для быстрой навигации по проекту, и т.д.

• Разработка. – При разработке и поддержки проекта IDE используется для добавления и удаления в проект исходных файлов и элементов пользовательских интерфейсов, а также их редактирования.

• Отладка. – Отладчик IDE позволяет идти по тексту программы и наблюдать состояния ресурсов программы в ходе ее исполнения.

Знакомство с любой новой системой лучше всего начинать с простого и понятного примера. Разработаем в Visual Prolog с использованием его IDE проект для вычисления факториала некоторого числа. На языке базового Prolog’а вычислительная часть этой задачи может быть представлена двумя правилами на основе предиката factorial(<число>,<результат>):

factorial(N, 1) :- N<1, !.

factorial(N, N\*F1) :- fact(N-1, F1).

Символ «!» в первом правиле обозначает отсечение и предотвращает попытку системы применить второе правило для случая N = 0.

**Разработка консольного проекта**

Среда Visual Prolog’а предоставляет возможность создания двух типов проектов: консольных, которые работают в командной строке Windows, и проектов с графическим интерфейсом, которые являются полноценными Windows приложениями. Знакомство с IDE начнем с создания консольного проекта.

**Этап создания проекта**

Любой тип проекта, который надо разработать, начинается с того, что он должен быть создан. Для этого надо выполнить следующие шаги:

• Запустить саму среду Visual Prolog.

• Выбрать в его главном меню опцию Project –> New. Откроется диалоговое окно (рис. 1), в котором требуется указать имя проекта, выбрать его тип и указать место его хранения на диске. На первых этапах работы с IDE достаточно заполнить только эти сведения и нажать кнопку Finish.

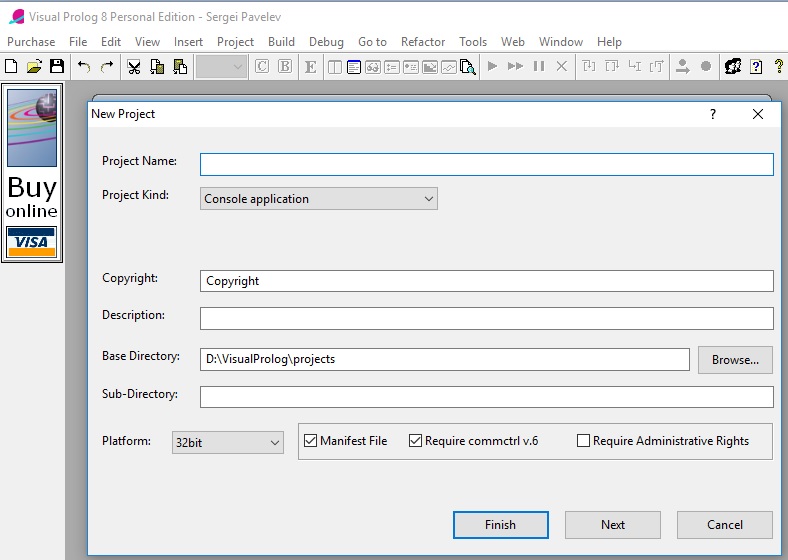


Рис. 5.1. Вид диалогового окна New Project.

Система автоматически создаст проект, выполнит его компиляцию. При этом последовательность действий, выполняемых системой, выводится в диалоговом окне Messages. После окончания этого процесса среда Visual Prolоg будет иметь вид, аналогичный тому, что приведен на рис. 2.

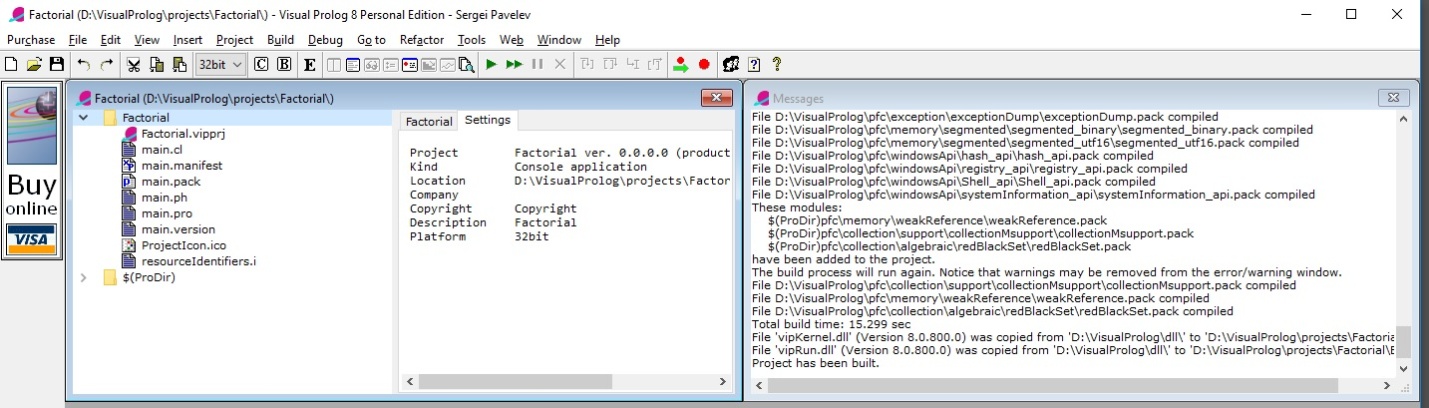


Рис. 2. Среда Visual Prolog по окончании создания проекта Factorial.

**Дерево проекта (Project Tree)**

В процессе создания, сформировался прототип проекта – класс Factorial, с именем, которое было дано проекту. Обратим внимание на дерево этого проекта (Project Tree), которое представлено в Project Window (рис. 3).

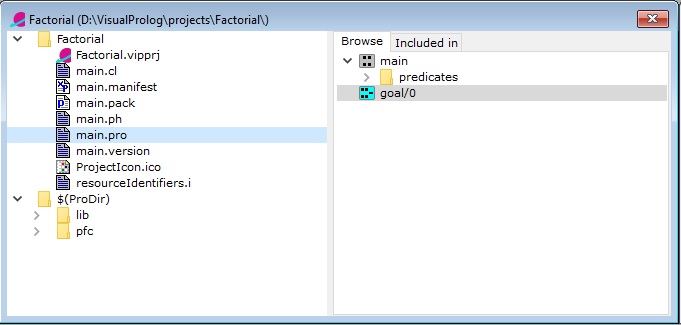


Рис. 3. Вид дерева проекта Factorial.

Само по себе это дерево имеет стандартный для Windows вид, поэтому не будем останавливаться на способах его использования, а рассмотрим его содержание. Самый верхний узел представляет собой проект и проектную директорию. В самом низу расположен логический узел $(ProDir), который представляет папку, в которой установлена сама система Visual Prolog. Эта папка содержит, как видно из рис. 3, библиотеки (lib) и исходные тексты классов системы Visual Prolog (pfc).

Из дерева проекта видно, что в состав проекта IDE включил ряд файлов, для которых в Visual Prolog имеет следующие специальные соглаш-ния:

• .ph – заголовки пакетов (package headers), пакет – это набор классов и интерфейсов, которые используются совместно.

• .pack – содержат исполняемые разделы или конкретизации файлов, перечисленных в соответствующих \*.ph файлах (packages).

• .i – содержит интерфейс (interface).

• .cl – содержит декларацию класса (class declaration)

• .pro – файл содержит имплементацию класса (class implementation).

Существует ряд стандартных действий, которые доступны при работе с деревом проектов. Правая кнопка мышки вызывает контекстное меню с операциями, которые можно выполнить над текущим узлом.

Некоторые сущности представлены в дереве дважды, поскольку они имеют как декларацию, так и имплементацию. Например, предикат run в классе main.

Двойной щелчок на узле вызывает редактор соответствующего элемента. Все коды редактируются с помощью текстового редактора, а ресурсы на основе окон, такие как диалоги, меню, редактируются с помощью графических редакторов. Графические редакторы мы увидим в следующем разделе, сейчас попытаемся вызвать текстовый редактор.

**Этап разработки программного кода проекта**

Если установить фокус на узел с именем класса main.pro, то в правой панели отобразятся все объекты, используемые и описанные в данном классе. При клике мышкой по обозначению этого класса, откроется окно со сгенерированным IDE прототипом программного файла проекта, который приведен слева на рис. 4.

Принимая во внимание, что implementation в переводе с английского – это реализация, приступим к реализации задачи вычисления факториала числа, используя при этом синтаксис и семантику Visual Prolog. Используя полученные ранее знания, убедимся, что для реализации поставленной задачи файл main.pro должен иметь вид, аналогичный тому, что приведен справа на рис. 4 .

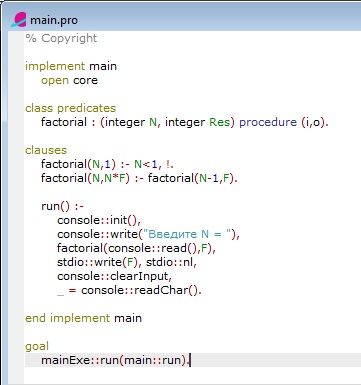
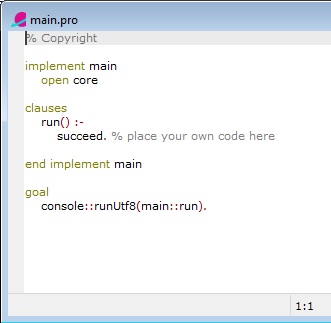


Рис.4. Прототип программного файла проекта Factorial и его реализация.

Следует отметить, что IDE Visual Prolog имеет достаточную развитую систему помощи (Help), контекстно-зависимую справку и подсказки при наборе текста. Полноценную документацию, как встроенную в систему, так и доступную в Интернете, можно получить, используя опцию Help из главного меню. Назначение любой конструкции можно узнать, если на нее переместить курсор и нажать клавишу F1. При наборе тех или иных строк, IDE, после ввода первых символов, выдает подсказки о возможных для ввода конструкциях.

После того, как файл main.pro будет отредактирован, проект полностью готов к работе.

**Этап компиляции и запуска проекта**

Его можно запустить в работу, используя для этого в главном меню Bild –> Run in Window. При выборе этой опции IDE сам выполнит компиляцию проекта, его загрузку и запуск на выполнение. Если не будет выявлено ошибок при компиляции, то IDE запустит копию командного процессора Windows, а в нем наше консольное приложение.

Остается только проверить работу программы, вводя в ответ на запрос программы какое-либо число и получая от нее соответствующий ответ. В этом режиме работы программы последняя строчка с readChar() является лишней. Приходится два раза нажимать какую-либо клавишу. Данная строчка нужна для корректной работы загрузочного модуля проекта вне среды Visual Prolog. Если сейчас обратиться в папку, определенную для нашего проекта, то можно обнаружить, что в ней образовалась еще одна папка с именем Exe, которая содержит наряду с другими файл Factorial.exe. Этот файл и является загрузочным модулем нашего проекта, который может выполняться вне среды Visual Prolog. Закроем среду и дважды кликнем по пиктограмме Factorial, приложение запустится в командной строке и будет готово к приему данных с клавиатуры. При отсутствии в программе readChar() приложение закроется раньше, чем мы увидим ответ.

**Разработка GUI проекта на Visual Prolog**

В этом разделе рассмотрим решение той же задачи, но реализуем ее в виде оконного приложения, которое сможет работать в среде Windows. При реализации этого проекта, как будет видно далее, мало что останется от чистого Prolog. В основном это визуальное программирование в среде Visual Prolog. При разработке проекта создания полноценного Windows приложения надо выполнить набор следующих действий:

• создать в среде Visual Prolog новый GUI проект;

• создать для взаимодействия с пользователем некоторую экранную форму, разместить на этой форме элементы управления, определить необходимые при реализации конкретной задачи свойства, как для самой формы, так и для каждого ее элемента управления;

• обеспечить возможность запуска этой формы из основного окна создаваемого приложения;

• создать внутри проекта класс, методы которого будут осуществлять необходимую обработку информации;

• определить события, возникающие в элементах управления формы, которые будут вызывать те или иные методы.

Применительно к задаче вычисления факториала сформированная выше последовательность действий может быть конкретизирована. Пусть требуется разработать Windows приложение, которое представляет собой стандартное окно с панелью управления и главным меню. Это приложение должно при выборе опции главного меню File –> New открывать экранную форму. На форме должны присутствовать два элемента управления: поле ввода и кнопка. Для кнопки следует определить событие нажатия, по которому запускается процедура вычисления факториала числа, введенного в поле ввода. Нужные для этого процедуры надо организовать в виде класса, включаемого в проект.

Рассмотрим реализацию последовательности всех этих действий, в среде Visual Prolog.

**Создание нового GUI проекта**

Создание проекта выполняется точно так же, как и для консольного приложения. После запуска Visual Prolog надо выбрать в его меню опцию Project –> New и в диалоговом окне New Project указать тип проекта и его имя.

Система автоматически создает проект, компилирует его, и протокол своих действий выводит в диалоговом окне Messages. В процессе создания сформировался прототип проекта – класс FactGUI. Его имя совпадет с тем, которое было задано при создании проекта. Обратим внимание, что набор элементов вновь сформированного проекта в дереве проектов во многом совпадает со случаем консольного проекта.

Существенным отличием является присутствие в составе GUI проектов узла TaskWindows. Если в дереве проекта раскрыть этот узел (рис. 5), то можно увидеть большое количество файлов, которые в Visual Prolog служат для поддержки графических приложений.

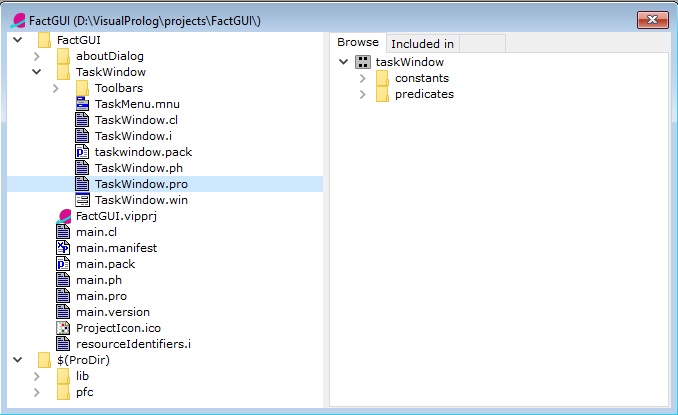


Рис. 5. Объекты Visual Prolog для поддержки GUI проектов.

Среди этих файлов присутствуют файлы из предопределенных в Visual Prolog категорий:

• .dlg файл – содержит окно диалог (dialog);

• .frm файл – содержит экранную форму (form);

• .win файл – содержит окно, которое может быть окном приложения или обычным окном класса window из PFC GUI (window);

• .mnu файл – содержит настройку и конфигурацию меню (menu);

• .ico файл – содержит иконку приложения(icon)

В проектах можно встретить также следующие типы файлов:

• .ctl файлы – содержат описание элементов управления (controls);

• .tb файлы – содержат панели инструментов (toolbars);

• .cur файлы – содержат курсоры (cursors);

• .bmp файлы – содержат картинки (bitmaps);

• .lib файлы – это библиотеки (libraries).

**Добавление формы в пустой проект**

Для того чтобы в проекте создать новую форму необходимо выполнить следующие действия:

• установить в дереве проекта фокус на его вершину – узел FactGUI,

• выбрать опцию File –> New in New Package в главном меню системы и откроется диалоговое окно Create Project Item.

• выбрать в левой панели диалогового окна Create Project Item объект Form, установить параметры, аналогично рис. 6.

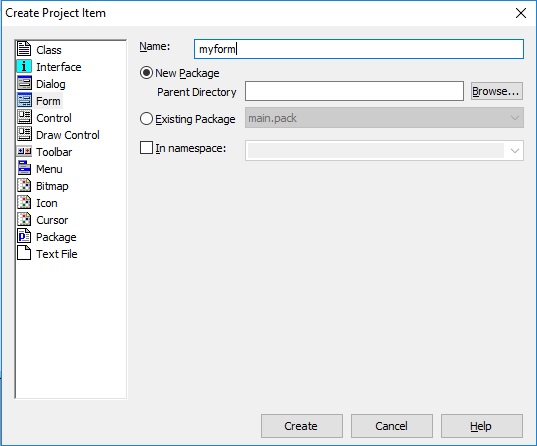


Рис. 6. Определение параметров создаваемой формы.

• После нажатия на кнопку Create диалогового окна Create Project Item, IDE выводит на экран окно прототипа новой формы и еще три окна, предназначенных для создания элементов управления этой формы и настройки их свойств и событий (рис. 7).

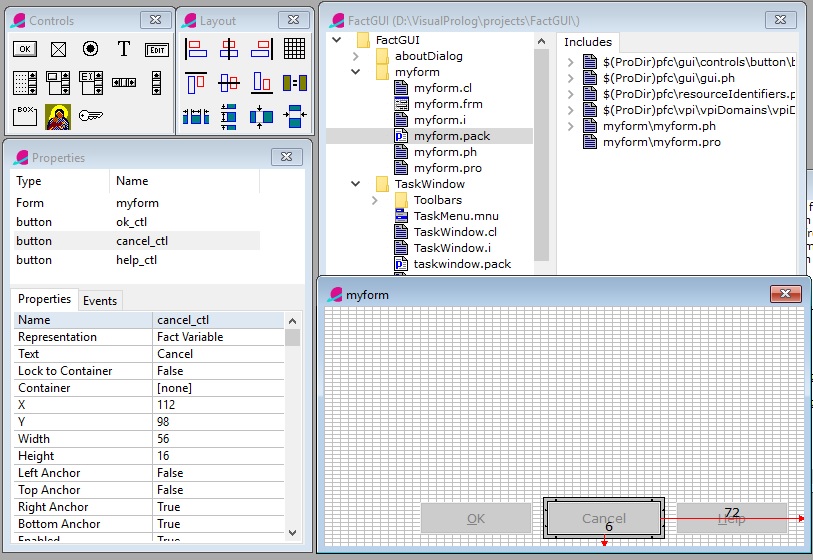


Рис. 7. Конструктор экранной формы.

**Настройка главного меню проекта**

Применительно к нашей задаче настройка меню будет состоять всего из двух действий: открытия доступа к опции меню File –> New и привязка к этой опции меню события открытия выше созданной формы.

Для выполнения этих операции открытия доступа к опции меню главного меню проекта необходимо:

• Найти в дереве проекта элемент TaskMenu.mnu, двойным кликом на нем открыть диалоговое окно TaskMenu.

• Затем, раскрывая узлы дерева, которое отображается в нижней части диалогового окна TaskMenu, найти пункт меню &New/tF7 и убрать флажок из поля Disabled (рис. 8).

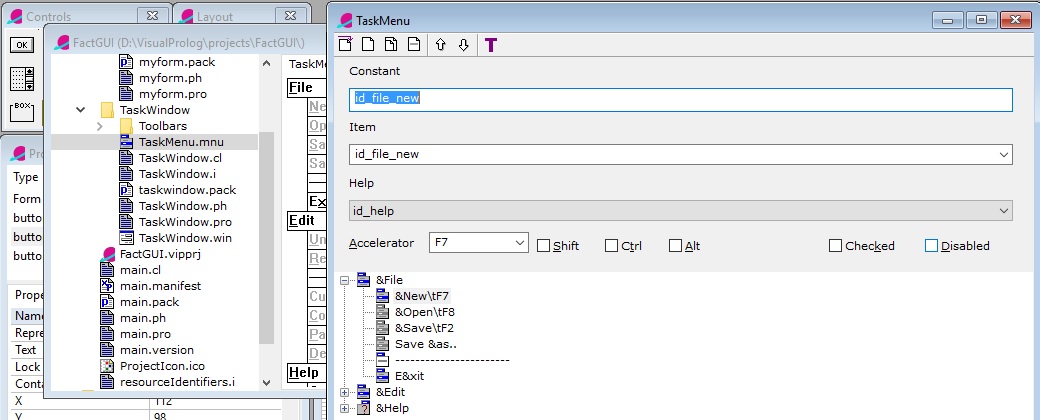


Рис. 8. Конструктор настройки меню.

Для привязки к событию выбора пункта меню нужного программного кода необходимо:

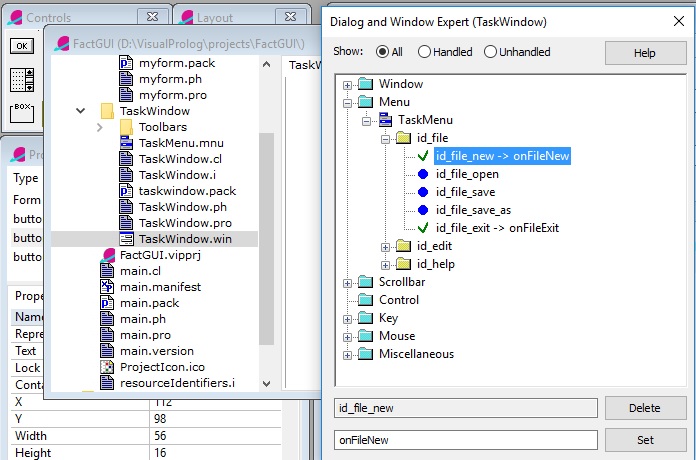
• Выделить элемент TaskWindow.win в дереве проекта, двойным кликом открыть диалоговое окно Dialog and Windows Expert.

• Раскрывая узлы дерева Menu, TaskMenu, …, дойти до узла id\_file\_new.

• Нажать кнопку Add. IDE создаст обработчик события onFileNew() и прототип программного кода для обработки этого события.

• Двойной клик в диалоговом окне на строке id\_file\_new –> onFileNew откроет файл TaskWindow.pro в области определения программного кода обработчика событий onFileNew.

• Заменить прототип кода реальным кодом, который создает экземпляр этой формы и отображает ее на экране (рис. 9).



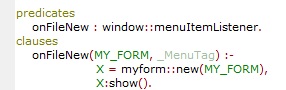


Рис. 9. Определение событий, реагирующих на выбор опций меню.

**Разработка экранной формы**

Для того чтобы разрабатываемый проект имел практическое значение необходимо наделить экранную форму возможностями по вводу нужных пользователю данных и запуску обработки этих данных. С этой целью надо разместить на форме два элемента управления: поле ввода и кнопку. Чтобы осуществить желаемое, следует:

• в дереве проекта выбрать FactGUI –> myform –> myform.frm;

• выполнить двойной клик на myform.frm для того, чтобы открыть конструктор форм (рис. 7);

• разместить на форме два элемента управления, изменяя их размеры, местоположение, обратив внимание на возможность использования инструментов выравнивания;

• используя окно Properties, установить желаемый набор свойств для каждого из элементов управления формы, установив: edit\_ctl – имя поля ввода, factorial\_ctl – имя кнопки и задать ее текст (рис. 10).

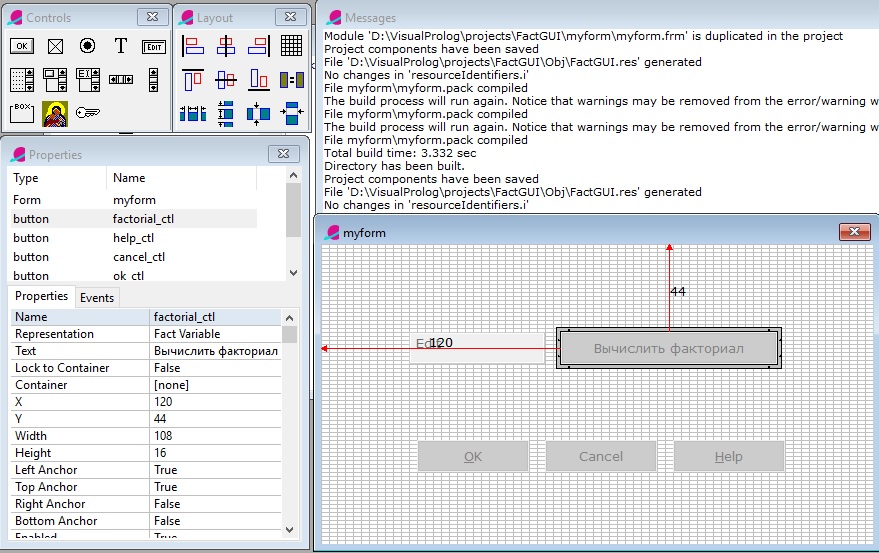


Рис. 10. Настройка свойств элементов управления формы.

Для того чтобы размещенная на форме кнопка могла реагировать на свое нажатие, требуется настроить ее обработчика событий. С этой целью следует:

• в окне Properties выбрать factorial\_ctl и перейти на вкладку Events;

• в списке событий выбрать ClickResponder, в поле, предназначенном для указания имени процедуры обработки события, щелкнуть мышкой и откроется окно с кодом файла myform.pro (рис. 11).

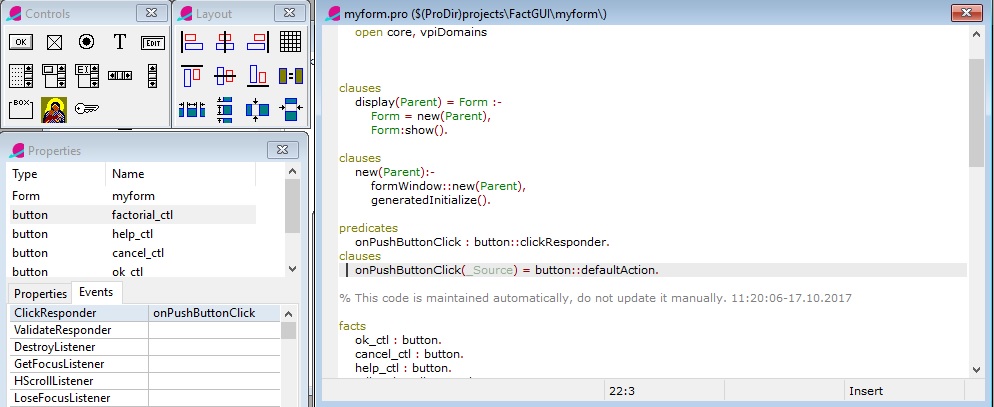


Рис. 11. Вкладка Events и прототип кода обработчика событий.

• фокус будет установлен на прототипе процедуры обработки событий onFactorialClick(), которую надо заменить приведенным ниже кодом

clauses

onPushButtonClick(\_Source) = button::defaultAction :-

Vvod = edit\_ctl:getText(),

stdio::write("Ввели N = ", Vvod, " Результат = ").

% myclass::calculate(Vvod).

Первая строка правой части правила вызывает метод getText() элемента управления edit\_ctl для получения содержимого поле ввода, которое унифицируется с переменной Vvod. Вторая строка, используя предикат write класса stdio, обеспечит вывод в стандартный поток. Для простоты мы предполагаем результат отображать не на форме, а выводить в консоль, роль которой будет выполнять стандартное окно системы Messages. Третья строка этого правила пока закомментирована и о ней разговор дальше.

**Организация вычислительных процедур проекта**

Все рассматриваемое ранее относится к визуальному проектированию в среде Visual Prolog, а возможности декларативных средств языка Prolog по настоящему используются только при разработке процедур обработки.

Вернемся к третьей строке кода процедуры обработки нажатия кнопки. Эта строка предполагает, что для вычисления результата вызывается метод calculate класса myclass, которому передается значение переменной Vvod. Однако, ни такого класса, ни такого метода у нас пока не существует. Для создания нового класса надо выполнить действия, которые во многом аналогичны тем, которые выполнялась при создании формы, а именно:

• установить в дереве проекта фокус на его вершину – узел FactGUI,

• выбрать опцию File –> New in Existing Package главного меню, откроется диалоговое окно Create Project Item.

• в левой панели выбрать объект Class и установить параметры, как указано на рис. 12;

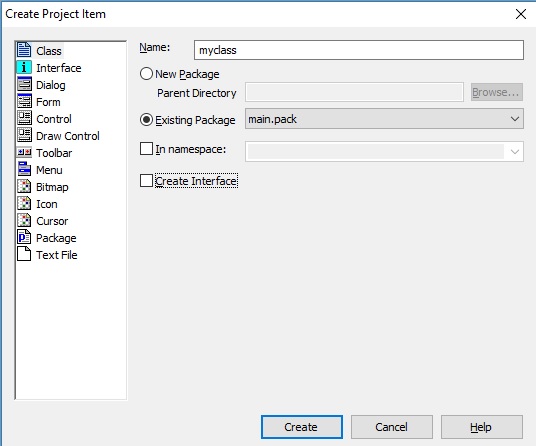


Рис. 12. Определение параметров создаваемого класса.

После того, как будет нажата кнопка Create, Visual Prolog создаст ряд прототипов файлов этого класса, которые добавятся в проект и отобразятся в окне дерева проектов. Одновременно с этим откроются два окна с файлами myclass.cl и myclass.pro, которые надо заполнить в соответствии с рис. 13.

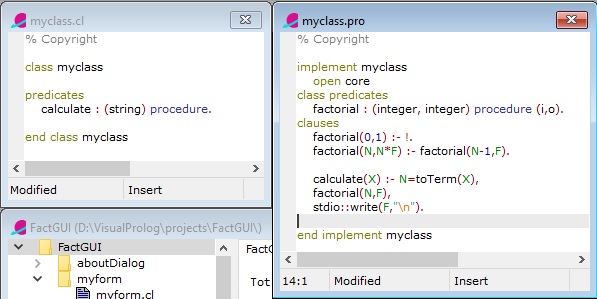


Рис. 13. Файлы декларации и описания класса myclass.

Если теперь снять комментарий в процедуре обработки нажатия кнопки и запустить проект на выполнение, то задача построения GUI приложения решена.

ЗАДАНИЕ

на Лабораторную работу №2

Реализовать с использованием IDE Visual Prolog и средств GUI базу знаний в предметной области согласно варианту задания на Лабораторную работу №1.

Реализовать возможность добавления новых информационных единиц в базу знаний пользователем.

Выбор используемых элементов GUI – свободный.