Функцион_Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Кафедра программного обеспечения информационных технологий

ФУНКЦИОНАЛЬНОЕ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Методическое пособие
для студентов специальности I-40 01 01
"Программное обеспечение информационных технологий"
В 4-х частях

Часть 2

Язык программирования Пролог

Минск 2006

УДК 681.3.06 (075.8)
ББК 32.973-018.1я73
Ф 94

Р е ц е н з е н т:
Зав. кафедрой программного обеспечения
сетей телекоммуникаций Высшего государственного колледжа связи,
д-р техн. наук, профессор А.А. Прихожий

Авторы:
С.В. Крицкий, И.М. Марина,
Е.В. Мельникова, Е.В. Шостак

Функциональное и логическое программирование: Метод. пособие
Ф 94 для студ. спец. I-40 01 01 «Программное обеспечение информационных технологий». В 4 ч. Ч. 2: Язык программирования Пролог / С.В. Крицкий, И.М. Марина, Е.В. Мельникова, Е.В. Шостак. – Мн.: БГУИР, 2006. – 28 с.
ISBN 985-444-967-х (ч.2)

Во второй части методического пособия изложены основы логического программирования на примере языка Пролог.

Пособие предназначено для студентов дневной и дистанционной форм обучения.

УДК 681.3.06(075.8)
ББК 32.973-018.1я73

Ч.1: Функциональное и логическое программирование: Метод. пособие для студ. спец. I-40 01 01 "Программное обеспечение информационных технологий" в 4-х ч. Ч. 1: Язык программирования Лисп/ И.А. Мурашко, И.М. Марина. – Мн.: БГУИР, 2002.

ISBN 985-444-967-х (ч.2)
ISBN 985-444-417-1

СОДЕРЖАНИЕ

Введение

Логическое программирование радикально отличается от других языков программирования.

Отличительной чертой фрагмента логики, используемого в логическом программировании, является то, что обычно целевые утверждения содержат кванторы существования – утверждения о том, что существуют некоторые объекты с заданными свойствами. В вычислениях используется конструктивный метод доказательства целевых утверждений: если доказательство прошло успешно, то в процессе доказательства находятся элементы, соответствующие неопределенным объектам, входящим в целевое утверждение. Такое соответствие и образует результат вычисления. Это можно сформулировать в виде двух метафорических равенств:

1) программа – множество аксиом;
2) вычисление – конструктивный вывод целевого утверждения из программы.

Название языка Пролог происходит от Programming in logic (программирование в логике). Теоретическое обоснование языку Prolog дали Ковальски и ван Эмден. Первый интерпретатор языка был разработан в 1971 году в Марсельском университете под руководством Кольмерора.

Программа на языке Пролог состоит из предложений, которые можно разделить на две группы: факты и правила вывода.

В виде фактов в программе записываются данные, которые принимаются за истину и не требуют доказательства. Данные в фактах могут быть использованы для логического вывода. Факт может описывать некоторые свойства объекта или отношения между объектами. Можно дать следующее определение для факта: факт – это свойство объекта или отношение между объектами, для которого известно, что они истинны.

Вторая группа предложений – правила вывода. Правило вывода состоит из двух частей, разделенных условным обозначением :- , которое читается как «если», или «при условии, что». Левая часть правила вывода называется заголовком или головной целью. Правая часть правила вывода называется хвостом или хвостовой частью. Хвостовая часть может состоять из нескольких условий (хвостовых целей), перечисленных через запятую или точку с запятой. Запятая означает операцию «логическое И», точка с запятой – операцию «логическое ИЛИ». Скобки в хвостовой части правила вывода не используются.

Головная цель правила вывода считается доказанной, если доказаны все хвостовые цели в правой части правила вывода.

Лабораторная работа № 1

1.1. Цель работы: изучение системы Турбо-Пролог фирмы Borland, приобретение практических навыков составления, отладки и выполнения простейшей программы в системе программирования Турбо-Пролог.

1.2. Краткие теоретические сведения

Объекты данных в Прологе называются **термами**. Терм может быть: константой, переменной, структурой (составной терм).

Числа в Турбо-Прологе записываются точно так же, как и в других языках программирования.

Константы относятся к одному из 6 стандартных типов данных (доменов), представленных в табл. 1.1

Таблица 1.1

Тип	Ключевое слово	Диапазон значений	Примеры
Символы	Char	все возможные одиночные символы	’a’,’B’,’?’
Целые числа	Integer	-32768 ... 32767	-15, 1235, 9
Действ. числа	Real	1E-307 ... 1E308	48, 2.45E-8
Строки	String	последовательность символов (до 250)	“Минск” “a_min”
Символические имена	Symbol	1. Последовательность букв, цифр, знака подчеркивания (первый символ – строчная буква) 2. Последовательность заключенных в кавычки символов	read_data1 “Delete f1” “Турбо-Си”
Файлы	File	допустимое в MS-DOS имя файла	a.txt progr.pro

Переменная – имя, начинающееся с большой (прописной) буквы или знака подчеркивания. Когда значение переменной несущественно, то в качестве имени переменной используется знак подчеркивания. Такая переменная называется **анонимной**.

Структуры (сложные термы) – это объекты, которые состоят из нескольких компонент. Структура записывается с помощью указания ее функтора и компонент. Компоненты заключаются в круглые скобки и разделяются запятыми. Число компонент в структуре называется **арностью** структуры.

Пример структуры: data_r(12,mart,1962). Здесь data_r – функтор; 12, mart, 1962 – компоненты. Арность приведенной структуры равна трем.

Структура программы

Программа на Турбо-Прологе состоит из нескольких разделов, каждому из которых предшествует ключевое слово. Типичная структура программы представлена ниже:

```
/*      комментарии      */
domains
<описание типов данных>
database
<описание предикатов динамической базы данных>
predicates
<описание предикатов>
clauses
    <утверждения>
 goal
    <целевое утверждение>
```

В программе наличие всех разделов не обязательно. Обычно в программе должны быть по крайней мере разделы predicates и clauses.

Раздел Domains

Существует 4 способа объявления типов данных (доменов):

1. name =d , где name – имена объектов стандартного типа, d – один из типов (char, symbol, integer, real, string)
2. list = element*, где list – список элементов element, element – элемент, описанный в разделе domains или один из стандартных типов, * – список.
3. num1=f1 (d11, …, d1M); f2 (d21, …, d2N). Тип num1 включает сложные объекты, которые объявляются путем установления функтора и описаний всех входящих в него компонент. collection = book (author, title); record (artist, album, type). Один оператор раздела domains описывает только один уровень дерева; books = book (title, author (name, surname)) – неверно.
4. file = name1; name2; … . Используется для обращения к файлам по символическим именам. В разделе domains может быть только один оператор этого типа. Символические имена файлов, если их несколько, задаются в качестве альтернативы.

Раздел Predicates

Предикат (отношение) в общем случае – это структура вида:

prednames(komp1,komp2,...),

где predname – имя предиката,

komp1,... – типы компонентов, описанных в разделе domains или стандартные типы.

Например:

```
domains
fio=string
den,god=integer
mes=symbol
predicates
```

anketa(fio,den,mes,god)

Если в предикатах используются только стандартные типы данных, то раздел domains может отсутствовать:

predicates

anketa(string,integer,symbol,integer)

Предикат может состоять только из одного имени, например:

predicates

result

Допускается многократное объявление предиката с одним и тем же именем. Одинаковое число компонентов в альтернативе необязательно

Раздел Clauses

В разделе clauses размещаются **предложения** (утверждения). Предложение представляет собой факт или правило, соответствующее одному из объявленных предикатов.

Факт – простейший вид утверждения, которое устанавливает отношение между объектами. Пример факта:

anketa(“Иванов”,5,august,1950).

Этот факт содержит атом anketa, который является именем предиката, и в скобках после него дается список термов, соответствующих компонентам этого предиката. **Факт всегда заканчивается точкой**. Факты содержат утверждения, которые всегда являются безусловно верными.

Правила отражают некую логическую зависимость некого предиката от других предикатов.

Правило состоит из заголовка и тела, соединенных символом :- (if). Заголовок правила – некий предикат, возможно, содержащий переменные. Тело правила (хвостовые цели) – список предикатов, разделённых запятыми. Заголовок if подцель1, подцель2, …, подцельN. Правило в общем случае гласит, что предикат, составляющий заголовок правила, будет истинным, если истинны все подцели, входящие в его тело, т.е. “,” имеет смысл конъюнкций. И заголовок, и подцель могут содержать переменные. Одноимённые переменные имеют смысл только в рамках одного правила, т.е. областью действия переменной в Пролог является **утверждение** (как факт правила или цель).

Правила заканчиваются точкой. Тело содержит список термов, разделенных запятыми или ; (:- if) (, and) (; or).

Все предложения раздела **clauses**, описывающие один и тот же предикат, должны записываться друг за другом.

Например:

любит (Саша, леденцы).

любит (Маша, X) if любит (Саша, X).

// Маша любит нечто, если это же самое любит Саша

// Выяснить: любит ли Маша леденцы

Goal: любит (Маша, леденцы)

Переменная X конкретизируется значением «леденцы» во всех частях правил. Порождается новая цель, выбирается первая подцель из тела правила: любит (Саша, леденцы). Эта цель новая и для неё поиск ведётся с начала

данных. И мы убеждаемся, что она согласуется. Переменная X получила новое значение.

Раздел Goal

В разделе **goal** записывается третий тип предложения – **вопрос (цель)**, состоящий из одного или нескольких целевых утверждений (подцелей), разделенных запятыми и оканчивающихся точкой. Пролог-система рассматривает вопросы как цели, к достижению которых нужно стремиться. Ответ на вопрос может оказаться или положительным, или отрицательным в зависимости от того, может ли быть цель достигнута или нет. Если на вопрос существует несколько ответов, то система может найти и выдать все из них.

В вопросах могут использоваться переменные.

Применение внешних целей бывает полезно при записи коротких вопросов, а также для получения всего набора допустимых решений. Другое преимущество внешних вопросов – возможность адресовать базе данных совершенно произвольные вопросы.

1.3. Содержание задания по лабораторной работе

Разработать программу, специфицирующую область семейных отношений. Базовые отношения, такие, как отец, мать, пол мужской или женский, описываются в разделе clauses фактами. Более сложные отношения, такие как сестра, брат, бабушка, дедушка, тетя, дядя, кузен, кузина, племянник, племянница, тесть, теща, свекровь, родитель, родственник и т.д. описываются правилами. Задачи формулируются в виде целей: найти всех родственников данного лица, найти всех бабушек и так далее.

Лабораторная работа № 2

2.1. Цель работы: изучение сложных доменов языка Пролог, определяемых посредством использования функторов; изучение методов простой и обобщенной рекурсии; приобретение практических навыков составления и отладки программ, работающих с бинарными деревьями.

2.2. Краткие теоретические данные

Правило может быть интерпретировано как некоторое определение предиката левой части через предикаты правой части. Правило, в котором правая часть включает предикат левой части, называется **рекурсивным.**

Рекурсия – это важный метод программирования (это способ мышления). В функциональном и логическом программировании – это основной прием.

Рассмотрим рекурсию на примере факториала:

0!=1

N!=N*(N-1)!, N>0

Шаги построения рекурсивных определений:

1. Выделить терминальные случаи.
2. Предположить, что уже есть предикат, который работает так, как необходимо для некоторого предыдущего (следующего) случая (в смысле движения по направлению к терминальному случаю).
3. Связать рекурсивными правилами предикат для некоторого общего случая с предикатом для предыдущего или следующего случая.
4. Проверить, протестировать и, возможно, исправить.

Рассмотрим по шагам нахождение факториала с помощью рекурсии. В Прологе это выглядит следующим образом:

f(N,F), где F – результат

Необходимо обязательно терминальное условие (терминальная ветвь). В нашем случае терминальное условие 0!=1 – это **факт.**

f(0,1).

f(N,F) if N>0, Nпред=N-1,f(Nпред,Fпред),F=N*Fпред.

Важен порядок подцели.

Goal: f(3, F)

Внутри Пролог-системы будет:

f(3,F). N=3

N1=2 (предыдущее)

Рекурсивный вызов f(2, F1) N=2

Рекурсивный вызов f(1, F2)

Рекурсивный вызов f(0, F3). Такая цель сопоставима с фактом, но не сопоставима с правилом.

F3=1

F2=1*N2=1

F1=F2*N1=2

F=F1*N=6

Рекурсия позволяет предположить, что требуемый предикат уже построен.

Пример: (сумма первых N квадратов):

сум(N,S) $\sum_{i=1}^{n} i^2$

сум(1,1) – граничное условие

сум(N,S) if N>1,N1=N-1,сум(N1,SS),

S=SS+N1*N1.

Нисходящая рекурсия – такая рекурсивная логическая программа, когда в теле правила, определяющего данный предикат, рекурсивное определение этому предикату происходит до полного завершения вычислений в данном правиле. Общий вид:

P if A`,A``,…,P`,P``,…,B`,B``… .

P – предикат

A’, A” – утверждения

P’, P” – обращение к предикату

B’, B” – остатки вычислений

Остатки вычислений являются цепочкой отложенных вычислений (основные вычисления). F_n if … f_{n-1}, x = … по аналогии с предыдущим x = и является остаточными вычислениями.

Восходящая рекурсия (частный случай – рекурсия с накапливающими параметрами) – это такая организация вычислительного процесса, когда часть вычислений, подлежащая повторению, завершается до рекурсивного вызова данного (определяемого) предиката.

P if A`,A``,…,P`.

Рекурсивный вызов стоит всегда на последнем месте, и он один единственный.

Примеры (факториал):

1. f(N, F) if f(0,1,N,F).

   Первый параметр f – счетчик, второй – накапливающий параметр.

   f(N,F,N,F). – терминальный случай.

   (N1,F1,N,F) if N>N1, N2=N1+1, F2=N2*F1, f(N2,F2,N,F).

2. f(N,F) if f(N,1,F).

   f(0,F,F).

   f(N,F1,F) if N>0, F2=F1*N, N1=N-1, f(N1,F2,F)

Рекурсия с недетерминированным выбором (с ветвлением) – определение данного предиката, когда имеется несколько правил, содержащих рекурсивное обращение. Может быть и нисходящим и восходящим.

Пример: Построить следующий ряд чисел: 2/1 2/3 4/3 4/5 6/5 6/7 … и найти их произведение.

Если n нечетное p(N) = (p(n -1) * (n +1)) / n

Если n четное p(N) = (p(n -1) * n) / n+1

P(1, 2).

P(I, X) if I > 1, I mod 2 = 0,

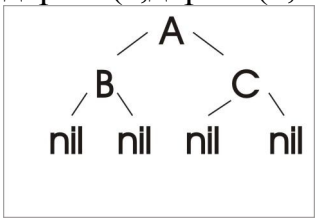
I1 = I –1,

P (I1, X1),
X = (X1 * I)/(I + 1).

P(I, X) if I > 1, I mod 2 = 1,
I1 = I –1,
P (I1, X1),
X = X1 * (I + 1) / I.

Бинарное дерево – структура данных, определяемая рекурсивно следующим образом: двоичное дерево состоит из корневой вершины, левого дерева и правого дерева или представляет собой пустое дерево. Пустое дерево – некоторое выделенное значение: пусто или nil.

дерево(a,дерево(b,nil,nil),дерево(c,nil,nil))



Деревья описываются в разделе Domains рекурсивными доменами, определяющими терминальные (листья) и нетерминальные вершины, которые могут именоваться символами или числами.

Определение бинарного дерева:

дв_дер=дерево(корень,л_дер,пр_дер);nil.

nil – т.е. пустое дерево.

л_дер, пр_дер = дв_дер.

корень = symbol.

Или можно проще:

дв_дер = дерево(корень,дв_дер,дв_дер);nil

Предикат, который проверяет, является ли элемент элементом двоичного дерева:

b_tree(nil).

b_tree(дерево(Вершина, ЛД, ПД)) if b_tree(ЛД), b_tree(ПД).

Принадлежность вершины дереву

Эл(X, дер(X, _, _)).

Эл(X, дер(Y, Л, П)):- Эл(X, Л), Эл(Y, П).

Обходы дерева

Во многих приложениях, использующих деревья, требуется доступ к элементам, указанным в вершинах. Основной является идея **обхода** дерева в предписанном порядке. Существуют три возможности линейного упорядочения при обходе: ***сверху–вниз***, когда сначала идет значение в вершине, далее вершины левого поддерева, затем вершины правого поддерева; ***слева–направо***, когда сначала приводятся вершины левого поддерева, затем вершина дерева и далее вершины правого поддерева; ***снизу–вверх***, когда значение в вершине приводится после вершин левого и правого поддерева.

Примеры:

Построить линейный одномерный список, содержащий вершины дерева.

*Обход дерева **сверху – вниз**:*
[a, b…(левое поддерево), c…(правое поддерево)]
обход1(nil, []).
обход1(дерево(X, Л, П), Y) if обход1(Л, $Л_s$),
обход1(П, $П_s$),
append([x| $Л_s$], $П_s$, Y).
*Обход дерева **слева – направо**:*
обход2(nil, []).
обход2(дерево(X, Л, П), Y) if обход2(Л, $Л_s$),
обход2(П, $П_s$),
обход2($Л_s$, [X| $П_s$], Y).
*Обход дерева **снизу – вверх**:*
обход3(nil, []).
обход3(дерево (X, Л, П), Y) if обход3(Л, $Л_s$),
обход3(П, $П_s$),
append($П_s$,[X], $П_s$1),
append($П_s$, $П_s$, Y).

Упорядоченное бинарное дерево – это дерево, у которого любая вершина левого поддерева меньше значения корня, а любая вершина правого – больше корня.

эл_т(X,дер(X,_,_)).
эл_т(X,дер(Y,Л,П)) if X>Y, эл_т(X,П).
эл_т(X,дер(Y,Л,П)) if X<Y, эл_т(X,Л).
Пример: добавить элемент к упорядоченному дереву.
вкл(Дс,X,Дт).
вкл(nil,X,дер(X,nil,nil)).
вкл(дер(Y,Л,П),X,дер(Y,Лх,П)) if X<Y, вкл(Л,X,Лх).
вкл(дер(Y,Л,П),X,дер(Y,Л,Пх)) if X>Y, вкл(П,X,Пх).

2.3. Содержание задания по лабораторной работе

Разрабатывается программа, реализующая различные операции над бинарными деревьями. Деревья описываются рекурсивными доменами, определяющими терминальные (листья) и нетерминальные вершины, которые могут именоваться символами или числами. Операции включают обход дерева сверху вниз, снизу вверх и слева направо, упорядочивание листьев дерева, вставку нового листа с сохранением отношения порядка, определение максимальной, минимальной и средней глубины дерева, нахождение суммы всех весов листьев. Программа тестируется на конкретных бинарных деревьях.

Лабораторная работа № 3

3.1. Цель работы: приобретение навыков работы со списками и множествами в программах на Турбо-Прологе.

3.2. Краткие справочные данные

Список – это специальный вид сложного терма, состоящий из последовательности термов, заключенных в квадратные скобки и разделенных запятой, например [1,2,-5,4,0].

Определение спискового домена:

domains

имя_списка = имя_базового_домена*

int_list = integr*

Список списков:

list_list = int_list*

Списки сопоставимы, если попарно сопоставимы их элементы.

Пример:

[8,2]=[8,2] – сопоставимы;

[8,3]=[8,2] – не сопоставимы;

[8,2,3]=[8,2] – не сопоставимы;

[8,x]=[8,2] – если x свободна, то сопоставление произойдет и x будет равен 2;

[8,x]=[8,y] – если конкретизированы, то произойдет сравнение, если свободны то x=y;

[[8,2],[4]]=[x,y] – если x & y были свободны, то x = [8,2], y = [4];

[x]=[] – не сопоставимы (слева список из одного элемента).

Списки являются основной структурой данных в программах на Прологе. Для удобства обработки списков введены два понятия: голова (head) и хвост (tail). Так, для списка [1,2,3] элемент 1 является головой списка, а список [2,3], включающий остальные элементы, является его хвостом.

Для отделения головы списка от хвоста используется символ |. Например, для списка [X|Y] X – голова списка, Y – хвост.

При работе со списками используются следующие основные операции.

Принадлежность элемента к списку:

member(X,L) – является ли X элементом L.

member(X,[X|_]).

member(X,[_|Y]) if member(X,Y).

Goal: member(A, [A,B]) – yes

Goal: member(X,[A,B,C]) В данном случае выдаст следующее:

x=a

x=b

x=c, т.е. перебор всех элементов списка.

Выделить последний элемент списка:

last(X,[X]).

last(X,[_|Y]) if last (X,Y).

Удаление из списка элементов < 5

del([],[]).

del([X|Y],Z) if X<5,del(Y,Z).

del([X|Y],[X|Z]) if X>=5,del(Y,Z).

Слить 2 упорядоченных по возрастанию списка чисел в один упорядоченный список

sl([X|XS],[Y|YS],[X,ZS]) if X<Y, sl(XS,[Y|YS],ZS).

sl([X|XS],[Y,YS],[X,Y|YS]) if X=Y, sl(XS,YS,ZS).

sl([X|XS],[Y|YS],[Y,ZS]) if X>Y, sl([X|XS],YS,ZS).

sl(X,[],X).

sl([],X,X).

Комментарий к последним двум строкам: термин ветви, порядок их (начало, конец) не влияет на результат.

Реверсирование простого списка

reverse([],[]).

reverse([X|XS],ZS) if reverse(XS,YS),append(YS,[X],ZS].

Тот же пример с помощью восходящей рекурсии:

reverse(X,Y) if reverse (X,[],Y).

reverse([],Y,Y).

reverse([X|T],A,Y) if reverse(T,[X|A],Y).

Работа с множествами.

Множества в Прологе строятся на основе списков.

Определение множества:

domains

set = integer*

Они отличаются от списков тем, что на множества накладываются ограничения(элементы не должны повторяться).

Над множествами определены следующие операции:

- генерация множества
- объединение множеств
- пересечение множеств
- вычитание множеств
- проверка на вхождение

Использование отсечения в Пролог-программах.

В процессе достижения цели Пролог-система осуществляет автоматический перебор вариантов, делая возврат при неуспехе какого-либо из них. Такой перебор – полезный программный механизм, поскольку он освобождает пользователя от необходимости программировать этот перебор самому. С другой стороны, ничем не ограниченный перебор может привести к неэффективности программы. Поэтому иногда требуется его ограничить или исключить вовсе. Для этого в Прологе предусмотрено специальное целевое утверждение !, называемое **отсечением** (cut).

Отсечение реализуется следующим образом: после согласования целевого утверждения, стоящего перед отсечением, все предложения с тем же предикатом, расположенные после отсечения, не рассматриваются.

Можно выделить три основных случая использования отсечения:

*1) **для устранения бесконечного цикла**;*

Пример: вычисление суммы 1 + 2 + … + N.

```
сумма( 1, 1 ) :- !.
сумма( N, R ) :- N1=N-1, сумма( N1, R1 ), R = R1 + N.
```

Если граничное условие будет записано в виде

```
сумма( 1, 1 ).
```

без отсечения, то сопоставление головы правила с запросом в разделе goal (или в окне Dialog) будет происходить успешно и при N <= 0, т.е. делается попытка доказать цель – сумма (0, R), что в свою очередь приводит к цели сумма(-1, R) и т.д., т.е. образуется бесконечный цикл.

*2) **при программировании взаимоисключающих утверждений**;*

Пример: нахождение большего числа среди двух чисел X и Y можно запрограммировать в виде отношения

max(X, Y, Max), где Max = X, если X >= Y и Max = Y, если Y > X.

Это соответствует двум предложениям программы:

```
max( X, Y, X ) :- X >= Y.
max( X, Y, Y ) :- X < Y.
```

Эти предложения являются взаимоисключающими, т.е. если выполняется первое, второе обязательно терпит неудачу, и наоборот. Поэтому возможна более экономная запись при использовании отсечения:

```
max( X, Y, X ) :- X >= Y, !.
max( _, Y, Y ).
```

*3) **при необходимости неудачного завершения доказательства цели**;*

Пример: присвоение какому-либо объекту категории в зависимости от величины заданного критерия в соответствии с табл. 3.1

Таблица 3.1.

Критерий	Категория
свыше 80 до 100	A
свыше 40 до 80	B
от 0 до 40	C

Возможный вариант программы:

```
категория( Кр, _ ) :- Кр > 100, !, fail; Кр < 0, !, fail.
категория( Кр, 'A') :- Кр > 80, !.
категория( Кр, 'B') :- Кр > 40, !.
категория( _, 'C').
```

При запросе категория (200, X) произойдет сопоставление запроса с головой первого утверждения. Цель Кр > 100 будет достигнута. Затем будет доказана цель-отсечение. Но когда встретится предикат fail, который всегда вызывает состояние неудачи, то стоящий перед ним предикат отсечения

остановит работу механизма возврата и в результате ответом на запрос будет No Solution (Нет решения).

Комбинация !, fail вводит возникновение неудачи.

3.3. Содержание задания по лабораторной работе

Разрабатывается программа, реализующая операции сортировки списка, построения обратного списка, конкатенации списков и другие. Разрабатывается программа, реализующая операции над множествами, представленными списками, включая операции объединения, пересечения, вычитания и генерации множества всех подмножеств данного множества.

Лабораторная работа № 4

4.1. Цель работы: научиться работать с динамическими базами данных

4.2. Краткие справочные сведения

Создание динамических баз данных (БД)

БД – это упорядоченная совокупность данных, которая является составной частью системы управления базами данных (СУБД). Существует три модели БД:

- сетевая;
- иерархическая;
- реляционная.

Сетевые СУБД используют модель представления данных в виде произвольного графа. В иерархической СУБД данные представлены в виде древовидной (иерархической) структуры. В реляционной модели данные хранятся в виде таблицы.

Практически все БД ПК поддерживают реляционную модель данных. В Турбо Паскале имеются специальные средства для организации БД. Эти средства рассчитаны на работу с реляционными БД. Внутренние унификационные процедуры языка осуществляют автоматическую выборку фактов с нужными значениями известных параметров. Механизм возврата позволяет находить все имеющиеся ответы на сделанный запрос.

Предикаты БД в ТП описываются в разделе databas, который должен располагаться перед разделом predicates. Все утверждения с предикатами, описанными в database, составляют динамическую БД. БД называется динамической, так как во время работы программы у нее (БД) можно удалять любые содержащиеся в ней утверждения, а также добавлять новые. В этом заключается отличие от статической, где утверждения являются частью кода программы и не могут быть изменены во время программы.

Другая особенность динамической БД (ДБД) состоит в том, что БД может быть записана на диск и считана с него в ОП. Например, в ДБД

содержатся сведения об игроках футболистах dplayer. Тогда требуется следующее описание в программе:

```
domains
    name, team=symbol
    number = integer
database
    dplayer (name, team, number)
```

Иногда выгоднее иметь часть информации БД в виде утверждений статической БД. Эти данные заносятся в ДБД сразу после активизации программы. Для этого используются предикаты asserta, assetz. Предикаты статической БД имеют другое имя, но ту же форму представления данных, что и предикаты ДБД. Предикат СБД, соответствующий предикату dplayer, может быть представлен в программе следующим образом:

```
predicates
    player (name, team, number)
clauses
    player (“Adamov”, “Dinamo”, 5).
```

В ДБД содержатся только факты, но не правила.

Правило для занесения в ДБД информации из утверждений предиката player служит:

```
assert_database:-
    player (Name, Team, Number),
    assertz (dplayer (Name, Team, Number)),
    % Добавление указанного факта в ДБД
    fail.
    assert_database:-!.
```

Предикаты для работы с утверждениями ДБД

Турбо Паскаль имеет большой набор встроенных предикатов. Большинство стандартных предикатов выполняют несколько функций в зависимости от состояния параметров, входящих в предикат. К моменту обращения к предикату каждый отдельный его параметр может быть определен или не определен. Известные параметры предиката – входные (i), неизвестные – выходные (o). Совокупность входных и выходных параметров определяет работу предиката и называется *поточным шаблоном*. Не для каждого варианта все возможные варианты поточного шаблона имеют смысл.

Занесение нового факта в начало БД, располагающейся в ОП (резидентная БД).

asserta (<fact>) (dbasedom) : (i)

домен, обозначенный как dbasedom, автоматически объявляется для каждого предиката из раздела database.

Пример: asserta (dplayer (“Ivanov”, “Torpedo”, 1)).

В разделе database должен быть определен предикат dplayer.

Занесение нового факта в конец БД

assertz (<fact>) (dbasedom) : (i)

Удаление утверждения из БД

retract (<fact>) (dbasedom) : (i)

Предикаты для работы с БД в целом

Запись на внешний накопитель ДБД:

save (ИмяФайлаДОС) (string) : (i)

После записи файл можно снова загрузить в ОП с помощью предиката consult.

Добавление текстового файла в ДБД

consult (ИмяФайлаДОС) (dbasedom) (string) : (i)

Текстовый файл может быть создан, помимо обычного редактора, с помощью предиката save. Этот файл содержит файлы, которые должны быть описаны в разделе database. Факты из текстового файла дополняют набор уже существующих фактов в ДБД, но не заменяют их.

Чтение из файла объектов, относящихся к определенному в программе домену.

readterm (Домен, Терм) (<ИмяФайлаДОС>,< терм>) : (i, o)

С помощью его осуществляется доступ к фактам в файле. Для получения доступа к файлу сначала необходимо воспользоваться предикатом **openread** и **readdevice**.

readterm **(**autorecord, auto (Name, Year, Price))

Здесь терм auto (…) определяет все наборы значений домена autorecord. Необходимое описание должно выглядеть так:

```
domains
    name = string
    year = integer
    price = real
    auto_record = auto (name, year, price)
    file = auto file
```

Например, в файле, открытом при помощи предиката openread и readdevice, содержится факт auto (“Toyota”, 1986, 5000).

Name = “Toyota”

Year = 1986

Price = 5000

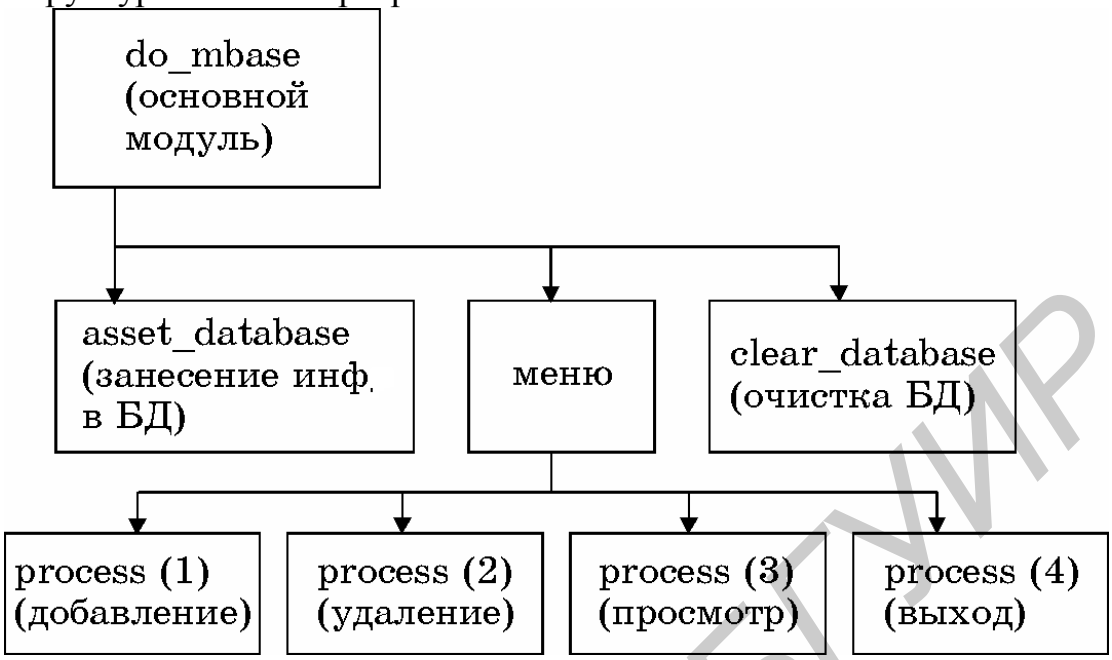
Сбор данных БД в список

findall (Переменная, Терм, Список) : (i, o, o)

Findall записывает значения объекта «Переменная» в список «Список». Переменная должна являться одним из аргументов предиката «Терм». «Список» должен быть описан в domains.

Пример программы, создающей ДБД

Структурная схема программы:



Текст программы

```
/* программа для работы с ДБД */H
domains
        name, command = string
        number = integer
database
        dplayer (name, command, number)
predicates
        repeat          % повтор
        do_mbase        % цель
        assert_database % создание БД
        menu            % интерфейс
        process (integer)   % операции из меню
        clear_database      % очистка БД
        error               % выдача сообщений об ошибках
        player (name,command,number)        % предикат статической БД
goal
        do_mbase
clauses
        repeat.
        repeat:-repeat.
/* Исходная СБД */
        player (“Иванов”, “Динамо”, 5).
        player (“Сидоров”, “Торпедо”, 10).
-*-*-*-*-*-
assert_database:-
        player (Name, Command, Number),
        assertz (dplayer (Name, Command, Number)),
```

```
        fail.
assert_database:-!.
/* Очистка БД */
clear_database:-
        retract (dplayer (_,_,_)), fail.
clear_database:-!.
/* Задание цели в виде правила */
do_mbase:-
        assert_database,
        makewindow (1, 7, 7, “БД”, 0, 0, 25, 80),
        menu, clear_database.
/* Правило для создания меню */
menu:-
        repeat, clearwindow,
        write (“******..**”), nl,
        write (“1.Добавить”), nl,
        write (“2.Удалить”), nl,
        write (“3.Просмотр”), nl,
        write (“4.Выход”), nl,
        write (“******..**”), nl, nl,
        write (“Введите пункт меню”), nl,
        readint (X), nl,
        process (X), X=4, !.
/* Добавление информации в ДБД */
process (1):-
        makewindow (2, 7, 7, “Добавление”, 2, 20, 18, 58),
shiftwindow (2),
write (“Введите имя игрока:”),
readln (Name),
write (“Введите название команды”),
readln (Command),
write (“Введите номер игрока”),
readint (Number),
assertz (dplayer (Name, Command, Number)),
write (Name, “ записан в ДБД”),
write (“Нажмите клавишу anykey”), readchar(_),
removewindow.
        /* Удаление информации из БД */
        process (2):-
        makewindow (3, 7, 7, “Удаление”, 10, 30, 7, 40),
        shiftwindow (3),
        write (“Введите имя для удаления:”),
        readln (Name),
        retract (dplayer (Name, _, _)), write (Name, “Удален из БД”), nl,
        write (“Нажмите клавишу anykey”), readchar(_),
```

```
        removewindow.
/* Просмотр информации из БД */
process (3):-
        makewindow (4, 7, 7, “Просмотр”, 7, 30, 11, 45),
        shiftwindow (4),
        write (“Введите имя игрока:”),
        readln (Name),
        dplayer (Name, Command, Number),
        write (“Данные об игроке”), nl, nl,
        write (“Имя игрока       : ”, Name), nl
        write (“Название команды : ”, Command), nl
        write (“Номер игрока     : ”, Number), nl
        write (“Нажмите клавишу anykey”), readchar(_),
        removewindow.
/* Введенного игрока нет в БД */
process (3):-
        makewindow (5, 7, 7, “Нет данных”, 4, 5, 6, 50),
        shiftwindow (5),
        write (“Введенного имени нет в БД”), nl, nl,
        write (“Нажмите клавишу anykey”), readchar(_),
        removewindow, shiftwindow (1).
/* Выход из диалога */
process (4):-
                write (“Работа завершена ? (Y/N)”),
        readln (Answer),
        upper_lower (Answer, Answ1) %   независимо от регистра
  frontchar (Answ1, ‘y’, _), !.
        process (X):-
                X<1, error,
                X>4, error.
        error:-
                write (“Введенное число должно быть в диапазоне 1 -4”), nl
                wriet (“Нажмите клавишу anykey”),
  readchar(_).
```

Возможные улучшения СУБД

1. Проверка введенных данных. Иногда полезно проверить введенные с клавиатуры данные перед тем, как засылать их в БД. В этом случае программа должна вывести данные на экран и дать возможность пользователю убедиться в их правильности. Если обнаружены ошибки, то должна быть возможность отменить этот ввод и набрать данные заново.

2. Редактирование данных. Введение операции редактирования позволяет добавлять в программу модуль, который выводит данные на экран, воспринимает сделанные пользователем изменения и делает перезапись данных в файл.

3. Возможность численных преобразований, то есть возможность обновления числовых данных путем выполнения над ними нужных арифметических операций.

4. Введение дополнительных предикатов в БД. Программа может содержать несколько предикатов ДБД.

5. Вывод на принтер ответа системы на запрос. Добавляется опция, позволяющая переадресовывать выдачу данных с экрана на принтер.

6. Возможность диалога с программой на естественном языке. Использовать в диалоге не меню, а естественный языковой интерфейс, понимающий команды, в рамках определенного словаря терминов.

4.3. Содержание по лабораторной работе:

Разрабатывается программа либо для игровой задачи, либо для части продукционной экспертной системы. Используются средства накопления, модификации и удаления фактов из базы данных, сохранения и восстановления базы.

Лабораторная работа № 5

5.1. Цель работы: изучение предикатов оконной системы

5.2. Краткие справочные данные

Определение окна

makewindow (НомерОкна, АтрОкна, АтрРамки, Заголовок, Строка, Столбец, Высота, Ширина) (integer, integer, integer, string, integer, integer, integer, integer): (все i) (все o)

Строка, столбец – координаты верхнего левого угла.

АтрЭкрана: =7 – позитивное изображение. 112 – негативное изображение. +1 – подчеркивание, +8 – высокое разрешение, +128 – мерцание символа.

Атрибуты рамки: 0 – нет рамки.

Смена текущего окна (считывание номера)

shiftwindow (НомерОкна) : (integer) (i) (o)

Окна перекрываются.

Быстрое переключение между двумя окнами

gotowindow (НомерОкна) : (integer) (i)

Окна не перекрываются и работают быстрее, чем shiftwindow.

Очистка текущего окна

clearwindow

Удаление текущего окна

removewindow

Графика Turbo Prolog v1.0

Управление графическими режимами и цветами в ТП осуществляется с помощью параметров, задаваемых предикатами graphics и maxiwinodws:

graphics (Режим, Палитра, Фон) : (integer, integer, integer) (i, i, i)

При переходе в графический режим очищается экран. Переход обратно в текстовый – предикатом text.

Для создания изображения используются предикаты line и dot.

line (X1,Y1, X2, Y2, Цвет) (integer, integer, integer, integer) : (i, i, i, i)

Координаты в интервале (0…31999). Цвет (0…15)

dot (Строка, Столбец, Цвет) (integer, integer, integer) : (i, i, i) (i, i, o)

Устанавливает или читает цвет в заданной точке.

Пример изображения эллипса:

```
predicates
        ellips (real)
        real_int (real, integer)
goal
        graphics (5, 0, 15), ellipse (0).
clauses
        ellips (A):- A>=6.28, ! %  Если полный круг , то A – угол
        ellips (A):- Xc=5000, Yc=18000, R=3600,
X=Xc+R*sin(A), Y=Yc-R*cos(A),
real_int (X, Xi), real_int (Y, Yi),
dot (Xi, Yi, 1), A1=A+0.02, ellips (A1).
real_int (R, I):- R=I;
```

Использование звука и музыки

beep – генерирует звук высокой частоты.

sound (T, F) (integer, integer) : (i, i)

Частоты для первой октавы представлены в табл. 5.1.

Таблица 5.1

Нота	Частота	Нота	Частота
До	262	фа#	370
до#	278	Соль	392
Ре	294	соль#	416
ре#	302	Ля	440
Ми	330	ля#	460
Фа	350	Си	494
		До	524

Графика Turbo Prolog v2.0

Существует более семидесяти предикатов для работы с графикой. При работе с графикой в программе с помощью директивы компилятора include подсоединяется текстовый файл GrapDecl.pro, в котором содержатся константы, используемые в графических предикатах (названия графических драйверов, режимы их работы, наименования цветов, стиль линий, коды возвращаемых ошибок, имена побитовых операций и другие).

Инициализация графического режима

initgraph (ГрДрайвер, ГрРежим, НовДрайвер, НовРежим, ПутьКДрайверу) (integer, integer, integer, integer, string) : (i, i, o, o, i)

Загружает драйвер списка или определяет уже загруженный драйвер и переводит систему в графический режим. Связывает переменные Нов* с фактически загруженными драйверами графического режима. Путь к драйверу описывает каталог, в котором ищется графический драйвер (*.bgi). Если необходимый драйвер не найден, то поиск повторяется в текущем каталоге; если путь пустой, то драйверы должны быть в текущем каталоге. При выполнении предиката возможны ошибки со следующими кодами:

6001 – невозможно определить графическую карту
6002 – невозможно найти файлы драйвера
6003 – неправильный драйвер
6004 – не хватает памяти для загрузки драйвера.

Чаще всего initgraph применяется в паре с предикатом

detectgraph (ГрДрайвер, ГрРежим) (integer, integer) : (o, o)

```
include “grapdecl.pro”
predicates
    demo
clauses
    demo:- detectgraph (GraphDriver, GraphMode),
writef (“Определен гр. драйвер %   и режим %  \n”,
GraphDriver, GraphMode),
readchar (_), getmaxx(X), X2=X div 2,
getmaxy (Y), bar (0, 0, X2, Y), readchar (_), closegraph.
goal
    demo
/* Автоматическая инициализация
goal
    initgraph (detect, 0, _, _, “ “), … , closegraph.
```

Графические предикаты TP v2.0

arc (X, Y, НачУгол, КонУгол, Радиус) (integer, integer, integer, integer, integer) (i, i, i, i, i) – рисует сектор круга;

bar (X1, Y1, X2, Y2) (integer, integer, integer, integer) : (i, i, i, i) – рисует заполненную полосу с координатами;

circle (X, Y, Радиус) (integer, integer, integer) (i, i, i) – окружность;

cleardevice – очистка экрана и установление пера в (0, 0) в текущем окне;

clearviewport – чистит текущее окно;

closegraph – отключает графическую систему, при этом освобождается память, выделенная для графической системы, и восстанавливается предыдущий режим экрана;

drawpoly (СписокВершин) : (integer*) (i) – рисует контур многоугольгика. Координаты первой и последней точки должны совпадать;

ellipse (X, Y, НачУгол, КонУгол, Храдиус, Урадиус) (integer, integer, integer, integer, integer, integer) (i, i, i, i, i, i) – рисует эллипс;

fillellipse (X, Y, Храдиус, Урадиус) – рисует и заполняет эллипс;

fillpoly (Список вершин) (integer *) – рисует и заполяет многоугольник;

getarccoords (X, Y, Хнач, Унач, Хкон, Укон) (o, o, o, o, o, o) – получает координаты из последнего вызова arg5;

getaspectratio (Хотн, Уотн) (o, o) – определяет отношение горизонтального и вертикального масштаба графического режима;

getbkcolor (ЦветФона) (integer) (o) – возвращает текущий цвет фона;

getcolor (Цвет) (integer) : (o) – возвращает текущий основной цвет;

getdefaultpalette (ПалитраУмолч) (integer) (o) – палитра, заданная по умолчанию;

getdrivername (ИмяДрайвера) (string) : (o) – возвращает имя загруженного драйвера;

getfillsettings (Шаблон, Цвет) (integer, integer) : (o, o) – информация о текущем шаблоне заполнения и цвете. Существует 11 заранее определенных шаблонов, описанных в модуле. Если шаблон = 12, то используется шаблон, определенный пользователем;

getfillpattern (Шаблон) (integer *) : (o) – возвращает шаблон заполнения, определенный пользователем с помощью setfillpattern. Шаблон имеет 8 сегментов, каждый 1 байт, отражающий на экране 8 пикселов, если бит равен 1, то он отображается на экране;

getgraphmode (ГрРежим) (integer) : (o) – возвращает значение текущего графического режима. Когда вызывается предикат, режим уже должен быть установлен;

getimage (Лев, Верх, Прав, Низ, ДвоичнМассив) (integer, integer, integer, integer, string) (i, i, i, i, o) – сохраняет битовое изображение, описанное прямоугольной областью экрана в памяти;

getmaxcolor (МаксЦвет) (integer) : (o) – максимальное значение цвета пиксела для текущего графического драйвера и режима;

getmaxx (X) (integer) : (o) – возвращает максимальную координату экрана по OX;

getmaxy (Y) (integer) : (o)-по OY;

getmoderange (ГрДрайвер, НизРежим, ВерхРежим) (integer, integer, integer) (i, o, o) – определяет диапазон режима для указанного графического драйвера. Если в ГрДрайвер указан неправильно, то НизРежим = ВерхРежим = -1);

getpalettesize (РазмерПалитры) (integer) (integer *) : (o);

getpixel (X, Y, Цвет) (integer) (i, i, o) – возвращает цвет в (X, У);

getX (X) (integer) : (o) – возвращает текущую позицию координаты X относительно текущего окна;

getY (Y) (integer) : (o) -- относительно Y;

graphdefaults – переключает все графические значения в значения по умолчанию;

imagesize (Лев, Верх, Прав, Низ, Размер) (integer, integer, integer, integer, integer) : (i, i, o, o, o) – определяет количество байтов для сохранения двоичного изображения прямоугольной области;

line (X1, Y1, X2, Y2) (integer, integer, integer, integer) : (i, i, i, i) – рисует линию;

linerel (Dx, Dy) (integer, integer) : (i, i) – рисует линию на относительное расстояние из текущей позиции;

lineto (X, Y) (integer, integer) : (i, i) – рисует линию из текущей позиции в точку;

moverel (Dx, Dy) (integer, integer) : (i, i) – передвигает позицию на относительное расстояние;

moveto (X, Y) (integer, integer) : (i, i) – передвигает позицию в точку;

outtext (Строка) (string) : (i, i) – выводит на экран строку с текущей позиции;

outtextXY (X, Y, ТекстоваяСтрока) (integer, integer, string) (i, i, i) – отображает строку с заданной позиции;

putimage (X, Y, ДвоичнМассив, On) (integer, integer, string, integer) (i, i, i, i) – выводит на экран изображение, сохраненное с помощью getimage. On указывает, как вычислить цвет каждого выводимого символа. Результат этой операции зависит от пиксела, имеющегося на экране, и пиксела, который выводится. Данные представлены в табл. 5.2

Таблица 5.2

Имя	Значение	Описание
Copy_put	0	копирование
XOR_put	1	исключающее ИЛИ
OR_put	2	ИЛИ
AND_put	3	И
NOT_put	4	отрицание

putpixel (X, Y, Цвет) (integer, integer, integer) (i, i, i);

rectangle (Лев, Верх, Прав, Низ) (integer, integer, integer, integer) (i, i, i, i);

restorecrtmode – переход на другой графический режим;

setpalette (СписокПалитры) (integer *) : (i) – изменяет всю палитру цветов. Размер зависит от графического драйвера.

Наиболее употребительные цвета:

0 – черный, 1 – синий, 2 – зеленый, 63 – белый, 3 – голубой, 62 – желтый, 4 – красный, 5 – фиолетовый, 61 – малиновый, 20 – коричневый, 60 – розовый, 7 – светло-серый, 59 – ярко-голубой, 56 – темно-серый, 58 – ярко-зеленый, 57 – ярко-синий;

setaspectratio (Хотн, Уотн) (o, o) – устанавливает отношение горизонтального и вертикального масштаба графического режима;

setbkcolor (ЦветФона) (integer) (o) – устанавливает текущий цвет фона;

setcolor (Цвет) (integer) : (o) – устанавливает текущий основной цвет;

setfillpattern (Шаблон) (integer *) : (o) – устанавливает шаблон заполнения, определенный пользователем с помощью setfillpattern. Шаблон имеет 8 сегментов, каждый объемом 1 байт, отражающий на экране 8 пикселов, если бит равен 1, то он отображается на экране;

setgraphmode (ГрРежим) (integer) : (o) – устанавливает значение текущего графического режима. Когда вызывается предикат, режим уже должен быть установлен;

getmaxcolor (МаксЦвет) (integer) : (o) – максимальное значение цвета пиксела для текущего графического драйвера и режима;

setviewport (Лев, Верх, Прав, Низ, Флаг) (integer, integer, integer, integer, integer) – (i, i, i, i, i) – устанавливает текущее графическое окно для вывода. Переменной флаг определяется возможность рисовать за границей окна (0 – нельзя);

setwritemode (РежимРис) (integer) : (i) – устанавливает режим рисования линии. Существует 2 константы:

Copy_put = 0 – цвет рисуемых линий будет накладываться на изображение на экране;

XOR_put = 1 – цвет будет смешиваться с изображением на экране.

5.3. Содержание задания по лабораторной работе: разрабатывается программа, реализующая интерфейс с пользователем либо в процессе игры, либо в процессе функционирования продукционной экспертной системы.

ЛИТЕРАТУРА

1. Братко И. Программирование на языке Пролог для искусственного интеллекта. –М.: Мир, 1990.
2. Ин П., Соломон Д. Использование Турбо-Пролога. – М.: Мир, 1993.
3. Стерлинг Л., Шапиро Э. Искусство программирования на языке Пролог. – М.: Мир, 1990.
4. Прихожий А.А. Функциональное и логическое программирование. – Мн.: БГУИР, 1998.

Св. план 2006, поз. 56

Учебное издание

Крицкий Сергей Вадимович
Марина Ирина Михайловна
Шостак Елена Викторовна
Мельникова Елена Владимировна

ФУНКЦИОНАЛЬНОЕ И ЛОГИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Методическое пособие для студентов специальности
I-40 01 01 «Программное обеспечение информационных технологий»
в 4-х частях

Часть 2

Язык программирования Пролог

Редактор Т.П. Андрейченко

Подписано в печать 22.06.2006. Формат 60x84 1/16. Бумага офсетная.
Гарнитура «Таймс». Печать ризографическая. Усл. печ. л 1,74.
Уч.-изд. л. 1,4. Тираж 100 экз. Заказ 6.

Издатель и полиграфическое исполнение: Учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники»
ЛИ №02330/0056964 от 01.04.2004. ЛИ №02330/0131666 от 30.04.2004.
220013, Минск, П.Бровки, 6.