Содержание

Предисловие 4

Глава 1. Неформальное введение 5
Кратко о главном 6
Условные циклы 12
Общая структура программы на КП 16
Условный оператор 21
Какие еще есть типы данных в КП 29
Массивы 30
Вложенные циклы 36
Многомерные массивы 49
Процедуры 57
Рекурсия 75
Записи 86
Указательные типы 94
Связные списки 96
Деревья 106
Файлы 114

Глава 2. Систематическое введение в КП 117
Введение 118
Понятие числа 127
Понятие идентификатора 127
Величины. Типы данных. Объявление и виды типов 130
Операции 142
Операторы 145
Модули 160
Полный список предопределенных процедур 161

Глава 3. Практикум 165
Раздел A. Разные задачи 181
Раздел B. Сортировки 198
Раздел C. Задачи перебора 201
Раздел Д. Графы 209

Приложение. Кратко о теории графов 226

Заключение 240

Предисловие

Книга, которую вы начинаете читать предназначена для начинающих изучать программирование, или имеющих небольшой программисткий опыт. В общем это книга для неискушенных, но желающих научится многому. Конечно, для профессионального познания любой области, одной книги всегда мало, но если у вас хватит терпения и упорства в проработке практического материала, а наша книга почти целиком посвящена практике, то можно быть уверенным, что ваш программисткий уровень станет достаточным для дальнейшего уже профессионального роста.

Книга состоит из трех глав и одного приложения. Первая глава «Неформальное введение», фактически самодостаточный логически завершенный самоучитель. Тщательное изучение неформального введения даст небольшой, но уверенный навык решения прикладных задач, и знание языка Компонентный Паскаль. Все неформальное введение от первой до последней страницы это решение задач. Каждая задача разбирается достаточно подробно, задачи используются и для рассказа о языке. Языковые конструкции вводятся по мере необходимости, тогда когда это нужно для решения очередной задачи. Это делает изучение языка хотя и несколько бессистемным (но в книге есть еще и систематическое введение), но прозрачным и понятным. Уровень сложности решаемых задач постепенное растет, но по настоящему сложных задач в неформальном введении все же нет, поэтому для его усвоения достаточно упорства и желания.

Вторая глава – систематическое введение в язык КП (Компонентный Паскаль). Здесь уже нет практики. Вся глава посвящена теории, а точнее изложению языка. Если в неформальном введении язык излагался «между делом», а главным было решение задач, то здесь главным становится язык, поэтому вторая глава отличается более строгой логикой и более трудна для понимания, но если неформальное введение вами пройдено успешно, то и вторая глава не должна создать серьезных затруднений. Еще одно важное отличие неформального введения от систематического. В первой главе используется не весь язык КП. Вторая глава излагает язык в полном объеме.

Третья глава – практикум, целиком посвящена задачам, но если в первой главе задачи использовались для объяснения, то в третьей главе задачи предлагаются для самостоятельного решения. Уровень сложности уже существенно выше. Но от вас не требуется решения с пустого места. Каждая предлагаемая в практикуме проблема снабжена пояснениями разного уровня. Где-то это описание алгоритма, где-то изложение идеи, иногда просто подсказка, иногда довольно детальная помощь. Конечно, несмотря на помощь, над каждой задачей придется основательно потрудится, но если первые две главы усвоены успешно, то и третья вполне преодолима.

Кроме трех основных глав есть еще небольшое приложение, посвященное теории графов. Оно невелико по размеру и не предназначено для хорошего изучения теории. Это приложение появилось лишь в силу того, что такая математическая конструкция как графы довольно часто используется в задачах практикума. Поэтому было сочтено полезным дать хотя бы небольшой справочный материал.

Глава 1

Неформальное введение

Кратко о главном 6
Условные циклы 12
Общая структура
программы на КП 16
Условный оператор 21
Какие еще
есть типы данных в КП 29
Массивы 30
Вложенные циклы 36
Многомерные массивы 49
Процедуры 57
Рекурсия 75
Записи 86
Указательные типы 94
Связные списки 96
Деревья 106
Файлы 114

Кратко о главном

Программирование – наука являющаяся предметом данной книги стоит на трех слонах и одной черепахе. Слоны это: постановка задачи, алгоритм, программа. Каждое из перечисленных понятий имеет сложную историю образования, различные понимания, но для начала можно отвлечься от строгой науки и запомнить что:

- постановка задачи – это описание задачи в строгой математической терминологии;
- алгоритм – это описание действий, выполняя которые некий ИСПОЛНИТЕЛЬ ***обязательно*** получит требуемый результат. Обратите внимание на выделенное слово, именно так, не может получить, а обязательно получит;
- программа – это запись алгоритма на строгом, однозначно понимаемом языке.

А черепаха, на которой стоят наши слоны – это исполнитель, способный выполнить алгоритм. Кстати совершенно не обязательно компьютер. В самом общем случае необходимо говорить о устройстве способном выполнять определенный, жестко заданный набор команд, но наша цель – программирование компьютера, поэтому далее исполнитель это всегда компьютер.

Следовательно, научится программировать, это значит научится:

- формулировать задачу в строгих математических терминах;
- находить решение в виде последовательности действий понятных компьютеру;
- записывать эту последовательность на языке программирования.

Зачем нужен язык программирования

Язык, есть способ записи мысли. Это утверждение верно, как для естественного языка, так и для любого другого, в том числе и для языка программирования. Поэтому, главная проблема любого начинающего изучать науку программирования, это алгоритмический способ мышления, некоторые специальные методы и приемы рассуждений. Мыслить алгоритмически мы все более или менее умеем, практически любой человек в состоянии понять запись алгоритма на естественном языке, если предмет алгоритма ему известен, иначе говоря, если человек является исполнителем с достаточным для данного алгоритма набором команд.

Поэтому, по крайней мере на первых порах, проблема кажущаяся второстепенной выходит на первый план. Эта проблема названа в заголовке выше. Зачем нужен специальный язык и как им пользоваться? Попробуем сейчас решить небольшую конкретную задачу и убьем двух зайцев: ответим на поставленный вопрос и получим первую информацию о языке Компонентный Паскаль, который в дальнейшем станет основой нашего движения вглубь науки о программировании (или искусства).

Задача 1. Дано множество чисел. Найти сумму положительных.

Пока мы не знаем никакого языка кроме естественного, поэтому попробуем записать необходимую последовательность действий (алгоритм) на русском языке и посмотрим чем это будет хорошо или наоборот плохо.

Вариант 1:

Для всех чисел из данного множества:
если число положительное, то прибавляем его к сумме

Наверное, это описание будет понятно каждому кто немного знаком с математикой, хотя бы в пределах арифметики. Но к сожалению, все не так просто. В «алгоритме» (пока в кавычках, так как это описание еще очень далеко от алгоритма) сказано, что некоторую операцию необходимо выполнить для всех чисел из заданного множества, но не сказано, каким образом выбирать числа из этого множества. Следовательно, предполагается, что исполнитель алгоритма умеет это делать с произвольным множеством. То есть, он в состоянии перебрать произвольное множество и ни разу не ошибиться, не взять одно и то же число дважды и ни одно число не пропустить. Пожалуй, исполнитель с такой способностью должен обладать довольно высоким уровнем интеллекта, а следовательно быть довольно сложным и дорогим устройством. Это не может быть приемлемо. Решаемая задача проста и должна решаться простым устройством. А это означает, что придется процедуру перебора множества чисел как-то описать. И естественно как-то доопределить само понятие множества.

Пусть, все элементы множества пронумерованы и пусть известно, сколько во множестве элементов, например – N. Тогда проблема решается легко. Исполнитель начинает перебор с элемента, имеющего нулевой номер, а для получения следующего элемента увеличивает текущий номер на 1.

Вариант 2:

Для Номера изменяющегося от нуля до числа N-1 с шагом 1
выполнять действие
    Если очередное число больше нуля то прибавлять его к сумме положительных.

Это уже значительно лучше, но согласитесь, выглядит очень громоздко. А сейчас пойдем по пути упрощения записи. Рассмотрим первую фразу:

Для Номера изменяющегося от нуля до числа N-1 с шагом 1
выполнять действие

Здесь описан процесс изменения некоторой переменной величины, которую мы назвали номером. Существенно в этой записи только то что:

- переменная имеет имя, и это не обязательно слово Номер;
- исходное значение переменной равно нулю;
- конечное значение переменной равно N-1;

- переменная изменяется с шагом 1;
- на каждом шаге изменения переменной выполняется некоторое действие.

Попробуем переписать фразу, так чтобы эти существенные пункты не изменили своего смысла, но фраза стала короче. Следующий вариант:

Для k=1 до **k=N-1** с шагом 1 делать

Выделенный фрагмент содержит еще одну возможность для упрощения. Записав k=0 мы уже дали исполнителю информацию о том, что в дальнейшем речь пойдет о переменной по имени k, поэтому k=N-1 это пожалуй лишний повтор, и окончательная запись окажется очень короткой:

Для k=1 до N-1 с шагом 1 делать

Последнее, запись выполнена на русском языке. Конечно, подобную запись можно выполнить и на немецком и на хинди и на китайском и любом другом языке, но так уж получилось, что в качестве основы языков программирования взят английский. Поэтому перепишем запись следующим образом:

FOR k:=0 TO N-1 BY 1 DO

и мы получим запись так называемой конструкции цикла на языке компонентный Паскаль. Если для вас этого не сложно, то постарайтесь сразу отметить, что такая форма цикла называется циклом с параметром или циклом с шагом. Далее, будем пользоваться вторым его названием.

Примечания:

- в записи k:=0 двоеточие обязательно;
- шаг 1 считается наиболее часто встречающимся. Поэтому если шаг равен 1, то в КП (Компонентный Паскаль) запись BY 1 можно опустить.

Займемся второй фразой алгоритма

если очередное число больше нуля, то прибавлять его к сумме положительных

Очередное число это величина. Все величины должны иметь имена. Так как все перебираемые числа принадлежат одному и тому же множеству, то логично их всех назвать одним именем, а различать по номеру. То есть a[1] это элемент множества «a» с номером 1. заметьте не первый элемент, а элемент с номером 1. Это существенная разница. a[k] – это соответственно элемент множества «a» с номером k. Такое множество пронумерованных элементов имеющих одно имя называется массивом.

Еще одна используемая величина это «сумма положительных», дадим ему имя sum и перепишем фразу так:

Если a[k]>0 то sum:=sum+a[k]

И перейдя на английский, получим еще одну команду языка КП

```
IF a[k]>0 THEN sum:=sum+a[k];
END;
```

Ключевое слово END не предусмотрено алгоритмом, оно является требованием языка. Правила языка требуют завершать сложную конструкцию таким ключевым словом. А условная команда является сложной конструкцией.

А теперь полная запись

Листинг 1

```
sum:=0;
FOR k:=0 TO N-1 DO
    IF a[k]>0 THEN sum:=sum+a[k];
    END;
END;
```

Здесь две сложных конструкции, поэтому два ключевых слова END. Одно из них закрывает условную команду, второе завершает цикл.

Важное замечание. В окончательной записи добавилась команда sum:=0 которой не было в исходной записи алгоритма. Для понимания необходимости этой команды посмотрим внимательнее на запись sum:=sum+a[k]. Что здесь происходит? Команда берет уже посчитанное значение величины sum (оно справа от знака :=) увеличивает это значение на величину a[k] и полученный результат присваивает снова величине sum, то есть результат становится новым sum. Таким образом, на втором шаге sum образуется от sum полученного на первом шаге. На третьем шаге sum образуется от sum полученного на втором шаге и далее все ясно. А вот чему равно самое первое sum, от которого мы должны получить новое sum на первом шаге цикла? В команде цикла об этом ничего не говорится. А раз так, то первое значение может оказаться равным чему угодно, его значение вообще говоря это какой-то числовой мусор который к моменту работы нашего фрагмента оказался в ячейках памяти хранящих величину sum.

Поэтому, программист должен перед началом процесса вычисления величины позаботится о ее исходном значении. И команда sum:=0 именно такую работу и выполняет, а называется это инициализацией, то есть определением первого, исходного значения.

Сравните полученную запись с тем, что было изначально и вы согласитесь, что язык программирования сохраняя смысл записи, позволяет ее очень существенно укоротить. Это конечно не законченная программа. До полноценных программ еще довольно далеко. Сейчас мы обсуждаем только общие вопросы. Поэтому запишем еще один короткий алгоритм и ответим на главный вопрос, для чего нужен язык программирования.

Задача 2. Найти сумму квадратов натуральных чисел от 1 до N.

Не будем тратить времени на длинные рассуждения, запишем сразу алгоритм на КП.

Листинг 2

```
sum:=0;
FOR k:=1 TO N DO
    sum:=sum+k*k;
END;
```

А теперь главный вопрос

Наверное, вы уже согласны, что запись на строгом языке сокращает текст, делает его чтение общедоступным для большого количества людей, то есть может быть общепринятым стандартом общения для специалистов занимающихся разработкой алгоритмов. Это важно, но не это главное. Самое важное в языке программирования то, что он является связующим звеном между естественным языком и языком который в действительности понятен компьютеру. Это дает возможность писать специальные программы – трансляторы способные переводить программы, написанные на алгоритмических языках в тексты уже малопонятные для человека, но исполняемые компьютером.

Для компьютера не существует таких понятий, как множество, массив, переменная. Он оперирует регистрами, адресами ячеек памяти и т.д. и т.п. То есть чем то очень далеким даже для того, кто неплохо владеет математическим аппаратом. Поэтому до появления языков – посредников программирование было уделом немногих, ибо требовало слишком больших усилий даже для написания несложных программ.

Вспомним еще раз, что программирование это постановка задачи, алгоритмизация и кодирование (запись на языке). Все три «слона» одинаково нужны для решения любой задачи, поэтому будем просто решать задачи и одновременно учиться и алгоритмизации и кодированию, а постановка задачи пока означает запись условия задачи строгими математическими терминами. Для тех кто желает заниматься программированием глубоко еще будет возможность убедиться, что постановка задачи достаточно сложный и трудоемкий процесс.

А сейчас главная проблема это расширение языкового аппарата. В общем-то весь язык программирования сводится к набору понятий для перечисления которых хватит пальцев на одной руке. Это:

- величина (переменная или константа);
- команда присваивания;
- конструкция цикла;
- условная конструкция;
- процедура.

Но пусть вас не расслабляет столь малый набор. Каждое из этих понятий будучи развито, до необходимого функционального уровня становится довольно емким и сложным для хорошего понимания, такого понимания, какого необходимо добиться если ваша цель серьезное и систематическое освоение программирования.

С четырьмя из пяти понятий мы уже встречались. Присваивание это действие обозначаемое знаком :=, его результатом будет вычисление выражения справа от знака и присвоение полученного результата величине чье имя находится слева от знака присваивания. Цикл позволяет многократно выполнить последовательность действий записанную только один раз. Условная конструкция позволяет выполнять ту или иную последовательность команд в зависимости от результата проверки условия. С процедурой мы пока не встречались, поэтому заметим лишь, что процедура это фрагмент программы который будучи записан один раз, может выполняться в разных точках программы. Про величины уже известно, что у них есть имя и значение, еще они имеют тип – описание позволяющее определить размер памяти для их хранения.

Задача 3. Арифметическая прогрессия задана тремя величинами.

- N – количество элементов прогрессии;
- a1 – значение первого члена прогрессии;
- d – разность прогрессии.

Вычислить сумму ее членов.

Конечно, для прогрессии существует формула суммы, но такие формулы есть не для любого числового ряда, а задачи счета каких-либо рядов встречаются достаточно часто. Поэтому на примере арифметической прогрессии посмотрим, что можно сделать, если математика не дает конкретной формулы.

Итак, что необходимо сделать:

- N – раз выполнить операции:
  - Расчета очередного члена прогрессии.
  - Прибавления его к уже известной сумме.
  - Соответствующий фрагмент на КП (с грубой ошибкой).

Листинг 3

```
sum:=0;
FOR k:=1 TO N DO
    a1:=a1+d;
    sum:=sum+a1;
END;
```

Тело цикла в нашем фрагменте состоит из двух команд присваивания, первая из которых a1:=a1+d находит значение очередного члена арифметической прогрессии, и вторая sum:=sum+a1 увеличивает значение суммы на величину только

что посчитанного члена. Логика вполне понятная, но есть в ней один изъян. Самый первый член прогрессии из процесса суммирования выпадает, так как уже на первом шаге расчетов к первому прибавляется величина d и он превращается во второй. Исправить ситуацию можно так:

Листинг 4

```
sum:=a1;
FOR k:=1 TO N DO
    a1:=a1+d;
    sum:=sum+a1;
END;
```

В этом варианте первый член будет учтен в момент инициализации величины суммы, из чего следует, что идея инициализации не сводится к обнулению, хотя конечно присваивание инициализируемой величине нуля встречается наиболее часто.

Условные циклы

Уже рассмотренный нами цикл с шагом, не единственная форма цикла и даже не самая сильная. Легко придумать задачу для которой цикл с шагом не даст решения. Ясно, что цикл с шагом хорош только тогда, когда программист точно знает сколько раз необходимо выполнить тело цикла. А это вполне может оказаться и не известным. Для примера вот такая задача:

Задача 4. Арифметическая прогрессия задана начальным членом a1 и разностью d. Необходимо найти номер члена N такого, что сумма прогрессии включая N-ый превысит некое заданное число W.

Это именно та ситуация в которой известно что делать:

- вычислять очередной член прогрессии;
- находить очередную сумму.

И не известно сколько раз это делать. Следовательно, пришло время расширить набор языковых конструкций. Вернемся на время к записи на русском языке (такая запись кстати называется псевдокодом). Общая конструкция решения такова:

```
Пока Сумма<= Предельного значения делать
        Вычислить очередной член прогрессии
        Вычислить очередное значение суммы
```

Теперь, то же самое на КП

Листинг 5

```
sum:=a1;
WHILE sum<=W DO
    a1:=a1+d;
```

```
    sum:=sum+a1;
END;
```

Данная конструкция называется циклом с условием продолжения. Это означает, что тело цикла (команды записанные между заголовком и ключевым словом END) выполняется до тех пор пока истинно условие записанное после ключевого слова WHILE (Пока). Отметьте себе, что условие проверяется на каждом шаге цикла, причем сначала проверяется условие и лишь затем выполняются команды тела цикла. Это например, означает, что тело цикла может быть не выполнено ни разу, если в момент входа в цикл условие окажется ложным.

Данный фрагмент разъясняет работу новой конструкции, но не решает поставленную задачу. Задача же будет решена в том случае, если по завершению работы цикла какая-либо величина окажется равна номеру очередного члена прогрессии. Ниже уточненный вариант программы:

Листинг 6

```
sum:=a1;
k:=1;
WHILE sum<=W DO
    a1:=a1+d;
    sum:=sum+a1;
    k:=k+1;
END;
```

Величина k увеличивается на 1 на каждом проходе тела цикла и фактически равна номеру суммируемого члена прогрессии.

Цикл WHILE наиболее универсальная форма цикла, позволяющая смоделировать любой процесс, чего нельзя сказать о FOR (цикле с шагом). Но за эту универсальность надо платить тщательным построением условия продолжения цикла. Ошибка в построении условия может привести к так называемому зависанию, то есть бесконечному выполнению цикла. И вот тому простой пример:

Листинг 7

```
k:=1;
WHILE k<5 DO
    sum:=sum+k;
END;
```

В данном фрагменте некая величина k складывается с суммой на каждом шаге цикла, но цикл никогда не завершится, так как начальное значение величины k известно, но в цикле оно никак не изменяется и следовательно всегда будет меньше 5. Правильный фрагмент может выглядеть например так:

Листинг 8

```
k:=1;
WHILE k<5 DO
```

```
    sum:=sum+k;
    k:=k+1;
END;
```

Как изменяется k конечно определяется задачей, и необязательно оно изменяется с шагом 1. Но внесенное исправление по крайней мере решает проблему зависания. Цикл выполнит несколько шагов и при k=5 прекратит свою деятельность.

Цикл WHILE это цикл с условием продолжения. И в КП есть так называемый цикл с условием завершения. То есть конструкция, в которой сначала выполняется тело цикла и лишь затем проверяется условие. На псевдокоде такая конструкция будет выглядеть так:

```
Повторять
    Вычислить очередной член прогрессии
    Вычислить очередное значение суммы
Пока Сумма > Предельного значения (не станет больше)
```

А на КП это же запишется так:

Листинг 9

```
sum:=a1;
k:=1;
REPEAT
    a1:=a1+d;
    sum:=sum+a1;
    k:=k+1;
UNTIL sum>W;
```

Здесь просто переписаны уже известные вычисления в новой форме. Давайте проанализируем, как это работает и нет ли проблем. Работает цикл так: На каждом шаге выполняется тело цикла и лишь затем проверяется условие. Следовательно, тело цикла будет выполнено хотя бы один раз. Цикл завершает свою работу при истинном условии. Отметьте существенное различие от цикла с условием продолжения. WHILE выполняет свою работу пока условие истинно, а REPEAT UNTIL до тех пор пока условие не станет истинным. Поэтому и появилось различие в записи условия. Еще одно отличие формы записи в том, что цикл с условием завершения не нуждается в ключевом слове END. Его тело, это все команды находящиеся между ключевыми словами REPEAT и UNTIL.

Есть в записи цикла REPEAT и небольшая содержательная проблема. Заметим, что и как в случае условия продолжения сумма инициализируется первым членом прогрессии, плюс к тому цикл REPEAT гарантированно посчитает еще один член прогрессии, то есть второй. Следовательно, если уже первый член прогрессии окажется больше чем W программа ошибется и завершит работу при k=2, при правильном ответе k=1. Таким образом форма цикла REPEAT существенно меняет логику и правильный вариант программы будет таков:

Листинг 10

```
sum:=0;
k:=0;
REPEAT
    a1:=a1+d;
    sum:=sum+a1;
    k:=k+1;
UNTIL sum>W;
```

Циклы с условием завершения и условием продолжения взаимозаменяемы и оба они годны для замены цикла с шагом. Продемонстрируем эту взаимозаменяемость еще одним примером.

Задача 5. Вычислить факториал числа N.
Форма цикла с шагом:

Листинг 11

```
fact:=1;
FOR k:=2 TO N DO
    fact:=fact*k;
END;
```

Форма цикла с условием продолжения:

Листинг 12

```
fact:=1;
k:=2;
WHILE k<=N DO
    fact:=fact*k;
    k:=k+1;
END;
```

Форма цикла с условием завершения:

Листинг 13

```
fact:=1;
k:=2;
REPEAT
    fact:=fact*k;
    k:=k+1;
UNTIL k>N;
```

Итак, что уже известно.
К данному моменту мы довольно детально рассмотрели три вида циклов. В каждом программном фрагменте использованы команды присваивания, в первой за-

даче использована условная конструкция, но информации о ней пока конечно не достаточно. В каждом фрагменте использовалось понятие переменной, но и оно пока никак не раскрыто. На текущий момент переменная в нашем представлении это целое число. Поэтому следующая учебная задача – это условная конструкция, после чего уже будет совершенно необходимо расширить представление о типах переменных. Но пока закрепим полученную информацию небольшим самоконтролем.

Задачи для самоконтроля:

1. Напишите три варианта (для каждой из трех форм цикла) программного фрагмента суммирования N последовательных натуральных чисел: 1+2+3+…+N
2. Дано два целых числа a и b. Найти значение выражения a^b, форма цикла на ваше усмотрение.
3. Определить номер (в натуральном ряду) четного числа, такого, что сумма всех предыдущих четных включая данное больше заданного W. Будем считать, что 2 имеет номер 1, 4 номер 2 и т.д.
4. Найти сумму квадратов натуральных чисел от 1 не превышающую заданное число W. Задачу решить в двух вариантах: циклом с условием продолжения и циклом с условием завершения.
5. Не пользуясь формулой суммы найти сумму членов геометрической прогрессии заданной начальным членом, количеством членов прогрессии и ее знаменателем. Используйте для решения цикл с шагом.
6. Найти произведение двух чисел A и B не пользуясь операцией умножения. Выбор формы цикла на ваше усмотрение.
7. Вычислить N членов ряда Фиббоначи. Ряд Фиббоначи это ряд чисел определяемый следующими условиями: $a_1=1$; $a_2=1$; $a_i=a_{i-1}+a_{i-2}$. Выбор формы цикла на ваше усмотрение.
8. Найти остаток и частное от деления числа A на меньшее число B. Операциями нахождения остатка и деления пользоваться запрещается. Форма цикла на ваше усмотрение.
9. Найти сумму первых N – нечетных чисел. Выбор цикла на ваше усмотрение
10. Арифметическая прогрессия задана начальным членом и разностью. Геометрическая прогрессия задана начальным членом и знаменателем. Выяснить номер k при котором член геометрической прогрессии станет впервые больше члена арифметической прогрессии. Все величины – целые, положительные числа.

Общая структура программы на КП

Все написанные ранее примеры обладают одним существенным недостатком, они не являются полноценными программами, которые можно запустить и получить требуемый результат. К настоящей главе, у вас уже должно выработаться неплохое

представление о том, что есть такое небольшая программа на КП и должен возникнуть вопрос, а как довести разобранные задачи до полноценной, результативной программы. Рассмотрим проблему на следующем уже решенном примере – расчета факториала:

Листинг 14

```
fact:=1;
k:=2;
WHILE k<=N DO
    fact:=fact*k;
    k:=k+1;
END;
```

Итак, что очень важное здесь отсутствует. Во-первых, по завершению работы фрагмента величина fact содержит значение факториала, но мы его не увидим, так как нет операции вывода на экран посчитанного значения, во-вторых, для работы фрагмента необходимо как-то задать исходное значение величины N, которая можно сказать является аргументом для расчетного процесса, но это тоже не сделано. Дополним фрагмент необходимыми командами:

Листинг 15

```
In.Open;
In.Int(N);
fact:=1;
k:=2;
WHILE k<=N DO
    fact:=fact*k;
    k:=k+1;
END;
StdLog.Int(fact);
```

Команда In.Int(N) читается так: взять из входного потока одно целое значение и присвоить его переменной N. Команда StdLog.Int(fact) читается так: передать в выходной поток целое значение переменной fact. Команда In.Open открывает входной поток данных, действие без которого команда In.Int не будет иметь смысла. Точное определение понятий входного и выходного потока нам пока не нужно, достаточно знать, что входной поток позволяет вводить данные с клавиатуры, а отсылка данных в выходной поток, позволяет визуально увидеть величину.

Далее, для того, чтобы программу на КП можно было запустить на выполнение она должна иметь имя. Это вполне естественное требование, нельзя обратиться к тому, что не имеет имени. Завершенный программный фрагмент называется процедурой и выглядит следующим образом:

Листинг 16

```
PROCEDURE Calculation;
BEGIN
```

```
In.Open;
In.Int(N);
fact:=1;
k:=2;
WHILE k<=N DO
    fact:=fact*k;
    k:=k+1;
END;
StdLog.Int(fact);
END Calculation;
```

Слово PROCEDURE означает, что ниже записан логически завершенный фрагмент программы, который на КП называется процедурой. Далее, после ключевого слова записывается имя процедуры и наконец между ключевыми словами BEGIN и END записывается текст процедуры. Эту процедуру уже можно попытаться исполнить, но к сожалению безуспешно. В тексте не хватает еще несколько важных вещей.

Процедура действительно является логической единицей, но не вполне самодостаточной. Процедуры КП объединяются в модули. Это необходимо, даже в том случае, если процедура в модуле будет только лишь одна. Дополним наш текст:

Листинг 17

```
MODULE Example;
PROCEDURE Calculation;
BEGIN
In.Open;
In.Int(N);
fact:=1;
k:=2;
WHILE k<=N DO
    fact:=fact*k;
    k:=k+1;
END;
StdLog.Int(fact);
END Calculation;
END Example.
```

Ключевое слово MODULE означает начало описания тела модуля, которое состоит из процедур и различной другой вспомогательной информации. После MODULE записывается имя модуля. Обратите внимание, на повтор имен модуля и процедуры после соответствующих END. Это обязательно. Кроме того, END завершающий модуль записывается с точкой, после имени модуля, а после END завершающего процедуру записывается точка с запятой, после имени процедуры.

В любом языке программирования и КП в том не исключение есть одно важное правило: любое имя, используемое в программе необходимо описать, то есть должно быть известно, что это такое и как с ним работать. К примеру слова WHILE,

DO, END, BEGIN, PROCEDURE, MODULE и некоторые другие являются ключевыми словами КП и они известны компилятору изначально. Но ввести в память компилятора все возможные команды невозможно по двум причинам: во-первых, их слишком много, во-вторых, никто не знает, что может еще понадобиться программистам. Поэтому языки программирования создают расширяемыми. Часть команд объявляют стандартом языка и эти команды компилятору заранее известны, а часть команд находится в так называемых библиотеках (модулях), то есть дополнительных файлах содержащих имена команд и их исполняемый код. В нашем фрагменте такими, библиотечными командами являются команды ввода/вывода. In и StdLog имена модулей – библиотек. Int это собственно команда указывающая на выполняемое действие (ввод и вывод величины определенного типа, в нашем случае целой величины). Но для использования библиотечных команд компилятору необходимо сообщить, что та или иная библиотека (в дальнейшем будем использовать только термин модуль) будет использоваться. Перепишем модуль с необходимыми дополнениями:

Листинг 18

```
MODULE Example;
IMPORT In, StdLog;
PROCEDURE Calculation;
BEGIN
In.Open;
In.Int(N);
fact:=1;
k:=2;
WHILE k<=N DO
    fact:=fact*k;
    k:=k+1;
END;
StdLog.Int(fact);
END Calculation;
END Example.
```

Последний шаг. Мы сообщили компилятору, что команды ввода/вывода находятся в модулях In и StdLog , ключевые слова ему и так известны, сейчас осталось определить переменные: N, fact, k. Точнее, определить их тип. Необходимо это вот для чего. Компилятор до запуска программы распределяет оперативную память компьютера. Для чего это нужно вопрос достаточно обширный и детально мы его рассматривать не будем, пока ограничимся следующей версией – если этого не сделать, то в процессе работы программы может случиться конфликт между различными структурами данных претендующих на одну и ту же память. Причем компилятор помочь программисту в разрешении этого конфликта не сможет, так как в процессе работы программы компилятора уже нет.

Определение типа выполняется в определенном блоке, который можно создать в каждой отдельной процедуре, а можно и в модуле. Запишем окончательно работоспособный модуль:

Листинг 19

```
MODULE Example;
IMPORT In, StdLog;
PROCEDURE Calculation*;
VAR
    N,k,fact:INTEGER;
BEGIN
In.Open;
In.Int(N);
fact:=1;
k:=2;
WHILE k<=N DO
    fact:=fact*k;
    k:=k+1;
END;
StdLog.Int(fact);
END Calculation;
END Example.
```

Этот вариант уже можно запустить на выполнение и получить результат, но сначала три небольших, но очень важных замечания:

Замечание о регистре символов. Для КП маленькие и большие буквы, это разные буквы. Например, переменные fact, Fact, FACT это три различных переменных. Ключевые слова в КП обязательно пишутся заглавными буквами. Поэтому, запись ключевого слова while или While будет воспринята как ошибочная.

Замечание о символе «».* Заметьте, что в последней версии нашего модуля после имени процедуры появился символ звездочка. Это означает, что к данной процедуре можно получить доступ из среды BlackBox, то есть попросить среду выполнить данную процедуру и эту процедуру можно вызвать из других модулей.

Блок определения переменных. В нашем примере переменные величины объявлены в теле процедуры Calculation. Это означает, что переменные известны только процедуре Calculation. Если в данном модуле будут описаны другие процедуры, то для них эти переменные окажутся недоступными. Переменные можно описать перед всеми процедурами после имени модуля и объявления требующихся модулей. Вот так:

Листинг 20

```
MODULE Example;
IMPORT In, StdLog;
VAR
    N,k,fact:INTEGER;
PROCEDURE Calculation*;
BEGIN
In.Open;
In.Int(N);
fact:=1;
```

```
k:=2;
WHILE k<=N DO
    fact:=fact*k;
    k:=k+1;
END;
StdLog.Int(fact);
END Calculation;
END Example.
```

В данном варианте объявленные переменные уже смогут использоваться всеми процедурами модуля. А хорошо это или плохо, надо это или нет определяется только логикой задачи и замыслом программиста.

Задачи для самоконтроля:

Доведите решения 10 задач из первого задания до завершенных программ.

Условный оператор

Мы уже немного касались работы условного оператора. Сейчас займемся им более детально. Начнем с примера.

Задача 6. Дано три целых числа: a, b, c. Выяснить могут ли они быть сторонами треугольника.

Геометрия утверждает, что в любом треугольнике сумма двух любых его сторон больше третьей. Это означает, что необходимо и достаточно проверить три неравенства:

a + b > c и a + c >b и b + c >a.

Если мы проверим первое неравенство и оно окажется ложным, то следующее неравенство уже можно и не проверять, но если оно окажется истинным, то ситуация останется неопределенной и потребуется проверка второго условия, если же и второе окажется истинным, то потребуется проверка третьего. Это можно записать так:

```
Если a + b > c то
    Если a + c >b то
        Если b + c >a то "Это стороны треугольника"
```

А сейчас то же самое на КП

Листинг 21

```
IF a + b > c THEN
    IF a + c >b THEN
        IF b + c >a THEN StdLog.String('Это стороны треугольника');
        END;
    END;
END;
```