```
MODULE i21егэDemo2010C4en;
(* не проверено по УСЛОВИЮ; русский вариант; см. инструкции в конце документа *)
  IMPORT  Log := StdLog,  In := i21sysIn,  Math;

  VAR
    минЦена, сколько: ARRAY 3 OF INTEGER;
    осталось, i, марка, цена: INTEGER;
    x: ARRAY 21 OF CHAR;

BEGIN
  FOR i := 0 TO 2 DO
    минЦена[ i ] := 3001;  сколько[ i ] := 0
  END;

  In.Open;
  In.Int( осталось );
  WHILE осталось > 0 DO
    In.String( x );  In.String( x ); (* пропускаем пару цепочек в данных *)

    In.Int( марка );
    IF  марка = 92  THEN  марка := 0
    ELSIF  марка = 95  THEN  марка := 1
    ELSE  марка := 2
    END;

    In.Int( цена );  ASSERT( (цена >= 1000) & (цена <= 3000) );

    IF цена < минЦена[ марка ] THEN
      минЦена[ марка ] := цена;
      сколько[ марка ] := 1
    ELSIF цена = минЦена[ марка ] THEN
      сколько[ марка ] := сколько[ марка ] + 1
    END;
    осталось := осталось - 1;
  END;

  Log.Int( сколько[0] );
  Log.Int( сколько[1] );
  Log.Int( сколько[2] );
  Log.Ln;
END i21егэDemo2010C4en.
```

```
МОДУЛЬ i21егэDemo2010C4ru;
(* не проверено по УСЛОВИЮ; англ. вариант; см. инструкции в конце документа *)
(* см. также Использование неанглийских ключевых слов *)
  ПОДКЛЮЧИТЬ  Вывод := i21eduВывод,  Ввод := i21eduВвод;

  ПЕРЕМЕННЫЕ  осталось, i, марка, цена: ЦЕЛЫЕ;
    минЦена, сколько: МАССИВ 3 ИЗ ЦЕЛЫХ;  x: МАССИВ 21 ИЗ ЛИТЕР;

НАЧАЛО
  ДЛЯ  i := 0  ДО  2  ДЕЛАТЬ
    минЦена[ i ] := 3001;  сколько[ i ] := 0
  КОНЕЦ;

  Ввод.Открыть;
  Ввод.Цел( осталось );
  ПОКА осталось > 0 ДЕЛАТЬ
    Ввод.Цепочка( x );  Ввод.Цепочка( x ); (* пропускаем пару цепочек в данных *)

    Ввод.Цел( марка );
    ЕСЛИ  марка = 92  ТО  марка := 0
    АЕСЛИ  марка = 95  ТО  марка := 1
    ИНАЧЕ  марка := 2
    КОНЕЦ;

    Ввод.Цел( цена );  УБЕДИТЬСЯ( (цена >= 1000) & (цена <= 3000) );

    ЕСЛИ цена < минЦена[ марка ] ТО
      минЦена[ марка ] := цена;
      сколько[ марка ] := 1
    АЕСЛИ цена = минЦена[ марка ] ТО
      сколько[ марка ] := сколько[ марка ] + 1
    КОНЕЦ;
    осталось := осталось - 1
  КОНЕЦ;

  Вывод.Цел( сколько[0] );
  Вывод.Цел( сколько[1] );
  Вывод.Цел( сколько[2] );
  Вывод.НовСтрока;

КОНЕЦ i21егэDemo2010C4ru.
```

Отличия от старого Паскаля

Быстро увидеть главные отличия Оберона/Компонентного Паскаля от старого Паскаля можно на примере решения задачи из ЕГЭ (задача С4 из демо-варианта ЕГЭ по информатике за 2010 год); вот решение (и версия с русскими ключевыми словами: рус.) Полезно их открыть (кликом по рссылкам) и справляться по ним при дальнейшем чтении.

Ключевые слова — заглавными буквами

Это удачное решение было введено ещё в Модуле-2. Оно позволяет избежать типографских преступлений вроде выделения всех ключевых слов жирным шрифтом, помогает легче видеть структуру программы, упрощает компилятор, упрощает инструментарий (устраняет необходимость в подсветке синтаксиса) — и, соответственно, освобождает цвет для гораздо более важных вещей (например, выделение новых фрагментов; хорошо известно, что ошибки чаще всего связаны с изменением программ).
Возможный дискомфорт здесь связан почти исключительно со старыми привычками и быстро проходит. «Почти» — потому что дизайнеры-шрифтовики не знают о таком использовании заглавных букв и проектируют их без учёта эстетических требований, возникающих в программировании. Однако эта проблема решается, по-видимому, достаточно легко (хотя бы заменой заглавных букв в шрифте капителью).

Люди с пылким воображением сразу начинают опасаться, что придётся слишком часто нажимать Shift, — однако это опасение напрасное: набрав одну-две, редко три первых (маленьких) буквы синтаксической конструкции и нажав F5, получаем *всю* конструкцию в раскрытом виде, так что теперь нужно гораздо меньше нажатий клавиш, чем в том же Турбо Паскале. Подробней об этом:
Об F5 в школьной версии (новый документ)
Об использовании клавиши F5 ... (старый документ)

Вместо слова Program — MODULE

Программы могут состоять из многих модулей, и все они устроены одинаково. В этом есть глубокий идеологический смысл, о котором подробней здесь.
Общая структура программы/модуля/процедуры сохранилась: есть раздел объявлений и есть выполняемое тело. Они идут друг за другом, и в Обероне/Компонентном Паскале обычно ясно разделены ключевым словом BEGIN/НАЧАЛО (см. примеры; см. ниже о том, куда делись обычно многочисленные в старом Паскале begin-ы).

Имена модулей и процедур нужно повторять после соответствующего END

Это помогает разбираться в программе и человеку, и компилятору: предотвращаются ситуации, когда из-за одной опечатки в процедуре компилятор усеивает следующие процедуры фиктивными ошибками вместо продуктивной проверки.

Комментарии имеют вид (* ... *)

Разрешён только этот тип комментариев. Они могут быть вложены, и если они вложены, то должен соблюдаться баланс составных скобок (* и *).

> Фигурные скобки используются для записи явных констант встроенного типа SET (МНОЖЕСТВО), который предназначен в Обероне/Компонентном Паскале прежде всего для работы с битами.

Об отступах

Этот пункт не относится к собственно языку, но слишком часто можно видеть школьные программы, в которых все инструкции начинаются с первой позиции.
Поэтому важно будет почитать Об отступах.

Инструкция IMPORT — подключение внешних библиотек (модулей)

Не углубляясь в проблематику построения программ из модулей (см. об этом здесь), просто скажем, что список модулей, которые нужно использовать в нашей программе, должен быть дан через запятую после ключевого слова IMPORT/ПОДКЛЮЧИТЬ.
В этом списке можно дать модулям сокращённые имена-псевдонимы для использования на протяжении данного модуля.

О том, как по именам узнать, где искать исходник модуля, можно прочитать в отдельном документе: О подсистемах (правила хранения модулей).

Часто используется библиотечный модуль StdLog, обеспечивающий вывод в Рабочий журнал (Log). Часто используется также i21sysIn для ввода данных (см. об этом ниже раздел о вводе-выводе). Их русскоязычные варианты: i21eduВывод и i21eduВвод.
Другие модули, часто используемые в учебном контексте:
Math — функции синус, косинус и т.п. (например, вычисление синуса: Math.Sin(x));
i21eduЧерепашка — модуль с командами управления черепашкой;
i21eduTPGraphics — модуль для рисования, достаточно близко воспроизводящий графику Турбо Паскаля.
В учебном контексте удобно использовать заготовки модулей с готовыми списками импорта в зависимости от изучаемой темы. См. об этом здесь

Ввод-вывод вынесен из языка в библиотеки

... причём ещё со времён Модулы-2 (1980). Примитивный «стандартный» ввод-вывод старого Паскаля, пригодный лишь для простых задач, не оправдывает усложнения компилятора. Серьёзный ввод-вывод — штука достаточно сложная и, как для всех сложных штук, требования к нему варьируется от приложения к приложению.

В частности, учебный контекст с его особой спецификой тоже требует особого подхода. По всей видимости, лучшее решение здесь — документо-ориентированный ввод-вывод. В примере это все вызовы процедур вида In.*** (ввод данных; см. инструкцию после текста модуля) и Log.*** (вывод в рабочий журнал; что это за журнал, объясняется здесь: Меню, документы, Рабочий журнал).
Подробней: Справка по учебному вводу-выводу

Что касается штатного ввода-вывода в Блэкбоксе, то тут есть два важных пункта:

во-первых, при полном использовании в прикладной программе средств Блэкбокса применять ввод-вывод низкого уровня с явным обращением в программе к файлам нужно лишь в специальных приложениях, имеющих дело с данными очень большого объёма;

во-вторых, средства ввода-вывода, как и весь Блэкбокс, построены систематически в парадигме «компонентно-ориентированного программирования» (это похоже на когда-то модное ООП, только круче). Соответственно, здесь требуется понимание языка вообще и реализации в нём средств ООП в частности.
Для справок:
Files — штатная документация для модуля Files (побайтовый ввод-вывод);
Stores — штатная документация для модуля Stores (чтение-запись всех элементарных типов Компонентного Паскаля).

VAR — объявления переменных

Смысл и структура раздела объявлений такая же, как в старом Паскале.
Комментария в примере заслуживают массивы:

Массивы

Элементы любого массива теперь нумеруются с нуля. Поэтому в объявлении указывается только общая длина массива.

> Начинать нумерацию с нуля — оптимальное соглашение. С одной стороны, именно такая нумерация удобна чаще всего (ср. аргументы Дейкстры о том, что в программировании нумерации должны начинаться с нуля). С другой стороны, именно с таким соглашением компилятор получается наиболее простым и эффективным (в том числе по качеству машинного кода).

Например, a: ARRAY 3 OF INTEGER — описание массива a из трёх элементов a[0], a[1], a[2], которые все суть переменные типа INTEGER.
Вот ещё основные возможности, связанные с массивами:

— В Компонентном Паскале можно объявлять массивы-параметры без указания длины (открытые массивы), так что процедуры могут обрабатывать массивы любой длины.

— Длину массива всегда можно узнать с помощью функции LEN(a), передавать длину отдельным параметром не нужно.

— Динамические массивы — то есть такие, длина которых определяется уже при выполнении программы — объявляются с помощью указателей (например, a: POINTER TO ARRAY OF INTEGER) и размещаются в памяти посредством инструкции NEW (например, NEW(a,3)). В остальном их использование почти не отличается от статических массивов. Подробней см. Сообщение о языке.

— Допустимы массивы любой размерности. Например, двумерные массивы:
VAR матрица: ARRAY 3, 3 OF INTEGER.

— Массивы литер служат для хранения литерных цепочек.

Литерные цепочки («строки»)

В Компонентном Паскале литерные цепочки (в кавычках вида "это цепочка") хранятся в литерных массивах (ARRAY OF CHAR). Длина массива должна быть больше (хотя бы на единицу), чем длина самой длинной цепочки, которую в нём планируется хранить. Лишний элемент массива нужен, чтобы в нём всегда могла поместиться терминальная литера 0X.
В примере объявлен массив x: ARRAY 21 OF CHAR — значит, в нём можно хранить цепочки длиной до 20 литер включительно, что требовалось в условии задачи.
Возможны присваивания вида x := "новое значение для x", когда массив «заряжается» сразу целой цепочкой.
Можно считывать цепочку из потока вввода целиком: In.String(x).
Или целиком печатать в рабочий журнал: Log.String(x).
См. еще примеры в i21eduВыводПример0, i21eduВводПример2

Нет «мусорных» begin

Теперь ключевое слово BEGIN появляется лишь как заголовок исполняемого тела процедур и модулей. Это видно в примере.
Уже к 1980 г. стало очевидно, что решение проблемы группировки операторов с помощью блоков begin-end — неудачное. Оно унаследовано старым Паскалем из Алгола-60 просто по инерции: внимание Н.Вирта в 1969 г. было поглощено новыми аспектами языка, до полировки старых руки дошли позже.

Более простое и чёткое решение — просто потребовать, чтобы каждая составная конструкция — цикл, условный оператор и т.д. — заканчивалась ключевым словом END. Например, условный оператор теперь выглядит так:

```
IF i > 0 THEN
  любая последовательность операторов
```

```
  ELSE
    другая последовательность операторов
  END;
```

Соответственно, нет путаницы с тем, что к чему относится, и можно спокойно разрешить ветвям содержать любое число операторов, включая циклы и т.п.
Ветка ELSE может отсутствовать, — но закрывающий END должен быть всегда.

Ещё одно важное усовершенствование: в условном операторе между ветками IF и ELSE можно поставить любое количество дополнительных веток вида:

```
  ELSIF другое условие THEN
    другая последовательность операторов
  ...
```

Условия всех веток условного оператора («охраны») проверяются последовательно, начиная с ветки IF, пока не будет найдена та охрана, которая даёт TRUE.

Даже в ЕГЭ-шном примере есть сразу два примера условных операторов с ELSIF.
Ясно, что последовательный выбор из нескольких вариантов встречается постоянно и поэтому его следует считать одной из базовых алгоритмических схем. Приятно, что теперь такой выбор можно выражать просто и чётко.

Раз уж речь зашла о многоветочных операторах: многоветочный вариант цикла WHILE (т.наз. цикл Дейкстры) — тоже фундаментальная алгоритмическая схема, которую в профильных курсах, видимо, нужно вводить уже в школе. См. О цикле Дейкстры.

Цикл FOR/ДЛЯ

... записывается примерно так:

```
FOR i := 0 TO n-1 DO
  последовательность операций
END
```

Если нужно идти через число, то в заголовке цикла нужно написать TO n-1 **BY 2** DO.
BY используется и для того, чтобы Если идти вниз по индексу:

```
FOR i := n-1 TO 0 BY -1 DO
  последовательность операций
END
```

В новом, Компонентном Паскале уже не нужно заморачиваться тонкостью о том, что значение управляющей переменной цикла по выходе из цикла оказывается неопределённым. Оно вполне себе определено, так как цикл FOR/ДЛЯ определяется через более фундаментальный цикл WHILE/ПОКА, см. определение в документе Сообщение о языке Компонентный Паскаль. Кому нужны тонкости, пусть лучше сосредоточится на понятии инварианта цикла.

Должно быть стыдно так забивать головы юным программерам FORами и ARRAYями, что потом эти FORы и ARRAYи из их голов приходится выбивать годами, чтобы освободить место для культурных WHILE-ов, списков и последовательностей.
Программы из сплошных FORов и ARRAYев, — мрачное наследие раннего каменного века Эры Цифры, когда смелый пионер Джон Бэкус только недавно изобретёл первый язык программирования — фортран, записи были только в коболе, а списки — только в лиспе, а гениальный Эдсгер Дейкстра ещё не помышлял ни о структурном программировании, ни о цикле своего имени.

Цикл WHILE/ПОКА

Это основной цикл, вокруг которого, по-хорошему, должно вертеться программирование для начинающих (остальные виды циклов в курсе для начинающих, по большому счёту, только мешают).

Единственная особенность Компонентного Паскаля по сравнению со старым — это зафиксированное уже в определении языка правило укороченного вычисления логических выражений: если результат операции заранее известен по значению первого операнда (да **или** что угодно -> да), то второй операнд не вычисляется. Это правило как средство оптимизации применялось ещё во втором древнейшем языке программирования лиспе. Но его настоящий глубокий смысл для систематического построения алгоритмов был выявлен Дейкстрой. Простейший пример — цикл линейного поиска. См. пример из книжки Вирта: ADruS18 Поиск.

Точки с запятой

В Компонентном Паскале по сравнению со старым правила насчёт точек с запятой немного ослаблены: можно считать, что точка с запятой просто завершает оператор. То есть она может появиться перед END или сразу после BEGIN и т.д. Подробней об этом см. здесь.

Вот и всё по примеру.

Официальные документы:
- Сообщение о языке Компонентный Паскаль
- Отличия Компонентного Паскаля от старого