**Кафедра конструирования
и производства радиоаппаратуры**

УТВЕРЖДАЮ
Заведующий кафедрой КИПР
_____**В.Н. ТАТАРИНОВ**
“___” _____2012 г.

Модульное и структурное программирование

Лабораторная работа по дисциплинам **«**Информатика**»** для студентов специальностей 211000.62 «Конструирование и технология электронных средств» (бакалавриат) и 162107.65 «Информатика и информационные технологии» (специалитет)

Разработчик:
Доцент кафедры КИПР
_____**Ю.П. Кобрин**

Томск 2012

Оглавление

1 Цели работы

- Овладение методиками модульного и структурного программирования при создании «больших» программ.
- Изучение возможностей использования различных видов подпрограмм в ***Borland Pascal***.
- Освоение механизмов обмена данными с подпрограммами в ***Borland Pascal***.
- Получение практических навыков в программировании процедур и функций на ***Borland Pascal***.
- Получение практических навыков в разработке и использовании модулей ***Borland Pascal***.

2 Порядок выполнения работы

В ходе выполнения этой работы следует:

1) Изучить описание лабораторной работы, обратив особое внимание на достоинства технологий модульного и структурного программирования, на особенности структуры и использования различных подпрограмм, а также включения их в модули на ***Borland Pascal***. При необходимости использовать дополнительную литературу [2 - 9].
2) Ответить письменно на контрольные вопросы.
3) Войти в свой личный каталог, и настроить интегрированную среду ***Borland Pascal*** для последующей работы. Записать файл конфигурации в личный каталог.
4) В качестве индивидуального задания надлежит провести доработку разработанной на предыдущем лабораторном занятии [1] программы в соответствии с требованиями модульного и структурного программирования.
5) Внимательно проштудировать учебный пример и по возможности максимально использовать его в своей задаче.
6) Ввести и отладить разработанную программу.
7) Продемонстрировать работоспособность программы для различных вариантов исходных данных.
8) Оформить отчет по лабораторной работе и защитить его у преподавателя.

3 Контрольные вопросы

Ответьте письменно на следующие контрольные вопросы:

1) Какие проблемы возникают при создании «больших» программ?
2) Каковы цели структурного программирования? Перечислите известные Вам принципы структурного программирования.
3) Оцените разработанную Вами на предыдущем занятии программу: соответствует ли она требованиям структурного программирования?
4) В чем преимущества модульного программирования? Каковы его недостатки?
5) В чем отличие процедур и функций, для каких целей они предназначены?
6) Как объявляются и чем отличаются глобальные и фактические параметры?
7) Какие возможности представляет интегрированная среда ***Borland Pascal*** для отладки «больших» программ?

8) Что такое модуль? Перечислите стандартные модули в системе ***Borland Pascal***.
9) Какое расширение получает файл после трансляции модуля в системе ***Borland Pascal***?
10) Каково назначение интерфейсного раздела модуля?

4 Отчетность

Отчет должен быть выполнен в соответствии с [10] и состоять из следующих разделов:

1) Тема и цель работы.
2) Индивидуальное задание.
3) Схема алгоритма решения задачи.
4) Текст программы и вводимые тестовые исходные данные.
5) Откомпилированный текст программы-заготовки (в электронном виде).
6) Результаты выполнения программы.
7) Ответы на контрольные вопросы.
8) Выводы.

При защите отчета по работе для получения зачета студент должен:

- *уметь* отвечать *на контрольные вопросы*;
- обосновать структуру выбранного алгоритма и показать его работоспособность;
- уметь пояснять работу программы;
- продемонстрировать *навыки работы в среде* ***Borland Pascal.***

5 Разработка «больших» программ

5.1 Проблемы разработки сложных программ

Проектирование современных радиоэлектронных средств требует применения высококачественных, надежных и эффективных программных систем автоматизации инженерного труда. В то же время, несмотря на свою сложность, подобные программные системы должны быть легко модифицируемыми, чтобы их можно было бы без особых усилий приспосабливать на месте к стремительно меняющимся запросам производства.

Очевидно, что крупные программные проекты сверх всякой меры сложны для успешного проектирования, кодирования и отладки в приемлемые сроки. Крайне сложно с уверенностью сказать, что разработанная программа всегда выполняет то, что требуется, и что она не выполняет ничего такого, что не требуется.

Несомненно, что создание таких трудоемких программных комплексов, как, впрочем, и изделий любого другого сложного серийного производства, нереально без применения современных индустриальных технологий, связанных с высоким уровнем специализации и разделения труда.

Качество разрабатываемых программ во многом зависят ***от стиля программирования***.

Стиль программирования – это набор приемов, используемых опытными программистами чтобы получить правильные, эффективные, удобные для применения и легкочитаемые программы.

5.2 Несколько советов по улучшению стиля программирования

1. Не забывайте, что программы пишутся ***для людей***, а не для компьютеров.

2. Пользоваться программой не обязательно будете только Вы, и пользователь не обязательно будет подготовленным программистом. Продуманный, дружественный интерфейс компьютер-пользователь – основа продолжительной и успешной жизни Вашей программы.

3. ***Стремитесь к простоте.*** Избегайте использования языковых конструкций с неочевидной семантикой и программистских «трюков». Изощренная программа говорит, скорее всего, о ее плохом качестве.

4. Используйте ***имена с подходящей мнемоникой*** для переменных, файлов и других объектов программы.

5. Разумно использовать ***одинаковые префиксы*** в родственных группах идентификаторов[1].

6. Создавайте ***универсальные, обобщенные программы***. Это поможет предвидеть дальнейшие их модификации и варианты использования.

7. Делайте комментариев больше, чем это кажется необходимым[2].

8. Используйте ***вводные комментарии***. Вводные комментарии объясняют назначение программы или подпрограммы, назначение переменных и других данных, узловые моменты применяемых математических методов и т.п.

9. ***Пояснительные комментарии*** должны содержать дополнительную информацию, а не перефразировать программу

10. Для улучшения наглядности используйте: пропуски строк между отдельными фрагментами программы, пробелы между словами, переменными и другими объектами, отступы от начала строки для выявления структуры программы и данных.

11. Один оператор в строке – достаточно! Если все-таки для записи оператора требуется больше, чем одна строка, то все последующие строки записывайте с отступами.

12. Не допускать вложенности операторов ***if*** более трех уровней.

Кроме употребления этих простейших, но действенных советов, для повышения качества программ применяют особые технологии программирования. Наиболее популярными из них являются модульное и структурное программирование.

6 Структурное программирование

6.1 Постановка задачи

При разработке сложных программ широкое признание и практическое применение получили идеи ***структурного программирования*** [2, 3]. Идеи структурного программирования основаны на том, что человек гораздо легче читает и понимает какой-либо текст в том случае, если он читает фразы в порядке следования в тексте. Если в тексте есть много ссылок на другие страницы, сноски, примечания и т.п., то это явственно затрудняет и восприятие, и понимание читаемого текста.

---

[1] Например, все имена нестандартных пользовательских типов данных начинать с префикса ***t*,** констант – с префикса ***c***, файлов – с префикса ***f*** и т.п.

[2] Важность включения комментариев в текст программы отмечал Д. Ван Тассел [2]: «Некомментируемая программа – это, вероятно, наихудшая ошибка, которую может сделать программист, а также свидетельство дилетантского подхода (пусть даже программист имеет десятилетний опыт работы); комментарии подобны ориентирам в незнакомом лесу. Только неразумные не оставляют ориентиров».

Большие программы обычно содержат огромное количество разветвлений, циклов. Естественно, что они разрабатываются и читаются многими людьми. И если программы написаны с большим количеством ссылок (операторов безусловного перехода ***GoTo*****),** то возникают те же проблемы – они крайне неудобны для чтения и понимания. Трудно проследить логику работы таких программ, вносить в них изменения, тестировать правильность.

Структурное программирование позволяет организовать проектирование программ и процесс кодирования таким образом, что большинство логических ошибок можно предотвратить с самого начала[3]. Кроме того, не составляет большого труда обнаружить те ошибки, которые были допущены.

Структурное программирование включает три главные составляющие:

1. Проектирование сверху вниз (нисходящее программирование).
2. Модульное программирование.
3. Структурное кодирование.

6.2 Проектирование «сверху вниз»

Наиболее разумным подходом к познанию сложных объектов или процессов считается ***системный подход.*** Он заключается в **декомпозиции** (разделении) сложной и труднообозримой задачи на отдельные подзадачи (подсистемы), каждая из которых имеет относительно небольшие размеры и сложность и потому существенно понятнее и реализуемее. Структурное и функциональное объединение (**композиция**) реализованных подзадач (подсистем) позволяет найти решение исходной проблемы.

В современном программировании (как, впрочем, и в проектировании РЭС) выделяют два подхода: ***восходящий*** (снизу вверх) и ***нисходящий*** (сверху вниз).

Если в Вашем распоряжении имеется широкий набор уже готовых фрагментов программы, выполняющих некоторые нужные действия, то разработку программы можно начать именно с них. Постепенно производят целенаправленное наращивание возможностей программы – от частного к общему (снизу вверх). Основными проблемами при ***восходящем программировании*** являются большая сложность объединения и согласования отдельных фрагментов программы, а также тестирования программы в целом.

Если разрабатывается совершенно новая программа, для которой не существует хороших наработок, то ее целесообразнее программировать сверху вниз. ***Нисходящая разработка***[4] предусматривает сначала определение задачи в общих чертах, а затем постепенное итеративное уточнение программы путем внесения более мелких деталей. При этом задача разбивается на отдельные подзадачи. Верхний уровень являет нам наивысшую абстракцию, упрощает взгляд на программу, позволяет разработчику представить требуемое решение проблемы без сиюминутного учета множества частностей. На каждом шаге детализации выявляют цели, которые необходимо осуществить в каждой подзадаче. Если цели этих подзадач невозможно реализовать операторами алгоритмического языка, то выполняют разбиение на еще более мелкие подзадачи и т.д. Самый нижний уровень учитывает все мелкие детали реализации.

---

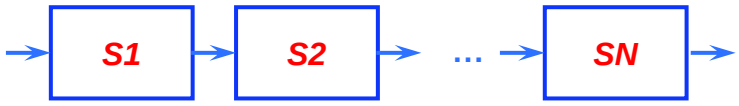
[3] Один из основоположников структурного программирования Эдсгер Дейкстра дал следующее определение: «**Структурное программирование – это дисциплина, которую программист навязывает сам себе**».

[4] Такой подход называется также *методом пошаговой детализации.*

Важным достоинством метода пошаговой детализации является то, что *на любом шаге разработки программы возможна тестовая проверка*, демонстрирующая логику работы программы и проявляющая ее слабые и критические места[5].

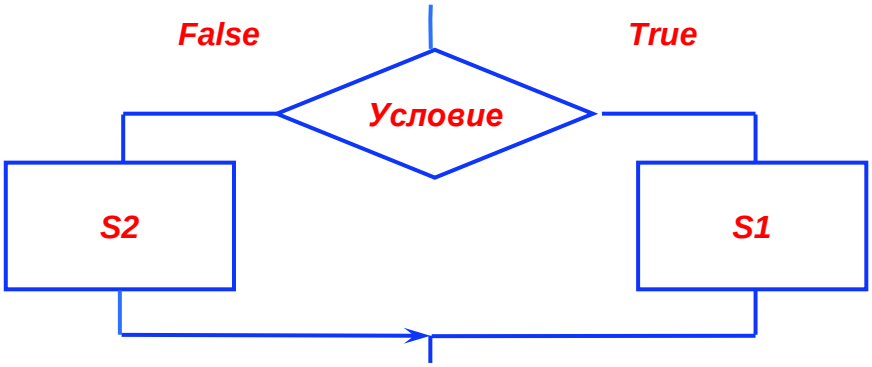
6.3 Структурное кодирование

Структурное кодирование состоит в получении правильной структурированной программы из *основных (базовых) логических структур* (*рис. 6.1 – 6.3*), каждая из которых имеет ***только один вход*** и ***только один выход***.



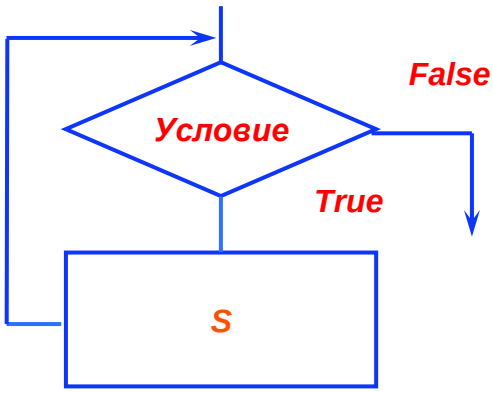

```
Оператор S1;
Оператор S2;
……..
Оператор SN
```

Рисунок 6.1 - *Логическая структура* «Следование»




```
if <Условие>
   then Оператор S1
   else Оператор S2;
```

Рисунок 6.2 - Логическая структура «Ветвление»




```
while <Условие> do
Оператор S;
```

Рисунок 6.3 - Логическая структура «Повторение»

Конструкция, представляющая собой последовательное выполнение двух или более операций, называется ***следованием***.

Конструкция, состоящая из развилки, двух операций и слияния, называется ***ветвлением***. Одна из операций может отсутствовать. Конструкция, имеющая

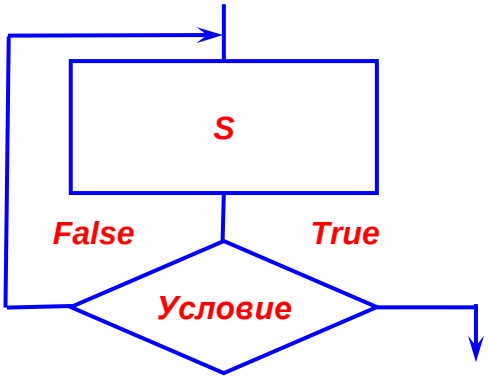
[5] Рекомендуется тестировать программу после каждого случая внесения изменений в программу. Очевидно, что ошибки, обнаруженные на ранней стадии исправлять гораздо проще, и обходятся они значительно дешевле. Целесообразно на начальном этапе тестирования использовать «заглушки».

линии управления, ведущие к предыдущим операциям или развилкам, называется ***циклом.***

Конструкции следование, ветвление и цикл можно представить как операции, так как они имеют единственный вход и единственный выход. Произвольную последовательность операций можно представить как одну операцию.

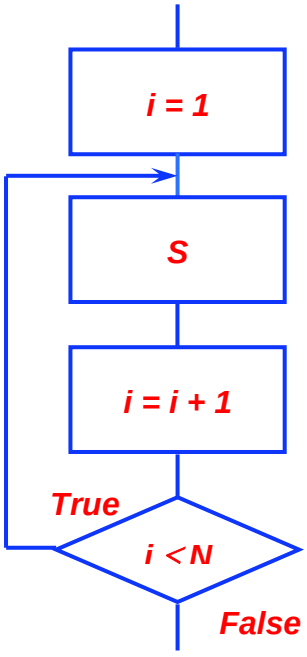
Базовые структуры могут легко комбинироваться друг с другом в любом сочетании, допускается их «вложение». На их основе можно создать любую корректную древовидную программу с одним входом, одним выходом, без зацикливаний и недостижимых команд.

Кроме приведенных базовых логических структур может быть использован *расширенный набор логических структур* (*рис. 6.4 – 6.7*), существенно облегчающих программирование и не нарушающих при этом структурированности программ.




```
Repeat
    Оператор S1;
    Оператор S2;
      ……..      ;
    Оператор SN;
until <условие> ;
```

Рисунок 6.4 - Логическая структура «Цикл с послеусловием»




```
for i := 1 to N do
    Оператор S;

или

for i := N downto 1 do
    Оператор S;
```

Рисунок 6.5 - Логическая структура «Цикл с заданным числом повторений»

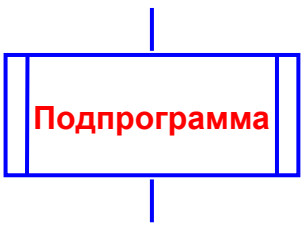
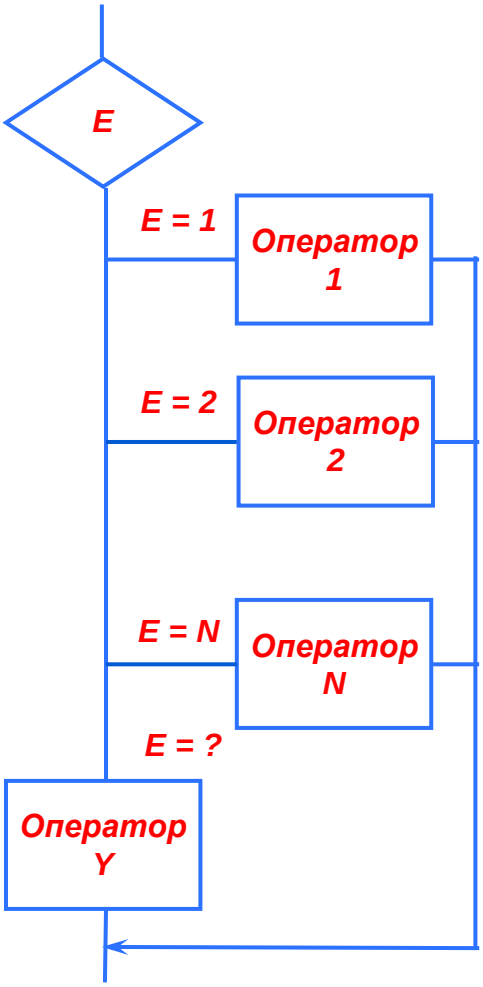


Рисунок 6.6 – Логическая структура «Вызов подпрограммы»



Формат оператора ***case***

```
case <выражение-селектор E> of
     <список 1>: Оператор 1;
     <список 2>: Оператор 2;
                 ……..      ;
     <список N>: Оператор N;
else Оператор Y
end;
```

Примечание. Выражение-селектор может быть только порядкового типа

Пример употребления оператора ***case***

```
Ch := …;  {какой-то символ}
Write(Ch, ‘ – символ  ’);

case Ch of
  ‘A’..’Z’, ‘a’..’z’: WriteLn(‘латинский’);
          ‘0’..’9’: WriteLn(‘цифра’);
    ‘-’, ’+’, ‘*’, ’/’: WriteLn(‘операция’)
else WriteLn(‘недопустимый’)
end;
```

Рисунок 6.7 - Логическая структура «Выбор»

7 Модульное программирование

7.1 Постановка задачи

Модульное программирование – это организация программы как совокупности небольших независимых блоков, называемые ***модулями***[6], структура и поведение которых подчиняются определенным правилам. В языке Паскаль роль модулей выполняют подпрограммы: ***процедуры и функции***. Процедуры и функции в свою очередь могут вызывать подпрограммы более низкого уровня и т.д. Следовательно, модульная программа имеет иерархическую структуру.

Часто некоторую последовательность действий требуется повторить в нескольких местах программы. Как правило, такие однотипные участки выполняют одни и те же вычисления, но с разными данными. Эти вычисления целесообразно выделить в подпрограммы, где описывается процедура вычисления некоторой подзадачи при формальных параметрах. В основной программе в нужных местах осуществляется обращение к подпрограмме при необходимых значениях фактических параметров.

[6] Заметим, что в ***Borland Pascal*** модулем также называется специальная синтаксическая конструкция для хранения программных ресурсов (***Unit***).

Кроме очевидной выгоды от сокращения размера разрабатываемых программ, расчленение программы на модули позволяет распараллелить работу, чтобы ее можно было бы исполнять коллективу программистов. Поэтому подпрограммы рекомендуется писать даже тогда, когда они вызываются однократно, и их использование явно не сокращает текста программы. Как показывает опыт, употребление подпрограмм оказывает весьма полезное влияние на стиль, качество и надежность разрабатываемых программных систем.

Очевидная выгода – хорошо зарекомендовавшие себя модули можно использовать как «крупные строительные блоки» и в последующих разработках.

Использование модульного программирования позволяет упростить тестирование программы и обнаружение ошибок. Аппаратно-зависимые подзадачи (например, драйверы) могут быть строго отделены от других подзадач, что улучшает мобильность создаваемых программ.

С целью повышения эффективности программ по быстродействию и затратам памяти модули можно многократно переделывать независимо от других, модифицируя и совершенствуя их внутренние алгоритмы.

Модульные программы значительно легче понимать, и их поведение гораздо надежнее и предсказуемее, чем у монолитных программ.

В языке Паскаль для построения модульных программ предусмотрены средства для организации подпрограмм двух видов: ***подпрограмм-процедур*** и ***подпрограмм-функций***. Существует возможность использовать как готовые (чужие), так и свои собственные подпрограммы.

Перед использованием подпрограмму необходимо объявить и описать. Это делается перед основным блоком вызывающей программной единицы[7]. Каждая подпрограмма объявляется только ***однажды***, а использовать ее можно многократно.

7.2 Требования к модулям

1. Программа должна разделяться ***на независимые части***, называемыми ***модулями***.

2. Модуль – это независимый блок, код которого физически и логически отделен от кода других модулей[8].

3. Модуль имеет только одну входную и одну выходную точку.

4. По возможности модуль должен представлять ***обобщенное*** решение некоторой ***единственной частной задачи***, чтобы его можно было использовать при решении других аналогичных задач.

5. Программный модуль ***не должен зависеть от контекста***, в котором он будет использоваться. Вы должны быть уверены, что не возникнет никаких ***побочных эффектов*** от применения любой подпрограммы. Для достижения такой независимости, кроме корректности самого алгоритма модуля, необходимо предусмотреть анализ допустимости входных и выходных значений, обработку особых и аварийных ситуаций (возможные «больших» «больших» ошибки в данных, сбои устройств и т.д.) и использование ***локальных переменных***, описание которых расположены внутри подпрограммы и только внутри нее действуют.

---

[7] Внутри подпрограмм могут быть объявления и других подпрограмм. Но эти подпрограммы считаются ***локальными***, так как доступны лишь внутри объемлющей их программной единицы.

[8] В предельном случае, если необходимо, чтобы выполняемое действие было гарантировано независимым от влияния других частей программы при любых коррекциях и изменениях, это может быть всего лишь один исполняемый оператор.

6. Следует стремиться к тому, чтобы из модулей можно было сформировать большие программы без каких-либо предварительных знаний о внутреннем устройстве модуля. Представление подпрограммы в виде абстрактного ***черного ящика***[9], реализующего определенные действия, позволяет уменьшить число одновременно рассматриваемых элементов программы. В результате программа может рассматриваться на новом иерархическом уровне абстракции - более простом и доступном для понимания.

7. Модули должны быть, по возможности, небольшого объема (обычно ***не более страницы текста***).

8. Организация модулей должна обеспечивать независимость их разработки, программирования и отладки. Это возможно, если модуль независим от источника входной и приемника выходной информации и от предыстории его использования.

9. Каждый модуль должен начинаться с комментария, объясняющего его назначение, назначение переменных, передаваемых в модуль и из него.

10. Идентификаторы всех переменных и модулей должны быть ***смысловыми***.

11. Использовать только стандартные управляющие конструкции (см. *разд. 6.3*).

12. Не использовать метки и операторы ***GoTo***.

13. В одной строке записывать только один оператор. Если для записи оператора требуется более, чем одна строка, то все последующие строки записываются с отступами (лучше всего, в две позиции).

7.3 Процедуры

Процедуры в Паскале применяют для решения определенной задачи или подзадачи.

Описание процедуры состоит из ***заголовка*** (имени) и ***тела процедуры***.

Заголовок процедуры состоит из зарезервированного слова ***procedure***, ***идентификатора (имени)*** – уникального в пределах программы; ***необязательного списка формальных параметров***, заключенных в круглые скобки (для каждого формального параметра указывается его тип).

Тело процедуры по структуре аналогично программе.

Описания объектов (меток, констант, типов, переменных и т.п.) внутри процедуры являются ***локальными***. Они действуют только в пределах этой процедуры. В теле процедуры можно также использовать (хотя и не очень желательно) любые ***глобальные***[10] типы, переменные и константы.

```
Procedure <имя> [(список формальных параметров)]; {заголовок процедуры}
{объявления локальных объектов процедуры}
const …;
type … ;
var … ;
   …
begin {начало тела процедуры}
   <операторы тела процедуры>
end; {конец тела процедуры}
```

[9] Принцип ***черного ящика*** подразумевает недоступность определенных в теле модуля понятий вне данного модуля.

[10] Объявления объектов, сделанные в вызывающей данную подпрограмму программной единице и доступные как в ней самой, так и во всех ее подпрограммах

Для вызова процедуры следует записать имя процедуры, перечислив в скобках фактические параметры того же типа и в том же порядке и количестве, что и формальные. Параметры обеспечивают механизм замены, который позволяет выполнять процедуру с различными начальными данными.

Пример 1. Написать программу, в которой используются процедуры, осуществляющие вывод текста на экране с заданными координатами и устанавливающие цвет фона и цвет символов на экране.

Решение.

```
Program DemoProc; {демонстрация работы процедур}
Uses
   Crt; {подключение стандартного модуля Паскаля с подпрограммами работы с экраном}
Var  {объявление глобальных переменных}
   Position_X, Position_Y: integer; {координаты курсора на экране}

{объявление пользовательской процедуры FonColor}
procedure FonColor(Fon, Color: byte); {установка фона и цвета символов на
                                       экране}
begin {начало тела процедуры}
   TextBackGround(Fon);  {Fon   -  номер цвета фона}
   TextColor(Color)          {Color – номер цвета символов}
end; {конец процедуры FonColor}

{объявление пользовательской процедуры TextXY,}
{выполняющей вывод на экран строки текста с координатами  X-й столбец и  Y-я строка}
procedure TextXY (X, Y: integer; stroka: string);
begin {начало тела процедуры}
   GoToXY(X,Y);  {перемещение курсора в Y-ю строку, X-й столбец}
   Writeln(stroka) {вывод строки текста  в Y-ю строку, X-й столбец}
end; {конец процедуры TextXY}

begin {начало основного блока программы DemoProc}
   FonColor(Blue, Yellow); {символы желтого цвета на синем фоне}
   ClrScr; {очистка экрана и заполнение его фоновым цветом (синим)}
   TextXY(1, 1, 'Строка  желтого цвета на синем фоне');
   Position_X := 5;  Position_Y  := 3; {координаты нового места курсора на экране}
   FonColor(Blue, Red); {символы красного цвета на синем фоне}
   TextXY(Position_X, Position_Y, 'Ярко-красная строка на синем фоне');
   FonColor(Cyan, Green); {символы зеленого цвета на голубом фоне}
   TextXY(5, 10, 'Зеленая строка на голубом фоне');
end. {конец основного блока программы DemoProc}
```

7.4 Функции

Подпрограмма-функция отличается от процедуры тем, что возвращает в точку вызова ***единственное значение результата*** расчета[11].

Объявление подпрограммы-функции, определенной пользователем, похоже на объявление процедуры и также состоит из заголовка и тела функции. Дополнительно при описании функции необходимо явно задавать тип возвращаемого значения, а в теле функции обязательно должен присутствовать хотя бы один ***опе-***

[11] Другими словами, процедура описывает действие и в вызывающей программной единице используется *как оператор.* В отличие от этого функция находит *результат*, и в выражении предстает в качестве *операнда.*

ратор, присваивающий имени функции значение возвращаемого результата.

{заголовок пользовательской функции}
Function <имя> [(список формальных параметров)] : <тип результата>;
{объявления локальных объектов функции**}**
const …;
type … ;
var … ;
…
begin {начало тела функции}
    <операторы тела функции>
    <имя> := … {присвоение имени функции выходного результата}
end; {конец тела функции}

Вызов подпрограммы-функции принципиально отличается от вызова подпрограммы процедуры. Для обращения к функции достаточно использовать в выражении ее имя со списком фактических параметров (если он есть).

Пример 2. Написать программу, в которой используются пользовательские функции, которые возводят в целую и вещественную степень.

Решение.

```
Program DemoFunc; {демонстрация работы подпрограмм-функций}
Uses
   Crt; {подключение стандартного модуля Паскаля с подпрограммами работы с экраном}
Const
   f = 50.0; {тестовая константа}

{объявление пользовательской функции SqrI, возводящей Expr в целую степень N}
function SqrI(Expr: real; N: integer): real; {заголовок функции SqrI}
var {объявление локальных переменных}
   i: integer; {счетчик циклов}
   R: real;     {вспомогательная переменная}
begin {начало тела функции SqrI}
   if N = 0
    then  SqrII := 1 {если нулевая степень}
     else begin
             if  expr = 0
              then SqrII := 0 {если возводим в степень выражение, равное нулю}
               else  begin
                        R := 1;
                        for i := 1 to abs(N) do {цикл возведения в степень}
                          R := R * expr;
                          If  N < 0
                          then SqrII := 1 / R   {если степень отрицательная}
                           else  SqrII := R              {если степень положительная}
                       end
          end
end; {конец функции SqrI}

{объявление пользовательской функции SqrR, возводящей Expr в вещественную степень
N}
function SqrR(Expr: real; N: real): real;
begin {начало тела функции SqrR}
   if N = 0
      then SqrIR := 1 {если нулевая степень}
```

```
      else  begin
              if expr = 0
               then    SqrIR := 0 {если возводим в степень выражение, равное нулю}
               else    begin
                         if expr <  0
                           then begin {если возводим в степень выражение, меньше нуля}
                                  Writeln('Недопустимый аргумент (expr <  0 ) : ',
                                          expr:10:3);
                                  Writeln('Возводим в степень N модуль аргумента');
                                   Writeln(' Для продолжения нажмите <<Enter>> ');
                                end;
                         SqrIR := exp(N * ln(abs(expr)))
                       end
            end
end; {конец функции SqrR}

begin {начало основного блока программы DemoFunc }
      Writeln(' Возведение в целую степень' );
      Writeln('                (2 * pi * f)^3 = ', SqrI(2 * pi * f, 3) );
      Writeln(' Возведение в вещественную  степень');
      Writeln('                (2 * pi * f)^pi = ', SqrR(2 * pi * f, pi) )
end. {конец основного блока программы DemoFunc}
```

7.5 Параметры

Параметры могут иметь любой тип (кроме файлового), в том числе ***параметры-процедуры*** и ***параметры-функции***, позволяющие передавать подпрограмме имена внешних подпрограмм[12].

Если параметров одного типа несколько, их можно объединить в ***группу.*** Параметры в группе перечисляют через запятую. Вслед за тем, после двоеточия, указывается их общий тип. Отдельные группы отделяются друг от друга точкой с запятой

В рассмотренных выше примерах при вызове подпрограммы в процедуру передаются лишь копии фактических параметров. Такие аргументы называются ***параметрами-значениями***.

Если же перед списком формальных параметров в объявлении заголовка подпрограммы указывается ключевое слово ***var***, то по завершению ее работы соответствующим фактическим параметрам присваиваются рассчитанные подпрограммой значения формальных параметров. Действие ***var*** распространяется в пределах одной группы (до ближайшей точки с запятой). Такие «возвращаемые» параметры называются ***параметрами-переменными[13].***

8 Разработка пользовательских модулей

Модуль ***Borland Pascal*** представляет собой отдельно хранимую и независимо компилируемую программную единицу. В отличие от программы модуль не может быть запущен на выполнение самостоятельно, он может только участвовать в построении программ и других модулей. Это эффективное средство разбиения

---

[12] Разумно, например, написать обобщенную процедуру вычисления определенного интеграла, и передавать в нее в качестве параметра имя подпрограммы, вычисляющей подынтегральную функцию.

[13] Как правило, подпрограмма-функция возвращает лишь одно значение, поэтому формальные параметры функции обычно не имеют атрибута ***var*** (хотя это и не запрещено).

монолитных программ на более мелкие составляющие и использования опыта предыдущих разработок программного обеспечения.

В систему ***Borland Pascal 7.0*** входят стандартные модули-библиотеки ***System, Crt, Dos, Graph, Overlay, Printer*** с большим количеством встроенных подпрограмм, констант, переменных и типов данных, которые затем могут подключаться к разрабатываемым программным продуктам с помощью предложения ***uses***. Назначение функций и процедур, имеющихся в этих модулях, и примеры их применения приведены в [3 - 9] и в справочной системе (Help) ***Borland Pascal***.

Не бойтесь, что подключение модуля равнозначно подключению всех имеющихся в нем подпрограмм и описаний, и это значительно увеличит объем программы. Транслятор Паскаля отберет и подключит к Вашей программе только то, что действительно используется.

Тем не менее, если вы накопили ряд полезных подпрограмм, которые можно применять в других программах, то их целесообразно объединить в ***личные специализированные модули*** (***UNIT***) - совокупности программных ресурсов, предназначенных для использования другими программами. Под программными ресурсами понимаются любые элементы языка ***Borland Pascal***: константы, типы, переменные, подпрограммы.

Все программные элементы модуля можно разбить на две части:

- программные элементы, предназначенные для использования другими программами или модулями (такие элементы называют видимыми вне модуля);
- программные элементы, необходимые только для работы самого модуля (их называют невидимыми или скрытыми).

В общем случае модуль имеет следующую структуру:

unit <имя модуля>; {Заголовок модуля}
interface {***Интерфейсная часть*** - описание «видимых» программных элементов модуля}
{Объявляются константы, типы, переменные, процедуры и функции, которые являются общими (доступными пользователям модуля). Для процедур и функций здесь перечисляются только их заголовки. Тела процедур и функций находятся в части реализации}
implementation { ***Часть реализации*** - описание «скрытых» программных элементов модуля }
{Здесь находятся тела всех общих процедур и функций. Кроме того, здесь объявляются константы, типы, переменные, процедуры и функции, которые являются *частными* и не доступны пользователям модуля}
Begin {Часть инициализации - содержит операторы инициализации элементов модуля }
{Состоит из операторной части, которая будет выполнена при запуске программы, использующей данный модуль *unit*}
end.

Здесь ***<имя модуля>*** - имя, которое Вы будете использовать при указании ссылки на данный модуль в разделе ***Uses*** другой программы. В частном случае модуль может не содержать части инициализации.

Использование в модулях процедур и функций имеет свои особенности. Заголовок подпрограммы содержит все сведения, необходимые для ее вызова: имя, перечень и тип параметров, тип результата для функций, эта информация должна быть доступна для других программ и модулей. С другой стороны, текст подпрограммы, реализующий ее алгоритм, другими программами и модулями не может

быть использован. Поэтому заголовок процедур и функций помещают в интерфейсную часть модуля, а текст - в часть реализации.

Интерфейсная часть модуля содержит только видимые (доступные для других программ и модулей) заголовки процедур и функций (без служебного слова ***forward***). Полный текст процедуры или функции помещают в часть реализации, причем заголовок может не содержать список формальных параметров.

Исходный текст модуля должен быть откомпилирован с помощью директивы ***Make*** подменю ***Compile*** и записан на диск. Результатом компиляции модуля является файл с расширением либо ***.TPU*** (*Turbo Pascal Unit*), либо ***.TPP*** (*Borland Pascal Unit*)[14].

Если в модуле имеется раздел инициализации, то операторы из этого раздела будут выполнены перед началом выполнения программы, в которой используется этот модуль.

Пример 3. Решить задачу, рассмотренную в примере 2 с использованием собственного библиотечного модуля, в который собраны все подпрограммы, рассмотренные в предыдущих примерах.

Решение.

```
Unit MyUnit; {Имя личного библиотечного модуля}
  {Файл с исходным текстом этого модуля должен иметь имя MyUnit.pas}
  {После компиляции Turbo Pascal (turbo.exe) получит имя MyUnit.tpu}
  {После компиляции Borland  Pascal (bp.exe) получит имя MyUnit.tpp}

Interface {начало интерфейсной секции модуля MyUnit}

  {В интерфейсной секции модуля описываются те константы, типы, переменные,
  заголовки процедур и функций, которые станут глобальными (видимыми в других
  программах). Эти объекты будут доступны той программе или модулю, в которых есть
  предложение Uses с именем данного библиотечного модуля}

Uses
  Crt; {подключение стандартного модуля Паскаля с подпрограммами работы с экраном}

  {объявление глобальной  процедуры FonColor,  учреждающей  фон и цвет символов на
  экране}
procedure FonColor(Fon, Color: byte);

{объявление глобальной процедуры TextXY,}
{выполняющей вывод на экран строки текста с координатами  X-й столбец и  Y-я строка}
procedure TextXY (X, Y: integer; stroka: string);

{объявление глобальной функции Sqrl, возводящей Expr  в целую степень N}
function Sqrl(Expr: real; N: integer): real;

{объявление глобальной функции SqrR, возводящей Expr в вещественную степень N}
function SqrR(Expr: real; N: real): real;

Implementation {начало секции реализации модуля MyUnit}

{В секции реализации определяются конкретные действия в процедурах и функциях,
заголовки которых перечислены в интерфейсной секции. В ней также описываются все
локальные объекты (метки, константы, переменные, процедуры и функции), которые не
будут доступны подключающей данный библиотечный модуль программной единице}
```

[14] Структуры этих модулей разные! Их нельзя использовать вместе.

{Списки параметров в заголовках подпрограмм в секции реализации можно опустить!
Заголовки подпрограмм разумно не набирать, а скопировать из интерфейсной секции!}

{Реализация процедуры ***FonColor***}
Procedure FonColor; {установка фона и цвета символов на экране}
begin
  <операторы тела процедуры ***FonColor***>
end;

{Реализация процедуры ***TextXY***}
procedure TextXY; {вывод строки текста в Y-ю строку экрана, начиная с X-го столбца}
begin
  <операторы тела процедуры ***TextXY***>
end;

{Реализация глобальной функции *Sqrl*, возводящей *Expr* в целую степень *N*}
function Sqrl;
begin
  <операторы тела функции ***Sqrl***>
end;

{Реализация глобальной функции ***SqrR***, возводящей ***Expr*** в вещественную степень ***N***}
function SqrR;
begin
  <операторы тела функции ***SqrR***>
end;

begin[15] {Начало секции инициализации модуля ***MyUnit***}

{В секции ***инициализации*** операторы, которые выполняются первыми, т.е. перед операторами основного блока главной программы, к которой подключен данный модуль. Например, очистка экрана, начальные установки, приветствия, проверка лицензий и т.п. Чаще всего она пуста, и тогда ***begin*** начинающий ее может быть опущен}

end. {конец пользовательского библиотечного модуля ***MyUnit***}

Теперь основная программа может выглядеть так:

```
Program DemoFunc1; {демонстрация работы подпрограмм-функций}

Uses
   Crt,       {подключение модуля Паскаля с подпрограммами работы с экраном}
   MyUnit;    {подключение собственного библиотечного модуля};

Const
   f = 50.0; {тестовая константа}

begin {начало основного блока программы DemoFunc1}
   Writeln(' Возведение в целую степень' );
   Writeln('          (2 * pi * f)^3 = ', Sqrl(2 * pi * f, 3) );
   Writeln(' Возведение в вещественную  степень');
   Writeln('          (2 * pi * f)^pi = ', SqrR(2 * pi * f, pi) )

end. {конец основного блока программы DemoFunc1}
```

[15] Этот ***begin*** не обязателен.

9 Индивидуальные задания

9.1 Требования к программе

В каждом варианте индивидуального задания исходной является строка с текстом. Слова в тексте разделяются пробелами и знаками препинания. Тестовый текст формируется самостоятельно. Он должен позволять оценить работоспособность программы. Целесообразно на период отладки тестовый пример оформить в виде тестовой константы.

При составлении программы по заданию предусмотреть:

- простейший диалог типа «запрос-ответ» при вводе данных;
- многократный ввод данных при исполнении программы, т.е. возможность повторной обработки при иных исходных данных. Признак окончания ввода данных – ввод пустой строки;
- вывод результатов в удобном для пользователя виде;
- освоить методику поиска причин и исправления ошибок, а также трассировки программы «по шагам»[16] по тестовому примеру.
- Предусмотреть вывод на экран протокола тестирования, в котором выводятся результаты обработки данных на каждом этапе.

9.2 Варианты заданий

1. Исключить из введенной строки комментарии, расположенные между парами скобок {} или (* *). Сами скобки также исключить.

2. Определить количество слов в строке текста (допустимые разделители пробелы и знак ;). Выяснить – является ли выделенное слово числом. Если да, то каким числом (целым или вещественным).

3. Найти во введенной строке текста некоторую последовательность символов и заменить ее иной последовательностью символов (замен может быть несколько).

4. Выделить в строке текста, состоящей только из одних цифр слова (допустимые разделители пробелы, знаки # и ;). Рассматривая каждое слово как число, определить сумму четных и нечетных чисел.

5. Выделить в строке текста, состоящей только из одних цифр и разделителей слова (допустимые разделители: пробелы, запятые, знаки $ и ;). Найти количество слов. Рассматривая каждое слово как число, найти минимальное и максимальное.

6. Дана строка, в которой слова произвольной длины состоят из нулей и единиц (разделители между словами: пробелы, запятые, знаки : и ;). Преобразовать эти двоичные коды в десятичную систему.

7. Выделить в строке символов, состоящей только из одних цифр и разделителей слова (допустимые разделители пробелы, запятые, знак ;). Рассматривая каждое слово как число, найти количество слов, делящихся на 3 без остатка

8. Выделить в строке текста, состоящей только из цифр, букв A, B, C, D, E, F слова, начинающиеся с # (окончание слова пробелы, запятые и знак ;). Рассматривая каждое такое слово как шестнадцатеричное число, перевести его в двоичную систему счисления.

9. Строка текста содержит информацию с фамилией (ФИО), годом и местом рождения. Сформировать и вывести на экран строку вида:
***Фамилия*: <ФИО>, *возраст:* <количество лет>, *родился в* <место рождения>**

[16] Используйте функциональные клавиши F7 или F8.

10. Разбить текст, содержащийся в строке, на отдельные строки по 25 символов. Сформатировать каждую строку так, чтобы выровнять текст по правой границе.

11. Найти во введенной строке комментарии, расположенные между парами скобок {} или (* *) и отпечатать каждые из них с новой строки. Сами скобки не печатать. В строке произвольное количество комментариев.

12. Разработать программу шифровки и расшифровки строки символов. Алгоритм шифрования: буква А – заменяется на Я, буква Б на Ю и т.д. в обратном порядке алфавита. Предусмотреть, что символы в строке могут быть как на верхнем, так и на нижнем регистре.

13. Найти во введенной строке тексты, расположенные между ключевыми словами *begin* и *end* и отпечатать каждый из них с новой строки. В строке произвольное количество таких конструкций.

14. Разбить текст, содержащийся в строке, на отдельные строки по 30 символов. Сформатировать каждую строку так, чтобы «прижать» текст по правой границе путем вставки необходимого количества пробелов перед словами строки. Лишние пробелы между словами убрать.

15. Дана строка, в которой слова произвольной длины состоят из нулей и единиц (разделители между словами пробелы или запятые). Преобразовать эти двоичные слова в шестнадцатеричную систему.

16. Выделить в строке символов слова (допустимые разделители пробелы и запятые). Выяснить – является ли каждое из этих слов числом. Отсортировать найденные числа в порядке возрастания.

17. Определить, сколько раз встречается в строке каждый символ. Найти слово, содержащее максимальное количество самого распространенного символа.

18. Проверить, имеется ли в заданной строке символов баланс открывающих и закрывающих скобок. Учесть, что открывающая скобка всегда предшествует соответствующей закрывающей скобке. Предусмотреть также, что скобки могут быть вложенными.

19. В заданной строке найти последовательности символов, состоящие только из цифр. Преобразовать эти последовательности в числа и найти их среднее арифметическое.

20. Найти повторяющиеся слова (допустимые разделители: пробелы, запятые, знаки $ и ;) в двух строках. Определить, какое слово повторяется максимальное количество раз.

21. Удалить из строки текста все повторяющиеся слова и идущие подряд разделители (естественно, кроме первого). Допустимые разделители: пробелы, запятые, знаки ; ! и ?

22. Найти самое длинное общее слово в двух заданных строках текста. Допустимые разделители: пробелы, запятые, знаки ; ! и ?

23. Выделить в строке символов, состоящей только из цифр, букв A, B, C, D, E, F слова, начинающиеся с # (окончание слова пробелы, запятые, знаки : и ;). Рассматривая каждое такое слово как шестнадцатеричное число, перевести его в десятичную систему счисления.

24. В предложении все слова начинаются с различных букв. Вывести (если можно) слова предложения в таком порядке, чтобы последняя буква каждого слова совпадала с первой буквой следующего слова.

25. Даны два предложения. Найти самое короткое из слов первого предложения, которого нет во втором предложении.

Список рекомендуемой литературы

1. Кобрин Ю.П. Формирование текстов. Лабораторная работа по дисциплине “Информатика” для студентов специальностей 210201 (200800) и 201300. – Томск, ТУСУР, 2007. – 16 с.
2. Д. Ван Тассел. Стиль, разработка, эффективность, отладка и испытание программ: Пер. с англ. – М.: Мир, 1985. – 332 с.
3. Н. Вирт. Алгоритмы и структуры данных. : Пер. с англ. – М.: Мир, 1989. – 360 с.
4. Основы информатики. Учеб. Пособие / Аладьев В.З., Хунт Ю.Я., Шишаков М.Л. - М.: Информационно-издательский дом "Филинъ", 1998. - 496 с.
5. Марченко А.И., Марченко Л.А. Программирование в среде Borland Pascal 7.0. – К.: ЮНИОР, 1998. – 480 с.
6. Зуев Е.А. Программирование на языке Турбо Паскаль 6.0, 7.0. - М: Веста, Радио и связь, 1993. - 384 с.
7. Епанешников А., Епанешников В. Программирование в среде Turbo Pascal 7.0. - М.: "ДИАЛОГ-МИФИ", 1993. - 288 с.
8. Фаронов В.В. Turbo Pascal 7.0. Начальный курс. Учебное пособие.- М.: "НОЛИДЖ", 2001. - 576 с.
9. Фаронов В.В. Turbo Pascal 7.0. Практика программирования. Учебное пособие.- М.: "НОЛИДЖ", 1998. - 432 с.
10. ОС ТУСУР 6.1-97. Работы студенческие учебные и выпускные квалификационные. - Томск: ТУСУР, 1999.- 10 с.