Министерство образования РФ

ГОУ ВПО «Кемеровский государственный университет»

Кафедра экспериментальной физики

Структурное программирование В ИСР « Free Pascal »

(учебное пособие)

Кемерово 2010

ББК В3я73
УДК 621.396.218
П12

Печатается по решению редакционно-издательского совета ГОУ ВПО «Кемеровский государственный университет»

Рецензенты:
Кафедра вычислительной техники и информационных технологий КемИ(филиала) РГТЭУ,
к.ф.-м.н., доцент Сергеева И.А.

Павлова, Т.Ю.
П12 Структурное программирование в ИСР « Free Pascal »/ Т.Ю. Павлова, ГОУ ВПО «Кемеровский государственный университет». – Кемерово: 2010. -91с.

ISBN

Учебное пособие по курсу «Программирование» для студентов, обучающихся по направлению 010700.62 Физика и специальности 010701.65 Физика. В пособии изложены основы языка Паскаль: информация о видах и типах данных, основных операторах, необходимых для реализации простейших линейных, ветвящихся и циклических алгоритмов, изложены и проиллюстрированы принципы структурного программирования, описаны особенности реализации графики в ИСР «Free Pascal», принципы построения графиков функций, простейшие алгоритмы рисования движущихся объектов. Рассмотрены примеры решения задач численного интегрирования с использованием различных численных схем.

Пособие может быть использовано для обучения студентов физических и технических факультетов и учащихся классов физико-математического профиля.

Рис. 16, табл. 14, библиогр.: 6 назв.

ISBN **ББК В3я73**

Содержание

ВВЕДЕНИЕ

§ 1. Язык программирования Паскаль.

Язык Паскаль был задуман как язык для обучения систематическому программированию. Сейчас наряду с Бейсиком и Си он является наиболее распространённым. **Паскаль** был разработан Никлаусом Виртом, профессором Федерального технологического института в Цюрихе как язык обучения программированию. Вирт считал, что ремесло программирования необходимо поднять до уровня если не науки, то сложной инженерной деятельности. Кроме того, язык должен определенным образом формировать мышление программистов, помогать им почувствовать законы программирования, его красоту. Эта идея поначалу привела к излишней формализованности языка, достаточно жестким требованиям к программисту (что наблюдается до сих пор). Из-за этого Паскаль даже сравнивали с квадратным колесом. Но за прошедшее время этот язык претерпел довольно много изменений в лучшую сторону. Основная заслуга в этом принадлежит фирме Borland International, долгое время выпускавшей линейку программных продуктов Turbo Pascal. Сейчас в этот язык включены все современные идеи программирования: операторы для реализации идей структурного и модульного программирования, объектно - ориентированного программирования.

Далее ИСР была переработана в систему программирования, в которую включена работа с визуальными объектами, - получившая специальное название –Delphi, версия языка стала называться Object Pasccal.

Среда Turbo Pascal и СП Delphi являются платным программным обеспечением. С 1993г. стал развиваться свободно распространяемый проект с открытым кодом –Free Pascal, близкий к версии языка Object Pascal. Эта интегрированная среда написана под ОС Windows, Linux и несколько других распространенных операционных систем. Free

Pascal реализует все основные возможности языка, а также в нем существует возможность программировать распространенный сейчас графический интерфейс – окна для ОС Windows, например. Однако, возможности визуального программирования в нем нет, такая возможность реализована в новом свободно распространяемом проекте - СП Lazarus.

§ 2. Интерактивно интегрированная среда разработки (ИСР) Free Pascal (FPC –Free Pascal Compiler).

ИСР Free Pascal – совокупность программ, предназначенных для написания и отладки программ пользователей, сочетает возможность редактора текстов, компилятора и отладчика, имеет библиотеки подпрограмм (т. е. позволяет набрать пример на языке программирования Паскаль, обработать и подготовить для выполнения, выполнить и, по желанию, выполнить пошагово).

После входа в систему на экране дисплея появляется информационный кадр он состоит и полей: главное меню (серая полоса с перечислением заголовков меню); окно редактора (синее поле для набора программы); строка состояния (серая полоса внизу окна).

Главное меню предназначено для выбора режима работы. Выбор пункта главного меню осуществляется нажатием левой кнопки мыши или клавиши F10. При выборе пункта главного меню появляется выпадающее меню.

Меню File предназначено для выбора режима работы с файлами:

- формирование нового файла (New);
- открытие файла (Open), записанного на диске;
- сохранение содержимого активной рабочей области в файле с текущим именем (Save) или с другим именем (Save as);
- смена текущего каталога (change dir), с которого считываются файлы;
- завершение работы в интегрированной среде (Exit)

и другие режимы:

меню Edit предназначено для выбора режима создания и редактирования текста программ.

В меню Search собраны режимы, выполняющие поиск объектов программы по заданным параметрам.

меню Run предназначено для выбора режима выполнения программ.

меню Compile позволяет задать способ компиляции и компоновки составных частей программы.

меню Debug выполняется настройка отладчика, в частности выбор переменных, значения которых выводятся в окно наблюдения.

меню Tools позволяет обращаться к ассемблеру и отладчику.

меню Options производится установка режимов работы компонентов интегрированной системы.

меню Windows производится установка текущей активной рабочей области и способа отображения содержимого рабочей области (их может быть несколько, то есть одновременно можно открыть несколько программ).

меню Help можно обратиться к встроенной справочной системе.

Многие пункты главного меню дублируются клавишами:

F10.....вход в главное меню ИСР;
ESC.....закрытие диалогового окна или окна меню;
ALT+X.....выход из ИСР;
CTRL+C.....прерывание выполнения запущенной программы ;
F1.....высвечивает информацию справочной системы
CTRL+F1.....вывод информации о термине на который установлен курсор
F2.....сохранить программу активного окна
F3.....вызвать диалоговое окно Open a File
CTRL+F9.....запуск программы
F9.....компиляция
ALT+F5.....просмотр результатов выполнения программы
ALT+0.....работа со списком доступных окон системы.

Окно редактора позволяет набрать и редактировать программу.

В ИС имеется система команд для управления перемещением курсора; удаления, перемещения маркированных блоков – это можно сделать с помощью меню Edit, вызывая контекстное меню, нажатием правой клавиши мыши, или комбинаций клавиш:

SHIFT+”стрелка”.....выделить;
SHIFT+DEL.....удалить выделенное;
CTRL+Insert.....скопировать выделенное;
SHIFT+ Insert.....вставить;
ALT+←(BACKSPASE). . отменить последнюю команду редактирования.

Строка состояния показывает соответствие между функциональными клавишами и возможными выполняемыми действиями (F1 – Help, F2-Save и т.д.).

Подробнее о настройке ИСР можно прочитать в руководстве программиста на английском языке, на официальном сайте проекта.

§ 3. Этапы разработки программы в ИСР Free Pascal

Для оптимальной разработки полноценной программы, необходимо последовательно пройти несколько этапов. Нарушение порядка выполнения работ, пропуск этапов, приводит обычно к появлению в программе трудно устранимых логических ошибок, потерям времени, необходимости возвращаться к уже проделанным этапам работы. Поэтому абсолютно необходимо вести разработку программы следующим образом:

1 этап – разработка алгоритма.

На этом этапе необходимо определить имена и типы данных, которые будут участвовать в программе, порядок действий программы (алгоритм), разработать контрольный пример для программы, который поможет подтвердить правильность ее результатов.

2 этап– разработка программы.

Теория программирования предлагает несколько подходов к разработке программ. Небольшие программы лучше писать, руководствуясь принципами структурного программирования.

Структурное программирование - технология разработки программ, основанная на принципах:

- использование трёх базовых алгоритмических структур: следование, ветвление, цикл;
- программирование "сверху-вниз";
- использование модулей.

Способ программирования «сверху-вниз» заключается в том, что задача решается от общего к частному, крупные задачи разбиваются на более мелкие и т.д. пока очередные задачи смогут быть реализованы одним или несколькими простыми операторами языка программирования. При этом, если какие-то фрагменты программы повторяются или представляют собой отдельные, логически цельные задачи, они оформляются в виде отдельных модулей (подпрограмм).

Существует другой способ программирования – «снизу-вверх». В этом случае сначала решаются малые задачи, используются готовые подпрограммы, затем они объединяются в более крупное решение.

Однако, первые программы лучше разрабатывать «сверху-вниз». Причем, план программы можно рассматривать как первый этап проектирования.

3 этап– набор программы в окне редактора.

Например, ваша первая программа может выглядеть так:

```
Program Troll;
Var a,t,v,s:real;
Begin
    A:=1.5;
    Read(t);
    V:=a*t;
    S:=v*t/2;
    Writeln(v,s);
End.
```

Её необходимо набрать с помощью клавиатуры, а затем записать

на диск с помощью меню file/save as в виде имя.pas (вместо слова «имя» нужно набрать имя программы латинскими буквами). Если это не сделано, при запуске программы, ИСР потребует записать ее на диск (откроет соответствующее окно). Но лучше сохранять текст программы периодически, в процессе набора, нажимая клавишу F2.

4 этап – компиляция программы.

Компилятор - программа, которая осуществляет перевод исходной программы в объектную программу на языке машинных команд или ассемблера.

Результатом работы компилятора будет являться программа (так называемый «объектный код», имеющий обычно расширение .OBJ) или сообщение об ошибках в исходной программе. Объектный код не является готовой для исполнения программой, требуется еще работа компоновщика.

Работа компилятора состоит из двух этапов, которые обычно выполняются в несколько проходов:

- Анализ (распознавание текста исходной программы, заполнение таблиц идентификаторов). При этом производится лексический анализ (выделяются лексемы - слова исходного языка), синтаксический разбор (устанавливается, принадлежат ли данные цепочки символов к разрешенным конструкциям языка), семантический анализ (распознавание смысла записанных команд) и заполнение таблиц идентификаторов (специально организованные наборы данных, которые используют потом для генерации объектного кода).
- Синтез (на основе проведенного анализа и построенных таблиц идентификаторов порождается текст результирующей программы).

Также компилятор исправляет некоторые ошибки или делает сообщения об ошибках, которые выводятся в окно сообщений компилятора. По умолчанию это окно появляется под окном редактора вместе с сообщением о наличии ошибок в программе. Нажав два раза клавишу ввода (Enter), в окне редактора можно увидеть сообщение о первой ошибке, и система выставит курсор в то

место в программе, где эта ошибка возникает. Если окна компилятора не видно, можно нажать клавишу F6. Если окно компилятора и в этом случае не появится, вызовите список доступных окон командой Alt+0, выделите Compiler Messages, нажмите кнопки Show и OK для того, чтобы сделать окно компилятора видимым.

5 этап – компоновка с библиотеками подпрограмм и запуск программы на выполнение.

4 и 5 этапы производятся по команде run (ctrl+F9), их можно проводить и раздельно.

6 этап. Отладка программы.

Синтаксические ошибки выявляются на этапе компиляции. Однако, в программе может содержаться некоторое количество логических ошибок, когда компьютер выдает неправильные результаты, из-за того, что в программе записан неверный алгоритм. Способы выявления логических ошибок описаны в § 4.

7 этап– Анализ результатов, тестирование.

На этом этапе необходимо протестировать результаты работы программы. Для этого используется один или несколько контрольных примеров. Если программа имеет ветвления, нужно подобрать контрольные примеры для каждой ветви. Если контрольные примеры выполняются неправильно, придется возвратиться на предыдущие этапы.

§ 4. Отладка программ в ИСР.

Из практики известно, что программист затрачивает на отладку программы (т.е. на исправление ошибок) столько же времени, сколько он ее писал. При запуске программы, компилятор, включенный в ИСР, указывает синтаксические и часть логических ошибок. Кроме того, проверяя работу программы с контрольными примерами, нужно выявить сложные логические ошибки. Техническими приемами, уменьшающими время отладки, являются прежде всего хороший стиль программирования, проверка правильности логики программы, хорошо разработанный интерфейс, продуманные контрольные

примеры.

Необходимо помнить, что некоторые конструкции, допустимые в языке программирования, которым Вы пользуетесь, часто приводят к ошибкам. Поэтому старайтесь не использовать или используйте очень осторожно: оператор goto, глобальные переменные, автоматическое преобразование типов.

Задачи, предлагаемые студентам в рамках данного курса, являются учебными и предполагают написание небольших по объёму программ. Для их отладки можно использовать простые приемы, приведенные ниже. Большинство ошибок можно обнаружить с помощью таких простых приемов:

- Проверяйте наличие точки с запятой в конце операторов;
- Обращайте внимание на цвет служебных слов в редакторе. Если набранное служебное слово (begin, for, then…) не выделено белым цветом (настройка редактора «по умолчанию»), то, скорее всего, оно набрано неправильно.
- Проверяйте соответствие типа и размера используемых в программе переменных и массивов их описанию;
- Пишите операторы вывода промежуточных значений переменных, чтобы понять, где программа начала работать неверно;
- Пишите операторы вывода информации, локализующей место ошибки. Для проверки того, исполняется ли какая-то группа операторов, можно поставить среди них оператор writeln(‘*******’). Если при выполнении программы на экране нет звездочек, значит, эта группа операторов почему-то не выполняется.
- Если при отладке возникла новая ошибка, проверьте самое последнее изменение, которое Вы внесли в программу;
- Тщательно читайте и обдумывайте оператор, прежде чем внести в него исправление. Изучайте теорию программирования. Ошибки начинающих часто объясняются тем, что они не знают формы записи оператора и плохо представляют его работу.
- Объясните работу своей программы кому-нибудь еще. В качестве слушателя можно использовать и непрограммистов, так как смысл

этого приема в том, что вы излагаете логику работы программы «непосвященному» человеку, при таком последовательном изложении логические «провалы» легко обнаруживаются.

Для сложных программ можно использовать отладчик, включенный в ИСР. Однако для начинающего программиста использование отладчика, в силу сложности и многообразия его возможностей, приведет к неоправданно большим затратам времени.

Обратите внимание на контрольный пример. **Контрольный пример** – это специально разработанный тест для проверки программы. За недостатком времени преподаватель обычно требует только один простой пример с такими входными данными, результат введения которых можно предсказать в уме. Однако, если вы хотите стать хорошим программистом, то надо понимать, что тестирование программы – дело ответственное и сложное и, кроме обычных начальных данных, обязательно должно включать все предельные, или особенные, случаи. В программе с ветвлениями нужно проверить каждую ветвь, в программе с циклами – случаи, когда цикл не должен выполняться или должен заканчиваться досрочно.

§ 5. Требования

Информатика в целом и программирование, в частности, являются прикладными дисциплинами, поэтому наибольшее внимание уделяется выработке навыков самостоятельной работы в процессе проектирования и отладки студентом индивидуальных задач. Для освоения тем, вынесенных на практические занятия студент получает индивидуальную задачу, пишет и отлаживает программу, которая решает эту задачу и сдает ее преподавателю.

Для сдачи задачи студент должен:

- знать теорию (т.е. знать назначение операторов и функций использованных в программе);
- иметь отлаженную программу (программа должна работать, иметь хороший интерфейс, т.е. понятный для пользователя ввод-вывод);
- иметь ясное представление о программе (т.е. знать логику решения

задачи, назначение отдельных блоков программы);
- продемонстрировать правильность работы программы на контрольных примерах (см. далее);
- уметь внести изменения в программу по требованию преподавателя.

Глава 1. ПРОСТЫЕ АЛГОРИТМЫ

§ 1. Линейные алгоритмы.

1.1. Задачи, имеющие линейные алгоритмы решения.

Имеется большой класс задач, решение которых представляется последовательностью действий, имеющих четкий порядок, такой, что без завершения предыдущего действия, переходить к следующему не имеет смысла. Программа, реализующая такую задачу, также состоит из последовательности операторов (команд для компьютера), которые выполняются одна за другой.

Рассмотрим задачу: Троллейбус, трогаясь с места, двигается с ускорением $a=1.5 м/с^2$. Какую скорость он приобретёт через время t? Какое расстояние он при этом пройдёт?

Перед написанием программы, нужно разработать алгоритм решения данной задачи: уяснить какие величины, фигурируют в ней и разделить их на две группы - данные и неизвестные; определить последовательность нахождения неизвестных и вывести формулы (если нужно).

В рассматриваемой задаче известными величинами (судя по тексту), являются ускорение и время движения. Причем, значение ускорения дано, а ввод значение времени нужно запланировать во время работы программы. Неизвестные скорость и расстояние можно найти по формулам:

$$v = a \cdot t$$

$$S = \frac{a \cdot t^2}{2} = \frac{v \cdot t}{2}$$

Затем нужно **определить тип и вид величин,** которые будут использоваться в задаче. По виду простые величины (в программировании они все называются данными) делятся на константы и переменные. Тип данного определяет, в каком виде оно хранится в компьютере и какие значения может принимать.

Некоторые простые типы:

1. Целые типы (ShortInt, Integer, LongInt, Byte, Word).
2. Вещественные типы (Real, Single, Double, Extended, Comp).
3. Логический (Boolean).
4. Символьный (Char).
5. Строковые типы (String, String [n]).

Если переменная в процессе выполнения программы может принимать значение дробного числа, участвует в выражениях с умножением, делением, математическими функциями, то ее нужно определить как вещественную, если значение ее – буква, то это – символьная переменная, если значение – слово, то переменная – строкового типа. Различия между переменными типов одной группы более тонкие, и первое время можно не вдумываться в эти отличия, а использовать в качестве целого - тип Integer, в качестве вещественного - тип Real, в качестве строкового - тип String.

Для написания логически стройной и правильной программы необходимо составить план.

План (алгоритм) рассмотренной задачи:

1. Описание констант и переменных a, t, v, S.
2. Ввод t, a.
3. Вычисление v, S.
4. Вывод v, S.

Программа на языке Паскаль:

```
program Troll;                                    {начало }
const                                             {не изменяемые}
  a=1.5;                                          {данные}

var                                               {переменные}
  t, v, s : real;

begin                                  {начало  исполняемой  части программы}
  write(' Введите время t    ');       {"подсказка" к вводу}
  read(t);                             {ввод с клавиатуры}
  v:=a*t;
  s:=v*t/2;
  writeln(' За ',t,' сек. скорость троллейбуса v=',v,' м/с'); {вывод на экран}
  writeln(' троллейбус пройдёт расстояние S=',s,' м');
end.                                   {Конец программы}
```

В языке программирования Паскаль программа состоит из заголовка, раздела описаний и исполняемой части. Данная программа имеет заголовок «Troll», раздел описаний состоит из объявления констант и переменных в блоках const и var соответственно. Исполняемая часть начинается с оператора begin и заканчивается оператором end. В фигурных скобках пишутся комментарии, они написаны для лучшего понимания программы человеком, компьютер их игнорирует и не выполняет.

Программа состоит из команд языка Паскаль, называемых операторами. Практически каждый оператор заканчивается точкой с запятой. Компьютер выполняет операторы последовательно, один за другим. Сначала будет выполнен первый оператор write. При этом на экране появится «подсказка» пользователю, будут выведены слова: «Введите время t » (без кавычек). Затем начнется выполнение оператора ввода с клавиатуры read(t). Компьютер при этом будет ожидать ввода с клавиатуры числа (т.к. переменная t –вещественная). После его ввода и нажатия клавиши ввода (Enter), переменная t примет значение, введенное с клавиатуры. Затем последовательно будут вычислены значения переменных v и s. Операторы writeln в конце программы выведут на экран переменные t, v, s. В операторе writeln (или write) все, что необходимо вывести на экран перечисляется через запятую. Если набор символов записан в апострофах (строковая константа), то эти символы будут выведены на экран без изменений, за исключением апострофов. Если в списке

содержится имя переменной (например, t), то будет выведено ее значение.

Г) Контрольный пример.

$$при \quad a = 1.5 \quad t = 1$$

.

$$v = a \cdot t = 1.5 \quad S = \frac{a \cdot t^2}{2} = 0.75$$

На самом деле, на экране появится следующий результат:

```
За 1.000000000E+00 сек. скорость троллейбуса v= 1.5000000000E+00 м/с
троллейбус пройдёт расстояние S= 7.5000000000E-01 м
```

Является ли он правильным? Чтобы программа выводила результаты на экран в привычном для человека виде, необходимо сделать форматированный вывод.

1.2. Форматированный вывод

Информация, выводимая любой программой должна быть легко читаемой, понятной для пользователя. Для этого, в частности, нужно, чтобы действительные числа были выведены как число с точкой, в записи содержались пробелы, отделяющие числа от текста и др. Для этого в языках программирования используются форматы.

При использовании форматов для вывода значений в текстовые файлы или на экран около имени выводимого данного явно указывается размер поля для размещения его значения.

Форматы для данных различных типов:

1. **Для целых данных (integer и др.)** - формат задается одним числом, определяющим число позиций, отводимых на это число. Например:

```
Writeln(i:5);
```

2. **Для вещественных данных (real и др.)** - формат задается либо одним числом, определяющим число позиций, отводимых на это число в экспоненциальной форме; либо двумя числами, первое из которых обозначает общий размер поля, отведенного под это число, второе - число знаков после запятой. Например:

```
Writeln(p:10);      или      Writeln(p:6:2);
```

3.**Для строковых и символьных данных (string, char)** - формат задается одним числом, определяющим число позиций, отводимых на значение этих данных (т.е. на символ или набор символов). Например:

Writeln(c1:10);

Если в разобранной выше задаче переписать оператор, который выводит на экран значение времени и скорости, таким образом:

writeln(‘ За ’,t:6:2,’ сек. скорость троллейбуса v=’,v:5:1,’ м/с’);

то на экране появится

```
За 1.00 сек. скорость троллейбуса v= 1.5 м/с
```

<u>1.3. Запись констант с порядком.</u>

Очень большие числа в алгебре принято представлять в виде степеней числа 10. В языках программирования такая запись тоже практикуется и называется она ***нормализованная (экспоненциальная) форма.*** Например, число $-1.17 \cdot 10^8$ записывается как -1.17E+08 или -1.17E8.
при этом

-	1.17	E	+08
Знак мантиссы	Мантисса-число от 1 до 10	Символ 10	порядок со знаком

Другие примеры:

алгебраическая запись	запись на языке Паскаль
$9.1 \cdot 10^{-21}$	9.1E-21
$-0.17 \cdot 10^3$	-0.17E3 или –1.7E2
10^5	1E5

Задача. Определить энергию электрона в атоме водорода на орбите с номером n.

Алгоритм. Искомая энергия определяется по формуле $E = \frac{m_e e^4}{8\varepsilon_0^2 h^2} \cdot \frac{1}{n^2}$, где m_e и e–масса и заряд электрона, ε_0 и h- электрическая постоянная и постоянная Планка.

В программе опишем постоянные величины как константы, номер орбиты введем с клавиатуры. Энергия электрона в атоме водорода имеет очень малую величину, будем выводить ее значение в экспоненциальном формате (формат задаётся одним числом), иначе при выводе оно будет выглядеть нулем.

```
program energy;
const
   m=9.109e-31;
   e=1.602e-19;
   e0=8.854e-12;
   h=6.626e-34;
var
  En,n:real;
begin
  Write('Введите номер орбиты ');
  read(n);
  En:=-m*sqr(e*e/e0/h/n)/8;
  writeln('   E(',n:2:0,' )=', en:12,’  Дж’);
end.
```

При запуске программы и введении n=3, на экране появится:

```
Введите номер орбиты  3
     E( 3 )=-2.42106E-19  Дж
```

1.4. Несколько примеров простых линейных программ.

1.Написать программу, которая описывает как вещественные переменные и вводит с клавиатуры скорость и время равномерного движения тела, рассчитывает и выводит на экран значение расстояния, пройденного телом за это время, с 4 знаками после запятой.

Решение:

```
Program Pr11;
var v,t,s:real;
begin
read(v,t);
s:=v*t;
writeln('S=',s:7:4);
end.
```

2.Написать программу, которая описывает как целую переменную tC (температура по Цельсию) и вещественные переменные tR, tF (температуры по Реомюру и Фаренгейту). Программа вводит tC с клавиатуры, рассчитывает и выводит tR, tF с одним знаком после запятой по формулам $tR=0.8\cdot tC$; $tF=9\cdot tC/5+32$.

Решение

```
Program Pr12;
var tR,tF:real;
    tC:integer;
begin
read(tC);
tR:=0.8*tC;
tF:=9*tC/5+32;
writeln('tR=',tR:5:1,' tR=',tF:5:1);
end.
```

3.Написать программу, которая описывает как целые переменные и вводит с клавиатуры длины сторон прямоугольника, рассчитывает и выводит на экран значение периметра (переменная вещественного типа) с двумя знаками после запятой

Решение

```
Program Pr13;
var p:real;
```

4.Написать программу, в которой описываются вещественные переменные x,y,d вводит с клавиатуры координаты точки x,y, рассчитывает расстояние d от нее до точки (1,3) и выводит на экран значение d с двумя знаками до и после запятой .

Решение

```
Program Pr13;
var x,y,d:real;
```

a,b:integer;
begin
read(a,b);
p:=2*(a+b);
writeln('p',p:5:2);
end.
	begin
	read(x,y);
	d:=sqrt(sqr(x-1)+sqr(y-3));
	writeln('d=',d:5:2);
	end.

§ 2. Ветвящиеся алгоритмы.

Задачи с ветвлением

На практике часто встречаются задачи, в которых, в зависимости от некоторого условия, выполняются те или иные действия. Например, общее сопротивление участка цепи из двух резисторов находится по разным формулам, в зависимости от типа соединения. Значение функции можно рассчитать, если аргумент принадлежит области её определения. Рассмотрим такую задачу подробнее.

Задача: найти значение функции:

$$f(x) = \frac{1}{\sqrt{x^2 - 3x - 15}}$$

Решение.

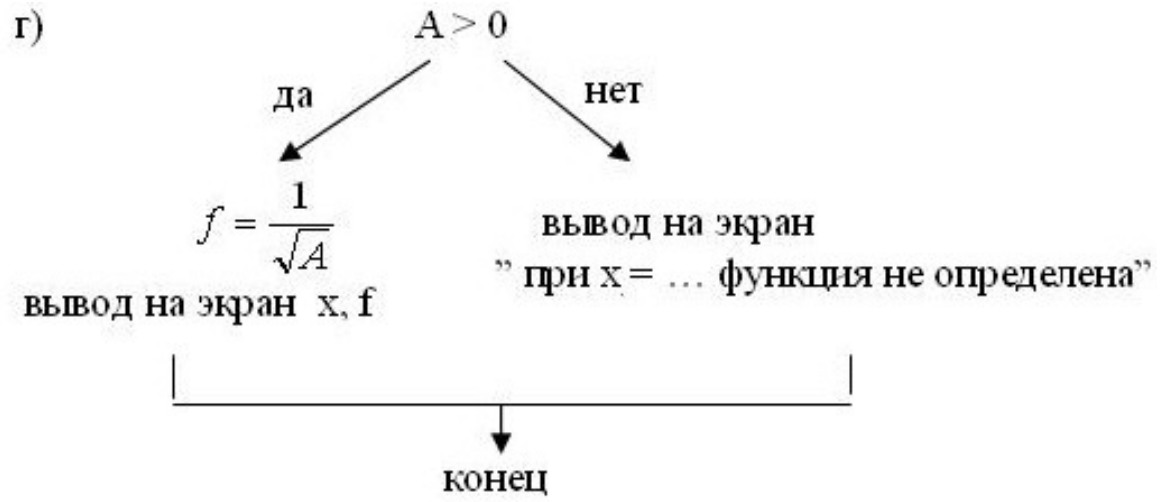
Найти значение функции можно, подставляя значение x в формулу для f(x), если она определена. Расчет значения функции возможен, если $A=x^2 - 3x - 15 > 0$, при $A \le 0$ то функция не определена.

План:

а) описание переменных x, f, A

б) ввод x

в) расчет $A= x^2 - 3x – 15$



Написать программу с выполнением разных действий, в зависимости от значения A (как предусмотрено в п. г) можно, если использовать условный оператор.

Условный и составной операторы

Для реализации ветвления можно использовать условный оператор:

```
if условие then оператор1
          else  оператор2;
```

где условие – это выражение логического типа (например, a>0, (d<0) and (c=5) и другие). При выполнении такого оператора, компьютер проверяет, выполняется ли условие. Если условие верно, то выполняется ветвь then, т.е. оператор1; если условие не выполняется – то ветвь else - оператор2.

Если в какой-либо ветви нужно выполнить не один, а несколько операторов, необходимо поместить эти операторы внутри **составного оператора** begin ... end, как в приведенной ниже программе. В этом случае последовательность операторов, объединенная в группу внутри begin ... end рассматривается как один оператор или блок. Составной оператор используется в тех случаях, когда по правилам языка допускается использовать только один оператор, а требуется выполнить несколько действий. Такой случай часто возникает при написании условных операторов, операторов выбора и цикла.

Описанную выше задачу реализует программа:

```
program vetv;
var     x, f, A: real;
begin
    write(‘Введите x ’);
    read(x);
    A:=sqr(x) – 3*x – 15;
    if A>0 then
                begin
                   f:=1/sqrt(A);
                  writeln(‘f(‘,x:4:1,’)=’,f:7:3);
                end
            else writeln (‘при x=’,x:4:1,’ функция не определена’);
 end.
```

Обратите внимание на то, что так как составной оператор стоит в ветви then, и после него условный оператор не заканчивается (есть ветвь else), то точка с запятой (знак конца оператора) не поставлена.

Если бы по логике задачи нужно было бы выполнить несколько действий в ветви else, то составной оператор нужно было бы использовать и там. Например:

```
    if A>0 then
                begin
                   f:=1/sqrt(A);
                  writeln(‘f(‘,x:4:1,’)=’,f:7:3);
                end
            else
               begin
                 writeln (‘при x=’,x:4:1,’ функция не определена’);
                 writeln (‘A=’,A:6:2);
               end;
```

Усеченный условный оператор и оператор exit.

Возможно использование усеченного условного оператора

if условие then оператор;

При этом, если условие не выполняется, то ветвь then игнорируется и управление передается следующему за if оператору.

Кроме того, в программе иногда удобно использовать оператор exit, чтобы досрочно прекратить ее выполнение.

Задача. Написать программу, которая рассчитывает работу 1 моля газа в течении одного из изопроцессов. Начальные температура, объем и давление известны. Программа запрашивает наименование процесса и недостающие параметры.

Решение

Обычно рассматриваются три вида изопроцессов:

- Изобарический (p-const). Работа газа $A = p(V_2 - V_1)$;
- Изотермический (T-const). Работа газа $A = RT \cdot \ln\left(V_2 / V_1\right)$;
- Изохорический (V-const). Работа газа $A = 0$

Приведенная ниже программа написана так, что человек сначала вводит с клавиатуры номер изопроцесса. Если процесс изохорический, то расчет производить не надо, программа сообщает результат (A=0) и заканчивает работу (по оператору exit). Если процесс не изохорический, то вводятся с клавиатуры начальный и конечный объемы газа, а затем, в зависимости от номера изопроцесса, запрашиваются дополнительные данные и производится расчет по соответствующей формуле. Использование логических операторов позволяет запрашивать только необходимые данные.

```
program vetv_1;
const R=8.31;
var
  i:integer;
  p,V1,V2,T, A: real;
begin
  write('Выберите изопроцесс:  1.p-const; 2.T-const; 3.V-const  ');
  read(i);
```

```
if i=3 then
        begin
         writeln(‘Изохорический процесс  A=0’);
         exit;
        end;
write(‘Введите начальный и конечный объем ');
read(V1,V2);
if i=1 then
        begin
         writeln(‘Изобарический процесс ’);
         write(‘Введите давление     ');
         read(p);
         A:=p*(V2-V1);
        end
       else
        begin
         writeln(‘Изотермический процесс ’);
         write(‘Введите температуру     ');
         read(T);
         A:=R*T*ln(V2/V1);
        end;
writeln (‘Работа газа A=’,A:6:1,’ Дж’);
end.
```

<u>Несколько примеров написания условного оператора.</u>

1. Написать оператор, который при условии x2≤2 , печатает на экране значение x и текст: «x вне диапазона». Если условие не выполняется, то рассчитывает значение переменной $f = \sqrt{10^{-5}(x^2 - 2)}$ и печатает её значение.	2. Написать оператор, который при условии 2<x<10 , рассчитывает значение переменной $f = 10^6 e^{2x} \sin x$ и печатает её. Решение if (2<x) and (x<10) then

Решение
if x*x<=2 then writeln(x,' x вне диапазона')
else
begin
f:=sqrt(1e-5*(x*x-2));
writeln(f);
end;
	begin
	f:=1e6*exp(2*x)*sin(x);
	writeln(f);
	end;

3.Написать оператор, который при условии x>2 или x=3y+1 , вводит с клавиатуры переменную m и печатает значение выражения 10-17(m+2m2), если условие не выполняется, то выводит на экран значение x.
Решение
if (x>2) or (x=3*y+1)
then
begin
read(m);
z:=1e-17*(m+2*m*m);
writeln(z);
end
else writeln(x);
	4.Написать оператор, который при условии x≠0 , вводит с клавиатуры переменную a, рассчитывает и печатает значение выражения f=a·sin(a/x) если условие не выполняется, то рассчитывает переменную r=(2x-1)(5-x) и печатает ее значение. (2 балла).
	Решение
	if x< >0 then
	begin
	read(a);
	f:=a*sin(a/x);
	writeln(f);
	end
	else
	begin
	r:=(2*x-1)*(5-x);
	writeln(r);
	end;

Оператор выбора.

Если для решения задачи нужно организовать более двух ветвей, удобно использовать оператор выбора CASE. Общий вид оператора:

```
case селектор of
        значение 1: оператор 1;
        значение 2: оператор 2;
        ……
        значение N: оператор N;
        else          оператор
  end;
```

где селектор это переменная или выражение целого, символьного (char) типов, допустимы и некоторые другие типы. Набор значений - это конкретные значения сектора, при которых будет выполнен соответственно нужный оператор.

Работа оператора: сначала вычисляется значение селектора, если он является выражением. Затем выполняется тот оператор, который соответствует значению, совпадающему со значением селектора. Если значение сектора не совпало ни с одним из указанных значений, то выполняется ветвь else.

Рассмотренную в п.2.1. задачу, можно написать с помощью этого оператора:

```
program vetv2;
var
  x, f, A: real;
   n: integer;
begin
 write('Введите x ');
 read(x);
 A:=sqr(x) – 3*x – 15;
n:=ord(A>0);              {  функция ord( )  равна 1, если выражение в}
                          { скобках верно, и  0 , если неверно}
case  n  of
    0:  writeln(' функция не определена');
    1:  begin
          f:=1/sqrt(A);
          writeln('x=',x:4:1,'    f=',f:6:3);
```

```
        end;
    else  writeln(‘ Такого случая нет’);
end;
end.
```

§ 3. Циклические алгоритмы.

<u>Цикл с параметром</u>

На практике часто встречаются задачи, в которых некоторое действие или группа действий повторяется с разными значениями какого-то параметра. Самыми распространенными задачами такого сорта являются задача распечатки таблицы функции и задача расчета суммы ряда. Для программирования таких задач используются операторы цикла.

Цикл с параметром (счетный цикл).

Общий вид оператора:

For параметр:=нач_значение To кон_значение Do оператор

Параметр цикла должен принадлежать к одному из следующих типов: целый, символьный, логический или перечислимый. Того же типа должны быть начальное и конечное значения параметра цикла, которые могут быть выражениями, переменными или константами. В цикле параметр увеличивается на минимальное значение (для целого типа – на единицу). В качестве оператора может использоваться простой или составной операторы.

Порядок выполнения оператора.

1. Вычисляются (если нужно) начальное и конечное значения параметра и фиксируются;
2. Если нач_значение<= кон_значения, то параметр= нач_значение и выполняется оператор;
3. Значение параметра цикла возрастает (для целого – на единицу);
4. Если значение параметра ≠ кон_значения, то выполняется оператор,

и компьютер снова переходит к выполнению п.3

5. Цикл выполняется последний раз, когда параметр= кон_значению, затем управление передается оператору после цикла.

Например, оператор (i - переменная целого типа)

```
for i:=1 to 100 do  writeln(i);
```

выведет на экран целые числа 1,2,3…100.

Оператор (n- переменная целого типа)

```
for n:=1 to 9 do  a:=10*n;
```

будет выполняться для значений параметра n=1,2,3,4,5,6,7,8,9. При этом переменная a будет принимать значения a=10,20, 30,40,50,60,70,80,90.

Для того, чтобы увидеть эти значения на экране, нужно записать оператор цикла в виде

```
for n:=1 to 9 do
    begin
        a:=10*n;
        writeln(n,'  ',a);
     end;
```

В этом случае, на каждом шаге цикла рассчитывается значение переменной a и выводятся на экран значения переменных n и a. Так как после служебного слова do в счетном цикле должен стоять только один оператор, использован составной оператор begin… end, внутрь которого и записываются все необходимые операторы. Таким образом, этот цикл начинается служебным словом for и заканчивается служебным словом end.

Другой пример цикла с параметром (m- переменная символьного типа):

```
for m:='a' to 'z' do   write(' ', m);
```

Этот цикл распечатывает через пробел на экране латинские буквы a, b,c,d,…z.

Важно понимать, что оператор цикла сам организует изменение параметра, и для целого параметра это изменение всегда равно 1. Попытка изменить значение параметра внутри цикла приводит к трудно распознаваемым логическим ошибкам, поэтому категорически

не рекомендуется! Если по условию задачи необходимо изменять значения с другим шагом h, опишите другую переменную (x) и изменяйте её на каждом шаге цикла, например, с помощью оператора x:=x+h; или так как показано ниже, в примере №4.

Несколько примеров написания оператора цикла с параметром.

<table>
<tr>
<td>1. Написать оператор, который выводит на экран в строку целые числа от 5 до 50
Решение
for i:=5 to 50 do write(i);</td>
<td>2. Написать оператор, который выводит на экран в строку четные числа от 4 до 50
Решение
for i:=2 to 25 do
begin
c:= 2*i;
write(c);
end;
или for i:=2 to 25 do write(2*i);</td>
</tr>
<tr>
<td>3. Написать оператор, который выводит на экран таблицу чисел от 1 до 20 и их квадратов.
Решение
for i:=1 to 20 do
begin
c:= i*i;
writeln(i:4, c:4);
end;</td>
<td>4. Написать оператор, который выводит на экран таблицу температур по Цельсию от -270 °C до 200°C через 10 °C и соответствующие температуры по Кельвину
Решение
for i:=-27 to 20 do
begin
tC:= 10*i;
tK:=tC+273;
writeln(tC:4, tK:4);
end;</td>
</tr>
</table>

<u>Цикл с предусловием.</u>

Если в задаче число повторений заранее не известно, цикл for использовать неудобно и можно воспользоваться оператором цикла while (общий вид):

while условие DO оператор;

Где условие – логическое выражение, оператор может быть простым или составным.

Порядок выполнения оператора.

Сначала проверяется условие: если оно верно, то выполняется оператор, затем опять проверяется условие и т.д., пока условие не перестанет выполняться. Если условие не верно, то оператор игнорируется и управление передается следующему за циклом оператору.

Пример цикла с предусловием:

```
while n <9 do   n:=n+1;
```

В этом операторе значение переменной n на каждом шаге цикла возрастает на 1. Однако, сколько шагов будет у цикла, зависит от значения переменной n до цикла. Если до цикла n=0, то выполнение оператора тела цикла n:=n+1 повторится 9 раз, при n=8 – 1 раз, а при n=9 –цикл выполняться не будет.

Необходимо помнить, что цикл while начинает выполняться, когда условие верно, и заканчивается, когда условие становится неверным. Следовательно, его нужно писать так, чтобы в условии были переменные или функции, которые меняются внутри цикла и он мог когда-нибудь закончиться. В противном случае, цикл будет выполняться бесконечно, т.е. программа «зациклится». Чтобы прервать выполнение такой неудачной программы, необходимо подать команду ctrl+c.

Приведенный выше оператор не имеет практической ценности, так как на каждом шаге только изменяет значение переменной n, которая играет роль номера шага цикла. Для организации расчета и распечатки нескольких значений, необходимо записать соответствующую группу операторов внутри составного оператора:

```
 n:= 0;
 while n <9 do
        begin
            n:=n+1;
            a:=10*n;
            writeln(n,’  ‘,a);
```

```
end;
```

Этот оператор, как и оператор for в п.3.1, выведет на экран попарно значения переменной n=1,2,3,4,5,6,7,8,9 и переменной a=10,20, 30,40,50,60,70,80,90.

Несколько примеров написания оператора цикла с предусловием.

<table>
<tr>
<td>1. Написать оператор, который выводит на экран в строку целые числа от 5 до 50
Решение
<pre>i:= 4;
while i <50 do
begin
i:=i+1;
write(i);
end;</pre></td>
<td>2. Написать оператор, который выводит на экран в строку четные числа от 4 до 50
Решение
<pre>i:= 2;
while i <50 do
begin
i:=i+2;
write(i);
end;</pre></td>
</tr>
<tr>
<td>3. Написать оператор, который выводит на экран таблицу чисел от 1 до 20 и их квадратов.
Решение
<pre>i:= 0;
while i <20 do
begin
i:=i+1;
writeln(i,’ ‘,i*i);
end;</pre></td>
<td>4. Написать оператор, который выводит на экран таблицу температур по Цельсию от -270 °C до 200°C через 10 °C и соответствующие температуры по Кельвину
Решение (tC, tK – целые):
<pre>tC:= -280;
while i <200 do
begin
tC:=tC+10;
tK:=tC+273;
writeln(tC:4,tK:6);
end;</pre></td>
</tr>
</table>

<u>Цикл с постусловием.</u>

Общий вид оператора:

```
Repeat
```

```
    Оператор 1;
    Оператор 2;
      …
    Оператор N;
Until условие;
```

Работа оператора: выполнение операторов 1- N повторяется, пока условие не станет верным.

Несомненным удобством цикла с постусловием является то, что внутри него можно записать несколько операторов без использования составного оператора.

Каждый из описанных выше циклов может быть использован для программирования задачи с циклическим алгоритмом.

Несколько примеров написания оператора цикла с предусловием.

1. Написать оператор, который выводит на экран в строку целые числа от 5 до 50
Решение
`i:= 4;`
`repeat`
`i:=i+1;`
`write(i);`
`until i >=50;`
	2. Написать оператор, который выводит на экран в строку четные числа от 4 до 50
	Решение
	`i:= 2;`
	`repeat`
	`i:=i+2;`
	`write(i);`
	`until i >=50;`

3. Написать оператор, который выводит на экран таблицу чисел от 1 до 20 и их квадратов.
Решение
`i:= 0;`
`repeat`
`i:=i+1;`
`writeln(i,' ',i*i);`
`until i >=20;`
	4.Распечатать таблицу температур по Цельсию от -270 °С до 200°С через 10 °С и соответствующие температуры по Кельвину (tC, tK – целые):
	`tC:= -280;`
	`repeat`
	`tC:=tC+10;`
	`tK:=tC+273;`
	`writeln(tC:4,tK:6);`
	`until tC >=200;`

Распечатка таблиц функций

Задача: Рассчитать таблицу значений функции $f(x) = x^3 e^{Sin\ x}$ для $x \in [1, 10]$.

Решение: В таких задачах предполагается, что необходимо рассчитать и распечатать таблицу значений аргумента и данной функции с некоторым шагом h. Обозначим предельные значения аргумента x0=1 и xk=10.

Равноотстоящие значения аргумента x всегда можно рассчитать по формуле x=x0 + h*i, где i – целое число i =0,1,2,…Если пользователь сам выбирает значение шага изменения аргумента h, то число значений x (равное числу строк будущей таблицы) можно рассчитать по формуле ik=целая_часть[(xk-x0)/h]+1. Можно, наоборот, выбрать число строк и рассчитать шаг h=(xk- x0)/(ik-1).
Если в нашей задаче выбрать шаг таблицы h = 1, получим ik=(xk-x0)/h+1=10 строк таблицы. Обозначим номер строки таблицы i, тогда значения аргумента в i-той строке можно рассчитать по формуле x= x0 + h*i, значение i = 0 ÷ ik (эта запись означает, что i изменяется от 0 до ik). Для каждого значения x можно рассчитать значение функции.
План программы:
а) описать переменные x, x0, h, xk, i, f
б) ввести x0, h, xk
в) распечатать заголовок таблицы
г) рассчитывать и печатать значения x, f(x) пока xi ≤ xk.
Последний пункт плана можно запрограммировать с помощью циклических структур, предусмотренных в языке Паскаль.

Программа с использованием цикла с параметром.

```
program chikl1;
var
        x0, xk, x, h, f: real;
        ik, i: integer;
 begin
```

```
      write('Введите x0,xk,h  ');
       read(x0, xk, h);
       ik:=trunc((xk-x0)/h);
       writeln('Таблица функции');
       writeln('_____________');
       writeln('|    x   |      f     |' );
       writeln('_____________');
        for i:=0 to ik do
          begin
            x:=x0 + i * h;
             f:= exp(3 * ln(x)) * exp(Sin(x));
             writeln(' | ',x:5:1,' | ',f:10:3,' | ');
          end;
         writeln('_____________');
end.
```

В данной программе учтено, что в языке Паскаль нет операции возведения в степень, поэтому $x^3 = \exp(3 \cdot \ln(x))$. Ввод трех значений в операторе read(x0, xk, h) производится через пробел. Функция trunc(..) используется для получения целого значения в правой части оператора присваивания, так как по правилам языка Паскаль целой переменной ik можно присвоить только целое значение.
Результат выполнения программы:

```
Введите x0,xk,h  1 10 1
Таблица функции
_____________
|   x   |      f    |
_____________
|   1.0 |     2.320 |
|   2.0 |    19.861 |
|   3.0 |    31.092 |
|   4.0 |    30.027 |
|   5.0 |    47.913 |
|   6.0 |    63.345 |
|   7.0 |   661.637 |
|   8.0 |  1377.028 |
|   9.0 |  1100.800 |
|  10.0 |   580.410 |
_____________
```

Программа с использованием цикла с предусловием.

```
program chikl2;
var
  x, x0, xk, h,f: real;
begin
  write('Введите x0, xk,h  ');
  read(x0, xk, h);
  writeln('Таблица функции');
  writeln('_____________________________');
  writeln('|   x   |      f       |' );
  writeln('_____________________________');
  x:= x0;
  while x <= xk do                    { Оператор после  do выполняется}
    begin                             { пока  условие x <= xk  верно}
          f:= exp(3  *  ln(x))  *  exp(Sin(x));          { Здесь оператором тела
цикла }
       x:= x + h;                              { является составной оператор}
       writeln(' | ',x:5:1,' | ',f:10:3,' | ');
    end;
  writeln('_____________________________');
```

end.
Обратите внимание, что в этой программе использован другой способ подсчета x.

Программа с использованием цикла с постусловием.

```
program chikl3;
var
  x, x0, xk, h,f : real;
begin
  write('Введите x0,xk,h   ');
  read(x0, xk, h);
  writeln('Таблица функции');
  writeln('_____________________________');
  writeln(' |       x       |            f            |' );
  writeln('_____________________________');
  x:= x0;
  repeat                    { Операторы между repeat и until выполняются}
                            { пока  условие x >xk  не станет верно}
     f:= exp(3 * ln(x)) * exp(Sin(x));
     writeln(' | ',x:5:1,' | ',f:10:3,' | ');
     x:= x + h;
  until x>xk;
  writeln('_____________________________');
end.
```

<u>Расчет сумм рядов</u>

Другой классической задачей с циклическим алгоритмом является задача нахождения суммы или произведения рядов.

Задача 1: Рассчитать сумму чисел 3+6+9+12+…+36

Решение.

Искомую сумму можно подсчитать с помощью оператора цикла,

если вывести формулу, которую можно записать внутри цикла, но на каждом шаге цикла, она должна давать значение очередного слагаемого. Для вывода такой формулы, пронумеруем слагаемые:

S=	3	+6	+9	+12	…+36
номер i	1	2	3	4	12

Легко видеть, что слагаемое под номером i можно вычислить по формуле $c_i = 3*i$. Тогда сумма

$$S = \sum_{i=1}^{12} c_i = \sum_{i=1}^{12} 3 \cdot i$$

Следовательно, сумму можно вычислить с помощью программы

```
program Summa1;
var
   s, i: integer;
begin
  s:=0;
  for i:=1 to 12 do s:=s+3*i;
  writeln('сумма s= ',s:6);
end.
```

Обратите внимание на распространенный в программировании прием, с помощью которого на каждом шаге рассчитывается сумма: s:=s+3*i;

Примеры расчета других простых сумм приведены в конце данного параграфа.

Задача 2: С точностью 10^{-6} рассчитать сумму ряда:

$$S = 1 - \frac{1}{3 \cdot 4} + \frac{1}{5 \cdot 6} - \frac{1}{7 \cdot 8} + \ldots$$

Решение.

Заметим, что в данном случае, ряд является знакопеременным (то есть, знак каждого следующего слагаемого отличается от предыдущего), Каждое, кроме первого, слагаемое можно представить как дробь со знаком, в знаменателе которой находится нечетное и последующее четное число. Пронумеруем однотипные слагаемые:

$$S = 1 - \frac{1}{3 \cdot 4} + \frac{1}{5 \cdot 6} - \frac{1}{7 \cdot 8} + \ldots$$

i = 1 2 3 …

i - номер слагаемого.

Тогда можно вывести формулу для i-го члена ряда:

$$c_i = \frac{(-1)^i}{(2i+1)(2i+2)},$$

при этом $$S = 1 + \sum_{i=1}^{\infty} c_i$$

здесь $c_1 = -1/(3 \cdot 4)$, $c_2 = 1/(5 \cdot 6)$ и т.д. Знак слагаемого получен с помощью возведения -1 в степень, пропорциональную номеру, поэтому меняется на каждом шаге.

Отметим, что если пронумеровать слагаемые, начиная с номера 2, то формула для слагаемого будет проще:

$$c_i = \frac{(-1)^i}{(2i-1) \cdot 2i} \quad i = 1,2,3,4\ldots$$

Ряд является бесконечным, компьютер же может суммировать лишь конечное число слагаемых. В этом случае можно либо заранее выбрать число слагаемых N, либо (как требуется в данной задаче) суммировать до достижения заданной точности. Обычно считается, что точность достигнута, если очередное слагаемое по модулю стало меньше заданной точности $| c_i | < \varepsilon = 10^{-6}$.

Задача нахождения суммы ряда с заданной точностью может быть запрограммирована с помощью цикла с пред- или постусловием. В приведенной программе ε обозначена как eps, переменная z содержит знак очередного слагаемого.

```
program Summa;
var eps,z,s,c,i:real;
begin
```

```
  i:=1;  s:=1;
  z:=-1;
  write(‘eps=‘);
  read(eps);
  repeat
    c:=z/(2*i+1)/(2*i+2);
    s:=s+c;
    i:=i+1;
    z:=-z;
  until abs(c)<eps;
  writeln('s=',s:8:3);
end.
```

Контрольный пример:

Выберем eps=0.04. Второе слагаемое по модулю становится меньше eps. При этом s=1-1/(3·4)+ 1/(5·6) =0,95.

Задача 3: Вычислим сумму n слагаемых ряда

$$S = \frac{x^3}{2!} + \frac{x^6}{3!} + \frac{x^9}{4!} + \ldots$$

В данной задаче вычислять очередное слагаемое через его номер не очень удобно, так как каждый раз нужно заново рассчитывать значение факториала. С другой стороны, легко видеть, что очередной член ряда можно получить из предыдущего по так называемой рекуррентной формуле: если взять первое слагаемое $C_1=x^3/2!$, то C2=C1*x3/3, C3=C2*x3/4, и так далее. В общем виде

$C_i=C_{i-1}*x^3/(i+1)$.

Сумму ряда с точностью eps можно посчитать с помощью прогрммы

```
Program pri6;
var s,x,eps,c,i:real;
begin
  i:=1;  s:=0;
  c:=1;
  write('eps=');
```

```
  read(eps);
  write('x=');
  read(x);
  repeat
    c:=c*x*sqr(x)/(i+1);
    s:=s+c;
    i:=i+1;
  until c<eps;
  writeln('s=',s:8:3);
end.
```

Проверить правильность вычисления отдельных слагаемых можно, поставив внутрь цикла оператор, который выведет на экран значения слагаемых и суммы на каждом шаге: writeln(i:5:0,c:8:4,s:10:5);

Еще несколько примеров написания операторов цикла, вычисляющих суммы, приведены в таблице.

1. Написать фрагмент программы, который вычисляет сумму чисел 4+6+8+…+22
Решение
s:=0;
for i:=2 to 11 do s:=s+2*i;
	2. Написать фрагмент программы,, который вычисляет сумму чисел 1+3+5+7+…+33
	Решение
	s:=0;
	for i:=1 to 17 do s:=s+2*i-1;

3. Написать фрагмент программы, который вычисляет сумму чисел 10+15+20+…+150
Решение
s:=0;
for i:=2 to 30 do s:=s+5*i;
	4. Написать фрагмент программы, который вычисляет произведение чисел от 3 до 100
	Решение
	p:=1;
	for i:=3 to100 do p:=p*i;

Глава 2. МАССИВЫ

§1. Определение массива

Массив представляет собой набор однотипных данных, имеющих общее имя. В алгебре массивам соответствуют векторы, матрицы, многомерные матрицы.

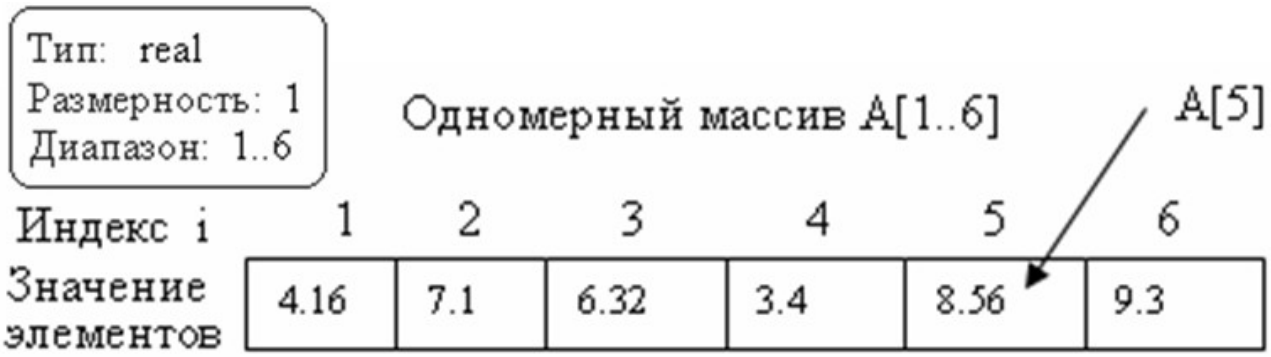
Характеристики массива:

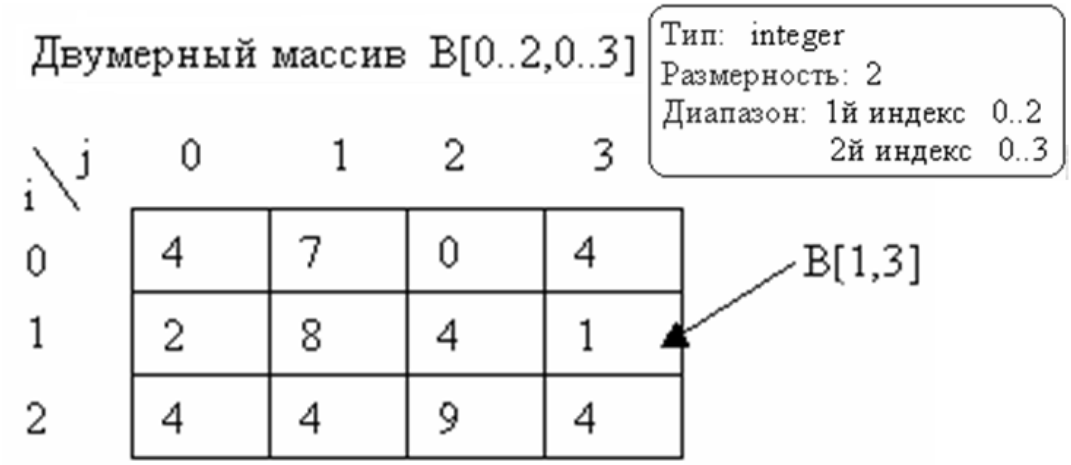
1.Тип - общий тип элементов массива;

2.Размерность (ранг) – количество индексов массива;

3.Диапазон – количество допустимых значений каждого индекса.

Индекс нумерует элементы массива. К данному элементу массива обращаются, указывая имя массива и значение индекса. Например, A[5] –пятый элемент массива A. Из рисунка видно, что значением этого элемента является число 8,56.





Действия над массивами

1. Описание (указание компилятору о выделении памяти для размещения массива.);
2. Ввод значений элементов массива в память компьютера;
3. Вывод значений элементов массива (на экран, в файл);
4. Операции преобразования (поиска) элементов массива.

Описывается массив целиком, остальные операции выполняются поэлементно.

§2. Описание массивов

Описание массива проводится для сообщения компилятору характеристик массива, в соответствии с которыми в памяти отводится место для его размещения. При этом числовой массив заполняется нулями, символьный и строковый – пробелами.

Описание массива проводится в блоке Var следующим образом:

Имя массива : array[is..ie, js..je,…] of тип элементов;

где array - означает “массив”; [is..ie, js..je,…] - начальное и конечное значение первого, второго и др. индексов массива (индекс может быть один), тип элементов - любой тип Паскаля.

Например, представленные выше массивы должны быть описаны так:

```
Var   a: array [1..6] of real;
      b: array [0..3,0..2] of integer;
```

Можно описать массив по-другому, пользуясь оператором описания типа (такое описание часто используется при работе с подпрограммами, описанными далее):

```
Type      Имя типа= array[is..ie, js..je,…] of  тип элементов;
Var       Имя массива :  имя типа;
```

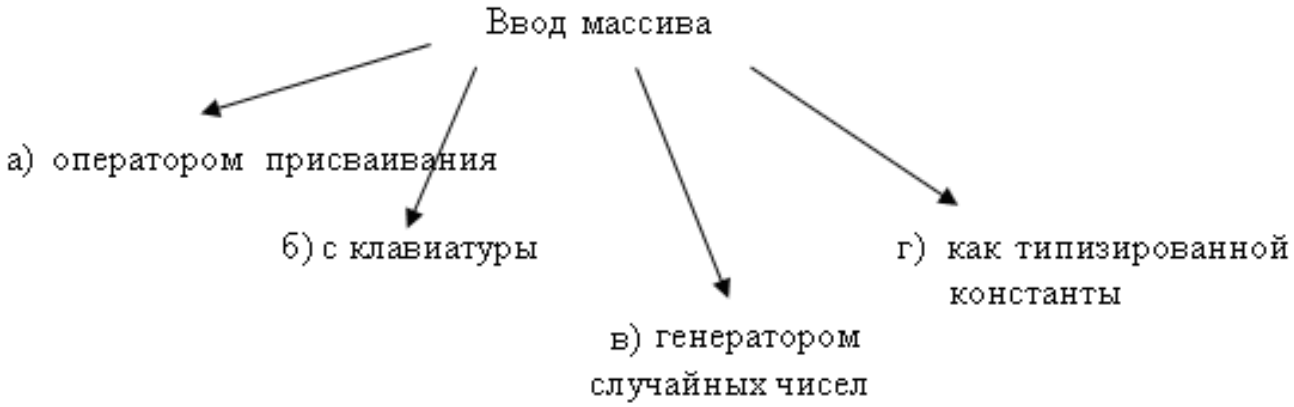
Например:

```
Type    vect = array [0...9] of real;
        matr = array [0..9, 0..4] of integer;
Var     a,b:vect;
```

m1,m2: matr;

§3. Ввод значений элементов массива

Ввод значений элементов массива можно осуществить несколькими способами.



а) Ввод элементов массива с помощью оператора присваивания.

Самый нерациональный способ ввода. Пример:

```
Program Pr1;
Var    a: array [1..4] of  word;
Begin
   a[1]:=2;
   a[2]:=21;
…
```

б) Ввод элементов массива с клавиатуры (используется, когда значения элементов массива вполне определенные, но разные при каждом запуске программы).

```
Program Pr1;
Var    a: array [1..4] of integer;
         i: integer;
Begin
  For i:=1 to 10 do   Read(a[i]);
```

…

Здесь цикл перебирает все значения индекса массива.

в) Ввод элементов массива с помощью генератора случайных чисел.

-используется, когда не важно, какие значения имеют элементы массива:

```
Var    a: array [1..4] of  real;
         i: integer;
Begin
  For i:=1 to 10 do
      a[i]:=random;
…
```

В этом случае, функция random генерирует псевдослучайные действительные числа в интервале от 0 до 1. Для генерации положительных и отрицательных действительных чисел в произвольном диапазоне, отрезок [0,1[растягивают и сдвигают, например, так a[i]:=10*random-5;

Для генерации значений элементов целочисленного массива используют функцию random(N):

```
Var    b: array [1..4] of integer;
         i: integer;
Begin
  For i:=1 to 10 do
      b[i]:=random(10);
…
```

В этом случае значениями элементов массива b будут целые числа от 0 до 9.

г) Ввод с помощью описания массива как типизированной константы.

-используется, когда значения элементов массива фиксированы.

```
Program Pr;
Const    a: array [1..5] of  integer =(5,4,9,1,2);
```

Begin

…

А далее в программе типизированные константы используются как переменные, в том числе могут изменяться.

§4. Вывод номеров и значений элементов массива

Вывод значений элементов массива производится в цикле. Например, номера i и значения mas[i] элементов массива mas можно вывести на экран в столбик таким образом:

```
for i:=1 to n do
    writeln(i:3, mas[i]:6);
```

Вывод значений элементов массива в строку, один за другим:

```
for i:=1 to n do
    write(mas[i]:6);
```

Формат вывода подобран так, чтобы при выводе элементы отделялись друг от друга пробелами. Если нужно выводить не все элементы, а только удовлетворяющие какому-то условию, используется условный оператор:

```
for i:=1 to n do
    if a[i]>5 then  writeln(i:3,a[i]:6);
```

§5. Примеры использования одномерного массива.

Простые примеры работы с одномерными массивами приведены в таблице.

1. Написать оператор, который вводит с клавиатуры одномерный массив B из 30 элементов, начиная с нулевого Решение	2. Написать оператор, который выводит на экран ненулевые элементы одномерного массива mas из 20 элементов. Решение for i:=1 to 20 do

for i:=0 to 29 do read(B[i]);	if mas[i]<>0 then writeln(mas[i]);

3. Написать оператор, который выводит на экран в столбик номера и значения элементов одномерного массива mas из 30 элементов.
Решение
for i:=1 to 30 do
writeln(i,mas[i]);
	4. Написать оператор, который описывает массив num из 10 элементов, индекс первого элемента равен 2.
	Решение
	var num: array[2..12] of real;

Ниже приведены программы, использующие массивы для хранения и обработки данных.

Задача 1. Найти максимальный элемент целочисленного одномерного массива, заполненного случайными числами.

Решение. Алгоритм решения такой задачи с помощью компьютера может быть таким:

1.Сформировать массив;

2.Присвоить вспомогательной переменной Emax значение первого элемента массива, а вспомогательную переменную Imax (в которой будет содержаться номер максимального элемента) приравнять к 1;

3.Сравнить все элементы массива с Emax, если найдется больший элемент, то значение Emax заменить значением этого элемента массива.

4.Вывести на экран полученные в результате перебора значения Imax и Emax.

```
Program Mass;
Const  M=100;
Var
   Mas:array[1..M] of integer;
   Imax, Emax, i: integer;
Begin
```

```
Randomize;          {устанавливает новый ряд случайных  чисел}
For i:=1 to M do
   Mas[i]:=random(500); {генерируются целые числа от 0 до 500}
Emax:=Mas[1]; Imax:=1;
For i:=1 to M do
 If  Emax< Mas[i] then   begin
                              Emax:=Mas[i];
                               Imax:=i;
                            end;
 Writeln('Максимальный элемент массива  Mas[',Imax:2,']=',
Emax:3);
End.
```

Задача 2. Определить, каким праздником является введенная календарная дата.

Разработаем программу, которая определяет, является ли число, введенное человеком, праздником и выводит на экран его наименование. Если введенная дата не является праздником, программа сообщает, что это – рабочий день. В данной задаче необходимо хранить даты и названия праздников.

Встроенного типа, соответствующего календарной дате в Паскале нет, поэтому для хранения дат можно поступить по-разному. Самый простой вариант, хранить праздничные даты как строковые значения, например, '1 января'. Также хранятся названия праздников.

Удобно организовать для хранения данных о праздниках 2 массива, в строковом массиве data будут храниться календарные даты, в строковом массиве prazdniki – наименования соответствующих праздников. Данные (дата, названия) об одном празднике будут соответствовать одному номеру (индексу) элементов массива. Индексы и значения элементов этих массивов приведены в таблице:

номер	data	prazdniki
1	1 января	Новый год
2	23 февраля	День Защитника Отечества
3	8 марта	Женский день

4	9 мая	День Победы
5	12 июня	День России
6	5 октября	День учителя
7	4 ноября	День народного единства

Массивы описаны в программе как типизированные константы, что позволяет сразу, в блоке const, задать значения их элементов. После ввода даты, программы перебирает все элементы массива data, пытаясь найти элемент, равный введенному значению. Если такой элемент находится, на экран выводится название праздника и программа заканчивает свою работу (оператор exit;). Если перебрав весь массив, программа не нашла такой даты, на экран выводится сообщение о том, что эта дата -рабочий день.

```
program pr11;
const    n=7;

    data: array [1..n] of string=('1 января ' ,'23 февраля','8 марта',' 9 мая','12 июня','5 октября','4 ноября');

    prazdniki: array [1..n] of string=('Новый год' , 'День Защитника Отечества ','Женский день',' День Победы', 'День России', 'День Учителя', 'День народного единства');

var
  dat:string;
  i:integer;
begin
  write('Vvod data  ');
  read(dat);
  for i:=1 to n do
    if  dat=data[i] then begin
                    writeln(' Это   ',prazdniki[i]);
                    exit;
                   end;
```

```
   writeln ('Это - рабочий день');

end.
```

§6. Особенности использования двумерных массивов.

При использовании двумерных массивов для работы с его элементами необходимо одновременно перебирать два индекса. Это делается с помощью вложенных циклов. Обычно внешний цикл перебирает номера строк, а внутренний – номера столбцов.

Например, ввести случайным образом элементы двумерного массива и распечатать их в виде двумерной таблицы можно так:

```
program pr;
var
   a:array[1..10,1..5] of integer;
   i,j:word;
begin
   for i:=1 to 10 do                    } заполнение двумерного
    for j:=1 to 5 do                    } массива случайными числами
     a[i,j]:=random(10);
   for i:=1 to 10 do
    begin
     for j:=1 to 5 do
       write(a[i,j]:2);                 } печать элементов массива
     writeln;
    end;
end.
```

В этом примере при печати Для каждого значения индекса i …для печати элементов одной строки массива (матрицы) используется оператор write(a[i,j]:2). Для печати следующей строки курсор переводится на строку вниз оператором writeln.

Еще пример. Программа находит в двумерном массиве минимальный элемент и перемещает его на место a[1,1].

```
program pr11;
const n=10;
var
  i,j,min,jm,im,b: byte;
  a: array [1..n,1..n] of byte;
begin
  randomize;                    { заполняем массив случайными числами }
  for i:=1 to n do
   for j:=1 to n do
      a[i,j]:=random(10);
  min:= a[1,1];
  im:=1;  jm:=1;
  for i:=1 to n do                          { ищем минимальный элемент }
    begin
      for j:=1 to n do
        begin

         if min>a[i,j] then  begin
                                 min:=a[i,j];
                                 im:=i;
                                 jm:=j;
                               end;
      end;  end;

if (im<>1) or (jm<>1)then
             begin
               a[im,jm]:= a[1,1];       {  переносим минимальный элемент  }
               a[1,1]:=min;                     {     на место a[1,1]  }
             end;
   for i:=1 to n do                          {  печать массива  }
     begin
```

```
    for j:=1 to n do
      write(a[i,j]:2);
    writeln;
  end;

end.
```

Глава 3. ПРОЦЕДУРЫ И ФУНКЦИИ

§1. Назначение и виды подпрограмм

Сложные задачи в программировании принято разбивать на части (подзадачи), которые могут быть написаны и отлажены (проверены) отдельно. Разбиение задачи на подзадачи называется декомпозицией. Выделенные подзадачи оформляются как подпрограммы и называются вспомогательными алгоритмами.

Особенно большой выигрыш от декомпозиции появляется, когда один и тот же вспомогательный алгоритм используется в программе несколько раз с разными параметрами. При выделении вспомогательного алгоритма, нужно стремиться сделать его как можно более универсальным, т.е. подходящим для разных задач.

Например, в задаче требуется вычислить расстояние между двумя точками с координатами (x_1,y_1), (x_2,y_2) и расстояние между каждой из этих точек и началом координат. Формулы для вычисления расстояния между точками $r_{12} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$ и расстояния от точки до начала координат $r = \sqrt{x^2 + y^2}$ очень похожи. Более того, можно заметить, что при x_1=x, x_2=0, y_1=y, y_2=0, можно использовать первую из них. Таким образом, написав подпрограмму, реализующую формулу для r_{12}, можно вызвать её для вычисления расстояний до начала координат.

В языке программирования Паскаль имеется два типа подпрограмм:

- **Функция** – подпрограмма, в которой производятся некоторые действия (например, вычисления, сравнение), в результате которых имя функции получает значение,

передаваемое в вызывающую программу.

- **Процедура** - подпрограмма, в которой производятся некоторые действия (например, печать, вычисления), при этом имя функции не получает никакого значения, полученные результаты либо не передаются в вызывающую программу, либо передаются специальным образом.

Для того, чтобы использовать подпрограмму (процедуру или функцию), ее описывают в начале программы. Описание подпрограммы состоит в записи ее заголовка с указанием списка параметров и тела подпрограммы (т.е. операторов, которые входят в подпрограмму). При этом компьютер ничего не выполняет, а принимает к сведению описание подпрограммы. Затем подпрограмма может быть вызвана несколько раз с разными параметрами. При каждом вызове производится выполнение операторов, составляющих тело подпрограммы.

Пример1.

Написать программу, в которой вводятся числа a,b,c и вычислить суммы (a+b+c), (a+b) в подпрограмме.

Решение:

```
Program pp1;
 function summa(a,b,c:real):real;
 begin
   summa:=a+b+c;
 end;
 var  x,y,z,s1,s2:real;
 begin
   write('vvod x,y,z: ');
   read(x,y,z);
   s1:=summa(x,y,z);
   s2:=summa(x,y,0);
   writeln(' summirovanie: x+y+z= ',s1:6:3,' x+y= ',s2:6:3);
 end.
```

Описание подпрограммы

Вызов подпрограммы

Еще один вызов подпрограммы

Вывод результата

§2. Подпрограммы - функции

Подпрограмма – функция наиболее проста для использования, так как обязательно возвращает в вызывающую программу значение, присвоенное её имени. Ее использование аналогично работе со встроенными функциями языка Паскаль, такими как sin(x), cos(x) и другие. Однако, она должна быть описана в начале программы.

Общий вид заголовка функции:

Function Имя (параметр1:тип1, параметр2:тип2,...): тип функции;

В скобках указывается список, так называемых, формальных параметров. После скобок указывается тип, который приписывается имени функции. После заголовка функции в блоке var могут быть описаны локальные переменные, которые можно использовать внутри тела функции, в других частях программы эти переменные недоступны. После описания переменных или сразу после заголовка функции оператор begin начинает тело подпрограммы-функции. В теле функции формальные параметры можно использовать для вычислений, считая, что их значения переданы из вызывающей программы. Имя функции должно обязательно получить значение в операторе присваивания.

Как уже указывалось, описание подпрограммы не приводит к ее выполнению. Подпрограмма выполняется, когда в тексте программы встречается ее вызов. Вызов подпрограммы – функции в основной программе или в другой подпрограмме можно осуществлять в операторе присваивания или в операторе вывода. При вызове функции указывается ее имя и в скобках - список фактических параметров (без указания их типов). Число и типы фактических параметров должны полностью соответствовать числу и типам формальных параметров в заголовке функции при описании. Значения фактических параметров присваиваются формальным параметрам (в порядке следования) и выполняются операторы тела подпрограммы. Значение, полученное именем функции, передается в вызывающую программу.

Тонкости взаимодействия формальных и фактических параметров обсуждаются в следующем параграфе. Разные способы вызова

подпрограммы – функции приведены в примере 2.

Пример2.

Найдем значения выражений c^d, 2^3+3^2, $(c+\sin(c))^3$.

Оформим возведение в степень как вспомогательный алгоритм. Воспользуемся свойством:

$\ln(x^a)=a*\ln x$, тогда $(x^a)=\exp(\ln(x^a))=\exp(a*\ln(x))$.

Понятно, что при $x\le 0$, функция ln(x) не определена, но нуль в любой степени равен нулю. Программа на языке Паскаль:

```
Program Prim3;
Function  Pur(x,a,:real):real;            { Описание подпрограммы - п/п }
  Begin
   Pur:=0;
     if x<=0  then begin
                     if x<0  then write ('функция не определена');
                     exit;
                   end
              else Pur:=exp(a*ln(x));
 end;                              {конец описания подпрограммы}
Var  c,d,y1,y2,y3:real;
 begin
   write('Введите основание и степень  ');
   read(c,d);
   writeln(c:5:2,'^',d:3:1,'=', Pur(c,d):5:2); { вызов п/п и вывод c^d }
   y2:=Pur(2,3)+Pur(3,2);              {  вызов п/п для получения 2^3+3^2 }
   y3:=Pur(c+sin(c),3);            {  вызов п/п для получения (c+sin(c))^3 }
   writeln('y2=',y2:5:2,'   y3=',y3:5:2);
end.
```

При первом вызове функции в операторе writeln, значение переменной c передается параметру x подпрограммы, значение d – параметру a. В операторе y3:=Pur(c+sin(c),3); значение выражения c+sin(c) передается параметру x, значение 3 – параметру a.

Еще несколько примеров простых подпрограмм –функций:

1.Написать программу с подпрограммой. В основной	2.Написать программу с подпрограммой. Основная

программе вводятся координаты вершин треугольника и находится его периметр. Длины сторон находятся с помощью подпрограммы.

Решение

```
Program Prim61;
function
dlina(xa,ya,xb,yb:real):real;
begin
  dlina:=sqrt(sqr(xa-xb)
+sqr(ya-yb));
end;
var
x1,y1,x2,y2,x3,y3,a,b,c,p:real;
begin
read(x1,y1,x2,y2,x3,y3);
a:=dlina(x1,y1,x2,y2);
b:=dlina(x2,y2,x3,y3);
c:=dlina(x1,y1,x3,y3);
p:=a+b+c;
writeln('p=',p);
end.
```

программа вводит с клавиатуры начальные скорости, ускорения и время движения двух тел и выводит значение скоростей, которые они приобрели за это время. Значение скорости вычисляется в подпрограмме.

Решение

```
Program Prim62;
function Skorost(v0,a,t:real):real;
begin
  Skorost:=v0+a*t;
end;
var
v01,a1,t1,v1,v02,a2,t2,v2:real;
begin
read(v01,a1,t1);
read(v02,a2,t2);
v1:=Skorost(v01,a1,t1);
v2:=Skorost(v02,a2,t2);
writeln('v1=',v1,'  v2=',v2);
end.
```

3.Написать программу с подпрограммой. Основная программа вводит с клавиатуры координаты двух точек, вычисляет и выводит на экран расстояние между точками и от каждой до начала координат. Расстояния между точками находятся с помощью подпрограммы.

Решение

4.Написать программу с подпрограммой. В основной программе ввести с клавиатуры значения переменных a и b. С помощью обращения к подпрограмме найти значения выражений a+b, 2a-b и вывести на экран. Подпрограмма рассчитывает сумму чисел.

Решение

```
Program Prim61;
function
dlina(xa,ya,xb,yb:real):real;
begin
  dlina:=sqrt(sqr(xa-xb)+sqr(ya-
yb));
end;
var x1,y1,x2,y2,a,b,c:real;
begin
read(x1,y1,x2,y2);
a:=dlina(x1,y1,0,0);
b:=dlina(x2,y2,0,0);
c:=dlina(x1,y1,x2,y2);
writeln('до начала координат
r1=',a,'  r2=',b);
writeln('между точками r=',c);
end.
```

```
Program Prim61;
function sum(c,d:real):real;
begin
  sum:=c+d;
end;
var a,b,s1,s2:real;
begin
read(a,b);
s1:=sum(a,b);
s1:=sum(2*a,-b);
writeln('a+b=',s1,'  2a-b=',s2);
end.
```

§3. Параметры подпрограмм

Переменные, указанные в списке после имени процедуры или функции называются параметрами этой подпрограммы. Параметры служат для обмена данными между основной программой и подпрограммами.

Параметры, указанные в списке после имени подпрограммы при ее описании называются **формальными**.

Параметры, указанные в списке после имени подпрограммы при ее вызове, называются **фактическими**.

Количество и тип фактических параметров должны точно совпадать с количеством и типом формальных параметров. Причем, они соответствуют друг другу в порядке указания.

Из примера видно также, что тип формального параметра задается прямо в списке при описании подпрограммы, а тип фактического параметра описывается стандартным образом в блоке var

вызывающей программы.

Параметры служат для обмена данными между вызывающим блоком и подпрограммой. Передаваемые в подпрограмму данные записываются в отдельную область памяти, называемую **стеком**. В языке Паскаль существуют различные способы передачи данных:

- С помощью параметров-значений;
- С помощью параметров-переменных;
- С помощью параметров-констант.

Параметры-значения_переписываются в стек целиком. Такой способ позволяет использовать в качестве фактического параметра не только переменную, но и константу.

Другой способ состоит в том, что в стек записывается не значение переменной, а адрес места памяти, которое под нее отведено. Такие параметры называются **параметрами-переменными**. В отличие от параметров-значений, перед описанием формальных параметров-переменных ставится служебное слово var. В этом случае можно существенно сэкономить память стека. С другой стороны, такие особенности хранения параметров приводят к тому, что параметры-значения подпрограмма не изменяет, а параметры-переменные может изменить и передать их новые значения в вызывающий блок. С помощью таких параметров в вызывающий блок можно вернуть не одно вычисленное значение, а несколько.

Параметры-константы передаются в подпрограмму как и параметры –переменные, с помощью адреса, однако, на их изменение в подпрограмме наложен запрет. Такая передача данных позволяет экономить память стека. При описании формального параметра константы перед его именем ставится служебное слово const.

Все примеры программ с подпрограммами, приведенные выше, использовали параметры-значения. В следующем параграфе описан пример, в котором используются параметры-переменные.

§4. Подпрограммы-процедуры.

Подпрограммы – процедуры также предназначены для

оформления и вызова вспомогательных алгоритмов, но, как правило, используются, если подпрограмма ничего не возвращает в вызывающий блок или возвращает несколько значений.

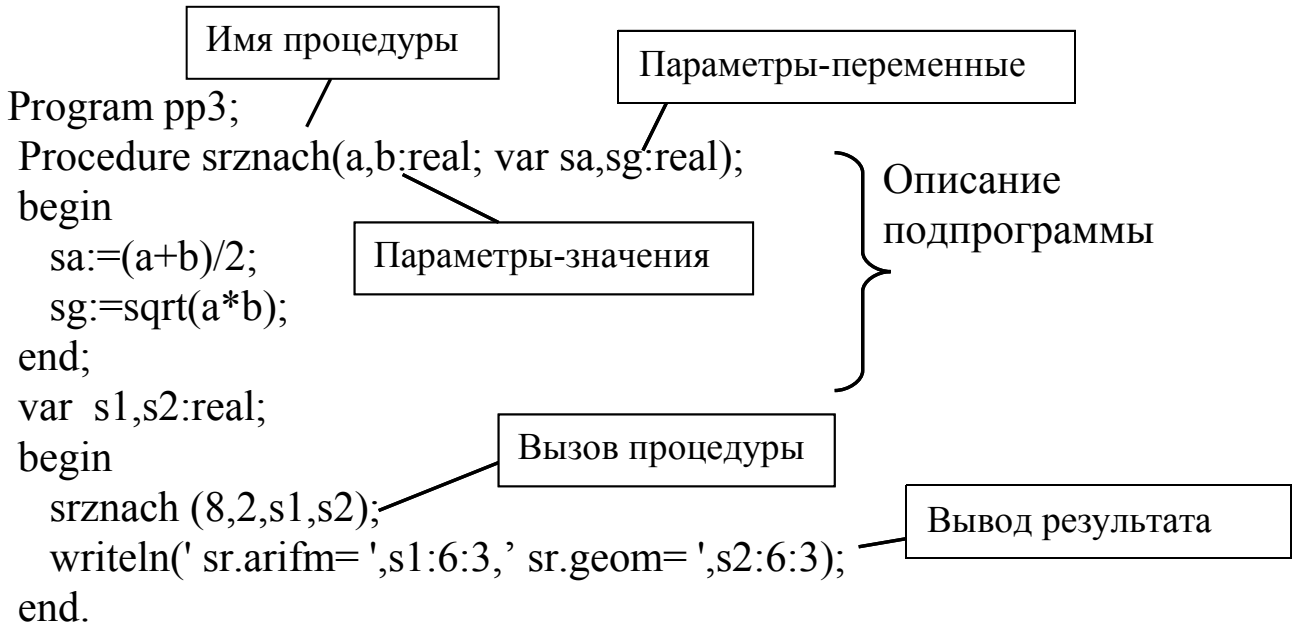
Описание процедуры отличается от описания функции тем, что имя процедуры не имеет типа и в теле подпрограммы оно не используется. Общий вид заголовка процедуры:

Procedure Имя (параметр1:тип1, параметр2:тип2,...);

Параметры процедуры используются для передачи в нее данных (при этом используются параметры-значения) и возвращения результатов обратно, в вызывающую программу (используются параметры-переменные).

Вызов подпрограммы-процедуры производится отдельно, оператором вызова процедуры с указанием её имени и значений параметров.

Пример 3. Написать программу с подпрограммой, которая рассчитывает среднее арифметическое и среднее геометрическое двух чисел.



В этой программе с помощью вызова подпрограммы-процедуры рассчитываются средние арифметическое и среднее геометрическое чисел 8 и 2. Для передачи результатов в основную программу

используются параметры-переменные sa и sg. Если их описать как параметры-значения (убрать var в описании), оператор writeln выведет на экран нули.

§5. Глобальные и локальные переменные.

Все переменные, объявленные в разделе описаний после оператора Program доступны для использования в основной программе, во всех подпрограммах и называются **глобальными**. Переменные, объявленные в разделе описаний подпрограмм, доступны только внутри этих подпрограмм и называются **локальными**. Локальными переменными являются также переменные, описанные вначале основной программы. Так, в первой программе, приведенной в следующем параграфе, перменные j,m – локальные переменные подпрограммы Proc, i,i1,v1 - локальные переменные основной программы, а константа n и тип matr являются глобальными и используются и в подпрограмме, и в основной программе.

Глобальные переменные удобны (не нужно думать о передаче значений в подпрограмму), но считаются опасными, так как случайное их изменение в подпрограмме приводит к изменению работы всей программы и такие ошибки трудно обнаружить. Кроме того, использование глобальных переменных в подпрограммах приводит к большим трудностям при независимой отладке подпрограмм и переносе их в другие программы. Поэтому **использование глобальных переменных в студенческих программах не рекомендуется**.

§6. Особенности использования массивов как параметров.

Паскаль накладывает ограничения на описание типа формального параметра. Для описания типа в заголовке процедуры допускается

для каждого параметра использовать лишь один идентификатор типа. Так как для описания массива используются два идентификатора, например, array[0..n] of real, то описать массив обычным образом не удается. Для этого существует два способа: с использованием специально введенного типа и с использованием открытых массивов.

Использование специально введенного типа:

Рассмотрим пример, в котором подпрограмма выполняет вывод значений элементов массива на экран, массив передается как параметр. Для передачи массива в подпрограмму введен тип matr.

```
program pr;
const  n=5;
type  matr=array[0..n,0..n] of real;
Procedure Proc( v:matr);
var j,m:integer;
begin
for j:=0 to  n do
 begin
    for m:=0 to n do
                write(v[j,m]:5:1);
            writeln;
          end;
end;
var
  v1:matr;
  i,i1:integer;
begin
   randomize;
  for i:=0 to  n do
    for i1:=0 to n do
                v1[i,i1]:=10*random-5;
   Proc(v1);
   end.                .
```

Использование открытых массивов:

Эта возможность очень удобна когда заранее не известно, каких размеров массив. Например:

```
program pr;
Procedure Proc(n: integer; v:array of integer);
var j:integer;
begin
   for j:=0 to  n-1 do
            write(v[j]:4);
          writeln;
end;
 var
  v1:array[0..100] of integer;
  i,n:integer;
begin
randomize;
write('Введите число элементов массива n=');
read(n);
 for i:=0 to  n do
     v1[i]:=random(10);
 Proc(n,v1);
end.
```

При использовании открытых массивов необходимо иметь в виду, что можно передавать лишь одномерный массив и начальное значение индекса массива внутри подпрограммы равно нулю.

Глава 4. ГРАФИКА В ИСР Free Pascal

§1. Библиотеки языка Паскаль

За время, прошедшее после создания языка программирования Паскаль, было написано огромное количество вспомогательных подпрограмм, которые можно использовать и в прикладных программах. Такие готовые подпрограммы собраны в библиотеки и оформлены в виде модулей. Модуль – это специальным образом

оформленный программный код, который не может быть исполнен самостоятельно, но может быть подсоединен и использован любой программой. Название имеющихся в языке Паскаль модулей, наименования составляющих их подпрограмм, можно найти в литературе.

Необходимость использовать библиотеки подпрограмм возникает, например, при рисовании на экране. При этом необходимо обычно использовать библиотеки Graph и Crt. Для подключения этих библиотек вначале программы нужно указать их имена в директиве uses:

```
Uses CRT, GRAPH;
```

Модуль CRT содержит процедуры и функции, обеспечивающие управление текстовым режимом работы экрана, работу со звуком, процедуры очистки экрана и работы с клавиатурой. Модуль GRAPH содержит обширный набор типов, констант, процедур и функций для управления графическим режимом работы экрана.

Указанные библиотеки содержат десятки процедур и функций, ниже приведено описание некоторых из них. Более полное описание этих модулей можно найти в литературе. В приведенном ниже описании указаны количество и тип параметров и тип функций, чтобы программист смог сориентироваться, какие данные можно использовать при вызове этих подпрограмм. Однако при написании операторов вызова, типы не указываются, как показано в примерах в следующем параграфе.

модуль **GRAPH**

putpixel(x, y: integer**; color:** word**) –** рисует точку с координатами (x, y) и цветом color.

line(x1, y1, x2, y2: integer**) –** рисует линию, начинающуюся в точке (x1, y1) и заканчивающуюся в точке (x2, y2).

rectangle(x1, y1, x2, y2: integer**) –** рисует прямоугольник, (x1, y1) – координаты левого верхнего угла, (x2, y2) – правого нижнего.

bar(x1,y1,x2,y2) - рисует закрашенный прямоугольник, (x1, y1) – координаты левого верхнего угла, (x2, y2) – правого нижнего. Цвет и шаблон закраски устанавливается процедурами SetFillStyle и

SetFillPattern.

circle(x, y: integer**; R:** word**) –** рисует окружность с центром в точке (x, y) и радиусом R.

arc(x,y:integer; **stangle, endangle, R :**word**);** - рисует дугу окружности, из угла stangle к endangle, с радиусом R, используя точку с координатами (x, y), как центр окружности.

ellipse(x, y: integer; **stangle, endangle, xr, xy:** word**) –** рисует дугу эллипса с центром в точке (x, y), полуосями xr, yr; stangle и endangle – начальный и конечный углы дуги (описываются от горизонтальной оси против часовой стрелки).

pieslice(x, y: integer; **stangle, endangle, R** :word) - строит сектор круга, закрашенный текущей штриховкой и цветом заполнения (устанавливается процедурами setfillstyle и setfillpattern). (x, y) — координаты центра сектора круга; stangle и endangle – начальный и конечный углы сектора, отсчитываемые против часовой стрелки от горизонтальной оси, направленной вправо; R — радиус сектора.

setcolor(color: word**) –** устанавливает цвет линии, color = 0..15.

setbkcolor(color :word); -устанавливает цвет фона color, номера цветов фона находятся в диапазоне от 0 до 15.

floodfill(x, y: integer; **border** : word); -закрашивает замкнутую область, используя стиль и цвет закраски, устанавленные процедурами setfillstyle и setfillpattern. Точка с координатами (x, y) - начальная точка внутри замкнутой области, с которой начнется закраска. Закрашивается область, ограниченная цветом с номером Border.

setfillstyle(pattern :word; **color** :word); - устанавливает цвет color и стиль закраски pattern (целое число 0-12).

outtextxy(x, y : integer; **textstring :** string**);** -отображает textstring с позиции (x,y) графического экрана.

Модуль **CRT**

delay(ms:word**)** – процедура приостанавливает выполнение программы на ms миллисекунд.

keypressed:boolean; – процедура возвращает значение TRUE, если была нажата какая-либо клавиша на клавиатуре и FALSE – если нет.

readkey:char. –Функция считывает символ, соответствующий

нажатой клавише. Если нажата функциональная клавиша, в первый раз функция принимает значение #0, во второй раз – значение кода этой клавиши. Если ни одна клавиша не была нажата, то функция приостанавливает работу программы и ожидает нажатия клавиши.

§2. Особенности реализации графического режима.

Программа пользователя может использовать один из двух режимов экрана: текстовый или графический. Текстовым называют режим работы, при котором имеется возможность управлять цветом и яркостью группы точек, образующих на экране прямоугольную матрицу, в которой размещается буква, цифра или какой-либо спецсимвол. Текстовый режим устанавливается по умолчанию, то есть всегда, когда специальными командами не установлен графический режим, закрывается - при установке графического режима

Графическим называется режим работы, при котором производится управление цветом и яркостью каждой точки экрана. Для того чтобы перейти в графический режим, его надо установить (инициализировать). Делается это с помощью процедуры

InitGraph(GraphDriver,GraphMode: Integer; Path: String);

GraphDriver -параметр, задающий тип графического драйвера, то есть специальной программы, управляющей экраном.

GraphMode –параметр, определяющий вид режима работы с выбранным драйвером. Режимы отличаются друг от друга количеством точек на экране (разрешением), палитрой цветов и, как следствие, занимаемой видеопамятью.

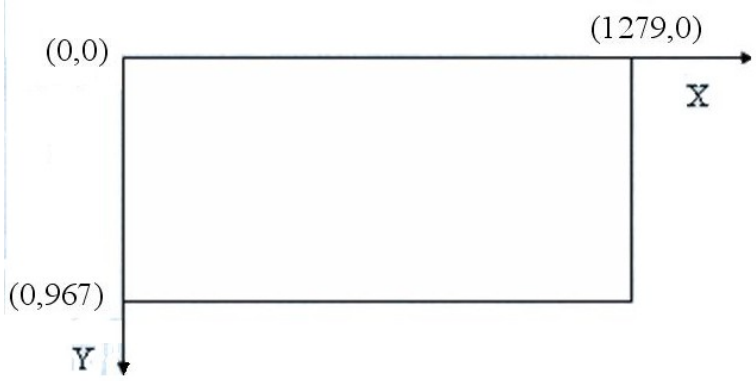
Path –параметр, содержащий путь к нужному драйверу, т.е. указывающий, в каком каталоге находится драйвер.

Обычному пользователю нет необходимости запоминать характеристики разных режимов. Поэтому имеется вариант, при котором компьютер автоматически выставляет эти параметры, используя константу detect, описанную в модуле graph:

```
program paint1;
uses graph;
```

```
var
gd,gm:integer;
begin
gd:=detect;
initgraph(gd,gm,'');
…
```

В качестве третьего параметра процедуры initgraph здесь указаны пустые апострофы. При этом процедура сама проверяет наличие и тип видеоадаптера (устройства сопряжения компьютера и монитора), загружает соответствующий графический драйвер и устанавливает значение второго параметра.



При работе с графическим режимом экрана, нужно помнить особенности расположения осей координат и знать максимальные значения координат, видимые на экране (см. рисунок - для графического режима, установленного по умолчанию).

Максимальные значения координат на экране можно узнать с помощью функций ***GetMaxX и GetMaxY***:

```
xmax:=GetMaxX;  ymax:=GetMaxY;
```

В ИСР Free Pascal при установке графического режима открывается новое окно, где и производится рисование. На панели задач внизу экрана отражается кнопка окна сообщений, названная Free Pascal и кнопка графического окна Graph Window Application. Если необходимо завершить программу нажатием клавиши Enter или управлять работой программы с помощью нажатия клавиш, активизируйте управляющее окно щелчком его кнопки на панели задач.

§3. Рисование неподвижных объектов

При написании программы, процедуры и функции, перечисленные в §1 этой главы, вызываются с указанием фактических параметров, которые могут быть как константами, так и переменными. Если изображение неподвижно, обычно используют константы.

Задача. Написать программу, которая рисует лицо человека. Овал лица (окружность), нос, уши и рот рисуются розовым цветом, глаза – синими точками.

Решение:

```
Program gr1;
uses graph;
var
grDriver, grMode: integer;

begin
grDriver:=detect;
InitGraph(grDriver,grMode,'');
Setcolor(4);
circle(200,300,100);
line(200,280,200,330);
line(180,370,220,370);
putpixel(150,250,1);
putpixel(250,250,1);
ellipse(80,300,0,360,20,30);
ellipse(320,300,0,360,20,30);
setfillstyle(1,4);
floodfill(320,300,12);
setfillstyle(5,15);
pieslice(200,230,35,145,50);
readln;
closegraph;
end.
```

Подключение графической библиотеки

Установка графического режима «по умолчанию»

Устанавливаем цвет пера

Рисуем окружность

Рисуем линию

Рисуем точку

Рисуем эллипс

Устанавливаем стиль заливки

Делаем заливку

Рисуем залитый установленным стилем сектор круга

Фиксируем картинку

Закрываем графический режим

В этой программе, чтобы дать возможность человеку рассмотреть нарисованное изображение, перед оператором closegraph, написан оператор readln. При его выполнении компьютер останавливается и ждет нажатия клавиши Enter. Для завершения программы нужно перейти в управляющее окно и нажать на Enter.

§4. Рисование движущихся объектов

Объект воспринимается как движущийся, если меняется его местоположение на экране. Перемещение объекта программируется с помощью цикла, на каждом шаге которого координаты объекта меняются на некоторую величину, постоянную, если он движется с постоянной скоростью и изменяющуюся, если объект движется неравномерно.

На каждом шаге цикла, должны быть вычислены новые координаты объекта, он должен быть нарисован, а затем стёрт, чтобы на следующем шаге его можно было нарисовать в новом месте.

При определении порядка написания операторов внутри цикла, нужно сделать всё, чтобы время, которое глаза человека рассматривают объект, было максимальным, а время, когда объект стерт – минимальным. Для этого используют следующий

Алгоритм рисования движущихся объектов:

На каждом шаге цикла

1. объект стирается (т. е. рисуется цветом фона);
2. вычисляется значение новых координат объекта;
3. объект рисуется;
4. организуется приостановка выполнения программы, чтобы глаза человека восприняли образ объекта.

В данной программе вагон движется по рельсам направо.

```
Program gr2;
uses graph,crt;
var
grDriver, grMode,i,x: integer;
begin
```

```
grDriver:=detect;                    {Инициализация (включение) }
InitGraph(grDriver,grMode,");     {   графического режима }
line(20,453,1250,453);               {  Рис. дорожки}
x:=100;                        {  Начальная координата середины вагона}
for i:=0 to 100 do
begin
    Setcolor(0);                              {  Стирание }
    rectangle(x-150,430,x+150,340);
    circle(x-100,430,20);
    circle(x+100,430,20);
    x:=x+10;                               {  Вычисление новой координаты}
    Setcolor(11);                             {  Рисование}
    rectangle(x-150,430,x+150,340);
    circle(x-100,430,20);
    circle(x+100,430,20);
    delay(100);                               {  задержка}
end;
readln;
closegraph;
end.
```

Итак, движение объекта (вагон) в этой программе организуется в цикле (операторы тела цикла сдвинуты вправо). Для того, чтобы все точки объекта двигались согласованно, нужно выбрать опорную точку (здесь это (x,430)) и все другие точки рассчитывать через ее координаты. Обратите внимание на то, что стирание и рисование производятся одинаковыми операторами.

Если рисуется сложный объект, то удобно оформить его рисование как отдельную процедуру (procedure), параметрами которой являются координаты опорной точки и цвет рисующегося объекта. Для стирания этого объекта процедура вызывается с параметром, равным цвету фона.

Если в задаче требуется нарисовать объект с отдельными движущимися частями, удобно нарисовать неподвижную часть

объекта до цикла, а подвижные части –внутри цикла.

§5. Управление объектом.

Для управления движением объекта с клавиатуры необходимо использовать функции keypressed и readkey. В задачах требуется либо начать движения объекта нажатием клавиши, либо изменить направление его движения, нажимая определенную клавишу. Следующая программа демонстрирует приемы организации таких действий.

Для начала движения нужно нарисовать объект и организовать ожидание нажатия любой клавиши с помощью цикла - цикл while в программе, приведенной ниже. Этот цикл выполняется (он, впрочем, ничего не делает), пока не нажата какая-нибудь клавиша, то есть пока логическое выражение (not keypressed)=true. В момент нажатия клавиши (not keypressed)=false, цикл завершается и компьютер переходит к выполнению следующего оператора.

Внутри цикла, рисующего движущийся шарик, оператор if сначала анализирует, была ли нажата какая-либо клавиша, а если была, то запоминает какая. Если была нажата буква “a “, то шарик продолжает двигаться влево, если – “s”, то вправо. При работе с такой программой в ИСР Free Pascal не забывайте, что система реагирует на нажатие клавиш, если активно основное, управляющее окно.

```
Program gr3;
uses graph,crt;
var  c:char;
grDriver, grMode,i,x,h: integer;
begin
grDriver:=detect;
InitGraph(grDriver,grMode,'');
line(20,453,1250,453);
x:=100; h:=5;
Setcolor(11);
circle(x,430,20);
```

```
while not keypressed do;   {ожидает нажатия любой клавиши для }
                                {начала движения}
for i:=0 to 100 do
begin
Setcolor(0);
circle(x,430,20);
x:=x+h;
Setcolor(11);
circle(x,430,20);
delay(100);
if keypressed then begin      {управление движением}
                  c:=readkey;
                  if c= 'a' then h:=-10;
                  if c= 's' then h:=10;
                 end;
end;
readln;
closegraph;
end.
```

§6. Графики функций

6.1. Построение графика функции на экране компьютера

В рассматриваемых задачах необходимо нарисовать на экране график заданной функции, например:

$$y=a\cdot Sin(x)\ ,\ x\in [x_{min},\ x_{max}].$$

Трудность заключается в том, что значения x_{min}, x_{max}, y_{min}, y_{max} могут быть отрицательными, дробными, очень большими или очень маленькими, а точки экрана нумеруются целыми положительными числами по горизонтальной оси от 0 до 1279, по вертикальной оси от 0 до 967, и эта ось направлена вниз.

Задача решается в два этапа:

1) выясняются пределы изменения x и y заданной функции (для этого надо провести анализ функции)

2) рисуется график функции с пересчетом координат (x, y) каждой точки графика в экранные координаты (i, j).

6.2.Выбор пределов изменения аргумента и заданной функции.

Задаём интервал изменения x: от x_{min} до x_{max} и распечатываем значения функции с таким шагом h (по x), чтобы в таблице было 10 – 20 строчек, например, так:

```
program grafic1;
var
 a, x, y, xmin, xmax, h: real;
begin
 write('Введите параметр a='); readln(a);
 write('Введите x min и max');
 readln( xmin, xmax);
 h:=(xmax - xmin)/20;
 x:=xmin;
 while x<=xmax do
 begin
   y:=a*sin(x);
   writeln(x:7:1, y:7:1);
   x:=x+h;
 end;
end.
```

Из таблицы на экране можно примерно определить максимальное и минимальное значения функции в этом интервале y_{max} и y_{min}. Если функция в задаче задана параметрически, т.е. x(t) и y(t), где t-параметр, то в программе именно его следует изменять с равным шагом, а вычисленные по формулам значения x и y –выводить на экран. Если изображаемая функция определена на промежутке $-\infty<t<+\infty$, то для построения графика необходимо взять достаточно большие, но конечные значения параметра, например, $-10<t<10$.

С другой стороны, изображаемые функции всегда состоят из комбинации элементарных функций, свойства которых известны. Поэтому можно определить пределы изменения значения функции,

анализируя ее вид. Например, для данной функции $y(x)=a\cdot\sin(x)$ легко видеть, что если $\sin(x)$ изменяется от –1 до 1, то $a\cdot\sin(x)$ будет изменяться от –a до a.

6.3. Программа рисования графика функции

Алгоритм рисования графика функции:

1. Задание max и min значений аргумента и функции, параметров функции, вычисление коэффициентов пересчета;
2. Подпись графика;
3. Рисование и подпись осей;
4. В цикле:
    a. вычисление очередных значений аргумента и функции
    b. изображение на экране точки (x, y);

Итак, для изображения графика заданной функции на экране, необходимо произвести масштабирование: пересчитать координаты (x,y) в экранные координаты (i, j). Это легко можно сделать, если ввести коэффициенты пересчета

$C_x=I_{ms}/(x_{max}-x_{min})$ $C_y=J_{ms}/(y_{max}-y_{min})$.

Здесь I_{ms} и J_{ms} -максимальные значения координат экрана. В любом режиме их можно найти с помощью функций

I_{ms}:=GetMaxX; J_{ms}:=GetMaxY;

x_{max}, x_{min} и y_{max},y_{min} -максимальное и минимальное значения x и y, которые нужно построить. Независимо от того, являются ли x и y параметрическими функциями (зависят от t или f) или y(x), коэффициенты пересчета координат рассчитываются по одним и тем же формулам. Экранные координаты точки (x,y) графика рассчитываются с помощью этих коэффициентов пересчета:

i= целая часть(C_x*(x-x_{min})); j= J_{ms} -целая часть(C_y*(y-y_{min})); (*)

В последней формуле также учтено, что направление настоящей оси y - вверх. Введение таких коэффициентов пересчета позволяет растянуть (или сжать) график так, что точке x_{min} соответствует экранная координата i=0, а точке x_{max} –экранная координата i=1269, в чем легко убедиться, подставив эти значения в формулу. То же самое

–для экранных координат по вертикальной оси, только y_{min} соответствует экранная координата j=967, а y_{max} соответствует экранная координата j=0.

В приведенной ниже программе для более эстетичного расположения графика на экране I_{ms} и J_{ms} сделаны меньше, при этом части графика не упираются в границы экрана.

Для увеличения читаемости графика, его следует подписать, т.е. указать название изображенной функции и значения ее параметров. Например, так:

OutTextXY(45, 15, ‘график функции y=a*Sin(x)’);

Числа в скобках – координаты точки начала текста на экране. Если часть текста, который должен быть выведен на экран, заранее не известен (например, значение параметра a), то его можно вывести с помощью переменной типа string, сформировав ее в процессе выполнения программы:

Str(a:5:1,sa); OutTextXY(45, 25, ‘ a=’); OutTextXY(45, 40, sa);

где функция Str(a,sa) переводит числовое значение переменной a по заданному формату в текст, помещаемый в строковую переменную sa, содержимое которой выводится на экран оператором OutTextXY(45, 40, sa).

Оси координат необходимо изобразить на графике с помощью операторов Line. При этом нужно помнить, что это прямые y=0 (ось абсцисс) и x=0 (ось ординат). Соответствующие экранные координаты можно получить, подставив эти значения в формулы (*) для i и j.

В программе, приведенной ниже, строится график функции y=a*Sin(x). Вы можете воспользоваться этой программой, устанавливая свои значения xmin, xmax, ymin, ymax, вписав свою функцию и исправив подпись к рисунку.

```
program grafic;
uses graph;
var
 a, x, y, xmin, xmax, h: real;
 ymin, ymax, cx, cy: real;
 gd, gm: integer;
```

```
 i, j, jmax: integer;
begin
 gd:=detect;                              {установка графического режима}
 initgraph(gd, gm, ‘’);
    {подпись графика функции}
 outtextxy(45, 15, ‘y=Sin(ax)’);
 outtextxy(45, 25, ‘a=1’);
    {вычисление коэффициента пересчета}
 ymin:=-1; ymax:=1; a:=1;
 xmin:=-3.14; xmax:=3.14; h:=0.01;
 cx:=(getmaxx-10)/(xmax-xmin);
 jmax:=getmaxy-5;
 cy:=jmax/(ymax-ymin);
      {рисование осей}
 i:=round(-xmin*cx);
 j:=jmax-round(-ymin*cy);
 setcolor(14);
 line(i, 5, i, jmax);
 line(10, j, getmaxx-10, j);
      {рисование графика}
 x:=xmin;
 while x<=xmax do
 begin
  y:=a*sin(x);          {вычисление функции}
  i:=round(cx*(x-xmin));       {вычисление экранных координат}
  j:=jmax-round(cy*(y-ymin));
  putpixel(i, j, 2);
  x:=x+h;
 end;
readln;
closegraph;
end.
```

Для параметрических функций, например, для окружности, где x=R·cos(t), y= R·sin(t), цикл будет выглядеть так (tmin и tmax

необходимо задать):

```
t:=tmin;
 while t<=tmax do
 begin
   x:=R*cos(t);
   y:=R*sin(t);
   i:=round(cx*(x-xmin));
   j:=jmax-round(cy*(y-ymin));
   putpixel(i, j, 2);
   t:=t+h;
 end;
```

Для функций, заданных в полярной системе координат, например, для трилистника r=a·cos(3f), пересчет координат и рисование графика может быть запрограммировано следующим образом:

```
f:=0;
 while f<=2*3.14159 do
 begin
   r:= a*cos(3*f);
   x:=r*cos(f);
   y:=r*sin(f);
   i:=round(cx*(x-xmin));
   j:=jmax-round(cy*(y-ymin));
   putpixel(i, j, 2);
   f:=f+h;
 end;
```

Глава 5. ЧИСЛЕННОЕ ИНТЕГРИРОВАНИЕ.

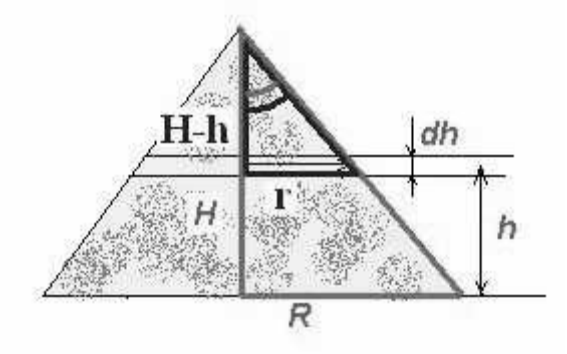
§1. Физические задачи, приводящие к интегрированию.

Интегрирование функций является составной частью многих научных и технических задач. Поскольку аналитическое интегрирование не всегда возможно, используют различные методы численного интегрирования.

Рассмотрим задачи, приводящие к интегрированию. В физике –

это задачи нахождения пути, работы, силы и др.

Задача 1. Какую работу надо затратить, чтобы насыпать кучу песка конической формы с радиусом основания R и высотой H. Плотность песка ρ.



Рассмотрим физическую модель решения этой задачи. В данном случае работа затрачивается при поднятии песка на некоторую высоту. Работу можно вычислить по формуле A=mgh, где m –масса песка, g – ускорение свободного падения, h – высота, на которую поднимают песок. Однако, трудность заключается в том, что разные части песка нужно поднимать на разную высоту. Разобьем наш конус на тонкие горизонтальные пласты толщиной dh. Рассмотрим один из таких пластов. Можно считать, что всю массу песка dm такого пласта подняли на одну и ту же высоту h.

При этом $dm= \rho \cdot dV= \rho \cdot S \cdot dh= \rho \cdot \pi r^2 dh$ и работа по поднятию этого тонкого пласта $dA=gh\, dm =\rho \cdot gh \cdot \pi r^2 dh$. Из рисунка видно, что выделенные треугольники подобны, имеют общий угол, причем

$$tg(\alpha) = \frac{R}{H} = \frac{r}{H - r}$$

Откуда $r =R \cdot (H-h)/H$. Величина h изменяется для разных пластов от 0 до H. Для того, чтобы найти полную работу, нужно просуммировать dA для всех тонких пластов. В идеале $dh \to 0$, а суммирование превращается в интегрирование.

$$A = \pi \cdot g\rho \frac{R^2}{H^2} \int_0^H (H - h)^2 h dh$$

Аналогично решается задача о выкачивании воды из резервуара.

§2.Методы интегрирования

Математическая постановка задачи: необходимо найти значение

определенного интеграла

$$I = \int_a^b f(x)dx$$

где a, b - конечны, f(x) - непрерывна на [a, b].

При решении практических задач часто бывает, что интеграл неудобно или невозможно взять аналитически: он может не выражаться в элементарных функциях, подынтегральная функция может быть задана в виде таблицы и пр. В таких случаях применяют методы численного интегрирования.

Известно, что определенный интеграл численно равен значению площади фигуры, ограниченной подынтегральной функцией, .осью x, прямыми x=a и x=b (см. рисунок ниже). Общий подход к вычислению интеграла численными методами, сводится к нахождению этой площади. Чаще всего интервал [a, b] разбивают множество меньших интервалов. Находят приблизительно площади каждой полоски и сумму площадей. Формулы численного интегрирования носят название **квадратурных формул**.

Можно выделить три группы методов:

1.Методы с разбиением отрезка интегрирования на равные интервалы. Разбиение на интервалы производится заранее, обычно интервалы выбирают равными (чтобы легче было вычислять функцию на концах интервалов). Вычисляют площади и суммируют их (методы прямоугольников, трапеций, Симпсона.

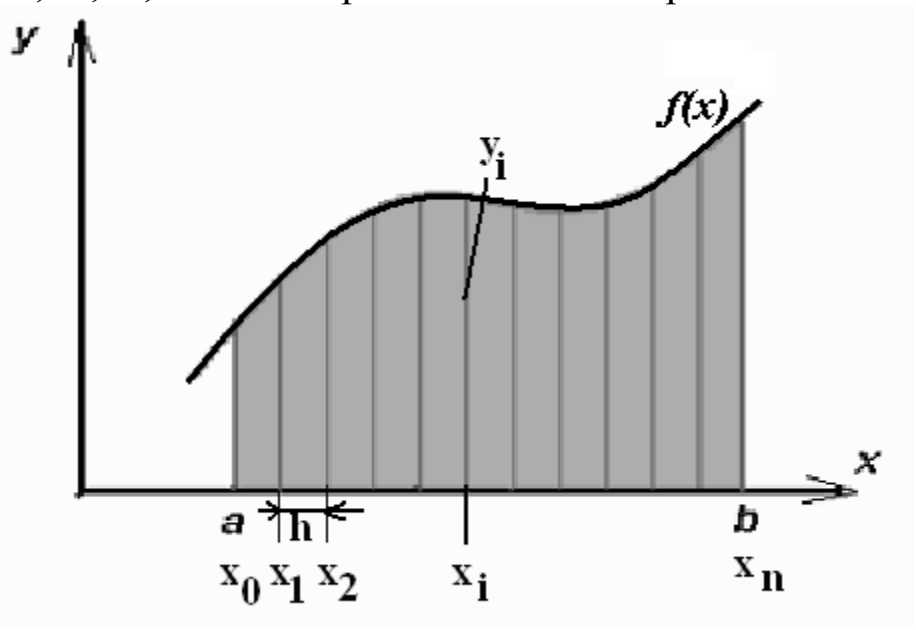
2.Методы с разбиением отрезка интегрирования с помощью специальных точек. (Формулы типа формул Гаусса.)

3.Вычисление интегралов с помощью случайных чисел (метод - Монте – Карло).

§3.Методы прямоугольников.

Для методов первой группы отрезок интегрирования [a, b] разобьем на n равных частей, таким образом определим (n+1) точку

$x_0, x_1, \ldots, x_n$. Число разбиений n выбирают.



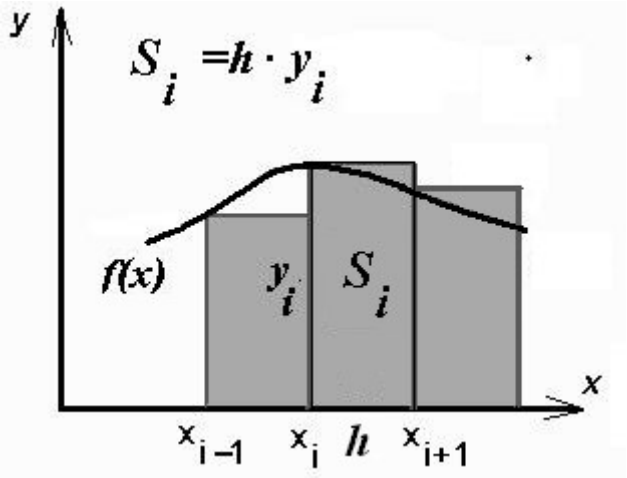
$$h = \frac{b - a}{n}$$

$$x_0 = a;$$

$$x_i = a + i * h$$

$$x_n = b$$

Здесь i –номер точки, h -шаг интегрирования, соответствующие значения функции будем обозначать $y_i = f(x_i)$.



В методе прямоугольников криволинейную трапецию, ограниченную функцией f(x) на каждом отрезке $[x_i, x_{i+1}]$ заменяют на прямоугольник. В **методе прямоугольников слева** высота прямоугольника выбирается равной $y_i=f(x_i)$ – значение функции в крайней левой точке отрезка $[x_i, x_{i+1}]$ (см.рис.). Площадь этого прямоугольника $S_i=h \cdot y_i$. Тогда интеграл приближенно может быть найден с помощью суммы

$$I \approx h \sum_{i=0}^{n-1} y_i \qquad (*)$$

Суммирование ведется с учетом того, что в первом прямоугольнике слева в качестве высоты выступает y_0, а в последнем прямоугольнике справа (внутри отрезка интегрирования [a, b]), в

качестве высоты выступает y_{n-1}.

В **методе прямоугольников справа** на каждом отрезке $[x_i, x_{i+1}]$ строится прямоугольник с высотой y_{i+1} (см. рис). Интеграл приближенно находится с помощью суммы

$$I = h\sum_{i=1}^{n} y_i ,$$

которая отличается от формулы для метода прямоугольников слева только пределами суммирования.

В **методе средних** в качестве высоты прямоугольника выбирается значение функции в точке, посередине отрезка $[x_i, x_{i+1}]$, то есть

$$y_{i+1/2} = f\left(\frac{x_i + x_{i+1}}{2}\right) \text{ и } I = h\sum_{i=0}^{n} y_{i+1/2}$$

Название метода прямоугольников, таким образом, зависит от того, в какой точке отрезка $[x_i, x_{i+1}]$, выбирается высота этого прямоугольника.

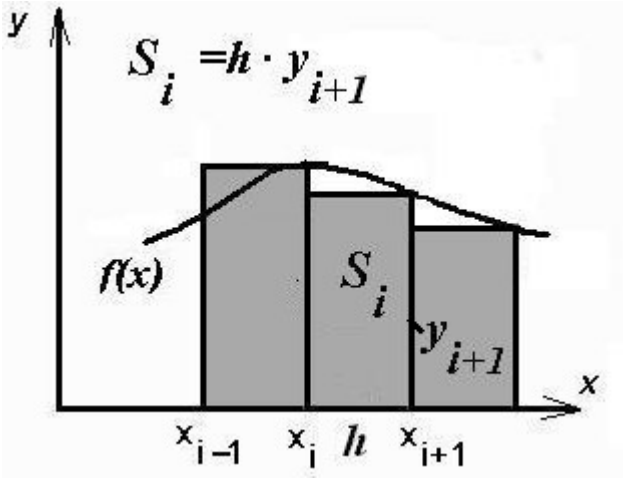


Рис. Замена криволинейной трапеции прямоугольником в методе прямоугольников справа.

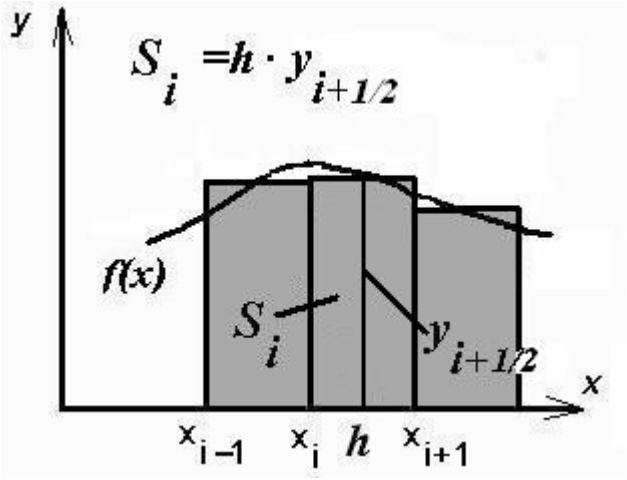


Рис. Замена криволинейной трапеции прямоугольником в методе средних.

Бывает, что подынтегральная функция задана в виде таблицы. Тогда расчет ведется с переменным шагом интегрирования. Считают, что шаг метода переменный и вычисляется по формулам $h_i=x_{i+1}-x_i$ – слева и $h_i=x_i-x_{i-1}$- справа.

§4.Метод трапеций.

В методе трапеций криволинейная трапеция на отрезке $[x_i, x_{i+1}]$ заменяется на прямолинейную, основаниями которой являются отрезки y_{i+1} и y_i. Площадь трапеции

$$S_i = h\frac{y_{i+1} + y_i}{2}$$

$$I = \sum_{i=0}^{n} S_i = \sum_{i=0}^{n} h\frac{y_{i+1} + y_i}{2} = h\frac{y_0 + y_n}{2} + h\sum_{i=1}^{n-1} y_i$$

§5.Формула Симпсона.

Площадь малой криволинейной трапеции S_i в методах, описанных выше, приближается площадью фигуры, ограниченной сверху прямой, т.е. полиномом первой степени. Понятно, что эту фигуру можно ограничить и полиномом более высокой степени. Наиболее известен метод, в котором используется так называемый полином Ньютона второй степени

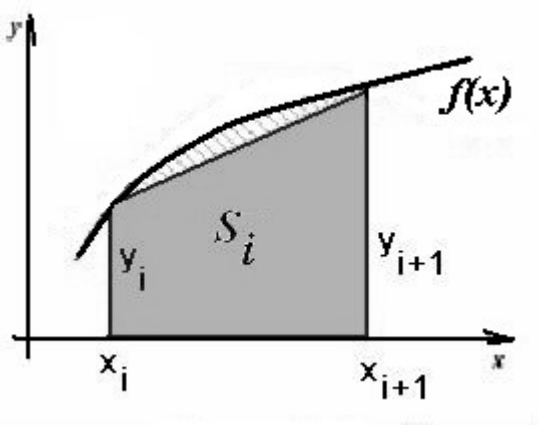


Рис. Замена криволинейной трапеции прямоугольной в методе трапеций.

$$y = y(x_{i-1}) + (x - x_{i-1})y(x_{i-1}, x_i) + (x - x_{i-1})(x - x_i)y(x_{i-1}, x_i, x_{i+1})$$

где $y(x_{i-1}, x_i)$ и $y(x_{i-1}, x_i, x_{i+1})$ –так называемые разделенные разности, числа, являющиеся комбинацией $y(x_{i-1})$, $y(x_i)$, $y(x_{i+1})$.

Полином Ньютона строится на смежных отрезках $[x_{i-1}, x_i]$ и

$[x_i, x_{i+1}]$, через три точки (x_{i-1},y_{i-1}), (x_i,y_i), (x_{i+1},y_{i+1}) и является параболой. Чтобы найти площадь криволинейной трапеции, ограниченной этой параболой, можно аналитически взять интеграл:

$$S_i = \int_{x_i}^{x_{i+1}} \left(y(x_{i-1}) + (x - x_{i-1}) y(x_{i-1}, x_i) + (x - x_{i-1})(x - x_i) y(x_{i-1}, x_i, x_{i+1}) \right) dx =$$

$$= h \frac{y_{i-1} + 4y_i + y_{i+1}}{3}$$

$$I = \int_a^b f(x)dx \approx \sum_{\substack{i \\ \text{шаг } 2}} S_i = \sum_{\substack{i=1 \\ \text{шаг } 2}}^{n-1} h \frac{y_{i-1} + 4y_i + y_{i+1}}{3}$$

- формула для нахождения определенного интеграла методом Симпсона.

§6.Метод Гаусса

Расчет интеграла в данном методе осуществляется в два этапа:

А) интеграл с пределами интегрирования [a,b] сводится к интегралу с пределами [-1,1]. $\int_a^b f(x)dx \to \int_{-1}^{1} Y(\mu)d\mu$

Б) полученный интеграл рассчитывается как сумма значений подынтегральной функции в специальных точках, умноженных на весовые коэффициенты.

$$\int_{-1}^{1} Y(\mu)d\mu = \sum_i A_i Y(\mu_i)$$

Для изменения пределов интегрирования делается замена переменных

$$\mu = \frac{2x - (b + a)}{(b - a)}$$

при этом если переменная x € [a,b], то переменная μ € [-1,1].

$$X = \frac{1}{2}(b - a)\mu + \frac{1}{2}(b + a)$$

тогда

$$dx = \frac{1}{2}(b - a)d\mu$$

$$f(x)dx = f(\frac{1}{2}(b - a)\mu + \frac{1}{2}(b + a))\frac{1}{2}(b - a)d\mu = Y(\mu)d\mu$$

Таким образом, этой заменой переменной интеграл с любым ограниченным отрезком интегрирования можно свести к виду

$$I = \int_{-1}^{1} Y(\mu)d\mu$$

Например, $I = \int_{0}^{2} x^2 dx$, $\mu = \frac{2x - 2}{2} = x - 1$, x=μ+1

$$Y(\mu) = \frac{1}{2}(2 - 0)(\frac{1}{2} \cdot (2 - 0)\mu + \frac{1}{2}(2 + 0))^2 = (\mu + 1)^2 \text{ и } \int_{0}^{2} x^2 dx = \int_{-1}^{1} (\mu + 1)^2 d\mu$$

Если в качестве Y(μ) брать степенную функцию, то можно подобрать такие веса A_i такие, что выполняется точное равенство

$\int_{-1}^{1} Y(\mu)d\mu = \sum_{i=1}^{n} A_i Y(\mu_i)$, где μ_i- корни полиномов Лежандра (специальные функции) степени n, A_i - коэффициенты. Для других функций эта формула будет приближенной. A_i и μ_i для разных n уже вычислены и сведены в таблицу.

n	μ_i	A_i
1	0	2
2	± 0.57735027	1
3	± 0.77459667 0	0.5555556 0.88888889
4	± 0.86113631 ± 0.33998104	0.34785484 0.65214516

Заметьте, что μ_i симметричны относительно начала координат, а коэффициенты A_i - одинаковы для $\pm\mu_i$

Рассмотренный выше в качестве примера интеграл может быть приближенно рассчитан по формуле Гаусса по трем точкам так:

$$\int_0^2 x^2 dx = \int_{-1}^{1} (\mu + 1)^2 d\mu \approx \sum_{i=1}^{3} A_i (\mu_i + 1)^2 =$$

$$=0.5555556\cdot(-0.77459667+1)^2+0.5555556\cdot(0.77459667+1)^2+$$
$$+0.88888889\cdot(0+1)^2=2.6666668$$

Точное значение интеграла

$$\int_0^2 x^2 dx = \frac{x^3}{3}\Big|_0^2 = \frac{2^3}{3} = 2.6666667$$

§7.Программирование методов численного интегрирования

Рассмотрим программу, которая рассчитывает работу по насыпке кучи песка конической формы. Чтобы оценить точность численного расчета, найдем интеграл аналитически:

$$A = \pi \cdot g\rho \frac{R^2}{H^2} \int_0^H (H - h)^2 h dh = k \cdot \frac{H^4}{12}, \quad k = \pi \cdot g\rho \frac{R^2}{H^2}$$

Численно будем находить интеграл по формуле прямоугольников слева $I \approx h\sum_{i=0}^{n-1} y_i$ с помощью цикла. Число разбиений n введем с клавиатуры. В конце программы выведем на экран оба (численное и аналитическое) полученных значения.

```
program pr;
const a=0; b=2;g=9.8;ro=5e3;
var k,h,si,R,Hb,hm,sum,int,int1:real;
 i,n:integer;
begin
Hb:=b; R:=2;
write('n=');
read(n);
h:=(b-a)/n;
sum:=0;
for i:=0 to n-1 do
begin
  hm:=a+i*h;
  si:=sqr(Hb-hm)*hm*h;
  sum:=sum+si;
end;
k:=pi*g*ro*sqr(R/Hb);
int:=k*sum;
int1:=k*sqr(Hb)*sqr(Hb)/12;
writeln('*************  n=',n,'  *******');
writeln(' chislenno  I=',int:7:4);
writeln(' tochno     I=',int1:7:4);
end.
```

Программа численного расчета интеграла по методу прямоугольников справа будет отличаться от приведенной только пределами суммирования в цикле. В методе трапеций на каждом шаге цикла нужно рассчитывать два значения подынтегральной функции:

```
for i:=0 to n-1 do
begin
  hm:=a+i*h;
  hm1:=a+(i+1)*h;
  si:=h/2*(sqr(Hb-hm)*hm+sqr(Hb-hm1)*hm1);
  sum:=sum+si;
end;
```

В методе Симпсона лучше воспользоваться оператором цикла другого вида, так как суммирование нужно вести с шагом 2:

```
sum:=0; i:=1;
while  i<n do
begin
  hm:=a+i*h;
  hm1:=a+(i+1)*h;  hm_1:=a+(i-1)*h;
  si:=h/3*(sqr(Hb-hm_1)*hm_1+4*sqr(Hb-hm)*hm+sqr(Hb-hm1)*hm1);
  sum:=sum+si;
  i:=i+2;
end;
```

Точность полученного при численном интегрировании результата

зависит от выбранного числа разбиений. Например, формула метода прямоугольников слева при n=100 даст значение I=205230.1950, при n=1000 I=205250.5148, тогда как точное значение интеграла I=205250.7200. Метод трапеций точнее методов прямоугольников: при том же числе разбиений ошибка будет меньше. Самым точным из представленных выше является метод Симпсона: даже при n=10 I=205250.7200.

Обычно численное интегрирование используется когда найти значение интеграла аналитически достаточно трудно. Поэтому оценить точность численного результата путем сравнения с точным значением нельзя. Поэтому поступают так. Сначала выбирают точность ε, с которой нужно получить значение интеграла. Точность ε - это достаточно малое положительное число, обычно 10^{-3}-10^{-6}. Если выбрать, например, ε=0.001, то можно считать, что в полученном интеграле все цифры до сотых будут верными. Затем выбирают число разбиений n, вычисляют значение интеграла I_n, затем число разбиений удваивают и вновь вычисляют значение интеграла I_{2n}. Точность считается достигнутой, если $|I_{2n} - I_n| < \varepsilon$.

Для достижения большей точности выбирают меньший шаг интегрирования, или другой метод.

Математические функции.

Функция	**назначение**	**Тип аргумента**	**Тип значения функции**
abs(x)	x	Целое, вещественное	Целое, вещественное как у аргумента
Pi	Число π	Целое, вещественное	Вещественное
sin(x)	sinx, x-радианы	Вещественное (радиан)	Вещественное
cos(x)	cosx, x-радианы	Вещественное (радиан)	Вещественное
ArcTan(x)	arctg(x), в радианах, в пределах от –π/2 до π/2	Вещественное	Вещественное, в радианах, в пределах от –π/2 до π/2
sqrt(x)	√x , x>0	Целое, вещественное	Целое, вещественное как у аргумента
sqr(x)	x^2	Целое, Вещественное	Как у аргумента
exp(x)	e^x	Вещественное	Вещественное
ln(x)	ln x, x>0	Вещественное	Вещественное
trunc(x)	целая часть x	Вещественное	longint
frac(x)	дробная часть x	Вещественное	Вещественное
int(x)	целая часть x	Вещественное	Вещественное
round(x)	округление x до ближайшего целого	Вещественное	longint
Random	случайное число [0,1]	-	Вещественное
random(x)	случайное число [0,x]	Word	Word

Основные операторы языка Паскаль

Общий вид оператора	Примеры	Пояснение примеров
Writeln(список); Write(список); -Оператор вывода на экран, в файл	Writeln(‘x=’);	Выводит на экран текст «x=» без кавычек
	Writeln(x);	Выводит на экран значение переменной x
	Writeln(x:5:1);	Выводит на экран значение действительной (real) переменной x, отводя всего 5 позиций, из них 1 –после десятичной точки
	Writeln(i:4);	Выводит на экран значение целой (integer) переменной i, отводя всего 4 позиции
	Writeln(‘x=‘,x:5:1,’ m’);	При x=3,1 выведет на экран: x= 3.1 m
	Writeln;	Вызывает перевод курсора на следующую строку.
read(список); - Оператор ввода с клавиатуры, из файла	read(x);	Вводит с клавиатуры значение переменной x
	read(x,y);	Вводит с клавиатуры значение переменных x и y
	readln;	Приостановливает выполнение программы до нажатия клавиши Enter
	writeln(‘x=’); read(x);	Операторы организуют ввод с клавиатуры с «подсказкой»
Переменная:= выражение; - Оператор присваивания	n1 := 1;	Переменной n1 присваивается значение константы 1
	a:=7.5+i;	Переменной a присваивается значение выражения
	i:=i+1;	Переменная i увеличивается на

		1
	b:=(1+sqr(a))/(3*a);	$b = \frac{1+a^2}{3a}$
	l:=sqrt(x*x+y*y);	$l = \sqrt{x^2 + y^2}$
	b:=exp(a*ln(x));	$b = e^{a \cdot \ln(x)}$ соответствует $b = x^a$
	c:=1E3*cos(sqrt(2*x+1))/(abs(y)*(2.17E-12+exp(5*ln(x))));	$c = \frac{10^3 \cos(\sqrt{2x+1})}{\lvert y \rvert \cdot (2.17 \cdot 10^{-12} + x^5)}$
If условие then оператор1 else оператор2; -условный оператор	if a>b then max:=a else max:=b;	Если a>b, то переменной max присваивается значение переменной a, иначе, переменной max присваивается значение переменной b
	if a>b then max:=a;	Если a>b, то переменной max присваивается значение переменной a, в других случаях никаких действий не выполняется
for параметр:= нач.знач. to кон.знач. do оператор; -счетный цикл	for i:=0 to 10 do writeln(i);	Выводит на экран в столбик целые числа от 0 до 10
	s:=0; for i:=1 to 10 do s:=s+i;	Суммирует целые числа от 1 до 10

while условие do оператор;
 	
-цикл с предусловием
	i:=0;
	while i <= 10 do
	begin writeln(i);
	i:= i + 1;
	end;
		Цикл повторяется, пока условие верно. Выводит на экран в столбик целые числа от 0 до 10

	i:=0; s:=0; while i < 10 do	Суммирует целые числа от 1 до 10

	begin i:= i + 1; s:=s + i; end;	

repeat
оператор;
until
условие;
	i:=0;
	repeat
	writeln(i);
	i:= i + 1;
	until i>10;
		Цикл повторяется, пока условие неверно. Выводит на экран в столбик целые числа от 0 до 10

-цикл с постусловием	i:=0; s:=0;
	repeat
	i:= i + 1;
	s:=s + i;
	until i>=10;
		Суммирует целые числа от 1 до 10

Литература

1. Рудаков П.И., Федотов М.А. Основы языка Паскаль.-М:Радио и связь, 2000.
2. Поляков Д.Б., Круглов И.Ю. Программирование в среде Турбо Паскаль (версия 5.5).-М:МАИ, 1992.
3. Турбо Паскаль 7.0. К:Торг.-изд.бюро BNV, 1996.
4. Фаронов В.В. Турбо Паскаль 7.0. Практика программирования: Учебное пособие. -М:Нолидж, 1997.
5. А.П. Полищук, С.А. Семериков. Программирование в X Window средствами Free Pascal. –С.Пб:БХВ-Петербург, 2003.
6. И.Л.Шихалев. Основы программирования для Win32 на Free Pascal. -М:Бином, 2005.

Павлова Татьяна Юрьевна

Структурное программирование В ИСР « Free Pascal »

Учебное пособие

Редактор
Подписано к печати . Формат 60х84 1/16. Печать офсетная. Бумага офсетная № .Уч.-изд.л.___. Печ.Л.___. Тираж 100 экз. Заказ №