

В.Н. Мальцев, Ф.Д. Поликарпов

ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ ПАСКАЛЬ

Учебное электронное текстовое издание
Подготовлено кафедрой общей и молекулярной физики

Учебно-методическое пособие с заданиями
для практических занятий по дисциплине «Программирование»
для студентов всех форм обучения направлений 03.03.02 «Физика»
и 03.03.03 «Радиофизика».

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	3
1. ПРОГРАММА НА ПАСКАЛЕ. НАЧАЛЬНЫЕ СВЕДЕНИЯ.....	6
2. УСЛОВНЫЕ ОПЕРАТОРЫ	21
2.1. Переменные логического типа	21
2.2. Логическое выражение.....	21
2.3. Условный оператор.....	23
2.4. Составной оператор (операторные скобки begin-end)	24
2.5. Алгоритм.....	25
3. ОПЕРАТОР МНОГОВАРИАНТНОГО ВЫБОРА	29
4. ОПЕРАТОРЫ ЦИКЛА. СЧЕТНЫЕ ОПЕРАТОРЫ ЦИКЛА.....	34
4.1. Оператор с увеличивающимся счетчиком цикла	34
4.2. Оператор с уменьшающимся счетчиком цикла.....	35
5. ОПЕРАТОРЫ ЦИКЛА. УСЛОВНЫЕ ЦИКЛЫ	38
5.1. Оператор цикла с предусловием	38
5.2. Оператор цикла с постусловием.....	39
5.3. Рекуррентные формулы.....	40
6. МАССИВЫ	44
7. СИМВОЛЫ. СТРОКИ	48
8. ФУНКЦИИ И ПРОЦЕДУРЫ	51
9. ПРОЦЕДУРЫ	55
10. ФУНКЦИИ И ПРОЦЕДУРЫ. ПРОЦЕДУРНЫЙ ТИП ДАННЫХ	59
11. ФАЙЛЫ. ВВОД-ВЫВОД В ФАЙЛ.....	64
ЗАКЛЮЧЕНИЕ	69
ЛИТЕРАТУРА	71

ВВЕДЕНИЕ

Данное пособие включает в себя материалы для практических занятий в рамках дисциплины «Программирование» для студентов 1-го курса направлений «Физика» и «Радиофизика» Уральского федерального университета. Поскольку в большинстве случаев использование навыков программирования студентами данных направлений предполагает реализацию вычислительных алгоритмов для решения физических задач, то основной целью практических занятий было – освоение студентами императивной парадигмы программирования, т.е. методов построения вычислений через задание последовательности шагов исполнения и вызова процедур. Изучение методов программирования ведется с использованием языка Паскаль. Выбор языка программирования обусловлен рядом причин. Во-первых, Паскаль является хорошо зарекомендовавшим себя инструментом для обучения программированию и хорошей базой для быстрого освоения других императивных языков. Во-вторых, имеются свободные реализации среды разработки на этом языке. В-третьих, современные реализации языка позволяют создавать программы, сравнимые по функционалу и по скорости выполнения с программами на языках промышленного уровня. В-четвертых, имеется огромное количество учебной литературы по Паскалю и его применению для решения различных вычислительных задач.

Нужно заметить, что данное пособие не является учебником по Паскалю. Более того, часть конструкций языка и типов данных в пособии вообще не рассматривается, также не рассматривается возможность рекурсивных вызовов функций и процедур и возможности для реализации объектно-ориентированной парадигмы программирования. Однако, изучаемого подмножества Паскаля достаточно для создания программ для решения вычислительных задач. Пособие может быть использовано и для самостоятельного освоения основ программирования.

Семестровый курс обучения содержит практические занятия, на которых осваивается использование конструкций языка и основные вычислительные алгоритмы, в течение семестра проводится два контрольных занятия для определения успешности освоения учебного материала, а также одно-два занятия для выполнения и сдачи домашних заданий.

Пособие разбито по занятиям. Последовательность занятий соответствует порядку представления теоретического материала на лекциях. В начале каждого занятия приводится справочная информация и примеры, относящиеся к теме занятия. Основной теоретический материал дается на лекциях, поэтому справочная информация представлена очень кратко. Предполагается, что на вопросы, касающиеся темы занятия, и которые недостаточно подробно освещены в пособии, можно получить ответ от преподавателя. Далее дается набор заданий для самостоятельного выполнения в учебном классе, при этом предполагается, что при выполнении можно консультироваться с преподавателем. Но, как показывает опыт, справочного материала вполне достаточно для успешного выполнения заданий. Задания, невыполненные в течение занятия, переходят в разряд домашних, их можно выполнить дома и показать преподавателю в установленное для этого время.

Для эффективного обучения студентам настоятельно рекомендуется перед занятием ознакомиться со справочной частью занятия. Это позволит определить те части материала, которые вызывают трудности в понимании, и сформулировать для себя вопросы. Кроме того, во время подготовки можно повторить справочный материал предыдущих занятий. Объем справочной части занятий с каждым занятием становится меньше и на последних занятиях составляет всего 2 страницы, поэтому предварительная подготовка потребует немного времени.

Список литературы составлен на основе предложенного Научно-исследовательским институтом мониторинга качества образования на сайте <http://www.i-exam.ru/>.

Программное обеспечение для выполнения домашних работ может быть взято из следующих источников.

1. <http://pascalabc.net/>. На этом сайте можно найти также описание языка и задачник по Паскалю.
2. <http://www.freepascal.ru/>. На этом сайте, кроме компилятора Паскаля можно найти интегрированную среду программирования на Паскале, а также среду для визуального программирования Лазарус (свободный аналог Delphi).
3. <http://tpdn.ru/files/turbo-pascal-download/>. На этом сайте можно скачать TurboPascal 7.1 – версию компилятора и среды разработки фирмы Borland (1992 год) с русифицированным файлом Помощи.

1. ПРОГРАММА НА ПАСКАЛЕ. НАЧАЛЬНЫЕ СВЕДЕНИЯ

Программа на Паскале представляет собой последовательность операторов и команд, записанных по определенным правилам. Эти операторы и команды выполняются последовательно в порядке записи в программе: слева направо, сверху вниз.

Текст программы набирается с помощью редактора текста и записывается в файл с расширением «pas».

Перевод текста программы в исполняемый код осуществляется с помощью специальной программы, называемой компилятором.

При наличии синтаксической ошибки на стадии компиляции в специальном окне или в специальной строке выводится сообщение, в котором указывается номер строки, содержащей ошибку, и указывается характер ошибки, а курсор в окне редактора устанавливается на следующей позиции от места, в котором найдена ошибка, если работа выполняется в интегрированной среде разработки программы.

Оператор – конструкция из символов и зарезервированных (служебных) слов языка Паскаль, определяющая характер выполняемых действий. Операторы обязательно отделяются друг от друга символом «;» (точка с запятой). В тексте программы последовательность операторов может быть записана в одной строке или по одному оператору в отдельной строке.

Комментарием называется любой текст, заключенный между фигурными скобками «{» и «}». «Закомментированный» текст может включать любое количество строк программы. Комментарии не считаются частью кода программы, т.е. «закомментировав» последовательность операторов и команд, мы исключаем эти операторы и команды из кода программы (но не из файла, в который записан текст программы).

В общем случае программа на Паскале состоит из трех основных частей: заголовка программы, раздела описания и основного блока программы (исполняемая часть).

```

PROGRAM <имя программы>; {ЗАГОЛОВОК ПРОГРАММЫ
(необязательный)}
{Директивы компилятору (необязательный)}
uses .....{Блок описания модулей (необязательный)}
const .....{Блок описания констант (необязательный)}
type..{Блок описания нестандартных типов (необязательный)}
var .....{Блок описания переменных (необязательный)}
procedure .....{Блок описания процедур (необязательный)}
function .....{Блок описания функций (необязательный)}
{ОСНОВНОЙ БЛОК ПРОГРАММЫ}
BEGIN
<оператор или группа операторов>
END.

```

Описательная часть всегда находится перед основным блоком программы. В описательной части, как следует из названия, описываются все используемые в программе имена переменных, констант, меток, а также все используемые в тексте программы функции и процедуры, созданные пользователем. Если какое-нибудь имя или функция не будут описаны, то компилятор укажет на это, что также может означать, что в тексте программы это имя записано с ошибкой.

Программа содержит обязательные элементы и необязательные. Так, раздел «Заголовок программы», а также некоторые блоки описаний могут быть пропущены.

Кратчайшая программа (ничего не выполняет) будет выглядеть следующим образом.

```

Begin
End.

```

Следует обратить внимание на тот факт, что программа должна обязательно заканчиваться оператором End с точкой!

Идентификаторы. Все имена переменных, констант, функций, процедур и модулей, определяемые программистом, должны быть идентификаторами. Имя, начинающееся с символа латинского алфавита или подчеркивания, называется идентификатором.

В Паскале не различаются прописные и строчные буквы, т.е. имена «ONE», «One», «oNe» и «one» будут восприниматься как одно и то же имя. Следует заметить, что не во всех языках программирования выполняется это правило. В качестве имени может использоваться последовательность, начинающаяся с любого символа латинского алфавита и содержащая символы латинского алфавита, цифры, знак подчеркивания. Использовать пробел в имени нельзя. Имя переменной или функции не может совпадать с именем встроенной функции (Sin, Cos, Exp и т.п.) или зарезервированного слова (Program, Begin, For и т.п.).

Правильно записанные идентификаторы: a1, VariableName1, next_variable_1, h123_35a, omega_, sin1.

Неправильные идентификаторы:

1variable – начинается с цифры;

имяпеременной – используются символы, отличные от символов латинского алфавита;

variable-name – используются неразрешенные символы;

atan, begin – имена совпадают с именем встроенной функции арктангенса и зарезервированным словом языка;

old name – в имени используется пробел.

Переменная – область физической памяти компьютера, содержание которой может меняться. К содержащемуся значению в этой области памяти обращаются, указывая не адрес ячеек, а их идентификатор (идентификатор переменной). Во время работы программы содержание этой области памяти не меняется до тех пор, пока в программе не появится оператор присваивания, изменяющий значение, содержащееся в этих ячейках. В некоторых компиляторах при запуске программы во все объявленные переменные

записываются нулевые значения, однако, при выполнении заданий рекомендуем задавать начальные значения переменных явным образом.

Тип переменной или константы определяет формат памяти, выделяемой для хранения значений, диапазон допустимых значений, а также операции, которые можно производить с данными переменными и константами.

Объявление переменных начинается с ключевого слова **Var** (от английского *variable* – переменная), после которого в этой же строке или на следующих строках через запятую перечисляются имена переменных одного типа. После перечисления ставится знак «:» (двоеточие) и записывается тип переменной: **Real** (вещественный) – для вещественных переменных, **Integer** (целый) – для целых, **Char** (от английского *character* – символ) – для символов; **String** (строка) – для строк. После указания обязательно ставится знак «;» (точка с запятой). Формально это записывается так:

```
Var <переменная1>, <переменная2>, <переменная3>: <тип
переменных>;
```

Или

```
Var
<переменная1>: <тип переменной1>;
<переменная2>: <тип переменной2>;
<переменная3>: <тип переменной3>;
```

Например:

```
Var a, b: real;
Var
c, d, q: real;
g: char;
s1, s2: string;
```

В описательной части слово **Var** с описаниями переменных может встречаться сколько угодно раз.

Вещественное число, в отличие от целого, содержит целую и дробную части, разделенные точкой.

1.2

-0.75

0.002

$0.2e-2$ (здесь записано $0.2 \cdot 10^{-2} = 0.002$), после буквы *e* следует показатель степени числа 10.

Символ – отдельный символ, заключенный в апострофы: 'q'; 'ж'; '_'; '1'; '!' и т.д. и т.п.

Строка – последовательность символов, заключенных в апострофы: 'stroka latinskih simvolov'; 'строка символов на русском'; 'строка с цифрами 12345'.

Почему нужно указывать тип переменных? Очевидно, что число 1, число 1.0, символ '1' и строка 'one', по смыслу выражают одно и то же, но для компьютера это разные сущности: 1 – целое число; 1.0 – вещественное число, имеющее дробную часть, равную нулю, '1' – символ с определенным номером (кодом символа) в таблице символов; 'one' – последовательность символов. Или иначе говорят, что эти значения имеют разные типы, на их хранение отводится различное число ячеек памяти. В выражении $1 + 'one'$ компьютеру предлагается выполнить арифметическую операцию сложения числа и строки, которые в компьютере представлены по-разному, естественно, что компилятор определит это выражение как ошибочное. С другой стороны, вещественное число фактически состоит из двух – целой части и дробной, поэтому значением выражения $1 + 1.0$ будет 2.0. Здесь целые числа сложились, а дробная часть осталась без изменения, поэтому результат – вещественное число, получившееся значение может быть присвоено только той переменной, которая описана как вещественная, т.е. в ней имеется все необходимое для хранения целой и дробной частей.

Константы – область физической памяти, содержание которой определено в самом начале работы программы, и которое не меняется на

протяжении всей работы программы. Обращение к содержимому происходит при указании идентификатора константы.

Объявление констант. Константы перечисляются после ключевого слова `Const` (от английского `constant` – постоянная), при этом они отделяются друг от друга знаком «;» (точка с запятой). Константа записывается следующим образом: записывается имя константы, затем ставится знак равенства «=», после которого записывается значение константы. Формально это записывается так:

```
Const <константа1> = <значение константы1>;  
<константа2> = <значение константы2>;
```

или

```
Const  
<константа1> = <значение константы1>;  
Const  
<константа2> = <значение константы2>;
```

Например:

```
Const  
pi2 = 1.5708;  
number = 5;  
symbol = 'G';  
title = 'Название программы';
```

Как и в случае переменных, описание констант может встречаться в описательной части программы сколько угодно раз. Порядок следования описаний переменных и констант может быть произвольным. Можно использовать встроенную константу `pi = 3.141592653`.

Некоторые наиболее часто встречающиеся типы данных и некоторые их характеристики приведены в табл. 1.

Типы данных

Название	Объем памяти, в байтах	Диапазон значений
Integer (целый со знаком)	2	-32768...+32767
Word (целый без знака)	2	0...65535
Real (вещественный)	6	$\pm 2.9 \cdot 10^{-39} \dots \pm 1.7 \cdot 10^{38}$
Extended (вещественный расширенный)	10	$\pm 3.4 \cdot 10^{-4932} \dots \pm 1.1 \cdot 10^{4932}$
Char (символ)	1	
String (строка символов)	не больше 255	

Оператор присваивания в Паскале представляет собой составную конструкцию, состоящую из двоеточия и знака равенства, применяющуюся следующим образом

<идентификатор> := <выражение>;

Слева стоит идентификатор переменной, значение которой мы меняем. Это значение равно вычисленному значению <выражения>. Типы идентификатора и значение выражения должны совпадать. Выражение может представлять собой математическое выражение, составленное из других идентификаторов и чисел, другой идентификатор или явно указанное значение.

Пример:

c := 123;

d := 35/7+1;

sy := 'f';

s1: = 'это строка';

Надо запомнить, что переменные, которым мы присвоили значение, сохраняют это значение до тех пор, пока мы не присвоим им другое значение. Допустим, имеются две области памяти для переменных целого типа, которым дали имена `first` и `second`. Их первоначальное содержание не определено. В переменную `first` поместили значение 10, а в переменную `second` – значение 4. Это выполняется с помощью последовательности операторов присваивания

`First := 10; second := 4;`

Допустим, далее в программе появился еще один оператор присваивания

`first := first-second-1,`

из содержимого `first` вычесть содержимое `second` и единицу, результат поместить в `first`. Тогда после выполнения этого оператора в ячейках, имеющих идентификатор `first`, т.е. в переменной `first`, будет храниться значение 5, а в ячейках, имеющих идентификатор `second`, т.е. в переменной `second`, по-прежнему будет храниться значение 4.

Для числовых типов данных (`Integer`, `Word`, `Real`, `Extended`) определены арифметические операции: сложения (+), вычитания (-), умножения (*) и деления (/). При выполнении арифметических действий сначала выполняются операции умножения и деления, а затем операции сложения и вычитания. Поэтому для правильной записи выражения используются скобки. В этом случае сначала вычисляется значение выражения в скобках, а затем выполняются другие арифметические операции. Например, результатом вычисления выражения `12/3+3` будет число 7, а результатом вычисления выражения `12/(3+3)` будет число 2.

Надо помнить, что результатом деления двух целых чисел (например, `13/2`) может быть вещественное число, такой результат может быть присвоен только переменной вещественного типа.

Для данных целого типа определена операция целочисленного деления `Div` (от английского `divide` – делить), которая возвращает частное от деления. Например, результатом целочисленного деления `7 div 3` будет целое число 2. Для данных целого типа определена также операция `Mod`, возвращающая

остаток от целочисленного деления, например, результатом $7 \bmod 3$ будет целое число 1.

В вычислительных задачах чаще всего важна скорость вычисления выражений, поэтому при записи выражения нужно пытаться использовать меньшее число арифметических операций и вызовов функций. Наиболее медленной является операция деления, поэтому следует стремиться к тому, чтобы заменить ее операцией умножения. Например, не делить выражение на 2.0, а умножать его на 0.5. Умножение более медленное, чем сложение и вычитание, поэтому нужно уменьшать число таких операций. Иногда возникают ситуации, когда удобно ввести переменные для промежуточных выражений, которые используются несколько раз в другом выражении. Тогда сложное выражение можно будет записать через промежуточные переменные, которые вычисляются всего лишь один раз. Например, код для вычисления выражения $\sin(1+x^2) + (1+x^2)^2 + \frac{x^2}{2}$ можно записать таким образом:

```
a:=1+x*x;
```

```
q:=Sin(a)+a*a+0.5*(a-1);
```

или еще более экономно:

```
q:=Sin(a)+a*(a+0.5)-0.5;
```

Отметим некоторые встроенные математические функции.

- **Тригонометрические** (аргумент вещественный, возвращают вещественное значение): Sin(x); Cos(x); Arctan(x) – синус, косинус и арктангенс x (x – угол в радианах).
- **Экспонента и логарифм** (аргумент вещественный, возвращают вещественное значение): Ln(x); Exp(x) – логарифм x и e^x , соответственно.
- **Модуль, квадратный корень, квадрат**. Abs(x) (x также может быть целым); Sqrt(x); Sqr(x) – модуль, квадратный корень и квадрат x, соответственно.

- **Целая и дробная части числа:** `Int(x)`, `Frac(x)` – возвращают целую и дробную часть числа `x`.

В языке программирования Паскаль определено довольно много различных типов данных, но для выполнения заданий этой и нескольких последующих практических работ достаточно будет знать, как объявить переменные вещественного и целого типа, символьные и строковые переменные, а также как объявить константы. Напоминаем, что все объявления производятся в описательной части программы.

Для ввода данных, используемых в программе, имеются две команды `ReadLn(<список переменных>)` и `Read(<список переменных>)`, где `<список переменных>` – последовательность переменных, разделенных запятой. Переменные должны быть описаны в области описания. Список может быть пустым. Отличие первой команды от другой заключается в том, что после чтения значений для переменных из списка, курсор перемещается вниз на следующую строку (`Ln` от слова `line` – линия).

При выполнении команд `ReadLn` и `Read` программа приостанавливает работу и ждет ввода значений с клавиатуры или считывает данные из файла, если ввод данных происходит через файл.

При вводе с клавиатуры значения переменных вводятся в строку через пробел в том порядке, в котором они перечислены в «списке переменных». Передача значений в программу происходит после нажатия на клавишу «ENTER». Можно передавать значения по одному, нажимая «ENTER» после каждого введенного значения.

Если введенных значений будет больше, чем переменных в `<список переменных>`, то первые значения будут присвоены переменным из списка, а лишние значения будут отброшены.

Например: требуется ввести значения для трех переменных целого типа `a, b, c`. Для этого используется команда `ReadLn(a, b, c)`. При исполнении программы, она приостановит свое выполнение в ожидании ввода значений. Например, набрав на клавиатуре через пробел числа `1 -5 20` и нажав «ENTER»,

программа продолжит свое выполнение, при этом значения переменных будут следующими: $a=1$, $b=-5$, $c=20$.

Если ввести `ReadLn`; без списка переменных, то программа приостановит свое выполнение и возобновит работу только после нажатия на «ENTER».

Для вывода данных используются команды `WriteLn(<список переменных и выражений>)` и `Write(<список переменных и выражений>)`, где <список переменных и выражений> – последовательность переменных, чьи значения надо вывести и/или вычисляемое выражение и/или символы, строки, перечисленные через запятую.

Оператор `WriteLn` отличается от `Write` тем, что после вывода <список переменных и выражений> курсор переместится на следующую строку. Например, эта часть программы выведет результаты в двух строках:

```
a:=3;
b:=5;
WriteLn('разность a-b= ', a-b);
WriteLn('сумма a+b= ', a+b);
```

На экране:

```
разность a-b= -2
сумма a+b= 8
```

С использованием команды `Write` результат будет выведен в одну строку

```
a:=3;
b:=5;
Write('разность ', a, ' - ', b, ' = ', a-b);
Write('сумма ', a, ' + ', b, ' = ', a+b);
```

На экране:

```
разность 3 - 5 = -2 сумма 3 + 5 = 8
```

Каждый элемент списка представляется своим количеством символов `width`. Но для вывода элемента списка, например, на экран, можно указать минимальное число (`min_width`) символов для представления этого элемента

на экране, т.е. произвести так называемый форматный вывод. Если `min_width > width`, то перед элементом ставятся недостающие пробелы, а если `min_width < width`, то величина `min_width` игнорируется и элемент выводится целиком. Например, (справа дается вывод команды `WriteLn`):

```
a:= 123;           {3 символа}
```

```
b:= 34567890;      {8 символов}
```

```
s:= 'Строка';      {6 символов}
```

{Ниже указано минимальное количество символов для вывода элемента: min_width =7}

```
WriteLn(a:7);      □□□□123 {min_width > width, поэтому  
перед числом добавлено 4 пробела }
```

```
WriteLn(b:7);      34567890 {min_width < width, число  
выводится целиком}
```

```
WriteLn(s:7);      □Строка {min_width > width, поэтому  
перед строкой добавлен пробел}
```

Для вещественных чисел также можно указать количество выводимых цифр после запятой (`dc`). В представлении вещественного числа десятичная точка и знак минус перед числом также учитывается при подсчете величины `width` (2 символа). Таким образом, в общем случае количество символов в представлении вещественного числа: `width = <количество цифр целой части числа>+dc+2`. Сокращение длины дробной части приводит к округлению числа. Если `dc` не указывать, то после точки будет выведено 13 цифр дробной части числа. Например,

```
a:=12.345678901234567;           {18 символов}
```

Ниже указано минимальное количество символов для вывода элемента: `min_width =7`, но разное число `dc` цифр после точки.

```
WriteLn(a:7:2);      □□12.34 {min_width > width, dc=2 поэтому  
перед числом добавлено 2 пробела }
```

`WriteLn(a:7:4);` `12.3457 {min_width < width, dc=4, число`
выводится с 4 цифрами после точки}
`WriteLn(a:7:5);` `12.34568 {min_width < width, dc=5`
число выводится с пятью цифрами после точки}
`WriteLn(a:7);` `12.3456789012346 {min_width < width, dc не`
указано, поэтому после точки выводится 13 цифр}

Пример текста программы на Паскале (в фигурных скобках комментарии, зарезервированные слова выделены жирным шрифтом):

Program `example1;` {необязательный оператор *Program* с указанием названия программы *example1*}

{Комментарии не являются частью кода программы и игнорируются компилятором}

{Начинается область описания, описываются все идентификаторы, используемые в программе}

Var

`a, b, c, d: real;` {определяются переменные вещественного типа}

`n, m: integer;` {определяются переменные целого типа}

`symbol: char;` {определяется переменная для хранения символа}

`s: string;` {определяется переменная для хранения строки}

Const

`pi2=1.5707;` {константа, тип определяется значением после знака равенства}

`title='Первая программа на Паскале';` {строковая константа}

Begin {Начинается исполняемая часть}

`a:=2.5; b:=2; c:=1.2e-5;` {вещественным переменным присваиваются значения}

`d:=(a*b-c)*pi2;` {вычисляется выражения, стоящее после знака присваивания “:=”, результат вычисления присваивается переменной *d* }

```
n:=1;
m:=8;
m:=m+n; {аналогично с переменными целого типа}
symbol:='A'; {символьной переменной присваивается
значение 'A'}
s:= 'это строка символов из русского алфавита'; {строковой
переменной присваивается значение}
WriteLn(title);
WriteLn('d = ', d);
WriteLn('m = ', m);
Write(symbol, ' ');
Write(s);
End.
```

Задания

1.1. Ввести с клавиатуры вещественные коэффициенты многочлена и значение x . Вывести значение многочлена $Z = a + bx^2 + cx^3 + dx^4$

1.2. Вычислить выражения:

a) $10^{1.5} - 9^3 + 100$; b) $\sqrt{\frac{1 + \cos(\pi / 6)}{3 - 5 / 2}}$; c) $\sqrt{1 + \frac{\sqrt{\cos^2(\pi / 5) + \sin^2(\pi / 3)}}{2}}$;

d) $\operatorname{tg}(\pi / 3.5) + \sin(\pi / 7) \cdot \sqrt{3 - 2 \cdot 5} \cdot \ln(3)$; e) $\frac{1 - e^{-\cos(3)}}{1 + \sin^2(2.5^4)}$;

f) $2^{16/3} + \frac{\ln(|\operatorname{ctg}(2)|)}{1 - 2^{-4/5}}$

1.3. Перевод температуры по шкале Цельсия в температуру по шкале Фаренгейта по формуле $f = \frac{9}{5}c + 32$, где c – температура по шкале Цельсия, вводится с клавиатуры.

1.4. Написать программу, вычисляющую путь, пройденный телом с ускорением a за время t , с начальной скоростью v_0 (вводятся с клавиатуры).

Использовать формулу $s = v_0 t + \frac{at^2}{2}$.

1.5. Написать программу, вычисляющую время, через которое тело, брошенное вертикально вверх со скоростью v (вводится с клавиатуры), упадет на землю.

1.6. Формула Герона. Написать программу, вычисляющую площадь треугольника по его сторонам a , b , c (вводятся с клавиатуры):

$S = \sqrt{p(p-a)(p-b)(p-c)}$, где $p = (a+b+c)/2$ – полупериметр.

1.7. Ввести с клавиатуры целое четырехзначное число, вывести на экран сумму и произведение цифр этого числа, а также число с обратным порядком цифр, относительно исходного.

2. УСЛОВНЫЕ ОПЕРАТОРЫ

Прежде, чем приступить к рассмотрению условных операторов, рассмотрим некоторые вопросы, связанные с переменными логического типа и алгеброй логики.

2.1. Переменные логического типа

Логические переменные принимают два значения: `True` («Истина») или `False` («Ложь»). Описание типа таких переменных осуществляется с помощью ключевого слова `Boolean` («Булевский» от фамилии английского логика). Например,

```
Var a, b: Boolean;
```

Логическим переменным значение присваивается с помощью оператора присваивания «:=», где с правой стороны от этого оператора стоит логическое выражение (ЛВ) или значения `True`, `False`.

2.2. Логическое выражение

Логическим выражением (ЛВ) можно назвать выражение, о котором можно сказать, что оно или истинно, или ложно. Например,

```
a>2; {Если a больше 2, то значение True, иначе – False}  
b<3; {Если b меньше 3, то значение True, иначе – False}  
x=10; {Если x равно 10, то значение True, иначе – False}  
y>=5; {Если y больше или равно 5, то значение True, иначе  
– False}  
z<=6; {Если z меньше или равно 6, то значение True, иначе  
– False}
```

Здесь использованы символы сравнения:

«<» – меньше;

«>» – больше;

«<=» – меньше или равно;

«>=» – больше или равно;

«=» – равно;

«<>» – не равно.

Выше приведены примеры простых логических выражений. Из логических выражений можно создавать сложные логические выражения с помощью основных логических операций сложения, умножения, отрицания. Пусть имеются два логических выражения – ЛВ_1 и ЛВ_2. Рассмотрим три случая.

Логическое сложение OR. Мы хотим создать более сложное выражение ЛВ, которое истинно, если истинно хотя бы одно из условий ЛВ_1 и ЛВ_2. Значение ЛВ (значение логической переменной ЛП) в этом случае представляет собой результат логического сложения выражений ЛВ_1 и ЛВ_2:

ЛП:=(ЛВ_1) OR (ЛВ_2);

Каждое логическое выражение ЛВ_1 и ЛВ_2 обязательно заключается в скобки. OR – знак логического сложения (логическое «ИЛИ»).

Логическое умножение AND. Мы хотим создать сложное выражение ЛП, которое истинно, только если истинны оба выражения ЛВ_1 и ЛВ_2:

ЛВ:=(ЛВ_1) AND (ЛВ_2);

AND – знак логического умножения (логическое «И»).

Эти знаки можно сочетать. Например, если имеется три выражения ЛВ_1, ЛВ_2 и ЛВ_3, то значение логической переменной ЛП

ЛП:=((ЛВ_1) OR (ЛВ_2)) AND (ЛВ_3);

будет True, если истинно выражение ЛВ_3 и истинно хотя бы одно из выражений ЛВ_1 и ЛВ_2.

Отрицание NOT. Здесь все достаточно просто. Например,

a := not b;

Логическая переменная a будет иметь значение True, если логическая переменная b имеет значение False, и наоборот.

2.3. Условный оператор

Условный оператор проверяет некоторое условие (логическое выражение или значение логической переменной) и, в зависимости от результатов проверки, выполняет нужное действие (последовательность операторов).

Оператор If-Then-Else.

Полная форма:

```
If ЛВ Then  
    оператор_1  
Else  
    оператор_2;
```

Выполняется этот оператор следующим образом: сначала проверяется истинность логического выражения ЛВ (или логической переменной), если ЛВ истинно, то выполняется оператор_1, а затем управление переходит к оператору, стоящему после условного. Если же ЛВ ложно, то выполняется оператор_2, а затем управление переходит к оператору, стоящему после условного.

Сокращенная форма:

```
If ЛВ Then  
    оператор_1;
```

В этом операторе сначала проверяется истинность логического выражения ЛВ (или логической переменной), если ЛВ истинно, то выполняется оператор_1, а затем управление переходит к оператору, стоящему после условного.

Пример:

```
Var  
    a : Integer;  
Begin  
    readln(a);  
    If a mod 3 = 0 Then  
        a := a+4
```

```

Else
    a := a+1;
    writeln('a = ', a);
End.

```

В этом примере, целое числовое значение, введенное с клавиатуры, помещается в переменную **a**. Затем проверяется является ли значение **a** кратным 3 (делится на 3 без остатка). Если **a** кратно 3, то значение **a** будет увеличено на 4. Если **a** не кратно 3, то значение **a** будет увеличено на 1.

2.4. Составной оператор (операторные скобки **begin-end**)

В операторе

```

If ЛВ Then
    оператор_1
Else
    оператор_2;

```

после **Then** и после **Else** стоят по одному оператору – **оператор_1** и **оператор_2**, соответственно. Если же необходимо, чтобы после этих слов стоял не один оператор, а последовательность из какого-либо количества **N** и **M** операторов, то используют составной оператор, т.е. последовательность операторов, разделенных точками с запятой, заключают в операторные скобки **Begin-End**:

```

If ЛВ Then
    Begin
        оператор_1;
        оператор_2;
        оператор_3;
        ...
        оператор_N;
    End
Else

```



```
Begin
    оператор_else_1;
    оператор_else_1;
    оператор_else_1;
    ...
    оператор_else_M;
End;
```

Слово **Begin** можно рассматривать как открывающую скобку, а слово **End** – как закрывающую. Обратите внимание, что перед **Else**, после слова **End** точка с запятой не ставится.

2.5. Алгоритм

Рассмотрим следующее важное понятие в программировании.

Алгоритм – одно из фундаментальных понятий информатики, им обозначают точное, понятное и однозначное предписание инструкций, которые должны быть выполнены для решения поставленной задачи. Исполнителем инструкций в нашем случае является компьютер. Другими словами, алгоритм – это правила, сформулированные на некотором языке и определяющие процесс обработки исходных данных для получения искомого результата.

Отметим основные свойства, которыми должен обладать алгоритм:

- **дискретность** (прерывность, раздельность) – выполнение алгоритма разбивается на последовательность законченных действий – шагов. Каждый шаг должен быть закончен прежде, чем начнется исполнение следующего;
- **определенность** (детерминированность) – каждая инструкция должна быть четкой и однозначной. Выполнение алгоритма носит механический характер и не требует никаких дополнительных указаний или сведений о решаемой задаче;



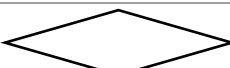


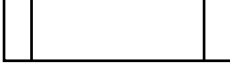


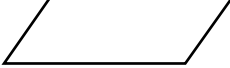

- **понятность** – инструкции должны быть даны на понятном для исполнителя языке и выполняться в строгом соответствии с их назначением;
- **массовость** – применимость для решения любой задачи из некоторого класса задач, различающихся лишь исходными данными;
- **конечность** – алгоритм должен быть результативным, приводить к решению задачи за конечное число шагов.

Среди различных способов представления алгоритмов графический способ является наиболее компактным и наглядным.

В табл. 2 приведены некоторые наиболее часто употребляемые символы.

Таблица 2

Некоторые основные символы, применяемы в блок-схемах

Название символа	Обозначение и пример заполнения	Пояснение
Ввод-вывод		Начало, конец алгоритма, вход и выход в подпрограмму
Процесс, серия		Вычислительное действие или последовательность действий
Решение, развилка		Проверка условий
Цикл		Начало цикла
		Конец цикла
Предопределенный процесс		Вычисления по подпрограмме, стандартной подпрограмме
Ввод-вывод		Ручной ввод данных
Документ		Вывод результатов на печать
Данные		Данные, носитель которых не определен
Соединитель		Выход и вход для различных частей схемы.

При графическом представлении алгоритм изображается в виде последовательности связанных между собой функциональных блоков, каждый из которых соответствует выполнению одного или нескольких действий. Такое графическое представление называется схемой алгоритма или блок-схемой.

В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями, определяющими очередность выполнения действий.

На рис. 1 представлен пример блок-схемы для расчета суммы натуральных нечетных чисел от 1 до 100.

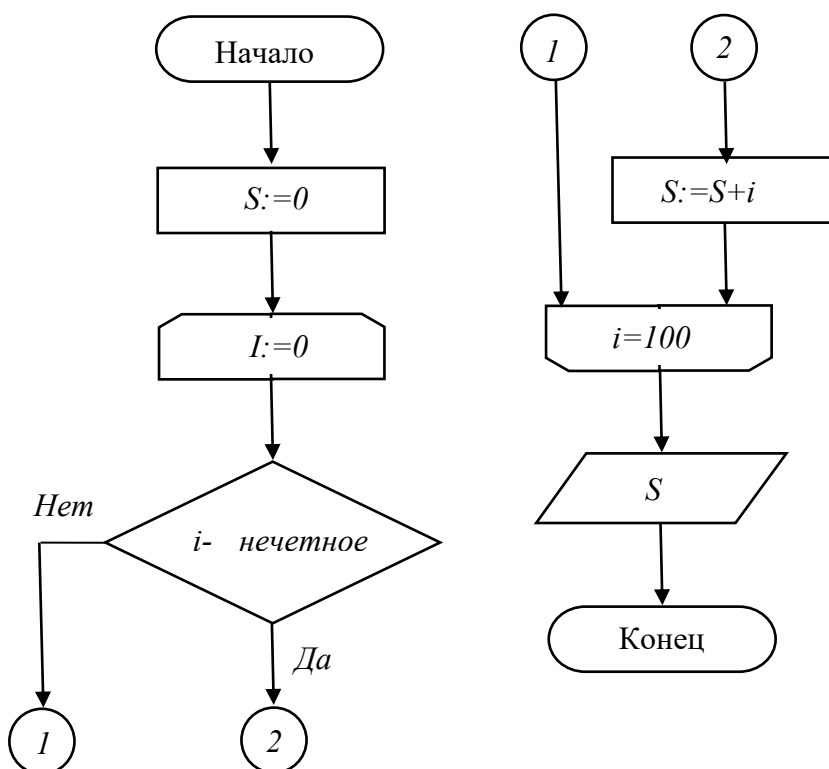


Рис. 1. Пример блок-схемы для расчета суммы натуральных нечетных чисел от 1 до 100

Задания

2.1. (обязательно) Решение квадратного уравнения $ax^2 + bx + c = 0$ при *любых* значениях коэффициентов a, b, c . Коэффициенты уравнения вводятся с клавиатуры, решения выводятся на экран. Нарисуйте блок-схему программы.

2.2. С клавиатуры вводится три целых числа a, b, c . На экран выводится наибольшее и наименьшее число из этих трех.

2.3. Вводятся координаты точки (x, y) . Вывести номер квадранта, в который попадает точка с такими координатами. Если одно из чисел равно нулю, то вывести на какой оси находится точка.

2.4. Вводится x , вычислить

$$y = \begin{cases} \frac{1}{\sqrt{x-2}}, & x > 2 \\ \sqrt{2-x}, & x \leq 2 \end{cases}$$

2.5. Вводятся координаты точки (x, y) , определить принадлежит ли эта точка области, ограниченной окружностью с радиусом $r = 5$ и центром в начале системы координат.

2.6. Ввести целое число и проверить делится ли оно на 2, 3 и 5 одновременно.

3. ОПЕРАТОР МНОГОВАРИАНТНОГО ВЫБОРА

Оператор выбора (переключатель) `Case-Of-Else-End` является условным оператором, он позволяет организовать выполнение программы в зависимости от значения ключа выбора (КВ). Этот оператор, как и другой условный оператор `If-Then-Else` имеет полную и укороченную формы записи.

Полная форма:

```
Case KB Of
    ЗН_1 : оператор_1 ;
        ЗН_2 : оператор_2 ;
        ЗН_3 : оператор_3 ;
        ...
Else
    оператор_по_умолчанию ;
End ;
```

Выполняется этот оператор следующим образом: анализируется значение ключа выбора КВ, если это значение совпадает с каким-нибудь из перечисленных значений ЗН_1, ЗН_2, ЗН_3, ..., то выполняется оператор, соответствующий этому значению, и после этого управление переходит к оператору, стоящему после переключателя (т.е. за «End;»). Если же значение КВ не совпадает ни с одним из перечисленных значений, то выполняется оператор_по_умолчанию, и управление передается к оператору, стоящему после переключателя.

Ключ выбора КВ – переменная порядкового типа, например `Integer` или `Char`. Таким образом, она не может быть переменной типов `Real`, `String`, `Array` и т.п.

Сокращенная форма:

```
Case KB Of
    ЗН_1 : оператор_1 ;
```

```
ЗН_2 : оператор_2 ;  
ЗН_3 : оператор_3 ;  
...  
End ;
```

В этой форме выполняется только оператор, соответствующий какому-нибудь из значений КВ: ЗН_1, ЗН_2, ЗН_3, ..., после чего управление передается оператору, стоящему после переключателя.

Напоминаем, что если в переключателе требуется выполнить не один оператор, а несколько, то эти операторы должны быть заключены в операторные скобки **Begin-End** (см. составной оператор).

Кроме того, возможны следующие реализации данного оператора.

```
Case КВ Of  
    ЗН_1, ЗН_2, ЗН_3: оператор_1;  
    ЗН_4 .. ЗН_5: оператор_2;  
    ЗН_6: Begin оператор_3; оператор_4; End;  
End;
```

В приведенном выше примере оператор_1 будет выполняться, если значение ключа выбора будет равно ЗН_1, или ЗН_2, или ЗН_3. В случае если переменная КВ принимает какое-либо значение из интервала от ЗН_4 до ЗН_5, тогда выполняется оператор_2. Последний вариант демонстрирует применение операторных скобок.

В переменных символьного типа можно хранить по одному символу. Символ может быть произвольным. Символ, присваиваемый соответствующей переменной, должен быть заключен в апострофы, например – '&'. Следует отличать символ цифры от самой цифры – символ всегда заключается в апострофы, например – '2'. С символами цифр нельзя производить арифметические операции. Пример описания и присваивания:

```
Var  
    a, b, c, d, f : Char;
```

Begin

a := '1' ;

b := '2' ;

c := '+' ;

d := 'd' ;

f := '=' ;

WriteLn(d,f,a,c,b);

{будет выведена последовательность символов d=1+2};

End.

Пример 1.

Светофор. Эта учебная программа считывает символ с клавиатуры и, если он совпадает с одним из символов 'g'; 'y'; 'r', то выводится соответствующая строка про цвет светофора. Если совпадений нет, то выводится строка «светофор не работает».

Var

a : Char ;

Begin

ReadLn(a);

Case a Of

'g' : WriteLn('горит зеленый свет') ;

'y' : WriteLn('горит желтый свет') ;

'r' : WriteLn('горит красный свет') ;

Else

WtiteLn('светофор не работает');

End;

WriteLn('программа закончила работу');

End.

Пример 2.

Учебная программа, считывающая число с клавиатуры и, в зависимости от его значения, выполняющая нужную последовательность операторов

```
Var
    a : Integer;
    b, c : Real;
Begin
    Read(a);
    b := 2.5;
    Case a Of
        1: b := b - a;
        2: b := b + a;
        3,4,5: Begin
            c:= a*(b-a);
            b:= c + b;
            End;
        End;
    WriteLn('b = ', b);
    WriteLn('Программа закончила работу')
End.
```

В последнем примере, если ввести 1, то выведется строка «b = 1.5»; если ввести 2, то выведется строка «b = 4.5»; если ввести 4, то выведется строка «b = -3.5».

Задания

3.1. Калькулятор. Используя оператор выбора, напишите программу, которая сначала принимает с клавиатуры два числа вещественного типа, а затем символ операции с этими числами и выводит на экран строку, в которой записана данная операция и результат ее действия. Калькулятор должен выполнять арифметические операции сложения, вычитания, умножения, деления и операцию возведения в степень. Предусмотреть вывод сообщений об ошибках деления на ноль и возведения в степень отрицательного числа. Например, сначала ввели числа 2 и 4, затем символ '^', на экран выводится: $2^4 = 16$.

3.2. Решить систему двух уравнений $\begin{cases} ax + by = c \\ dx + fy = g \end{cases}$ для коэффициентов a, b, c, d, f, g , вводимых с клавиатуры. Использовать для решения правило Крамера:

решение существует, если $q = af - db \neq 0$, в этом случае решение системы двух

линейных уравнений: $x = \frac{fc - bg}{q}$; $y = \frac{ag - dc}{q}$.

3.3. Определить является ли год, введенный с клавиатуры, високосным. Год считается високосным, если число этого года не кратно 100 и кратно 4 либо кратно 400.

4. ОПЕРАТОРЫ ЦИКЛА. СЧЕТНЫЕ ОПЕРАТОРЫ ЦИКЛА

Операторы цикла повторяют некоторую последовательность операторов, поэтому их еще называют операторами повторений. Однократное исполнение операторов в цикле еще называют итерацией цикла. В Паскале существует два типа операторов цикла: счетные и условные. В счетных операторах цикла (их два) количество повторений задается.

Рассмотрим операторы цикла со счетчиком.

4.1. Оператор с увеличивающимся счетчиком цикла

Оператор с увеличивающимся счетчиком цикла (СЦ) записывается следующим образом.

`For сц : = нз To кз Do оператор_f;`

здесь `сц` – переменная целого типа, `нз` – начальное значение (целое число или переменная целого типа) этой переменной, `кз` – конечное значение (целое число или переменная целого типа) этой переменной. Причем, $кз \geq нз$.

Оператор выполняется следующим образом: сначала `оператор_f` выполняется для значения счетчика равного `нз`: `сц=нз`. Затем значение счетчика `сц` увеличивается на 1 и `оператор_f` выполняется уже для нового значения `сц=нз+1`. Затем значение `сц` снова увеличивается на 1 и `оператор_f` выполняется уже для этого значения счетчика цикла. Повторения продолжаются до тех пор, пока значение `сц` не станет больше конечного значения: `сц>кз`. Как можно видеть, количество итераций в таком цикле равно $кз - нз + 1$.

Пример.

Найти сумму ста первых чисел натурального ряда.

`Var`

`s, k : Integer ;`

`Begin`

`s := 0 ;`

```
For k:=1 To 100 Do
    s := s + k ;
Writeln('сумма = ', s)
End.
```

4.2. Оператор с уменьшающимся счетчиком цикла

```
For сц : = нз DownTo кз Do оператор_f;
```

Здесь сц – переменная целого типа, нз – начальное значение (целое число или переменная целого типа) этой переменной, кз – конечное значение (целое число или переменная целого типа) этой переменной. Причем, $кз \leq нз$.

Отличие действия этого оператора от предыдущего состоит в том, что значение счетчика цикла сц уменьшается на 1 после каждой итерации. Число итераций в этом операторе циклов определяется так же, как и в предыдущем.

Напоминаем, что в случае, если требуется повторения не одного оператора оператор_f, а последовательности операторов, то из этой последовательности нужно образовать составной оператор с помощью операторных скобок **Begin – End**.

Для более удобной работы с циклами **For-To-Do** и **For-DownTo-Do** предусмотрены две процедуры:

Break – немедленно прекращает работу цикла и передает управление операторам, стоящим после оператора цикла;

Continue – пропускает данную итерацию и возвращает к заголовку цикла (счетчик меняется на единицу и оператор выполняется для нового значения счетчика).

Задания

4.1. Дано целое число n . Вычислить значение факториала $n!$

4.2. Дано натуральное число n . Вычислить сумму ряда:

а) $\frac{1}{1^5} + \frac{1}{2^5} + \frac{1}{3^5} + \dots + \frac{1}{n^5}$; б) $\frac{1}{3^2} + \frac{1}{5^2} + \dots + \frac{1}{(2n+1)^2}$;

в) $-\frac{1}{3} + \frac{1}{5} - \dots + \frac{(-1)^n}{2n+1}$; г) $\frac{1}{1 \cdot 2} - \frac{1}{2 \cdot 3} + \dots + \frac{(-1)^{n+1}}{n \cdot (n+1)}$.

4.3. Нахождение значения определенного интеграла от функции $f(x)$ на интервале (a, b) с помощью метода трапеций осуществляется по формуле

$$\int_a^b f(x) dx = \left(\sum_{k=1}^{N-1} f(x_k) + \frac{f(a) + f(b)}{2} \right) \cdot h$$

Весь интервал (a, b) поделили на N отрезков длиной $h = \frac{b-a}{N}$. Концы этих отрезков внутри интервала (a, b) определяются формулой $x_k = a + k \cdot h$, $k = 1, 2, 3, \dots, N-1$. Используя оператор цикла, найдите для $N=100$ значение интегралов

а) $\int_{-1}^2 \frac{x}{(x^2+1)^2} dx$; б) $\int_0^{1/2} 4 \cos^2 x dx$; в) $\int_0^{\pi/3} \frac{1}{\cos^2 x} dx$; г) $\int_4^9 \frac{x+1}{\sqrt{x}} dx$.

4.4. Вычислить определенный интеграл

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

по правилу трапеций

$$J_t = (y_0 + 2y_1 + \dots + 2y_{n-1} + y_n) \frac{h}{2},$$

и по правилу Симпсона

$$J_s \approx \frac{h}{3} (y_0 + 4y_1 + 2y_2 + 4y_3 + 2y_4 + \dots + 2y_{n-2} + 4y_{n-1} + y_n).$$

Здесь $a=0$, $b=1$, $h = \frac{(b-a)}{n}$, $y_i = \exp(a + i \cdot h)$, $i = 0, 1, 2, \dots, n$, $n=100$.

Вывести расчетное значение, точное значение (оно равно $e - 1.0$) и относительную погрешность расчета. Сравните полученные результаты для различных методов.

4.5. Билеты имеют шестизначные номера. В рулоне у первого билета номер 304561, а у последнего – 306259. Найти количество «счастливых» билетов в рулоне и общее число билетов в рулоне. Билет считается «счастливым», если сумма первых трех цифр номера равна сумме оставшихся цифр номера.

4.6. Вычислить цепную дробь

$$1 + \frac{1}{2 + \frac{1}{3 + \frac{1}{\dots \frac{1}{99 + \frac{1}{100}}}}}$$

5. ОПЕРАТОРЫ ЦИКЛА. УСЛОВНЫЕ ЦИКЛЫ

В условных операторах цикла (их два) итерации производятся до тех пор, пока заданное логическое выражение имеет значение «истина», либо до тех пор, когда заданное логическое выражение примет значение «истина».

5.1. Оператор цикла с предусловием

While ЛВ Do оператор_w;

Здесь ЛВ – логическое выражение (условие), которое должно иметь значение «истина», чтобы выполнялся оператор_w. Оператор выполняется следующим образом: сначала проверяется истинность ЛВ, и если условие выполняется, то выполняется оператор_w. Это оператор произносят так: «пока справедливо ЛВ выполняй оператор_w».

Пример

Найти сумму ряда $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots$ с точностью $\varepsilon = 0,0001$, т.е. суммирование прекращается, если очередной элемент ряда по модулю меньше ε .

Var

S: real;

k, f : Integer ;

Begin

s := 1;

k := 2;

f := -1;

While (1/k) >= 0.0001 Do

Begin

s := s + f/k;

k := k + 1;

f := -f;

End;

```
WriteLn('сумма = ', s)
End.
```

5.2. Оператор цикла с постусловием

```
Repeat
    оператор_1;
    оператор_2;
    ...
Until ЛВ;
```

Когда логическое выражение ЛВ (условие) принимает значение «истина», тогда оператор цикла прекращает работу. Оператор выполняется следующим образом: сначала выполняются операторы: оператор_1, оператор_2 и другие, стоящие до слова **Until** (Пока), затем проверяется истинность условия ЛВ, если условие не выполняется, то снова повторяются последовательность оператор_1; оператор_2;.... Итерации повторяются до тех пор, пока ЛВ не примет значение «истина». Этот оператор произносят так: «повторяй оператор_1; оператор_2;... пока не выполнится условие ЛВ».

Пример

Составить программу нахождения наибольшего общего делителя произвольных двух натуральных чисел с помощью алгоритма Евклида.

```
Var
    a, b, c: word;
Begin
    ReadLn(a, b);
    Repeat
        If a < b then {меняем значения переменных
местами}
            Begin
                a:= a + b ;
                b:=a - b;
```

```

    a:=a - b;
    end;
    a:= a mod b; {вычисляем остаток двух чисел}
Until a = 0;
WriteLn('НОД = ', b)
End.

```

Как можно видеть, условные операторы могут выполняться бесконечное число раз («зацикливаться»), поэтому нужно всегда помнить о том, чтобы среди повторяющихся операторов был такой, который бы обеспечивал выполнение ЛВ - условия выхода из цикла.

Для более удобной работы с циклами **While-Do** и **Repeat-Until** предусмотрены две процедуры:

Break – немедленно прекращает работу цикла и передает управление операторам, стоящим после оператора цикла;

Continue – пропускает данную итерацию и возвращает к заголовку цикла.

5.3. Рекуррентные формулы

Рекуррентная формула выражает n -й член последовательности через номер этого члена и k предыдущих членов этой же последовательности

$$a_n = f(n, a_{n-1}, \dots, a_{n-k}).$$

При использовании рекуррентной формулы нужно сначала присвоить значения одному или нескольким членам последовательности, а затем с помощью рекуррентной формулы выразить все другие члены последовательности через члены, вычисленные ранее, и через номер вычисляемого элемента последовательности. Обычно начальные значения присваивают первым членам последовательности.

Примеры рекуррентных формул

1. Сумма S_N первых N членов последовательности a_k ($k = 1, 2, \dots, N$).

$$S_k = \begin{cases} S_0 = 0; \\ S_k = S_{k-1} + a_k, \quad k > 0 \end{cases}.$$

Здесь рекуррентная формула определяет член последовательности сумм

вида $S_k = \sum_{n=1}^k a_n.$

2. Факториал $k!$.

$$f_k = \begin{cases} f_0 = 1; \\ f_k = f_{k-1} \cdot k, \quad k > 0 \end{cases}$$

3. Числа Фибоначчи.

$$F_k = \begin{cases} F_1 = 1; \\ F_2 = 1; \\ F_k = F_{k-1} + F_{k-2}, \quad k > 2 \end{cases}.$$

4. Часто вычисление последующих членов ряда удобно производить, используя вычисленные предыдущие члены ряда. Например, для ряда

$$\frac{x^1}{1!} + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots + \frac{x^n}{n!}$$

произвольный k -й член этого ряда может быть выражен

через предыдущий $a_k = \frac{x^k}{k!} = a_{k-1} \cdot \frac{x}{k}.$

5. Итерационная формула для нахождения приближенного значения квадратного корня $x_n \approx \sqrt{a}$ из положительного числа a (Герон).

$$x_n = \begin{cases} x_0 = a, \quad n = 0; \\ x_n = \frac{1}{2} \left(x_{n-1} + \frac{a}{x_{n-1}} \right), \quad n > 0 \end{cases}.$$

6. Нахождение приближенного значения корня уравнения $x = f(x)$ вблизи начального значения x_0 (корней может быть несколько, поэтому требуется указать, где искать корень). Корень $x_n = f(x_{n-1}), n > 0$ считается вычисленным с точностью ε , если $|x_n - x_{n-1}| \leq \varepsilon.$

Задания

5.1. Найти сумму ряда $1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots$ с точностью $\varepsilon = 0,0001$, т.е. суммирование прекращается, если очередной элемент ряда по модулю меньше ε .

5.2. Составьте программу для вычисления следующей суммы

$$S = \sum_n (-1)^{n+1} \left(\frac{1}{n^2} \right)$$

с учетом слагаемых, которые больше 10^{-6} .

Определите $\sqrt{12S}$.

5.3. Ввести с клавиатуры целое положительное число N , вывести первое число Фибоначчи, большее N , а также номер этого числа.

5.4. Вычислить сумму первых чисел Фибоначчи, которые не превосходят 1000.

5.5. Вычислить число π с точностью до третьего знака после запятой, через вычисление суммы $\pi = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \dots \right) = 4 \sum_{k=1}^{\infty} \frac{(-1)^{k+1}}{2k-1}$.

5.6. Вычислить приближенное значение $\cos(x)$, где x вводится с клавиатуры ($|x| < 1$, аргумент в радианах), через разложение в ряд:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \dots = \sum_{k=0}^{\infty} \frac{(-1)^k x^{2k}}{(2k)!}.$$

Точность вычисления $\varepsilon = 0,001$.

5.7. Вычислить приближенные значения квадратного корня положительного числа (вводится с клавиатуры), используя формулу Герона. Точность вычисления $\varepsilon = 0,0001$.

5.8. Вычислить приближенное значение корня уравнения $x = \frac{1}{4} \cos(x)$ вблизи нуля.

5.9. Для положительных чисел a и b (вводятся с клавиатуры) определены

две последовательности $\left\{ \begin{array}{l} x_1 = a; \\ x_k = \frac{1}{2}(x_{k-1} + y_{k-1}); k > 1 \\ y_1 = b; \\ y_k = \sqrt{x_{k-1}y_{k-1}}, k > 1. \end{array} \right.$. Начиная с какого значения

x_n удовлетворяется условие $|x_n - y_n| < \varepsilon$. Точность $\varepsilon = 0,001$.

5.10. Напишите программу вывода на экран последовательности целых чисел, отличающихся на единицу, в виде «лестницы»

```

98
  99
    100
      101
        102

```

Необходимо учесть изменение числа пробелов при изменении количества цифр в числе. Начальное число и количество чисел вводится с клавиатуры.

6. МАССИВЫ

Массив – это пронумерованная последовательность однотипных элементов: целых чисел, вещественных чисел, символов и т.п. Например, последовательность целых чисел $m=\{2, 5, 1, 10, 3, 2\}$ – массив целых чисел с именем m , состоящий из 6 элементов. Первому элементу присваивается номер 1, второму – 2, третьему – 3 и т.д. Запись $m[3]$ означает третий элемент в массиве m , значение которого равно числу 1.

Массивы описываются в декларативной части программы

```
Var    m: Array [1..6] Of Integer;  
      c, b : Array [1..4] Of Real;
```

Здесь определен массив с именем m из 6 элементов – каждый является целым числом и два массива c и b из 4 элементов, каждый элемент является вещественным числом. Первое число в квадратных скобках – номер первого элемента, потом две точки, последнее число – номер последнего элемента.

Примеры:

```
x:=m[2];  
y:= m[3]*m[1];  
m[4]:= x;  
m[5]:=6;
```

Переменной x присваивается значение элемента с номером 2 из массива m . Переменной y присваивается значение, равное произведению элементов массива m с номерами 3 и 1. Элементу с номером 4 из массива m присваивается значение переменной x . Элементу с номером 5 из массива m присваивается значение 6.

В Паскале для получения случайных чисел используется функция `Random`. Функция `Random` при каждом вызове возвращает случайное вещественное число от 0.0 до 1.0. Для того, чтобы при запуске программы последовательности псевдослучайных чисел не повторялись, необходимо

предварительно вызвать команду `Randomize`, которая включает генератор случайных чисел.

Вариант функции `Random` – функция `Random(k)`, где `k` – целое положительное число. Эта функция при каждом вызове возвращает случайное целое число от 0 до `k-1`, т.е. `k` указывает количество целых чисел, из которых (включая 0) случайным образом выбирается число. Примеры:

```
Var m:Array [1..9] Of Integer;  
d: Array [1..5] Of Real; i: Integer;  
Begin  
    Randomize;  
    For i:=1 To 9 Do m[i]:=Random(100);  
    For i:=1 To 5 Do d[i]:=Random;
```

В этом примере после выполнения операторов каждому элементу массива `m` будет присвоено случайное целое число от 0 до 99, а каждому элементу массива `d` будет присвоено вещественное число от 0.0 до 1.0.

Задания

6.1. Сгенерировать массив m из 10 элементов. Каждый элемент – случайное целое число от 0 до 20. Сгенерированный массив вывести на экран в виде таблицы из двух строк по пять элементов в каждой.

- а) найти сумму всех элементов;
- б) вывести на экран номера тех элементов, значения которых четны;
- в) найти произведение всех элементов;
- г) найти сумму $a[1] * a[2] + a[2] * a[3] + a[3] * a[4] + \dots a[9] * a[10]$;
- д) вывести на экран $a[1]$, $a[1] * a[2]$, $a[1] * a[2] * a[3]$, ..., $a[1] * a[2] \dots a[10]$;
- е) вывести на экран номера и значения максимального и минимального элементов массива;

ж) образовать новый массив z из 10 элементов случайных чисел от 0 до 20, вывести его на экран в таблице из 2 строк, и сложить поэлементно с массивом m . Вывести массив – результат сложения на экран в таблице из двух строк.

6.2. Сгенерировать массив x из 20 элементов случайных вещественных чисел от 0 до 50, вывести этот массив в виде таблицы из четырех строк по пять элементов в каждой. Из элементов этого массива сформировать новый массив a по следующему правилу:

$$a_1 = x_1; a_{20} = x_{20};$$

$$a_k = (x_{k-1} + x_k + x_{k+1}) / 3, \quad k = 2, 3, \dots, 19$$

и вывести его на экран в таблице из четырех строк.

Из этого массива a сформировать еще один массив b , элементы которого циклически сдвинуты на 2 позиции

$$b_{19} = a_1; b_{20} = a_2;$$

$$b_k = a_{k+2}, \quad k = 1, 2, \dots, 18$$

и вывести его на экран в таблице из четырех строк.

Вывести на экран результат поэлементного вычитания массива b из массива a .

6.3. Сформировать десятиэлементный массив целых чисел с помощью генератора псевдослучайных чисел в диапазоне значений от 1 до 100.

– произвести сортировку полученного массива методом простого выбора, суть которого заключается в следующем. Находится минимальный элемент и ставится на первое место. Элемент, который находился на первом месте ставится на место минимального. Затем в последовательности, исключая первый элемент, ищется вновь минимальный и ставится на второе место и так далее, пока не встанет на свое место предпоследний элемент, сдвинув максимальный на последнее место массива;

– произвести сортировку полученного массива методом простого обмена. Последовательно просматриваются элементы массива a_1, \dots, a_n . Если $a_i > a_{i+1}$, то они меняются местами и возобновляется просмотр с элемента a_{i+1} и т.д. Таким образом, наибольшее число передвигается на последнее место. Следующий просмотр начинается сначала, уменьшая на единицу количество просматриваемых элементов. Массив будет упорядочен после просмотра, в котором не был переставлен ни один элемент.

7. СИМВОЛЫ. СТРОКИ

Для описания символов используется зарезервированное слово `Char`. Для символов определены операции сравнения: `>`; `<`; `>=`; `<=`; `=`. Сравнение происходит по коду символа в таблице ASCII-кодов. Получить код символа в таблице кодировки можно с помощью функции `Ord(c)`, которая возвращает целое число – код символа `c` в таблице кодов ASCII.

Обратная функция – `Chr(a)`, возвращает символ, соответствующий коду в таблице кодов ASCII равному целому числу `a`.

Строка – одномерный массив, состоящий из символов: букв, цифр, пробела, подчеркивания, знаков препинания, скобок и т.п. Доступ к символу строки осуществляется также как в обычном массиве – указанием номера символа в строке. Описание переменных, имеющих тип строки, в Паскале осуществляется следующим образом:

```
Var
```

```
s1, s2, s3, s4: String;
```

Значение строкового типа представляет собой последовательность символов, заключенную в одиночные апострофы «'». Примеры:

```
s1 := 'Это пример строки. This is a string';
```

```
s2 := '1+2=3. Это строка. Числа в строке представляют собой  
символы';
```

```
WriteLn(s1[3]); {Будет выведен третий символ строки s1 'о'}
```

Для строк определена операция соединения строк (конкатенация). При этой операции порядок соединяемых строк важен. Например,

```
s1:='Соеди';
```

```
s2:='нение';
```

```
s3:=s1+s2; {s3='Соединение'}
```

```
s4:=s2+s1; {s4='нениеСоеди'}
```


Функция `Length(s)` возвращает целое число, равное числу символов в строке `s`, т.е. длину строки. Например, для переменной `s1` из предыдущего примера, команда

```
WriteLn('длина строки s1 = ', Length(s1))
```

выведет на экран число 5.

Для работы со строками, содержащими числа, удобны функции преобразования строки в число, и обратные функции преобразования числа в строку.

Преобразование строки в число

`StrToInt(s1)` – преобразует строку `s1` в целое число;

`StrToFloat(s2)` – преобразует строку `s2` в вещественное число.

Преобразование числа в строку

`IntToStr(a)` – преобразует целое число `a` в строку;

`FloatToStr(b)` – преобразует вещественное число `b` в строку.

Задания

7.1. Обращение строки. Введите строку символов и выведите на экран эту строку в обратной последовательности.

7.2. Шифровка 1. Введите с клавиатуры строку символов и в каждой паре соседних символов этой строки поменяйте символы местами. Выведите зашифрованную строку на экран. Примените этот же способ для восстановления строки.

7.3. Шифровка 2. Введите строку символов, преобразуйте символы в коды таблицы кодировки и выведите получившуюся строку на экран. Раскодируйте получившуюся строку.

7.4. Используя коды чисел, решите задачу о проверке «счастливого» билета. Номер билета вводится с клавиатуры и состоит из четного числа цифр (число цифр неизвестно), если сумма цифр первой половины номера равна сумме оставшихся цифр номера, то этот билет считается «счастливым».

7.5. Напишите программу, которая производит подсчет количества каждого символа, встречающегося во вновь созданной строке.

Результат вывода на экран должен иметь следующий вид.

aasdfgfasgas

a – 4

s – 3

d – 1

f – 2

g – 2

Сначала выводится исходная строка, затем результат подсчета для каждого символа только один раз.

8. ФУНКЦИИ И ПРОЦЕДУРЫ

Повторяющиеся в программе участки кода можно оформить в виде функции или процедуры, а затем в тексте главной программы вызывать эти функции и процедуры с нужными параметрами. Также можно использовать функции, в которых вычисляются громоздкие выражения. Это делается для того, чтобы не усложнять код главной программы, делая его более понятным.

Например, в стандартном Паскале нет функции тангенса, и каждый раз, когда в программируемом выражении встречается тангенс нужно для него писать примерно следующее:

```
x:=pi;  
a:=sin(x)/cos(x); {вычисленное значение тангенса  $\pi$   
присваивается переменной a}  
b:=sin(x/4)/cos(x/4); {вычисленное значение тангенса  $\pi/4$   
присваивается переменной b}
```

Было бы проще определить заранее функцию $\tan(x)$ и вызывать ее с разными аргументами:

```
x:=pi;  
a:=tan(x); {вычисленное значение тангенса  $\pi$  присваивается  
переменной a}  
b:=tan(x/4); {вычисленное значение тангенса  $\pi/4$   
присваивается переменной b}
```

В Паскале определение функции размещается в области описания программы. Функция оформляется следующим образом. Сначала идет заголовок функции, затем, если необходимо – область описания переменных (Var, Const и т.п.), используемых в функции, затем, начиная со слова Begin, идет код функции, который заканчивается присваиванием значения имени функции, в конце функции ставится End; (End и точка с запятой). Переменные, определенные в области описания переменных функции, не видны за пределами функции.

Функции, определенные в тексте выше, могут использоваться в функциях, описываемых ниже.

Заголовок функции записывается следующим образом:

```
Function <имя функции>(v1,v2: <тип переменных>): <тип  
возвращаемого значения функции>;
```

Пример определения и использования функций (учебный код):

```
Function tan(x:Real):Real;           {определяем функцию  
тангенса - tan}
```

```
Begin
```

```
tan:=sin(x)/cos(x);
```

```
End;
```

```
Function poly2(x,a,b,c:Real): Real; {определяем функцию  
 $poly2 = ax^2 + bx + c$ }
```

```
Begin
```

```
poly2:=x*(a*x+b)+c;
```

```
End;
```

```
Function sum(n:Integer):Real; {определяем функцию,  
вычисляющую сумму первых n натуральных чисел}
```

```
Var
```

```
k:Integer;
```

```
s:Real;
```

```
Begin
```

```
s:=0;
```

```
For k:=1 To n Do
```

```
s:=s+k;
```

```
sum:=s;
```

```
End;
```

```

Function kosinus(x:Real):Real; {определяем собственную
функцию косинуса, вычисляемую через тангенс, используя
определенную выше функцию tan}
Begin
    kosinus:=1/sqrt(1+sqr(tan(x)));
End;
Var {Здесь описываются переменные, используемые в главной
программе}
    a, b, c, x, y, z:Real;
    n:Integer;
Begin                                {Начало главной программы}
    x:=pi; y:=pi/4;
    Writeln(tan(x-y));                {Выводится на экран тангенс
3π/4}
    a:=1; b:=-5; c:=2;
    Writeln(poly2(x,a,b,c,));        {Выводится на экран значение
выражения  $\pi^2 - 5\pi + 2$  }
    n:=150;
    Writeln(sum(n)) {Выводится на экран значение суммы чисел
от 1 до 150}
    z:=kosinus(y); {Переменной z присваивается значение
cos(π / 4) }
    Writeln(z);                      {Выводится на экран значение
переменной z}
End. {Конец главной программы }

```

Задания

8.1. Создайте функцию, которой в качестве аргумента передается целое число, а функция возвращает значение факториала от этого числа. В функции сделайте проверку, что аргумент положительный.

8.2. В Паскале нет функции возведения в степень. Создайте функцию, которой в качестве аргументов передается число и степень, в которую надо возвести число. Используйте, имеющиеся функции $\text{Exp}(x)$ и $\text{Ln}(x) - e^x$ и $\ln(x)$, соответственно, и тождество $a^b = e^{(b \cdot \ln(a))}$. Число a должно быть больше нуля.

8.3. Создайте функцию, которой в качестве аргумента передается целое число n , а функция вычисляет и выводит на экран первые n чисел Фибоначчи: $x_0 = 1; x_1 = 1; x_k = x_{k-1} + x_{k-2}; k = 2, 3, \dots, n$.

8.4. Создайте функцию, которой передается вещественное число x , и которая возвращает значение функции: $f(x) = \frac{x^5 + 2.5 * x^4 + \cos(x) * x^3 - 1.2 * x}{(1 + x + x^2)^2 + 1}$

8.5. Создайте функцию для вычисления площади треугольника по длинам двух катетов a и b и угла α между ними по формуле: $S = \frac{1}{2} a * b * \sin(\alpha)$. Угол передается в функцию в градусах.

8.6. Создайте функцию, которой передается вещественное число x , и которая в зависимости от значения x возвращает следующие значения:

$$f(x) = \begin{cases} 1, & \text{если } |x| \leq 1; \\ 0, & 1 < |x| \leq 2; \\ 2x, & 2 < |x| \leq 10; \\ x^2, & 10 < |x| \end{cases}$$

9. ПРОЦЕДУРЫ

Процедуры в Паскале используются для тех же целей, что и функции, т.е. в них реализован определенный алгоритм вычисления некоторой величины, с использованием передаваемых в функцию или процедуру значений аргументов. Затем эта процедура или функция могут многократно вызываться из главной программы для разных значений их аргументов. В стандартном Паскале отличие функции от процедуры состоит в том, что результат вычисления функции присваивается имени функции, поэтому функция при ее вызове возвращает только одно число или символ, в зависимости от ее типа. Например, определенная пользователем функция $\text{Tan}(X)$, возвращает значение тангенса угла X . Если же требуется вычислить массив значений, или несколько значений, то для этого используется процедуры. Например, нахождение корней квадратных уравнений $ax^2 + bx + c = 0$ можно реализовать в виде процедуры: передавать этой процедуре коэффициенты a , b , c , а процедура при вызове будет возвращать корни уравнения x_1 , x_2 . Корней уравнения два, поэтому в общем случае использовать функцию для этих целей будет затруднительно и нужно использовать процедуру. Или, другой пример, процедуре передается неупорядоченный массив целых чисел, а процедура возвращает упорядоченный массив. Процедуры описываются подобно функциям, описание процедуры находится в области описания главной программы.

Нужно отметить, что в общем случае процедура может и не возвращать никаких значений. Примером такой процедуры может быть последовательность операторов для вывода информации на экран или в файл.

Описание процедуры начинается с заголовка процедуры, затем следует область описания переменных и констант, используемых в теле процедуры, затем, начиная со слова **Begin**, идет последовательность исполняемых операторов тела процедуры, которое заканчивается словом **End**; (**End** с точкой с запятой).

Заголовок процедуры начинается со слова `Procedure`, затем следует имя процедуры, затем в круглых скобках описываются передаваемые в процедуру параметры, при этом, переменная, которая будет возвращаться при вызове процедуры, должна быть описана со словом `Var`. Заканчивается заголовок точкой с запятой. Пример заголовка процедуры, в которой по коэффициентам квадратного уравнения вычисляются его корни:

```
Procedure roots (a, b, c : Real; Var x1, x2 : Real);
```

В программу будут возвращены корни уравнения `x1` и `x2`, поэтому они описаны со словом `Var`. Пример вызова этой процедуры и использование ее результатов представлен ниже:

```
Procedure roots (a, b, c : Real; Var x1, x2 : Real);
```

```
Var
```

```
    d: Real;
```

```
Begin
```

```
    {выполняемые операторы, с помощью которых находятся корни}
```

```
    x1:=...;
```

```
    x2:=...;
```

```
End;
```

```
Var
```

```
k1,k2,k3, y1,y2: Real;  {Описание переменных, используемых в  
главной программе}
```

```
Begin
```

```
    k1:=1; k2:= 3; k3:=2.5;
```

```
    roots(k1,k2,k3,y1,y2);    {решается уравнение для  
коэффициентов k1,k2,k3, корни будут в переменных y1,y2}
```

```
    WriteLn('первый корень= ', y1); {Выводим на экран значения  
корней}
```

```
    WriteLn('второй корень= ', y2);
```

```
End.
```


Пользовательский тип данных. Если в процедуру или функцию необходимо передать массив, то предварительно следует определить новый тип данных. Так, для массива из трех элементов можно ввести новый тип с названием **Vector** и описывать переменные с использованием этого типа. Описание нового типа начинается со слова **Type**. Пример приведен ниже:

```
Type Vector = Array [1..3] Of Real;
Function scalar(a,b: Vector): Real;    {определяется
функция, которой в качестве аргументов передаются два массива
a и b, по три элемента в каждом}
Begin
...
End;
Var v1,v2: Vector; s:Real;    {Описываются переменные,
используемые в главной программе}
Begin
...
End.
```

Задания

9.1. Составить процедуру для нахождения решений квадратного уравнения по коэффициентам уравнения. Используйте код ранее решенной задачи для нахождения корней. Продемонстрировать работу процедуры.

9.2. Составить процедуру для вычисления суммы двух векторов: компоненты нового вектора вычисляются по формуле $c[i] = a[i] + b[i]$.

9.3. Составить функцию для вычисления длины вектора a по формуле:

$$s = \sqrt{\sum_{i=1}^3 (a[i])^2}$$

9.4. Составить функцию для вычисления скалярного произведения двух векторов a и b по формуле: $s = \sum_{i=1}^3 a[i] * b[i]$

9.5. Составить функцию для вычисления угла между векторами a и b по формуле:

$$\varphi = \arccos \left(\frac{(\vec{a} \cdot \vec{b})}{|\vec{a}| \cdot |\vec{b}|} \right)$$

9.6. Составить процедуру, которая получает строку символов, а возвращает строку, в которой с обратной последовательностью символов (зеркально отраженную)

10. ФУНКЦИИ И ПРОЦЕДУРЫ. ПРОЦЕДУРНЫЙ ТИП ДАННЫХ

Часто, при решении вычислительных задач, один и тот же алгоритм применяется для различных функций. Например, алгоритм нахождения определенного интеграла от гладкой функции не зависит от вида функции, алгоритм нахождения нуля гладкой функции также не зависит от вида функции и т.п. Поэтому очень удобно оформить алгоритм решения вычислительной задачи в виде отдельной функции или процедуры и передавать имя той функции, для которой нужно реализовать вычислительный алгоритм, как фактический параметр процедуры.

В современных вариантах Паскаля такая возможность реализуется через использование процедурного типа данных. Рассмотрим пример использования процедурного типа данных. Предположим, требуется вычислить среднее значение для произвольной гладкой функции $f(x)$ на отрезке $[a, b]$. Алгоритм вычисления следующий:

$$\bar{f} = \frac{\sum_{k=0}^N f(x_k)}{N+1},$$

где $x_k = a + k\Delta x$, $k = 0, 1, 2, \dots, N$, $\Delta x = (b - a) / N$, а $N+1$ – количество точек, в которых вычисляется функция. Нужно вычислить средние значения функций $f_1(x) = x^2$; $f_2(x) = \sin(x)$; $f_3(x) = x \cos(x)$ на отрезке $[0, \pi]$.

Ниже приведен возможный текст программы.

{определяем процедурный тип с именем func как вещественную функцию от одного вещественного}

Type

Func = Function (x:Real):Real;

{определяем функцию вычисления среднего значения функции f на отрезке [a, b] по значениям некоторой функции f в n точках}

Function middlfunc(a,b:Real; n:Integer; f:Func): Real;

```

Var
    x,s,d: Real; k:Integer;
Begin
    d:=(b-a)/n;
    s:=0;
    For k:=0 To n Do
        Begin
            x:=a+k*d
            s:=s+f(x);
        End;
    middlfunc:=s/(n+1);
End;

```

```

{определяем функцию f1}
Function f1(x:Real):Real;
Begin
    f1:=x*x;
End;

```

```

{определяем функцию f2}
Function f2(x:Real):Real;
Begin
    f2:=sin(x);
End;

```

```

{определяем функцию f3}
Function f3(x:Real):Real;
Begin
    f3:=x*cos(x);

```

```

End;
  {Описание переменных и констант, используемых в главной
программе}
Var
mf1, mf2, mf3:Real;
Const
a=0; b=pi; n=100;
Begin
  {Вызываем функцию вычисления среднего значения для
функций f1, f2, f3}
  mf1:=mddlfunc(a,b,n,f1);
  mf2:=mddlfunc(a,b,n,f2);
  mf3:=mddlfunc(a,b,n,f3);
  WriteLn(mf1, ' ',mf2, ' ', mf3)
End.

```

Имена стандартных функций Паскаля (sqr, cos, sin, ln и т.д) не могут передаваться в качестве параметра, поэтому в примере для вычисления квадрата x и $\sin(x)$ были созданы функции f1 и f2.

Задания

10.1. Использовать алгоритм вычисления определенного интеграла методом трапеций для функции $f(x)$ на отрезке $[a, b]$, поделенном на N отрезков:

$$\int_a^b f(x) dx = \left(\sum_{k=1}^{N-1} f(x_k) + (f(a) + f(b)) / 2 \right) \cdot h, \quad \text{где} \quad h = (b - a) / N,$$

$$x_k = a + k\Delta x, \quad k = 1, 2, \dots, N - 1.$$

Задавая нужное значение N , в одной программе вычислить значения интегралов:

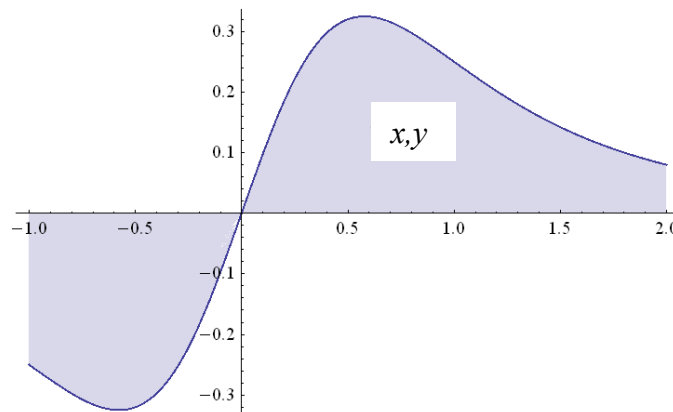
а) $\int_{-1}^2 \frac{x}{(1+x^2)^2} dx$ (ответ: 0.15)

б) $\int_0^{0.5} 4 \cos^2(x) dx$ (ответ: 1.8414709...)

в) $\int_0^{\pi/3} \frac{dx}{\cos^2(x)}$ (ответ: $\sqrt{3} = 1.73205080 \dots$)

г) $\int_4^9 \frac{1+x}{\sqrt{x}} dx$ (ответ: 14.6666...)

10.2. Вычисление интеграла методом Монте-Карло. В этом методе N раз случайным образом генерируются координаты точки $z(x, y)$, где $a \leq x \leq b$; $\min f \leq y \leq \max f$. Для каждого раза проверяется, попадает ли эта точка в область под кривой, определяемой интегрируемой функцией.



Отношение числа точек, попавших в область под кривой, к общему числу генерируемых точек N , равно интегралу от заданной функции. Вычислить те же интегралы, что и в задании 1, учитывая:

а) $\min(f) = -0.3273$; $\max(f) = 0.3232$;

б) $\min(f) = 0.0$; $\max(f) = 4.0$;

в) $\min(f) = 0.0$; $\max(f) = 4.0$;

г) $\min(f) = 0.0$; $\max(f) = 2.0005$;

10.3. Решите следующие уравнения

а) $\exp(-x) - x = 0$ б) $\sin(x) + 1 - x = 0$ в) $x - \frac{\cos^2(x)}{2} = 0$ г) $x - \cos(x/2) = 0$.

методом простой итерации, суть которого состоит в последовательном вычислении чисел по следующему алгоритму (для уравнения $x - \cos(x) = 0$):

$$x_{\text{последующее}} = \cos(x_{\text{предыдущее}})$$

Решением считать такой результат итерации, если он отличается от результата предыдущей итерации не более, чем на 10^{-6} (в любую сторону).

11. ФАЙЛЫ. ВВОД-ВЫВОД В ФАЙЛ

Файлы – последовательность элементов одного типа, записанные на какой-либо носитель информации.

Файлы бывают текстовые и бинарные.

В текстовом файле при записи данные отделяются друг от друга пробелом или символом новой строки.

В бинарных файлах чтение и запись определяются типом записанных данных. Бинарные файлы могут быть типизированными и бестиповыми.

Таблица 3

Описание файловых переменных	
<i>Текстовый файл</i>	<i>Бинарный файл</i>
Var f1: Text;	Var f2: File Of Integer; {типизированный} f3: File Of Real; {типизированный} f4: File Of Char; {типизированный} f5: File Of T; {типизированный, вместо T указывается тип} f6: File; {бестиповой}

Таблица 4

Последовательность команд для записи в файл или считывания из файла		
<i>Действие</i>	<i>Текстовый файл</i>	<i>Бинарный файл</i>
1. Связывание файловой переменной с именем файла	Assign(f, s); здесь f – файловая переменная (описанная как text), s – строка с именем файла, например: 'c:/newfile.txt'	Assign(f, s); здесь f – файловая переменная (описанная как file of T), s – строка с именем файла, например: 'c:/newfile.txt'
2. Открытие файла:		
а) на чтение существующего файла	Reset(f);	—

б) на добавление записей в существующий файл	Append(f);	—
в) на чтение и запись в существующий файл	—	Reset(f);
г) на запись в файл: если файл существовал, то предварительно удаляются записанные в нем данные; если не существовал, то создается файл	Rewrite(f);	—
д) на чтение и запись в файл: если файл существовал, то предварительно удаляются записанные в нем данные; если не существовал, то создается файл	—	Rewrite(f);
3) Запись в файл	<p>Write(f, a, ' ', b, ' ',...); f – файловая переменная, a, b – данные, записываемые в файл и разделенные пробелом.</p> <p>WriteLn(f, a, ' ', b, ' ',...); f – файловая переменная, a, b – данные, записываемые в файл и разделенные пробелом, по окончании записи переход на новую строку в файле.</p>	<p>Write(f, a, b,...); f – файловая переменная, a, b – данные, совпадающие с типом файла</p>
4) Считывание из файла	<p>Read(f, a, b,...); f – файловая переменная, a, b – переменные, в которые читаются данные</p>	<p>Read(f, a, b,...); f – файловая переменная, a, b – переменные, в которые читаются данные</p>

	ReadLn(f, a, b,...); f – файловая переменная, a, b – переменные, в которые читаются данные, затем курсор переводится на новую строку в файле	ReadLn(f, a, b,...); f – файловая переменная, a, b – переменные, в которые читаются данные, затем курсор переводится на новую строку в файле
5) Заккрытие файла	Close(f);	Close(f);

Пример

Создание текстового файла, запись в него целых чисел, закрытие файла.

Открытие этого файла для считывания первых четырех чисел, закрытие файла.

Открытие файла для добавления с переводом строки.

Const

c='d:/new.txt'; {имя файла, с которым будем работать }

Var

i:Integer;

f1:Text; {объявляем файловую переменную}

a:Array [1..10] Of Integer;

Begin

Assign(f1,c); {связываем файловую переменную с именем файла}

Rewrite(f1); {образуем файл с указанным именем}

For i:=1 To 10 Do

Write (f1,i,' '); {записываем в него числа}

Close(f1); {закрываем файл}

Assign(f1,c); {связываем файловую переменную с именем файла}

reset(f1); {открываем файл с указанным именем для чтения}

```
For i:=1 To 4 Do
Write (f1, a[i]); {читаем первые четыре числа}
Close(f1);
Assign(f1,c);{связываем файловую переменную с именем
файла}
Append(f1); {открываем файл с указанным именем для
добавления записей}
For i:=1 To 10 Do
WriteLn (f1, i);
Close(f1);
End.
```

Задания

11.1. Создать функцию $f(n)$ для расчета факториала числа n . Вывести в файл значения факториала для чисел от 2 до 10.

11.2. Создать функцию $f(x) = x^5 + 10x^4 - 3x^2 + 2x \cos(2x) + 1$. Вывести в бинарный (или текстовый) файл в два столбика значения x от 0 до 10 с шагом в 0.1 и соответствующие значения $f(x)$. Считать эти данные из файла и вывести их в строчку на экран компьютера.

11.3. Сгенерировать массив случайных целых чисел, состоящий из 10 элементов. Записать его в строчку в текстовый файл. Считать эти данные из файла, упорядочить этот массив и дописать в файл новую строчку с уже упорядоченным массивом.

11.4. Составьте программу для нахождения целых решений линейного Диофантова уравнения

$$ax + by = c$$

в интервале значений x и y от 0 до 100.

Ввод коэффициентов организуйте через текстовый файл `in.txt`, а все найденные решения – `out.txt`.

ЗАКЛЮЧЕНИЕ

В заключении приведем пример варианта контрольной работы, которая дается в конце семестра.

1. С клавиатуры вводится натуральное число n и вещественное x .

Вычислить произведение

$$\left(1 + \frac{\sin x}{1!}\right) \left(1 + \frac{\sin 2x}{2!}\right) \dots \left(1 + \frac{\sin nx}{n!}\right)$$

2. Вычисления значений функций с помощью рядов, с заданной точностью $\varepsilon = 10^{-4}$. Для вещественного x (вводится с клавиатуры) с заданной точностью найти сумму ряда

$$x - \frac{x^2}{2} + \frac{x^3}{3} - \dots = \sum_{n=1}^{\infty} (-1)^{n+1} \frac{x^n}{n} \quad (\text{функция } \ln(1+x))$$

3. Сгенерировать случайный массив a из 20 элементов, равных случайным числам от 0 до 100 и вывести его на экран.

а) вывести на экран последовательность: $a_1 - a_{20}; a_2 - a_{19}; a_3 - a_{18}; \dots$;

б) вывести на экран элементы a_k , для которых остаток от деления их на 4 равен 2, а также номер k этого элемента;

в) вывести на экран номер наибольшего элемента массива a

4. Написать функцию или процедуру, которой передается два вектора a и b (два массива, по три элемента в каждом), и которая возвращает значение угла между ними в градусах. Для вычисления угла использовать выражение

$$\cos \varphi = \frac{(\vec{a}, \vec{b})}{|\vec{a}| \cdot |\vec{b}|}. \text{ Длины векторов } a \text{ и } b \text{ вычисляются в другой функции по}$$

формуле $|\vec{a}| = \sqrt{a_1^2 + a_2^2 + a_3^2}$, а скалярное произведение еще в одной функции по

формуле $(\vec{a}, \vec{b}) = a_1 b_1 + a_2 b_2 + a_3 b_3$. В главной программе сгенерировать два вектора и продемонстрировать работу функции. (Всего надо написать 3 функции или процедуры)

5. Вывести в файл таблицу значений функции $f(x) = \begin{cases} e^{-(x^3+x^2+x)}, & x < -1 \\ 1, & |x| \leq 1 \\ 1 + \sqrt{1 + \cos(x)}, & x > 1 \end{cases}$ в

точках $x_n = -2 + n \cdot 0,4$, где $n=0, 1, 2, \dots, 10$. Таблицу выводить в два столбца: в одной строке x_n и значение $f(x_n)$. Найти сумму всех значений $f(x_n)$.

ЛИТЕРАТУРА

1. Долинский, М.С. Алгоритмизация и программирование на TURBO PASCAL: от простых до олимпиадных задач: учеб. пособие / М.С. Долинский – СПб.: Питер, 2004. – 240 с.
2. Иванова, Г.С. Основы программирования: учеб./Г.С. Иванова.- 3-е изд., испр. – М.: Изд-во МГТУ им. Н.Э. Баумана, 2004. – 415 с.
3. Культин, Н. Б. Turbo Pascal в задачах и примерах / Н.Б. Культин. – СПб.: БХВ-Петербург, 2006. – 256 с.
4. Лукин, С.Н. Turbo Pascal 7.0. Самоучитель для начинающих / С.Н. Лукин. – М.: Диалог-МИФИ, 2002. – 384 с.
5. Меженный, О.А. Turbo Pascal. Самоучитель: учеб. пособие / О.А. Меженный. – М.: Вильямс: Диалектика, 2008. – 330 с.
6. Немнюгин, С.А. TURBO PASCAL. Программирование на языке высокого уровня: учеб. для вузов / С. А. Немнюгин. – СПб.: Питер, 2005. – 544 с.
7. Немнюгин, С.А. TURBO PASCAL: практикум / С. А. Немнюгин. – СПб.: Питер, 2003. – 272 с.
8. Павловская, Т.А. Паскаль. Программирование на языке высокого уровня: учеб. для вузов / Т.А. Павловская. – СПб: Питер, 2007. – 400 с.
9. Плаксин, М.А. Тестирование и отладка программ для профессионалов будущих и настоящих / М.А. Плаксин. – М.: БИНОМ; Лаборатория базовых знаний, 2007. – 167 с.
10. Программирование на языке Паскаль: задачник / под ред. О.Ф. Усковой. – СПб.: Питер, 2002. – 336 с.
11. Сухарев, М.В. Turbo Pascal 7.0. Теория и практика программирования: учеб. пособие / М.В. Сухарев. – СПб.: Наука и техника, 2006. – 544 с.
12. Фаронов, В.В. Турбо Паскаль 7.0. Начальный курс: учеб. пособие / В.В. Фаронов. – М.: Нолидж, 2007. – 616 с.

- 13.Фаронов, В.В. Турбо Паскаль 7.0. Практика программирования: учеб. пособие / В.В. Фаронов. – М.: КноРус, 2011. – 414 с.
- 14.Юркин, А.Г. Задачник по программированию: учеб. пособие / А.Г. Юркин.– СПб.: Питер, 2002. – 192 с.

Учебное электронное текстовое издание

Мальцев Владимир Николаевич

Поликарпов Филипп Джонович

ПРАКТИКУМ ПО ПРОГРАММИРОВАНИЮ НА ЯЗЫКЕ ПАСКАЛЬ

Редактор

Н.В. Лутова

Компьютерная верстка

авторская

Рекомендовано Учебно-методическим советом ФГАОУ ВПО УрФУ

Разрешено к публикации 07.12.2015

Электронный формат – pdf

Объем 3,84 уч.-изд. л.



620002, Екатеринбург, ул. Мира, 19

Информационный портал

<http://study.urfu.ru/>