МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

**Федеральное государственное автономное образовательное учреждение**

**высшего образования**

**«Национальный исследовательский нижегородский университет им. Н.И. Лобачевского»**

**И.Д. Камскова**

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ для самостоятельной работы по дисциплине «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ»**

Учебно-методическое пособие

Рекомендовано методической комиссией института экономики и предпринимательства ННГУ для обучающихся по специальность среднего профессионального образования 09.02.04 «Информационные системы (по отраслям)»

2017

УДК 681.3

ББК 32.81

Камскова И.Д. Методические рекомендации для самостоятельной работы по дисциплине «Основы алгоритмизации и программирования»

Учебно-методическое пособие. - Нижний Новгород: Нижегородский государственный университет им. Н.И. Лобачевского, 2017. – 62 с.

Рецензент: к.т.н., доцент Штанюк А.А.

Методические рекомендации направлены на реализацию самостоятельной работы по дисциплине «Основы алгоритмизации и программирования» для учащихся CПО по специальности 09.02.04 «Информационные системы (по отраслям)».

Главной задачей самостоятельной работы является развитие общих и профессиональных компетенций, умений приобретать научные знания путем личных поисков, формирование активного интереса к творческому подходу к работе.

Методические рекомендации по  выполнению самостоятельной работы разработаны в соответствии с программой по учебной дисциплине «Основы алгоритмизации и программирования».

Ответственный за выпуск:

председатель методической комиссии ИЭП ННГУ Летягина Е.Н.

УДК 681.3

ББК32.81

© **Национальный исследовательский**

**Нижегородский государственный**

**Университет им. Н.И. Лобачевского, 2017**

СОДЕРЖАНИЕ

[ПОЯСНИТЕЛЬНАЯ ЗАПИСКА 4](#_Toc512454221)

[ТЕМАТИЧЕСКОЕ ПЛАНИРОВАНИЕ 5](#_Toc512454222)

[МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ САМОСТОЯТЕЛЬНЫХ ЗАДАНИЙ 6](#_Toc512454223)

[Раздел 1. Основы алгоритмизации 6](#_Toc512454224)

[Раздел 2. Основы программирования в среде программирования 10](#_Toc512454225)

[Раздел 3. Структурированные типы данных 48](#_Toc512454226)

[Литература для учащегося 61](#_Toc512454227)

# ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Данные методические рекомендации направлены на реализацию самостоятельной работы по учебной дисциплине «Основы алгоритмизации и программирования» для студентов по специальности CПО по специальности 09.02.04 «Информационные системы (по отраслям)».

Самостоятельная работа студента является одним из основных методов приобретения и углубления знаний, познания общественной практики. Главной задачей самостоятельной работы является развитие общих и профессиональных компетенций, умений приобретать научные знания путем личных поисков, формирование активного интереса к творческому самостоятельному подходу в учебной и самостоятельной работе, выполнению самостоятельных практических заданий.

Самостоятельная работа складывается из изучения конспектов лекций, учебной и специальной литературы, как основной, так и дополнительной, выполнения самостоятельных практических заданий.

Методические рекомендации по  выполнению самостоятельной работы разработаны в соответствии с программой по учебной дисциплине «Основы алгоритмизации и программирования».

При реализации программы у студентов формируются компетенции:

ОК 1. Понимать сущность и социальную значимость своей будущей профессии, проявлять к ней устойчивый интерес

ОК 2. Организовывать собственную деятельность, выбирать типовые методы и способы выполнения профессиональных задач, оценивать их эффективность и качество

ОК 3. Принимать решения в стандартных и нестандартных ситуациях и нести за них ответственность

ОК 4. Осуществлять поиск и использование информации, необходимой для эффективного выполнения профессиональных задач, профессионального и личностного развития

ОК 5. Использовать информационно - коммуникационные технологии в профессиональной деятельности

ОК 6. Работать в коллективе и команде, эффективно общаться с коллегами, руководством, потребителями

ОК 7. Брать на себя ответственность за работу членов команды (подчинённых), результат выполнения заданий

ОК 8. Самостоятельно определять задачи профессионального и личностного развития, заниматься самообразованием, осознанно планировать повышение квалификации

ОК 9. Ориентироваться в условиях частой смены технологий в профессиональной деятельности

ПК 1.2. Взаимодействовать со специалистами смежного профиля при разработке методов, средств и технологий применения объектов профессиональной деятельности

ПК 1.3. Производить модификацию отдельных модулей информационной системы в соответствии с рабочим заданием, документировать произведённые изменения ПК

ПК 2.2 Программировать в соответствии с требованиями технического задания.

ПК 2.3 Применять методики тестирования разрабатываемых приложений.

# ТЕМАТИЧЕСКОЕ ПЛАНИРОВАНИЕ

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Раздел / тема учебной дисциплины** | **Форма самостоятельной работы** | **Кол-во часов** | | | |
| **2017** | **2016** | **2015** | **2014** |
| **Раздел 1. Основы алгоритмизации** | |  | | | |
| Раздел 1. Тема 1.1. Алгоритм и его свойства |  | 0 | 0 | 0 | 0 |
| Раздел 1. Тема 1.2. Основные алгоритмические конструкции | Практические задания | 6 | 6 | 6 | 10 |
| **Раздел 2. Основы программирования в среде программирования** | |  | | | |
| Раздел 2. Тема 2.1.  Эволюция языков программирования | Практическое задание | 4 | 0 | 0 | 0 |
| Раздел 2. Тема 2.2. Программирование на языке Паскаль | Практические задания | 30 | 30 | 30 | 36 |
| **Раздел 3. Структурированные типы данных** | |  | | | |
| Раздел 3. Тема 3.1. Понятие массива | Практические задания | 12 | 12 | 12 | 18 |
| Раздел 3. Тема 3.2. Строковый тип данных | Практические задания | 6 | 10 | 10 | 10 |
| **Итого** | | **58** | **58** | **58** | **74** |

# МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ САМОСТОЯТЕЛЬНЫХ ЗАДАНИЙ

## Раздел 1. Основы алгоритмизации

**Тема 1.2. Основные алгоритмические конструкции**

**Методические рекомендации по изучению лекций и литературы**

Ключевым для данной темы являются понятие линейного, циклического алгоритмов и алгоритма ветвления. В процессе освоения материала студенты должны понять суть задач, решаемых с помощью различных алгоритмических структур.

Студент должен изучить назначения алгоритмов, знать графический способ представления алгоритмических структур, знать категории задач, решаемых при помощи разных алгоритмических структур.

**Основные алгоритмические конструкции**

К основным алгоритмическим структурам относятся:

следование;

ветвление;

повторение;

Каждая структура имеет единственный вход и единственный выход, причем составляющие их компоненты изображаются по вертикали. Исходя из структуры алгоритмов, различают алгоритмы линейного, разветвляющегося и циклического вычислительного процесса, а также алгоритмы со структурой вложенных циклов.

Алгоритмы решения сложных задач могут включать все перечисленные структуры, которые используются для реализации отдельных участков общего алгоритма.

**Алгоритмы линейной структуры**

Линейным называется вычислительный процесс, в котором операции выполняются последовательно в естественном порядке. Блоки в этой структуре располагаются на схеме в том же порядке, в каком должны быть выполнены предписываемые ими действия. Такой порядок исполнения действий, называется естественным

**Алгоритмы разветвляющейся структуры**

Разветвляющимся называют вычислительный процесс, в котором операции выполняются в одном из направлений в зависимости от заданных условий. В общем случае число ветвей в алгоритме разветвляющейся структуры не обязательно равно двум.

Отсюда, различают три вида структур разветвляющегося алгоритма.

|  |  |
| --- | --- |
| Полный алгоритм ветвления  Действие 1  Действие 2  Условие  Да  Нет  Применяется, когда в зависимости от условия нужно выполнить либо одно, либо другое действие. Действие 1 или Действие 2 может в свою очередь содержать несколько этапов. |  |
| Обход  Условие  Действие  *Да*  *Нет*  Частный случай разветвления, когда одна ветвь не содержит никаких действий. |  |
| Множественный выбор  *Выражение*  *Действие 1*  *Действие 2*  *Действие N*  *Константа 1*  *Константа 2*  *Константа N* | |

**Алгоритмы циклической структуры**

Циклическим называют вычислительный процесс, в котором многократно повторяются некоторые этапы вычислений.

Часто при решении задач приходится многократно вычислять значения по одним и тем же математическим зависимостям для различных значений входящих в них величин. Такие многократно повторяемые участки вычислительного процесса называются циклами. Использование циклов позволяет существенно сократить объем схемы алгоритма и длину соответствующей ей программы. Различают циклы с заданным и неизвестным числом повторений. К последним относятся итерационные циклы, характеризующиеся последовательным приближением к искомому значению с заданной точностью.

Для организации цикла необходимо выполнить следующие действия:

Задать перед циклом начальное значение переменной, изменяющейся в цикле;

Изменять переменную перед каждым новым повторением цикла;

Проверять условие окончания или повторения цикла;

Управлять циклом, т.е. переходить к его началу, если он не закончен, или выходить из него по окончании.

Переменная, изменяющаяся в цикле, называется параметром цикла. В одном цикле может быть несколько параметров.

Переменную, значения которой вычисляются машиной и хранятся в одной и той же ячейке памяти, называют простой переменной а переменную, являющуюся элементом массива переменной с индексом. Следует иметь в виду, что параметром цикла является при использовании простой переменной сама переменная, а при использовании переменной с индексом – ее индекс.

Возможны три способа организации структур циклических алгоритмов.

|  |  |
| --- | --- |
| Цикл «До»  Начальные присваивания  Тело цикла  *Условие*  Применяется при необходимости выполнить какие-либо вычисления несколько раз до выполнения некоторого условия. Особенность этого цикла в том, что он всегда выполняется хотя бы один раз, так как первая проверка условия выхода из цикла происходит после того, как тело цикла выполнено. Тело цикла — та последовательность действий, которая выполняется многократно (в цикле). Начальные присвоения — задание начальных значений тем переменным, которое используются в теле цикла. |  |

|  |  |
| --- | --- |
| Цикл «Пока»  Начальные присваивания  Тело цикла  *Условие*  Отличается от цикла До тем, что проверка условия поводится до выполнения тела цикла, и если при первой поверке условие выхода из цикла не выполняется ни разу. |  |
| Счетный цикл (цикл с управляющей переменной) |  |

|  |  |  |
| --- | --- | --- |
| **Пример.**  Составить схему алгоритма для вычисления и печати значения выражения    при изменении параметра  от  до  с шагом . Такие вычисления значений функции называются табулированием. Выполняя табулирование, необходимо последовательно перебрать все значения параметра цикла  в указанном диапазоне его величин. Как только это условие будет нарушено, следует приостановить вычисления. | | |
| Начало          Конец  Начало              Конец  *да*  *нет*  Начало              Конец  *да*  *нет* |  |  |

**Практические задания для самостоятельного выполнения.**

Разработать блок-схему алгоритма решения нижеприведённых задач. Данные практические задания рекомендуется выполнить в редакторе Microsoft Word. Для изображения блок-схем инструменты Рисования.

Задание 1. Составить схему алгоритма вычисления площади  круга, полагая, что в качестве исходных данных заданы значения радиуса

Задание 2. Сосчитать Y(x)=x+a, Z(x)=x-b

Задание 3. Вычислить



Задание 4. Вычислить



Задание 5. В зависимости от номера введенного месяца вывести время года.

Задание 6. Сосчитать сумму и количество целых чисел от M до N. Использовать До

Задание 7. Сосчитать количество четных чисел от M до N. Использовать Пока

Задание 8. Сосчитать сумму ряда . Использовать Счетный цикл

Задание 9. Сосчитать сумму ряда . Использовать До и счетный цикл

Задание 10. Дано натуральное число  n. Вычислите  cosx+cosx2+…+cosxn

**Критерии оценки самостоятельного задания:**

* Правильность построения алгоритма
* Аккуратность оформления.

Оценка «Отлично» ставится за правильный алгоритм и аккуратное оформление.

Оценка «Хорошо» ставится за правильный алгоритм с незначительными ошибками и аккуратное оформление.

Оценка «Удовлетворительно» ставится за правильный алгоритм с существенными ошибками замечаниями и неаккуратное оформление.

## Раздел 2. Основы программирования в среде программирования

**Тема 2.1.  Эволюция языков программирования**

**Методические рекомендации по изучению лекций и литературы**

Ключевым для данной темы является понятие языка программирования. В процессе освоения материала студенты должны понять каким образом эволюционировали языки программирования – от процедурных до объектно-ориентированных. Знать, что такое визуальная среда программирования.

**Эволюция языков программирования**

Развитие языков программирования.

Индустрия разработки автоматизированных информационных систем управления зародилась в 1950-х - 1960-х годах и к концу века приобрела вполне законченные формы. Вместе с ними развивались и языки программирования.

Первый шаг - «стихийное» программирование.

Этот этап охватывает период от момента появления первых вычислительных машин до середины 60-х годов XX в. В этот период отсутствовали сформулированные технологии, и программирование фактически было творчеством, искусством.

Первые программы имели простейшую структуру. Они состояли из программы на машинном языке и обрабатываемых ею данных. Программы в машинных кодах не позволяли программисту четко отслеживать последовательность выполняемых операций и местонахождение данных при программировании.

Появление ассемблеров позволило вместо двоичных или 16-ричных кодов использовать символические имена данных и коды операций. В результате программы стали более «читаемыми».

Создание языков программирования высокого уровня, таких, как FORTRAN и ALGOL, существенно упростило программирование вычислений, но увеличило сложность программ.

На данном этапе использовалась разработка программного обеспечения «снизу-вверх» - подход, при котором вначале реализовывали простые подпрограммы, из которых затем пытались построить сложную программу.

Второй шаг – появление структурного программирования

В конце 60-х – начале 70-х годов XX в. стало разрабатываться сложное ПО: операционные системы, системы управления космическими объектами, управления оборонным комплексом, автоматизации крупного финансового учреждения и т.д.

Появилось структурное программирование.

Основные принципы технологии структурного проектирования и кодирования:

Нисходящее функциональное проектирование, при котором в системе выделяются основные функциональные подсистемы, которые потом разбиваются на подсистемы и т.д «сверху-вниз».

Применение трех основных алгоритмических структур.

Принципы структурного программирования были заложены в основу так называемых процедурных языков программирования. Среди наиболее известных языков этой группы стоит назвать PL/1, ALGOL-68, Pascal, С.

Одновременно со структурным программированием появилось огромное количество языков, базирующихся на других концепциях, но большинство из них не выдержало конкуренции. Какие-то языки были просто забыты, идеи других были в дальнейшем использованы в следующих версиях развиваемых языков.

Третий шаг – Появление объектно-ориентированного программирования

Основным достоинством объектно-ориентированного программирования является «более естественная» декомпозиция (разбиение) программного обеспечения при программировании процессов. В объектно-ориентированном программировании вводится понятие Объект. Объекты в ООП — это объекты реального мира, свойства которых можно запрограммировать при помощи соответствующих языков программирования. Объекты выполняют необходимые действия, передавая друг другу сообщения.

Объектно-ориентированный подход основан на представлении программы в виде совокупности объектов. Взаимодействие программных объектов в такой системе осуществляется путем обмена сообщениями.

Объектная структура программы используется в новых версиях универсальных языков программирования, таких, как Object Pascal, C++, Java.

В дальнейшем были созданы среды, поддерживающие визуальное программирование, например, Delphi, Visual C++ и т. д. При использовании визуальной среды у программиста появляется возможность проектировать некоторую часть системы с применением визуальных средств. Результатом визуального проектирования является заготовка будущей программы, в которую уже внесены соответствующие коды.

Введение в Object Pascal.

Язык Pascal (Паскаль) был создан [Никлаусом Виртом](https://ru.wikipedia.org/wiki/%D0%92%D0%B8%D1%80%D1%82,_%D0%9D%D0%B8%D0%BA%D0%BB%D0%B0%D1%83%D1%81) в 1968—1969 годах. Язык назван в честь французского математика, физика, литератора и философа [Блеза Паскаля](https://ru.wikipedia.org/wiki/%D0%9F%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D1%8C,_%D0%91%D0%BB%D0%B5%D0%B7) (1623-1662). Блез Паскаль [французский](https://ru.wikipedia.org/wiki/%D0%A4%D1%80%D0%B0%D0%BD%D1%86%D0%B8%D1%8F) [математик](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), [механик](https://ru.wikipedia.org/wiki/%D0%9C%D0%B5%D1%85%D0%B0%D0%BD%D0%B8%D0%BA%D0%B0), [физик](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%B7%D0%B8%D0%BA), [литератор](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%82%D0%B5%D1%80%D0%B0%D1%82%D1%83%D1%80%D0%B0) и [философ](https://ru.wikipedia.org/wiki/%D0%A4%D0%B8%D0%BB%D0%BE%D1%81%D0%BE%D1%84).



Классик французской литературы, один из основателей [математического анализа](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%B0%D0%BD%D0%B0%D0%BB%D0%B8%D0%B7), [теории вероятностей](https://ru.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BE%D1%80%D0%B8%D1%8F_%D0%B2%D0%B5%D1%80%D0%BE%D1%8F%D1%82%D0%BD%D0%BE%D1%81%D1%82%D0%B5%D0%B9) и [проективной геометрии](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D0%B2%D0%BD%D0%B0%D1%8F_%D0%B3%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%8F), создатель первых образцов счётной техники, автор [основного закона](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%BA%D0%BE%D0%BD_%D0%9F%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D1%8F) [гидростатики](https://ru.wikipedia.org/wiki/%D0%93%D0%B8%D0%B4%D1%80%D0%BE%D1%81%D1%82%D0%B0%D1%82%D0%B8%D0%BA%D0%B0), создал первую в мире механическую машину, складывающую два числа.



В [1983 году](https://ru.wikipedia.org/wiki/1983_%D0%B3%D0%BE%D0%B4) появилась первая версия [интегрированной среды разработки](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D0%B3%D1%80%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%B0%D1%8F_%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8) Turbo Pascal фирмы [Borland](https://ru.wikipedia.org/wiki/Borland), основывавшаяся на одноимённой реализации Паскаля.

В [1986 году](https://ru.wikipedia.org/wiki/1986_%D0%B3%D0%BE%D0%B4) фирма [Apple](https://ru.wikipedia.org/wiki/Apple) разработала [объектное](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5) расширение языка Паскаль, получив в результате [Object Pascal](https://ru.wikipedia.org/wiki/Object_Pascal).

В [1989 году](https://ru.wikipedia.org/wiki/1989_%D0%B3%D0%BE%D0%B4) объектное расширение языка было добавлено в Turbo Pascal версии 5.5. Последняя версия (7.0) была переименована в Borland Pascal.

Дальнейшее развитие Паскаля от Borland породило вариант [Object Pascal](https://ru.wikipedia.org/wiki/Object_Pascal) от Borland, впоследствии, в ходе развития среды программирования [Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%81%D1%80%D0%B5%D0%B4%D0%B0_%D1%80%D0%B0%D0%B7%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B8)), получивший [одноимённое название](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)).

Современные версии Object Pascal

Важным шагом в развитии языка является появление свободных реализаций языка Паскаль [Free Pascal](https://ru.wikipedia.org/wiki/Free_Pascal) и [GNU Pascal](https://ru.wikipedia.org/wiki/GNU_Pascal), которые не только вобрали в себя черты множества других диалектов языка, но и обеспечили чрезвычайно широкую переносимость написанных на нём программ.

Кроме того, разработан [PascalABC.NET](https://ru.wikipedia.org/wiki/PascalABC.NET) — язык программирования Паскаль, включающий большинство возможностей языка [Delphi](https://ru.wikipedia.org/wiki/Delphi_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), а также ряд собственных расширений. Он основан на платформе [Microsoft.NET](https://ru.wikipedia.org/wiki/.NET_Framework) и содержит почти все современные языковые средства: [классы](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D1%81%D1%81_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [перегрузку операций](https://ru.wikipedia.org/wiki/%D0%9F%D0%B5%D1%80%D0%B5%D0%B3%D1%80%D1%83%D0%B7%D0%BA%D0%B0_%D0%BE%D0%BF%D0%B5%D1%80%D0%B0%D1%82%D0%BE%D1%80%D0%BE%D0%B2), [интерфейсы](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D1%84%D0%B5%D0%B9%D1%81_(%D0%BE%D0%B1%D1%8A%D0%B5%D0%BA%D1%82%D0%BD%D0%BE-%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), [обработку исключений](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D0%BA%D0%B0_%D0%B8%D1%81%D0%BA%D0%BB%D1%8E%D1%87%D0%B5%D0%BD%D0%B8%D0%B9), [обобщенные классы и подпрограммы](https://ru.wikipedia.org/wiki/%D0%9E%D0%B1%D0%BE%D0%B1%D1%89%D1%91%D0%BD%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5), [сборку мусора](https://ru.wikipedia.org/wiki/%D0%A1%D0%B1%D0%BE%D1%80%D0%BA%D0%B0_%D0%BC%D1%83%D1%81%D0%BE%D1%80%D0%B0_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)), лямбда-выражения.

Особенности языка Паскаль

Особенностями языка являются [строгая типизация](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D0%BE%D0%B3%D0%B0%D1%8F_%D1%82%D0%B8%D0%BF%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F) и наличие средств [структурного (процедурного) программирования](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5). Паскаль был одним из первых таких языков. По мнению Вирта, язык должен способствовать дисциплинированному программированию.

**Практическое задание для самостоятельного выполнения**

Используя материалы лекций, рекомендованную литературу и Internet-ресурсы, проанализировать эволюцию языка Паскаль. Рекомендуется ответ представить в форме таблицы:

|  |  |  |  |
| --- | --- | --- | --- |
| Год появления версии Паскаль | Фирма | Особенности версии | Пояснения |

**Тема 2.2. Программирование на языке Паскаль**

**Методические рекомендации по изучению лекций и литературы**

Ключевыми для данной темы являются следующие понятия: структура программы Паскаль, операторы присваивания, ввода, вывода, типы данных: целые, логические и вещественные. Студенты должны выучить функции преобразования типов, стандартные математические функции Паскаль, арифметические, логические операции и операции отношения. Условный оператор If и оператор выбора Case. Три оператора цикла For, While, Repeat.

Студенты должны знать категории задач, решаемых при помощи линейных, условных и циклических операторов Паскаль, уметь применять различные виды операций к решению задач.

Студенты должны знать синтаксис и семантику вышеперечисленных операторов: присваивания, ввода, вывода, условных операторов и операторов цикла.

**Линейные вычислительные процессы**

**Линейный вычислительный процесс** представляет собой последовательность действий, выполняемых один за другим. Основу программы линейного вычислительного процесса составляют оператор присваивания и операторы ввода-вывода данных.

**Оператор присваивания**

**Оператор присваивания** служит для вычисления выражения и записи результата в память компьютера.

Общий вид записи оператора:

**переменная := выражени>;**

Знак **:=** читается как “**присвоить**”. **Выражение** – это переменные, константы, функции и формулы. Конец любого оператора в Турбо Паскале фиксируется точкой с запятой.

**A:=3\*4.8;**

**Su:=X+X\*4.78;**

**C:=C+1;**

При выполнении оператора присваивания происходит замена текущего значения переменной, стоящей слева от знака присваивания, новым значением, полученным в результате вычисления выражения. Например, после выполнения оператора

**k : = k+1**; текущее значение переменной k увеличится на 1.

При записи оператора присваивания необходимо строго следить за типами переменной и выражения Тип переменной в левой части оператора присваивания обычно должен совпадать с типом значения выражения в правой части. Возможны случаи несовпадения типов, например, когда слева переменная вещественного типа, а справа выражение целого типа:

**Операторы ввода данных с клавиатуры**

**READ(список переменных);**

**READLN(список переменных);**

Переменные в списке перечисляются через запятую.

Операторы ввода обеспечивают чтение данных с клавиатуры, преобразование из внешнего представления во внутреннее и присваивание их значений переменным, указанным в списке.

При выполнении команды Read или Readln выполнение программы останавливается и компьютер ждет, пока пользователь не введет с клавиатуры нужное количество значений для переменных.

Вводятся только **значения** переменных. При вводе с клавиатуры нескольких значений, они отделяются друг от друга одним или несколькими **пробелами**. Ввод заканчивается нажатием клавиши ENTER.

Readln отличается от Read тем, что после его выполнения автоматически осуществляется переход на следующую строку.

**Пример**. Пусть в задаче определены следующие исходные данные: a = 4; x = 3,8; y = 1,2⋅104 ; k = 5; m = 74.

Операторы ввода

**read(a, x, y);**

**read(k, m);**

вызывают чтение данных с клавиатуры, набранных следующим образом:

**4 3.8 1.2E4 <Enter>**

**5 74 <Enter>.**

**Операторы вывода данных на экран**

**WRITE(список выражений);**

**WRITELN(список выражений);**

Выражения в списке перечисляются через запятую. В качестве простейших выражений могут быть константы, переменные, символы, строки. Символы и строки выводятся в апострофах. Если необходимо вывести значения переменных или констант (любых, в том числе и символьных), то они выводятся без апострофов.

Оператор **WRITELN** отличается от оператора **WRITE** лишь тем, что после вывода значения последнего выражения из списка осуществляется автоматический переход курсора к началу следующей строки.

Например, операторы

**y=7;**

**writeln(‘Выходные данные’);**

**writeln(‘К =’, 8+5, ‘ y =’, y);**

выводят на экран результаты в следующем виде:

**Выходные данные**

**K =13 y =7**

**Writeln –** выводит пустую строку

**Управление выводом данных**

При стандартной форме вывода в Object Pascal вещественные числа отображаются на экране с порядком. Мантисса выводится в нормализованном виде - с одной значащей цифрой в целой части и с десятью цифрами в дробной части. На порядок отводится четыре позиции: первая позиция - под букву Е, вторая позиция - под знак порядка, третья и четвертая позиции – под цифры порядка. В PascalABC.Net стандартная форма вывода вещественного числа – десятичная форма.

Если программиста не устраивает стандартная форма вывода, то он может использовать форматированный вывод, предусмотренный в Паскале. При форматированном выводе чисел используются два формата:

а) ширина поля, определяющая число позиций отводимых для вывода всего числа, включая целую, дробную части, знак и десятичную точку;

б) точность представления вещественного числа, определяющая число позиций в дробной части.

Вещественные числа с указанием формата всегда выводятся в десятичной форме. Для целых чисел используется только формат ширины поля. В операторах вывода форматы записываются после выражения и отделяются друг от друга двоеточием. Например, при записи оператора

**writeln(‘K =’, 8+5:3, ‘ y =’, 7/10:5:1);**

результаты отображаются на экране в следующем виде:

**K = 13 y = 0.7**

В приведенном примере форматы 3 и 5 задают ширину поля, формат 1 – точность выводимого числа.

**Пример 1**

Var I, j, k, l:integer;

begin

I:=1;

J:=2;

K:=3;

L:=4;

Write (I, j);

Writeln (k);

Write (l, i);

Writeln (k);

End.

На экране после выполнения этой программы будет выведено

123

413

**Пример 2**

Var I, j, k:integer;

begin

I:=1;

J:=2;

K:=3;

Writeln (I:3, j:3, k:3);

Writeln (‘i=’, I,’, j=’,j);

Writeln (‘i=’, I:4,’, j=’,j:4);

End.

На экране после выполнения этой программы будет выведено

1 2 3

i=1, j=2

i= 1, j= 2

**Пример 3**

Var I, j, k:integer;

Begin

Readln(I,j,k);

Writeln (‘i=’,I,‘ j=’, j, ‘ k=’,k);

Writeln;

I:=10;

J:=12;

K:=14;

Writeln (‘i=’,I:3,‘ j=’, j:3, ‘ k=’k:3);

End.

При выполнении этого кода программа приостанавливается и ждет ввода значения трех переменных I, j, k. Допустим, мы ввели значения

1

3

5

Тогда результат выполнения программы будет следующий.

1

3

5

I=1 J=3 K=5

I= 10 J= 12 K= 14

**Типы данных**

Каждая переменная и константа в программе на языке Паскаль имеет свой тип данных. Тип определяет набор операций, которые могут быть к ней применимы, а также тип результата выполнения этих операций. Требование предварительного описания идентификатора кажется довольно строгим, но здесь проявляется тенденция в сторону надежности создаваемой программы. В большой программе бывает трудно обнаружить ошибочно введенный или пропущенный символ в идентификаторе.

Понятие типа – одно из фундаментальных понятий в Паскале. Описать или определить идентификатор – это означает указать тип связанного с этим идентификатором объекта программы: константы или переменной. Тип определяет:

* Способ внутреннего представления объекта для компьютера
* Действия, которые разрешаются над ним выполнять.

В программе перед использованием переменной в специальном разделе описаний переменных должно стоять зарезервированное слово VAR. Однотипные переменные в разделе перечисляются через запятую и отделяются от объявленного типа двоеточием. После определения типа ставится точка с запятой.

**VAR** идентификатор [,идентификатор,…]**:**тип**;**

[идентификатор [,идентификатор,…]**:**тип**;**…]

Например,

**VAR A : INTEGER;**

**B, C: REAL;**

**Классификация типов данных**

В Pascal можно выделить следующие группы типов данных:

* **Простые типы** для хранения информации в форме чисел и других "упорядоченных" значении.
  + Целые
  + Логический
  + Символьный
  + Вещественный
  + Перечисляемый
  + Тип-диапазон

Все простые типы, кроме вещественного, называют **порядковыми**.

* **Строковые типы** для хранения последовательностей символов.
* **Структурированные типы** для одновременного хранения информации разных типов.
  + Массивы
  + Записи
  + Множества
  + Файлы
* **Указательные типы** для косвенного обращения к переменным заданных типов.
* **Процедурные типы** для обращения к процедурам и функциям, рассматриваемым как переменные.
* **Классы**

**Простые типы данных**

Простой тип данных в определяет конечное линейное множество значений параметра.

Рассмотрим первые 4 типа

**Целый тип данных**

Существует 5 стандартных **целых типов**:

|  |  |  |  |
| --- | --- | --- | --- |
| Тип | Диапазон | Объем памяти | Примечания |
| ShortInt | -128..127 | 1 байт | Со знаком |
| Byte | 0..255 | 1 байт | Без знака |
| SmollInt | -32768..32767 | 2 байта | Со знаком |
| Word | 0..65535 | 2 байта | Без знака |
| Integer | -2147483648..2147483647 | 4 байта | Со знаком |

Выбор того или иного целого типа зависит от диапазона допустимых значений соответствующего параметра.

**Пример:**

Даны числа

014; 25\*100; 27,3\*102.

Как в Паскале будут выглядеть эти числа, если они имеют тип Integer.

**Ответ**: 14; 2500; 2730

**Пример:**

Даны числа

4.0; 2,5\*100; 273\*100.

Как в Паскале будут выглядеть эти числа, если они имеют тип Integer.

**Ответ**: 4; 250; 273

**Логический тип**

Логический тип данных- это тип **Boolean**, принимающий значения True и False. Логическое значение в памяти занимает 1 байт. На самом деле, конечно, достаточно и одного бита, но оперировать ячейками меньше байта, мы, к сожалению, не можем. False< True.

**Символьный тип**

Символьные типы обеспечивают хранение отдельных символов. Основной тип данных - **Char**, который содержит символы с кодами 0..255. Занимает в памяти 1 байт. Для кодирования символов используется кодировка Windows-1251 и, что реже – Unicode.

**Вещественный тип**

Вещественные числа могут быть записаны в виде десятичной дроби, привычном для школьника (пример, 0.1 или -5.8125) – дробная часть отделяется от целой точкой. А также в форме с плавающей запятой (с указанием порядка)



В форме с плавающей запятой число представляется двумя компонентами: а – **мантиссой числа и р – десятичным порядком числа.**

**Мантисса** используется для записи цифр числа, а **порядок** – для указания положения запятой.

Мантисса числа может быть целыми или десятичным числом, а знак мантиссы определяет знак всего числа.

Например, в десятичной системе счисления число 3,14 представим в виде

3,14 = 0,314∙101,

где мантисса равна 0,314, а порядок 1.

Очевидно, такое представление далеко не однозначно. Можно ведь 3,14 записать так:

3,14 = 3,14∙100 = 31,4∙10-1 = 0,0314∙102,

Порядок числа определяет положение запятой в записи мантиссы. Если записываемое число меньше единицы, экспонента будет отрицательной. При корректировке порядка соответствующим образом меняется и положение запятой – запятая как бы «плавает». Отсюда и название метода представления чисел.

Представьте число

-36,6 в форме с плавающей запятой с разными порядками.

В Паскале числа с порядками записываются в экспоненциальной форме с использованием буквы Е (вместо 10) за которым следует порядок.

0.0314Е2

Буква Е читается как «умножить на 10 в степени…».

**Примеры** записи вещественных чисел в экспоненциальной форме

|  |  |
| --- | --- |
| **Математическая запись** | **Запись в Паскале** |
| 3,14\*105 | 3.14Е5 |
| -17\*10-2 | -17Е-2 |
| 25,625 | 2.5625Е1 |
| 10-6 | 1Е-6 |
| 0.00048 | 4.8Е-4 |

**Обратные преобразования**

**Пример:** Даны вещественные числа в экспоненциальной форме или в форме с порядком, записать, как они будут выглядеть в десятичной форме

|  |  |
| --- | --- |
| **Вещественные числа в экспоненциальной форме или в форме с порядком** | **Десятичная дробь** |
| 3,14\*105 | 314000,0 |
| -17\*10-2 | -0,17 |
| 0.505Е-4 | 0,0000505 |
| -10-6 | -0,00001 |
| 0.00048 | 0,00048 |

Наиболее популярный вещественный тип – **Real** – диапазон значений

От -1,8Е308 до 1,8Е308. Занимает 8 байт памяти.

**Некоторые функции преобразования типа**

Эти функции предназначены для преобразования типов величин, например – символа в целое число, вещественное в целое и пр.

**Chr(a: byte): char** - Преобразует код аргумента **а** в символ в кодировке Windows 1251. Аргумент должен быть в диапазоне **0 .. 255.**

**Ord(a: char): byte** - Преобразует символ в код в кодировке Windows 1251

**chr(33)=!**

**ord(z)=122**

**ChrUnicode(a: word): char** -  Преобразует код в символ в кодировке Unicode

**OrdUnicode(a: char): word** - Преобразует символ в код в кодировке Unicode

**chrunicode(33)=!**

**ordunicode(z)=122**

**Round(x:real):integer** – округление значения **вещественного (real)** аргумента **x** до ближайшего **целого.**

**Round(36.6)=37**

**Round(36.4)=36**

**Trunc(x:real):integer** - Выделение целой части **вещественного (real)** аргумента **x**. Тип результата **целый.**

**Trunc(36.6)=36**

**Trunc(36.4)=36**

**Int(x:real):real -** Выделение целой части **вещественного (real)** аргумента **x**. Тип результата **вещественный (real)**

**Int(36.6)=36.0**

**Frac(x:real):real -** Выделение дробной части **вещественного (real)** аргумента **x**. Тип результата **вещественный (real)**

**Frac(36.6)=0.6**

**Odd(i: integer): boolean** - возвращает True, если i нечетно.

**Odd(2)=False**

**Odd(3)=True**

**Inc(var i: integer)** - Увеличивает значение переменной i на 1

**Inc(var i: integer; n: integer) -** Увеличивает значение переменной i на n

**Dec(var i: integer)** - Уменьшает значение переменной i на 1

**Dec(var i: integer; n: integer)** - Уменьшает значение переменной i на n

**Примеры решения задач**

**Задание 1**

Дан радиус окружности r. Найти площадь круга и длину окружности.

**Var r:integer; //r-переменная, в которой хранится радиус**

**Dok,Pk:real; //Dok-переменная для расчета длины окружности, Pk-**

**переменная для расчета площади круга**

**Begin**

**Write ('Введите радиус ');**

**Readln(r);**

**dok:=2\*Pi\*r;**

**pk:=Pi\*r\*r;**

**Writeln ('Площадь круга=',Pk:0:3);**

**Writeln ('Длина окружности=',Dok:0:3);**

**End.**

Результат:

**Введите радиус 3**

**Площадь круга=28.274**

**Длина окружности=18.850**

**Задание 2**

Даны два целых числа. Найти их среднее арифметическое и среднее арифметическое их квадратов.

**Var a1,a2:integer;**

**s,sk:real;//s-среднеарифметическое а1 и а2**

**//sk-среднеарифметическое квадратов а1 и а2**

**Begin**

**Writeln ('Введите два числа ');**

**Readln(a1,a2);**

**s:=(a1+a2)/2;**

**sk:=(a1\*a1+a2\*a2)/2;**

**Writeln ('Cреднее арифметическое=',s:0:2);**

**Writeln ('Cреднее арифметическое квадратов=',sk:0:2);**

**End.**

Результат:

**Введите два числа**

**12**

**7**

**Cреднее арифметическое=9.50**

**Cреднее арифметическое квадратов=96.50**

**Задание 3**

Написать программу обмена значениями двух переменных, используя и не используя дополнительную переменную.

Обмен значений двух переменных протекает по следующему алгоритму:

1 - В дополнительной переменной сохраняется значение первой переменной.

2 - Первой переменной присваивается значение второй.

3 - Второй переменной присваивается значение дополнительной переменной, это значение равно старому значению первой переменной.

**Var x,y,d:integer;//d-дополнительная переменная**

**Begin**

**Writeln ('Введите два числа ');**

**Readln(x,y);**

**Writeln('Первоначальные значения x=',x,' y=',y);**

**d:=x;**

**y:=x;**

**x:=d;**

**Writeln('Значения после изменений x=',x,' y=',y);**

**End.**

Результат:

**Введите два числа**

**5**

**11**

**Первоначальные значения x=5 y=11**

**Значения после изменений x=5 y=5**

Без дополнительной переменной х=3, у=6

**x:=x+y;** х=9

**y:=x-y;** у=3

**x:=x-y;** х=6

**Var x,y:integer;**

**Begin**

**Writeln ('Введите два числа ');**

**Readln(x,y);**

**Writeln('Первоначальные значения x=',x,' y=',y);**

**x:=x+y;**

**y:=x-y;**

**x:=x-y;**

**Writeln('Значения после изменений x=',x,' y=',y);**

**End.**

Результат:

**Введите два числа**

**6**

**3**

**Первоначальные значения x=6 y=3**

**Значения после изменений x=3 y=6**

**Арифметические выражения**

**Арифметические выражения** состоят из операндов и операций. Операции находятся между операндами и обозначают действия, которые выполняются над операндами. В качестве операндов выражения можно использовать: переменную, константу, функцию или другое выражение.

Все операции в Паскале можно разбить на следующие группы

Унарные

Арифметические

Логические

Операции отношения

**Унарные операции**

**Унарная операция** относится к одному операнду, и ее знак записывается перед операндом, например, -х. Унарный плюс не меняет операнд. +5 – так и останется 5. А унарный минус меняет знак операнда на противоположный.

**Бинарная операция** выражает отношение между двумя операндами, и ее знак записывается между операндами, например, Х+У.

**Арифметические операции**

В бесскобочных арифметических выражениях операции выполняются слева направо в соответствии с их приоритетом.

Приоритет операторов влияет на порядок их выполнения. При вычислении значения выражения в первую очередь выполняются операторы с более высоким приоритетом. Если приоритет операторов в выражении одинаковый, то сначала выполняется тот оператор, который находится левее.

**Приоритет арифметических операций.**

\* (умножение)

/ (деление)

\_DIV\_ (деление целых чисел нацело) 5 div 2 = 2;

\_MOD\_ (выделение остатка от деления целых чисел). 5 mod 2 = 1

Эти перечисленные операции имеют одинаковый приоритет

+ (сложение) и - (вычитание) – тоже имеют одинаковый приоритет, но он более низкий, чем у предыдущих операций.

Например,

123\*0,001\*i+1

1 2 3

А + В/С\*Summa\*0,75\*В1+ВЗ+ВЗ/3

6 1 2 3 4 7 8 4

А - В/С+S\*0,75 mod В1+ВЗ+ВЗ/3 div 4

6 1 7 2 3 8 9 4 5

Изменить порядок выполнения операций можно с помощью круглых скобок. Выражение, заключенное в круглые скобки, выполняется в первую очередь. Например, выражению a/b\*c соответствует математическая запись , а выражению а/(b\*с) соответствует запись .

Тип арифметического выражения определяется типом входящих в него операндов.

|  |  |  |  |
| --- | --- | --- | --- |
| **Операция** | **Знак** | **Типы операндов** | **Тип результата** |
| Сложение | + | I  Хотя бы один R | I  R |
| Вычитание | - | I  Хотя бы один R | I  R |
| Умножение | \* | I  Хотя бы один R | R |
| Деление | / | I или R | R |
| Деление целых чисел | div | I | I |
| Остаток от деления целых чисел | mod | I | I |

**Логические операции**

Логические операции применяются к величинам логического типа, результат операции- тоже логический. Имеется одна унарная логическая операция not (логическое отрицание) и три бинарные – and (логическое и), or (логическое или), xor (логическое исключающее или).

|  |  |
| --- | --- |
| **х** | **Not x** |
| False  True | True  False |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **x** | **y** | **X and Y** | **X or Y** | **X xor Y** |
| False | False | False | False | False |
| False | True | False | True | True |
| True | False | False | True | True |
| True | True | True | True | False |

**Операции отношения**

Операции отношения предназначены для сравнения двух величин. Результат сравнения имеет логический тип.

= - равно

<> - не равно

< - меньше

> - больше

<= - меньше или равно

>= - больше или равно

К логическим и операциям отношения также применимы круглые скобки для изменения приоритета операций.

**Стандартные функции**

Часто используемые в арифметических выражениях элементарные математические функции оформлены в виде стандартных функций Паскаля.

Для правильного обращения к стандартной функции, необходимо записать имя функции, за которым в круглых скобках следует аргумент (параметр). Функцию можно использовать в качестве операнда выражения, например в инструкции присваивания.

Приоритет вычисления функций выше, чем приоритет арифметических операций.

Так, чтобы вычислить квадратный корень, достаточно записать k:=Sqrt(n), где Sqrt() — функция вычисления квадратного корня, n — переменная, которая содержит число, квадратный корень которого надо вычислить.

Ниже представлен набор стандартных функций с указанием типов функции и аргумента. В таблице приняты обозначения: I-целый тип, R- вещественный тип.

У тригономертических функций аргумент вводится в радианах.

|  |  |  |  |
| --- | --- | --- | --- |
| **Математическая запись** | **Запись в Турбо Паскале** | **Тип аргумента** | **Тип функции** |
| **| x |** | **Abs(x)** | I  R | I  R |
| **x2** | **Sqr(x)** | I  R | I  R |
| **sin x**  **cos x**  **arctg x** | **Sin(x)**  **Cos(x)**  **Arctan(x)** | I или R  **Аргумент по умолчанию вводится в радианах** | R |
| **ex**  **ln x** | **Exp(x)**  **Ln(x)** | I или R | R |
|  | **Sqrt(x)** | I или R | R |
| **Число Пи - π** | **Pi** |  | R |
| **Число Экспонента - Е** | **Е** |  | R |
| **Дополнение к стандартным функциям в PascalABC.Net** | | | |
| **xy** | **Power(x,y)** | I или R | R |
| **tg(x)** | **Tan(x)** | I или R | R |
| **Log2(x)**  **Lg(x)**  **LogN(x)** | **Log2(x)**  **Log10(x)**  **LogN(n,x)** | I или R | R |
| **Возвращает знак числа** | **Sign(x)**  Sign(-4)=-1  Sign(7)=1 | I или R | I |
| **Перевод радиан в градусы** | **RadToDeg(x)** | I или R | R |
| **Перевод градусов в радианы** | **DegToRad(x)** | I или R | R |
| **Инициализация датчика случайных чисел** | **Randomize** |  |  |
| **Возвращает случайное целое в диапазоне от a до b** | **Random(a,b)** | I | I |

**Нестандартные функции Pascal**

**Возведение в n-ю степень**

При необходимости вычисления некоторых математических функций, для которых не существует стандартных функций в языке Паскаль, их выражают через имеющиеся стандартные функции.

В Паскале отсутствует операция возведения в степень. Реализация этой операции зависит от типа показателя.

Если показатель целого типа, то операция возведения в степень реализуется многократным умножением.

Например:

**x4 =>x \* x \* x \* x;**

**sin2x => sqr(sin(x)).**

Если показатель вещественного типа, то пользуются соотношением**:**, где x>0; a – вещественное число или выражение. Запись в Турбо Паскале имеет вид**: xa=>exp(a\*ln(x)).**

**Извлечение корня n-ой степени**



Например: .

**Вычисление тангенса угла**

**tg(X)=Sin(X)/Cos(X)**

**Вычисление десятичного логарифма**

**lg(X)=Ln(X)/Ln(10)**

**Приоритет действий при вычислении выражений**

|  |  |  |
| --- | --- | --- |
| Группа | Тип действия | Операции или элементы |
| 1 | Вычисления в круглых скобках | () |
| 2 | Вычисление значений функций | Функции |
| 3 | Унарные операции | Not, ун+, ун- |
| 4 | Операции типа умножения | \* / div mod and shl shr |
| 5 | Операции типа сложения | + - or xor |
| 6 | Операции отношения | = <> < > <= >= in |

***Примеры программирования арифметических выражений***

|  |  |
| --- | --- |
| Математическая  запись | Запись в Турбо Паскале |
|  | (a + 12 \* b)/(c1 - 1.8E3) |
|  | (2\*sin(4\*x)+sqr(cos(x\*x)))/(3\*x) |
|  | Exp((2\*sin(4\*x)+sqr(cos(x\*x)))/(3\*x)) |
| ln(tg α)-sin(α3) | Ln(sin(al)/cos(al))-sin(al\*al\*al) |
| Ln(tg α-sin(α3)) | Ln(sin(al)/cos(al)-sin(al\*al\*al)) |

**Пример 1**

Дано целое двузначное число.

Используя операции div и mod, вывести на экран его левую и правую цифру

**Var N, c1, c2: byte; { Двузначное число }**

**begin**

**write('N = ');**

**readln(N); { <-- вводим число }**

**{ Находим и выводим левую цифру: }**

**C1:= N div 10;**

**writeln(' Левая цифра: ',с1:1);**

**{ Находим и выводим правую цифру: }**

**C2:= N mod 10;**

**writeln(' Правая цифра: ', с2:1);**

**end.**

**Пример 2**

Дано целое трехзначное число. Используя операции div и mod, найти сумму его цифр.

**Var n,sum,n1,n2,n3:integer;**

**Begin**

**Write('Введите 3-х значное число: ');**

**Readln(n);**

**n1:=n mod 10; //единицы**

**n2:=(n div 10) mod 10; //десятки**

**n3:=n div 100; //сотни**

**sum:=n1+n2+n3;**

**Writeln('Сумма цифр числа = ',sum);**

**end.**

**Пример 3**

Вывести синус угла 900

**Var** n,n1:real; //n-угол в градусах, n1-угол в радианах

**Begin**

Write('Введите угол в градусах: ');

Readln(n);

writeln('Синус ',n,' радиан=',sin(n):0:2);

n1:=Degtorad(n);//переводим в радианы для использования в

функции sin()

writeln('Синус ',n,' градусов=',sin(n1):0:2);

writeln('Синус Pi/2=',sin(Pi/2):0:2);//Для проверки

**end**.

**Ответ**

Введите угол в градусах: 90

Синус 90 радиан=0.89

Синус 90 градусов=1.00

Синус Pi/2=1.00

**Практические задание на самостоятельную работу:**

1. По запросу пользователя введите два числа и выведите на экран результат деления первого на второе. После запятой должно быть три знака.
2. Дана длина ребра куба. Найти площадь грани, площадь полной поверхности и объем этого куба.
3. Вычислите значение выражения (a+4b)(a−3b)+a2.
4. Вычислите |x|+x2 (функция abs - модуль).
5. Вычислите значение выражения  ex−2+|sin(x)|−x2⋅cos(x)
6. Сосчитать

20 div 6;

15 mod ( 20 div 3)

Записать алгоритм в общем виде для расчета обоих выражений.

1. Дано целое четырехзначное число. Используя операции div и mod, найти количество и сумму его цифр.
2. Найти периметр и площадь прямоугольной трапеции с основаниями a и b (a > b) и острым углом alpha (угол дан в радианах).
3. Найти расстояние между двумя точками с заданными координатами (x1, y1) и (x2, y2).
4. Даны координаты трех вершин треугольника (x1, y1), (x2, y2), (x3, y3). Найти его периметр и площадь.
5. Пользователь вводит количество недель, месяцев, лет и получает количество дней за это время. Считать, что в месяце 30 дней.
6. Даны две переменных с некоторыми значениями. Поменять местами значения этих переменных. Использовать два алгоритма: с использованием и без использования вспомогательной переменной)
7. Дан прямоугольник размером 647 x 170. Сколько квадратов со стороной 30 можно вырезать из него?
8. Из трехзначного числа x вычли его последнюю цифру. Когда результат разделили на 10, а к частному слева приписали последнюю цифру числа x, то получилось число 237. Найти число x
9. Дана сторона равностороннего треугольника. Найти площадь этого треугольника и радиусы вписанной и описанной окружностей.
10. Дана длина окружности. Найти площадь круга, ограниченного этой окружностью.
11. Дана площадь круга. Найти длину окружности, ограничивающей этот круг.
12. Найти решение системы уравнений вида A·x + By = C, Dx + Ey =F, заданной своими коэффициентами A, B, C, D, E, F, если известно, что данная система имеет единственное решение.
13. Присвоить целой переменной h третью от конца цифру в записи положительного целого числа k (например, если k= 130985, то h=5).
14. Идет k-ая секунда суток. Определить, сколько полных часов (h) и полных минут (m) прошло к этому моменту (например, h=3 и m=40, если k=12357=3\*3600+40\*60+57).
15. Дано вещественное число. Округлить это число до ближайшего целого. Выделить из этого числа дробную и целую часть. Определить знак числа.
16. Вычислить синус, косинус и тангенс угла 450.

**Критерии оценки самостоятельного задания:**

Работоспособный программный код

Оценка «Отлично» ставится за работоспособный оптимальный программный код

.Оценка «Хорошо» ставится за работоспособный оптимальный программный код, имеющий незначительные недочеты.

Оценка «Удовлетворительно» ставится за работоспособный неоптимальный программный код, имеющий незначительные недочеты.

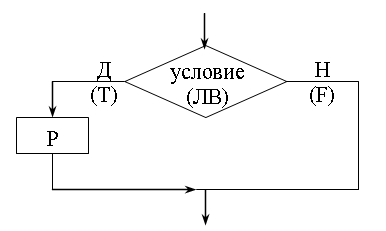
**Условные операторы языка Pascal**

В Паскале различают два вида условных операторов (оператора ветвления): неполный (короткий) и полный.

**Неполный (короткий) оператор ветвления (условный оператор)**

**IF <логическое выражение> THEN <оператор P>;** где **оператор P** – любой оператор.

Сначала вычисляется логическое выражение (ЛВ), и если оно имеет значение **TRUE**, то выполняется оператор **P**, стоящий за служебным словом **THEN**. В противном случае оператор **P** игнорируется и сразу переходим к оператору, который следует за условным оператором.

**Графическая интерпретация оператора.** В блок-схемах короткому условному оператору соответствует структура **ЕСЛИ – ТО.**

****

**Пример**

Вывести на экран квадратный корень целого числа.

**Var а: Integer;**

**Ka: real;**

**Begin**

**WriteLn (‘Введите число’);**

**ReadLn (а);**

**If а >=0 Then ka := sqrt(а);**

**WriteLn (‘koren =', kа);**

**readln**

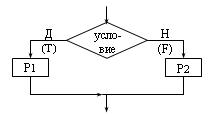
**End.**

**Полный оператор ветвления (условный оператор)**

**IF <логическое выражение> THEN <оператор P1>**

**ELSE <оператор P2>;**

где **оператор P1, оператор P2** – любые операторы.

Вычисляется логическое выражение, и если оно имеет значение **TRUE**, то выполняется оператор **P1**, стоящий после служебного слова **THEN**. В противном случае выполняется оператор **P2**, стоящий после служебного слова **ELSE** .

**Графическая интерпретация оператора.** В блок-схемах полному условному оператору соответствует структура **ЕСЛИ – ТО – ИНАЧЕ**.

**Пример**. Вычислить значение переменной у по одной из двух ветвей:

y = 

**Program P\_1;**

**Var**

**x, y: real;**

**Begin**

**writeln (‘Введите число x’);**

**readln(x);**

**if x>0 then y := ln(x) else y := exp(x);**

**writeln (‘y =’ , y:6:2);**

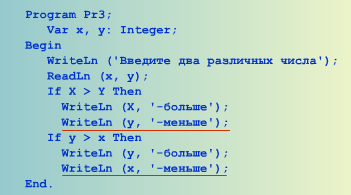
**End.**

**Замечание***.*

По определению в конструкцию короткого условного оператора может входить только один оператор P, а в конструкцию полного условного оператора может входить только один оператор Р1 и один оператор Р2.

**Пример**

Дано два числа. Сравнить эти числа и вывести на экран какое число больше, а какое меньше.



Выполнив эту программу, вы заметите, что подчеркнутые строки выполняются вне зависимости от того, какие числа вы ввели. Такая ошибка связана с тем, что после ключевого слова **Then** в условном операторе **If** выполняется только одна команда, стоящая после него. Остальные же строки выполняются вне зависимости от условных операторов, стоящих перед ними.

Если возникает необходимость выполнить в ветвях несколько операторов, то их заключают в **операторные скобки** **BEGIN…..END,** образуя таким образом единый составной оператор. Операторные скобки охватывают фрагмент исходного текста программы, который необходимо считать одним оператором. В качестве операторов, используемых в составном операторе, могут использоваться любые операторы, допустимые в Турбо-Паскале, в том числе условные операторы, а также составные операторы.

Тогда запись оператора с использованием операторных скобок имеет следующий вид:

**IF <логическое выражение> THEN**

**begin**

Составной

оператор P

**<оператор 1>;**

**………………**

**<оператор n>;**

**end;**

**ELSE**

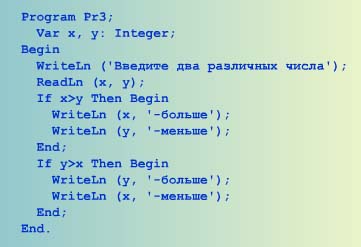
**begin**

**<оператор 1>;**

**……………... P2**

**<оператор m>;**

**end;**

****

**Пример**

Переменной y присвоить минимальное значение из трех различных чисел, т.е.

**y = min (a, b, c).**

**Программа**

**Program P;**

**Var**

**a, b, c, min: real;**

**Begin**

**writeln (‘Введите числа a, b, c’);**

**read (a, b, c);**

**min := a;**

**if b<min then min := b;**

**if c<min then min := c;**

**writeln (‘min =’, min:6:2);**

**readln**

**End.**

**Пример.**

Составить программу, которая выводит на экран компьютера пример на умножение двух однозначных чисел, запрашивает ответ пользователя, проверяет его и выводит сообщение «Правильно» или «Вы ошиблись» и правильный результат.

**Var m1, m2, p, otv: integer;**

**Begin**

**Randomize;**

**M1:=random(9)+1;**

**M2:=random(9)+1;**

**P:=m1\*m2;**

**Writeln (‘Vvedite otvet’)**

**Write (m1,‘ x ’, m2,’=’);**

**Readln (otv);**

**If otv=p Then Writeln (‘Правильно’);**

**Else Writeln (‘Вы ошиблись ’, m1, ‘ x ’, m2,’ =’, p);**

**Readln;**

**End.**

**Оператор выбора**

При многократном вложении условных операторов программа становится громоздкой и ее трудно понять. Считается, что число вложений не должно превышать двух–трех вложений. При большем числе вложений рекомендуется использовать оператор выбора CASE.

**CASE <селектор> OF**

**<список 1>: <оператор 1>;**

**………………………………………………**

**<список n>: <оператор n>;**

**ELSE**

**END;**

**Селектор** – это выражение целого или символьного типа.

**Список** – всевозможные значения селектора, в списке перечисляются через запятую.

По вычисленному значению селектора выбирается для исполнения **оператор** той строки, в списке которой содержится константа, равная значению селектора.

После выполнения выбранного оператора управление передается на конец оператора **CASE**.

Списки могут быть заданы перечислением значений через запятую, например, 2, 5, 10. Или интервалом значений, например, 1..4, 15..67.

**Пример:**

|  |  |
| --- | --- |
| **Вид списка** |  |
| Список, состоящий из одного значения | ‘a’  8 |
| Список, содержащий перечисление значений | 1, 4, 5  ‘a’, ‘b’,’\*’,’@’ |
| Список, содержащий интервалы значений | 1**..**5, 20**..**30  ‘a’ **..**’z’ |
| Список, содержащий смешанную форму представления значений | 1 **..** 5, 15, 55  ‘a’ **..**’z’, ’A’, ’B’ |

**Пример 1:**

Составить программу, которая спрашивает у пользователя номер месяца и выводит соответствующее названия времени года. В случае, если пользователь укажет недопустимое число, программа должна вывести сообщение об ошибке «Ошибка ввода данных».

Ниже приведен рекомендованный вид экрана во время работы программы:

**Введите номер месяца (число от 1 до 12) и нажмите <Enter>**

**-> 12**

**Зима**

Program Pr;

Var month: Integer;

Begin

Writeln (‘Введите номер месяца (число от 1 до 12) и нажмите <Enter>’);

Read (month);

Case month of

1, 2, 12: Writeln(‘Зима’);

3 . . 5: Writeln(‘Весна’);

6 . . 8: Writeln(‘Лето’);

9 . .11: Writeln (‘Осень’);

Else Writeln (‘Ошибка ввода данных – число должно быть от 1 до 12’);

End;

Readln;

End.

**Пример 2.** Написать оператор выбора для вычисления y.



**case n of**

**1: y := x;**

**2, 3: y := 2 \* sqrt(abs(x));**

**4: y := exp(x);**

**end**;

**Замечание**. Если в строке выбора необходимо записать несколько операторов, то их заключают в операторные скобки **BEGIN....END**.

**Пример 3. Вычислить значение y.**



Если значение x не принадлежит рассматриваемым промежуткам, то вывести на экран соответствующее сообщение.

В задаче переменная x является вещественной и не может использоваться в качестве селектора оператора CASE. Введем новую переменную целого типа, которой присваивается целая часть значения x:

**n := trunc(x);**

Тогда программа решения данной задачи с использованием оператора выбора может быть составлена следующим образом.

**Program P\_2;**

**Var**

**x, y: real;**

**n: integer;**

**Begin**

**writeln(‘Введите число x’);**

**read (x);**

**if (x<1) or (x>=5) then**

**writeln(‘x не принадлежит ‘,**

**‘рассматриваемой области’)**

**else**

**begin**

**n := trunc(x);**

**case n of**

**1: y := sin(x);**

**2: y := exp(-x);**

**3: y := ln(x);**

**4: y := sin (x)/cos(x);**

**end;**

**writeln(‘y=’, y:6:2);**

**end;**

**End.**

**Пример 4.** Даны два числа и арифметическая операция. Вывести на экран результат этой операции.

var a,b,s:real;

z:char;

begin

writeln ('Vvedite A = ');

readln(a);

writeln ('Vvedite znak: + - / \* : ' );

readln(z);

writeln ('Vvedite B = ');

readln(b);

case z of

'+': begin s:=a+b; writeln(a,' + ',b,' = ',s); end;

'-': begin s:=a-b; writeln(a,' - ',b,' = ',s); end;

'\*': begin s:=a\*b; writeln(a,' \* ',b,' = ',s); end;

'/': begin s:=a/b; writeln(a,' / ',b,' = ',s); end;

Else

writeln('error: incorrect operation symbol!!!');

end;

readln;

end.

**Самостоятельные задания**

1. Написать программу, которая проверяет, положительно ли число, введенное с клавиатуры.
2. Дано два числа. Вывести наибольшее из них.
3. Написать программу, которая проверяет, четное ли число, введенное с клавиатуры. Применить два способа проверки (через оператор mod и функцию odd()).
4. Написать программу, которая проверяет, делится ли на 3 целое число, введенное с клавиатуры.
5. Дано число. Если оно больше 3, то увеличить число на 10, иначе уменьшить на 10.
6. Дано число. Если оно меньше 7, то вывести Yes, если больше 10, то вывести No, если равно 9, то вывести Error.
7. Дано два числа. Вывести yes, если они отличаются на 100, иначе вывести No.
8. Дано три числа. Найти количество положительных чисел среди них.
9. Даны целые числа a и b (вводить их с клавиатуры после запуска программы).

Определить:

* 1. есть ли среди них отрицательное число; если отрицательное число есть – то вывести на экране произведение чисел a и b, если же отрицательных чисел нет, то на экране вывести сообщение «все введенные числа положительные» (например, если будут введены два числа a и b равные 2 и -7 соответственно, то на экране должно появиться следующее: *произведение чисел равно -14)*
  2. являются ли эти числа четными; вывести на экран только те числа, которые будут четными, либо сообщение «все числа являются нечетными»
  3. найти остаток от деления большего числа на меньшее число и вывести результат этого деления (например, если будут введены два числа a и b равные 14 и 5 соответственно, то на экране должно появиться следующее: *остаток от деления равен 4*)

1. Для данного x вычислить значение следующей функции f, принимающей значения целого типа:

0, если x < 0,

1, если x принадлежит [0,1) и [2,3)

–1 если x принадлежит [1,2) и [3,∞).

1. Дано число. Если оно более 100 или менее -100, то занулить, иначе увеличить его на 1.
2. Вычислите , если x и y вводит пользователь. Перед вычислением выполнить проверку на существование квадратных корней.
3. Пользователь вводит два числа. Если они не равны 10 и первое число четное, то вывести их сумму, иначе вывести их произведение.
4. Пользователь вводит три числа. Если все числа больше 10 и первые два числа делятся на 3, то вывести yes, иначе no
5. Пользователь вводит три числа. Найти сумму тех чисел, которые делятся на 5. Если таких чисел нет, то вывести error.
6. Даны три числа. Найдите наибольшее число из них.
7. Даны три числа. Найдите те два из них, сумма которых наибольшая.
8. Дано число. Если оно от 2 до 5 включительно, то увеличить его на 10. Если оно от 7 до 40, то уменьшить на 100. Если оно не более 0 или более 3000, то увеличить в 3 раза (то есть умножить на 3). Иначе занулить это число.
9. Пользователь вводит четыре числа. Найдите наибольшее четное число среди них. Если оно не существует, выведите фразу "not found"
10. Даны три числа. Написать "yes", если можно взять какие-то два из них и в сумме получить третье
11. Дано четыре числа, если первые два числа больше 5, третье число делится на 6, четвертое число не делится на 3, то вывести yes, иначе no.
12. Дана дата из трех чисел (день, месяц и год). Вывести yes, если такая дата существует (например, 12 02 1999 - yes, 22 13 2001 - no). Считать, что в феврале всегда 28 дней.
13. Дано две даты, каждая из которых состоит из трех чисел (день, месяц и год). Вывести yes, если первая дата раньше второй, иначе вывести no.
14. Значения переменных X, Y, Z поменять местами так, чтобы они оказались упорядоченными по убыванию.
15. Даны три переменные: X, Y, Z. Если их значения упорядочены по убыванию, то удвоить их; в противном случае заменить значение каждой переменной на противоположное
16. Дано четырехзначное число. Образуют ли цифры этого числа убывающую последовательность? Например, 4311 – не бразуют, 4321 - да, 5542 - нет, 5631 - нет, 9871 - да.
17. Дано трехзначное число. Переставьте первую и последнюю цифры.
18. Дано четырехзначное число. Определите, есть ли одинаковые цифры в нем.
19. Даны два трехзначных числа. Найдите шестизначное число, образованное из двух данных чисел путем дописывания второго числа к первому справа.
20. Дано четырехзначное число. Если оно читается слева направо и справа налево одинаково, то вывести yes, иначе no.
21. Даны два трехзначных числа. Получите новое число присоединением второго числа справа к первому без последних цифр у каждого. Например, 123 и 456 Ответ: 1245
22. Дано четырехзначное число. Поменяйте местами наименьшую и наибольшую цифры.
23. Даны коэффициенты a,b,c уравнения ax2+bx+c=0. Найти решение.
24. Пользователь вводит три числа - длины сторон треугольника. Найти площадь треугольника. Сделать проверку на существование треугольника (например, 1, 2, 3 - такого треугольника не существует).
25. Даны целочисленные координаты трех вершин прямоугольника. стороны которого параллельны координатным осям. Найдите координаты его четвертой вершины (после проверки введенных данных на правильность).
26. Вычислить двумя способами, используя короткий оператор IF и полный вложенный оператор IF:



Ответы

|  |  |
| --- | --- |
| х | f |
| -10 | -11.000 |
| -5 | -6.000 |
| -1 | 1.683 |
| 0 | 0.000 |
| 2 | 3.637 |
| 5 | 0.598 |
| 10 | 0.754 |

1. Вычислить, используя полный вложенный оператор IF. Числа a, b, c вводятся с клавиатуры – вещественные.



Ответы

При а=0.54, b=2.9, с=5

|  |  |
| --- | --- |
| х | у |
| -5 | 77.392 |
| -1 | 7.360 |
| 0 | Решений нет |
| 2 | 16.870 |
| 5 | -654.588 |
| 10 | -9200.746 |

1. Вычислить и вывести на экран значения функции F в зависимости от введенного х



где x, a, b – вещественные числа – задаются с клавиатуры

Ответы

При а=15.1, b=-4.6

|  |  |
| --- | --- |
| х | f |
| -7 | -0,103 |
| -3 | 0,522 |
| -1 | 5,467 |
| 0 | -0,199 |
| 5 | -0,588 |
| 7,55 | Решений нет |

1. Вычислить и вывести на экран значения функции F в зависимости от введенного х



где x, a, b, c – вещественные числа – задаются с клавиатуры

1. Определить значение U по формуле:

Значения x, y, z – вещественные - задаются с клавиатуры.

1. Пользователь вводит номер месяца, вывести название месяца.
2. Пользователь вводит номер месяца. Вывести название времени года (весна, лето и т.д.)
3. Составить программу, которая после введенного с клавиатуры числа (в диапазоне от 1 до 99), обозначающего денежную единицу, дописывает слово «Копейка» в правильной форме. Например, 5 копеек, 41 копейка и т. д.
4. Составить программу, которая в зависимости от порядкового номера дня недели (1, 2, 3, … 7) выводит на экране его название (понедельник, вторник, среда, …, воскресенье). Если пользователь введет неправильный порядковый номер дня недели, то на экране отобразится сообщение «Ошибка в порядковом номере дня недели»
5. Дано целое число в диапазоне 100 – 999. Вывести строку — словесное описание данного числа, например: 256 — "двести пятьдесят шесть", 814 — "восемьсот четырнадцать"
6. Элементы окружности пронумерованы следующим образом: 1 — радиус (R), 2 — диаметр (D), 3 — длина (L), 4 — площадь круга (S). Дан номер одного из этих элементов и его значение. Вывести значения остальных элементов данной окружности (в том же порядке). В качестве значения Pi использовать 3.14.
7. Арифметические действия над числами пронумерованы следующим образом: 1 — сложение, 2 — вычитание, 3 — умножение, 4 — деление. Дан номер действия и два числа A и B (В не равно нулю). Выполнить над числами указанное действие и вывести результат.

**Критерии оценки самостоятельного задания:**

Работоспособный программный код

Оценка «Отлично» ставится за работоспособный оптимальный программный код

.Оценка «Хорошо» ставится за работоспособный оптимальный программный код, имеющий незначительные недочеты.

Оценка «Удовлетворительно» ставится за работоспособный неоптимальный программный код, имеющий незначительные недочеты.

**Циклы Pascal-Паскаль**

При решении задач может возникнуть необходимость повторить одни и те же действия несколько или множество раз. В программировании блоки кода, которые требуется повторять не единожды, оборачиваются в специальные конструкции – ***циклы***.

**Циклы позволяют многократно выполнять одну или группу команд, причем в тексте программы нет необходимости записывать эти команды несколько раз.**

Можно выделить три **оператора цикла**, присутствующих в том или ином виде во всех языках программирования:

1. **арифметический оператор цикла** (с параметром с шагом 1 и с параметром произвольного шага),
2. **итерационный оператор цикла с предусловием**,
3. **итерационный оператор цикла с постусловием**.

**Арифметический оператор цикла For (цикл с параметром)**

Рассмотрим задачу вычисления суммы первых n членов гармонического ряда:

Y= 1+ 1/2+ 1/3 + …+ 1/ n

При n=5 требуемые вычисления можно задать с помощью оператора присваивания вида:

Y:= 1+1/2+1/3+1/4+1/5

Если значение n заранее не известно и вводится в процессе выполнения программы, то аналогичный оператор присваивания записать невозможно, так как запись вида Y:=  1+1/2+1/3+…+1/ n в языках программирования недопустима.

На самом деле вычисление этой суммы можно осуществить по очень простому и компактному алгоритму: предварительно положим y=0 (с помощью оператора присваивания y:=0), а затем выполним оператор присваивания y:= y+1/ i для последовательных значений i= 1,2,…, n. При каждом очередном выполнении этого оператора к текущему значению y будет прибавляться очередное слагаемое. Как видно, в этом случае процесс вычислений будет носить циклический характер: оператор **y:= y+1/i** должен выполняться многократно, т.е. циклически, при различных значениях i.

Этот пример циклического вычислительного процесса является весьма типичным; его характерные особенности состоят в том, что

* число повторений цикла известно к началу его выполнения (в данном случае оно равно значению **n**, которое предполагается заданным к этому времени);
* управление циклом осуществляется с помощью переменной порядкового типа, которая в этом циклическом процессе принимает последовательные значения от заданного начального до заданного конечного значений (в нашем случае – это целочисленная переменная i, принимающая последовательные значения от 1 до n).

Для компактного задания подобного рода вычислительных процессов и служит **арифметический** **оператор цикла с параметром For**.

Цикл **For** существует в двух формах:

**For** **параметр:=**начальное**\_**значение **to** конечное\_значение **do**

тело\_цикла;

**For** **параметр:=** начальное\_значение **downto** конечное\_значение **do**

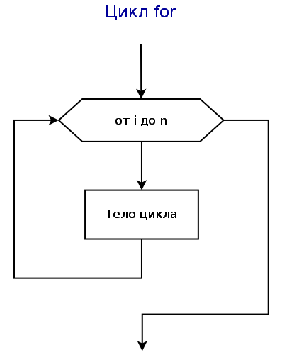
тело\_цикла;

**For** (для), **to** (увеличиваясь к) и **do** (выполнять, делать) – служебные слова,

**параметр** – переменная порядкового типа, первоначально принимающая начальное значение. Если между начальным и конечным значением указано служебное слово **to**, то на каждом шаге цикла значение параметра будет увеличиваться на единицу. Если же указано **downto**, то значение параметра будет уменьшаться на единицу. Иногда параметр цикла называют **счетчик** цикла

Начальные и конечные значения могут быть не только числами, но и выражениями.

**Тело цикла** - любые операторы Паскаль. Заметим, что в Паскале после **do** должен стоять один оператор, если необходимо выполнить несколько действий, то они должны быть объединены в один составной оператор путем заключения в операторные скобки.



**Особенностью арифметического цикла является то, что число повторений операторов цикла должно быть известно заранее. Решение о выполнении или невыполнении в очередной раз тела цикла принимается до начала его прохождения, поэтому может случиться так, что тело цикла не будет выполнено ни разу.**

Алгоритм выполнения цикла For

1. Вычисляются начальное и конечное значения
2. Параметру цикла присваивается начальное значение
3. Если полученное значение параметра меньше (при to) (больше меньше при downto) конечного значения параметра цикла, то выполняется тело цикла, в противном случае выполнение цикла прекращается.
4. Значение параметра цикла увеличивается (уменьшается) на 1 и переходим к пункту 3.

Пример:

С использованием **оператора цикла с параметром** вычислить суммы первых n членов ряда Y= 1+ 1/2+ 1/3 + …+ 1/ n

Var i:Word;

N,y:integer;

begin

Readln(n);

Y:= 0;

For i:= 1 to n do y:= y+1/i;

Если начальное и конечное значения равны, то тело цикла выполнится один раз. Количество повторений тела цикла можно вычислить по формуле

Конечное\_значение – Начальное\_значение + 1

Количество выполнений тела цикла определяется до первого выполнения тела цикла, поэтому, если в выражении n2 используются переменные, которые изменяются в теле цикла, это никак не повлияет на количество выполнений тела цикла.

**Например**, в следующей последовательности операторов:

n := 10;

For i := 1 to n do

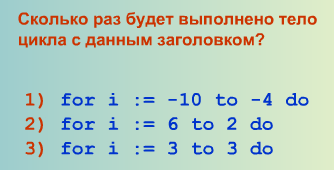
Begin

Write (i);

n := 100;

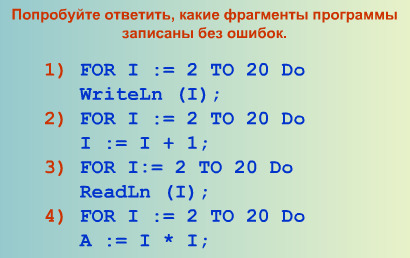
End;

тело цикла будет выполняться 10 раз, хотя уже после первого выполнения тела цикла конечное значение параметра цикла станет равным 100.



Заметим так же, что **параметр цикла** может и не использоваться в теле цикла, так что основное его назначение – это управление числом повторений цикла.

Значение счетчика цикла может использоваться в выражениях, входящих в операторы тела цикла, но изменение значения счетчика цикла этими операторами недопустимо



Правильно записаны фрагменты под номерами 1 и 4. Во фрагментах 2 и 3 в операторах тела цикла производится изменение значение счетчика цикла.

**Пример1:**

Вывести на экран значения N!.

Var i:Word;

N,y:integer;

begin

Readln(n);

Y:= 1;

For i:= 1 to n do y:= y\*i;

Writeln(y:5);

**Пример 2:**

Вывести на экран значение y и y2 (y=1,2,…,10) в возрастающем и убывающем порядке

Var kw, y: integer;

Begin

For y:=1 to 10 do

Begin

kw:= sqr(y);

Writeln (y:3,kw:5);

End;

Writeln;

For y:=10 downto 1 do

Begin

kw:=sqr(y);

Writeln (y:3,kw:5);

End;

Readln;

End.

**Пример 3:**

Выбрать наименьшие значение из 10 чисел, вводимых с клавиатуры

var i:word;

x,y : real;

begin

writeln ('Vvedite 1-e chislo');

readln (x);

for i:=2 to 10 do

begin

write ('Vvedite ‘,I,’-e chislo');

readln (y);

if y<x then x:= y

end;

writeln ('min=',x:6:2);

readln;

end.

**Пример 4:**

Среди всех двузначных чисел найти те, сумма цифр которых равна n (0<n≤18) и количество этих чисел.

**var n, i, k, p1, p2:integer;**

**begin**

**k:=0;**

**writeln (‘N=’);**

**readln (n);**

**for i:=10 to 99 do**

**begin**

**p1:=i div 10;**

**p2:= i mod 10;**

**if (p1+ p2) =n then**

**begin**

**writeln (i);**

**k:=k+1;**

**end;**

**end;**

**writeln (‘Kolichestvo=’, k);**

**readln;**

**end.**

**Итерационные циклы Паскаль**

При составлении программ часто возникают ситуации, когда

1. Заранее не известно количество повторений цикла;

2. Переменная – счетчик цикла должна изменяться с шагом ≠ 1.

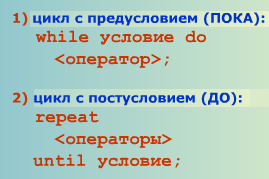
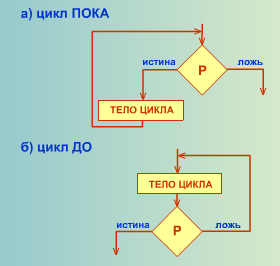
В таких случаях используют итерационные циклы.

Итерационные циклы отличаются от циклов с параметром тем, что в них заранее неизвестно число повторений.

Итерационные циклы - это циклическое повторение блока команд, пока выполняется (или не выполняется) некоторое условие. Отсюда получаются два варианта реализации итерационных циклов: с предусловием и с постусловием.

В цикле с предусловием сначала проверяется условие, а потом делается шаг. В цикле с постусловием – сначала шаг, а потом проверка.

**При использовании логических циклов необходимо всегда заботиться о том, чтобы переменная, использующаяся в условии завершения цикла, каким-то образом обязательно изменялась в теле цикла. Это может быть команда присваивания или команда ввода.**



**Оператор цикла Паскаля с предусловием**

**While <** логическое выражение **> do Р,**

где **while** (пока), **do** (делать, выполнять) – служебные слова, Р – оператор. Здесь оператор Р выполняется ноль или более раз, но перед каждым очередным его выполнением вычисляется значение логического выражения, и оператор Р выполняется только в том случае, когда значение логического выражения =true.

Выполнение оператора цикла завершается, когда логическое выражение впервые принимает значение false. Если это значение выражение В принимает при первом же его вычислении, то оператор S не выполнится ни разу.

**Пример 1**



При каком количестве слагаемых последовательности

их сумма превысит 10?

sum:=0;

k:=1;

While sum<=10 do

Begin

Sum:=sum+1/k;

k:=k+1;

end;

Writeln('Kolichestvo elementov=',k);

readln;

end.

Оператор цикла Паскаля с предусловием можно считать наиболее универсальным – с использованием таких операторов можно задать и циклические процессы, определяемые операторами цикла с параметром и постусловием.

**Оператор цикла Паскаля с постусловием**

Рассмотрим теперь математическую задачу. Пусть нам необходимо вычислить сумму первых членов гармонического ряда, удовлетворяющих условию 1/i>= K, где 0< K<1, а i=1,2,3…...

Эту задачу можно решить по следующему алгоритму:

положить предварительно y=0 и i=0,

затем в цикле увеличивать i на 1,

к значению y добавлять очередное слагаемое 1/ i до тех пор, пока текущее значение 1/ i впервые окажется больше заданного значения 0< K<1.

Очевидно, что число повторений этого цикла заранее не известно. В подобного рода случаях мы можем лишь сформулировать условие, при выполнении которого процесс добавления к сумме очередного слагаемого должен завершиться. Для задания таких вычислительных процессов и служит оператор цикла Паскаля с постусловием. Этот оператор имеет вид:

**Repeat P1; P2;…; Pi until <** логическое выражение **>,**

где **repeat** (повторять) и **until** (до) – служебные слова, через **Si** обозначен любой оператор Паскаля.

При выполнении этого оператора цикла последовательность операторов, находящихся между словами **repeat** и **until**, выполнится один или более раз. Этот процесс завершается, когда после очередного выполнения заданной последовательности операторов логическое выражение примет (**впервые**) значение **true**. Таким образом, с помощью логического выражения задается условие завершения выполнения оператора цикла. Поскольку в данном случае проверка условия производится после выполнения последовательности операторов (тела цикла), этот оператор цикла и называется оператором цикла с постусловием.

С использованием этого вида оператора цикла Паскаля задача о суммировании первых членов гармонического ряда, удовлетворяющих заданному условию, может быть реализована следующим образом:

**Пример 2**

readln(К);

i:=0;

y:=0;

Repeat

i:=i+1;

y:=y+1/i;

Until 1/i<К;

Какой алгоритм выбрать? Это зависит от конкретной задачи.

Если, сделав шаг без проверки, можно получить ошибку, то лучше проверка вначале. Ну, а если шаг без проверки вас не пугает, то можно отложить ее до завершения шага.

Нужно также проанализировать событие, которого мы ожидаем. Если оно может случиться до первого шага, то нужен цикл с предусловием. А если событие не может случиться до первого шага, то нужен цикл с постусловием.

**Сравнение циклов**

|  |  |
| --- | --- |
| **Цикл с предусловием While (пока условие истинно)** | **Цикл с постусловием Repeat (до истинности условия)** |
| * До начала цикла должны быть сделаны начальные установки переменных, управляющих условием цикла, для корректного входа в цикл | |
| 1. В теле цикла должны присутствовать операторы, изменяющие переменные условия так, чтобы цикл через некоторое число итераций завершился | |
| 1. Проверка условия выполнения цикла находится в начале цикла | 1. Проверка условия выполнения цикла находится в конце цикла |
| 1. Цикл работает пока условие истинно (пока True) | 1. Цикл работает пока условие ложно (пока False) |
| 1. Цикл завершается, когда условие становится ложным (до False) | 1. Цикл завершается, когда условие становится истинным (до True) |
| 1. Цикл может не выполниться ни разу, если исходное значение условия при входе в цикл False | 1. Цикл обязательно выполнится как минимум один раз |
| 1. Если в теле цикла требуется выполнить более одного оператора, то необходимо использовать составной оператор | 1. Независимо от количества операторов в теле цикла, использование составного оператора не требуется |
| **Цикл со счетчиком (с параметром) For** | |
| * Начальная установка переменной счетчика цикла до заголовка не требуется | |
| * Изменение в теле цикла значений переменных, стоящих в заголовке не допускается | |
| * Количество итераций цикла неизменно и точно определяется значениями нижней и верхней границ и шага приращения | |
| * Нормальный ход работы цикла может быть нарушен оператором goto или процедурами Break и Continue | |
| * Цикл может не выполниться ни разу, если шаг цикла будет изменять значение счетчика от нижней границы в направлении, противоположном верхней границе | |

**Пример 3**

Даны числа a, b (a > 1). Получить все члены последовательности a, 2a, 3a, … na, пока na <b

При выполнении этой программы переменная el последовательно принимает значение a, 2a, 3a … . Изменение el происходит до тех пор, пока оно не станет больше b. Если а>b, то не будет выводить ни одного значения.

var a,b,el : real;

begin

writeln ( ‘ Введите числа a и b ’ );

readln ( a, b );

el := a;

while el < b do

begin

writeln (el:8:2) ;

el := el\*a;

end;

readln;

end.

**Пример 4. Алгоритм Евклида**

Это алгоритм вычисления наибольшего общего делителя (НОД) двух целых положительных чисел. Он основан на следующих свойствах величин: пусть a и b-одновременно ≠ 0, и пусть a≥ b.

Если b=0, то НОД(a,b) = a,

Если b ≠ 0, то для чисел a,b,c, где c- остаток от деления a на b (c=a mod b) выполнено равенство: НОД(a,b)=НОД(b,c),

Например:

НОД(15,6)=НОД(6,3)=НОД(3,0)=3.

Var a,b,c: integer;

Begin

Writeln (‘Enter a, b. (a>b)’);

readln( a, b);

While b>0 do

Begin

c:= a MOD b;

a:= b;

b:= c;

end;

Writeln (‘NOD(a,b)= ‘,a);

Readln;

**end**.

**Пример 5.**

Вывести на экран значения х и функции y(х)=3x2-10 в виде таблицы для каждого значения х. Х меняется на интервале от х1 до х2 с шагом h . Значения х1, х2, и h вводятся с клавиатуры.

var x,y,x1,x2,h: real;

BEGIN

Readln(x1,x2,h);

x:=x1;

writeln('----------------------------');

writeln('| X | Y | ');

writeln('----------------------------');

writeln;

repeat

  y:=3\*sqr(x)-10;

  writeln('        |   ',x:5:2,'   |    ',y:5:2,'    | ');

  x:=x+h;

until x>x2;

writeln('        ---------------------------');

readln;

END.

**Пример 6.**

С клавиатуры вводится натуральное число. Вывести в столбик каждую цифру этого числа и ее номер, а также количество цифр в числе.

var x, y, c: longint;

r, k: integer;

Begin

clrscr;

Writeln ('Enter zeloe chislo do 10 snakov');

Readln (x);

writeln;

y:=x;

k:=0;

Writeln ('Zifry zadannogo chisla:');

repeat

r:=y mod 10;

c:=y div 10;

k:=k+1;

y:=c;

Writeln (k,' zifra = ',r);

until y<1;

Writeln ('Chislo - ',x,' coderjit ', k, ' zifr');

readln;

end.

**Пример 7**



Вычислить и вывести на экран сумму ряда:

с точностью абсолютного значения последнего члена ряда **ε**. При расчете использовать [цикл](http://moodle/mod/glossary/showentry.php?courseid=198&concept=%D0%A6%D0%B8%D0%BA%D0%BB) **While**

readln(b,eps);

sum:=0;

el:=1/b;

k:=1;

While abs(el)>=eps do begin

If k mod 2 =0 then Sum:=sum+el

Else Sum:=sum-el;

el:=el/b;

k:=k+1;

end;

Writeln('s=',sum);

readln;

end.

**Пример 8**

Использовать цикл **Repeat**.

Дано вещественное число X и целое число N (> 0). Сосчитать сумму ряда  c точностью абсолютного значения разности последних членов ε.

var x,s,elpr,elsl,sum,eps: real;

i,z: integer;

Begin

Readln(x,s,eps);

i:=1;

elpr:=x\*x/(s\*2);

sum:= 1-elpr;

repeat

elsl:=elpr\*x\*x/(i);

if I mod 2=0 then sum:=sum+elsl

else sum:=sum-elsl;

elpr:=elsl;

i:=i+1;

until abs(elpr-elsl)<eps {до тех пор, пока элемент в разложении не меньше точности}

Writeln('sum = ', sum);

End.

**Операторы завершения цикла**

Данные операторы применяются внутри операторов цикла с параметром или условием. Операторы имеют вид: **Continue**; - ограничение цикла, **Break**; - прерывание цикла.

Операторы Continue и Break позволяют производить действия не для всех операторов внутри цикла. Действие оператора Continue заключается в передаче управления на начало цикла, при этом контролируется условие выхода из цикла. Действие оператора Break заключается в передаче управления оператору, следующему за последним оператором цикла, при этом не контролируется условие выхода из цикла.

Во вложенных циклах операторы Continue и Break действуют только на цикл в котором они записаны. Приведем пример использования операторов для блокировки несанкционированного доступа в программу.

**Пример 9**

Выписать все простые числа, не превышающие данного числа N.

**Var**

X, N, i:integer; // проверяемое число, указанное число , счетчик

b:boolean;{true-число простое false-число не простое}

**Begin**

x:=2; //задаём начальное значение

Write('N=');

readln(N);

Writeln(1); // выводим 1

**While** x<=N **do begin**

b:=true; // предпологаем что число простое}

**for** i:=2 **to** x-1 **do** // перебираем все числа от 2 до x

**begin**

**if** (x **mod** i)=0 **then** {проверяем остаток от деления если число делится без остатка значит оно не простое и цикл можно прервать}

**begin**

b:=false;

**break**;

**end**;

**end**;

**if** b=true **then** Writeln(x);

X:=X+1;

**end**

**end**.

**Самостоятельные задания**

**1.** Решить задачу с использованием цикла Repeat: Вывести на экран числа следующим образом:

50, 45, 40, 35, … 5

Обратите внимание, что числа выводятся в строку через запятую!

**2.** Решить задачу с использованием цикла While: Задать интервал целых чисел от A до B. Вывести на экран все четные числа в интервале от A до B. Числа вывести в стобец!

**3**. Задание выполнить с использованием цикла For. Вычислить и вывести на экран в виде таблицы значения функции  на интервале от  до  с шагом .



где x, dx, a, b, c – вещественные числа.

**4**. Задание выполнить с использованием любого цикла:

В первый день тренировок велосипедист проехал S км. Каждый следующий день он будет увеличивать пробег на D км (по сравнению с предыдущим днем). Определить и вывести на экран пробег велосипедиста за второй, третий, … , десятый день тренировок. Данные вывести в столбец.

1. Вывести на экран ряд чисел 1001,  1002,  1003,  ... 1025.
2. Вывести на экран числа 20, 19, 18, ... до последнего положительного включительно.
3. Дано вещественное число A и целое число N (> 0). Вывести все целые степени числа A от 1 до N.
4. Выведите на экран 10 раз фразу "Hello, world!"
5. Даны натуральные числа от 20 до 50.Напечатать те из них, которые делятся на 3, но не делятся на 5.
6. Вывести на экран значения функции из промежутка [a,b]. Промежуток должен включать промежуток [1;3]



1. Вывести на экран значения функции  из промежутка [a,b].
2. Найти сумму   1+4+7+11+...+112.
3. Найти сумму cos35+cos57+cos79+...+cos111113.
4. Найдите сумму квадратов первых n натуральных чисел
5. Дано натуральное число n. Вычислите cosx+cosx2+…+cosxn
6. Дано число D (> 0). Последовательность чисел AN определяется следующим образом: A1 = 2, AN = 2 + 1/AN–1, N = 2, 3, ... Найти первый из номеров K, для которых выполняется условие |AK – AK–1| < D, и вывести этот номер, а также числа AK–1 и AK.
7. Вычислить сумму n чисел, вводимых с клавиатуры.
8. Написать программу, которая находит сумму целых положительных чисел от А, которые больше 20, меньше 100, кратны 3 и заканчиваются на 2 или 8.
9. В ЭВМ вводятся по очереди данные о росте N учащихся класса. Определить средний рост учащихся класса.
10. Найти произведение 5\*6\*7\*...\*13.
11. Начав тренировки, лыжник в первый день пробежал 5 км. Каждый следующий день он увеличивал пробег на 2 км. Определите: а) пробег лыжника за второй, третий, ..., десятый день тренировок; б) какой суммарный путь он пробежал за первые 7 дней тренировок.
12. Задано натуральное число N. Найти количество натуральных чисел, не превосходящих N и не делящихся ни на одно из чисел 2,3,5.
13. Составьте программу получения в порядке убывания всех делителей данного числа. Подсчитайте их количество
14. Составьте программу, определяющую максимальное и минимальное значение из всех вводимых вами N чисел
15. Найти наибольшее и наименьшее значение функции y=3x2+x-4, х принимает целые значения на интервале [a,b]
16. С клавиатуры вводится 10 различных произвольных чисел.   
    а) Найти разность между наибольшим и наименьшим из них.  
    б) Порядковый номер наибольшего и наименьшего числа. Использовать счетный цикл

**Критерии оценки самостоятельного задания:**

Работоспособный программный код

Оценка «Отлично» ставится за работоспособный оптимальный программный код

.Оценка «Хорошо» ставится за работоспособный оптимальный программный код, имеющий незначительные недочеты.

Оценка «Удовлетворительно» ставится за работоспособный неоптимальный программный код, имеющий незначительные недочеты.

## Раздел 3. Структурированные типы данных

**Тема 3.1.  Понятие массива**

**Методические рекомендации по изучению лекций и литературы**

Ключевыми для данной темы являются следующие понятия: одномерный массив, двумерны массив, индекс.

Студенты должны знать категории задач, решаемых при помощи структуры массив.

Студенты должны знать синтаксис операторов, объявляющих массив.

**ОДНОМЕРНЫЕ МАССИВЫ**

Предположим, что программа работает с большим количеством однотипных данных. Скажем около ста разных целых чисел нужно обработать, выполнив над ними те или иные вычисления. Как вы себе представляете 100 переменных в программе? И для каждой переменной нужно написать одно и тоже выражение вычисления значения? Это очень неэффективно.

Есть более простое решение. Это использование такой структуры (типа) данных как **массив**. Массив представляет собой последовательность ячеек памяти, в которых хранятся однотипные данные. При этом существует всего одно имя переменной связанной с массивом, а обращение к конкретной ячейке происходит по ее индексу (номеру) в массиве.

Нужно четко понимать, что индекс ячейки массива **не** является ее содержимым. Содержимым являются хранимые в ячейках данные, а индексы только указывают на них. Действия в программе над массивом осуществляются путем использования имени переменной, связанной с областью данных, отведенной под массив.

Итак, **Массив — это структура данных, представляющая собой набор переменных одинакового типа, имеющих общее имя.**

Массив – это именованная группа однотипных данных, хранящихся в последовательных ячейках памяти. Каждая ячейка содержит элемент массива. Элементы нумеруются по порядку, но необязательно начиная с единицы (хотя в языке программирования Pascal чаще всего именно с нее). Порядковый номер элемента массива называется индексом этого элемента.

Массивы состоят из ограниченного числа компонент. Все элементы одного массива имеют один и тот же тип. У разных массивов типы данных могут различаться. Например, один массив может состоять из чисел типа integer, а другой – из чисел типа real.

Индексы элементов массива обычно целые числа. Индексировать можно как константами и переменными, так и выражениями, результат вычисления которых дает значение перечислимого типа.

Массив в Delphi обозначается как **array**. Чтобы объявить переменную типа массива нужно описать её в разделе **var** следующим образом:

var Имя\_массива: array [длина массива] of тип данных;

где:

имя — имя массива;

array — зарезервированное слово языка Delphi, обозначающее, что объявляемое имя является именем массива;

длина массива — количество элементов (размер) массива, задается в [], в которых объявляется начальный и конечный индекс массива.

тип — тип элементов массива.

Примеры объявления массивов:

temp:array[1..31] of real;

кf:array [0. .2] of integer;

name:array[1..30] of char;

Обращение к определенному элементу массива осуществляется путем указания имени переменной массива и в квадратных скобках индекса элемента. Номер элемента массива называется **индексом. Индекс** – это значение порядкового типа, определенного, как тип индекса данного массива. Очень часто это целочисленный тип.

С помощью индекса массива можно обращаться к отдельным элементам любого массива, как к обычной переменной: можно получать значение этого элемента, отдельно присваивать ему значение, использовать его в выражениях.

Индекс массива в Паскале не обязательно задавать в явном виде. В качестве индекса массива можно использовать переменную или выражение, соответствующее индексному типу. Иначе говоря, индексы можно вычислять.

Этот механизм – весьма мощное средство программирования. Но он порождает распространенную ошибку: результат вычислений может оказаться за пределами интервала допустимых значений индекса, то есть будет произведена попытка обратиться к элементу, которого не существует. Эта типичная ошибка называется «выход за пределы массива».

Простой массив является одномерным. Он представляет собой линейную структуру.

**Ввод массива Паскаля**

Для того чтобы ввести значения элементов массива, необходимо последовательно изменять значение индекса, начиная с первого до последнего, и вводить соответствующий элемент. Для реализации этих действий удобно использовать цикл с заданным числом повторений, т.е. простой арифметический цикл, где параметром цикла будет выступать переменная – индекс массива Паскаля. Значения элементов могут быть введены с клавиатуры или определены с помощью оператора присваивания.

Пример фрагмента программы ввода массива Паскаля

program massiv1;

Var

A : array [1..10] of integer ;

I : byte; {переменная I вводится как индекс массива}

Begin

for i:=1 to 10 do readln (a[i]);

for i := 1 to 10 do write (a[i]:3);

readln

end.

Рассмотрим теперь случай, когда массив Паскаля заполняется автоматически случайными числами, для этого будем использовать функцию **random (N)**.

Процедура **Randomize** используется в Паскаль для включения генератора случайных чисел. Функция **Random** определяет диапазон случайных чисел. Процедура Randomize и функция Random очень часто используются для демонстрации работы массивов в Паскаль.

Пример фрагмента программы заполнения массива Паскаля случайными числами

program massiv2;

var

A : array [1..15] of integer ;

I : byte; {переменная I вводится как индекс массива}

begin

randomize;

for i:=1 to 15 do

begin

a[i]:=random(10); { i -му элементу массива присваивается «случайное» целое число в диапазоне от 0 до 10}

writeln(‘a[‘,I,’]=’,a[i]:2);

end;

readln

end.

**Основные действия с массивами Паскаля**

Как известно, определение типа данных означает ограничение области допустимых значений, внутреннее представление в ЭВМ, а также набор допустимых операций над данными этого типа. Мы определили тип данных как массив Паскаля. Какие же операции определены над этим типом данных? **Единственное действие, которое можно выполнять над массивами целиком, причем только при условии, что массивы однотипны, – это присваивание**. Если в программе описаны два массива одного типа, например,

Var

a , b : array [1..10] of real ;

то можно элементу массива a присвоить значение соответствующего элемента массива b

a := b;

**Все остальные действия над массивами Паскаля производятся поэлементно (это важно!).**

**Вывод массива Паскаля**

Вывод массива в Паскале осуществляется также поэлементно, в цикле, где параметром выступает индекс массива, принимая последовательно все значения от первого до последнего.

Вывод можно осуществить и в столбик с указанием соответствующего индекса. Но в таком случае нужно учитывать, что при большой размерности массива все элементы могут не поместиться на экране и будет происходить скроллинг, т.е. при заполнении всех строк экрана будет печататься очередной элемент, а верхний смещаться за пределы экрана.

При выполнении программы massiv2 на экране мы увидим, к примеру, следующие значения:

a [1]=2

a [2]=4

a [3]=1 и т.д.

Функция sizeof, примененная к имени массива или имени массивного типа, возвращает количество байтов, отводимое под массив.

Пример программы с ошибкой массива Паскаля

Program massive\_error ;

var

A : array [1..6] of integer ;

I : byte; {переменная I вводится как индекс массива}

begin

i:=3;

a[i\*3]:=25;

end .

Хотя данная программа полностью соответствует синтаксису языка, и транслятор «пропустит» ее, на стадии выполнения произойдет ошибка выхода за пределы массива Паскаля. При i:=5 выражение i\*3=9, компьютер сделает попытку обратиться к элементу массива a [9], но такого элемента нет, поскольку описан массив размерностью 6.

Будем считать, что хорошая программа должна выдавать предупреждающее сообщение в случае попытки обращения к несуществующим элементам массива. Не лишним будет проверять возможный выход как за правую, так и за левую границы массива, ведь не исключено, что в результате вычисления значения выражения получится число, находящееся левее границы массива Паскаля.

Из всего этого следует сделать вывод: программисту надо быть очень аккуратным при работе с индексами массива.

Пример: Требуется заполнить массив числами, которые вводит пользователь, и вычислить их сумму. Если пользователь вводит ноль или превышен размер массива, то запросы на ввод должны прекратиться.

**Алгоритм решения задачи:**

Для решения подобного рода задач иногда бывает уместно использовать цикл с постусловием (repeat).

Программа на языке Паскаль:

program massiv3;

const n = 10;

var

arr: array[1..n] of integer;

sum: integer;

i: byte;

begin

i := 1;

sum := 0;

repeat

write (I,'-e число = ');

readln(arr[i]);

sum := sum + arr[i];

i := i + 1

until (arr[i-1] = 0) or (i > n);

writeln

for i := 1 to n do

write (arr[i],' ');

writeln;

writeln ('sum = ', sum);

readln

end.

**Двумерные массивы**

Двумерный массив в Паскале трактуется как одномерный массив, тип элементов которого также является массивом (массив массивов). Положение элементов в двумерных массивах Паскаля описывается двумя индексами. Их можно представить в виде прямоугольной таблицы или матрицы.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A[1,1] | A[1,2] | A[1,3] | A[1,4] | A[1,5] |
| A[2,1] | A[2,2] | A[2,3] | A[2,4] | A[2,5] |
| A[3,1] | A[3,2] | A[3,3] | A[3,4] | A[3,5] |
| A[4,1] | A[4,2] | A[4,3] | A[4,4] | A[4,5] |

Каждый элемент имеет свой номер, как у одномерных массивов, но сейчас номер уже состоит из двух чисел – номера строки, в которой находится элемент, и номера столбца. Таким образом, номер элемента определяется пересечением строки и столбца. Например, a[2,1] – это элемент, стоящий во второй строке и в первом столбце.

Описание двумерного массива Паскаля.

**Существует несколько способов объявления двумерного массива Паскаля**

Привычное простое нам объявление массива

Var

m: array [1..5,1..10] of <тип\_элементов>;

Более универсальное объявление массива можно сделать в разделе описания типов данных.

Type

Vector = array [1..5] of <тип\_элементов>;

Matrix= array [1..10] of vector;

Var

m: matrix;

Мы объявили двумерный массив Паскаля m, состоящий из 10 строк, в каждой из которых 5 столбцов. При этом к каждой i-й строке можно обращаться m[i], а каждому j-му элементу внутри i -й строки – m[i,j].

Определение типов для двумерных массивов Паскаля можно задавать и в одной строке:

Type

Matrix= array [1..5] of array [1..10] of < тип элементов >;

или еще проще:

Type

Matrix = array [1..5, 1..10] of <тип элементов>;

Обращение к элементам двумерного массива имеет вид: M [i,j]. Это означает, что мы хотим получить элемент, расположенный в i-й строке и j-м столбце. Тут главное не перепутать строки со столбцами, а то мы можем снова получить обращение к несуществующему элементу. Например, обращение к элементу M [10, 5] имеет правильную форму записи, но может вызвать ошибку в работе программы.

**Основные действия с двумерными массивами Паскаля**

Все, что было сказано об основных действиях с одномерными массивами, справедливо и для матриц. Единственное действие, которое можно осуществить над однотипными матрицами целиком – это присваивание. Т.е., если в программе у нас описаны две матрицы одного типа, например,

type

matrix= array [1..5, 1..10] of integer;

var

a , b : matrix ;

то в ходе выполнения программы можно присвоить матрице a значение матрицы b (a := b). Все остальные действия выполняются поэлементно, при этом над элементами можно выполнять все допустимые операции, которые определены для типа данных элементов массива.

Это означает, что если массив состоит из целых чисел, то над его элементами можно выполнять операции, определенные для целых чисел, если же массив состоит из символов, то к ним применимы операции, определенные для работы с символами.

**Ввод двумерного массива Паскаля.**

Для последовательного ввода элементов одномерного массива мы использовали цикл **for**, в котором изменяли значение индекса с 1-го до последнего. Но положение элемента в двумерном массиве Паскаля определяется двумя индексами: номером строки и номером столбца. Это значит, что нам нужно будет последовательно изменять номер строки с 1-й до последней и в каждой строке перебирать элементы столбцов с 1-го до последнего. Значит, нам потребуется два цикла for , причем один из них будет вложен в другой.

Пример программы ввода двумерного массива Паскаля с клавиатуры

type

matrix= array [1..5, 1..10] of integer;

var

a, : matrix;

i, j: integer; { индексы массива }

begin

for i :=1 to 5 do {цикл для перебора всех строк}

for j :=1 to 10 do {перебор всех элементов строки по столбцам}

readln ( a [ i , j ]); {ввод с клавиатуры элемента, стоящего в i -й строке и j -м столбце}

end,

Двумерный массив Паскаля можно заполнить случайным образом, т.е. использовать функцию random (N), а также присвоить каждому элементу матрицы значение некоторого выражения. Способ заполнения двумерного массива Паскаля выбирается в зависимости от поставленной задачи, но в любом случае должен быть определен каждый элемент в каждой строке и каждом столбце.

**Вывод двумерного массива Паскаля на экран.**

Вывод элементов двумерного массива Паскаля также осуществляется последовательно, необходимо напечатать элементы каждой строки и каждого столбца. При этом хотелось бы, чтобы элементы, стоящие в одной строке, печатались рядом, т.е. в строку, а элементы столбца располагались один под другим. Для этого необходимо выполнить следующую последовательность действий (рассмотрим фрагмент программы для массива, описанного в предыдущем примере):

Пример программы вывода двумерного массива Паскаля

program Mas2;

type

matrix=array [1..3, 1..4] of integer;

var

a:matrix;

i,j:integer;

begin

randomize;

for i:=1 to 3 do

begin

for j :=1 to 4 do

begin

a[i,j]:=random(20);

write('a[',i,',',j,']=',a[i,j]:2,' ');

end;

writeln;

end;

readln

end.

Замечание (это важно!): очень часто в программах студентов встречается ошибка, когда ввод с клавиатуры или вывод на экран массива пытаются осуществить следующим образом: readln (a), writeln (a), где а – это переменная типа массив. При этом их удивляет сообщение компилятора, что переменную этого типа невозможно считать или напечатать. Может быть, вы поймете, почему этого сделать нельзя, если представите N кружек, стоящих в ряд, а у вас в руках, например, чайник с водой. Можете вы по команде «налей воду» наполнить сразу все кружки? Как бы вы ни старались, но в каждую кружку придется наливать отдельно. Заполнение и вывод на экран элементов массива также должно осуществляться последовательно и поэлементно, т.к. в памяти ЭВМ элементы массива располагаются в последовательных ячейках.

**Примерные задания самостоятельной работы**

**Дан массив из N случайных целых чисел. Сосчитать:**

1. Произведение положительных элементов массива.
2. Произведение отрицательных элементов массива.
3. Сумма абсолютных значений элементов массива.
4. Количество отрицательных чисел в массиве.
5. Количество положительных чисел в массиве.
6. Составить программу обработки двумерного массива размерности NxN. В программе предусмотреть вывод всего массива в виде таблицы и вывод указанного ниже задания. Выполнить программу для N=4.
7. Определить сумму элементов главной диагонали.
8. Определить сумму элементов побочной диагонали.
9. Вычислить произведение элементов главной диагонали.
10. Вычислить произведение элементов побочной диагонали.
11. Заменить отрицательные элементы главной диагонали их произведением на -2.
12. Заменить положительные элементы побочной диагонали их произведением на 2.
13. Определить количество положительных, отрицательных и нулевых элементов массива.
14. Вычислить произведение положительных элементов массива.

**Критерии оценки самостоятельного задания:**

Работоспособный программный код

Оценка «Отлично» ставится за работоспособный оптимальный программный код

.Оценка «Хорошо» ставится за работоспособный оптимальный программный код, имеющий незначительные недочеты.

Оценка «Удовлетворительно» ставится за работоспособный неоптимальный программный код, имеющий незначительные недочеты.

**Тема 3.2. Строковый тип данных**

**Методические рекомендации по изучению лекций и литературы**

Ключевыми для данной темы являются следующие понятия: символ, строка, длина строки.

Студенты должны выучить функции преобразования типов, функции и процедуры работы со строками, их синтаксис и семантику.

**Строки**

Строки используются для хранения последовательностей символов. Она представляет собой особую форму одномерного массива, состоящего из символов. Но строка отличается от массива тем, что:

1. Массив имеет фиксированную длину:

var m : array[1..20] of char;

и у каждого элемента есть свое значение или 0.

2. Строка символов имеет две разновидности длины:

-общая длина строки, которая определяет величину памяти, выделяемый строке при описании;

-текущая длина строки (всегда **≤** общей длины) количество смысловых символов строки в данный момент времени.

-max длина строки 255 символов.

Под каждый символ отводится по одному байту, в котором хранится код символа. Еще один байт отводится под фактическую длину строки. Для коротких строк использовать стандартную строку неэффективно, поэтому в язык введена возможность самостоятельно задавать максимальную длину строки, например:

**type**

str4 = string [4];

Здесь описан собственный тип данных с именем str4. Переменная этого типа занимает в памяти 5 байт. Длина строки должна быть константой или константным выражением.

Примеры описания строк:

**var**

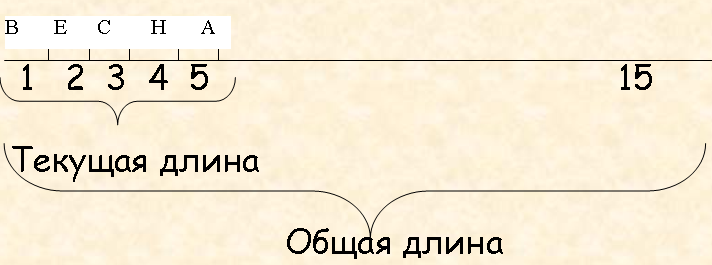
slovo : string; { Если в объявлении переменной описании длина строки не указывается, то она =255 – это строка стандартого типа }

s1 : str4; { строка типа str4, описанного выше }

s2 : string [15]; { описание типа задано при описании переменной }

Например, присвоим переменной значение:

**S2:= ‘весна’;**



В большинстве случаев переменные типа **String** исполняются для хранения слов и различных сообщений.

**Операции над строками**

Строки можно **присваивать** друг другу. Если максимальная длина результирующей строки меньше длины исходной, лишние символы справа отбрасываются:

S2:= ‘весна’;

s1:= s2; { в s1 будут помещены символы "весн" }

Строки можно **складывать** между собой с помощью операции конкатенации, которая обозначается знаком +, например:

s1 := 'ком';

s2 := s1 + 'пот'; { результат - "компот" }

или

s1 := 'ком';

s2 := 'пот';

slovo:=s1+s2; { результат - "компот" }

Строки можно сравнивать друг с другом с помощью **операций сравнения**. При сравнении строки рассматриваются **посимвольно слева направо**, при этом сравниваются коды соответствующих пар символов. Строки равны, если они имеют одинаковую длину и посимвольно эквивалентны:

'abc' > 'ab'

'abc' = 'abc'

'abc' < 'abc '

Имя строки может использоваться в операторах ввода **read и readln** и вывода **write и writeln**:

readln (s1, s2);

write (s1);

**При вводе** в строку считывается из входного потока количество символов, равное длине строки или меньшее, если символ перевода строки (Enter) встретится раньше.

**При выводе** под строку отводится количество позиций, равное ее фактической длине.

Переменная типа **String** состоит из цепочки символов, т.е. элементов типа **Char**

К отдельному символу строки можно обращаться как к элементу **массива символов**, например, **s1[4]**. Один символ строки имеет тип **char**, их можно использовать в выражениях одновременно, например:

s2:= ‘весна’;

тогда s1[3] = c

Таким образом существует 2 способа присваивать значения переменным типа строка:

1. **Как единый объект**

slovo:=‘университет’;

присваиваем сразу целую строку

stroka:=‘ННГУ’+’\_’+’им.Н.И.Лобачевского’

word:= slovo1 + slovo2;

1. **Строка- это совокупность символов (элементов типа Char)**, которые при обработке доступны каждый в отдельности. Доступ к отдельному элементу строки аналогичен доступу к отдельному элементу массива.

s2:= ‘весна’;

s2[4]:=‘т’**-** четвертому символу переменной s2 присвоили значение – т, получили s2 = ‘веста’;

Т.о., для доступа к отдельному элементу строки необходимо указать имя этой сроки и номер элемента в строке в [ ].С отдельным элементом строки возможно проделывать все те операции, что и с переменными типа **Char**.

**Стандартные функции работы со строками**

**Length()**

Функция length(str) - определяет фактическую длину строки str. результат имеет тип byte.

переменная := length (строка)

переменная – имя переменной типа integer

строка – имя строки, длина которой определяется.

Program one;

var N: integer; word: string;

begin

word:=‘информатика’;

n:=length(word);

write (N);

end.

**Copy()**

Функция **Copy(s, start, len)** возвращает подстроку длиной **len**, начинающуюся с позиции **start** строки **s**.

**Str:=Copy(s, start, len);**

**s** - исходная строка,

**start** - позиция, начиная с которой будет копироваться фрагмент,

**len** - число копируемых символов,

**str** - полученный фрагмент строки

Параметры len и start должны быть целого типа, str и s строковые.

**Пример:**

program p2;

var word; w1, w2, w3: string;

begin

word:=‘ картографирование’;

w1:=copy (word, 6,4);

writeln (w1);

w2:=copy (word, 2,3);

writeln (w2);

w3:=copy (word, 11, 3);

writeln (w3);

end.

**Pos()**

Функция **Pos(subs, s)** ищет вхождение подстроки subs в строку s и возвращает номер первого символа subs в s или 0, если subs не содержится в s.

program p3;

var word, w: string; N: integer;

begin

word:=‘электрификация’;

writeln (‘введите фрагмент для поиска’);

readln (w);

N= pos (w, word);

if N < > 0 then writeln (‘позиция =’,N);

else writeln (‘такого фрагмента нет’);

end.

**Concat()**

Функция **Concat(s1, s2, ..., sn)** возвращает строку, являющуюся слиянием строк s1, s2, ..., sn. Ее действие аналогично операции конкатенации.

**Стандартные процедуры работы со строками**

**Insert ()**

Процедура **Insert(subs, s, start)** вставляет в строку s подстроку subs, начиная с позиции start.

**Delete ()**

Процедура **Delete(s, start, len)** удаляет из строки s, начиная с позиции start, подстроку длиной len.

Program P4;

var word, word1: string;

begin

word:=‘колледж;

writeln (word); //колледж

delete (word,1,3);

writeln (word); //ледж

delete (word,4,1);

writeln (word); //лед

word1:=‘п’;

insert (word1, word, 1);

writeln (word); //плед

word1:=‘аа’;

insert (word1, word, 3);

writeln (word); //плааед

end.

**Процедуры преобразования типов в Паскале**

**Str()**

**Str (x, S)** преобразует число x в строковый формат. Здесь x – любое числовое выражение, S – строковая переменная. В процедуре есть возможность задавать формат числа x. Например, str(x: 8: 3, S), где 8 – общее число знаков в числе x, а 3 – число знаков после запятой.

Str (sin(1):6:4, S) – S=‘0.0175’

Str (3456, S) – S=‘3456’

**Val()**

**Val(S, x, kod)** преобразует строку символов S в число x. Здесь S – строковое выражение, x – числовая переменная (именно туда будет помещен результат), kod – целочисленная переменная (типа integer), которая равна номеру позиции в строке S, начиная с которой произошла ошибка преобразования, если преобразование прошло без ошибок, то переменная kod равна 0.

Тип X Оператор процедуры Значение X Значение kod

Real Val(’12.34’, x, kod) 12.34 0

Integer Val(’12.34’, x, kod) 12 3

program STR\_and\_VAL;

var chislo,kod: integer;

stroka: string;

begin

writeln ('Задайте число ');

readln(chislo);

str (chislo,Stroka);

writeln ('Строковое представление числа- ', stroka);

writeln ('stroka+stroka= ', stroka+stroka);

readln ;

val( stroka, chislo, kod);

if kod <> 0 then

writeln ('Ошибка ввода ')

else writeln ('Числовое значение ',chislo, ' chislo+chislo= ', chislo+chislo);

readln;

end.

Результат

Задайте число

123

Строковое представление числа- 123

stroka+stroka= 123123

Числовое значение 123 chislo+chislo= 246

**Примерные задания для самостоятельной работы**

Составить и отладить программу, выполняющую следующее задание

1. Определить количество символов «\*» во введённой строке.
2. Определить встречается ли во введённой строке русские буквы.
3. Подсчитать количество слов в строке. Слова разделены одиночными пробелами.
4. Определить количество русских букв во введённой строке.
5. Подсчитать количество предложений во введённой строке.
6. Определить встречается ли во введённой строке «ъ».
7. Определить количество слов во введённой строке, оканчивающихся на «ая». Слова разделены одиночными пробелами.
8. Определить количество латинских букв во введённой строке.
9. Определить встречается ли во введённой строке цифры.
10. Определить каких букв во введённой строке больше «и» или «е».
11. Определить номер позиции последнего знака «:».
12. Определить количество слов во введённой строке, оканчивающихся на «ий». Слова разделены одиночными пробелами.

**Критерии оценки самостоятельного задания:**

Работоспособный программный код

Оценка «Отлично» ставится за работоспособный оптимальный программный код

Оценка «Хорошо» ставится за работоспособный оптимальный программный код, имеющий незначительные недочеты.

Оценка «Удовлетворительно» ставится за работоспособный неоптимальный программный код, имеющий незначительные недочеты.

# Литература для учащегося

**Основные источники:**

1. 1. Семакин И.Г. Основы алгоритмизации и программирования: учебник для студ.учреждений СПО / И.Г.Семакин, А.П.Шестаков. – 3-е изд., - М.: Издательский дом «Академия», 2012. – 400с. Допущено Министерством образования и науки РФ в качестве учебника для студентов образовательных учреждений среднего профессионального образования. - Режим доступа: <http://znanium.com/catalog.php?bookinfo=507976>
2. 2. Голицына О.Л. Языки программирования: Учебное пособие / О.Л. Голицына, Т.Л. Партыка, И.И. Попов. - 3-e изд., перераб. и доп. - М.: Форум: ИНФРА-М, 2015. - 400 с. Допущено Министерством образования и науки РФ в качестве учебника для студентов образовательных учреждений среднего профессионального образования - Режим доступа: <http://znanium.com/catalog.php?bookinfo=507976>

**Дополнительные источники:**

1. Программирование и основы алгоритмизации: Для инженерных специальностей технических университетов и вузов. /А.Г. Аузяк, Ю.А. Богомолов, А.И. Маликов, Б.А. Старостин. Казань: Изд-во Казанского национального исследовательского технического ун-та - КАИ, 2013, 153 с.

2. Канцедал С.А. Алгоритмизация и программирование : Учебное пособие / С.А. Канцедал. - М.: ИД ФОРУМ: НИЦ Инфра-М, 2013. - 352 с.: ил.

3. Колдаев В.Д. Основы алгоритмизации и программирования: Учебное пособие / В.Д. Колдаев; Под ред. Л.Г. Гагариной. - М.: ИД ФОРУМ: ИНФРА-М, 2012. - 416 с.: ил.;

4. В.Б. Попов В.Б. Turbo Pascal для школьников: Учебно-методическое пособие / В.Б. Попов. - 2-e изд. - М.: ИЦ РИОР: ИНФРА-М, 2012. - 373 с.

**Ресурсы INTERNET**

1. [Электронный ресурс]/(http://www.ict.edu.ru/catalog/index.php).
2. [Электронный ресурс]/(http://artishev.com/texnologii/setevaya-os.html).
3. [Электронный ресурс]/(http://inoblogger.ru/2010/03/31/operacionnaya-sistemanterneta/).
4. [Электронный ресурс]/(http://www.tver.mesi.ru/e-lib/res/648/14/1.html).
5. [Электронный ресурс]/ <http://learnpascal.ru>.
6. [Электронный ресурс]/ [www.pas1.ru](http://www.pas1.ru)

**Инна Дмитриевна Камскова**

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ для самостоятельной работы по дисциплине «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ»**

Учебно-методическое пособие

Федеральное государственное автономное образовательное  
 учреждение высшего образования

Нижегородский государственный университет имени Н.И. Лобачевского

603950, Нижний Новгород, пр. Гагарина, 23.