Министерство образования и науки Российской Федерации
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

А.В. РОДЫГИН

Информационные технологии
АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Утверждено Редакционно-издательским советом университета
в качестве учебного пособия

НОВОСИБИРСК
2017

УДК 004.421(075.8)
Р 617

Рецензенты:
канд. техн. наук, доцент *В.М. Кавешников*
канд. техн. наук, доцент *Д.А. Павлюченко*

Работа подготовлена на кафедре Электропривода и автоматизации промышленных установок для студентов I курса ФМА всех форм обучения

Родыгин А.В.

Р 617 Информационные технологии. Алгоритмизация и программирование: учебное пособие / А.В. Родыгин. – Новосибирск: Изд-во НГТУ, 2017. – 92 с.

ISBN 978-5-7782-3300-3

Пособие содержит общие принципы составления алгоритмов различных инженерных задач и их программной реализации на языке программирования Паскаль. Предназначено для студентов направления 13.04.02 «Электроэнергетика и электротехника» всех форм обучения, а также может быть рекомендовано для студентов других направлений подготовки.

УДК 004.421(075.8)

ISBN 978-5-7782-3300-3

Оглавление

Введение 4
1. Основные понятия и определения языка программирования Паскаль 7
1.1. Словарь языка Паскаль 7
1.2. Компиляция 9
1.3. Структура программы на языке Паскаль 10
1.4. Пунктуация в программах на Паскале 13
1.5. Операторы Паскаль 14
1.6. Переменные и константы 17
1.7. Типы данных в Паскале 19
1.8. Логические выражения и логические операции 28
2. Алгоритмизация простых вычислительных процессов 31
2.1. Алгоритм и его свойства 31
2.2. Способы представления алгоритмов 32
2.3. Виды алгоритмов 34
2.4. Алгоритмизация обработки одномерных массивов 57
3. Алгоритмизация сложных вычислительных процессов 64
3.1. Алгоритмизация обработки двумерных массивов 64
3.2. Алгоритмы обработки бесконечных числовых рядов 72
3.3. Алгоритмизация исследования областей двумерного пространства 75
3.4. Алгоритмизация с использованием подпрограмм 78
3.5. Алгоритмы решения алгебраических уравнений 84
Библиографический список 90
Приложение 91

ВВЕДЕНИЕ

Модуль «Алгоритмизация и программирование» является одним из центральных в дисциплине «Информационные технологии». Изучение алгоритмизации и программирования имеет несколько целевых аспектов.

Развивающий аспект. Под развивающим аспектом понимается развитие алгоритмического, конструктивного, логического мышления, а также формирование операционного типа мышления, которое направлено на выбор оптимального из нескольких возможных решения поставленной задачи. Развитие этих специфических видов мышления делает весомый вклад в формирование общего научного мировоззрения и умственных способностей личности.

Формирование алгоритмического стиля мышления — это социальный заказ информационного общества образовательным учреждениям. Информатика и информационные технологии имеют в своем составе систему понятий, которая позволяет в полном объеме сформировать у обучающегося умения и навыки, имеющие общекультурную, общеобразовательную, общечеловеческую ценность, которые нужны каждому человеку в современном информационном обществе; информатика может предложить столь необходимый для формирования алгоритмического мышления дидактический инструментарий.

Перечислим основные умения и навыки, соответствующие алгоритмическому стилю мышления, которые формирует информатика: умения и навыки планирования структуры действий, необходимых для достижения цели при помощи фиксированного набора средств; структурирование сообщений; понимание и использование формальных способов кодирования решения задачи; технические навыки и умения взаимодействия с компьютером; проектирование и построение информационных и компьютерных моделей; инструментирование всех видов деятельности; умение производить структурный анализ задачи, разбивать большие задачи на малые, сводить нерешенные задачи к решенным.

В процессе составления алгоритмов решения задач, а затем преобразования (кодирования) алгоритмов в программы для определенных исполнителей у обучающихся развиваются навыки проведения логических рассуждений и характерные для дедуктивного мышления умения находить логические следствия из заданных начальных условий, способности абстрагировать, т. е. выделять в конкретной ситуации существенные свойства объекта, процесса, отвлекаясь от несущественных, умения анализировать, сравнивать, обобщать, ставить вопросы, давать четкие ответы, выдвигать конструктивные решения, обосновывать предложенные решения, специализировать, определять понятия, составлять суждения. Все это формирует мышление обучающихся и способствует развитию их речи, особенно таких качеств выражения мысли, как порядок, точность, ясность, краткость, обоснованность. Каждое из перечисленных умений и навыков имеет самостоятельное и важное значение в системе навыков умственных действий, необходимых любому современному человеку.

Практический аспект. Важной целью изучения предмета «Информационные технологии» является получение студентами опыта построения и исследования моделей реальных объектов на компьютере, в частности, с использованием систем программирования. Фундаментом любых компьютерных технологий являются компьютерные модели, которые отображают реальные объекты, явления, человеческую деятельность в компьютерные информационные структуры; именно в своей алгоритмической части информатика и информационные технологии имеют дело с формальным описанием внешнего мира.

Изучать идеи и методы программирования лучше с использованием тех систем, которые специально созданы для обучения искусству программирования.

Существует множество языков программирования. Среди них выделяется десяток наиболее известных и используемых в те или иные периоды компьютерной истории. В эту группу широко распространенных языков входит язык программирования *Паскаль*.

Язык Паскаль был создан в конце 60-х годов XX века Н. Виртом как специальный язык для обучения студентов. Однако вскоре язык Паскаль получил распространение среди программистов из-за реализации в нём прогрессивных идей того времени. Язык Паскаль широко использовался для написания прикладных программ и как язык системного программирования. Программное обеспечение многих мини и микрокомпьютеров было написано на *Паскаль*. Язык *Паскаль* разви-

вался и совершенствовался, включал в себя новые возможности. Производились новые трансляторы и среды разработки для *Паскаль*.

Трансляторы с этого языка имелись на наиболее распространенных типах ЭВМ во всем мире. Наличие специальных методик создания трансляторов с Паскаля упростило их разработку и способствовало широкому распространению языка. Трансляторы могли оптимизировать код и это позволяло создавать эффективные программы. Это как раз и послужило одной из причин использования Паскаля в качестве языка системного программирования.

Среди других достоинств языка программирования *Паскаль* можно отметить следующие:

– простота языка позволяет быстро его освоить и создавать алгоритмически сложные программы;

– развитые средства представления структур данных обеспечивают удобство работы как с числовой, так и с символьной и битовой информацией;

– в языке Паскаль реализованы идеи структурного программирования, что делает программу наглядной и дает хорошие возможности для разработки и отладки.

В настоящее время по свободной лицензии *LGPL* можно использовать систему программирования ***PascalABC.NET.*** Адрес сайта: *http://pascalabc.net/*

1. ОСНОВНЫЕ ПОНЯТИЯ И ОПРЕДЕЛЕНИЯ ЯЗЫКА ПРОГРАММИРОВАНИЯ ПАСКАЛЬ

1.1. Словарь языка Паскаль

Язык программирования Паскаль использует следующий набор символов.

1. Английские и русские буквы, которые могут быть как строчными, так и прописными.

2. Арабские цифры.

3. Знаки операций и знаки, входящие в составное обозначение операций:

+ - * / : = < >

4. Ограничители:

. , ( ) [ ] { } : ; ‘ “

5. Специальные символы:

$ @ # & ^ _ ~ %

Для обращения к ячейкам памяти компьютера используются переменные. **Переменная** – это имя физического участка памяти, в котором в любой момент времени может храниться только одно значение. Для обозначения участков памяти используют идентификаторы (имена переменных).

Идентификаторы (имена переменных, констант программ, модулей, функций, процедур) записываются с помощью допустимых в Паскале символов, и они должны соответствовать следующим правилам:

– имя должно состоять только из английских букв, цифр и символов подчеркивания (в идентификаторах в Паскале прописные и строчные буквы не различаются);

– имя может начинаться только с английской буквы или символа подчеркивания.

Слова, употребляемые в большинстве языков программирования, в том числе и Паскаль, делятся на три группы:

– зарезервированные (ключевые) слова самого языка;

– предопределенные имена;

– идентификаторы, определяемые программистом.

Служебные (зарезервированные, ключевые) слова – это слова языка программирования, которые имеют специальное, раз и навсегда закрепленное за ними значение. В программе нельзя использовать идентификаторы, совпадающие по написанию с ключевыми словами (например, нельзя обозначить переменную именем *begin*, так как это слово используется в самом языке).

Ключевые слова языка программирования Паскаль:

and	*array*	*begin*	*case*	*const*	*div*	*do*
downto	*else*	*end*	*file*	*for*	*function*	*goto*
if	*in*	*label*	*mod*	*nil*	*not*	*of*
or	*packed*	*procedure*	*program*	*record*	*repeat*	*set*
then	*to*	*type*	*until*	*var*	*while*	*with*

Предопределенные (стандартные) имена также имеют специальный (заранее заданный) смысл. Однако программист может обходить закрепленные за ними значения и использовать их в качестве имен, определяемых программистом. Если программист не определит явно, для каких целей применяется то или иное стандартное имя в программе, оно будет восприниматься в присущем данному имени предопределенном смысле. Например, предопределенными являются имена *Integer*, *Writeln*.

Пользовательские имена определяются программистом и должны быть явно объявлены в программе.

Десятичные числа всегда начинаются с цифры, перед которой может стоять знак числа: + или –.

Действительные числа изображаются в двух форматах. В формате с фиксированной точкой явно указывают положение десятичной точки (4.505, –16.25, +1.0). В формате с плавающей точкой используется десятичный порядок, обозначаемый прописной или строчной буквой E,

после которой идет целое число, указывающее значение порядка (8*E*10, 1.62E-3, 0.8E+3).

Распространенные компиляторы с языка Паскаль позволяют оперировать действительными числами до 38 порядка, а некоторые – до 67 порядка.

В языке Паскаль максимально допустимое целое число равняется 2 147 483 647, оно содержится в предопределенной целой константе *MaxLongInt*. В Паскале возможен диапазон целых чисел от –2 147 483 648 до 2 147 483 647.

Предопределенная в Паскале целая константа *MaxInt* содержит в себе значение 32 767. Целое число может задаваться не только в десятичной, но и в шестнадцатеричной системе счисления. Шестнадцатеричному числу предшествует знак $.

Текстовым литералом (строкой) в языке Паскаль называют последовательность любых допустимых символов, стоящих между апострофами (`*Hello*!`). Если в качестве символа строки необходимо использовать апостроф, то записывают подряд два апострофа.

Строку можно задавать также в виде последовательности, образованной из символов # с последующим цифровым кодом требуемого символа (например, запись #72#73#33 эквивалентна строке *'HI*!'). В строковых данных прописные и строчные буквы различаются.

Пробелы являются разделителями. Между любыми именами, числами, ключевыми словами должен стоять, по крайней мере, один разделитель. Но нельзя отделять один символ от другого внутри одного имени, числа, либо ключевого слова.

1.2. Компиляция

Программу на Паскале надо предварительно скомпилировать. Компиляция означает перевод исходной программы с языка *Паскаль* в объектную программу – на язык компьютера. После компиляции имеются две версии программы: одна на Паскале, другая на языке компьютера (или близком к нему). При запуске программы, вычисления производятся по программе в объектном коде, а не по исходной программе.

Программы на языке *Паскаль* и других компилируемых языках выполняются быстрее, чем программы с интерпретируемых языков. Это связано с тем, что объектная программа на языке, близком к языку

компьютера (или непосредственно в командах компьютера), выполняется быстро, в то время как инструкции программы на интерпретируемых языках выполняются путем непосредственного перевода исходного кода на язык компьютера.

Сначала исходная программа записывается программистом (редактируется), после чего сохраняется на диске под произвольным именем. На следующем шаге в память загружаются компилятор и исходный текст программы. Компилятор «читает» ее, превращая в объектный код, который затем также сохраняется на диске.

Последний шаг – выполнение программы. В компьютерную память загружается именно объектный (исполняемый) код. Выполняющаяся программа может предполагать ввод данных (*input*) с клавиатуры и вывод результатов (*output*) на экран.

1.3. Структура программы на языке Паскаль

Программа состоит из заголовка и блока. В заголовке указывается имя программы и список параметров. Общий вид:
Program n(input, output, x, y);
здесь *n* – имя программы; *input* – файл ввода; *output* – файл вывода; *x*, *y* – внешние файлы, используемые в программе.

Заголовка может и не быть или он может быть без параметров.

Блок программы состоит из шести разделов, следующих в строго определенном порядке:

– раздел меток (*label*);
– раздел констант (*const*);
– раздел типов (*type*);
– раздел переменных (*var*);
– раздел процедур и функций;
– раздел действий (операторов).

Раздел действий должен присутствовать всегда, остальные разделы могут отсутствовать.

Каждый из первых четырех разделов начинается с соответствующего ключевого слова (*label*, *const*, *type*, *var*), которое записывается один раз в начале раздела и отделяется от последующей информации только пробелом, либо концом строки, либо комментарием.

1.3.1. Раздел меток (*label*)

Любой выполняемый оператор может быть снабжен меткой – целой положительной константой, содержащей не более четырех цифр. Все метки, встречающиеся в программе, должны быть описаны в разделе *label*.

Общий вид:

label l1, l2, l3;

здесь *l1*, *l2*, *l3* – метки.

Пример:

label 5, 10, 100;

Метка отделяется от оператора двоеточием.

Пример. Пусть выражение *a:=b* имеет метку 20. Тогда этот оператор выглядит так:

20: a := b;

1.3.2. Раздел констант (*const*)

Если в программе используются константы, имеющие достаточно громоздкую запись (например, число π с восьмью знаками), либо сменные константы (для задания варианта программы), то такие константы обычно обозначаются какими-либо именами и описываются в разделе *const*, а в программе используются только имена констант. Это делает программу более наглядной и удобной при отладке и внесении изменений.

Общий вид:

const c1 = z1;
c2 = z2;

здесь *c1, c2* – имена констант, *z1, z2* – значение констант.

Пример:

const pi = 3.14; c = 2.7531;

1.3.3. Раздел типов (*type*)

Если в программе вводится тип, отличный от стандартного, то этот тип описывается в разделе *type*.

Общий вид:

type t1 = вид_типа;
t2 = вид_типа;

где *t1* и *t2* – идентификаторы вводимых типов.

Затем тип используется при объявлении переменных. Пример использования нестандартных типов:

```
const  len=40;
type   year=1930..2010;
       names=string[len];
var    empl: names;
       y: year;
```

Раздел описания типов имеет большое значение в программе на языке *Паскаль*. Если в программе не использовать типы, то можно столкнуться с несовместимостью типов переменных, даже если они описаны одинаково.

1.3.4. Раздел переменных (*var*)

Пусть в программе встречаются переменные *v1*, *v2*, *v3, v4*; все они должны быть описаны следующим образом:

```
var    v1, v3: type1;
       v2, v4: type2;
```

здесь *v1*, *v2*, *v3*, *v4* – имена переменных; *type1* – тип переменных *v1*, *v3*; *type2* – тип переменных *v2*, *v4*.

Пример:

```
var
        k, i, j: integer;
        a, b: real;
```

Каждая переменная должна быть описана до ее использования в программе и отнесена к одному и только одному типу.

Таким образом, в разделе *var* вводится имя каждой переменной и указывается, к какому типу эта переменная принадлежит. Тип переменной можно задать двумя способами: указать имя типа (например: *real*, *color*) либо описать сам тип, (например: *array[1..16] of integer*).

1.3.5. Раздел процедур и функций

В этом разделе присутствуют заголовки и тела пользовательских процедур и функций.

1.3.6. Раздел действий (операторов)

Эта часть программы начинается с ключевого слова *begin* и заканчивается словом *end*, после которого должна стоять точка. Раздел действий есть выполняемая часть программы, состоящая из операторов.

1.4. Пунктуация в программах на Паскале

1. Заголовок завершается точкой с запятой.

2. В любом объявлении каждый список завершается точкой с запятой.

3. Операторы отделены один от другого точкой с запятой.

Слова *begin* и *end* не являются операторами – они служат знаками пунктуации и играют роль операторных скобок. Слово *begin* выступает в качестве левой, а *end* – правой скобки. Так как они сами знаки пунктуации, то точка с запятой после *begin* и перед *end* не обязательна.

В программах на Паскаль слова *begin* и *end* используются преимущественно для образования составных операторов. Если при некотором условии надо выполнить определенную последовательность операторов, то их объединяют в один составной оператор.

Составной оператор начинается ключевым словом *begin* и заканчивается словом *end*. Между этими словами помещаются составляющие операторы, которые выполняются в порядке их следования. После *end* ставится точка с запятой, а после *begin* – только пробелы (либо комментарий).

Тело самой программы также имеет вид составного оператора. После последнего *end* программы ставится точка. Нельзя извне составного оператора передавать управление внутрь его.

Составной оператор может быть использован в любом месте, где мог бы быть использован простой оператор. Пример составного оператора, осуществляющего обмен значениями двух переменных:

```
begin
    t := a;
    a := b;
    b := t
end;
```

Слова в других операторах также действуют как знаки пунктуации.

```
if a > b
    then write('yes')
    else write('no');
```

Слова *if*, *then*, *else* выступают внутри оператора в качестве знаков пунктуации.

Операторы разделены знаками пунктуации, поэтому расположение программы на странице с точки зрения компилятора значения не имеет. Вполне достаточно придерживаться двух правил:

– не писать слова вместе;

– не разрывать слово пробелами или переходом на новую строку.

В остальном компилятору все равно, как будет расположена программа, однако это совсем не безразлично для программиста. Польза отступов в прояснении структуры программы. Взгляды на выбор отступов различны, но все согласны в одном – отступы должны делать структуру программы максимально наглядной.

Слова *program*, *const*, *var*, *begin*, *end*, а также множество других называются зарезервированными словами. Зарезервированные слова нельзя расширять (например, *constant* будет ошибкой) и сокращать (например, *prog* также будет ошибкой).

Использовать в программном коде на *Паскале* можно как прописные, так и строчные буквы, а также их чередовать. Однако в строках (тип данных) разница между прописными и строчными буквами существует.

1.5. Операторы Паскаль

Под операторами в языке Паскаль подразумевают только описание действий. Операторы отделяются друг от друга только точкой с запятой. Если оператор стоит перед *end*, *until* или *else*, то в этом случае точка с запятой не ставится.

1.5.1. Оператор присваивания

Общий вид: *v := a;*

здесь *v* – переменная, *a* – выражение, **:=** – операция присваивания. Выражение *a* может содержать константы, переменные, названия функций, знаки операций и скобки.

Пример. $f := 3 \cdot c + 2 \cdot \sin(x)$;

Вид выражения однозначно определяет правила его вычисления: действия выполняются слева направо с соблюдением приоритета операций (см. разд. 1.8.3.4). Любое выражение в скобках вычисляется раньше, чем выполняется операция, предшествующая скобкам. Присваивание допускается для переменных всех типов, за исключением типа файл.

В операции *v := a* переменная *v* и выражение *a* должны иметь один и тот же тип, а для интервального типа — одно и то же подмножество значений.

Примечания

1. Разрешается присваивать переменной типа *real* выражение типа *integer*.
2. Нельзя присваивать переменной типа *integer* выражение типа *real*.

1.5.2. Выражения в Паскале

В операторах присваивания можно использовать арифметические выражения.

Пример:

```
num := (d + n) / 10;
s := trunk(num) + 1;
```

Скобки обеспечивают необходимый порядок вычислений. Если бы в первом операторе примера скобки были опущены: *num := d + n / 10;* то сначала было бы выполнено деление, приоритет которого выше. Приоритет в арифметических выражениях выше у операций умножения (·) и деления (/), ниже у сложения и вычитания.

Во втором операторе примера производится присваивание значения целого числа. Функция ***trunc*** дает целый результат, а число 1 записано без десятичной точки; таким образом, оба слагаемых в сумме дают целое значение. Вообще, когда все члены выражения – целые, само выражение принимает целое значение.

У сформулированного выше правила существует важное исключение: деление (с использованием знака /) всегда дает вещественный результат:

$$6.5 / 2 = 3.25 \qquad 6 / 2 = 3.0$$

Деление нацело (нахождение частного и остатка) может быть выполнено при помощи операций ***div*** и *mod*.

Выражение может включать в себя и целые и вещественные члены. Наличие хотя бы одного вещественного члена или знака / приводит к тому, что значение результата будет вещественным. Функции ***trunc*** и ***round*** могут быть использованы для преобразования вещественного числа в целое.

Функция ***sqr*** возводит значение аргумента (записанного внутри скобок) в квадрат. В Паскале нет оператора возведения в произвольную степень. Возведение в степень здесь осуществляется с использованием логарифмов. Использованием равенства $e^{ln(y)} = y$ и $ln(a^x) =$

$= x \cdot ln(a)$ вместо математического выражения $y = a^x$ на языке *Паскаль* можно написать *y := exp(ln(a)·x)*.

Знаки <, >= и подобные также играют роль операций. Выражения, содержащие подобные операции, принимают логическое значение и называются логическими выражениями. В состав логических выражений могут входить логические операции инверсии *not* (не), конъюнкции *and* (и), дизъюнкции *or* (или). Такие логические выражения называются сложными.

1.5.3. Ввод-вывод данных в Паскале

Компьютерные программы обрабатывают (изменяют) различные данные. Программа получает данные, производит с ними действия и выводит их в измененной форме или выводит другие данные.

Следовательно, любой язык программирования должен иметь инструменты как для ввода данных, так и их вывода. Стандартным устройством ввода является клавиатура, а вывода – монитор. Стандартные – значит, «работающие по умолчанию»; т. е. если не указано ничего иного, то программа будет считывать данные с клавиатуры, а выводить их на монитор.

Вместе клавиатуру и монитор называют консолью. Таким образом, консоль представляет собой стандартное устройство ввода-вывода.

1.5.3.1. Вывод данных на экран

Вывод данных на экран и в файл в языке программирования Паскаль осуществляется с помощью процедур ***write()*** и ***writeln()***.

Допустим, нам требуется отобразить на экране пару фраз. Если мы хотим, чтобы каждая из них начиналась с новой строки, то надо использовать ***writeln()***, если нет – то ***write()***.

Write() чаще используется, когда надо вывести для пользователя сообщение на экран, после чего получить данные, не переводя курсор на новую строку. Например, выводим на экран «Введи число:» и не переводим курсор на новую строку, а ждем ввода.

write(`введите число n=`);

В программе может быть осуществлен так называемый форматированный вывод. При этом для выводимого значения указывается ширина поля вывода (количество знакомест). Если мы выводим вещественное (дробное) число, то вторым числом через двоеточие указывается количество знаков после запятой. Если для вещественных чисел не

осуществлять форматирование, то они отобразятся так, как определено для данного компьютера. Если указать только число знакомест без фиксирования дробной части, то вывод будет в экспоненциальной форме.

1.5.3.2. Ввод данных с клавиатуры

Ввод данных в языке программирования Паскаль обеспечивается процедурами ***read()*** и ***readln()***. Ввод данных осуществляется либо с клавиатуры, либо из файла.

Когда данные вводятся, то они помещаются в ячейки памяти, доступ к которым обеспечивается с помощью механизма переменных. Поэтому, когда в программе на *Паскале* используется процедура ***read()*** или ***readln()***, то в качестве фактического параметра (аргумента) ей передается имя переменной, которая будет связана с вводимыми данными. Потом эти данные можно будет использовать в программе или просто вывести на экран.

В процедуры ввода можно передавать не один фактический параметр, а множество. При вводе данных их разделяют пробелом, табуляцией или переходом на новую строку (*Enter*). Данные символьного типа не разделяются или разделяются переходом на новую строку.

Существуют особенности ввода данных с помощью операторов ***read()*** и ***readln()***. Если используются подряд несколько операторов ***read()***, то вводимые данные можно разделять всеми допустимыми способами. При использовании нескольких вызовов ***readln()*** каждый последующий срабатывает только после нажатия Enter.

1.6. Переменные и константы

1.6.1. Переменные

Любая программа обрабатывает данные (информацию, объекты). Данные, с которыми работает программа, хранятся в оперативной памяти компьютера. Программа должна знать, где они лежат, каким объемом памяти она располагает, как следует интерпретировать данные (например, как числа или строки). Для обеспечения программе доступа к участкам памяти существует механизм переменных.

Переменные описываются в начале программы и как бы сообщают о том, с какими данными будет работать программа и какой объем па-

мяти они займут. Другими словами, резервируется память. Но это не значит, что в эти ячейки памяти помещаются конкретные значения (данные). На момент резервирования памяти в них может быть что угодно.

В процессе выполнения программы в ячейки памяти будут помещаться конкретные значения, извлекаться оттуда, изменяться, снова записываться. Мы же через программу обращаемся к ним посредством имен переменных, которые были описаны в начале программы.

Имена переменных (идентификаторы) могут быть почти любым сочетанием английских букв и цифр (без пробелов). Нельзя чтобы имена переменных совпадали со словами, которые являются какими-либо командами самого языка программирования. Нельзя начинать имена переменных с цифры или специального символа. Для того чтобы имена переменных были удобны для восприятия, надо стараться придерживаться следующих правил. Если программа не простейший пример, то имена переменных должны быть осмысленными словами или их сокращениями. Желательно, чтобы имена переменных не были слишком длинными.

В *Паскале* прописные и строчные буквы в именах переменных не различаются.

При описании переменных указывается не только их имя, но и тип. Тип переменных сообщает о том, сколько отвести под них памяти и что за данные там планируется сохранять. И хотя храниться там будут числа в двоичной системе счисления, их значение может быть чем угодно: целым или дробным числом, символом, строкой, массивом, записью и др. Таким образом, тип переменной определяет то, что мы можем сохранить в участке памяти, с которым связана описываемая переменная.

Например, под тип *integer* в языке программирования Паскаль отводится два байта, что равно 16 битам, а это значит, что можно хранить 2^{16} (65 536) значений (отрицательные и положительные числа, а также ноль). Если тип переменных *integer*, то им можно присваивать только целые числа в диапазоне от –32 768 до 32 767. В этом диапазоне переменные типа *integer* могут принимать какие угодно значения. При попытке записи в переменную значения не ее типа возникнет ошибка.

Таким образом, переменные связаны с участками памяти, содержимое которых может меняется по ходу выполнения программы в определенных пределах.

1.6.2. Константы

Если в программе требуется постоянно использовать какое-нибудь одно и то же число, то вроде бы можно описать переменную, затем присвоить ей значение и не изменять его в программе. Однако это не всегда удобно (можно случайно изменить), поэтому в языках программирования для хранения данных помимо переменных существуют константы.

Главное преимущество констант заключается в том, что они описываются в начале программы и им сразу там же присваивается значение, а при выполнении программы константы не изменяются. Но если при правке кода программист решит поменять значение константы, он впишет в ее описание другое значение, а сам код программы редактировать не придется. Поэтому, если в программе часто планируется использовать какое-то значение, опишите его в разделе констант, который в программе располагается до раздела переменных:

```
const
  конст1 = значение;
  конст2 = значение;
```

Значениями констант могут быть данные большинства типов, используемых в языке Паскаль.

1.6.3. Типизированные константы

В языке Паскаль помимо обычных констант используются типизированные константы. Можно сказать, что они занимают промежуточное положение между переменными и константами. Они получают значение при описании (как константы), но могут его менять в теле программы (как переменные).

Описываются типизированные константы в разделе констант:

```
const
  конст1: тип=значение;
  конст2: тип=значение;
```

1.7. Типы данных в Паскале

Любая программа, написанная на любом языке программирования, по большому счету предназначена для обработки данных. В качестве данных могут выступать числа, тексты, графика, звук и др. Одни данные являются исходными, другие – результатом, который получается путем обработки исходных данных программой.

Данные хранятся в памяти компьютера. Программа обращается к ним с помощью имен переменных, связанных с участками памяти, где хранятся данные. Переменные описываются до основного кода программы. Здесь указываются имена переменных и тип хранимых в них данных.

В языке программирования Паскаль достаточно много типов данных. Кроме того, сам пользователь может определять свои типы. Тип переменной определяет, какие данные можно хранить в связанной с ней ячейке памяти.

Переменные типа *integer* могут быть связаны только с целыми значениями обычно в диапазоне от –32 768 до 32 767. В *Паскале* есть другие целочисленные типы (*byte*, *longint*).

Переменные типа *real* хранят вещественные (дробные) числа.

Переменная булевского (логического) типа (*boolean*) может принимать только два значения – *true* (1, правда) или *false* (0, ложь).

Символьный тип (*char*) может принимать значения из определенной упорядоченной последовательности символов.

Интервальный тип определяется пользователем и формируется только из порядковых типов. Представляет собой подмножество значений в конкретном диапазоне.

Можно создать собственный тип данных простым перечислением значений, которые может принимать переменная данного типа. Это так называемый перечисляемый тип данных.

Все описанное выше – это простые типы данных. Но бывают и сложные, структурированные, которые базируются на простых типах.

Массив – это структура, занимающая в памяти единую область и состоящая из фиксированного числа компонентов одного типа.

Строки представляет собой последовательность символов. Причем количество этих символов не может быть больше 255 включительно. Такое ограничение является характерной чертой *Паскаля*.

Запись – это структура, состоящая из фиксированного числа компонент, называемых полями. В разных полях записи данные могут иметь разный тип.

Множества представляют собой совокупность любого числа элементов, но одного и того же перечисляемого типа.

Файлы для *Паскаля* представляют собой последовательности однотипных данных, которые хранятся на устройствах внешней памяти (например, жестком диске).

Понятие такого типа данных, как указатель связано с динамическим хранением данных в памяти компьютера. Часто использование динамических типов данных является более эффективным в программировании, чем статических.

1.7.1. Целые типы

В языке Паскаль определено пять целых типов.

Таблица 1.1

Целые типы Паскаля

Тип	Диапазон допустимых значений	Отводимая память, в байтах
shortint	–128…127	1
integer	–32 768…32 767	2
longint	–2 147 483 648…2 147 483 647	4
byte	0…255	1
word	0…65 535	2

Переменные целого типа могут принимать только целые значения. Такие переменные в программе описываются следующим образом:

```
var
  a, b, c: integer;
  k: shortint;
  i, j: byte;
```

Таблица 1.2

Операции над целыми типами, дающие значение целого типа

Знак операции	Операция
+	Сложение
-	Вычитание
*	Умножение
div	Целочисленное деление (остаток отбрасывается)
mod	Деление по модулю (выделение остатка от деления)

Операции над операндами целого типа выполняются правильно только при условии, что результат и каждый операнд не меньше минимального (крайнего левого) и не больше максимального (крайнего правого) значения диапазона. Например, в Паскале существует константа *maxint*, в которой содержится максимально допустимое значение для типа *integer*.

Над целыми типами, как и многими другими, допустимы операции отношения (сравнения): равно, не равно, больше или равно, больше, меньше или равно, меньше. Результат таких операций относится к типу *boolean* и может принимать одно из двух значений – либо *true* (истина), либо *false* (ложь).

Целые типы могут приниматься в качестве фактических параметров рядом стандартных функций языка программирования *Паскаль*.

Таблица 1.3

Стандартные функции Паскаля применимые к аргументам целых типов

Функция	Тип результата	Результат выполнения
abs(x)	Целый	Модуль *x* (абсолютная величина *x*)
sqr(x)	Целый	Квадрат *x*
succ(x)	Целый	Следующее значение *x* := *x*+1
pred(x)	Целый	Предыдущее значение *x* := *x*-1
random(x)	Целый	Случайное целое число из интервала 0…*x*-1
sin(x)	Действительный	Синус *x* (угол в радианах)
cos(x)	Действительный	Косинус *x* (угол в радианах)
arctan(x)	Действительный	Арктангенс *x* (угол в радианах)
ln(x)	Действительный	Натуральный логарифм *x*
exp(x)	Действительный	Экспонента *x*
sqrt(x)	Действительный	Квадратный корень из *x*
odd(x)	Логический	Значение *true*, если *x* – нечетное число; *false* – если четное

Функция *random* возвращает равномерно распределенное случайное целое число, если ей передан целый аргумент. При повторном запуске программы она возвращает те же значения. Во избежание этого следует в начале программы вызвать процедуру без параметров *randomize*.

Процедуры *inc* и *dec* могут иметь по одному или по два параметра целого типа. Если параметров два, то значение первого увеличивается (для *inc*) или уменьшается (для *dec*) на величину, равную значению второго параметра. Например, *inc(x,2)* равнозначно *x*+2. Если параметр один, то его значение увеличивается (для *inc*) или уменьшается (для *dec*) на единицу. Например, *dec(x)* равнозначно *x*-1.

Следующие функции принимают в качестве аргументов значения вещественного типа, а возвращают значения целого типа:

trunc(x) – отбрасывание десятичных знаков после точки;

round(x) – округление до целого.

1.7.2. Вещественные типы

В языке Паскаль существует несколько типов для представления действительный чисел. Однако чаще всего для их представления используется тип *Real*.

Таблица 1.4

Вещественные типы в Паскале

Тип	Диапазон изменения	Число цифр	Память, байт
real	$2{,}9\cdot10^{-39}\ldots1{,}7\cdot10^{38}$	11…12	6
single	$1{,}5\cdot10^{-45}\ldots3{,}4\cdot10^{38}$	7…8	4
double	$5\cdot10^{-324}\ldots1{,}7\cdot10^{308}$	15…16	8
extended	$1{,}9\cdot10^{-4951}\ldots1{,}1\cdot10^{4932}$	19…20	10
comp	$-2^{63}+1\ldots2^{63}-1$	19…20	8

Число цифр определяет точность, с которой будет храниться вещественное число. Тип *Comp* содержит только целые значения, которые представляются в вычислениях как вещественные.

Над действительными числами выполнимы операции сложения (+), вычитания (-), умножения (·) и деления (/). Результатом этих операций является также действительное число. Даже если хотя бы один из операндов вещественный, то результат этих операций также будет вещественным. Операция деления (/) дает вещественный результат и в случае двух целых операндов.

Для действительных чисел допустимы такие же операции отношения (сравнения), что и для целых чисел.

Стандартная функция *abs(x)* – модуль *x* – от целого аргумента дает целый результат, а от вещественного – вещественный, как и *sqr(x)* – квадрат *x*.

Нижеперечисленные функции дают вещественный результат как для вещественного, так и для целого аргумента:

sin(x), cos(x), ln(x), exp(x), sqrt(x), arctan(x).

Функция *int* возвращает в виде действительного значения целую часть аргумента, *frac* возвращает дробную часть аргумента.

Функции *trunc* и *round* возвращают результат целого типа. Первая отсекает дробную часть от аргумента, а вторая выполняет округление до ближайшего целого.

Функция *random* без аргументов возвращает равномерно распределенное случайное число от 0 до 1.

Не имеющая аргументов функция *pi* возвращает число Пифагора.

Переменные и константы вещественного типа нельзя использовать:

– в функциях *pred*, *succ*, *ord*;

– в качестве индексов массивов;

– в операторах передачи управления в качестве меток.

1.7.3. Булевский тип (*Boolean*)

Переменная булевского типа принимает значения *true* (истина) или *false* (ложь). Эти величины упорядочены следующим образом: *false* < *true*. Операции *and*, *or*, *not* (применяемые к булевским операндам) дают булевские значения.

Операция *not* (отрицание, операция НЕ): выражение *not a* имеет значение, противоположное значению *a*.

Операция *and* (логическое умножение, пересечение, операция И): выражение *a and b* дает значение *true* только в том случае, если *a* и *b* имеют значение *true*. Во всех остальных случаях значения выражения *a and b – false*.

Операция *or* (логическое сложение, объединение, операция ИЛИ): выражение *a or b* дает значение *false*, только в том случае, если *a* и *b* имеют значение *false*. Во всех остальных случаях результат – *true*.

Стандартные булевские функции:

odd(x) = true, если *x* нечетный (*x* – целый);

eoln(x) = true, если встретился конец строки текстового файла *x*;

eof(x) = true, если встретился конец файла *x*.

В остальных случаях эти функции принимают значение *false*.

1.7.4. Символьный тип (*Char*)

Переменная типа *char* может принимать значения из определенной упорядоченной последовательности символов. Переменная этого типа занимает один байт и принимает одно из 256 значений кода *ASCII* (американский стандартный код для обмена информацией). Символы упорядочены в соответствии с их кодом, поэтому к данным символьного типа применимы операции отношения.

В программе вместо символа можно использовать его код, состоящий из # и номера кодируемого символа (например, #51). Обычно символы, имеющие экранное представление, записывают в явном виде, заключив в апострофы (например, `*A*`, `*b*`, `*`*`).

Две стандартные функции позволяют поставить в соответствие данную последовательность символов множеству целых неотрицательных чисел (порядковым номерам символов последовательности). Эти функции называются функциями преобразования:

ord(ch) – выдает номер символа (нумерация с нуля),

chr(i) – выдает *i*-й символ из таблицы символов.

Пример

ord(`H`) – выдает номер символа *H* в последовательности всех символов, используемых транслятором.

chr(15) – выдает 15-ый символ этой последовательности.

Кроме того, для символьных переменных применяются такие функции:

pred(ch) – возвращает предыдущий символ;

succ(ch) – возвращает следующий символ;

upcase(ch) – преобразует строчную букву в заглавную (обрабатывает буквы только латинского алфавита).

Можно также использовать процедуры *inc* и *dec*.

1.7.5. Явное преобразование типов

В ряде случаев в *Паскале* происходит автоматический переход от одного типа данных к другому (от целого к действительному, от символьного к строковому и т. д.). Существует также ряд функций, осуществляющих преобразование типов (*ord*, *chr*, *trunc*, *round*). Наряду с этим в Паскале возможно явное преобразование типов (ретипизация данных). Для того чтобы осуществить явное преобразование типа, необходимо использовать имя типа аналогично тому, как используется имя функции. В качестве параметра в этом случае указывается имя преобразуемой переменной.

Преобразовать можно любой тип к любому другому, однако следует выполнять требование: в операторе присваивания переменная слева должна занимать в памяти столько же или больше байт, сколько занимает преобразуемое значение.

1.7.6. Перечисляемый тип

В программу можно ввести и переменные какого-либо типа, не совпадающие ни с одним из стандартных типов. Такой тип задается перечислением значений при объявлении типа; любое из этих значений может принимать переменная данного типа, объявленная далее в программе. Общий вид описания перечисляемого типа:

```
type
   slova = (word1, word2, word3);
var
   w: slova;
```

здесь *slova* – идентификатор типа (произвольный), *word1, word2, word3* – конкретные значения, которые может принимать переменная *w*, принадлежащая типу *slova*. Значения данного типа считаются упорядоченными, т.е. описание типа одновременно вводит упорядочение *word1* < *word2* < *word3*. Порядковые значения отсчитываются с 0.

К переменным типа перечисления можно применять функции *ord*, *pred*, *succ* и процедуры *inc* и *dec*.

Ко всем переменным одного и того же скалярного типа применимы операции отношения: =, <>, <=, >=, <, >.

Особенностью переменных типа перечисления является то, что их значения нельзя вводить с клавиатуры и выводить на экран (но можно использовать при работе с типизированными файлами).

Пример 1. Здесь определено, что *red* < *yellow* < *green* < *blue*. Переменная типа *color* может принимать одно из перечисленных значений.
type color = (red, yellow, green, blue);

Функция *succ(x)*. По элементу *x* определяется та упорядоченная последовательность, которой принадлежит *x*, и выдается элемент, следующий за *x* в этой последовательности.

Пример 2. Пусть задана последовательность букв в алфавитном порядке. Тогда *succ(A)* есть *B*; *succ(L)* есть *M* и т. д. Для примера 1: *succ(red)* есть *yellow*.

Функция *pred(x)*. По элементу *x* определяется последовательность, которой принадлежит *x*, и выдается предыдущий элемент этой последовательности.

Пример 3. *pred(F)* есть *E*; *pred(Z)* есть *Y* и т. д.

Функция *ord(x)*. Выдается номер элемента *x* в последовательности.
Пример 4. Для примера 1: *ord(red)* равен 0, а *ord(green)* равен 2.

1.7.7. Диапазонный, или интервальный, тип

Для переменной скалярного (перечисляемого) типа можно указать некоторое подмножество значений, которые может принимать данная переменная.

Общий вид:
a: min..max;
здесь *a* – интервальная переменная, *min* – левая граница, *max* – правая граница подмножества (диапазона). Границы диапазона разделяются двумя точками; граница *min* всегда должна быть меньше *max*.

Константы *min* и *max* должны принадлежать одному и тому же типу. Они определяют базовый тип переменной *a*. Так, если границы являются целыми числами типа *integer*, то под переменную *a* будет выделен такой же объем памяти, что и под тип *integer*. Однако переменная *a* сможет принимать только те значения, которые определены границами ее диапазона.

Пример 1. Пусть переменная *k* должна принимать значения из множества -10..10. Тогда ее следует объявить как *k*: -10..10. При этом базовым типом переменной *k* является тип *integer*, т.к. границами диапазона являются целые константы -10 и 10.

Пример 2. Если переменная *b* может принимать одно из значений *red*, *yellow*, *green*, то эту переменную можно описать так: *b: red..green;* базовым типом для *b* является тип *color*.

1.8. Логические выражения и логические операции

1.8.1. Простые логические выражения

Для того чтобы программа была нелинейной (т. е. в зависимости от ситуации выполнялись разные инструкции), в языках программирования используются логические выражения, результат которых может быть либо правдой (*true*), либо ложью (*false*). Результат логических выражений обычно используют для определения пути выполнения программы.

Простые логические выражения являются результатом операций отношения между двумя операндами (значениями). В примерах ниже операндами являются значения переменных *x* и *y*. Операндами могут быть числа, символы и другие типы данных.

В Паскале предусмотрены следующие операторы отношений: меньше: $x < y$; меньше или равно: $x <= y$; больше: $x > y$; больше или равно: $x >= y$; равно: $x = y$; не равно: $x <> y$.

1.8.2. Булевы типы

Результатом логического выражения всегда является булево (логическое) значение. Булев тип данных (*boolean*) может принимать только два значения (*true* или *false*). Эти величины упорядочены следующим образом: *false* < *true*. Это значит, что данные булевого типа являются не только результатом операций отношения, но и могут выступать в роли операндов операции отношения. Также к ним можно применять функции *ord, succ, pred,* процедуры *inc* и *dec*.

Значение типа *boolean* занимает в памяти 1 байт.

1.8.3. Битовая арифметика и операции над битами

В *Паскале* над целыми типами (*byte, shortint, word, integer, longint* и их диапазоны) допустимы побитовые операции.

1.8.3.1. Логические операции над битами

Над битами двух целых операндов можно выполнять логические операции: *not, and, or, xor*. Отличие между побитовыми и логическими операциями состоит в том, что побитовые (поразрядные) операции выполняются над отдельными битами операндов, а не над их значением в десятичном (обычно) представлении.

Например, число 5 в двоичном представлении (в одном байте) имеет значение 0000 0101. Операция *not* инвертирует биты и мы получим 1111 1010, т. е. число 250. Если побитовую операцию *or* использовать к числам 5 (0000 0101) и 3 (0000 0011), то получится число 7 (0000 0111).

1.8.3.2. Операции циклического сдвига

В Паскале определены еще две операции над данными целого типа, имеющие тот же уровень приоритета, что и операции *and*, *****, */*, *div* и *mod*. Это операции *shl* и *shr*, которые сдвигают последовательность битов на заданное число позиций влево или вправо соответственно. При этом биты, которые выходят за разрядную сетку, теряются. При выполнении операции *shl* освободившиеся справа биты заполняются нулями. При выполнении операции *shr* освободившиеся слева биты заполняются единицами при сдвиге вправо отрицательных значений и нулями в случае положительных значений.

С помощью операции *shl* возможна замена операции умножения целых чисел на степени двойки. Следующие пары выражений приводят к одинаковому результату:

$$(a \text{ shl } 1) = a \cdot 2, \qquad (a \text{ shl } 2) = a \cdot 4, \qquad (a \text{ shl } 3) = a \cdot 8.$$

1.8.3.3. Практическое значение побитовых операций

Операция **and** практически всегда используется только для достижения одной из двух целей: проверить наличие установленных в единицу битов или осуществить обнуление некоторых битов.

Подобная проверка нужна, если число представляет набор признаков с двумя возможными значениями (набор флагов). Так, многие системные ячейки памяти содержат сведения о конфигурации компьютера или его состоянии. При этом установка бита с конкретным номером в лог. 1 трактуется как включение какого-либо режима, а в лог. 0 – выключение этого режима.

Пусть переменная *a* имеет тип *byte* и является байтом с восемью флагами. Необходимо проверить состояние бита с номером 5 (биты нумеруются справа налево от 0 до 7). Единица в бите 5 – это пятая степень числа 2, т. е. 32 (0010 0000). Поэтому, если в пятом бите переменной *a* стоит единица, то выполняется условие (*a and* 32) = 32, которое можно проверить в операторе ветвления *if*. Если необходимо проверить состояние нескольких одновременно установленных в единицу

битов, то нужно вычислить соответствующее число как сумму степеней числа 2, где показатели степени равны номерам битов, установленных в лог. 1. Например, для битов 5, 2 и 0 имеем 32+4+1=37. Если *a* имеет среди прочих единицы в битах 5, 2, 0, то выполнится условие (a *and* 37) = 37.

Пусть нужно обнулить какой-либо бит в переменной *a* типа *byte* (например, бит 3). Определим сначала число, содержащее единицы во всех битах, кроме третьего. Максимальное число, которое можно записать в тип *byte*, равняется 255. Чтобы в нем обнулить третий бит, вычтем из этого числа третью степень числа 2 (255-8=247). Если это число логически умножить на *a*, то его единицы никак не скажутся на состоянии переменной *a*, а нуль в третьем бите независимо от значения третьего бита переменной *a* даст в результате 0. Итак, имеем *a := a and 247*. Аналогично можно обнулить несколько битов.

Операция **or** применяется при установке в единицу отдельных битов двоичного представления целых чисел. Так, чтобы установить бит 4 переменной *a* в единицу без изменения остальных битов, следует записать *a := a or 16*, где 16 – четвертая степень числа 2. Аналогично устанавливаются в единицу несколько битов.

Операция **xor** применяется для смены значения бита (или нескольких битов) на противоположное (1 на 0 или 0 на 1). Так, чтобы переключить в противоположное состояние 3-й бит переменной *a*, следует записать *a := a xor 8*, где 8 – третья степень числа 2.

1.8.3.4. Порядок выполнения операций

В сложных выражениях порядок операций определяется их приоритетом. Операции одного приоритетного уровня выполняются слева направо. Порядок операций можно изменить, воспользовавшись круглыми скобками. Значения функций вычисляются раньше, чем выполняются другие операции. Приоритетные уровни операций следующие (по убыванию приоритета):

– одноместные (унарные) операции: *not*;
– мультипликационные операции: *, /, *div*, *mod*, *and*;
– аддитивные операции: +, -, *or*, *xor*;
– операции отношения: <, <=, >, >=, =, <>.

2. АЛГОРИТМИЗАЦИЯ ПРОСТЫХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

2.1. Алгоритм и его свойства

Для решения любой задачи необходимо четко представлять последовательность действий по достижению конечного результата, т. е. алгоритм.

Алгоритм – система точно сформулированных правил, определяющая процесс преобразования допустимых исходных данных (входной информации) в желаемый результат (выходную информацию) за конечное число шагов.

Алгоритм решения задачи имеет ряд обязательных свойств.

Дискретность – алгоритм должен представлять процесс решения задачи как последовательное выполнение некоторых простых шагов. При этом для выполнения каждого шага алгоритма требуется конечный отрезок времени, т. е. преобразование исходных данных в результат осуществляется во времени дискретно.

Детерминированность (определенность). В каждый момент времени следующий шаг работы однозначно определяется состоянием системы. Таким образом, алгоритм выдает один и тот же результат (ответ) для одних и тех же исходных данных. С другой стороны, существуют вероятностные алгоритмы, в которых следующий шаг работы зависит от текущего состояния системы и генерируемого случайного числа. Однако при включении метода генерации случайных чисел в список «исходных данных» вероятностный алгоритм становится подвидом обычного.

Понятность – алгоритм должен включать только те команды, которые доступны исполнителю и входят в его систему команд.

Завершаемость (конечность) – при корректно заданных исходных данных алгоритм должен завершать работу и выдавать результат за конечное число шагов.

Массовость (универсальность). Алгоритм должен быть применим к разным наборам исходных данных при решении задач данного класса.

Результативность – завершение алгоритма определенными результатами. Алгоритм содержит ошибки, если приводит к получению неправильных результатов либо не дает результатов вовсе. Алгоритм не содержит ошибок, если он дает правильные результаты для любых допустимых исходных данных.

2.2. Способы представления алгоритмов

Возможны различные способы представления алгоритмов.

1. Словесно-формульное описание алгоритма. Это описание алгоритма с помощью слов или формул.

2. Описание алгоритма на алгоритмическом языке. Алгоритмический язык – это средство для записи алгоритмов в аналитическом виде, промежуточном между записью алгоритма на естественном языке и записью на языке ЭВМ (языке программирования).

3. Описание алгоритма на языке программирования.

4. Графическое описание алгоритма. Описание с помощью схем.

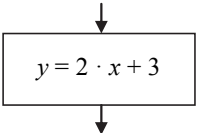
Схема алгоритма представляет собой систему связанных геометрических фигур. Каждая фигура обозначает один этап процесса решения задачи и называется блоком. Порядок выполнения этапов указывается стрелками, соединяющими блоки. В алгоритме блоки стараются размещать сверху вниз и справа налево.

2.2.1. Графическое описание алгоритма

Блок начала-конца программы изображается овалом, внутри которого записываются слова «начало» (*begin*) и «конец» (*end*). Этот блок имеет один выход (блок начало) или один вход (блок конец). В программе возможно наличие только одного блока начала и одного блока конца программы. В подпрограмме в блоке конца подпрограммы записывается слово «возврат» (*return*).

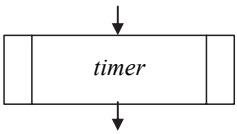
Блок ввода-вывода информации изображается параллелограммом, внутри которого записывается набор входных (блок ввода) или выходных (блок вывода) переменных. Этот блок имеет один вход и один выход.

Блок вычислений изображается прямоугольником, внутри которого записываются операция присваивания и вычислительные действия. Этот блок имеет один вход и один выход.

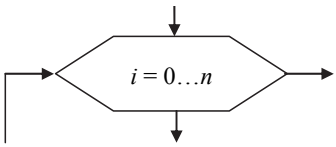


Логический блок изображается ромбом, внутри которого записывается условие ветвления. Этот блок имеет один вход и два выхода. Первый выход (когда условие истинно) обозначается «да» или «+», второй выход (когда условие ложно) обозначается «нет» или «-».

Блок подпрограммы изображается в виде прямоугольника с тремя полями, в среднее поле записывается имя вызываемой подпрограммы. Этот блок имеет один вход и один выход.



Блок модификации (заголовок цикла со счетчиком) изображается в виде выпуклого шестиугольника, внутри которого записывается имя переменной цикла, ее начальное и конечное значения. Этот блок имеет два входа и два выхода.



Блоки необходимо стараться размещать так, чтобы стрелки (пути решения) не пересекались. Это упростит визуальное восприятие и понимание алгоритма.

2.3. Виды алгоритмов

2.3.1. Линейный вычислительный алгоритм

Под линейным вычислительным алгоритмом (рис. 2.1) будем понимать последовательность действий, состоящую из команд ввода, вывода и присваивания значений некоторым величинам. При описании таких алгоритмов используется только одна управляющая структура – следование.

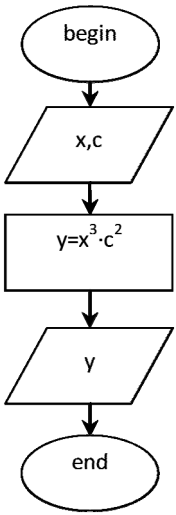


Рис. 2.1. Линейный алгоритм

Из определения алгоритма следует, что он должен начинаться с описания величин, которые используются при вычислениях. Такие величины подразделяют на переменные и константы.

Переменная – именованная величина, значение которой может меняться в ходе вычислений. Имя переменной – это последовательность из букв латинского алфавита, десятичных цифр и символов подчеркивания, начинающаяся не с цифры. Такую последовательность называют идентификатором. Переменные бывают различных типов: целые, вещественные, символьные и другие, поэтому при их описании необходимо указывать тип каждой переменной.

Константа – это величина, представляющая фиксированное числовое, строковое или символьное значение.

После описания переменных и констант (если они есть) следует описание действий, которые и составляют собственно алгоритм решения поставленной задачи.

Помимо операций ввода-вывода основным элементарным действием в алгоритме является операция присваивания значения переменной. Формат команды присваивания: <имя переменной> = <выражение>

Знак «=» следует читать как «присвоить». Под командой присваивания понимается последовательность действий:

– вычисляется выражение;

– полученное значение записывается в переменную.

Отметим три основных свойства команды присваивания:

1) до тех пор, пока переменной не присвоено никакое значение, она остается неопределенной;

2) значение, которое было присвоено переменной, сохраняется в ней до следующей операции присваивания другого значения этой переменной;

3) значение, вновь присвоенное переменной, заменяет ее предыдущее значение.

Пример программы на языке Паскаль, реализующей алгоритм вычисления значения некоторой функции, приведенный на рис. 2.1.

```
Program LineAlg;                    {объявляем имя программы}
uses wincrt;                        {используем встроенную библиотеку}
var                                 {начало раздела описания переменных }
  c, x, y : real;                   {описание действительных переменных}
begin                               {начало раздела операторов}
  write(`введи c=`); read(c);       {диалоговый режим ввода переменной c}
  write(`введи x=`); read(x);       {диалоговый режим ввода переменной x}
  y := x · SQR(x · c);              {вычисление значения функции y}
  write(`y=`, y);                   {вывод результатов на экран}
end.                                {конец программы}
```

Задание 1. Линейный вычислительный алгоритм

В соответствии с вариантом задания необходимо вычислить значение заданной функции с заданным набором исходных данных. При этом одно из исходных данных принимается константой; другая исходная величина – переменная, которая вводится в диалоговом режиме; остальные исходные данные – переменные, которые задаются через оператор присваивания.

Цель работы: освоение навыков программирования линейных вычислительных процессов.

При выполнении всех заданий необходимо:

– ознакомиться с условиями задачи;

– составить блок-схему алгоритма решения задачи;

– на основании полученного алгоритма составить программу решения задачи на ЯВУ Паскаль;

– ввести программу в ЭВМ и получить результаты решения задачи;

– оформить отчет по работе.

Таблица 2.1

Варианты к заданию 1

Номер варианта	Исходные данные	Функция
1	$a = 1{,}713$ $y = 0{,}29$ $b = 12{,}372$ $q = 0{,}341$ $t = 0{,}475$	$Z = \dfrac{a^2 \sin y}{y^4 + \sqrt{1 + 2y^2 + 3y^3}} - \left(\dfrac{q}{e^t}\right)^2 + b^t$
2	$a = 1{,}627$ $b = 0{,}564$ $x = 3{,}571$	$E = \dfrac{0{,}276x^2 + \dfrac{0{,}596}{a \cdot b} + e^{a+b}}{\sqrt{21{,}76 - \sin(\pi x)} + a^2 \dfrac{\pi x}{b}}$
3	$x = 0{,}42$ $y = 1{,}441$ $h = 3{,}571$	$Z = (x^2 + y^2)^5 \sin^3\left(\dfrac{x}{4}\right)$ $D = Z^2 h + Z^{y+1{,}5}$
4	$a = 0{,}332$ $b = 3{,}746$ $s = 1{,}5576$ $d = 2{,}73$ $c = 0{,}336$ $x = 2{,}445$	$y = \dfrac{(0{,}264 + x)^2 b}{(a + x)\sin\left(\dfrac{x}{2}\right)}$ $Z = \left(\dfrac{1}{7}\dfrac{a^3 b + y}{cds}\right)^2 \sqrt{bs}$
5	$a = 1{,}816$ $b = 0{,}632$ $x = 0{,}1983$	$D = \dfrac{ax^5\sqrt{a^2 bx}}{x + \sin\left(\pi\dfrac{ab}{x}\right)} \operatorname{arctg}\dfrac{a}{b}$
6	$a = 1{,}932$ $b = 0{,}594$ $x = 0{,}2336$	$y = \dfrac{\sin^3(x - \arcsin b)}{\sqrt{\cos x + 3{,}56} + a^2} e^{\frac{a^2 x + b}{b - x}}$

Продолжение табл. 2.1

Номер варианта	Исходные данные	Функция
7	$a = 1{,}447$ $b = 1{,}724$ $x = 2{,}444$	$y = \dfrac{a\sqrt{x}\cdot\sin^2\left(\dfrac{x}{2}\right)b}{x + ae^{-x}}$ $Z = \dfrac{a\sin(2-x)^2}{(2+x)^3 y}$
8	$a = 0{,}35741$ $x = 1{,}7983$	$y = e^{a\sin(\pi x)}\ln\sqrt{\lvert a - x^2\rvert} + a^x$
9	$a = 0{,}34721$ $x = 1{,}7996$	$y = e^{\cos(\pi x^2)} - \sin\left(\pi\sqrt{\lvert a^x - x^a\rvert}\right)$
10	$a = 0{,}34721$ $x = 1{,}7996$	$y = \dfrac{1}{\pi}\ln\lvert a^2 - \sin(\pi x)\rvert - \sqrt{a^2\cdot\operatorname{arctg}\dfrac{x}{a}}$
11	$a = 0{,}96$ $n = 1{,}09$ $k = 0{,}96$ $x = 1{,}32$	$y = \dfrac{\operatorname{arctg}^2\left(\pi\dfrac{x}{na}\right) - \sin\left(a\cdot(x-k)\right)}{\ln\left\lvert arctg^2\left(\pi\dfrac{x}{n\cdot a}\right) - \sin(a\cdot(x-k))\right\rvert} + a^x$
12	$a = 0{,}9$ $x = 1{,}24$ $k = 1{,}27$ $n = 1{,}07$	$Z = \sin(a(x-k)) + e^{\operatorname{arctg}^2\left(\pi\frac{x}{na}\right)\sin(a(x-k))}$
13	$a = 0{,}93$ $x = 1{,}27$ $n = 1{,}01$ $k = 0{,}98$	$Z = \dfrac{\ln\left\lvert k^{ax} - \operatorname{arctg}^2\left(\pi\dfrac{x}{na}\right) + 0{,}56a\right\rvert}{\sqrt[3]{\left\lvert k^{ax} - \operatorname{arctg}^2\left(\pi\dfrac{x}{na}\right) + 0{,}56a\right\rvert}} - 2{,}97e^{kx}$
14	$a = 0{,}34721$ $x = 1{,}7896$	$D = a\sin\left(\pi\dfrac{a}{x}\right) - \sqrt{\left\lvert\ln\lvert a^2 - x^2\rvert + 0{,}036x\right\rvert}$
15	$a = 0{,}96$ $n = 1{,}07$ $x = 1{,}24$ $k = 0{,}93$	$Z = a\sin(\pi x) + \ln\left(2\left(a^x + \sin^2\left(\pi\dfrac{kx}{n+a}\right)\right)\right)$

Окончание табл. 2.1

Номер варианта	Исходные данные	Функция
16	$a = 0{,}93$ $n = 1{,}05$ $x = 1{,}23$ $k = 0{,}97$	$y = e^{a^x + \sin^2\left(\pi\frac{kx}{n+a}\right)} \sqrt[3]{a^x + \sin^2\left(\pi\frac{kx}{n+a}\right)}$
17	$a = 0{,}35712$ $x = 1{,}6983$	$y = e^{a^2}\sin^2(\pi x) - \sqrt{\lvert a - x\rvert \arcsin(a^x)}$
18	$a = 0{,}35712$ $x = 1{,}7312$	$y = e^{a^2 \cdot \sin^2(\pi x)} - \sqrt{\lvert a - x\rvert \arccos(a^x)}$
19	$a = 1{,}48$ $b = 1{,}47$ $x = 2{,}51$	$Z = \cos(a(x-b)) + e^{\operatorname{arcctg}^2\left(\pi\frac{x}{ab}\right)\cos(a(x-b))}$
20	$a = 2{,}4$ $b = 3{,}7$ $x = 1{,}54$	$D = \arcsin\left(\pi\frac{a}{x}\right) - \sqrt{\left\lvert \ln\left\lvert b^2 - x^2\right\rvert + bx\right\rvert}$

Вопросы для самопроверки

1. Основные свойства алгоритма?
2. Что такое линейный вычислительный процесс?
3. Какие типы переменных использовались в данной работе?
4. Что понимается под операцией присвоения? Её свойства?
5. Какими способами можно задать численное значение величины в программе на языке Паскаль?
6. Структура программы на языке Паскаль?

2.3.2. Разветвляющийся вычислительный алгоритм

Алгоритм разветвляющегося вычислительного процесса – алгоритм, в котором в зависимости от значений некоторого признака производится выбор одного из нескольких направлений, называемых ***ветвями***.

В основе организации разветвления лежит проверка логического условия, которое может быть истинно или ложно.

Частный вид логического условия – это операции сравнения типа «равно» (=), «неравно» (≠), «больше» (>), «меньше» (<),«больше или равно» (≥),«меньше или равно» (≤).

Условие при его словесном описании может выглядеть следующим образом: «Если …, то …», «Если …, то …, иначе …», «Если …, то …, иначе если…, то…», и т. д. Данные конструкции называются ***логическими выражениями***.

При составлении разветвляющихся алгоритмов можно столкнуться с различными видами записи блок-схемы. Это зависит от типа ветвления и от выполнения действий той или иной ветвью. Ветвления бывают: полные, неполные, последовательные и вложенные.

Полное ветвление. Эта структура (рис. 2.2) соответствует логическому выражению:

«Если …то …иначе …».

Неполное ветвление. К неполным ветвлениям относятся алгоритмы, выполняющие следующую структуру логического выражения:

«Если …то …».

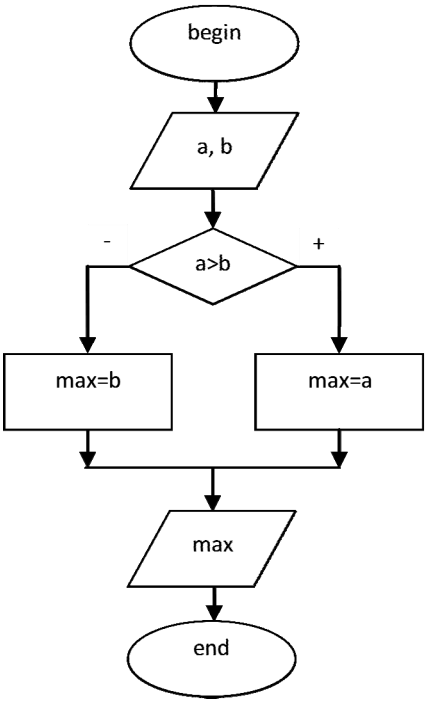


Рис. 2.2. Разветвляющийся алгоритм

Последовательные и вложенные ветвления представляют собой несколько иную структуру логического выражения. Они могут быть как полными, так и неполными. Полное и неполное – это вид самого ветвления, а последовательное и вложенное – это уже комбинации ветвлений, используемых в алгоритме. Они используются при программировании задач с несколькими условиями.

Последовательные ветвления – это те, которые следуют друг за другом. После выполнения действий ветвей первого условия следует второй условный блок и т. д.

В структуре ***вложенного ветвления*** следующая особенность: одна или обе ветви условия могут продолжаться не блоками вычислительных операций или ввода-вывода, а дополнительным блоком условия.

2.3.2.1. Условные операторы в языке Паскаль

Если в процессе выполнения программы требуется реализовать разный набор команд в зависимости от произошедших до этого событий, то в языках программирования это достигается с помощью специальных конструкций – условных операторов.

Чаще всего в качестве условного оператора в языках программирования используется конструкция *if-else* или ее сокращенный вариант *if*. Также существует оператор выбора *case*, который имеет более специфичное применение.

Оператор if-else

Когда выполнение основной ветки программы доходит до условного оператора *if-else*, то в зависимости от результата логического выражения в его заголовке выполняются разные блоки кода. Если логическое выражение вернуло *true*, то выполняется один блок (в Паскале начинается со слова *then*), если *false* – то другой (начинается со слова *else*). После выполнения одного из вложенных блоков кода ход программы возвращается в основную ветку. Другой вложенный блок не выполняется.

Бывают неполные формы условных операторов. В таком случае вложенный в *if* блок кода выполняется только при истинности логического выражения заголовка. В случае *false* выполнение программы сразу передается в основной блок. Понятно, что ветка *else* при этом отсутствует.

В качестве условия может стоять любое выражение, результатом вычисления которого является одно из булевых значений – *true* или *false*.

Непосредственно после *then* может стоять только один оператор. При необходимости выполнения нескольких операторов они должны быть заключены в операторные скобки *begin-end*.

Допустимо вложение одного оператора *if* (или *if-else*) в другой. При этом следует соблюдать осторожность, т. е. бывает трудно определить, какому *if* (внешнему или внутреннему) принадлежит ветка *else*. Рекомендуют использовать вложенную конструкцию *if* только в ветке *else*. К тому же в языке Паскаль действует следующее правило: каждому *then* соответствует ближайшее *else*, не задействованное при установлении соответствия с другим *then*. Глубина вложенности операторов *if* может быть сколь угодно большой, но разобраться в таком коде будет очень сложно.

Пример программы на языке Паскаль, реализующей алгоритм поиска максимального из двух чисел, приведенный на рис. 2.2:

```
Program Vetvlen;                  {объявляем имя программы}
uses wincrt;                      {используем встроенную библиотеку}
var                               {начало раздела описания переменных}
   a, b, max : real;              {описание действительных переменных}
begin                             {начало раздела операторов}
   write(`введи a=`); read(a);    {диалоговый режим ввода переменной a}
   write(`введи b=`); read(b);    {диалоговый режим ввода переменной b}
   if a>b                         {поиск максимального из двух чисел}
     then max := a                {при истине присваиваем результату
                                                        значение a}
     else max := b;               {иначе присваиваем результату значение b}
   write(`max=`, max);            {вывод результата на экран}
end.                              {конец программы}
```

Оператор case (оператор выбора)

Кроме оператора *if* в языке программирования Паскаль предусмотрен так называемый переключатель *case*. Его можно трактовать как некий вопрос, имеющий большое число ответов (а не только два, как это, например, в операторе *if-else*). Однако в отличие от *if case* имеет ряд принципиальных ограничений. Его формат следующий:

```
case селектор of
        значение1: оператор1;
        значение2: оператор2;
        значение3: оператор3;
        else операторN
end;
```

В заголовке оператора *case* вместо логического выражения фигурирует переменная, которую называют **селектором**. До этого в программе ей присваивается какое-либо значение. Эта переменная может иметь только перечисляемый тип (например, она не может быть вещественного типа). По ходу выполнения оператора *case*, значение переменной-селектора сравнивается с различными, описанными в нем альтернативами (метками-значениями).

Как только совпадение будет найдено, выполняется блок кода при данной метке и происходит выход в основную ветку программы. Значения-метки являются константами, которые может принимать селектор. Их тип и тип селектора должны быть совместимы по присваиванию.

Если совпадений не будет, то выполняется блок *else*. Если блок *else* отсутствует (он является не обязательным), то никакой блок кода в операторе *case* не выполняется.

На использование оператора выбора накладываются следующие ограничения:

– селектор должен иметь какой-либо порядковый тип;

– каждая альтернатива должна быть константой, диапазоном, списком диапазонов, но не переменной или выражением.

Задание 2. Разветвляющийся вычислительный алгоритм

В соответствии с вариантом задания необходимо вычислить значение заданной функции с заданным набором исходных данных. При этом одно из исходных данных принимается константой; переменная x вводится в диалоговом режиме; остальные исходные данные – переменные, которые задаются через оператор присваивания.

Цель работы: освоение навыков программирования разветвляющихся вычислительных процессов.

Таблица 2.2

Варианты к заданию 2

Номер варианта	Исходные данные	Функция
1	$a = 1{,}627$ $b = 0{,}21$	$y = \begin{cases} b - \dfrac{x^2\left(a - \dfrac{x+1{,}5}{\sin(x\pi)}\right)}{\sqrt{2{,}76a + b^{ab}}}, & \text{при } 0{,}1 < x < 0{,}21 \\ ax - \dfrac{\left(\dfrac{\sin\left(\dfrac{b}{x\pi}\right)}{\pi/\mathrm{b}} + \cos(x\pi)\right)}{5 - \left(x^2 + x + \sqrt{bx}\right)^2}, & \text{при } x > 0{,}21 \end{cases}$
2	$a = 1{,}53$ $b = 0{,}56$	$y = \begin{cases} \sqrt{x^2 + a} - \left\lvert \sin\left(\pi(ax - b)\right)\right\rvert, & \text{при } 0 < x < 0{,}21 \\ e^{a\sin(\pi x)} + a^x, & \text{при } x > 0{,}21 \end{cases}$
3	$a = 1{,}53$	$y = \begin{cases} \ln\sqrt{\left\lvert a - x^2\right\rvert + a^x}, & \text{при } 0{,}2 < x < 0{,}85 \\ \pi^a e^{\sin\left(\pi\frac{x}{a}\right)}, & \text{при } x = 0{,}85 \\ 0{,}364x^3 a \sin\dfrac{x}{4}, & \text{при } x > 0{,}85 \end{cases}$
4	$a = 0{,}34$	$y = \begin{cases} \left\lvert a^2 - x\right\rvert + (9x^2 - 1{,}25x^3)\sin\dfrac{x}{2}, & \text{при } 1 < x < 9 \\ 0{,}57x^3 - 0{,}024\cos\dfrac{x}{4}, & \text{при } 0 < x < 1 \\ (1{,}54x^2 + 0{,}044)\,\mathrm{tg}\dfrac{x}{2}, & \text{при } x > 9 \end{cases}$

Продолжение табл. 2.2

Номер варианта	Исходные данные	Функция
5	$a = 1{,}435$ $c = 40{,}7$	$y = \begin{cases} 2xe^{-x} + \pi^a e^{\sin\frac{x}{a}}, \text{ при } 0{,}36 < x < 0{,}76 \\ x^2 + a\sqrt{x + \lvert c^3 \rvert}, \text{ при } 0{,}76 < x < 0{,}84 \\ \sqrt{a^2 + x^2 e^x}, \text{ при } 0{,}84 < x < 0{,}97 \end{cases}$
6	$a = 0{,}56$ $b = 0{,}734$ $c = 0{,}421$	$y = \begin{cases} a \cdot \operatorname{arctg}\sqrt{(x+5)^3}, \text{ при } x < 0{,}365 \\ b \cdot \operatorname{arctg}\dfrac{x}{(x+1)}, \text{ при } 0{,}365 < x < 0{,}525 \\ c \cdot e^{x+3} \cdot \operatorname{arctg}\dfrac{x-1}{x+1}, \text{ при } x > 0{,}525 \end{cases}$
7	$a = 3{,}1$	$y = \begin{cases} \cos(ax) + \ln\lvert x/2 \rvert, \text{ при } x < 0 \\ \lvert x^2 + ax \rvert, \quad \text{при } 0 < x < 1 \\ e^x, \quad \text{при } x > 1 \end{cases}$
8	$a = 3{,}7$	$y = \begin{cases} \sin\left(e\sqrt{\lvert x \rvert}\right) + \ln\lvert 2x \rvert, \text{ при } x < -2 \\ \lvert x^2 + 1 \rvert, \text{ при } -2 < x < -1 \\ e^{2x^2} + \operatorname{arctg}\left(\dfrac{x}{a}\right), \text{ при } x > -1 \end{cases}$
9	$a = 3{,}2$ $b = 8{,}1$	$y = \begin{cases} \sin(bx+1) + \dfrac{b-a}{\lvert x \rvert}, \text{ при } x < 0 \\ x^{ax} + 2x, \text{ при } 0 < x \le 5 \\ e^{\frac{x}{2}} + \operatorname{arctg}(ax), \text{ при } x > 5 \end{cases}$

Продолжение табл. 2.2

Номер варианта	Исходные данные	Функция
10	$a = 3{,}4$ $b = 0{,}5$	$y = \begin{cases} \dfrac{\sin x}{x} + a \cdot \ln\left\lvert\dfrac{2x}{a}\right\rvert, & \text{при } x < 0 \\ \arcsin x, & \text{при } 0 < x < 1 \\ b \cdot \sin(x-a) + e^{\sin x^2}, & \text{при } x > 1 \end{cases}$
11	$a = 3{,}1$ $b = 8{,}3$	$y = \begin{cases} 0{,}25\,\mathrm{tg}\,(ax) + \sin\lvert(b-a)\cdot x\rvert, & \text{при } x < 0 \\ 1, & \text{при } x = 0 \\ b \cdot \ln\lvert 10x\rvert + (b-a)\cdot\cos x^2, & \text{при } x > 0 \end{cases}$
12	$a = 3{,}2$ $b = 8{,}7$	$y = \begin{cases} b \cdot \ln\lvert 10x\rvert \cdot \mathrm{arctg}\, x, & \text{при } x < -1 \\ \lvert x\rvert \cdot \sqrt{\lvert x^2 + ax\rvert} - e^x, & \text{при } -1 < x < 0 \\ e^{\frac{x}{a}} - \sin x^2, & \text{при } x > 0 \end{cases}$
13	$a = 5{,}1$ $b = 8{,}3$	$y = \begin{cases} \ln\lvert\sin(2x) + a^2\rvert, & \text{при } x < -1 \\ \arccos(x+1), & \text{при } -1 < x < 1 \\ x^3 + bx^2 - ae^x, & \text{при } x > 1 \end{cases}$
14	$a = 1{,}73$	$y = \begin{cases} \ln\sqrt{\lvert a + x^2\rvert + x^a}, & \text{при } 0{,}2 \le x < 0{,}87 \\ \pi^a e^{\sin\left(\frac{\pi x}{a}\right)}, & \text{при } x = 0{,}87 \\ a^2 x^3 \sin\dfrac{\pi x}{4}, & \text{при } x > 0{,}87 \end{cases}$

Продолжение табл. 2.2

Номер варианта	Исходные данные	Функция
15	$a = 0{,}34$	$y = \begin{cases} \left\|a^2 - x^2\right\| + (ax^2 - a^4x^3)\cos\dfrac{\pi x}{2a}, & \text{при } 2 < x < 8 \\ ax^3 - a^3\sin\dfrac{\pi x}{4}, & \text{при } 0 < x < 2 \\ (x^2 + a^3)\operatorname{ctg}\dfrac{\pi x}{2}, & \text{при } x > 8 \end{cases}$
16	$a = 1{,}45$ $c = 4{,}7$	$y = \begin{cases} cxe^{-x} + \pi^a e^{\sin\frac{\pi x}{a}}, & \text{при } 0{,}3 < x < 0{,}8 \\ cx^2 + a\sqrt{x + \left\|c^3 - x^{-4}\right\|}, & \text{при } 0{,}8 < x < 0{,}9 \\ \sqrt{a^c + x^2 e^{cx}}, & \text{при } 0{,}9 < x < 0{,}99 \end{cases}$
17	$a = 3{,}1$ $b = 0{,}52$	$y = \begin{cases} \dfrac{\sin x}{b \cdot x} + a \cdot \ln\left\|\dfrac{2x^b}{a}\right\|, & \text{при } x < 0 \\ a \cdot \arcsin(bx), & \text{при } 0 < x < 1 \\ b \cdot \sin(x - a) + e^{a \cdot \sin x^2}, & \text{при } x > 1 \end{cases}$
18	$a = 4{,}1$	$y = \begin{cases} \cos(ax) + \ln\left\|x/2\right\|, & \text{при } x < 0 \\ \left\|x^{2a} + ax\right\|, & \text{при } 0 < x < 1 \\ e^{ax}, & \text{при } x \ge 1 \end{cases}$
19	$a = 2{,}4$	$y = \begin{cases} \sin\left(e \cdot \sqrt{a \cdot \left\|x\right\|}\right) + \ln\left\|2x\right\|, & \text{при } x < -2 \\ \left\|x^2 + a\right\|, & \text{при } -2 \le x < -1 \\ e^{ax^2} + \operatorname{arctg}\left(\dfrac{x}{a}\right), & \text{при } x > -1 \end{cases}$

Окончание табл. 2.2

Номер варианта	Исходные данные	Функция
20	$a = 3{,}4$ $b = 8{,}6$	$y = \begin{cases} \sin(bx + a) + \dfrac{b - a}{\lvert x \rvert}, & \text{при } x < 0 \\ x^{a \cdot x} + 2 \cdot x, & \text{при } 0 < x \le 5 \\ e^{\frac{x}{2b}} + \operatorname{arctg}(ax), & \text{при } x > 5 \end{cases}$

Вопросы для самопроверки

1. Что такое разветвляющийся вычислительный процесс?
2. Какие существуют типы ветвлений?
3. Приведите примеры логических условий? Какие логические операции вы знаете?
4. Нарисуйте блок-схему алгоритма с последовательным ветвлением.
5. Нарисуйте блок-схему алгоритма с вложенным ветвлением.
6. Нарисуйте блок-схему алгоритма с полным ветвлением.
7. Нарисуйте блок-схему алгоритма с неполным ветвлением.
8. Условные операторы в языке Паскаль.

2.3.3. Циклический вычислительный алгоритм

Алгоритм циклической структуры предусматривает многократное повторение действий в одной и той же последовательности по одним и тем же математическим зависимостям, но при разных значениях некоторой специально изменяемой величины.

Циклические алгоритмы позволяют существенно сократить объем программы за счет многократного выполнения группы повторяющихся вычислений, так называемого **тела цикла**.

Специально изменяемый по заданному закону параметр, входящий в тело цикла, называется **переменной цикла**.

Переменная цикла используется для подготовки очередного повторения цикла и отслеживания условий его окончания. В качестве переменной цикла используют любые переменные, индексы масси-

вов, аргументы вычисляемых функций и тому подобные величины. Во время выполнения тела цикла параметры переменной цикла изменяются в интервале от начального до конечного значения с заданным шагом.

Следовательно, при организации циклических вычислений необходимо предусмотреть задание начального значения переменной цикла, закон ее изменения перед каждым новым повторением и ее конечное значение, при достижении которого произойдет завершение цикла.

Циклы, в теле которых нет разветвлений и других встроенных в них циклов, называют простыми. В противном случае их относят к сложным. Циклические алгоритмы разделяют на детерминированные и итерационные.

Циклы, в которых число повторений заранее известно из исходных данных или определено в ходе решения задачи, называют *детерминированными*. Для организации детерминированных циклов наиболее целесообразно использовать блок модификации, внутри которого указывается переменная цикла, ее начальное и конечное значение, а также шаг ее изменения (если шаг изменения равен единице, то его допускается не указывать). Организовать подобный цикл возможно и при использовании блока проверки условия вместо блока модификации, однако при этом несколько усложняется алгоритм и теряется его рациональность.

Циклы, в которых число повторений неизвестно из исходных данных и не определено по ходу решения задачи, называют *итерационными*. В итерационных циклах для организации выхода из тела цикла предусматривается проверка некоторого заранее заданного условия, для чего используют блок проверки условия. В итерационных циклах невозможно использовать блоки модификации, так как при организации таких циклов заранее неизвестно количество изменений переменной цикла и ее конечное значение.

В зависимости от местонахождения блока проверки условия итерационные циклы могут быть организованы как циклы *с предусловием* (блок проверки условия размещен перед телом цикла) или *с постусловием* (блок проверки условия размещен после тела цикла). Если в цикле с предусловием входящие в тело цикла команды могут не выполняться ни разу (если начальное значение параметра цикла удовлетворяет условию выхода из цикла), то в цикле с постусловием — выполняются как минимум один раз (даже если начальное значение параметра цикла удовлетворяет условию выхода из него).

Вид итерационного цикла (с пост- или предусловием) определяется условием задачи и допустимыми или возможными значениями исходных данных.

2.3.3.1. Организация цикла в языке 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. У циклов выделяют заголовок и тело. Заголовок определяет, до каких пор или сколько раз тело цикла будет выполняться. Тело содержит выражения, которые выполняются, если в заголовке цикла выражение вернуло логическую истину (*true*, не ноль). После того как достигнута последняя инструкция тела, поток выполнения снова возвращается к заголовку цикла. Снова проверяется условие выполнения цикла. В зависимости от результата тело цикла либо повторяется, либо поток выполнения переходит к следующему выражению после всего цикла.

В языке программирования Паскаль существует три вида циклических конструкций.

2.3.3.2. Цикл с предусловием (*while*)

Цикл *while* является циклом с предусловием. В заголовке цикла находится логическое выражение. Если оно возвращает *true*, то тело цикла выполняется, если *false* – то нет.

Когда тело цикла было выполнено, то ход программы снова возвращается в заголовок цикла. Условие выполнения тела снова проверяется (находится значение логического выражения). Тело цикла выполнится столько раз, сколько раз логическое выражение вернет *true*. Поэтому очень важно в теле цикла

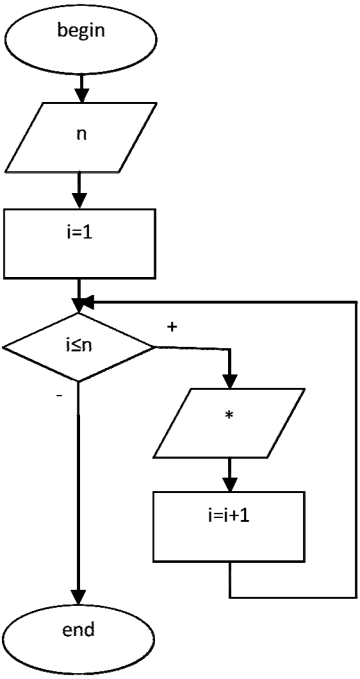


Рис. 2.3. Цикл с предусловием

предусмотреть изменение переменной, фигурирующей в заголовке цикла, таким образом чтобы когда-нибудь обязательно наступала ситуация *false*. Иначе произойдет так называемое зацикливание, одна из самых неприятных ошибок в программировании.

Пример. Ниже приведен пример программы на языке Паскаль, реализующей алгоритм (рис. 2.3), который выводит на экран заданное количество одинаковых символов «*».

```
Program Cycle1;          {задаем имя программы}
var                      {объявляем переменные}
   i, n: integer;        {задаем целочисленный тип переменным}
begin                    {начало раздела операторов}
   write ('Введите кол-во знаков= ');     {организуем диалоговый
                                                режим ввода}
   readln (n);           {считывается заданное кол-во итераций}
   i := 1;               {присваиваем переменной цикла начальное
                                                значение}
   while i <= n do       {задаем условие выполнения цикла}
    begin                {формируем составным оператором тело
                                                цикла}
    write ('*');         {выводим символ на экран}
    i := i + 1           {задаем переменной цикла следующее
                                                значение}
     end;                {окончание тела цикла}
end.                     {конец раздела операторов и программы}
```

В данной программе, если *n* будет равно 0, ни одна звездочка не будет напечатана.

2.3.3.3. Цикл с постусловием (*repeat*)

Цикл *while* может не выполниться ни разу, если логическое выражение в заголовке сразу вернуло *false*. Однако такая ситуация не всегда может быть приемлемой. Бывает, что тело цикла должно выполниться хотя бы один раз, независимо от того, что вернет логическое выражение. В таком случае используется цикл *repeat* – цикл с постусловием.

В цикле *repeat* логическое выражение стоит после тела цикла. Причем в отличие от цикла *while* здесь все наоборот: в случае *true* происходит выход из цикла, в случае *false* – его повторение.

Пример. Ниже приведен пример программы на языке Паскаль, реализующей алгоритм (рис. 2.4), который выводит на экран заданное количество одинаковых символов «*».

```
Program Cycle2;          {задаем имя программы}
var                      {объявляем переменные}
   i, n: integer;        {задаем целочисленный тип переменным}
begin                    {начало раздела операторов}
   write ('Введите кол-во знаков=');  {организуем диалоговый ввод}
   readln (n);           {считывается заданное кол-во итераций}
   i := 1;               {присваиваем переменной цикла начальное
                                                        значение}
   repeat                {задаем начало тела цикла}
    write('*');          {выводим символ на экран}
    i := i + 1           {задаем переменной цикла следующее
                                                        значение}
   until i > n;          {задаем условие выхода из цикла}
end.                     {конец раздела операторов и программы}
```

В примере, даже если *n* будет равно 0, одна звездочка все равно будет напечатана.

2.3.3.4. Оператор безусловного перехода (*goto*)

Паскаль является структурным языком программирования. Несмотря на это, в нем присутствует ряд особенностей, которые широко использовались на начальных этапах развития программирования. В те времена идея о том, что программа может рассматриваться как система логически связанных блоков, еще не оформилась. Поэтому, если требовалось изменить линейный ход программы, программисты использовали оператор безусловного перехода *goto*.

Позже большинство программистов отказались от регулярного использования оператора *goto*, однако бывают случаи, когда он может быть полезен.

Необходимо знать, что *всегда можно обойтись без оператора goto*. Его использование затрудняет чтение и понимание программы.

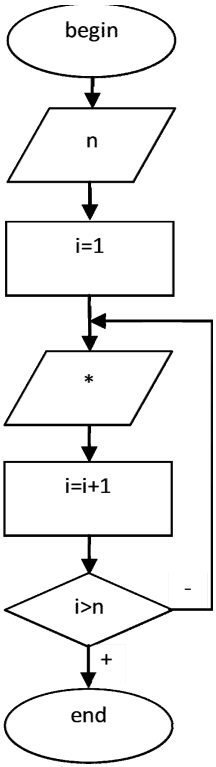


Рис. 2.4. Цикл с постусловием

Оператор *goto* осуществляет переход к оператору, помеченному специальной меткой, которая отделяется от самого оператора двоеточием. В качестве метки может быть использовано любое целое число без знака, содержащее менее четырех цифр, или любое имя. Чтобы можно было использовать метку, она должна быть в обязательном порядке объявлена в разделе меток в описательной части программы. Этот раздел начинается служебным словом *label*, после которого через запятую перечисляются метки.

2.3.3.5. Операторы прерывания цикла (*break* и *continue*)

Бывает, что цель выполнения цикла достигается раньше, чем он будет прекращен по условию выхода. Так, например, в программе для определения, является ли число простым, цикл будет выполняться *n* раз, хотя то, что число не является простым, может быть обнаружено на первых шагах цикла.

Чтобы уменьшить количество шагов цикла, можно воспользоваться оператором *goto* либо сформировать сложное условие выполнения (прекращения) цикла. Однако существуют специальные операторы, использующиеся для прерывания хода выполнения цикла. *Break* и *continue* являются процедурами, хотя обычно их называют операторами.

Оператор ***break*** выполняет полный выход из цикла, т. е. все возможные итерации цикла прерываются.

Оператор ***continue*** прерывает только текущую итерацию.

Операторы *break* и *continue* выполняются в любом из видов циклов (*repeat*, *while*, *for*) и действительны только для внутреннего цикла. Например, если нужно обеспечить принудительный выход из двойного цикла, оператор *break* должен быть расположен как во внутреннем, так и во внешнем цикле. Операторы *break* и *continue*, по сути, являются видоизмененными операторами *goto* с известной точкой, в которую осуществляется переход.

2.3.3.6. Принудительное прекращение программы

Обычно программа завершает свою работу по достижении последнего оператора (т. е. при выходе на оператор *end* с точкой). Если возникает необходимость прекратить выполнение программы где-либо внутри нее, то можно воспользоваться процедурой *halt*, которая вызывается как отдельный оператор. Эту процедуру можно вызвать, задав в круглых скобках параметр в виде целого неотрицательного числа от 0 до 255. Это значение возвращается в операционную систему в виде кода ошибки (*ERRORLEVEL*) и может быть проанализирована ОС в случае запуска данной программы из командного файла. Отсутствие параметра в процедуре *halt* соответствует значению параметра 0 (нормальное завершение программы).

Второй процедурой, с помощью которой можно прекратить выполнение программы, является процедура без параметров *exit* при ее размещении в исполнимой части программы (а не в теле подпрограммы). Чаще эта процедура применяется для выхода из подпрограммы без прекращения выполнения вызывающей программы.

Задание 3. Циклический вычислительный алгоритм

В соответствии с вариантом задания необходимо вычислить значение функции на заданном интервале с требуемой дискретностью. При этом одно из исходных данных принимается константой; шаг *h* вводится в диалоговом режиме; остальные исходные данные – переменные, которые задаются через оператор присваивания. По результатам вычислений необходимо построить график функции.

Цель работы: освоение навыков программирования циклических вычислительных процессов.

1. Для заданного варианта используется цикл с предусловием.
2. Для заданного варианта используется цикл с постусловием.

Таблица 2.3

Варианты к заданию 3

Номер варианта	Исходные данные	Функция
1	$0 \le x \le 0{,}35$ $\Delta x = 0{,}05$ $m = 2{,}5$ $a = 0{,}25$	$y = \dfrac{2\ln\left(ax^2 + \sqrt{a+m^2}\right)}{m\sqrt{m+a^2} + \cos(\pi x)}$
2	$0{,}1 \le x \le 0{,}5$ $\Delta x = 0{,}05$ $a = 0{,}2$ $n = 1{,}8$	$y = \dfrac{ax + \sin(\cos(\pi x) - n)}{a^6 x - \sqrt{nx^6 + a}}$
3	$0 \le x \le 0{,}4$ $\Delta x = 0{,}05$ $a = 0{,}3$ $k = 2$	$y = \dfrac{kx^2 + \sqrt{k^2 + x\ln\lvert\sin(\pi x) + a\rvert}}{k - \sqrt{\cos(\pi x + a)}}$
4	$4{,}25 \le x \le 4{,}75$ $\Delta x = 0{,}05$ $a = 0{,}35$	$y = 0{,}5\ln\left\lvert\sin\dfrac{x}{a}\right\rvert$
5	$0 \le x \le 0{,}4$ $\Delta x = 0{,}05$ $a = 0{,}2$ $n = 2{,}5$	$y = \dfrac{n - \sqrt{ax + \arcsin^2(\pi x)}}{an - (n + \cos(\pi x))}$
6	$0{,}35 \le x \le 0{,}5$ $\Delta x = 0{,}01$ $a = 0{,}55$ $n = 1{,}5$	$y = \dfrac{x}{a} + \dfrac{1}{a}\left(x + n^{e^{nx}}\right)$
7	$0{,}15 \le x \le 0{,}25$ $\Delta x = 0{,}01$ $a = 1{,}8$	$y = \dfrac{a^2\sin(x)}{x^4 + \sqrt{1 + 2x^2 + 3x^3}}$
8	$0{,}27 \le x \le 0{,}4$ $\Delta x = 0{,}01$ $a = 1{,}324$	$y = \dfrac{e^{x^3} + \operatorname{arctg} x}{a \cdot \sin\dfrac{x}{2}}$
9	$\Delta x = 0{,}05$ $a = 0{,}3$	$y = \begin{cases} 3\sin x^2 + ax, & \text{при } 0{,}5 \le x < 0{,}8 \\ \ln(0{,}7x^2 - a), & \text{при } 0{,}8 \le x \le 1 \end{cases}$

Окончание табл. 2.3

Номер варианта	Исходные данные	Функция
10	$\Delta x = 0{,}1$ $a = 0{,}28$	$y = \begin{cases} \sin(\pi x) - \sqrt{\lvert \ln(a^2 + x^2) \rvert}, & \text{при } 0 \le x < 0{,}5 \\ a \cdot \cos^2 \dfrac{\pi x}{2}, & \text{при } 0{,}5 < x \le 1 \end{cases}$
11	$\Delta x = 0{,}1$ $a = 0{,}28$	$y = \begin{cases} e^{\cos(\pi x)} - a \cdot \sin^2(\pi x), & \text{при } 0{,}5 \le x \le 1 \\ a \cdot x^2 - \cos(\pi x), & \text{при } 1 < x \le 1{,}5 \end{cases}$
12	$\Delta x = 0{,}1$ $a = 0{,}3$	$y = \begin{cases} \operatorname{tg} \dfrac{x}{4}, & \text{при } 0{,}1 \le x < 0{,}5 \\ e^{a \cdot \sin x}(\sin^2 x + a^2), & \text{при } 0{,}5 < x \le 1{,}0 \end{cases}$
13	$0 \le x < 0{,}15$ $\Delta x = 0{,}01$ $a = 0.25$	$y = \ln\left(a \cdot \sin^2(\pi x) + \sqrt{a^2 + x^2} \right)$
14	$0 \le x < 0{,}15$ $\Delta x = 0{,}01$ $a = 0{,}2$	$y = e^{a \cdot \sin \frac{\pi x}{a}} \sin(\pi x)$
15	$0{,}05 \le x < 0{,}25$ $\Delta x = 0{,}01$ $a = 0{,}3$	$y = \sqrt{x} \cdot \arcsin(\pi(a^2 + x^2))$
16	$0 \le x < 0{,}2$ $\Delta x = 0{,}01$ $a = 0{,}25$	$y = \sqrt{a^2 + x^2} \cdot \ln(a^2 + \sin(\pi x))$
17	$0 \le x < 0{,}15$ $\Delta x = 0{,}01$ $a = 0{,}2$ $b = 0{,}3$	$y = e^{b^2 + \sin(\pi x)} \cdot \cos \dfrac{\pi x}{a}$
18	$0 \le x \le 0{,}2$ $\Delta x = 0{,}01$ $Q = 0{,}25$ $a = 0{,}2$	$y = \lg\left(Q^2 + \sin^2 \dfrac{\pi x}{a} \right)$
19	$0{,}01 \le x < 0{,}2$ $\Delta x = 0{,}01$ $a = 0{,}1$	$y = \sqrt{x + 1} \cdot \arccos(\pi(a^2 + x^2))$
20	$0 \le x < 0{,}15$ $\Delta x = 0{,}01$ $a = 0.25$	$y = \ln\left(a \cdot \cos^2(\pi x) + \sqrt{a^3 + x^3} \right)$

Вопросы для самопроверки

1. Что такое циклический вычислительный процесс?
2. Что может использоваться в качестве переменной цикла?
3. Структуры итерационных циклов. Каковы их отличия?
4. Нарисуйте блок-схему алгоритма циклического процесса с предусловием.
5. Нарисуйте блок-схему алгоритма циклического процесса с постусловием.
6. Операторы цикла в языке Паскаль.

2.3.3.7. Цикл со счетчиком (*for*)

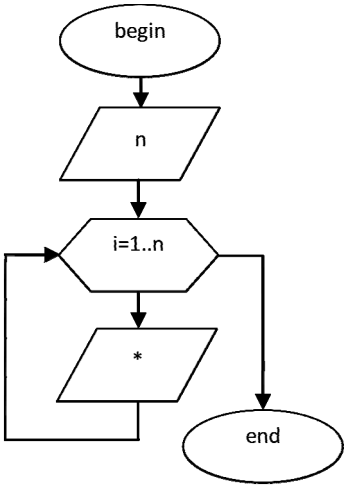


Рис. 2.5. Цикл со счетчиком

Часто цикл *for* называют циклом со счетчиком (рис. 2.5). Этот цикл используется, когда число повторений не связано с тем, что происходит в теле цикла, т. е. количество повторений может быть известно заранее.

В заголовке цикла указываются два значения. Первое значение присваивается так называемой переменной-счетчику, от этого значения начинается отсчет количества итераций (повторений). Отсчет идет всегда с шагом равным единице. Второе значение указывает, при каком значении счетчика цикл должен остановиться. Другими словами, количество итераций цикла определяется разностью между вторым и первым значением плюс единица.

В языке *Pascal* тело цикла не должно содержать выражений, изменяющих счетчик. Цикл *for* существует в двух формах:

```
for счетчик:=значение to конечное_значение do
    тело_цикла;

for счетчик:=значение downto конечное_значение do
    тело_цикла;
```

Счетчик – это переменная любого из перечисляемых типов (целого, булевого, символьного, диапазонного, перечисления). Начальные и конечные значения могут быть представлены не только значениями, но и выражениями, возвращающими совместимые с типом счетчика типы данных. Если между начальным и конечным выражением 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.

Количество итераций цикла *for* известно именно до его выполнения, но не до выполнения всей программы. Так, в примере, представленном ниже, количество выполнений цикла определяется пользователем. Значение присваивается переменной, а затем используется в заголовке цикла.

```
Program Cycle3;          {задаем имя программы}
var                      {объявляем переменные}
   i, n: integer;        {задаем целочисленный тип переменным}
begin                    {начало раздела операторов}
   write('Количество знаков = ');   {организуем диалоговый ввод}
   readln(n);            {считывается заданное кол-во итераций}
   for i := 1 to n do    {задаем счетчику начальное, конечное значение}
    write('*');          {выводим символ на экран}
end.                     {конец раздела операторов и программы}
```

2.4. Алгоритмизация обработки одномерных массивов

Наряду с простыми переменными в алгоритмических языках используются и другие переменные – массивы. Массив является самой распространенной структурой, реализованной практически во всех языках программирования. Каждый массив определяется его именем и размерностью.

Элементами массива являются индексированные переменные. Индексированная переменная обозначается именем массива и одним или несколькими индексами, разделенными запятой. Для одномерного массива индекс определяет порядковый номер элемента. Например, пятый элемент одномерного массива $A - A_5$.

Массив – однородная совокупность элементов. Массивы состоят из ограниченного числа компонент, причем все компоненты массива имеют один и тот же тип, называемый базовым. Структура массива всегда однородна. Массив может состоять из элементов типа *integer*, *real* или *char* либо других однотипных элементов. Из этого, правда, не следует делать вывод, что компоненты массива могут иметь только скалярный тип.

Другая особенность массива состоит в том, что к любой его компоненте можно обращаться произвольным образом, т. е. программа может сразу получить нужный ей элемент по его порядковому номеру (индексу).

Индекс – это значение порядкового типа, определенного как тип индекса данного массива. Очень часто это целочисленный тип (*integer*, *word* или *byte*), но может быть и логический, и символьный.

2.4.1. Описание массива в языке Паскаль

Перед первым использованием массива или индексированной переменной они должны быть описаны. Описание сводится к объявлению имени массива с указанием в скобках максимальных значений индексов.

В языке Паскаль тип массива задается с использованием специального слова *array* (англ. – массив), и его объявление в программе выглядит следующим образом:

type <имя_типа> = array [I] of T;

где *I* – тип индекса массива; *T* – тип его элементов.

Можно описывать сразу переменные типа массив, т.е. в разделе описания переменных:

var a, b: array [I] of T;

Обычно тип индекса характеризуется некоторым диапазоном значений любого порядкового типа: *I1 .. In*. Например, индексы могут изменяться в диапазоне 1..20 или '*a*'..'*n*'.

При этом длину массива характеризует выражение: *ord(In) - ord(I1) + 1*.

Вот, например, объявление двух типов: *vector* в виде массива на языке Паскаль из десяти целых чисел и *stroka* в виде массива из 256 символов.

```
type
  vector = array [1..10] of integer;
  stroka = array [0..255] of char;
```

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.

Опишем переменные типа *vector* и *stroka*:

```
Var a: vector; c: stroka;
```

далее в программе мы можем обращаться к отдельным элементам массива *a* или *c*. Например, `a[5] := 23; c[1] := 'w'; a[7] := a[5] * 2; writeln(c[1], c[3])`.

Индекс массива в языке Паскаль 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, т. е. будет произведена попытка обратиться к элементу, которого не существует. Эта типичная ошибка называется «выход за пределы массива».

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

2.4.2. Основные действия с массивами

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 две переменные одного типа, например,

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

то можно переменной *a* присвоить значение переменной *b* (*a := b*). При этом каждому элементу массива *a* будет присвоено соответствующее значение из массива *b*. Все остальные действия над массивами в языке Паскаль производятся поэлементно!

2.4.3. Ввод массива

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 цикл со счетчиком, где параметром цикла будет выступать переменная – индекс массива. Значения элементов могут быть введены с клавиатуры или определены с помощью оператора присваивания.

Пример фрагмента программы ввода массива на языке Паскаль

```
Var
  A: array [1..10] of integer;   {описываем массив A из 10 целых чисел}
  i: byte;                       {переменная i будет индекс массива}
Begin
  For i:=1 to 10 do      {перебираем элементы массива, изменяя индекс}
  begin
    Write (`введите a(`,i,`)=`); {реализуем диалоговый ввод значений}
    Readln(a[i]);      {ввод i- го элемента производится с клавиатуры}
  end;
```

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

Пример фрагмента программы заполнения массива десятью случайными целыми числами в диапазоне от 0 до 5:

```
Var
  A: array[1..10] of integer;
  i: byte;
```

```
Begin
   For i := 1 to 10 do
      A[i] := random(5);
```

2.4.4. Вывод массива

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

Ниже приведен пример фрагмента программы на языке Паскаль, осуществляющий вывод элементов массива в одну строку, после каждого элемента ставится пробел для отделения элементов массива друг от друга.

```
Var
   A: array [1..10] of integer;
   i: byte;
Begin
   For i := 1 to 10 do
      Write (a[i],' ');
```

Вывод элементов массива 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.

Пример программы вывода массива в столбик

```
Var
   A: array [1..10] of integer;
   i: byte;
Begin
   For i := 1 to 10 do
      Writeln ('a[', i, ']=', a[i]);
```

Задание 4. Формирование векторов

Необходимо составить алгоритм формирования вектора $\{P_k\}_{k=1}^{n}$, количество элементов которого не превышает десяти, а элементы вектора вычисляются по заданному закону в соответствии с вариантом из

базовых векторов. Реализовать ввод исходных данных $\{B_k\}_{k=1}^{n}$ и $\{C_k\}_{k=1}^{n}$ в диалоговом режиме. Результирующий вектор необходимо вывести в одну строку.

Цель работы: освоение навыков программирования циклических вычислительных процессов со счетчиком.

Таблица 2.4

Варианты к заданию 4

Номер варианта		Номер варианта	
1	$P_k = \begin{cases} 5B_k + 2C_k + 7, B_k > 0 \\ B_k,\ B_k \le 0 \end{cases}$	14	$P_k = \begin{cases} B_k - C_k,\ B_k \ge 0 \\ B_k,\ B_k < 0 \end{cases}$
2	$P_k = \begin{cases} C_k / \lvert B_k \rvert,\ \ \lvert B_k \rvert > 1 \\ B_k,\ \lvert B_k \rvert \le 1 \end{cases}$	15	$P_k = \begin{cases} B_k + 0.5 \cdot C_k,\ B_k < 0 \\ B_k,\ B_k \ge 0 \end{cases}$
3	$P_k = \begin{cases} 1 - \sin B_k, B_k > 0 \\ 1 + \cos C_k,\ B_k \le 0 \end{cases}$	16	$P_k = \begin{cases} B_{k-2} + B_{k-1},\ k > 2 \\ C_k,\ B_k \le 2 \end{cases}$
4	$P_k = \begin{cases} B_k + C_k + 1,\ k < 2 \\ B_{k-1},\ k > 2 \end{cases}$	17	$P_k = \begin{cases} B_k + C_k,\ B_k > C_k \\ C_k,\ B_k \le C_k \end{cases}$
5	$P_k = \max\left(B_k, C_k\right)$	18	$P_k = \max\left(\frac{1}{B_k}, \frac{1}{C_k}\right)$
6	$P_k = \begin{cases} B_k^2 + k,\ k > \sqrt{n} \\ C_k^2 + k,\ k \le \sqrt{n} \end{cases}$	19	$P_k = \begin{cases} B_k,\ B_k > 1 \\ 0,\ B_k \le 1 \end{cases}$
7	$P_k = \begin{cases} B_k,\ B_k > 10 \\ C_k,\ B_k \le 10 \end{cases}$	20	$P_k = \begin{cases} C_k, C_k \le 0 \\ -B_k,\ C_k > 0 \end{cases}$
8	$P_k = \begin{cases} \lvert C_k \rvert,\ B_k > 0 \\ B_k,\ B_k \le 0 \end{cases}$	21	$P_k = \begin{cases} B_k, C_k + B_k > 0 \\ -C_k,\ C_k + B_k \le 0 \end{cases}$

Окончание табл. 2.4

Номер варианта		Номер варианта	
9	$P_k = \begin{cases} 1, & B_k = C_k = 0 \\ \sqrt{B_k^2 + C_k^2}, & B_k \neq 0,\ C_k \neq \end{cases}$	22	$P_k = \min\left(B_k, B_k - C_k\right)$
10	$P_1 = B_n - C_n,\ P_2 = B_{n-1} - C_{n-1}$ …, $P_n = B_1 - C_1$	23	$P_k = \begin{cases} C_k, & C_k > 5 \\ C_k - 1, & C_k \leq 5 \end{cases}$
11	$P_k = \begin{cases} B_k + C_k, & k - \text{нечетные} \\ 0, & k - \text{четные} \end{cases}$	24	$P_k = \begin{cases} C_k + B_k, & C_k + B_k - \text{целое} \\ INT(C_k + B_k), & \\ C_k + B_k - \text{дробное} & \end{cases}$
12	$P_k = \begin{cases} B_k + 1, & \sin B_k > 0 \\ C_k - 1, & \sin B_k \leq 0 \end{cases}$	25	$P_k = \begin{cases} ln\left(C_k \cdot B_k^2\right), & C_k > 0 \\ 0, & C_k \leq 0 \end{cases}$
13	$P_k = \begin{cases} 2B_k + 3C_k, & \cos B_k > 0 \\ (B_k + C_k)/2, & \cos B_k \leq 0 \end{cases}$	26	$P_k = \begin{cases} B_k, & B_k + C_k \text{ кратно } 2 \\ C_k, & B_k + C_k \leq 0 \end{cases}$

Вопросы для самопроверки

1. Расскажите о логических функциях и их реализации на ЯВУ.
2. Как проверить полученное значение выражения на четность, дробность, кратность заданному значению?
3. Что такое циклический вычислительный процесс со счетчиком?
4. Что может использоваться в качестве переменной цикла?
5. Структуры итерационных циклов. Каковы их отличия?
6. Нарисуйте блок-схему алгоритма циклического процесса со счетчиком.
7. Оператор цикла со счетчиком в языке Паскаль.

3. АЛГОРИТМИЗАЦИЯ СЛОЖНЫХ ВЫЧИСЛИТЕЛЬНЫХ ПРОЦЕССОВ

3.1. Алгоритмизация обработки двумерных массивов

3.1.1. Двумерные массивы

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

Каждый элемент имеет свой номер, как у одномерных массивов, но этот номер состоит из двух чисел – номера строки, в которой находится элемент, и номера столбца. Таким образом, номер элемента определяется пересечением строки и столбца. Например, a_{21} – это элемент, стоящий во второй строке и в первом столбце.

3.1.2. Описание двумерного массива

Существует несколько способов объявления двумерного массива.

Можно описывать одномерные массивы, элементы которых могут быть массивами. Пример подобного описания двумерного массива:

```
type
  vector = array[1..5] of <тип_элементов>;
  matrix = array[1..8] of vector;
var m: matrix;
```

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

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

```
type
  matrix = array[1..8, 1..5] 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.

Главное не перепутать строки со столбцами, а то можно получить обращение к несуществующему элементу. Например, обращение к элементу *m[5, 8]* имеет правильную форму записи, но может вызвать ошибку в работе программы.

3.1.3. Основные действия с двумерными массивами

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. Например, если в программе описаны две матрицы одного типа, 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.

3.1.4. Ввод двумерного массива

Положение элемента в двумерном массиве определяется двумя индексами: номером строки и номером столбца. Это значит, что нам нужно будет последовательно изменять номер строки с первой до последней и в каждой строке перебирать элементы столбцов с первого до последнего. Значит, нам потребуется два цикла со счетчиком, причем один из них будет вложен в другой.

Фрагмент программы ввода двумерного массива с клавиатуры.

```
type
   matrix = array[1..8,1..5] of integer;
var
   a: matrix;
   i, j: integer;              {описание индексов массива}
begin
   for i := 1 to 8 do          { перебор всех номеров строк}
      for j := 1 to 5 do       {перебор всех номеров столбцов в строке}
         begin
```

```
    Write (`введите a(`, i, `,` , j, `)=`);
    readln(a[i, j]);
  end;
```

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

3.1.5. Вывод двумерного массива на экран

Вывод элементов двумерного массива 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

```
for i := 1 to 8 do          {перебор номеров всех строк}
  begin
    for j := 1 to 5 do      {перебор номеров всех столбцов в строке}
      write(a[i,j], ` `);   {печать эл-тов i -й строки в одну строку}
    writeln();              {перевести курсор на новую строку}
  end;
```

Примечание: 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 *a* – это переменная типа массив. Заполнение и вывод на экран элементов массива должно осуществляться последовательно и поэлементно, так как в памяти ЭВМ элементы массива располагаются в последовательных ячейках.

3.1.6. Представление двумерного массива в памяти

Элементы абстрактного массива в памяти машины физически располагаются последовательно согласно описанию. При этом каждый элемент занимает в памяти количество байт, соответствующее его размеру. Например, если массив состоит из элементов типа *integer*, то

каждый элемент будет занимать по два байта. А весь массив займет $2 \cdot n$ байт, где n – количество элементов в массиве.

Сколько места займет массив, состоящий из массивов, т.е. матрица? Очевидно: $2 \cdot n \cdot m$, где n – количество строк, а m – количество элементов в каждой строке. Например, для массива типа: *matrix = = array[1..8,1..5] of integer;* потребуется 80 байт памяти.

Размещение в памяти осуществляется «снизу вверх». Элементы размещаются в порядке изменения индекса, что соответствует схеме вложенных циклов: сначала размещается первая строка, затем вторая, третья и т. д. Внутри строки по порядку идут элементы: первый, второй и т. д. Конкретная память выделяется для переменной при загрузке программы, т. е. устанавливается взаимное соответствие между переменной и адресом ячейки. Если мы объявили переменную как массив, то программа «знает» адрес начала массива, т. е. первого его элемента.

Каков максимально допустимый размер массива, учитывая ограниченный объем памяти. Для работы программы память выделяется сегментами по 64 Кбайт каждый, причем как минимум один из них определяется как сегмент данных. Вот в этом-то сегменте и располагаются те данные, которые будет обрабатывать программа. Ни одна переменная программы не может располагаться более чем в одном сегменте. Поэтому, даже если в сегменте находится только одна переменная, описанная как массив, то она не сможет получить более чем 65 536 байт. Но почти наверняка, кроме массива в сегменте данных, будут описаны еще некоторые переменные, поэтому реальный объем памяти, который может быть выделен под массив, находится по формуле: $65536 - V$, где V – объем памяти, уже выделенный под другие переменные.

Если в программе встретится правильное с точки зрения синтаксиса описание типа:

type MyArray = array[1..60000] of integer;

то при компиляции транслятор выдаст сообщение об ошибке объявления слишком длинного массива.

Учитывая двухбайтовое представление целых чисел, реально можно объявить массив с количеством элементов, равным 65536/2 – 1 = 32767. И то лишь в том случае, если других переменных не будет. Двумерные массивы должны иметь еще меньшие границы индексов.

Задание 5. Циклический вычислительный алгоритм со счетчиком

В соответствии с вариантом необходимо выполнить задание по обработке заранее заданного двумерного массива. Элементы матрицы задаются в программе через оператор присваивания.

Цель работы: освоение навыков программирования циклических вычислительных процессов с использованием счетчика для обработки массивов данных.

Дана матрица:

$$B = \begin{Bmatrix} 5.32 & 8 & -4 & 0 \\ -3 & 2 & 0.75 & -6 \\ 1.25 & 0 & 0 & 1.4 \end{Bmatrix}$$

Варианты заданий

1. Найти сумму элементов по каждой строке матрицы.
2. Заменить элементы нечетных столбцов четными числами по порядку.
3. Найти сумму элементов по каждому столбцу матрицы.
4. Найти максимальные элементы в столбцах матрицы.
5. Найти число элементов, значение которых по модулю больше двух.
6. Найти номера строк, где находятся минимальные элементы в столбцах матрицы.
7. В матрице отрицательные элементы заменить на нули.
8. В матрице определить номер строки и столбца ее максимального элемента.
9. Найти средние арифметические значения отрицательных элементов каждого столбца матрицы.
10. Найти максимальные элементы в строках матрицы.
11. В матрице определить номер строки и столбца её минимального элемента.
12. Найти номера столбцов, где находятся максимальные элементы в строках матрицы.
13. Найти средние арифметические значения положительных элементов каждой строки матрицы.

14. Найти сумму элементов матрицы.

15. Вычислить норму матрицы (квадратный корень из суммы квадратов элементов матрицы).

16. Поменять местами элементы первого столбца с третьим и второго с четвертым.

17. Заменить элементы четных строк на нечетные числа по порядку.

18. Найти максимальный по модулю элемент матрицы.

19. Найти число элементов, значение которых по модулю меньше пяти.

20. Найти максимальный элемент матрицы.

Вопросы для самопроверки

1. Как организуется детерминированный циклический вычислительный процесс?

2. Состав блока модификации и его программная реализация на языке Паскаль.

3. Пунктуация в языке Паскаль.

4. Как производится описание многомерного массива?

5. Что выступает в качестве переменной цикла при обработке элементов массива?

В следующем задании задачи решаются в общем виде. Такой подход позволяет лучше понять принципы построения программ и особенности использования операторов изучаемого алгоритмического языка.

Задание 6. Преобразования матриц и векторов

Необходимо составить алгоритм решения задачи с использованием оператора цикла со счетчиком. Исходные данные задаются преподавателем после составления программы решения поставленной задачи в общем виде.

Цель работы: освоение навыков программирования сложных циклических вычислительных процессов при преобразовании матриц и векторов.

Имеются некоторые векторы $\{B_k\}_{k=1}^{n}$, $\{C_k\}_{k=1}^{n}$ и матрица $\{A_{i,k}\}_{i=1,\ k=1}^{n\qquad n}$.

Варианты заданий

1. Найти среднее арифметическое элементов вектора $\boldsymbol{B}$, предшествующих первому отрицательному элементу.

2. Найти среднее геометрическое всех положительных элементов вектора $\boldsymbol{B}$.

3. Найти среднее арифметическое тех элементов вектора $\boldsymbol{C}$, модуль которых меньше заданного числа y, но больше числа x, где x, y – заданные числа.

4. Для всех элементов вектора $\boldsymbol{B}$, удовлетворяющих условию $x < B_k < y$, найти сумму выражений S_k, где p, h, x, y – заданные числа:

$$S_k = \frac{pB_k - hB_{k-1}}{pB_{k+1} - hB_k}.$$

5. Найти среднее арифметическое всех четных по номеру элементов вектора $\boldsymbol{B}$.

6. Подсчитать число элементов вектора $\boldsymbol{C}$, значение которых больше x, где x — заданное число.

7. Найти среднее геометрическое от абсолютных величин элементов вектора $\boldsymbol{B}$, удовлетворяющих условию $|B_K| > x$, где x – заданное число.

8. Найти среднее арифметическое от всех нечетных по номеру элементов вектора $\boldsymbol{C}$.

9. Найти произведение всех элементов вектора $\boldsymbol{B}$, удовлетворяющих условию $B_k < P$, где P – заданное число.

10. Найти произведение элементов вектора $\boldsymbol{C}$, предшествующих первому положительному элементу.

11. Найти среднее арифметическое всех целых элементов вектора B.

12. Найти длину вектора $\boldsymbol{B}$.

13. Найти среднее арифметическое всех дробных элементов вектора C.

14. Для всех элементов вектора $\boldsymbol{B}$, удовлетворяющих условию $B_k < P$, найти сумму выражений S_k:

$$S_k = \frac{(B_k - P)^3}{B_k}.$$

15. Найти среднее арифметическое всех четных по величине элементов вектора $\boldsymbol{B}$.

16. Найти скалярное произведение векторов $\boldsymbol{B}$ и $\boldsymbol{C}$, определяемое следующим образом:

$$P = \sum_{k=1}^{n} B_k \cdot C_k .$$

17. Найти среднее арифметическое всех нечетных по величине элементов вектора $\boldsymbol{C}$.

18. Найти сумму элементов пятого столбца матрицы $\boldsymbol{A}$, удовлетворяющих условию $A_{i,k} > E$, где E – заданное число.

19. Найти среднее арифметическое всех положительных элементов вектора $\boldsymbol{C}$.

20. Найти среднее арифметическое всех отрицательных элементов главной диагонали матрицы $\boldsymbol{A}$.

21. Рассматривая элементы вектора $\boldsymbol{B}$ как координаты одной точки n-мерного пространства, а элементы вектора $\boldsymbol{C}$ – как координаты другой точки, найти расстояние P между ними.

22. Найти число отрицательных элементов вектора $\boldsymbol{C}$.

23. Найти число элементов вектора $\boldsymbol{B}$, которые меньше P, где P – заданное число.

24. Вычислить обратную величину суммы обратных величин всех ненулевых элементов вектора $\boldsymbol{B}$.

25. Для всех положительных элементов вектора $\boldsymbol{B}$ найти произведение выражений P_k:

$$P_k = 1 + \frac{B_k^2}{\pi^2}.$$

26. Найти среднее арифметическое всех отрицательных элементов вектора $\boldsymbol{B}$.

Вопросы для самопроверки

1. Какова структура алгоритма для обработки массивов?
2. Операторы ввода-вывода в языке Паскаль.
3. Расскажите о типах данных в языке Паскаль.
4. Арифметические выражения в языке Паскаль.

3.2. Алгоритмы обработки бесконечных числовых рядов

Числовой ряд – это числовая последовательность, рассматриваемая вместе с другой последовательностью, которая называется последовательностью частичных сумм (ряда).

Ряд может состоять из членов ряда, чередующихся по знакам через один, такой ряд называется знакопеременным (знакочередующимся) и имеет вид

$$S = a_1 - a_2 + a_3 - \ldots + (-1)^{i+1} \cdot a_i + \ldots = \sum_{i=1}^{\infty} (-1)^{i+1} \cdot a_i .$$

Ряд может состоять из членов ряда одного знака, такой ряд называется знакопостоянным и имеет вид:

$$S = a_1 + a_2 + a_3 + \ldots + a_i + \ldots = \sum_{i=1}^{\infty} a_i .$$

Реккурентная формула выражает каждый член последовательности через предыдущие члены и возможно номер члена последовательности i. Примерами реккурентных формул являются функция факториала натурального числа ($a_0=0!=1$, $a_i=i!=(i-1)!\cdot i$), вычисления чисел Фибоначчи ($a_0=0$, $a_1=1$, $a_i=a_{i-1}+a_{i-2}$) и т. д.

При нахождении суммы убывающего числового ряда вычислительными методами встает вопрос о точности E вычисления суммы S. Для нахождения суммы числового ряда мы будем учитывать только n первых членов ряда и пренебрежем влиянием на итоговый результат членов ряда, начиная с $n-1$, поскольку их общий «вес» в результирующей сумме незначителен.

Для организации цикла с неизвестным числом повторений необходимо использовать операторы цикла с постусловием или с предусловием. Если первый член ряда известен, то можно использовать цикл с предусловием, в противном случае необходимо применять цикл с постусловием.

Задание 7. Обработка бесконечных числовых рядов

Необходимо найти сумму S числового ряда включая члены, превышающие заданное число E. При составлении алгоритма нахождения суммы необходимо воспользоваться рекуррентной формулой. При ее

отсутствии используется член ряда, записанный в общем виде. Точность вычислений задается в диалоговом режиме.

Цель работы: освоение навыков программирования циклических вычислительных процессов при вычислении суммы бесконечных числовых рядов.

Таблица 3.1

Варианты к заданию 7

Номер варианта	Числовой ряд	Рекуррентная формула
1	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{1}{(i-1)!}$	$a_{i+1}=-\frac{a_i}{i},\ a_1=1$
2	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{1}{i}$	
3	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{1}{2^{i-1}}$	$a_{i+1}=\frac{a_i}{2},\ a_1=1$
4	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{1}{2i-1}$	
5	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{1}{i^2}$	
6	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{1}{i^4}$	
7	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{2^{i-1}}{(i-1)!}$	$a_{i+1}=-\frac{2a_i}{i},\ a_1=1$
8	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{3^{i-1}}{(i-1)!}$	$a_{i+1}=-\frac{3a_i}{i},\ a_1=1$
9	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{5^{i-1}}{(i-1)!}$	$a_{i+1}=-\frac{5a_i}{i},\ a_1=1$
10	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{1}{2^{i-1}(i-1)!}$	$a_{i+1}=-\frac{a_i}{2i},\ a_1=1$

Продолжение табл. 3.1

Номер варианта	Числовой ряд	Рекуррентная формула
11	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{1}{(2i-1)!}$	$a_{i+1}=-\frac{a_i}{2i(2i+1)},\ a_1=1$
12	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{2^{2i-1}}{(2i-1)!}$	$a_{i+1}=-\frac{a_i}{2i(2i+1)},\ a_1=2$
13	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{1}{3^{2i-1}(2i-1)!}$	$a_{i+1}=-\frac{a_i}{3^2 2i(2i+1)},\ a_1=\frac{1}{3}$
14	$\sum_{i=1}^{\infty}(-1)^{i+1}\frac{1}{i!(i+1)(i+2)}$	$a_{i+1}=-\frac{a_i}{(i+3)},\ a_1=\frac{1}{6}$
15	$\sum_{i=1}^{\infty}\frac{1}{(i-1)!}$	$a_{i+1}=\frac{a_i}{i},\ a_1=1$
16	$\sum_{i=1}^{\infty}\frac{1}{2^{i-1}}$	$a_{i+1}=\frac{a_i}{2},\ a_1=1$
17	$\sum_{i=1}^{\infty}\frac{1}{i(i+1)}$	
18	$\sum_{i=1}^{\infty}\frac{1}{(2i-1)(2i+1)}$	
19	$\sum_{i=1}^{\infty}\frac{1}{2i(2i+2)}$	
20	$\sum_{i=1}^{\infty}\frac{1}{i(i+1)(i+2)}$	
21	$\sum_{i=1}^{\infty}\frac{1}{(2i-1)^2}$	
22	$\sum_{i=1}^{\infty}\frac{1}{i^4}$	

Окончание табл. 3.1

Номер варианта	Числовой ряд	Рекуррентная формула
23	$\sum_{i=1}^{\infty} \frac{1}{(2i-1)^4}$	
24	$\sum_{i=1}^{\infty} \frac{1}{i^2}$	
25	$\sum_{i=1}^{\infty} \frac{1}{(4i-1)(4i+1)}$	

Вопросы для самопроверки

1. Расскажите, что вы знаете об этапах цикла в языке Паскаль?
2. Что вы знаете о вложенных условных операторах в языке Паскаль?

3.3. Алгоритмизация исследования областей двумерного пространства

Для исследования попадания заданной точки в определенную область двумерного (или *n*-мерного) пространства вначале необходимо аналитически описать границы заданной области.

После этого необходимо сформулировать условия попадания точки в область определяемую данными границами. Например, если точка находится ниже границы, описываемой некоей прямой линией, то значит ее ордината меньше ординаты границы; если точка лежит внутри окружности, то расстояние до центра окружности меньше чем радиус, и т. д.

Ряд условий связываются между собой логическими функциями: конъюнкцией, дизъюнкцией и др. Проверка итогового условия однозначно определяет принадлежность исследуемой точки заданной области.

Задание 8. Исследование областей двумерного пространства

На плоскости заданы N точек с координатами (X_i, Y_i) $(i=1,...,N)$ и множество ***D***. Требуется найти число точек ***K***, лежащих внутри множества ***D***.

Цель работы: освоение навыков программирования вычислительных процессов при исследовании областей двумерного пространства.

Для облегчения составления алгоритма и дальнейшего контроля правильности решения необходимо начать с графического изображения заданного множества на плоскости и нанести исследуемые точки.

Таблица 3.2

Координаты точек для вариантов № 1–15 задания 8

X	–0,77	–0,81	–0,2	1,5	–3,91	1,5
Y	1,44	–2,4	0,3	3	–1,81	0,5

Таблица 3.3

Координаты точек для вариантов № 16–25 задания 8

X	0,9	0,7	1,2	1,4	2,5	3,1	–1
Y	2,2	1,65	–0,96	–3,7	3,06	1,5	2,3

Варианты заданий описания множества ***D***

1. Круг радиусом $R = 7$ с центром в точке A(–0.3; –3.4) делится прямыми $Y = X$, $Y = –X$ на четыре части; D – «верхняя» часть круга.
2. Круг радиусом $R = 2$ с центром в точке A(–0.3; –3.4) делится прямыми $Y = X – 2$, $Y = –X – 2$ на четыре части. D – «правая» часть круга.
3. D – круг радиусом $R = 2$ с центром в точке A(2,2).
4. D – внутренность квадрата со стороной $H = 3$ и с центром в точке A(1,1) со сторонами, параллельными координатными осями.
5. Множество D заключено между сторонами двух квадратов с общим центром в точке A(2,2), стороны которых $H_1 = 2$ и $H_2 = 4$ параллельны осям координат.
6. D – общая часть кругов, радиусы которых $R_1 = 2$ и $R_2 = 1.5$, и центры расположены соответственно в точках A(0,1) и B(–2,1).

7. D – часть круга радиусом $R_1 = 2$ с центром в точке A(0,1), лежащая вне круга радиусом $R_2 = 1.5$ с центром в точке B(–2,1).

8. D – объединения кругов радиусами $R_1 = 2$ и $R_2 = 1.5$ с центрами, соответственно, в точках A(0,1) и B(–2,1).

9. D – внешность круга радиусом $R = 2$ с центром в точке A(2,2).

10. D – внешность квадрата с центром в точке A(2,2) стороны которого $H = 2$ параллельны координатным осям.

11. Круг радиусом $R = 1{,}5$ с центром в точке A(2,2) пересекается прямой $X = 1.7$. D – часть круга, лежащая справа от секущей.

12. Круг радиусом $R = 1.5$ с центром в точке A(2,2) пересекается прямой $Y = 1.6$. D – часть круга, лежащая над секущей.

13. Круг радиусом $R = 1{,}5$ с центром в точке A(2,2) пересекается прямыми $Y = 1.2$ и $Y = 2.15$. D – часть круга, лежащая между секущей.

14. Круг радиусом $R = 1.5$ с центром в точке A(2,2) пересекается прямыми $X = 3$ и $X = 0.8$. D – часть круга, лежащая между секущими.

15. Круг радиусом $R = 1.5$ с центром в точке A(2,2) пересекается прямыми $Y = 1.2$ и $X = 1$. D – правая верхняя часть круга.

16. Круг радиусом $R = 1.5$ с центром в точке A(2,2) пересекается прямыми $Y = 2.3$ и $X = 1.9$. D – левая верхняя часть круга.

17. Круг радиусом $R = 1.5$ с центром в точке A(2,2) пересекается прямыми $Y = 3$ и $X = 1$. D – правая нижняя часть круга.

18. Круг радиусом $R = 1.5$ с центром в точке A(2,2) пересекается прямыми $Y = 2.5$ и $X = 2.7$. D – левая нижняя часть круга.

19. D – окружность радиусом $R = 2$ с центром в точке A(2,2)

20. D – граница квадрата со стороной $H = 2$ и с центром в точке A(2,2). Стороны квадрата параллельны осям координат.

21. D – кольцо, заключенное между окружностями радиусов $R_1 = 1.5$ и $R_2 = 2$, имеющими общий центр в точке A(2,2).

22. D – часть круга радиусом $R = 1.5$ с центром в точке A(2,2), лежащая под прямой $Y = X$.

23. D – пересечение круга радиусом $R = 2$ с центром в точке A(0,1) и квадрата со стороной $H = 1{,}5$ и центром в точке B(–2,1). Стороны квадрата параллельны осям координат.

24. D – часть круга радиусом $R = 2$ с центром в точке A(0,1), лежащая вне квадрата со стороной $H = 1{,}5$ и центром в точке B(–2,1). Стороны квадрата параллельны осям координат.

25. D – объединение круга радиусом $R = 2$ с центром в точке A(0,1) и квадрата со стороной $H = 1{,}5$ и центром в точке B(–2,1). Стороны квадрата параллельны осям координат.

Вопросы для самопроверки

1. Расскажите о логических выражениях в языке Паскаль.
2. Как реализуется описание заданной области в двумерном пространстве?

3.4. Алгоритмизация с использованием подпрограмм

Подпрограмма – это фрагмент кода, который имеет свое имя. В случае необходимости выполнять этот код несколько раз создается подпрограмма, которая описывается один раз перед началом основной программы. Компилятор пропускает данный фрагмент кода, пока в основной программе не встретит «вызов» подпрограммы, который выглядит как обращение к ней по имени (или имени с аргументами, указанными в скобках).

Структура процедуры повторяет структуру программы. Каждая процедура должна иметь одну точку входа и одну точку выхода (т. е. использование *goto*, *break*, *exit* нежелательно), использование глобальных переменных в процедуре следует свести к минимуму, взаимодействие с процедурой должно осуществляться (по возможности) только через параметры.

```
var ...;                          {объявление глобальных переменных}
procedure имя(параметры);         {начало процедуры}
  var ...;                        {объявление локальных переменных}
  begin
  ...                             {тело процедуры}
  end;                            {конец процедуры}
begin                             {начало раздела операторов}
...                               {основная программа}
end.                              {конец программы}
```

Параметры процедуры (аргументы) указываются в скобках после ее имени в объявлении. В любой программе все переменные делятся на два типа: локальные и глобальные. Глобальные переменные – это переменные из раздела описаний основной части программы, а локальные – из раздела описаний процедур и функций. Локальные переменные существуют только в течение времени работы процедуры, определяются (создаются) при ее вызове и исчезают после завершении работы процедуры.

При описании процедуры указывается список формальных параметров. Каждый параметр является локальным по отношению к описываемой процедуре, к нему можно обращаться только в пределах данной процедуры. Формальный параметр процедуры указывается в скобках при ее описании. Обязательно необходимо указать тип формального параметра через двоеточие.

Фактические параметры – это параметры, которые передаются процедуре при обращении к ней. Фактический параметр – это значение, которое указывается в скобках при вызове процедуры. Фактическим параметром может быть конкретное значение (число, символ, строка) либо переменная, которые компилятор подставит вместо формального параметра. Поэтому тип данных у формального и фактического параметра процедуры должен быть одинаковым.

Число фактических параметров должно быть равно числу формальных параметров; соответствующие фактические и формальные параметры должны совпадать по порядку следования и по типу данных.

Если в качестве формального параметра указана обычная переменная с указанием ее типа, то такой параметр есть параметр-значение, или входной параметр. Тип данных формального параметра-значения должен соответствовать типу данных его фактического параметра. Копия фактического параметра становится значением соответствующего формального параметра. Внутри процедуры можно производить любые действия с данным формальным параметром (допустимые для его типа), но эти изменения никак не отражаются на значении фактического параметра, каким он был до вызова процедуры, то таким же и останется после завершения ее работы.

Если перед именем формального параметра в объявлении процедуры стоит служебное слово *var*, то такой параметр называется параметром-переменной, или выходным параметром. Для него используются те ячейки памяти, которые отведены под соответствующий параметр при вызове процедуры. Любые операции с формальным параметром выполняются непосредственно над фактическим параметром. Фактический параметр, соответствующий параметру-переменной, может быть только переменной (не константой и не выражением).

Пример использования подпрограммы для сложения двух чисел.

```
var a, b, c: integer;
procedure sum(x, y: integer; var z: integer);
  begin
```

```
    z := x + y;
  end;
begin
  write('Введите два числа='); readln(a, b);
  sum(a, b, c);
  writeln(`сумма=`, c);
end.
```

Функция выглядит почти так же, как и процедура. Отличие в том, что заголовок функции начинается с ключевого слова *function* и кончается типом возвращаемого данной функцией значения:

function <имя функции>(<список формальных операторов>): <тип возвращаемого значения>;

Кроме того, в теле функции обязательно должен быть хотя бы один оператор присваивания, где в левой части стоит имя функции или переменная *result*, а в правой – ее значение.

Пример использования функции для сложения двух чисел.

```
var a, b: integer;
function sum(x, y: integer): integer;
  begin
    sum := x + y;
  end;
begin
  write('Введите два числа='); readln(a, b);
  writeln(`сумма=`, sum(a, b));
end.
```

Параметром процедуры может быть любая переменная предопределенного типа, это означает, что для передачи в процедуру массива в качестве параметра, тип его должен быть описан заранее. Например:

```
type matrix = array[1..4,1..4] of integer;
procedure Name(a: matrix; var max:integer);
```

Задание 9. Использование подпрограмм

В данной работе студент самостоятельно выбирает подходящие средства программирования: процедуры и функции. При выполнении работы набором исходных данных (не менее трех наборов) студент задается самостоятельно.

Цель работы: освоение навыков программирования вычислительных процессов с использованием подпрограмм.

Варианты заданий

1. Даны действительные числа S и t. Получить

$$z = f\left(t,-2S,1.17\right) + f\left(2.2,t,S-t\right),$$

где $f(a,b,c) = \dfrac{2a-b-\sin c}{5+|c|}$.

2. Даны действительные числа a и b. Получить

$$z = f\left(2a,-2.5,5b\right) + 3\cdot f\left(a+b,a-b,b\right),$$

где $f(x,y,z) = \dfrac{x+y+z}{2x+3z}$.

3. Даны действительные числа d и c. Получить

$$z = f\left(d,c,1.5\right) + 2f\left(d,1.3,d+c\right) + 3f\left(2d,-3c,1.8\right),$$

где $f(x,y,z) = \dfrac{5x-3y+5z}{3y^2+2z}$.

4. Даны действительные числа S и t. Получить

$$z = g\left(1.2,S\right) + g\left(t,S\right) - g\left(2S-1,St\right),$$

где $g(a,b) = \dfrac{a^2+b^2}{a^2+2ab+3b^2+4}$.

5. Даны действительные числа S и t. Получить

$$z = g\left(S,2t\right) - g\left(5t,3S\right) + g\left(8t,10S-t\right),$$

где $g(a,b) = \dfrac{a^2+3b}{2a^2+3ab+5b^2+8}$.

6. Даны действительные числа *a* и *b*. Получить

$$z = g(1.5, b) + g(a - b, b) - 3g(5a - b, ab),$$

где $g(x, y) = \dfrac{x^3 + y^2}{x^3 + 3xy + y^2}$.

7. Дано действительное число *y*. Получить

$$z = \frac{1{,}7 f(0.25) + 2 f(1.4)}{6 - f(y^2 - 1)},$$

где $f(x) = \dfrac{\sum_{k=1}^{10} \dfrac{x^{2k+1}}{2k+1}}{\sum_{k=1}^{10} \dfrac{x^{2k}}{2k}}$.

8. Даны действительные числа *a*,*b*,*c*. Получить

$$z = \frac{\max(a, a + b) + \max(a, b + c)}{1 + \max(a + b \cdot c, 1.15)}.$$

9. Даны действительные числа *a*, *b*. Получить

$$U = \min(a, b), \quad V = \min(a \cdot b, a + b), \quad z = \min\left(U - V^2, 3.14\right).$$

10. Даны действительные числа *m*, *n*, *k*, целые числа $a_1, \ldots, a_n$, $b_1, \ldots, b_m, c_1, \ldots, c_k$. Получить

$$l = \begin{cases} \min(b_1, \ldots, b_m) + \min(c_1, \ldots, c_k), & \text{при } \left|\min(a_1, \ldots, a_n)\right| > 10, \\ 1 + \left(max(c_1, \ldots, c_k)\right)^2, & \text{в противном случае.} \end{cases}$$

11. Даны действительные числа k, n, m, действительные числа $x_1,\ldots,x_k, y_1,\ldots,y_n, z_1,\ldots,z_m$. Получить

$$l=\begin{cases}(\max(y_1,\ldots,y_n)+\max(z_1,\ldots,z_m))/2, & \text{при } \max(x_1,\ldots,x_k)\ge 0,\\ 1+(\max(x_1,\ldots,x_k))^2, & \text{в противном случае.}\end{cases}$$

12. Даны действительные числа S и t. Получить

$$z=h(S,t)+max\left(h^2(S-t,S\cdot t),h^4(S-t,S+t)\right)+h(1,1),$$

где $h(a,b)=\dfrac{a}{1+b^2}+\dfrac{b}{1+a^2}-(a-b)^3$.

13. Даны действительные числа $a_1,\ldots,a_6$. Получить для x = 1, 3, 4 значения

$$z=P(x+1)-P(x),$$

где $P(y)=a_6y^6+a_5y^5+\ldots+a_1y$.

14. Даны действительные числа S, t, $a_1,\ldots,a_{12}$. Получить

$$z=P(S)-P(t)+P^2(S-t)-P^3(S),$$

где $P(y)=\sum_{i=1}^{12}a_iy^i$.

15. Даны действительные числа $a_1,\ldots,a_n$. В этой последовательности заменить все члены, идущие за членом с наибольшим значением (за первым по порядку, если их несколько), на 0.

16. Даны действительные числа $b_1,\ldots,b_n$. В этой последовательности заменить все члены, идущие за членом с наименьшим значением (за первым по порядку, если их несколько), на 1.

17. Даны целые числа $a_1,\ldots,a_n,k$. Если в этой последовательности нет ни одного члена со значением k, то заменить первый по порядку член этой последовательности, не меньший всех остальных членов, на значение k.

18. Даны целые числа $b_1,\ldots,b_n,k$. Если в этой последовательности нет ни одного члена со значением k, то заменить первый по порядку член этой последовательности, не больший всех остальных членов, на значение k.

19. Даны действительные числа $a_1,\ldots,a_n$. В этой последовательности заменить все члены, идущие за последним по порядку членом с наибольшим значением, на 0.

20. Даны действительные числа $b_1,\ldots,b_n$. В этой последовательности заменить все члены, идущие за последним по порядку членом с наименьшим значением, на 1.

Вопросы для самопроверки

1. Какие виды процедур используются в ЯВУ, их различия?
2. Объясните, в чем различие фактических и формальных параметров?
3. Объясните на примерах применение рекурсии.

3.5. Алгоритмы решения алгебраических уравнений

Во многих научных и инженерных задачах возникает необходимость решения уравнения

$$f(x, p_1, p_2, \ \ldots, p_n) = 0 \qquad (3.1)$$

где f – заданная функция; x – неизвестная величина; p_i – параметры задачи.

При фиксированном наборе p_i уравнение (3.1) может иметь либо конечное, либо бесконечное число решений. Не нарушая общности задачи можно решать уравнение (3.1) относительно любого параметра p_i, поменяв его местами с переменной x.

Решениями, или **корнями** уравнения (3.1) называются такие значения переменной x, которые при подстановке в уравнение, обращают его в тождество.

Не все уравнения можно разрешить аналитически, т. е. выразить неизвестную переменную x через параметры p_i. Численное решение уравнения (3.1) проводят в два этапа.

3.5.1. Отделение корней

На первом этапе происходит отделение корней, т. е. нахождение таких интервалов изменения переменной *x*, где расположен только один корень. Другими словами, на этом этапе находится приближенное значение корня с погрешностью, задаваемой длиной каждого интервала.

Отделение корней можно провести исходя из физического смысла задачи или из анализа ее упрощенной математической модели. При этом используют: аналитический, графический или табличный метод отделения корней.

Аналитический метод. Если функция $f(x)$ непрерывна на интервале [*a, b*] и на границах этого интервала значения функции имеют разный знак, то внутри этого отрезка имеется корень (возможно, и не единственный). Если же и производная $f(x)$ не меняет своего знака на интервале [*a, b*], то корень – единственный.

Графический метод. Исходную функцию $f(x)$ представляем в виде разности двух функций $\gamma(x)$ и $\psi(x)$:

$$f(x) = \gamma(x) - \psi(x).$$

Далее строятся графики функций $\gamma(x)$ и $\psi(x)$ и определяются приближенные интервалы их пересечения. Поскольку $f(x) = 0$, то это и будет местоположение корней.

Табличный метод. В интересующей нас области при фиксированных параметрах p_i вычисляется ряд значений функции $f(x)$. С точностью до выбранного шага определяют приближенные значения корней. Корни находятся между точками таблицы, где изменяется знак функции.

3.5.2. Уточнение отдельных корней

На втором этапе проводят уточнение отдельных корней, т. е. находят корни с требуемой точностью. Для этого используются более экономичные методы: метод дихотомии, метод хорд, метод касательных, метод секущих, метод простых итераций.

3.5.3. Метод Ньютона (метод касательных)

Предположим, что графическим методом определено начальное приближение к корню x_0. В точке x_0 вычисляем значение функции $f(x_0)$, а также значение производной в этой точке $f'(x_0)$. Сле-

дующее приближение к корню будет в точке x_1, где касательная к функции $f(x)$, проведенная в точке $(x_0, f(x_0))$, пересекает ось абсцисс.

Процесс уточнения корня продолжается до получения требуемой точности ε. Когда модуль разности полученного на *i*-м шаге приближенного значения корня и предыдущего значения корня на *i*–1-м шаге станет меньше требуемой точности, вычислительный процесс заканчивается.

Исходя из геометрических соотношений, получаем основную формулу метода Ньютона:

$$x_1 = x_0 - f(x_0) / f'(x_0).$$

Метод Ньютона обладает высокой скоростью сходимости. В качестве недостатка можно отметить необходимость вычисления не только значения функции, но и ее производной.

При аналитическом или табличном методе отделения корней за начальное приближение следует выбирать тот конец интервала, в котором знак функции $f(x)$ совпадает со знаком ее второй производной $f'(x)$.

3.5.4. Метод простых итераций

От уравнения (3.1) перейдем к эквивалентному уравнению

$$x = \varphi(x). \qquad (3.2)$$

Если известно начальное приближение к корню x_0, то, подставив его в правую часть уравнения (3.2), получим новое приближение. Признаком окончания итерационного процесса является достижение требуемой точности.

Переход от уравнения (3.1) к уравнению (3.2) можно осуществлять разными способами в зависимости от вида функции $f(x)$. Чтобы процесс итераций сходился необходимо выполнить условие

$$\left|\varphi'(x)\right| < 1.$$

При расчете методом итераций необходимо ограничивать число итераций.

Один из общих алгоритмов перехода: обе части уравнения (3.1) умножаются на произвольный параметр b, далее прибавляем к обеим частям неизвестное x. В результате получим:

$$x + bf(x) = x. \tag{3.3}$$

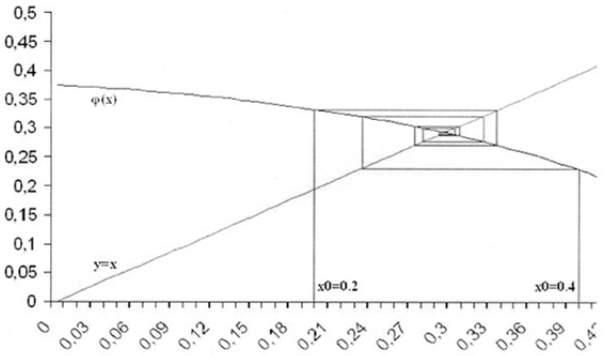
Следовательно,

$$\varphi(x) = x + bf(x).$$

Если подобрать параметр b так, чтобы

$$-1 < \varphi'(x) < 0.$$

то сходимость итерационного процесса будет двусторонней (см. рисунок).



Итерационный процесс

Пример

Необходимо найти корни уравнения $f(x) = x^2 - 2x + 0,5$.

1. Функцию $\varphi(x)$ будем искать в виде: $\varphi(x) = x + bf(x)$.

2. Таким образом, после подстановки получим: $\varphi(x) = x + b(x^2 - 2x + 0,5)$.

3. Продифференцируем: $\varphi'(x) = 1 + b(2x - 2)$.

4. Для обеспечения условия сходимости подбираем b так, чтобы выполнялось условие $-1 < \varphi'(x) < 0$.

5. Выбираем $b = -3/(4(x-1))$.

6. Подставляя полученное значение b в выражение для $\varphi(x)$, получим

$$\varphi(x) = x/4 + 3/4 + 3/(8(x-1)).$$

7. Построим графики функций $\varphi(x)$ и $y = x$ (см. рисунок).

8. Из рисунка видно, что сходимость итерационного процесса будет двусторонней.

Задание 10. Решение алгебраических уравнений

Цель работы: освоение навыков программирования вычислительных процессов решения алгебраических уравнений.

1. В уравнении по заданному варианту произвести отделение корней одним из методов по выбору преподавателя.

2. Произвести уточнение корня на отрезке [0, 1] методом Ньютона с погрешностью $\varepsilon = 10^{-3}$.

3. Произвести уточнение корня на отрезке [0, 1] методом простых итераций с погрешностью $\varepsilon = 10^{-3}$..

4. Сравнить результаты.

Таблица 3.4

Варианты к заданию 10

Номер варианта	Уравнение	Номер варианта	Уравнение
1	$x^4 + 2x^3 + x - 1 = 0$	11	$x^4 + 3x^3 + 2x - 1 = 0$
2	$x^4 + 3x^3 + x - 1 = 0$	12	$x^4 + 4x^3 + 2x - 1 = 0$
3	$x^4 + 4x^3 + x - 1 = 0$	13	$x^4 + 5x^3 + 2x - 1 = 0$
4	$x^4 + 5x^3 + x - 1 = 0$	14	$x^4 + 6x^3 + 2x - 1 = 0$
5	$x^4 + 6x^3 + x - 1 = 0$	15	$x^4 + 7x^3 + 2x - 1 = 0$
6	$x^4 + 2x^3 + 2x - 1 = 0$	16	$x^4 + 3x^3 + 3x - 1 = 0$
7	$x^4 + 2x^3 + 3x - 1 = 0$	17	$x^4 + 3x^3 + 4x - 1 = 0$
8	$x^4 + 2x^3 + 4x - 1 = 0$	18	$x^4 + 3x^3 + 5x - 1 = 0$
9	$x^4 + 2x^3 + 5x - 1 = 0$	19	$x^4 + 3x^3 + 6x - 1 = 0$
10	$x^4 + 2x^3 + 6x - 1 = 0$	20	$x^4 + 3x^3 + 7x - 1 = 0$

Вопросы для самопроверки

1. Дайте определение выражению «корень уравнения».
2. Расскажите о способах отделения корней.
3. Расскажите о способах уточнения корней.
4. Опишите алгоритм метода Ньютона.
5. Опишите алгоритм метода простых итераций.

Содержание отчета по работе над заданиями

1. Титульный лист (образец в приложении 1).
2. Цель работы.
3. Условие задачи и исходные данные.
4. Блок-схема алгоритма решения задачи.
5. Программа решения на ЯВУ.
6. Результаты вычислений, сведенные в таблицу.
7. Графики, построенные по результатам вычислений.

Отчеты выполняются только в печатном виде на листах формата А4, скрепленных вместе. При оформлении отчета используются навыки, полученные при изучении дисциплины «Информатика» в предыдущем семестре.

Допуск студента до выполнения следующего задания осуществляется только после защиты предыдущей работы.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. *Ахо А.В*. Структуры данных и алгоритмы / А.В. Ахо [и др.] – М.: Вильямс, 2016. – 400 с.

2. *Вирт Н.* Алгоритмы и структуры данных / Н. Вирт. – СПб.: Невский Диалект, 2008. – 352 с.

3. *Гашков С.Б.* Арифметика. Алгоритмы. Сложность вычислений / С.Б. Гашков, В.Н. Чубариков. – М.: Вильямс, 2005. – 320 с.

4. *Голицына О.Л.* Основы алгоритмизации и программирования / О.Л. Голицына. – М.: Инфра-М, 2015. – 432 с.

5. *Игошин В.И.* Теория алгоритмов: учебное пособие / В.И. Игошин. – М.: Инфра-М, 2013. – 320 с.

6. *Игошин В.И.* Математическая логика и теория алгоритмов. Сборник задач: учебное пособие / В.И. Игошин. – М.: Инфра-М, 2017. – 392 с.

7. *Левитин А.В.* Алгоритмы. Введение в разработку и анализ / А.В. Левитин. – М.: Вильямс, 2006. – 576 с.

8. *Фаронов В.В.* Турбо Паскаль 7.0. Начальный курс: учебное пособие / В.В. Фаронов. – М.: ОМД Групп, 2003. – 616 с.

9. Современное программирование на языке Паскаль: [Электронный ресурс]. URL: http://pascalabc.net/

Приложение

Образец титульного листа к работе над заданием

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ МЕХАТРОНИКИ И АВТОМАТИЗАЦИИ

Кафедра ЭАПУ

Задание №___

на тему _____

по дисциплине «**Информационные технологии**»

Вариант № _____

Выполнил: «___» _____
Студент: _____
Группа: _____

Преподаватель: _____
Отметка о защите _____

НОВОСИБИРСК
20__ г.

Родыгин Александр Владимирович

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

АЛГОРИТМИЗАЦИЯ И ПРОГРАММИРОВАНИЕ

Учебное пособие

Редактор *Л.Н. Ветчакова*
Выпускающий редактор *И.П. Брованова*
Дизайн обложки *А.В. Ладыжская*
Компьютерная верстка *С.И. Ткачева*

Налоговая льгота – Общероссийский классификатор продукции
Издание соответствует коду 95 3000 ОК 005-93 (ОКП)

Подписано в печать 18.07.2017. Формат 60 × 84 1/16. Бумага офсетная. Тираж 100 экз.
Уч.-изд. л. 5,34. Печ. л. 5,75. Изд. № 34. Заказ № 974. Цена договорная

Отпечатано в типографии
Новосибирского государственного технического университета
630073, г. Новосибирск, пр. К. Маркса, 20