МИНИСТЕРСТВО НАУКИ И ВЫСШЕО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

СЕВЕРО-КАВКАЗСКАЯ ГОСУДАРСТВЕННАЯ АКАДЕМИЯ

Институт прикладной математики и информационных технологий

Кафедра Информатики и информационных технологий

П. А. Кочкарова
М.У. Эркенова

СТРУКТУРЫ И АЛГОРИТМЫ КОМПЬЮТЕРНОЙ ОБРАБОТКИ ДАННЫХ

Учебно-методическое пособие для обучающихся 3 курса направлению подготовки 09.03.03 Прикладная информатика

Черкесск, 2023

УДК 004.6
ББК 32.973.05
К 75

Рассмотрено на заседании кафедры «Прикладная информатика».
Протокол № 2 от 19 сентября 2022 г.
Рекомендовано к изданию редакционно-издательским советом СКГА.
Протокол № 24 от 26 сентября 2022 г

Рецензенты: – Бостанова Л.К. – к.п.н., доцент кафедры информатики и информационных технологий

К75 **Кочкарова, П. А.** Структуры и алгоритмы компьютерной обработки данных: учебно-методическое пособие для обучающихся направления подготовки 09.03.03 Прикладная информатика / П. А. Кочкарова.– Черкесск: БИЦ СКГА, 2023. – 60 с.

УДК 004.6
ББК 32.973.05

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Предлагаемое учебно–методическое пособие предназначено для обучающихся по направлению подготовки 09.03.03 «Прикладная информатика» и направлено на получение навыков использования основных структур данных и алгоритмов их обработки.

В пособии по каждой теме имеется теоретический материал, контрольные примеры и задания для самостоятельного выполнения.

На контрольных примерах показаны способы реализации алгоритмов на языке TurboPascal, хотя задания можно выполнить с использованием и другого языка программирования.

Тема 1. Рекурсивные функции и процедуры

Краткие теоретические сведения

Алгоритм называется рекурсивным, если он прямо или косвенно обращается к самому себе. Часто в основе такого алгоритма лежит рекурсивное определение какого-то понятия.

Например, вычисление факториала можно определить следующим образом:

$$N! = \begin{cases} 1, \text{если N} = 1 \\ N * (N-1)!, \text{если N} > 1 \end{cases}$$

Рекурсивный алгоритм обращается к самому себе, пока не выполнится определенное условие, поэтому в любой рекурсивной подпрограмме должна быть не рекурсивная ветвь

(в примере это: if N=1 then Fact:=1).

Примером программы с использованием рекурсии может быть программа вычисления факториала числа.

Пример1.

Вычисление факториала числа N с использованием рекурсивной функции.

```
program F_Rekurs;
var
N : integer;
F : longint;
{Описание функции, N — формальный параметр-значение типа integer, результат выполнения функции типа longint}.
function Fakt(N : integer): longint;
begin
{Начало вычисления функции}
if N=1 then Fakt:=1 {Проверка условия завершения рекурсии}
else Fakt:=N*Fakt(N-1); {Рекурсивное вычисление N!}
end;
begin
{Начало главной программы}
Writeln(' Введите число N : ') ;
Read(N) ;
F:=Fakt(N); {Вызов функции для фактического параметра N}
Writeln('Для числа ,N,' значение факториала= ', F);
end.
```

При выполнении рекурсивной подпрограммы осуществляется многократный переход от некоторого текущего уровня организации алгоритма к нижнему уровню последовательно до тех пор, пока, наконец, не будет получено тривиальное решение поставленной задачи. В рассмотренном выше примере решение при N=1 тривиально, т.е. Fakt=1. Затем осуществляется возврат на верхний уровень с последовательным вычислением значения функции Fakt.

Следует учитывать, что использование рекурсивной формы организации алгоритма обычно выглядит изящнее итерационной и дает более компактный текст программы, но при выполнении, как правило, медленнее и может вызвать переполнение стека. Это объясняется тем, что при каждом входе в подпрограмму ее локальные переменные размещаются в особым образом организованной области памяти, называемой программным стеком.

Пример 2.

Расчет суммы первых N натуральных чисел

Вычисление суммы 1+2+3+…+N можно определить следующим образом:

$$Sum(n) = \begin{cases} 0, \text{если} N = 0 \\ \text{Sum}(N - 1) + N \end{cases}$$

```
Program Pr2;
Var
N: integer;
Function Sum(N: integer): integer;
begin
if N=0 then Sum:=0 else Sum:= N+Sum(N-1);
end;
BEGIN {основная программа}
Write ('Введите число N='Введите число ); Readln(N);
Writeln ('Введите число S='Введите число , Sum(N));
Readln
END.
```

Пример 3.

Вычисление n-го члена последовательности Фибоначчи.

Последовательность Фибоначчи определяется следующим образом: первые два числа равны 1, а каждое следующее равно сумме двух предыдущих (1, 1, 2, 3, 5,…).

Вычисление n-го члена последовательности Фибоначчи можно определить следующим образом:

$$Fib(n) = \begin{cases} 1, \text{если } N = 1 \text{ или } N = 2 \\ \text{Fib}(n - 1) + \text{Fib}(n - 2), \text{если } N > 2 \end{cases}$$

```
Program Pr3;

Var
n:integer;
Function Fib(n:integer):integer;
begin
if n<=2 then Fib:=1
else
Fib:=Fib(n-1)+Fib(n-2);
end;
```

```
BEGIN {основная программа}
Write(‘Введите число n=’);
Readln(n);
Write(Fib(n):5);
Readln
END.
```

Пример 4.

Расчет n-члена арифметической прогрессии, заданной значением первого члена a_1 и разностью d.

```
Program Pr4;
Var
n:integer; d,a1:real;
Function Arifm(a1,d:real; n:integer):real;
begin
if n=1 then Arifm:=a1 else Arifm:=Arifm(a1,d,n-1) + d;
end;
BEGIN {основная программа}
Write('Введите число a1='Введите число ); Readln(a1);
Write('Введите число d='Введите число ); Readln(d);
Write('Введите число n='Введите число ); Readln(n);
Write(Arifm(a1,d, n):5:2);
Readln
END.
```

Контрольные вопросы

Контрольные вопросы

1. Определение рекурсивного алгоритма.
2. Какое условие должно обязательно присутствовать в любой рекурсивной
3. Где размещаются локальные переменные при рекурсивном вызове процедуры?
4. Пример программы с использованием рекурсии.

Задания для самостоятельной работы

1. Вычислить (a! + b!)/a!, используя рекурсивную функцию вычисления факториала
2. Вычислить m!/(m + n)!, используя рекурсивную функцию вычисления факториала
3. Вычислить (1+2+3+4+5)/(1+2+3+4+5+6+7+8), используя рекурсивную функцию вычисления суммы первых n натуральных чисел.
4. Составить рекурсивную функцию для вычисления S = 2 + 4 + 6 +…+2*n
5. Составить рекурсивную функцию для вычисления S = 5 + 10 + 15 +…+5*n
6. Составить рекурсивную функцию для вычисления P = 2* 4* 6*…*2*n

7. Составить рекурсивную функцию вычисления n-го члена арифметической прогрессии 3, 7, … и вывести первые 10 членов прогрессии .

8. Составить рекурсивную функцию вычисления n-го члена последовательности: $a_1=1$, $a_i = a_{i-1}*i$. Найти сумму 2-го и 5-го членов последовательности.

9. Составить рекурсивную функцию вычисления n-го члена последовательности: $a_1=0$, $a_i = 2*a_{i-1}+i$. Найти произведение 3-го и 7-го членов последовательности.

10. Составить рекурсивную функцию нахождения суммы n членов арифметической прогрессии 1, 3, …Найти сумму с 5-го по 10-й членов прогрессии.

11. Составить рекурсивную процедуру, которая печатает введенное натуральное число в восьмеричном представлении

12. Найти наибольший общий делитель для чисел A, B, C, используя рекурсивную функцию нахождения наибольшего общего делителя двух натуральных чисел

13. Сократить дробь a/b (a, b – введенные натуральные числа), используя рекурсивную функцию нахождения наибольшего общего делителя двух натуральных чисел

14. Составить рекурсивную функцию вычисления суммы цифр произвольного натурального числа.

15. Составить рекурсивную функцию вычисления среднего арифметического цифр произвольного натурального числа.

16. Составить рекурсивную функцию, которая вычисляет среднее арифметическое элементов одномерного массива.

17. Найти первые N чисел Фибоначчи двумя способами: с помощью рекурсии и с помощью итерации. Сравнить эффективность алгоритмов.

18. Составить рекурсивную функцию для вычисления суммы элементов одномерного массива.

19. Составить рекурсивный алгоритм нахождения максимального элемента одномерного массива

20. Составить рекурсивный алгоритм нахождения минимального элемента одномерного массива

Тема 2. Структура данных – массив

Краткие теоретические сведения

Массивы являются представителями структурированных типов данных, то есть таких, переменные которых составлены из более простых элементов согласно определённому порядку. Для массивов характерно то, что они являются совокупностью некоторого числа *одинаковых* элементов. В простейшем случае эти элементы могут быть занумерованы натуральными числами из некоторого диапазона. Рассмотрим пример такой переменной в Турбо Паскале:

```
var a: array [1..10] of real;
```

Переменная a состоит из десяти ячеек типа real, можно записывать и извлекать значения из них, пользуясь записью a[<*номер ячейки*>].

В качестве типа элементов массива можно использовать все типы, известные на данный момент (к ним относятся все числовые, символьный, строковый и логический типы).

Если каждый элемент массива содержит только один индекс, то такие массивы называется *одномерными*. Одномерный массив можно представить как строку или столбец переменных. Он аналогичен одномерному числовому вектору и имеет индивидуальное имя. На Паскале массивы объявляют в разделе описания переменных, служебным словом array и описывают его тип.

Описание одномерных массивов:

Type тип=array[1..n] of тип элементов;

Var имя массива:тип;

Нумеровать элементы массивов можно не только от единицы, но и от любого целого числа. Вообще для индексов массивов подходит любой порядковый тип, то есть такой, который в памяти машины представляется целым числом. Рассмотрим некоторые примеры объявления массивов:

Пример. В массиве X(20) определить максимальный по значению элемент и напечатать значения этого элемента.

```
program max;
type mas=array[1..20]of integer;
var A: mas; i: byte; max: integer;
begin
{блок заполнения}
for i: =1 to 20 do
readln (A[i]);
{поиск максимального элемента}
max:=A[1];
for I:=2 to 20 do
if A[i]>max then
max:=A[i];
write('макс.элемент=',max);
end.
```

Если каждый элемент массива содержит несколько индексов, то он называется многомерным. Двумерному массиву соответствует матрица, то есть прямоугольная таблица.

Описание двумерных массивов:

Type тип=array[1..n,1..m] of тип элементов;

Var имя массива: тип;

Пример описания двумерного массива:

var Matrix: array[1..4,1..3] of real;

Пример описания трехмерного массива:

Cube3D: array[1..5,1..5,1..5] of integer;

Пример. В массиве X(2,3) определить максимальный по значению элемент и напечатать значения этого элемента.

```
program max;
type mas=array[1..2,1..3]of real;
var A: mas; i,j: integer; max: real;
begin
{ввод данных}
for i:=1 to 2 do
for j:=1 to 3 do
readln (A[i,j]);
{поиск максимального элемента}
max:=A[1,1];
for i:=1 to 2 do
for j:=1 to 3 do
if A[i,j]>max then
max:=A[i,j];
write('макс.элемент=',max);
end.
```

Контрольные вопросы:

1. Дайте определение массива.
2. Какие типы недопустимы для компонентов массива? Почему?
3. В каком разделе программы надо написать описание данных?
4. Как определяется общее число элементов массива?
5. Что такое индекс? Какие типы данных используется в качестве индекса?
6. Какой массив называется одномерным?
7. Существует ли ограничения на размерность массива?
8. Какой массив называется двумерным?
9. Как описывается двумерный массив?
10. Как осуществляется доступ каждому элементу двумерного массива?

Задания для самостоятельной работы

Задание 1.

1. Дан одномерный массив A(20) . Упорядочить компоненты массива так, чтобы сначала размещались все отрицательные компоненты, а затем положительные.
2. Найти среднее арифметическое элементов массива A(20), расположенных после первого отрицательного элемента.
3. Найти среднее арифметическое элементов массива A(20), предшествующих первому отрицательному элементу.
4. В массиве B(10) вычислить сумму отрицательных, произведение положительных и количество нулевых элементов.
5. Ввести массив X(15). Подсчитать количество всех чисел, расположенных в промежутке [-10,10] и сумму всех остальных.

6. Дан массив X(25). . Создать новый массив Y(25) элементы которого вычисляются следующим образом: $y_i = \begin{cases} \sqrt[3]{\lvert x_i + 1 \rvert} & , если \quad x_i \le 0 \\ \sin^2 x_i & , для \ остальных \quad x_i \end{cases}$.

7. Дан массив B(10) . В элемент B_i , содержащий наименьшее значение, записать среднее арифметическое значение элементов массива.

8. В массиве B(15) подсчитать количество нулевых элементов k и вычислить k!.

9. Дан массив C(20) . Все элементы, стоящие после C_i , имеющего наибольшее значение, заменить нулями.

10. Дан массив A(24) . Вычислить сумму

$S = a_1 a_2 a_3 a_4 + a_5 a_6 a_7 a_8 + \ldots + a_{21} a_{22} a_{23} a_{24}$.

11. Дан массив A(15). Найти произведение всех элементов, значения которых меньше 50, и сложить его с произведением элементов больших 100.

12. Даны два массива X(12) и Y(12) . На место массива X записать массив Y, а на место массива Y – массив X.

13. Написать программу, которая проверяет, представляют ли элементы введенного с клавиатуры массива неубывающую последовательность.

14. Написать программу, которая определяет количество студентов в группе, чей рост превышает средний.

15. Даны целые числа A(n). Если в данной последовательности ни одно четное число не расположено после нечетного, то получить все отрицательные члены последовательности, иначе – все положительные.

16. Даны действительные числа $a_1, \ldots, a_n$. Оставить без изменения последовательность $a_1, \ldots, a_n$, если она упорядочена по не убыванию или не возрастанию; в противном случае удалить из последовательности те члены, порядковые номера которых кратны четырем, сохранив прежним порядок остальных членов.

17. Даны действительные числа $a_1, \ldots, a_n$. Выяснить, имеются ли среди чисел $a_1, \ldots, a_n$ совпадающие, и если есть, то определить их количество и порядковые номера.

18. Даны вектора $a_1, \ldots, a_n$ и $b_1, \ldots, b_n$. Получить вектор c, компонентами которого являются $a_1 b_k, a_2 b_{k-1}, \ldots, a_k b_1$. Найти минимальный и максимальный элемент вектора c и их порядковые номера.

19. Даны целые числа $a_1, \ldots, a_n$, среди которых могут быть повторяющиеся. Выяснить, сколько чисел входит в последовательность по одному разу.

20. Даны целые числа $m, a_1, a_2, \ldots, a_{10}$. Найти три натуральных числа i, j, k, каждое из которых не превосходить десяти, такие что $a_i + a_j + a_k = m$. Если таких чисел нет, то сообщить об этом.

Задание 2.

1. Вычислить вектор, равный произведению матрицы A(5,5) и вектора B(5) .
2. Для матрицы B(5,4) на место последнего элемента каждой строки записать сумму предыдущих ему элементов в этой строке.
3. В матрице C(4,5) найти среднее арифметическое всех элементов, удовлетворяющих условию $5<c_{ij}<10$.
4. В матрице A(4,5) определить max и min элементы и переставить их местами.
5. В матрице A(4,3) определить среднее арифметическое положительных и отрицательных элементов. Подсчитать количество нулевых элементов.
6. Вычислить k=n! , где n - количество нулевых элементов в четных строках матрицы B(6,6).
7. В матрице A(6,6) все поддиагональные элементы заменить нулями.
8. В матрице D(5,6) найти сумму элементов каждой четной строки и произведение элементов каждой нечетной строки. Из полученных сумм и произведений образовать массив X(5) .
9. В матрице X(5,4) в каждой строке найти сумму элементов, лежащих в интервале $-10<x_{ij}<10$.

Образовать из этих сумм пятый столбец исходной матрицы.

10. Задана матрица X(5,5). Образовать матрицу Y(5,5) путем деления всех элементов исходной матрицы на ее элемент, наибольший по абсолютной величине.
11. В матрице A(10,10) найти сумму элементов, расположенных в строках с отрицательным элементом на главной диагонали.
12. Задана матрица X(6,6). Вычислить сумму элементов, расположенных на главной диагонали и находящихся выше нее.
13. В матрице X(8,8) найти среднее арифметическое элементов, расположенных под главной диагональю; на главной диагонали; над главной диагональю.
14. Дана действительная матрица X(5,5)натуральное число m. Вычислить произведение тех элементов матрицы сумма индексов которых равна m.
15. 12. Вычислить количество и сумму отрицательных элементов массива X(5,4).
16. Сформировать матрицу X(5,4) по формуле $x_{ij}=\frac{i+j}{2}$
17. Поменять местами максимальный и минимальный элементы действительного массива X(5,4)
18. Дана матрица A(5,10). Построить вектор $B(b_1,\ldots,b_5)$, i-я компонента которого равна количеству положительных элементов i-строки матрицы A.
19. Дана целочисленная квадратная матрица A порядка n. Найти ее определитель, приведя матрицу к треугольному виду.

20. Целочисленная матрица A(20,4) содержит сведения о результатах сессии из 4 экзаменов для группы из 25 студентов. Найти количество и фамилии студентов, имеющих все оценки «5».

Тема 3. Работа со строками

Краткие теоретические сведения

Строка представляет собой особую форму одномерного массива символов, которая имеет существенное отличие. Массив символов имеет фиксированную длину (количество элементов), которая определяется при описании. Строка имеет две разновидности длины: – **Общая длина строки,** которая характеризует размер памяти, выделяемый строке при описании; – **Текущая длина строки** (всегда меньше или равна общей длине), которая показывает количество смысловых символов строки в каждый конкретный момент времени.

Строка в Паскале – упорядоченная последовательность символов. Количество символов в строке называется ее длиной. Длина строки в Паскале может лежать в диапазоне от 0 до 255. Каждый символ строковой величины занимает 1 байт памяти и имеет числовой код в соответствии с таблицей кодов ASCII.

Для описания строковых переменных в Паскале существует предопределенный тип string. В общем виде описание строковой переменной будет выглядеть следующим образом:

Var : string[]

Операции со строками

Операция слияния (сцепления, конкатенации) применяется для соединения нескольких строк в одну, обозначается знаком «+». Операция слияния применима для любых строковых выражений, как констант, так и переменных.

Операции отношения позволяют сравнивать строки на отношение равенства (=), неравенства (<>), больше (>), меньше (=), меньше или равно (<=).

В результате сравнения двух строк получается логическое значение (true или false).

Сравнение строк производится слева направо посимвольно до первого несовпадающего символа, большей считается та строка, в которой первый несовпадающий символ имеет больший код в таблице кодировки. Если строки имеют различную длину, но в общей части символы совпадают, считается, что короткая строка меньше. Строки равны, если они имеют равную длину и соответствующие символы совпадают.

Стандартные функции для работы со строками в Паскале

Copy (S, poz, n) – выделяет из строки S, начиная с позиции poz, подстроку из n символов. Здесь S – любое строковое выражение, poz, n – целочисленные выражения.

Concat (s1, s2,...,sn) – выполняет слияние строк s1, s2,...,sn в одну строку.

Length(S) – определяет текущую длину строкового выражения S. Результат – значение целого типа.

Pos(subS, S) – определяет позицию первого вхождения подстроки subS в строку S. Результат – целое число, равное номеру позиции, где находится первый символ искомой подстроки. Если вхождение подстроки не обнаружено, то результат функции будет равен 0.

Стандартные процедуры для работы со строками в Паскале:

Delete (S, poz, n) – удаляет из строки S, начиная с позиции poz, подстроку из n символов. Здесь S – строковая переменная (в данном случае нельзя записать никакое другое строковое выражение, кроме имени строковой переменной, т.к. только с именем переменной связана область памяти, куда будет помещен результат выполнения процедуры); poz, n – любые целочисленные выражения.

Insert(subS, S, poz) – вставляет в строку S, начиная с позиции poz, подстроку subS. Здесь subS – любое строковое выражение, S – строковая переменная (именно ей будет присвоен результат выполнения процедуры), poz – целочисленное выражение.

Процедуры преобразования типов в Паскале:

Str(x, S) – преобразует число x в строковый формат. Здесь x – любое числовое выражение, S – строковая переменная. В процедуре есть возможность задавать формат числа x.

Например, str(x: 8: 3, S), где 8 – общее число знаков в числе x, а 3 – число знаков после запятой.

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

Пример.

Дана строка. Определить, сколько раз в нее входит группа букв «гол».

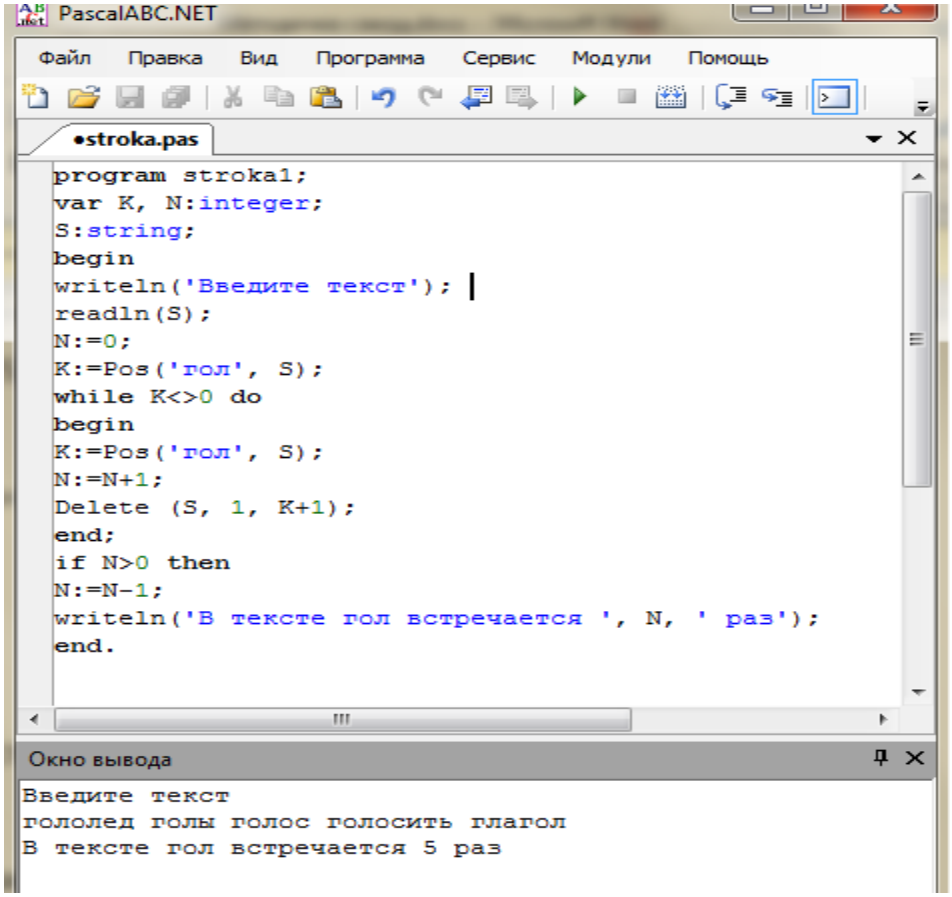
Алгоритм задачи:

а) для поиска первого вхождения сочетания «гол» используем функцию Pos, она покажет первый символ к;

б) дальше надо отрезать от строки символы с первого по к+1 и снова воспользоваться указанной функцией;

в) выполнять эти действия, пока значение Pos не станет нулевым, но когда это произошло, N все равно увеличивается на 1, поэтому печатаем N-1.

```
program stroka1;
var K, N:integer;
S:string;
begin
writeln('Введите текст');
readln(S);
N:=0;
K:=Pos('гол', S);
while K<>0 do
begin
K:=Pos('ешь', S);
N:=N+1;
Delete (S, 1, K+1);
end;
if N>0 then
N:=N-1;
writeln('В тексте ешь встречается ', N, ' раз');
end.
```



Контрольные вопросы

1. Понятие строки.
2. Чему равна длина строки в Паскале?
3. Операции над строками в Паскале

4. Стандартные функции работы со строками
5. Стандартные процедуры для работы со строками в Паскале
6. Процедуры преобразования типов в Паскале

Задания для самостоятельной работы

1. Даны целые положительные числа n1 и n2 и строки S1 и S2. Получить из этих строк новую строку, содержащую первые n1 символов S1 и последние n2 символов строки S2.

2. Даны целое положительное число n и строка S. Преобразовать строку S в строку длины n следующим образом: если длина S больше n, то отбросить первые символы, если меньше, то в начало добавить «.».

3. Даны строки S1 и S2. Проверить, содержится ли S1 в строке S2.

4. Даны строки S1 и S2. Удалить из строки S1 подстроку S2, если S2 не содержится в S1, вывести S1 без изменений

5. Даны строки S1 и S2. Удалить из строки S1 первую подстроку S2, если S2 не содержится в S1, вывести S1 без изменений

6. Даны строки S1 и S2. Удалить из строки S1 последнюю подстроку S2, если S2 не содержится в S1, вывести S1 без изменений

7. Даны строки S1 и S2. Удалить из строки S1 все подстроки S2, если S2 не содержится в S1, вывести S1 без изменений

8. Даны строки S, S1 и S2. Заменить в строке S последнюю подстроку S1 на S2, если S1 не содержится в S, вывести S без изменений

9. Даны строки S, S1 и S2. Заменить в строке S первую подстроку S1 на S2, если S1 не содержится в S, вывести S без изменений

10. Даны строки S, S1 и S2. Заменить в строке S все подстроки S1 на S2, если S1 не содержится в S, вывести S без изменений

11. Дано полное имя файла, т.е. путь к файлу, имя и расширение. Выделить из строки только имя файла без расширения.

12. Дано полное имя файла, т.е. путь к файлу, имя и расширение. Выделить из строки только расширение файла без точки.

13. Дано полное имя файла, т.е. путь к файлу, имя и расширение. Выделить из строки только имя первого каталога без символа \.

14. Дано полное имя файла, т.е. путь к файлу, имя и расширение. Выделить из строки только имя последнего каталога без символа \.

15. Дана строка, содержащая круглые скобки. Выдать сообщение, правильно ли расставлены скобки (количество открывающихся соответствует количеству закрывающихся)

16. Строка содержит одно слово. Проверить, будет ли оно читаться одинаково слева направо и наоборот.

17. В строке заменить все пробелы на «_», посчитать количество замененных символов

18. В строке заменить все пробелы на «*»,посчитать количество замененных символов

19. Дана строка. Определить, сколько в ней «;»

20. Удалить часть символьной строки в скобках (вместе со скобками)

21. Удалить часть символьной строки в кавычках (вместе с кавычками)

22. Дана строка, содержащая кавычки. Выдать сообщение, правильно ли расставлены кавычки (количество открывающихся соответствует количеству закрывающихся)

23. Дана строка. Найти количество слов, начинающихся с буквы А

24. 26 Дана строка. Заменить первую букву слов, начинающихся с буквы А, на малую букву

25. Дана строка. Заменить последнюю букву слов, заканчивающихся на е, на большую букву

26. Дана строка. Определить, сколько раз в нее входит группа букв ель

27. Дана строка. Определить, сколько раз в нее входит группа букв ого

Тема 4. Множественный тип данных

Краткие теоретические сведения

Множеством называется совокупность однотипных элементов, рассматриваемых как единое целое. В Турбо Паскале множество может содержать от 0 до 255 элементов.

В отличие от элементов массива элементы множества не пронумерованы, не упорядочены. Каждый отдельный элемент множества не идентифицирован, с ним нельзя выполнить какое-либо действие. Действия могут выполняться только над множеством в целом.

Тип элементов множества называется базовым типом. Базовый тип может быть любым скалярным, за исключением типа **Real.**

Конкретные значения множества задаются с помощью конструктора множества, представляющего собой список элементов, заключенный в квадратные скобки. Сами константы могут быть константами или выражениями базового типа.

Примеры задания множеств:

[3,4,7,9,12]- множество из пяти целых чисел;

[1..100] – множество целых чисел от 1 до 100;

[‘a’,’b’,’c’] – множество из трех букв a,b,c;

[‘A’..’Z’,’!’] –множество, содержащее все прописные латинские буквы, а также знак !.

Описание переменных множественного типа:

Var <идентификатор>: Set of <базовый тип>;

Например:

Var A,D: Set of Byte;

B: Set of ‘a’..’z’;

C: Set of Boolean;

Нельзя вводить значения во множественную переменную оператором ввода и выводить оператором вывода. Множественная переменная может получить конкретное значение только в результате оператора присваивания:

<множественная переменная>:=<множественное выражение>;

Например:

A:=[50,90,450];

B:=[‘m’, ‘n’, ‘k’];

C:=[True, False];

D:=A;

Операции над множествами.

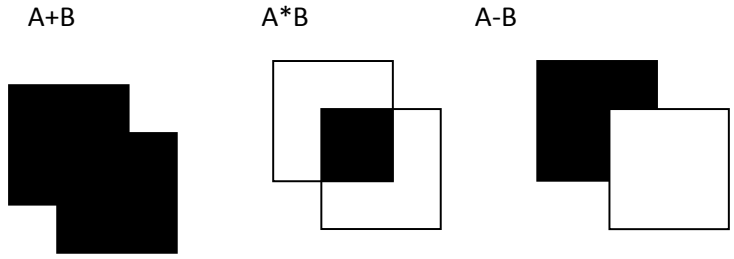
Объединение множеств. Объединением двух множеств A и B называется множество, состоящее из всех элементов, принадлежащих хотя бы одному из множеств A или B. Знак операции объединения в Паскале +.

Пересечение множеств. Пересечением двух множеств A и B называется множество, состоящее из всех элементов принадлежащих, одновременно множеству A и множеству B. Знак операции пересечения в Паскале *.

Разность множеств. Разностью двух множеств Аи B называется множество, состоящее из элементов множества A, которые не входят в B.

Знак операции -.

На рисунке схематически показаны результаты объединения, пересечения и разности двух множеств A и B.



Операции отношения. Множества можно сравнивать между собой, т.е. для них определены операции отношения. Результатом отношения является логическая величина true или false. Предполагается, что A и B содержат элементы одного типа.

Отношения:

A=B {Множества A и B совпадают}

A<>B { Множества A и B не совпадают }

A<=B {Все элементы A принадлежат B }

A>=B { Все элементы B принадлежат A }

Операция вхождения. Эта операция устанавливает связь между множеством и скалярной величиной, тип которой совпадает с базовым типом множества.

Если x –такая скалярная величина, а M – множество, то операция вхождения записывается так:

X In M

Результат – логическая величина true, если значение x входит в множество M и false – в противном случае.

Пример 1.

Дана символьная строка. Подсчитать в ней количество знаков препинания **(. - , ; : ! * ?).**

```
Program P1;
Var S: String; I,k: Byte;
Readln(S); K:=0;
For I:=1 To Length(S) Do
If  S[I] In ['.','-', ',',';',':','!','*','?']
   Then K:=K+1;
Writeln(' Число знаков препинания равно',K);
End.
```

Пример 2.

Напечатать в возрастающем порядке все цифры, входящие в десятичную запись натурального числа.

```
Program MNV;
Var Sd: Set of 0..9;
n, I: Integer;
d: 0..9;
Begin
{Ввод исходного числа}
Readln(n);
{Выделение справа налево цифр из числа n и запись их во множество sd}
sd:=[ ];
Repeat
d:=n mod 10;   sd:=sd+[d];   n:=n div 10;
until n=0;
{Печать результата}
For d:=0 To 9 Do
If d In sd Then Write(d:3);
END.
```

Контрольные вопросы

1. Что такое множество?
2. Как описать тип множество?
3. Как присвоить значение множеству?
4. Какие операции выполняются над множествами?

Задания для самостоятельной работы

Составить программу для решения задачи.

1. Подсчитать количество различных чисел в десятичной записи натурального числа.

2. Напечатать в возрастающем порядке все цифры, не входящие в десятичную запись натурального числа.

3. В возрастающем порядке напечатать все целые числа из диапазона 1..10000, представимые в виде n^2+m^2, n,m≥0.

4. Дан текст из строчных латинских букв, за которым следует точка. Определить, каких букв – гласных (a,e,i,o,u) или согласных - больше в этом тексте.

5. Дан текст из цифр и строчных латинских букв, за которым следует точка. Определить каких символов больше, букв или цифр.

6. Дан текст из строчных латинских букв, за которым следует точка. Напечатать все буквы, входящие в текст не менее двух раз.

7. Дан текст из строчных латинских букв, за которым следует точка. Напечатать первые вхождения букв в текст, сохраняя их исходный взаимный порядок.

8. Дан текст из строчных латинских букв, за которым следует точка. Напечатать все гласные буквы, входящие в текст не менее двух раз.

9. Дан текст из строчных латинских букв, за которым следует точка. Напечатать все буквы, входящие в текст по одному разу.

10. Дан текст, за которым следует точка. Напечатать в алфавитном порядке все строчные русские буквы, входящие в этот текст.

11. Дан текст, за которым следует точка. Напечатать в алфавитном порядке все строчные гласные русские буквы (а,у,и,о,у,ы,э,ю,я), входящие в этот текст.

12. Var x,y,z: Set of [0..22]; переменной x присвоить множество всех целых чисел от 8 до 22, y – множество всех простых чисел из этого диапазона, а переменной z – множество всех составных чисел из этого же диапазона.

13. Дан текст из строчных латинских букв, за которым следует точка. Напечатать все гласные буквы, входящие в текст.

14. В возрастающем порядке напечатать все целые числа из диапазона 1..256, представимые в виде n2+m, n,m≥0.

15. В возрастающем порядке напечатать все целые числа из диапазона 1..10000, представимые в виде n2+2m2, n,m≥0.

16. Из множества целых чисел 1..20 выделить: множество чисел, делящихся без остатка на 6; множество чисел, делящихся без остатка или на 2, или на 3

17. Из множества целых чисел 1..100 выделить множество чисел, являющихся, в свою очередь, квадратами целых чисел

18. Из множества целых чисел 1..100 выделить множество чисел на которые делится без остатка число 444. Вывести это множество на экран

19. Пусть заданы множество A и множество B целых чисел, вывести напечатать все элементы множества A которые не входят в множество B.

20. Задано множество целых чисел от 1 до 255. Получить из этого множества новое множество, в которой отсутствуют числа, кратные 3

Тема 5. Структура данных -запись

Краткие теоретические сведения

Запись представляет собой объединение фиксированного числа логически связанных компонентов, называемых полями. Компоненты записи различаются между собой именами полей и могут относиться к разным типам.

Тип запись описывается по схеме:

```
Type < имя типа записи >=record
<Имя поля 1>: тип 1;
<Имя поля2>: тип 2;
……..
<Имя поля n>: тип n;
End;
```

Опишем тип Abitur – сведения об абитуриенте (фамилия, имя, отчество, оценки по 3 предметам, средний балл).

```
Type Abitur = Record
Fam, Im, Otch: String[20];
Pr1,Pr2,Pr3:1..5;
Sroc: Real;
End;
Var Ab: Abitur;
```

Записи предоставляют возможность прямого доступа к любому полю. Доступ к отдельным полям переменных входящих в запись, осуществляется с помощью составных имен, включающих в себя имя записи и имя ее компонента (поля), разделенных точкой. Например, для полей переменной Ab типа Abitur можно использовать операторы:

```
Redln(Ab.Fam,Ab.Im);
Ab.Pr1:=4;
Ab.Pr2:=5;
Ab.Pr3;=5;
Ab.Sroc:=(Ab.Pr1+ Ab.Pr2+ Ab.Pr3);
```

При неоднократном обращении к одному и тому же полю записи или к нескольким полям одной и той же записи удобно воспользоваться оператором присоединения. Общий вид записи оператора присоединения:

```
With v Do s;
```

Где v – имя записи, s – оператор (простой или составной).

Используя оператор присоединения к предыдущему примеру, можно записать:

```
With Ab Do
Begin
Redln(Fam,Im);
Pr1:=4;
Pr2:=5;
```

```
Pr3;=5;
Sroc:=(Pr1+ Pr2+Pr3);
End;
```

Оператор присоединения позволяет использовать вместо составных имен непосредственно имена полей. При этом имя записи выносится в заголовок оператора присоединения. Внутри оператора к полям записи обращаются только по имени поля, а транслятор подставляет имя записи.

Пример. Составить программу для решения задачи.

В Группе 30 студентов. По известным фамилиям и оценкам по 4 предметам найти среднюю оценку каждого студента и выдать на печать фамилию и средний балл лучшего студента.

Для решения задачи потребуется массив, в котором будут храниться сведения обо всех студентах группы, причем эти сведения будут иметь вид записи.

```
Program zapis;
Type
Sved=record
Fam:string[25];
p1,p2,p3,p4:1..5;
sroc:Real;
End;
Var spisok: Array[1..30] of sved;
I,n: Integer;
champ: Real;
Begin                {Ввод сведений о студентах}
For I:=1 To 30 Do
With spisok[I] Do
Begin
Writeln('Введите фамилию студента ');
Readln(Fam);
Writeln(' Введите оценки по 4 предметам');
Readln(p1,p2,p3,p4);
End;
For I:=1 To 30 Do  {Нахождение среднего балла студента}
With spisok[I] Do
sroc:=(p1+p2+p3+p4)/4;
champ:=0;          {Нахождение максимального среднего балла}
For I:=1 To 30 Do
If spisok[I].sroc>=champ Then champ:= spisok[I].sroc;
{Печать списка студентов с максимальным средним баллом}
For I:=1 To 30 Do
If spisok[I].sroc=champ  Then
With spisok[I] Do
Writeln(Fam:30,' ',sroc:8:3);
Readln;
End.
```

Контрольные вопросы

1. Комбинированный тип. Записи
2. Описание записей
3. Операции с записями
4. Оператор With

Задания для самостоятельной работы

1. В отделе работает восемь человек. Вывести на экран следующие данные: фамилию, имя, отчество, стаж работы работников, чей возраст не превышает 30 лет. Исходные данные ввести с клавиатуры.

2. Сотрудник налоговой инспекции оштрафовал за день шесть человек. Вывести на экран фамилии, номера машин, сумму штрафа для водителей, оштрафованных более чем на 200 рублей и общую сумму штрафов. Исходные данные ввести с клавиатуры.

3. Расчетная ведомость содержит данные о фамилии, табельном номере и сумме заработка за январь и февраль семи работников. Вывести на экран все данные о работниках, чей январский заработок превышает февральский. Исходные данные ввести с клавиатуры.

4. Акт инвентаризации материальных ценностей содержит шесть наименований предметов (шкаф, стол и т.д.), их количество и стоимость. Вывести на экран данные акта и суммарную стоимость по наименованиям. Исходные данные ввести с клавиатуры.

5. Реестр поступлений за коммунальные услуги за год жильцов восьмиквартирного дома содержит лицевые счета и суммы поступлений за отопление, газ и электроэнергию. Вывести на экран данные реестра и общую сумму оплаты по каждому счету. Исходные данные ввести с клавиатуры.

6. Ведомость заработной платы содержит фамилии десяти работников, их табельные номера, сумму зарплаты, сумму премии, составляющей 20% от зарплаты и сумму к выдаче. Подсчитать сумму премии и сумму к выдаче, остальные данные ввести с клавиатуры. Вывести на экран ведомость.

7. Журнал отгрузки готовой продукции содержит наименования шести видов продукции, количество, цену за единицу продукции, суммарную стоимость по наименованиям. Вычислить суммарную стоимость, остальные данные ввести с клавиатуры. Вывести на экран все данные журнала.

8. Реестр актов на покупку продуктов содержит наименование десяти продуктов. Ввести с клавиатуры стоимость за 1 кг, количество продукта в кг. Найти общую стоимость по наименованиям. Вывести на экран все данные реестра.

9. Ведомость затрат по шести заказам содержит номер заказа, фамилию, сумму затрат по трем статьям расходов и общую сумму затрат по каждому заказу. Вычислить общую сумму затрат, остальные данные ввести с клавиатуры. Вывести на экран все данные ведомости.

10. Ведомость выработки по пяти бригадам за три месяца содержит наименование месяца, сумму выработки по бригадам за месяц, общую сумму

выработки за месяц. Вычислить общую сумму выработки, остальные данные ввести с клавиатуры. Вывести на экран все данные ведомости.

11. Ведомость расхода материалов на строительство содержит шесть наименований материалов, расход материала в рублях, допустимые нормы расхода. Эти данные ввести с клавиатуры. Вывести на экран наименование, расход и сумму расхода сверх нормы для материалов, по которым допущен перерасход.

12. Выработка статистических данных содержит информацию о фамилии, сумме доходов и количестве членов семьи и десяти работников предприятия. Эти данные ввести с клавиатуры. Вычислить сумму доходов, приходящуюся на одного члена семьи каждого работника. Вывести на экран все имеющиеся данные.

13. Расчетная ведомость содержит данные о фамилии, табельном номере, сумме зарплаты и подоходном налоге восьми работников отдела. Вычислить сумму налога, составляющего 13% от зарплаты, остальные данные ввести с клавиатуры. Вывести на экран все данные ведомости.

14. Ведомость расходов материалов на строительство содержит данные о наименовании материала, количестве израсходованного материала, стоимости единицы материала. Вычислить сумму расходов на закупку материала и общую сумму расходов. Вывести на экран все данные ведомости.

15. Расчетная ведомость бригады за три месяца содержит информацию о выработке бригады в рублях, суммарной заработной плате. Эти данные ввести с клавиатуры. Если выработка бригады превышает суммарную зарплату, то бригаде выплачивается премия в размере 10 % выработки. Вывести на экран информацию за июль, август, сентябрь о выработке бригады, суммарной заработной плате, премии (если премия не начисляется выводится 0) и общей сумме заработка.

Тема 6. Структура данных –файл

Краткие теоретические сведения

В Паскале понятие файла употребляется в двух смыслах:

Как поименованная информация на внешнем устройстве (внешний файл);

Как переменная файлового типа в Паскаль-программе (внутренний файл).

В программе между этими объектами устанавливается связь. Вследствие этого все, что происходит в процессе выполнения программы с внутренним файлом, дублируется во внешнем файле. С элементами файла можно выполнять только две операции: читать из файла и записывать в файл.

Файловый тип переменной – это структурированный тип, представляющий собой совокупность однотипных элементов, количество которых заранее (до использования программы) не определено.

Структура описания файловой переменной:

Var <имя переменной>: File of <тип элемента>;

Где <тип элемента> может быть любым, кроме файлового.

Например:

```
Var  Fi: File of  Integer;
     Fr: File of  Real;
     Fc: File of Char:
```

В Турбо Паскале предварительно определен следующий стандартный тип:

```
Type Text = File of Char;
```

Поэтому переменную Fc можно описать: Fc: Text;

Процедуры и функции для работы с файлами.

Assign(<имя файловой переменной>,<идентификатор внешнего файла>); устанавливает связь между файловой переменной и внешним файлом.

Reset(<имя файла>); процедура открытия существующего файла для чтения.

Rewrite(<имя файла>); - процедура открытия создаваемого файла для записи. Указатель файла устанавливается на первую запись.

Read(<имя файла>,<переменная>); - процедура чтения очередного компонента файла в переменную, тип которой должен совпадать с типом компонент файла. Указатель файла при этом передвигается на количество прочитанных компонент.

Write(<имя файла>,<переменная>); - процедура записи содержимого переменной в файл согласно положению указателя. Указатель автоматически сдвигается на число записанных компонент.

Seek(<имя файла>,<номер компоненты>);- процедура установки текущего указателя для чтения или записи требуемой компоненты файла. Используется для организации прямого доступа к записям файла.

Close(<имя файла>) – процедура закрытия файла. Обязательно должна использоваться после создания файла, иначе может произойти потеря данных.

Erase(<имя файла>) – процедура уничтожения файла. Открытый файл прежде должен быть закрыт.

Rename(<старое имя файла>, <новое имя файла>) - процедура для переименования файла. Используется после закрытия файла.

Filepos(<имя файла>) – функция определения номера текущей записи файла.

Filesize(<имя файла>) – функция определения общего количества записей файла.

Eof(<имя файла>) – функция определения признака конца файла. Получает значение TRUE при чтении последней записи файла.

Eoln(<имя файла>) – функция обнаружения конца строки в текстовом файле Имеет значение TRUE, если найден конец строки.

Особенности обработки типизированных файлов.

Типизированный файл состоит из последовательности записей одинаковой длины и одинакового внутреннего формата. Записи следуют непрерывно друг за другом. Первые 4 байта первого сектора файла содержат количество и длину записи. К файлам с такой организацией можно обращаться последовательно и выборочно (с прямым доступом).

При последовательном доступе записи располагаются на внешнем носителе последовательно в порядке их поступления, т.е. чтение или запись I+1 компоненты возможно только после I-ой компоненты.

При прямом доступе предполагается, что данные располагаются в определенных областях, имеющих последовательные номера, начиная с нуля. Вычисляя значение указателя, фиксирующего номер записи, можно обеспечить прямой доступ к нужной записи, используя процедуру позиционирования SEEK.

Пример 1. Дан символьный файл f. Получить копию файла в файле g.

```
Program FF;
F,g: Text;
C: Char;
Begin
Assign(f,'P1.dat');   Assign(g,'P2.dat');
Reset(f);  Rewrite(g);
While Not EOF(f) Do Begin
Read(f,c);   Write(g,c);   End;
Close(f);   Close(g);  Erase(f);   End.
```

Пример 2. Найти сумму элементов файла из целых чисел.

```
Program  P2;
Var F: File of Integer;
C,S: Integer;
Begin
Assign(F,'Fm.dat');   Reset(F);   S:=0;
While Not Eof(F) Do   Begin   Read(F,c);   S:=S+c;End;
Close(F);   Writeln(S);   End.
```

Пример 3. Текстовый файл t скопировать посимвольно в текстовый файл t1. Считываемый символ выдавать на экран в виде кода по ASCII.

```
Program P3;
Var t,t1: Text;
tname,t1name: String[30];
c:char;
Begin
Writeln('Введите имя файла с данными');
Readln(tname);
Writeln('Введите имя нового файла');
Readln(t1name);
```

```
Assign(t,tname); Assign(t1,t1name);
Reset(t);
Rewrite(t1);
While not eof(t) Do
Begin
Read(t,c);
Writeln(ord(c):4);
Write(t1,c)
End;Close(t)
Close(t1);
End.
```

```
Program zapis;
Type
Sved=record
Fam:string[25];
p1,p2,p3,p4:1..5;
sroc:Real;
End;
Var spisok: Array[1..30] of sved;
I,n: Integer;
champ: Real;
Begin
    Writeln(' Введите n');
    Readln(n);
    {Ввод сведений о студентах}
  For I:=1 To n Do
    With spisok[I] Do
             Begin
          Writeln('Введите фамилию студента ');
           Readln(Fam);
```

Окно вывода

```
Климова Л.Г.
 Введите оценки по 4 предметам
5 5 3 3
Введите фамилию студента
Мамаев Н.Р.
 Введите оценки по 4 предметам
3 4 5 3
Введите фамилию студента
Петрова А.П.
 Введите оценки по 4 предметам
5 5 4 4
              Герасименко О.Л.     4.500
                  Петрова А.П.     4.500
```

Контрольные вопросы

1. Понятие файла.
2. Типы файлов
3. Текстовые файлы
4. Процедуры и функции для работы с текстовыми файлами

Задания для самостоятельной работы

1. Записать в файл N действительных чисел. Вычислить произведение компонентов файла и вывести на печать.

2. Заполнить файл f целыми числами (признаком конца ввода пусть будет число 9999). Получить в файле g все компоненты файла f, которые делятся на m и не делятся на n.

3. Записать в файл N произвольных натуральных чисел. Переписать в другой файл те элементы, которые кратны K. Вывести полученный файл на печать.

4. Записать в файл N произвольных натуральных чисел. Переписать в другой файл те элементы, которые не кратны K. Вывести полученный файл на печать.

5. Записать в файл N произвольных натуральных чисел. Найти разность первого и последнего компонентов файла.

6. Записать в файл N произвольных натуральных чисел. Найти максимальный элемент этого файла.

7. Дан файл Bibl, содержащий сведения о книгах. Сведения о каждой из книг – это фамилия автора, название и год издания. Найти названия книг данного автора, изданных начиная с 1960 г.

8. Дан файл Bibl, содержащий сведения о книгах. Сведения о каждой из книг – это фамилия автора, название и год издания. Определить, имеется ли книга с названием «Информатика». Если да, то напечатать фамилию автора и год издания. Если таких книг несколько, то напечатать имеющиеся сведения обо всех этих книгах.

9. Багаж пассажира характеризуется количеством вещей и их общим весом. Дан файл Bagazh, содержащий сведения о багаже нескольких пассажиров. Сведения о багаже каждого пассажира представляют собой запись с двумя полями: одно поле целого типа (количество вещей) и одно – действительного (вес в килограммах). Найти багаж, в котором средний вес одной вещи наименьший.

10. Багаж пассажира характеризуется количеством вещей и их общим весом. Дан файл Bagazh, содержащий сведения о багаже нескольких пассажиров. Сведения о багаже каждого пассажира представляют собой запись с двумя полями: одно поле целого типа (количество вещей) и одно – действительного (вес в килограммах). Найти число пассажиров, имеющих более двух вещей.

11. Багаж пассажира характеризуется количеством вещей и их общим весом. Дан файл Bagazh, содержащий сведения о багаже нескольких пассажиров. Сведения о багаже каждого пассажира представляют собой запись с двумя полями: одно поле целого типа (количество вещей) и одно – действительного (вес в килограммах). Найти число пассажиров, багаж которых состоит из одной вещи весом не более m килограммов..

12. Заполнить файл f натуральными числами, полученными с помощью генератора случайных чисел. Найти количество четных чисел среди компонентов файла.

13. Текстовый файл t скопировать построчно и посимвольно в текстовый файл t1. Считываемый символ выдавать на экран в виде кода по ASCII.

14. Дан файл, содержащий текст, записанный строчными русскими буквами. Получить в другом файле тот же текст, записанный заглавными буквами. (Текстовый файл создать с помощью какого-либо текстового процессора).

15. Дан файл, содержащий произвольный текст. Выяснить, чего в нем больше: русских букв или цифр. (Текстовый файл создать с помощью какого-либо текстового процессора).

Тема 7. Линейные списки
Краткие теоретические сведения

Статистическими называются такие величины, память под которые выделяются во время компиляции и сохраняются в течение всей работы программы. Под динамические данные память отводится во время выполнения программы.

Использование динамических величин позволяет увеличить объем обрабатываемых данных, создавать структуры данных переменного размера. Если потребность в каких то данных отпала до окончания программы, то занятую ими память можно освободить для другой информации.

Для работы с динамическими структурами данных используются указатели. Указатели представляют собой специальный тип данных. Они принимают значения, равные адресам размещения в оперативной памяти соответствующих динамических переменных.

Списком называется структура данных, каждый элемент которой посредством указателя связывается со следующим элементом. На самый первый элемент (голову списка) имеется отдельный указатель.

Из определения следует, что каждый элемент списка содержит поле данных (оно может иметь сложную структуру) и поле ссылки на следующий элемент. После ссылки последнего элемента должно содержать пустой указатель (nil).

Число элементов связанного списка может расти или уменьшаться в зависимости от того, сколько данных мы хотим хранить в нем. Чтобы добавить новый элемент в список, необходимо:

1. Получить память для него;
2. Поместить туда информацию;
3. Добавить элемент в конец списка (или начало).

Элемент списка состоит из разнотипных частей (хранимая информация и указатель), и его естественно представить записью. Перед описанием самой записи описывают указатель на нее:

```
Type { описание списка из целых чисел }
  PList = ^TList;
  TList = record
    Inf : Integer;
    Next : PList;
  end;
```

Создание списка.

Задача. Сформировать список, содержащий целые числа 3, 5, 1, 9.

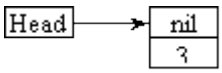
Определим запись типа TList с полями, содержащими характеристики данных – значения очередного элемента и адреса следующего за ним элемента

```
PList = ^TList;
  TList = record
    Data : Integer;
    Next : PList;
  end;
```

Чтобы список существовал, надо определить указатель на его начало. Опишем переменные.

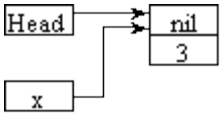
```
Var
  Head, x : PList;
```

Создадим первый элемент: New(Head); { выделяем место в памяти для переменной Head } Head^.Next := nil; { указатель на следующий элемент пуст (такого элемента нет) } Head^.Data := 3; { заполняем информационное поле первого элемента }

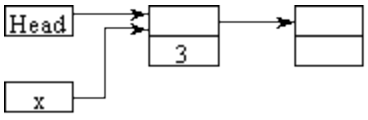


Продолжим формирование списка, для этого нужно добавить элемент в конец списка.

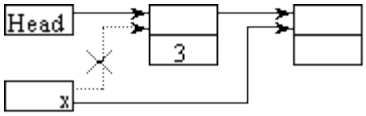
Введем вспомогательную переменную указательного типа, которая будет хранить адрес последнего элемента списка: x := Head; {сейчас последний элемент списка совпадает с его началом}



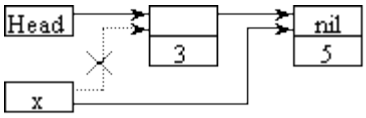
New(x^.Next); { выделим области памяти для следующего (2-го) элемента и поместим его адрес в адресную часть предыдущего (1-го) элемента }



x := x^.Next ; { переменная x принимает значение адреса выделенной области. Таким образом осуществляется переход к следующему (2-ому) элементу списка }



x^.Data := 5; { значение этого элемента } x^.Next := nil; {следующего значения нет }



Остальные числа заносятся аналогично: New(x^.Next); { выделим области памяти для следующего элемента } x := x^.Next ; { переход к следующему (3-му) элементу списка } x^.Data := 1; { значение этого элемента } x^.Next := nil; {следующего значения нет } New(x^.Next); { выделим области памяти для следующего элемента } x := x^.Next ; { переход к следующему (4-му) элементу списка } x^.Data := 9; { значение этого элемента } x^.Next := nil; {следующего значения нет }

Замечание. Как видно из примера, отличным является только создание первого (Head) элемента – головы списка. Все остальные действия полностью аналогичны и их естественно выполнять в цикле.

Присоединение нового элемента к голове списка производится аналогично: …….. New(x); { ввод значения элемента x^.Data := … } x^.Next := Head; Head := x; ……..

В этом случае последний введенный элемент окажется в списке первым, а первый – последним.

Просмотр списка.

Просмотр элементов списка осуществляется последовательно, начиная с его начала. Указатель List последовательно ссылается на первый, второй и т. д. элементы списка до тех пор, пока весь список не будет пройден. При этом с каждым элементом списка выполняется некоторая операция– например, печать элемента. Начальное значение List – адрес первого элемента списка (Head). Digit – значение удаляемого элемента.

```
List := Head;
While List^.Next <> nil do
  begin
    WriteLn(List^.Data);
    List := List^.Next; { переход к следующему элементу; аналог для массива i:=i+1 }
  end;
```

Удаление элемента из списка.

При удалении элемента из списка необходимо различать три случая:

1. Удаление элемента из начала списка.
2. Удаление элемента из середины списка.
3. Удаление из конца списка.

Удаление элемента из начала списка.

```
List := Head; { запомним адрес первого элемента списка }
Head := Head^.List; { теперь Head указывает на второй элемент списка }
Dispose(List); { освободим память, занятую переменной List^ }
```

Удаление элемента из середины списка.

Для этого нужно знать адреса удаляемого элемента и элемента, находящегося в списке перед ним.

```
List := Head;
While (List<>nil) and (List^.Data<>Digit) do
  begin
     x := List;
     List := List^.Next;
  end;
x^.Next := List^.Next;
Dispose(List);
```

Удаление из конца списка.

Оно производится, когда указатель x показывает на предпоследний элемент списка, а List – на последний.

```
List := Head; x := Head;
While List^.Next<>nil do
  begin
     x := List;
     List := List^.Next;
  end;
x^.Next := nil;
Dispose(List);
```

Контрольные вопросы

1. Что такое указатели? Какие значения они могут принимать? Какие операции возможны над указателями?
2. Что представляют собой динамические структуры данных? Для чего они используются? Чем отличаются от данных статического типа?
3. Какие стандартные процедуры существуют в языке Pascal для работы с указателями?
4. Какие операции требуется выполнить для вставки и удаления элемента списка?
5. Сколько элементов может содержать список?
6. Создание списка
7. Добавление элемента в список
8. Удаление элемента из списка
9. Поиск и извлечение элемента в списке

Задания для самостоятельной работы

1. Составить программу занесения в динамическую память вещественного массива из 10000 чисел, хранящихся в файле на магнитном

диске, а также поиска в нем значения и номера первого максимального элемента.

2. Связанный список представляет собой символьную цепочку из строчных латинских букв. Описать процедуру или функцию, которая определяет, является ли список пустым.

3. Написать программу, которая вставляет в список L новый элемент за каждым вхождением элемента E.

4. Составить программу, которая удаляет из списка L все элементы E, если таковые имеются.

5. Связанный список представляет собой символьную цепочку из строчных латинских букв. Описать процедуру или функцию, которая меняет местами первый и последний элементы непустого списка.

6. Описать процедуру, которая удаляет из непустого списка L первый элемент.

7. Описать процедуру, которая удаляет из непустого списка L последний элемент.

8. Описать процедуру, которая удаляет из непустого списка L первый неотрицательный элемент, если такой есть.

9. Описать процедуру, которая удаляет из непустого списка L второй элемент.

10. Написать программу. Заданный во входном файле текст (за ним следует точка) распечатать в обратном порядке.

11. Написать программу. Дана непустая последовательность натуральных чисел, за которой следует 0. Напечатать порядковые номера тех чисел последовательности, которые имеют наибольшую величину.

12. Дано целое n>0, за которым следует n вещественных чисел. Напечатать эти числа в порядке их убывания.

13. Описать процедуру или функцию, которая проверяет на равенство списки L1 и L2.

14. Описать процедуру или функцию, которая добавляет в конец списка L1 все элементы списка L2.

15. Описать процедуру или функцию, которая переносит в начало непустого списка L его последний элемент.

Тема 8. Структуры данных –стек и очередь
Краткие теоретические сведения

Стеком называется динамическая структура данных, добавление компоненты в которую и исключение компоненты из которой производится из одного конца, называемого ***вершиной стека***.

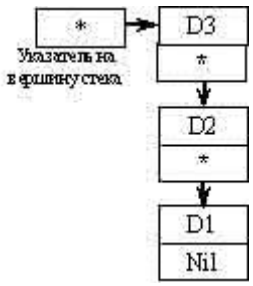


Рисунок 1. Схематичное изображение стека

В организации стека используется доступ по принципу "последней пошел, первый вышел". Такой метод доступа называют методом **LIFO (Last-In, First-Out)**. Представим себе стопку тарелок. Нижняя тарелка из этой стопки будет использована последней, а верхняя тарелка, которая была установлена в стопку последней, будет использована первой. Стеки широко используются в системном программном обеспечении, включая, компиляторы и интерпретаторы.

Обычно ***над стеками выполняются три операции***:

- начальное формирование стека (запись первой компоненты);
- добавление компоненты в стек;
- выборка компоненты (удаление).

Интуитивными моделями стека могут служить колода карт на столе при игре в покер, стопка тарелок на полке буфета; во всех этих моделях взять можно только верхний предмет, а добавить новый объект можно, только положив его на верхний. Стеки также иногда называют «магазинами» по аналогии с магазином патронов, в этом случае патрон, помещенный в магазин первым, выстреливает последним.

Со стеками обычно выполняются следующие действия:

Процедуры установки в стек и выборки из стека.

```
const
  MAX = 100;

var
  stack:array[1..100] of integer;
  tos:integer; {points to top of stask}

{ помещение объекта в стек }
procedure Push(i:integer);
begin
  if tos>=MAX then WriteLn('Stask full')
  else
  begin
    stack[tos]:=i;
    tos:=tos+1;
  end;
end; { конец  процедуры  помещения объекта в стек}
```

```
{ выборка объекта из стека }
function Pop:integer;

begin
  tos:=tos-1;
  if tos<1 then
  begin
    WriteLn('Stack underflow');
    tos:=tos+1;
    Pop:=0;
  end
  else Pop := stack[tos];
end; { конец функции выборки объекта из стека }
```

Пример стек

```
const
  MAX = 10;
var
  stack:array[1.. MAX] of integer;
  top: integer; x,y:integer; a:char;
  procedure Push(x:integer);
begin
  if top=MAX then WriteLn('Стек заполнен')
  else
  begin
    top:=top+1;
stack[top]:=x;
end;
end;
function Pop: integer;
begin
if top<1 then
WriteLn ('Стек пуст')

else
 begin
Pop := stack[top];
Top:=top-1;
end;
end;
begin
repeat
writeln('введите число');
readln(x);
Push(x);
writeln('повторить ввод?');
readln(a); {ввести y или n }
until a='n';
repeat
y:=pop;
writeln(y);
until top=0;
end.
```

Структура данных – очередь

Очередь — динамическая структура данных, у которой в каждый момент времени доступны только два элемента: первый и последний. Добавление элементов возможно только с одного конца (конца очереди), а удаление элементов – только с другого конца (начала очереди).

Существует сокращение для очереди: **FIFO** = **F**irst **I**n – **F**irst **O**ut, с английского — «Кто первым вошел, тот первым вышел».

Для очереди доступны **следующие операции:**

- добавить элемент в конец очереди (PushTail);
- удалить элемент с начала очереди (Pop).

Работа с очередью обычным массивом:

Это достаточно простой способ, который подразумевает два неблагоприятных момента: заблаговременное выделение массива, сдвиг элементов при удалении из очереди.

Программа для добавления и удаления из очереди, а также перемены местами данных в двух указанных местах.

```
Program Ochered;
{$APPTYPE CONSOLE}
uses
SysUtils;
Type
Ukaz = ^Stack;
Stack = record
Inf: integer;
Next: ukaz
End;
var Pp, Kon, newel, Right, Left: ukaz; num, value, max, i, Pervoe, Vtoroe: integer;
Procedure Sozd; {Организация очереди}
Begin
New(Kon);
Kon^.next := Left;
Kon^.inf := value;
Right := Kon;
Left := Kon;
End;
Procedure Dob; {Добавление новых элементов в уже существующую очередь.
Добавление происходит справа}
Begin
New (newel);
Right^.Next := Newel;
Newel^.next := nil;
Newel^.inf := value;
Right := newel;
```

```
End;
Procedure Udal; {Удаление элемента из очереди, удаление происходит слева}
Begin
Left:= Left^.next;
End;
Procedure writ; {Печать элементов очереди}
Begin
Pp := Left;
While Pp <> nil do
Begin
Writeln(Pp^.inf);
Pp := Pp^.next;
End;
End;
Begin
write('Vsego elementov ocheredi: ');
readln(max);//Вводим общее количество элементов очереди
write('Vvedite znachenie: ');
readln(value);
sozd; //создаем очередь
for i := 1 to max-1 do begin//вводим каждое значение
if max=1 then break;
write('Vvedite znachenie: ');
readln(value);
dob; //добавляем новые элементы
end;
write('Pervoe: '); //вводим число которое надо поменять местами
readln(Pervoe);
write('Vtoroe: '); //вводим второе число которое надо поменять местами
readln(Vtoroe);
Pp := Left;
While Pp <> nil do
Begin
if Pp^.inf = Pervoe then begin
Pp^.inf := Vtoroe;
Pp := Pp^.next;
continue;
end;
if Pp^.inf = Vtoroe then begin
Pp^.inf := Pervoe;
Pp := Pp^.next;
continue;
end;
```

```
Pp := Pp^.next;
End;
writ;
End.
```

Контрольные вопросы

1. В чем сходство и отличие динамических структур данных типа список и стек?
2. Можно ли добраться до середины или конца («дна») стека, минуя его начало («вершину»)?
3. Приведите примеры из жизни, где встречается «принцип стека».
4. Оформите в виде процедуры Push занесение в стек значения.
5. Оформите в виде процедуры Pop извлечение значения из стека.

Задания для самостоятельной работы

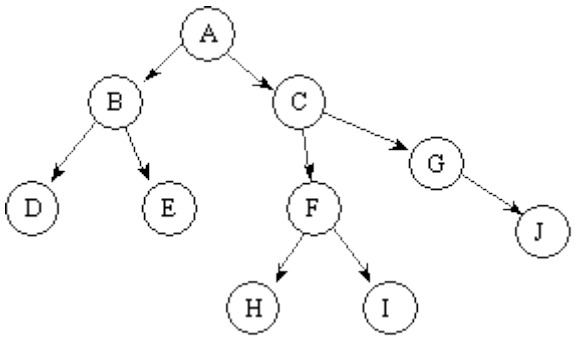
1. Подсчитать количество элементов в стеке.
2. Сформировать стек, содержащий строки и сохранить его в текстовом файле.
3. Восстановить стек, содержащий строки, из текстового файла.
4. Написать функцию, которая вычисляет среднее арифметическое элементов стека.
5. Написать процедуру присоединения стека S2 к стеку S1.
6. Определить симметричность произвольного текста любой длины. Текст должен оканчиваться точкой. Задачу решить с помощью двух стеков.
7. Слить два стека, содержащих возрастающую последовательность целых положительных чисел, в третий стек так, чтобы его элементы располагались также в порядке возрастания.
8. В данном тексте проверить соответствие открытия и закрытия скобок.
9. Напечатать содержимое текстового файла, выписывая символы каждой его строки в обратном порядке.
10. Проверить, является ли строка палиндромом.

Тема 9. Двоичные деревья

Краткие теоретические сведения

Бинарное дерево - это иерархическая динамическая структура данных, состоящая из узлов и соединяющих их дуг. В каждый узел, кроме одного, ведет ровно одна дуга. Этот единственный узел называется корнем дерева.

С каждой вершиной дерева связывается конечное число отдельных деревьев, называемых поддеревьями. На рисунке изображено дерево, в узлах которого располагаются символы:



Вершина Y, находящаяся непосредственно ниже вершины X, называется непосредственным потомком X, а вершина X называется предком Y. Если вершина не имеет потомков, то она называется листом, если имеет, то называется внутренней вершиной. Например, вершины D, E являются непосредственными потомками вершины B; вершины H, I, J являются листьями. Очевидно, что для описания дерева требуются ссылки. Опишем бинарное дерево как структуру типа Record, содержащую как минимум два поля – указатели на левое и правое поддеревья, и поле данных, например типа Integer:

```
Type
  PTree = ^TTree;
  TTree = Record
    Data : Integer;
    Left, Right : PTree;
  end;
```

Корень дерева описывается ссылочной переменной:

```
Var
  Tree : PTree;
```

Основные операции над деревом

К основным операциям над деревьями относятся:

• занесение элемента в дерево;
• обход дерева;
• удаление элемента из дерева.

Примеры

Вставка элемента в дерево.

Создать и вывести на экран дерево, элементы которого вводятся с клавиатуры и имеют целый тип. Причем для каждой вершины дерева во всех левых вершинах должны находиться числа меньшие, а в правой большие, чем числа, хранящиеся в этой вершине. Такое дерево называется деревом поиска.

Опишем процедуру вставки в дерево новой вершины. При вставке в дерево вершина вставляется либо как поддерево уже существующей вершины или как единственная вершина дерева. Поэтому и левая, и правая связи новой вершины должны быть равны nil. Когда дерево пусто, значение передаваемой в виде параметра ссылки равно nil. В этом случае нужно изменить ее так, чтобы она указывала на новую вершину, которая была

вставлена как корневая. При вставке второго элемента переданный из основной программы параметр ANode уже не будет равен nil, и надо принимать решение о том, в какое поддерево необходимо вставить новую вершину.

```
Procedure InsTree(var ANode : PTree; n : lnteger);
Begin
  if ANode = nil then
    Begin
      new(ANode);
      With ANode^ do
        Begin
          Left := nil;
          Right := nil;
          Data := n;
        end;
    end
  else if n< ANode^.Data then InsTree(ANode^.Left, n) else
InsTree(ANode^.Right, n);
End;
```

Вывод элементов дерева.

Опишем процедуру вывода значений элементов двоичного дерева на экран. Для этого необходимо выполнить полный обход дерева. При обходе дерева его отдельные вершины посещаются в определенном порядке. Вывод двоичного дерева можно производить рекурсивно, выполняя для каждой вершины три действия:

- вывод числа, хранящегося в узле;
- обход левого поддерева;
- обход правого поддерева.

Порядок выполнения этих действий определяет способ обхода дерева. Способы обхода:

- прямой обход (сверху вниз);
- симметричный обход (слева направо);
- обратный обход (снизу вверх).

Процедура симметричного вывода дерева имеет следующий вид:

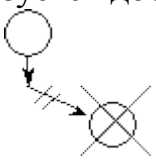
```
Procedure PrintTree(ANode : PTree);
Begin
  if ANode <> nil then
    Begin
      PrintTree(ANode^.Left);
      WriteLn(ANode^.Data);
      PrintTree(ANode^.Right)
    End;
End;
```

Основная программа осуществляет ввод чисел с клавиатуры. Используются: переменная Tree типа PTree – значение указателя на корень дерева; переменная Digit типа Integer для хранения очередного введенного числа.

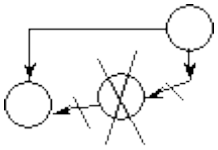
```
Var
  Tree : PTree;
  Digit : Integer;
BEGIN                    {основная программа}
  Writeln('Окончание ввода – 0');
  Tree := nil;
  Read(Digit);
  While Digit<>0 Do
    Begin
      InsTree(Tree,Digit);
      Write('Введите очередное число: ');
      ReadLn(Digit);
    End;
  PrintTree(Tree);
END.
```

Удаление из дерева.

Непосредственное удаление элемента из упорядоченного дерева реализуется достаточно просто, если эта вершина является конечной:



или из нее выходит только одно ребро:



Для этого нужно изменить соответствующую ссылку у предшествующей вершины. Если же из удаляемой вершины выходит две ветви, то нужно найти подходящую вершину дерева, которую можно было бы вставить на место удаляемой вершины. В этом случае удаляемый элемент нужно заменить либо на самый правый элемент его левого поддерева, либо на самый левый элемент его правого поддерева. Такие элементы не могут иметь более одного потомка.

Пусть задана следующая структура:

```
Type
  PTree = ^TTree;
  TTree = Record
    Data : Integer;
    Left, Right : PTree;
  end;
```

```
Var
  Tree : PTree;
```

Необходимо удалить из дерева Tree узел со значением поля Data=x.

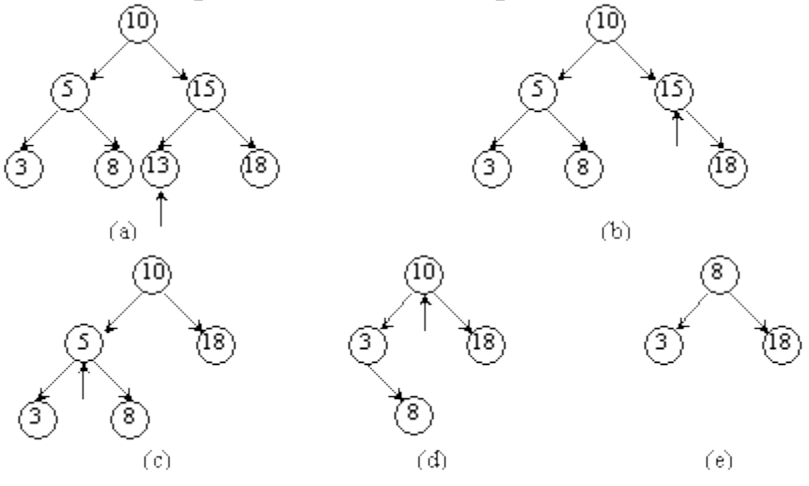
В процедуре DeleteNode различаются три случая:

1. Узла со значение, равным x, нет.
2. Узел со значением x имеет не более одного потомка.
3. Узел со значением x имеет двух потомков.

```
Procedure DeleteNode(x : Integer; var ANode : PTree);
Var q : PTree;
Procedure Del(var R : PTree);
Begin
  if R^.Right <> nil then Del(R.^Right)
  else
     begin
       q^.Data := R^.Data;
       q := R;
       R := R^.Left;
       Dispose(q);
     end;
End; { Del }
Begin  { DeleteTree }
  if ANode = nil then Writeln('Элемента с ключом ',x,' в дереве нет')
  else
     if x < ANode^.Data then DeleteNode(x,ANode^.Left)
     else if  x > ANode^.Data then DeleteNode(x,ANode^.Right)
       else
          begin
            q := ANode;
            if q^.Right = nil then
              Begin
                 ANode := q^.Left;
                 Dispose(q);
              end
            else
              if q^.Left = nil then
                 begin
                   ANode := q^.Right;
                   Dispose(q);
                 end
              else Del(q^.Left);
          end;
End;
```

Вспомогательная процедура Del вызывается только в третьем случае. Она «спускается» вдоль самой правой ветви левого поддерева удаляемого узла q^

и затем заменяет данные (поле Data) в q^ соответствующими значениями самого правого узла R^ этого левого поддерева, после чего от R^ можно освободиться. На рисунке задано начальное дерево (a), из которого последовательно удаляются узлы со значениями 13, 15, 5, 10. Полученные деревья показаны на рис. b –



e.

Контрольные вопросы

1. Каков принцип построения динамической структуры «дерево»?
2. Перечислите сходства и отличия динамических структур типа «линейный список», «стек», «дерево».
3. Перечислите структуры, которые можно представить в виде дерева, которые встречаются в повседневной жизни.
4. Закончите фразу: «Список – это дерево, в котором …».

Задания для самостоятельной работы

Во всех задачах подразумеваются деревья двоичного поиска.

1. Написать рекурсивную числовую функцию, подсчитывающую сумму элементов дерева.
2. Написать функцию, которая находит наибольший элемент дерева.
3. Написать функцию, которая находит наименьший элемент дерева.
4. Напишите процедуру, которая удаляет из дерева все четные элементы.
5. Написать рекурсивную процедуру, которая определяет число вхождений заданного элемента в дерево.
6. Написать рекурсивную процедуру, которая печатает элементы из всех листьев дерева.
7. Написать рекурсивную функцию, которая определяет глубину заданного элемента на дереве и возвращает –1, если такого элемента нет.
8. Написать процедуру, которая печатает (по одному разу) все вершины дерева.
9. Написать процедуру, которая по заданному n считает число всех вершин глубины n в заданном дереве.
10. Написать процедуру, которая считает глубину дерева.
11. Отсортировать массив A путем включения его элементов в дерево и скопировать отсортированные данные обратно в A.

12. Задана последовательность слов. Определить частоту вхождения каждого из слов в последовательность.

Указание. Для решения задачи любое слово ищется в дереве, которое на начальном этапе пусто. Если слово найдено, то счетчик его вхождений увеличивается на единицу, если нет, то слово включается в дерево с единичным значением счетчика.

Тема 10. Алгоритмы сортировки

Краткие теоретические сведения

При работе с массивами данных не редко возникает задача их **сортировки по возрастанию или убыванию, т.е. упорядочивания.** Это значит, что элементы того же нужно расположить строго по порядку. Например, в случае сортировки по возрастанию предшествующий элемент должен быть меньше последующего (или равен ему).

Существует множество методов сортировки. Одни из них являются более эффективными, другие – проще для понимания.

Самые популярные методы сортировки:

- Сортировка методом прямого обмена (пузырьковая).
- Сортировка методом прямого включения.
- Сортировка методом прямого выбора.
- Сортировка с помощью дерева.

Алгоритм сортировки прямым обменом (пузырьковая сортировка):

Как известно воздух легче воды, поэтому пузырьки воздуха всплывают. Это просто аналогия. В сортировке методом пузырька по возрастанию более легкие (с меньшим значением) элементы постепенно "всплывают" в начало массива, а более тяжелые друг за другом опускаются на дно (в конец массива).

Алгоритм сортировки:

1. При первом проходе по массиву элементы попарно сравниваются между собой: первый со вторым, затем второй с третьим, следом третий с четвертым и т.д. Если предшествующий элемент оказывается больше последующего, то их меняют местами.

2. Не трудно догадаться, что постепенно самое большое число оказывается последним. Остальная часть массива остается не отсортированной, хотя некоторое перемещение элементов с меньшим значением в начало массива наблюдается.

3. При втором проходе незачем сравнивать последний элемент с предпоследним. Последний элемент уже стоит на своем месте. Значит, число сравнений будет на одно меньше.

4. На третьем проходе уже не надо сравнивать предпоследний и третий элемент с конца. Поэтому число сравнений будет на два меньше, чем при первом проходе.

5. В конце концов, при проходе по массиву, когда остаются только два элемента, которые надо сравнить, выполняется только одно сравнение.

6. После этого первый элемент не с чем сравнивать, и, следовательно, последний проход по массиву не нужен. Другими словами, количество проходов по массиву равно m-1, где m – это количество элементов массива.

7. Количество сравнений в каждом проходе равно m-i, где i – это номер прохода по массиву (первый, второй, третий и т.д.).

8. При обмене элементов массива обычно используется "буферная" (третья) переменная, куда временно помещается значение одного из элементов.

Программа на языке Паскаль:

```
const
   m = 10;
 var
arr: array[1..m] of integer;
i, j, k: integer;
 begin
randomize;
    write ('Исходныймассив: ');
fori := 1 to m do begin
arr[i] := random(256);
write (arr[i]:4);
end;
writeln; writeln;
     fori := 1 to m-1 do
for j := 1 to m-i do
ifarr[j] >arr[j+1] then begin
k := arr[j];
arr[j] := arr[j+1];
arr[j+1] := k
end;
    write ('Отсортированныймассив: ');
fori := 1 to m do
write (arr[i]:4);
    writeln;
 readln
end.
```

Алгоритм сортировки прямым включением

Такой метод широко используется при игре в карты.

Элементы мысленно делятся на уже “готовую” последовательность a_1, … , a_{i-1} и исходную последовательность. При каждом шаге, начиная с I = 2 и увеличивая i каждый раз на единицу,

изисходнойпоследовательностиизвлекаетсяiйэлементыиперекладывается в г отовую последовательность, при этом он вставляется на нужное место.

В реальном процессе поиска подходящего места удобно, чередуя сравнения и движения по последовательности, как бы просеивать x , т. е. x сравнивается с очередным элементом a_j , а затем либо x вставляется на свободное место, либо a_j сдвигается вправо и процесс "уходит" влево. Обратите внимание, что процесс просеивания может закончиться при выполнении одного из двух следующих различных условий:

1.Найден элемент a_j с ключом, меньшим чем ключ у x.

2.Достигнут левый конец готовой последовательности

Программа сортировки с помощью прямого включения.

```
PROGRAM SI;
VAR
 I,J,N,X:INTEGER;
 A:ARRAY[0..50] OF INTEGER;
BEGIN
 WRITELN('Введите длину массива');
 READ(N);
 WRITELN('Введите массив');
 FOR I:=1 TO N DO READ(A[I]);
 FOR I:=2 TO N DO BEGIN
  X:=A[I];
  A[0]:=X;
  J:=I;
  WHILE X<A[J-1] DO BEGIN
   A[J]:=A[J-1];
   DEC(J)
  END;
  A[J]:=X
 END;
 WRITELN('Результат:');
 FOR I:=1 TO N DO WRITE(A[I],' ')
END.
```

Такой типичный случай повторяющегося процесса с двумя условиями окончания позволяет нам воспользоваться хорошо известным приемом "барьера" (sentinel).

Сортировка методом вставки

Сортировка вставками — очень простой метод сортировки, при котором элементы данных используются как ключи для сравнения. Этот алгоритм сначала упорядочивает A[0] и A[1], вставляя A[1] перед A[0], если A[0] > A[1]. Затем оставшиеся элементы данных по очереди вставляются в этот упорядоченный список. После k-й итерации элемент A[k] оказывается в своей правильной позиции и элементы от A[0] до A[k] уже отсортированы.

Контрольные вопросы

Понятие сортировки .

Алгоритмы сортировки

Алгоритм сортировки прямым обменом (пузырьковая сортировка)

Алгоритм сортировки прямым включением

Алгоритм сортировки вставками

Задания для самостоятельной работы

1. Дан ряд, содержащий n элементов. Отсортировать их в порядке возрастания, отбрасывая при этом все повторяющиеся элементы.

2. Определить моду данного ряда – значение, встречающееся среди его элементов чаще всего, предварительно отсортировав его.

3. Исходный набор данных представляет собой последовательность записей, состоящих из фамилии, возраста и стажа работы. Распечатать этот список: 1) в алфавитном порядке; 2) в порядке увеличения возраста; 3) в порядке увеличения стажа работы.

4. Написать процедуру сортировки методом пузырька по убыванию.

5. Дан ряд целых чисел. Получить в порядке возрастания все различные числа, входящие в этот ряд.

6. Дан ряд из n различных целых чисел. Получить различные целые числа $i_1, i_2, \ldots, i_n$ такие, что $a_{i_1} < a_{i_2} < \ldots < a_{i_n}$.

7. Даны целые $a_1, a_2, \ldots, a_n$. Найти наибольшее значение в этой последовательности после выбрасывания из нее всех членов со значением $\max(a_1, a_2, \ldots, a_n)$

8. Дан двумерный массив чисел A(5,4). Отсортировать строки массива в порядке возрастания.

9. Дан двумерный массив чисел A(4,4). Отсортировать по столбцам элементы массива в порядке возрастания.

10. Заполнить двумерный массив X(5,4) целыми случайными числами в диапазоне от -15 до +15. Отсортировать в порядке убывания элементы 2 столбца.

11. Дан двумерный массив числаA(4,4). Отсортировать в порядке возрастания элементы 4 строки.

12. Дан двумерный массивA(3,3). Отсортировать в порядке убывания элементы 4 столбца.

13. Дан двумерный массивA(4,4). Изменить массив таким образом, чтобы элементы главной диагонали были упорядочены в порядке возрастания.

14. Дан двумерный массив 77. Изменить массив таким образом, чтобы элементы побочной диагонали были упорядочены в порядке возрастания.

15. Дан двумерный массив 78. Отсортировать в порядке возрастания элементы последней строки, а затем – последнего столбца.

16. Дан двумерный массив чисел 87. Отсортировать в порядке убывания отрицательные элементы последней строки.

17. Дан двумерный массив чисел 78. Отсортировать в порядке возрастания положительные элементы последнего столбца.

Тема 11. Алгоритмы поиска
Краткие теоретические сведения

Поиск – процесс нахождения конкретного элемента в заданном множестве данных.

Поиск является одним из наиболее часто встречаемых действий в программировании. Существует множество различных алгоритмов поиска, которые принципиально зависят от способа организации данных.

Последовательный (линейный) поиск – это простейший вид поиска заданного элемента на некотором множестве, осуществляемый путем последовательного сравнения очередного рассматриваемого значения с искомым до тех пор, пока эти значения не совпадут.

Идея этого метода заключается в следующем. Множество элементов просматривается последовательно в некотором порядке, гарантирующем, что будут просмотрены все элементы *множества* (например, слева направо). Если в ходе просмотра *множества* будет найден искомый элемент, просмотр прекращается с положительным результатом; если же будет просмотрено все множество, а элемент не будет найден, *алгоритм* должен выдать отрицательный результат.

Алгоритм последовательного поиска

Пусть дан массив элементов A и ключ b. Для b необходимо найти совпадающий с ним элемент массива A. Линейный поиск подразумевает последовательный просмотр всех элементов массива A в порядке их расположения, пока не найдется элемент равный b. Если заранее неизвестно, имеется данный элемент в массиве или нет, то необходимо следить за тем, чтобы поиск не вышел за границы массива, что достигается использованием стоппера (барьерного элемента). Рассмотрим алгоритм последовательного поиска на языке Паскаль:

```
const N=10;
type Any=integer;
var A:array[0..N-1] of Any;
    i:integer;
    b:Any;
begin
  writeln('Введитеэлементымассива:');
   for i:=0 to N-1 do
    readln(A[i]);
  write('Введитеключ:');
  readln(b);
  i:=0;
   while (i<>N) and (A[i]<>b) do
```

```
    i:=i+1;
  ifi<>N then
   writeln('Элемент, совпадающих с ключом, найден. Позиция элемента -',i+1)
else
writeln('Элементов, совпадающих с ключом, нет');
end.
```

2) с использованием стоппера

```
const N=11;
type Any=integer;
var A:array[0..N] of Any;
    i:integer;
    b:Any;
begin
  writeln('Введитеэлементымассива:');
   for i:=0 to N-1 do
    readln(A[i]);
  write('Введитеключ:');
  readln(b);
   A[N]:=b; {стоппер}
  i:=0;
    while A[i]<>b do
      i:=i+1;
  ifi<>N then
   writeln('Элемент, совпадающих с ключом, найден. Позиция элемента -',i+1)
else
   writeln('Элементов, совпадающих с ключом, нет');
end.
```

Следует отметить, что в общем случае при линейном поиске среди N элементов требуется в среднем N/2 сравнений в случае успешного поиска и N сравнений в наихудшем случае, и затраты времени для больших массивов поэтому велики.

Применяется данный алгоритм, если никакой дополнительной информации о данных массива нет.

Реализация алгоритма дихотомии (бинарный поиск)

Бинарный поиск

Очевидно, что нельзя каким-либо способом ускорить поиск, если отсутствует информация о данных, среди которых производится поиск. Также известно, что если данные упорядочены, то поиск можно сделать значительно эффективнее. Поэтому рассмотрим алгоритм, который основан на том, что массив A упорядочен (т. е. a[i-1]<=a[i] при 1<=i<N).

Суть бинарного поиска: берут средний элемент массива и сравнивают его с ключом. В результате возможны 3 случая:

1) если элемент массива равен ключу, то искомый элемент найден;

2)если элемент массива меньше ключа, то все элементы массива с индексами, которые меньше N/2 исключаются из рассмотрения;

3)если элемент массива больше ключа, то все элементы массива с индексами, которые больше N/2 исключаются из рассмотрения;

Затем поиск продолжается в одной из половин массива аналогичным образом.

```
const N=10;
type Any=integer;
var A:array[0..N] of Any;
   Left,Right,m,i,j:integer;
   b:Any;
begin
 writeln('Введите упорядоченную последовательность эл-тов массива ');
for i:=0 to N do readln(A[i]);
writeln('Введитеключ');
readln(b);
Left:=0;Right:=N;
while Left<Right do
begin
m:=(Left+Right) div 2;
if A[m]<b then Left:=m+1
else Right:=m;
end;
writeln('Эл-т найден. Позиция эл-та: ',Right)
else
writeln('Элемента нет');
end.
```

Нахождение элемента бинарным поиском осуществляется очень быстро. При поиске среди N элементов требуется log2(N) сравнений в наихудшем случае. Кроме того, бинарный поиск уже при N порядка 100 значительно эффективнее линейного - как по скорости обнаружения, так и по способности к получению отрицательного результата. Для доказательства этого приведем следующие характеристики линейного и бинарного поиска:

Контрольные вопросы

1. Алгоритмы поиска.
2. Последовательный (линейный) поиск
3. Метод бинарного поиска (дихотомии

Задания для самостоятельной работы

1. В целочисленном массиве B (20) поменять местами наибольший и наименьший элементы.
2. Дан действительный массив A (15). Найти наименьший элемент из положительных элементов массива.
3. В действительном массиве B (20) найти наибольший элемент из отрицательных элементов.

4. В массиве В (20) найти максимальный из элементов с четными номерами.

5. Дан массив В (15) из действительных чисел. Найти среднее арифметическое максимального и минимального элементов массива.

Тема 12. Оценка сложности алгоритма

Краткие теоретические сведения

Алгоритм должен удовлетворять следующим требованиям:

1)быть простым для понимания, перевода в программный код и отладки;

2)эффективноиспользоватьвычислительныересурсыивыполнятьсяповозможности быстро.

Сложность алгоритма – это величина, отражающая порядок величины требуемого ресурса (времени или дополнительной памяти) в зависимости от размерности задачи.

Сложность алгоритма:

- Пространственная сложность V(n).
- Временная сложность T(n).

Теоретическая оценка сложности алгоритма

1. Если операция выполняется а фиксированное число шагов, не зависящее от количества данных, то принято писать O(1).

2. Время выполнения операций присваивания, чтения, записи обычно имеют порядок O(1).

3. Время выполнения операций в данной последовательности совпадает с наибольшим временем выполнения операции в последовательности (правило сумм: если T1(n) имеет порядок O(f(n)), а T2(n) - порядок O(g(n)), то T1(n)+T2(n) имеет порядок O(max(f(n),(g(n))).

4. Время выполнения конструкции ветвления (if-then-else) состоит из времени вычисления логического выражения (обычно имеет порядок O(1)) и на и большего из времени, необходимого для выполнении сполняемых при истинном значении логического выражения и при ложном значении логического выражения.

5. Время выполнения цикла состоит из времени вычисления прекращения цикла(обычно имеет порядок O(1)) и произведения количества выполняемых итераций цикла на наибольшее возможное время выполнения операций тела цикла.

6. При наличии в алгоритме операции безусловного перехода, необходимо учитывать изменения последовательности операций, осуществляемых с использованием этих операций безусловного перехода.

О-символика

Пусть даны две функции f(n) и g(n) натурального аргумента n, значениями которых

являются действительные числа.

Говорят, что f=O(g) («f растет не быстрее g»), если существует такая константа c>0,

что f(n)≤c·g(n) для всех n.

Вычисление сложности алгоритма метода Гаусса

Системы линейных уравнений возникают при решении многих прикладных задач. Матрицы коэффициентов систем линейных уравнений могут иметь различные структуру и свойства. Мы будем полагать, что решаемая система имеет плотную матрицу высокого порядка. Линейная система n уравнений с n неизвестными x_0, x_1, …, x_{n-1} может быть представлена в виде:

$$a_{0,0}x_0+a_{0,1}x_1+\cdots+a_{0,n-1}x_{n-1}=b_0$$

$$a_{0,1}x_0+a_{1,1}x_1+\cdots+a_{1,n-1}x_{n-1}=b_1$$

$$\cdots$$

$$a_{n-1,0}x_0+a_{n-1,1}x_1+\cdots+a_{n-1,n-1}x_{n-1}=b_{n-1}$$

В более кратком (матричном) виде система имеет вид

$$A\cdot x=b$$

где A=a(i,j) есть вещественная матрица размера n×n, а вектора b и x состоят из n элементов.

Под задачей решения системы линейных уравнений для заданных матрицы A и вектора b обычно понимается нахождение значения вектора неизвестных x, при котором выполняются все уравнения системы.

Метод Гаусса является широко известным прямым алгоритмом решения систем линейных уравнений, для которых матрицы коэффициентов являются плотными. Если система линейных уравнений является невырожденной, то метод Гаусса гарантирует нахождение решения с погрешностью, определяемой точностью машинных вычислений. Основная идея метода состоит в приведении матрицы A посредством эквивалентных преобразований (не меняющих решение системы) к треугольному виду, после чего значения искомых неизвестных может быть получено непосредственно в явном виде.

Изучение алгоритма Гаусса решения систем линейных уравнений

Метод Гаусса основывается на возможности выполнения преобразований линейных уравнений, которые не меняют при этом решение рассматриваемой системы (такие преобразования носят наименование эквивалентных). К числу таких преобразований относятся:

Умножение любого из уравнений на ненулевую константу

Перестановка уравнений

Прибавление к уравнению любого другого уравнения системы

Метод Гаусса включает последовательное выполнение двух этапов. На первом этапе – прямой ход метода Гаусса – исходная система линейных

уравнений при помощи последовательного исключения неизвестных приводится к верхнему треугольному виду

$$U \cdot x = b$$

где матрица коэффициентов получаемой системы имеет вид

$$U = \begin{pmatrix} u_{0,0} & u_{0,1} & \dots & u_{0,n-1} \\ 0 & u_{1,1} & \dots & u_{1,n-1} \\ & \dots & & \\ 0 & 0 & \dots & u_{n-1,n-1} \end{pmatrix}$$

На обратном ходе метода Гаусса (второй этап алгоритма) осуществляется определение значений неизвестных. Из последнего уравнения преобразованной системы может быть вычислено значение переменной xn-1, после этого из предпоследнего уравнения становится возможным определение переменной xn-2 и т.д.

Прямой ход метода Гаусса

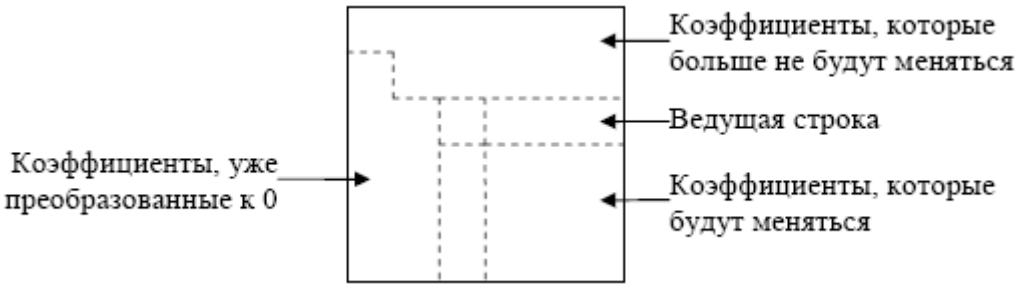
Прямой ход метода Гаусса состоит в последовательном исключении неизвестных в уравнениях решаемой системы линейных уравнений. На итерации i метода производится исключение неизвестной i для всех уравнений с номерами k, большими i. Для этого из этих уравнений осуществляется вычитание строки i, умноженной на константу (aki/aii), с тем, чтобы результирующий коэффициент при неизвестной xi в строках оказался нулевым. Вычисления, выполняемые над элементами матрицы A и вектора b, определяются следующими соотношениями.

$$a'_{kj} = a_{kj} - (a_{kj}/a_{ii}) \cdot a_{ij}$$

$$b'_k = b_k - (a_{kj}/a_{ii}) \cdot b_i$$

$$i \le j \le n-1, \qquad i < k \le \tilde{n} - 1, \qquad 0 \le i < n-1.$$

На рисунке представлена общая схема состояния данных на i-ой итерации прямого хода алгоритма Гаусса. Все коэффициенты при неизвестных, расположенные ниже главной диагонали и левее столбца i, уже являются нулевыми. На i-ой итерации прямого хода метода Гаусса осуществляется обнуление коэффициентов столбца i, расположенных ниже главной диагонали, путем вычитания строки i, умноженной на нужную ненулевую константу. После проведения (n-1) подобной итерации матрица, определяющая систему линейных уравнений, становится приведенной к верхнему треугольному виду.



При выполнении прямого хода метода Гаусса строка, которая используется для исключения неизвестных, носит наименование ведущей, а диагональный элемент ведущей строки называется ведущим элементом. Как можно заметить, выполнение вычислений является возможным только, если ведущий элемент имеет ненулевое значение. Более того, если ведущий элемент a(i,i) имеет малое значение, то деление и умножение строк на этот элемент может приводить к накоплению вычислительной погрешности и вычислительной неустойчивости алгоритма.

Возможный способ избежать подобной проблемы может состоять в следующем – при выполнении каждой очередной итерации прямого хода метода Гаусса следует определить коэффициент с максимальным значением по абсолютной величине в столбце, соответствующем исключаемой неизвестной, т.е.

$$y = \max_{i \le k \le n-1} |a_{ki}|$$

и выбрать в качестве ведущей строку, в которой этот коэффициент располагается (данная схема выбора ведущего значения носит наименование метода главных элементов).

Обратный ход алгоритма Гаусса

После приведения матрицы коэффициентов к верхнему треугольному виду становится возможным определение значений неизвестных. Из последнего уравнения преобразованной системы может быть вычислено значение переменной xn-1, после этого из предпоследнего уравнения становится возможным определение переменной xn-2 и т.д. В общем виде, выполняемые вычисления при обратном ходе метода Гаусса могут быть представлены при помощи соотношений:

$$x_{n-1} = b_{n-1}/a_{n-1,n-1}$$

$$x_i = (b_i - \sum_{j=i+1}^{n-1} a_{ij}x_j)/a_{ii} \quad i = n-2 \ldots 0$$

Определим вычислительную сложность метода Гаусса. В прямом ходе алгоритма для выбора ведущей строки на каждой итерации должно быть определено максимальное значение в столбце с исключаемой неизвестной. По мере исключения неизвестных количество строк и элементов в строках сокращается.

Текущее число элементов строки (включая правую часть), с которыми производятся действия сложения и вычитания (первый элемент без

вычислений полагается равным нулю), равно (n-i), где i, – номер итерации прямого хода. Поскольку кроме выполнения двух операций (умножения и вычитания) с каждым элементом строки предварительно должен быть вычислен масштабирующий коэффициент aik/aii, общие затраты на выполнение действий в одной строке составят 2(n-i)+1 операций.

С учетом того, что на каждой итерации обрабатывается n-i строк, общее число операций в прямом ходе метода Гаусса определяется выражением

$$T' = \sum_{i=0}^{n-2} 2(n-i)^2 + n - i$$

Для реализации обратного хода на каждой i-й итерации,(итерация для удобства присваиваем номера от n-2 до 0) необходимо произвести n-i-1 умножений, столько же вычитаний, а также одно деление для определения очередной неизвестной. Следовательно, общая вычислительная сложность обратного хода составит

$$T'' = \sum_{i=0}^{n-2} (2(n-i-1)+1) = \sum_{i=0}^{n-2} (2(n-i)-(n-1))$$

Суммируя затраты на реализацию прямого и обратного хода, получаем

$$T = T' + T'' = \sum_{i=0}^{n-2} (2(n-i)^2 + (n-i)) + \sum_{i=0}^{n-2} \big(2(n-i)-(n-1)\big)$$
$$= \sum_{j=2}^{n} (3j + 2j^2) - (n-1)$$

В последнем равенстве пределы и порядок суммирования изменены с учетом замены j=n-i.

Используя формулы суммирования,

$$1^2 + 2^2 + 3^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

получаем следующее соотношение для оценки вычислительной сложности метода Гаусса при его реализации в виде последовательного алгоритма:

$$T = \frac{4n^3 + 9n^2 + 10n - 12}{6}$$

Вычислительная сложность алгоритма Гаусса имеет порядок O(n3).

```
usescrt;
const
Nmax=9;
eps=0.00001; { все числа, меньшие eps, в процессе решения полагаются равными 0 }
```

```
type
matr=array [1..Nmax,1..Nmax] of real;
mas=array [1..Nmax] of real;
var
i,j,N:integer;
b,x:mas;
a:matr;
{*** методГаусса *******}
proceduregausss(ag:matr; bg:mas; varxg:mas; Ng:integer);
Var
k,ig,jg:byte;
m,s:real;
blg:boolean;
c:mas;
begin

{ приведение к треугольному виду}
For k:=1 to Ng-1 do
begin
   If ABS(ag[k,k])<eps then
begin
ig:=k;
blg:=false;
repeat
Inc(ig);
if ABS(ag[ig,k])>eps then
begin
blg:=true;
c:=ag[k];
ag[k]:=ag[ig];
ag[ig]:=c;
s:=bg[k];
bg[k]:=bg[ig];
bg[ig]:=s;
end;
untilblg;
end;
m:=ag[k,k];
forjg:=k to Ng do
ag[k,jg]:=ag[k,jg]/m;
bg[k]:=bg[k]/m;
forig:=k+1 to Ng do
if ABS(ag[ig,k])>eps then
begin
```

```
m:=ag[ig,k];
forjg:=k to Ng do
ag[ig,jg]:=ag[k,jg]-ag[ig,jg]/m;
bg[ig]:=bg[k]-bg[ig]/m;
end
else
ag[ig,k]:=0;
end;
{расчет неизвестных x в обратном порядке}
xg[Ng]:=bg[Ng]/ag[Ng,Ng] ;
forig:=(Ng-1) downto 1 do
begin
s:=0;
  For jg:=ig+1 to Ng do
s:=s+ag[ig,jg]*xg[jg] ;
xg[ig]:=bg[ig]-s;
end;
end;

BEGIN {***** тело программы ******}
clrscr;
Write('N= ');
Readln(N);
writeln ('ввод матрицы коэффициентов при неизвестных x');
for i:=1 to N do
for j:=1 to N do
begin
write('a[',i,',',j,'] = ');
readln(a[i,j]);
end;
writeln ('ввод столбца свободных членов');
for i:=1 to N do
begin
write('b[',i,'] = ');
readln(b[i]);
end;
Writeln;
gausss (a,b,x,N);
writeln ('Вывод результатов решения системы уравнений методом Гаусса');
for i:=1 to N do
writeln('x[',i,'] = ',x[i]:0:5);
readln
END.
```

Контрольные вопросы

1. Определение сложности алгоритма
2. Временная сложность алгоритма.
3. Объемная сложность алгоритма

Задания для самостоятельной работы

1. Определить сложность алгоритма пузырьковой сортировки
2. Определить сложность алгоритма сортировки прямым включением

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Медведев, Д.М. Структуры и алгоритмы обработки данных в системах автоматизации и управления [Электронный ресурс]: учебное пособие/ Д.М. Медведев. — Электрон. текстовые данные. — Саратов: Ай Пи Эр Медиа, 2018. — 100 с. — 978-5-4486-0192-7. — Режим доступа: http://www.iprbookshop.ru/71591.html
2. Никлаус, Вирт Алгоритмы и структуры данных [Электронный ресурс]/ Вирт Никлаус. — Электрон. текстовые данные. — Саратов: Профобразование, 2017. — 272 с. — 978-5-4488-0101-3. — Режим доступа: http://www.iprbookshop.ru/63821.html
3. Самуйлов, С.В. Алгоритмы и структуры обработки данных [Электронный ресурс]: учебное пособие/ С.В. Самуйлов. — Электрон. текстовые данные. — Саратов: Вузовское образование, 2016. — 132 с. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/47275.html
4. Сундукова, Т.О. Структуры и алгоритмы компьютерной обработки данных [Электронный ресурс]/ Т.О. Сундукова, Г.В. Ваныкина. — Электрон. текстовые данные. — М.: Интернет-Университет Информационных Технологий (ИНТУИТ), 2016. — 749 с. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/57384.html
5. Вирт, Н. Алгоритмы и структуры данных: пер. с англ. [Текст]: учеб. пособие/ Вирт Н. - СПб.: Невский Диалект , 2008.- 352 с.
6. Игошин, В.И. Математическая логика и теория алгоритмов [Текст]: учеб. пособие для вузов/ В.И. Игошин - М.: Академия, 2008.- 448 с.
7. Курапова, Е.В. Структуры и алгоритмы обработки данных [Электронный ресурс]: лабораторный практикум/ Е.В. Курапова, Е.П. Мачикина. — Электрон. текстовые данные. — Новосибирск: Сибирский государственный университет телекоммуникаций и информатики, 2015. — 23 с. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/55501.html
8. Практикум по дисциплине Структуры и алгоритмы обработки данных [Электронный ресурс]/. — Электрон. текстовые данные. — М. : Московский технический университет связи и информатики, 2016. — 16 с. — 2227-8397. — Режим доступа: http://www.iprbookshop.ru/61551.html
9. Синюк, В.Г. Алгоритмы и структуры данных [Электронный ресурс]: лабораторный практикум. Учебное пособие/ В.Г. Синюк, Ю.Д. Рязанов. — Электрон. текстовые данные. — Белгород: Белгородский государственный технологический университет им. В.Г. Шухова, ЭБС АСВ, 2013. — 204 с — 978-5-361-00194-1. — Режим доступа: http://www.iprbookshop.ru/28363.html

КОЧКАРОВА Паризат Ахматовна
ЭРКЕНОВА Мадина Умаровна

СТРУКТУРЫ И АЛГОРИТМЫ КОМПЬЮТЕРНОЙ ОБРАБОТКИ ДАННЫХ

Учебно-методическое пособие для обучающихся 3 курса
направлению подготовки 09.03.03 Прикладная информатика

Корректор Чагова О.Х.
Редактор Чагова О.Х.

Сдано в набор 18.09.2023 г.
Формат 60х84/16
Бумага офсетная.
Печать офсетная.
Усл. печ. л.3,48
Заказ № 4786
Тираж 100 экз.

Оригинал-макет подготовлен
в Библиотечно-издательском центре СКГА
369000, г. Черкесск, ул. Ставропольская, 36