№ 30 (223) август 1999

ИНФОРМАТИКА

еженедельное приложение
к газете «ПЕРВОЕ СЕНТЯБРЯ»

# Практикум по Турбо Паскалю

И.А. Бабушкина,
Н.А. Бушмелева,
С.М. Окулов,
С.Ю. Черных


Выпуск 3

# Содержание*

Предисловие

1. НАЧАЛА ПАСКАЛЯ

1.1. Первые занятия
1.1.1. Первое знакомство с системой программирования Турбо Паскаль
1.1.2. Простейшие линейные программы
1.1.3. Целый и логический типы данных. Условный оператор
1.1.4. Целый тип данных. Цикл с параметром
1.1.5. Работа с окнами. Метод пошагового выполнения программ
1.1.6. Решение задач с использованием цикла с параметром
1.2. Циклы с условиями
1.2.1. Длинные целые числа. Циклы с предусловием
1.2.2. Цикл с постусловием
1.2.3. Алгоритм Евклида
1.2.4. Вложенные циклы
1.2.5. Решение задач с использованием циклов с условием
1.3. Простые типы данных
1.3.1. Символьный тип данных
1.3.2. Вещественный тип данных
1.3.3. Ограниченный и перечисляемый типы данных. Оператор варианта
1.3.4. Описание переменных, констант и типов
1.3.5. Преобразование типов. Совместимость типов
1.4. Контрольные работы
1.4.1. Контрольная работа № 1
1.4.2. Контрольная работа № 2

2. ОСНОВЫ ПАСКАЛЯ

2.1. Процедуры и функции
2.1.1. Описание процедуры. Оператор процедуры
2.1.2. Функции
2.1.3. Примеры рекурсивного программирования
2.2. Файловый тип данных
2.2.1. Общие положения
2.2.2. Файловый тип данных. Операции для работы с файлами последовательного доступа
2.2.3. Текстовые файлы
2.3. Регулярные типы данных
2.3.1. Одномерные массивы. Работа с элементами
2.3.2. Методы работы с элементами одномерного массива
2.3.3. Удаление элементов из одномерного массива
2.3.4. Вставка элементов в одномерный массив
2.3.5. Перестановки элементов массива
2.4. Двумерные массивы
2.4.1. Описание. Работа с элементами
2.4.2. Двумерные массивы. Работа с элементами
2.4.3. Вставка и удаление элементов
2.4.4. Перестановка элементов массива

**2.5. Строковый тип данных** ..... 3
**2.6. Множественный тип данных** ..... 6
**2.7. Комбинированный тип данных (записи)** ..... 11
**2.8. Контрольные работы** ..... 13
**2.8.1. Одномерные массивы. Работа с элементами. К/р № 1** ..... 13
**2.8.2. Одномерные массивы. Работа с элементами. К/р № 2** ..... 15
**2.8.3. Одномерные массивы. Удаление, вставка и перестановка элементов** ..... 15
**2.8.4. Двумерные массивы. Работа с элементами** ..... 16
**2.8.5. Двумерные массивы. Работа с элементами, вставка, удаление и перестановка строк** ..... 17
**2.8.6. Строковый тип. Множественный тип** ... 17
**2.8.7. Комбинированный тип данных** ..... 17

**3. МЕТОДЫ СОРТИРОВКИ И ПОИСКА ДАННЫХ**

**3.1. Алгоритмы сортировки информации** ..... 18
**3.1.1. Повторение материала предыдущих глав** ..... 18
**3.1.2. Сортировка методом простого выбора** ..... 19
**3.1.3. Сортировка методом простого обмена** ..... 21
**3.1.4. Сортировка методом прямого включения** ..... 22
**3.1.5. Сортировка методом слияний** ..... 23
**3.1.6. Обменная сортировка с разделением (сортировка Хоара)** ..... 25
**3.1.7. Задания для контрольной работы** ..... 26
**3.2. Алгоритмы поиска информации** ..... 26
**3.2.1. Линейный поиск** ..... 26
**3.2.2. Линейный поиск с использованием барьера** ..... 27
**3.2.3. Бинарный поиск** ..... 27
**3.2.4. Поиск подстроки в строке. Прямой поиск** ..... 28
**3.2.5. Поиск подстроки в строке. Алгоритм Р.Бойера и Дж. Мура** ..... 29

4. РЕКУРСИВНЫЕ АЛГОРИТМЫ

4.1. Знакомство с рекурсией
4.2. Простые задачи
4.3. Фрактальные кривые
4.4. Перебор с возвратом

5. ДИНАМИЧЕСКИЕ СТРУКТУРЫ ДАННЫХ

5.1. Введение: о статических и динамических переменных
5.2. Ссылки и указатели
5.3. Линейные списки (основные операции)
5.3.1. Создание списка
5.3.2. Основные операции над списками
5.4. Стек
5.4.1. Введение понятия
5.4.2. Реализация стеков на языке программирования
5.4.3. Реализация основных операций
5.5. Очереди
5.5.1. Введение понятия
5.5.2. Основные операции над очередью
5.6. Деревья
5.6.1. Введение основных понятий
5.6.2. Представление деревьев
5.6.3. Основные операции над деревом
5.6.4. Поиск и включение элемента в дерево
5.6.5. Удаление дерева

* Книга выходит в нескольких номерах нашей газеты. Номера страниц указаны у разделов, вошедших в данный номер.

## 2.5. СТРОКОВЫЙ ТИП ДАННЫХ

### Повторение

**1.** Как описать одномерный массив символов?
**2.** Как заполнить его? Что является элементом такого массива?
**3.** Как вывести все элементы друг за другом?

### Описание

*Строкой* называется последовательность заданной длины, состоящая из символов.

Строки (переменные типа `String`) могут быть объявлены, например, следующим образом:

```
Var Str1: String[30]; Str2: String;
```

При объявлении строковой переменной в квадратных скобках может указываться длина строки. Если длина строки не указана, то она принимается равной 255. Максимальная длина строки также равна 255. В данном случае в первой строке может содержаться максимум 30 символов, а во второй — 255. Надо заметить, что строка похожа на одномерный массив символов: она имеет определенную длину (не больше некоторого числа), к каждому символу можно обратиться по его номеру (как в массиве) — `Str1[i]` — это обращение к `i`-му элементу строки `Str1`.

Переменные типа `String` выводятся на экран посредством стандартных процедур `Write` и `Writeln` и вводятся с помощью стандартных процедур `Readln` и `Read`. То есть вводятся и выводятся не поэлементно, как массивы, а целиком.

**Примечание.** Если при вводе задать символов больше, чем максимально допустимо, то лишние символы будут проигнорированы.

### Операции со строками

В Паскале имеется два основных способа обработки переменных типа `String`. Первый способ предполагает обработку всей строки как единого целого, т.е. единого объекта. Кроме того (это второй способ), можно рассматривать строку как составной объект, состоящий из отдельных символов, то есть элементов типа `Char`, которые при обработке доступны каждый в отдельности.

### Склеивание

Под склеиванием понимается последовательное объединение нескольких строк.

#### Пример

```
Var Str1, Str2, Str3: String[20];
Str1:='У Егорки';
Str2:='всегда отговорки';
Str3:=Str1+' '+Str2;
```

Строка `Str3` имеет значение `'У Егорки всегда отго'`. В данном примере максимальная длина строки `Str3` равна 20 символам, поэтому будут взяты только первые 20 символов суммы строк, а остальные рассматриваться не будут. Паскаль позволяет выполнять операции объединения (сцепления) нескольких строк в процессе их присвоения какой-либо переменной: `Str3:= 'У Егорки '+' всегда '+' отговорки'`. В результате такой операции в переменной `Str3` будет то же самое содержимое, что и в предыдущем примере.

**Примечание**. "Склеить" строки можно также при помощи функции `Concat(Str1, Str2,..., StrN)`.

### Сравнение

Паскаль позволяет выполнять операции сравнения двух строк. Сравнение происходит посимвольно слева направо: сравниваются коды соответствующих символов до тех пор, пока не нарушится равенство или не кончится одна из строк (или обе сразу), при этом сразу делается вывод о знаке неравенства. Две строки называются равными, если они равны по длине и совпадают посимвольно.

#### Пример

```
'Balkon' <'balkon' (Ord('B' )<Ord('b' ));
'balkon'>'balken'  (Ord('o')>Ord('e'));
'balkon'>'balk'    (длина первой строки больше);
'кошка '>'кошка'   (длина первой
                    строки больше);
'Кот'='Кот'        (равны по длине и совпадают
                    посимвольно).
```

Можно использовать любые операции отношения (`>`, `<`, `=`, `<>`, `>=`, `=<`) и их комбинации в условных операторах. Их результат — это одно из двух значений: `True` или `False`.

Для доступа к отдельному символу в строке необходимо указать имя строки и в квадратных скобках номер позиции элемента (символа) в строке. При этом по отношению к отдельному символу строки возможны все те же операции, что и к переменной типа `Char`.

### Стандартные процедуры и функции

Паскаль предоставляет в распоряжение программиста целый ряд встроенных функций и процедур, предназначенных для обработки строк. Рассмотрим наиболее важные из них.

### Удаление

Для удаления из строки фрагмента используется процедура `Delete(Str,n,m)`, которая вырезает из строки `Str` `m` символов начиная с `n`-го, таким образом, сама строка изменяется.

**Пример**

```
Str1:='ABCDEFGH';
Delete(Str1, 3, 4);
Writeln(Str1);
```

После выполнения этих операторов из строки будут удалены четыре символа начиная с третьего, то есть строка будет такой: `Str1='ABGH'`.

### Вставка

Для вставки подстроки в строку используется процедура `Insert(Str1,Str2,n)`, которая вставляет строку `Str1` в строку `Str2` начиная с `n`-го символа, при этом первая строка остается такой же, как и была, а вторая получает новое значение.

**Пример**

```
Str1:='ABCDEFGH';

Str2:='abcdefgh';
Insert(Str1, Str2, 3);
```

В результате выполнения данной процедуры строка будет такой: `Str2='abABCDEFGHcdefgh'`. Этот же результат будет и после выполнения такой последовательности операторов:

```
Str2:='abcdefgh';
Insert('ABCDEFGH',  Str2, 3);
```

### Копирование

Функция `Copy(Str,n,m)` копирует `m` символов строки `Str` начиная с `n`-го символа, при этом исходная строка не меняется.

**Пример**

```
Srt1:='ABCDEFGH';
Str2:='abcdefgh';
Str3:=Copy(Str1,4,3);
Writeln(Str3);
Writeln(Copy(Str2,4,3));
```

Значение переменной `Str3='DEF'`. А на экран будут выведены следующие строки:

```
DEF
def
```

### Длина строки

Под длиной строки понимается фактическое (а не максимально возможное!) количество символов в строке. Это значение можно найти при помощи функции `Length(Str)`, результат которой — целое число, равное количеству символов.

**Пример**

```
Str1:='ABCDEFGH';
Str2:='Мама мыла раму.';
k1:=Length(Str1);
k2:=Length(Str2).
```

В результате значения целых переменных будут равны: `k1=8, k2=15`.

### Поиск подстроки

Имеется функция, определяющая позицию подстроки в строке, — `Pos(Str1, Str2)`. Результат этой функции — целое число, и оно определяет номер элемента, с которого начинается первое вхождение подстроки `Str1` в строку `Str2`. Если `Str1` не входит в `Str2`, то значение функции равно `0`.

**Пример**

```
Str1:= 'CDF'; Str2:= 'ABCDEFGH';
k1:=Pos(Str1, Str2);
k2:=Pos(Str2, Str1);
```

Значение переменной `k1` равно `3`, так как строка `Str1` входит в строку `Str2` с третьего символа, а значение `k2` равно `0`.

**Пример**

```
k1:=Pos('ша', 'Наша Таня громко плачет.')
```

В этом случае значение `k1` равно `3`, так как функция `Pos` возвращает номер элемента, начиная с которого подстрока встречается первый раз.

### Числа и строки

Часто возникает необходимость получить строковое представление числа и наоборот (например, получить строку '13' из числа 13). Для работы с числами и строками применяются две процедуры.

`Str(N,Str1)` — переводит числовое значение `N` в строковое и присваивает результат строке `Str1`, причем можно переводить как целые числа, так и вещественные.

**Примеры**

`Str(1234, Str1)` — после выполнения `Str1='1234'`;

`Str(452.567,Str1)` — переводим вещественное число с фиксированной запятой, результат `Str1='452.567'`;

`Str(4.52567e+2,Str1)` — переводим вещественное число в экспоненциальной форме, результат `Str1='4.52567e+2'`.

Вторая процедура выполняет обратное действие. `Val(Str, N, K)` — переводит строковое значение в числовое. Если данная строка действительно является записью числа (целого или вещественного), то значение `K=0`, а `N` — это искомое число, иначе `K` будет равно номеру первого символа, с которым процедура `Val` "не справилась".

**Примеры**

```
Val('1234',n,k) - n=1234,  k=0;
Val('234.56',n,k) - n=234.56, k=0;
Val('2.3456e+2',n,k) - n=2.3456e+2, k=0;
```

`Val('12-45',n,k)` - `k=3`, так как знак "—" в записи чисел может быть только на первом месте;

Val('2,567m',n,k) - k=2, так как разделительным знаком между целой и дробной частями является точка, а не запятая;

Val('5.87c-5) - k=5, так как символ "c" не должен встречаться в записи вещественного или целого числа.

**Пример**

Сколько раз в данной строке встречается символ 'a'?

*Решение*

Опишем функцию, которой будем передавать строку. Результат выполнения — целое число.

```
Function Q_Ch(st: String): Byte;
Var i, k:  Byte;
Begin
  k:=0;
  {Просматриваем все символы строки, их
   количество равно длине строки;
   если очередной символ равен 'a',
   то увеличиваем счетчик}
   For i:=1 To Length(st) Do
   If st[i]='a' Then Inc(k);
   Q_Ch:=k;
End;
```

**Пример**

Если в строке нечетное число символов, то удалить средний.

*Решение*

```
Procedure Del( Var st: String);
Var k: Byte;
Begin
  k:=Length(st);
  If k Mod 2=1 Then Delete(st, k Div 2+1,1);
End;
```

**Пример**

Заменить все вхождения подстроки 'del' на 'Insert'.

*Решение*

Пока такая подстрока встречается, необходимо находить номер первого символа очередного вхождения подстроки 'del', удалять 'del' и вставлять 'Insert'.

```
Procedure Ins(Var st: String);
Var k: Byte;
Begin
  While Pos('del',st)<>0 Do
  Begin
   k:=Pos('del',st);
   Delete(st,k,Length('del'));
   Insert('Insert',st,k);
  End;
End;
```

**Пример**

Дана строка, состоящая из нескольких слов, между словами один пробел, в конце строки — точка. Подсчитать количество слов и вывести на экран только те из них, которые начинаются с буквы 'a' (слов не больше 30).

*Решение*

Разобьем предложение на отдельные слова и каждое будем хранить как элемент массива строк.

```
Program Example_48;
Const n=30;
Type Myarray_Str=Array[1..n] Of String;
Var   A: Myarray_Str;
    str: String[255];
      k: Byte;
Procedure Init(Var b: Myarray_Str);
Var i: Integer;
Begin
  k:=1;
  {Пока не встретится пробел, формируем
    очередное слово k, прибавляя по одной букве}
  For i:=1 To Length(str)-1 Do
  If str[i]<>' ' Then b[k]:=b[k]+str[i]
  Else
  {Если это не последний символ, то увели-
   чиваем счетчик слов и начинаем формировать
   очередное слово}
  If i<>Length(str)-1 Then
  Begin Inc(k); b[k]:='' End;
End;
Begin
  Writeln('Введите предложение');
  Readln(str);
  Init(A);
  Writeln('Всего слов: ',k);
  {Просматриваем все слова, если первый
   символ очередного слова равен букве 'a',
   то выводим его}
   For i:=1 To k Do If A[i][1]='a'
   Then  Write(A[i],' ');
  Readln;
End.
```

**Пример**

Подсчитать сумму цифр, входящих в данную строку.

```
Function Sum(st: String): Integer;
Var  i, k, d: Byte; s: Integer;
Begin
  s:=0;
  For i:=1 To Length(st)  Do
  Begin**
   Val(st[i],d,k);
   If k=0 Then s:=s+d;
  End;
  Sum:=s;
End;
```

***Решение задач***

**1.** Подсчитать, сколько раз в данной строке встречается данный символ.

**2.** Дан текстовый файл. Сколько раз в каждой строке встречается данный символ?

**3.** Сколько раз в данной строке встречаются гласные буквы?

**4.** Из строки удалить среднюю букву, если длина строки нечетная, иначе — удалить две средние буквы.

**5.** Заменить все символы `Ch1` в строке на `Ch2` (`Ch1` и `Ch2` вводятся с клавиатуры).

**6.** Заменить все вхождения подстроки `Str1` на подстроку `Str2` (`Str1` и `Str2` вводятся с клавиатуры).

**7.** После каждого символа `Ch` вставить строку `Str1`.

**8.** Удвоить каждое вхождение символа `Ch`.

**9.** Даны две строки. Если они начинаются с одинаковых символов, то напечатать `ДА`, иначе — `НЕТ`.

**10.** Дана последовательность слов. Напечатать все слова, отличные от слова `hello`.

**11.** Дана последовательность слов. Напечатать все слова в алфавитном порядке.

**12.** Дана последовательность слов. Напечатать все слова последовательности, которые встречаются в ней по одному разу.

**13.** Дано предложение. Напечатать все различные слова.

**14.** Дана последовательность слов. Напечатать все слова, предварительно преобразовав каждое из них по следующему правилу:

a) удалить из слова все вхождения последней буквы (кроме нее самой);

b) оставить в слове только первые вхождения каждой буквы.

**15.** Дана последовательность слов. Напечатать те слова последовательности, которые отличны от последнего слова и удовлетворяют следующему свойству:

a) в слове нет повторяющихся букв;

b) буквы слова упорядочены по алфавиту;

c) слово совпадает с начальным отрезком латинского алфавита *(a, ab, abc, abcd, ...)*;

d) слово симметрично.

**16.** Составить программу вывода самой большой цифры в строковой записи заданного числа.

**17.** Найти сумму всех чисел строки.

## 2.6. МНОЖЕСТВЕННЫЙ ТИП ДАННЫХ

*Множество* в языке Паскаль представляет собой набор различных элементов одного (базового) типа.

*Базовый тип* — это совокупность всех возможных элементов множества. Всего в базовом типе должно быть не более 256 различных элементов. Значение переменной множественного типа может содержать любое количество различных элементов базового типа — от нуля элементов (пустое множество) до всех возможных значений базового типа (напомним, что их должно быть не более 256). Иными словами, возможными значениями переменных множественного типа являются все подмножества базового типа.

Множества, используемые в программе, могут быть описаны либо в разделе **`Type`**:

```
Type <имя типа> = Set Of <тип элементов>;
Var <имя множества> : <имя типа>;
```

либо непосредственно в разделе описания переменных **`Var`**:

```
Var <имя множества> : Set Of <тип элементов>;
```

**Пример**

```
Type mnog_Char= Set Of Char;
Var mn1 : Set Of Char;
mn2: mnog_Char;
mn3: Set Of 'A'.. 'Z';
s1: Set Of Byte;
s2: Set Of 1000..1200;
```

Здесь `mn1` и `mn2` — это множества символов; так как различных символов всего 256, то тип `Char` можно использовать в качестве базового;

`mn3` — множество больших латинских букв;

`s1` — множество целых чисел (от 0 до 255); так как тип `Byte` содержит только целые числа от 0 до 255 (всего 256 различных чисел), его тоже можно использовать в качестве базового типа элементов;

`s2` — множество целых чисел от 1000 до 1200*.

*Формирование (конструирование) множеств.* В программе элементы множества задаются в квадратных скобках, через запятую. Если элементы идут подряд друг за другом, то можно использовать диапазон.

**Пример**

```
Type digit = Set Of 1..5;
Var s : digit;
```

Переменная `s` может принимать значения, состоящие из любой совокупности целых чисел от 1 до 5:

`[ ]` — пустое множество;

`[1], [2], [3], [4], [5]` — одноэлементные множества;

`[1, 2], [1,3], ..., [2,4], [4,5]` — двухэлементные (пара любых элементов);

`[1, 2, 3], [1,2,4], ..., [3,4,5]` — трехэлементные (тройка элементов);

`[1, 2, 3, 4], [1,2,3,5], [1,2,4,5], [1,3,4,5], [2,3,4,5]` — четырехэлементные;

`[1, 2, 3, 4, 5]` — полное множество (взяты все элементы базового типа).

* Турбо Паскаль требует, чтобы элементы множества занимали в памяти 1 байт, поэтому это описание он воспримет как ошибочное. — **Прим. ред.**

## Операции над множествами

*Объединением* двух данных множеств называется множество элементов, принадлежащих хотя бы одному из этих множеств. Знак операции объединения множеств в Паскале — "+".


### Примеры

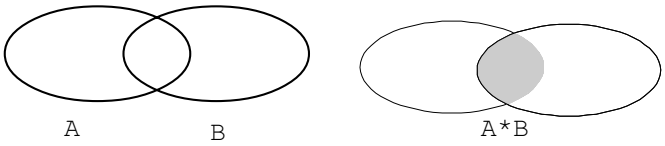
1) `['A', 'F'] + ['B', 'D'] = = ['A', 'F', 'B', 'D'];`

2) `[1..3,5,7,11] + [3..8,10,12,15..20] = = [1..8,10..12,15..20]`

Пусть `S1:= [1..5,9]`, а `S2:= [3..7,12]`. Тогда если `S:= S1 + S2`, то `S=[1..7,9,12]`.

Пусть `A1:=['a'..'z']; A1:=A1 + ['A']`. Тогда `A1=['A', 'a'..'z']`.

*Пересечением* двух множеств называется множество элементов, принадлежащих одновременно и первому, и второму множеству. Знак операции пересечения множеств в Паскале — "*".



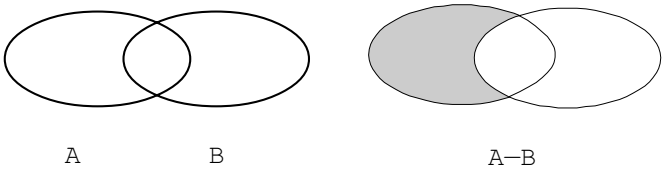

### Примеры

1) `['A', 'F'] * ['B', 'D'] = [ ]`, так как общих элементов нет;

2) `[1..3,5,7,11] * [3..8,10,12,15..20] = = [3,5,7];`

3) если `S1:= [1..5,9]` и `S2:= [3..7,12]`, а `S:= S1 * S2`, то `S=[3..5]`.

*Разностью* двух множеств называется множество, состоящее из тех элементов первого множества, которые не являются элементами второго. Знак операции вычитания множеств — "—".




### Примеры

1) `['A', 'F'] - ['B', 'D'] = ['A', 'F']`, так как общих элементов нет;

2) `[1..3,5,7,11] - [3..8,10,12,15..20] = = [1,2,11];`

3) `S1:= [1..5,9]; S2:= [3..7,12]; S:= S1 - S2;` Тогда `S=[1,2,9];`

4) `A1:=['A'..'Z']; A1:=A1 - ['A']`. Тогда `A1=['B'..'Z']`.

## Операция определения принадлежности элемента множеству

Эта логическая операция обозначается служебным словом `in`. Результат операции имеет значение `true`, если элемент входит в множество, и `false` в противном случае.

### Примеры

1) Выражение `5 in [3 .. 7]` имеет значение `true`, так как $5 \in [3; 7]$;

2) выражение `'a' in ['A' .. 'Z']` имеет значение `false`, так как маленькой латинской буквы "a" нет среди больших латинских букв.

**Примечание.** Операцию проверки принадлежности элемента множеству удобно использовать для исключения более сложных проверок. Например, оператор вида:

```
If (ch='a') or (ch='b') or (ch='x')
    or (ch='y') Then...
```

— может быть переписан в компактной и наглядной форме:

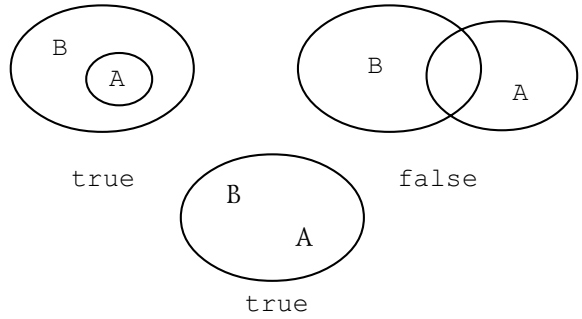
```
If ch in ['a', 'b', 'x', 'y' ] Then...
```

## Сравнение множеств

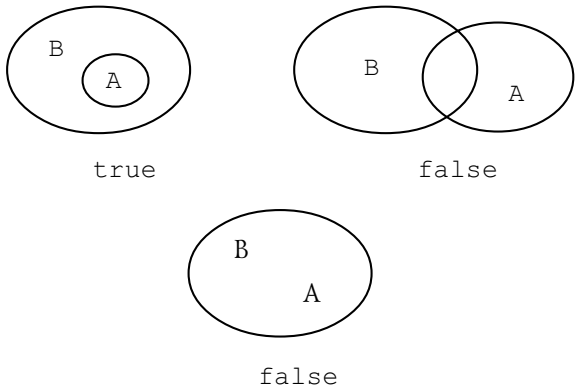
Для сравнения множеств используются операции отношения:

- `=` — проверка на равенство (совпадение) двух множеств;
- `<>` — проверка на неравенство двух множеств;
- `<=, <` — проверка на вхождение первого множества во второе множество;
- `>=, >` — проверка на вхождение второго множества в первое множество.

**Первое множество меньше или равно второму (A ≤ B):**



**Первое множество меньше второго (A < B):**




### Пример

Составить программу выделения следующих множеств из множества целых чисел от 1 до 30:

— множества чисел, кратных 2;
— множества чисел, кратных 3;
— множества чисел, кратных 6;
— множества чисел, кратных 2 или 3.

### Вопросы для обсуждения

**1.** Сколько множеств надо описать? Каков тип их элементов? (Четыре множества с элементами типа Byte.)

**2.** Каково начальное значение множеств? (Начальное значение множеств — пустое множество.)

**3.** Как формируются множества? (Первые два формируются перебором всех чисел данного промежутка и отбором подходящих, а третье и четвертое получаются из первых двух путем применения операций пересечения или объединения.)

**4.** Как осуществить вывод сформированных множеств? (Вывод множеств производится ТОЛЬКО поэлементно, поэтому удобно составить процедуру и передавать в нее множество, элементы которого и будут выводиться на экран. Для этого в разделе типов надо создать соответствующий тип и использовать его в дальнейшем.)

Программа для решения данной задачи выглядит так:

```
Program Example_49;
Const n = 30;
Type mn=Set Of 1..n;
Var n2, n3, n6, n23 : mn;
{ n2 - множество чисел, кратных 2,
  n3 - кратных 3,
  n6 - кратных 6,
  n23 - кратных 2 или 3}
k: Integer;
Procedure Print( m: mn);
Var i: Integer;
Begin
  For i:=1 To n Do If i In m Then Write(i:3);
  Writeln;
End;
Begin
  n2=[ ]; n3=[ ];
  {начальные значения множеств}
  For k:=1 To n Do   {формирование n2 и n3}
  Begin
    {если число делится на 2, то заносим его в n2}
    If k Mod 2 = 0 Then n2:=n2+[k];
    {если число делится на 3, то добавляем его в n3}
    If k Mod 3 = 0 Then n3:=n3+[k];
  End;
{числа, кратные 6, - это те, которые кратны
и 2, и 3, поэтому это пересечение двух первых
множеств, а числа, кратные 2 или 3, - это
объединение этих же множеств}
n6:=n2*n3; n23:=n2+n3;
     {вывод множеств}
Writeln('числа, кратные 2');
Print(n2);
Writeln('числа, кратные 3');
Print(n3);
Writeln('числа, кратные 6');
Print(n6);
Writeln('числа, кратные 2 или 3');
Print(n23);
Readln;
End.
```

### Задание

Изменить программу так, чтобы результатом ее работы являлось множество чисел, делящихся на 3, но не делящихся на 2.

### Пример

"Мешанина". Если взять то общее, что есть у боба с ложкой, добавить кота и поместить в тепло, то получится муравей. Так ли это? Состоит ли муравей из кота?

### Вопросы для обсуждения

**1.** Сколько множеств надо задать и каков тип их элементов? (Четыре множества с элементами символьного типа.)

**2.** Как сформировать множества? (С помощью оператора присваивания: например, s1:= ['б', 'о', 'б'])*

**3.** Как сформировать искомые множества? (С помощью операций пересечения, объединения, вычитания множеств.)

Программа для решения этой задачи приведена ниже:

```
Program Example_50;
Var y1, y2, y3, y4, x : Set Of Char;
s : Char;
Begin
 y1:=['б', 'о', 'б']; y2:=['л', 'о', 'ж', 'к', 'а'];
 y3:=['к', 'о', 'т']; y4:=['т', 'е', 'п', 'л', 'о'];
 x:=(y1*y2)+y3-y4;
```

* Для того чтобы не исключать повторяющиеся элементы самостоятельно, а переложить эту работу на компилятор, действительно можно писать так. Но надо понимать, что множество s1 будет состоять всего из двух элементов.

```
 Writeln('множество x');  {вывод множества x}
 For s:='a' To 'я' Do If s In x
   Then Write(s); Writeln;
     {проверка: состоит ли муравей из кота}
   If y3<=x Then Write('муравей состоит из кота')
   Else Write('муравей не состоит из кота');
End.
```

### Пример

Дано натуральное число n. Составить программу, печатающую в порядке возрастания все цифры, не входящие в десятичную запись данного натурального числа.

### Вопросы для обсуждения

**1.** Какое множество нужно сформировать? (Множество цифр, входящих в десятичную запись данного числа.)

**2.** Как сформировать это множество? (С помощью операций div и mod последовательно выделить цифры данного числа и занести их во множество.)

**3.** Как получить искомый результат и как его вывести?

Программа для решения этой задачи такова:

```
Program Example_51;
Type mn = Set Of 0..9;
Var s : mn;
n : Longint;
l,k : Integer;
Begin
  Writeln ('введите число n');
  Readln(n);
  s:=[ ]; {формирование множества цифр
        десятичной записи натурального числа}
  While n<>0 Do
  Begin
    k:=n Mod 10;
    n:=n Div 10;
    If Not (k In s) Then s:=s+[k];
  End;
  {вывод цифр в порядке возрастания}
  For k:=0 To 9 Do
    If Not (k In s) Then Write(k:2);
    Writeln;
    Readln;
End.
```

### Пример

"Решето Эратосфена". Составить программу поиска простых чисел в промежутке $[1; n]$. Число $n$ вводится с клавиатуры.

*Решение*

*Простым* называется число, которое не имеет других делителей, кроме единицы и самого этого числа. Для решения этой задачи воспользуемся методом "решета Эратосфена", идея которого заключается в следующем: сформируем множество M, в которое поместим все числа заданного промежутка. Затем последовательно будем удалять из него элементы, кратные 2, 3, 4 и так далее, до $[n/2]$ (целая часть числа). После такого "просеивания" в множестве M останутся только простые числа.

**Примечание.** Легко доказать, что можно удалять числа, кратные 2, 3, 4 и так далее, до $[\sqrt{n}\,]$, то есть до целой части квадратного корня числа $n$.

### Вопросы для обсуждения

**1.** Как сформировать множество M?

**2.** Как организовать просмотр элементов этого множества?

**3.** Как организовать перебор делителей?

**4.** Как удалить элемент из множества?

**5.** Как организовать вывод "просеянного" множества?

Ниже приведен вариант решения этой задачи.

```
Program Example_52;
Var m: Set Of Byte;
i,k,n: Integer;
Begin
Writeln(' введите размер промежутка (до 255) ');
Readln(n);
m:=[2..n]; {начальное значение}
For k:=2 To n Div 2 Do
                    {перебираем все делители }
  For i:=2 To n Do
   {если число кратно делителю и отлично
    от него, то удаляем его}
    If (i Mod k=0) And (i<>k) Then m:=m-[i];
     { распечатаем оставшиеся элементы }
  For i:=1 To n Do If i In m Then Write(i:3);
Readln;
End.
```

### Пример

Дан ребус:

$$\begin{array}{r} \text{МУХА} \\ +\ \text{МУХА} \\ \hline \text{СЛОН} \end{array}$$

*Решение*

Каждой букве соответствует некоторая цифра, разным буквам соответствуют разные цифры. Необходимо заменить буквы цифрами так, чтобы получилось верное равенство. Найти все решения (если есть несколько).

Для решения этой задачи применим метод перебора с возвратом. Используем множество S1 для хранения цифр слова МУХА, причем будем заносить в него цифры последовательно, учитывая уже внесенные цифры. Начальное значение S1 — пустое множество. После выбора всех цифр первого слова формируем соответствующее число и находим число, соответствующее слову СЛОН. Выделяем цифры СЛОНа (множество S2), и если слова состоят из разных цифр (то есть пересечение S1 и S2 пустое) и все цифры СЛОНа разные, то выводим решение на экран. Далее удаляем из множества S1 последнюю внесенную цифру и пытаемся выбрать еще одно ее значение. Таким образом, мы перебираем все возмож-

ные варианты и выводим на экран только те, которые удовлетворяют равенству.

Заметим, что букве "М" в слове МУХА может соответствовать цифра от 1 до 4, а букве "А" в этом же слове не может соответствовать 0.

Ниже приводится одно из решений.

```
Program Example_53;
Type mn = Set Of 0..9;
Var m,y,x,a: 0..9;   {цифры числа МУХА}
n1,n2: Integer; {числа МУХА и СЛОН}
a1,a2,a3,a4: 0..9;   {цифры числа СЛОН}
s1, s2:mn;
{для хранения цифр каждого из чисел}
Procedure Print(x,y:Integer);  {вывод
решения в виде ребуса}
Begin
  Writeln(x:5);
  Writeln('+');
  Writeln(x:5);
  Writeln(' ____')
  Writeln(y:5);
End;
Begin
  s1:=[ ];s2:=[ ];
  For m:=1 To 4 Do
  Begin
    s1:=s1 + [m];
    {заносим первую использованную цифру}
     For y:=0 To 9 Do
     {если эта цифра не была еще взята,
      то добавляем ее во множество цифр числа
      МУХА и выбираем цифру для следующей
      буквы}
     If Not (y In s1) Then
     Begin
       s1:=s1+[y];
       For x:=0 To 9 Do
       If Not (x In s1) Then
       Begin
         s1:= s1 + [x];
         For a:=1 To 9 Do
         If Not (a In s1) Then
         Begin
          s1:=s1+[a];
          n1:=1000*m+100*y+10*x+a;
          {число для слова МУХА}
           n2:=2*n1;
          {число для слова СЛОН}
          a1:=n2 Div 1000;
          {выделяем цифры СЛОНа}
          a2:=n2 Div 100 Mod 10;
          a3:=n2 Div 10 Mod 10;
          a4:=n2 Mod 10;
          s2:=[a1,a2,a3,a4];
          {множество цифр СЛОНа}
          {если слова состоят из разных
           цифр и в слове СЛОН нет
           одинаковых букв, то выводим
           решение ребуса на экран}
           If (s1*s2=[ ]) And
             ([a1]*[a2]*[a3]*[a4]=[ ])
           Then Print(n1,n2);
           s1:=s1 - [a];
         {удаляем занесенную цифру}
         End;
         s1:=s1-[x];
       End;
      s1:=s1-[y];
    End;
    s1:=s1-[m];
  End;
  Readln;
End.
```

***Решение задач***

**1.** Дана непустая последовательность символов. Построить и напечатать множества, элементами которых являются встречающиеся в последовательности:

a) цифры от '0' до '9' и знаки арифметических операций;

b) буквы от 'A' до 'F' и от 'X' до 'Z';

c) знаки препинания и буквы от 'E' до 'N'.

**2.** Составить программу подсчета общего количества цифр и знаков '+', '-', '*' в строке s, введенной с клавиатуры.

**3.** Составить программу печати элементов данного множества символов в алфавитном порядке.

**4.** Составить программу формирования множества строчных латинских букв, входящих в строку, введенную с клавиатуры, и подсчета количества знаков препинания в ней.

**5.** Составить программу подсчета количества цифр в заданной строке и печати их.

**6.** Составить программу печати по одному разу в алфавитном порядке всех строчных русских гласных букв, входящих в заданный текст.

**7.** Составить программу печати в алфавитном порядке всех букв текста (текст оканчивается точкой), входящих в него:

a) не менее двух раз;

b) не более двух раз;

c) более двух раз.

**8.** Составить программу печати в возрастающем порядке всех цифр, входящих в десятичную запись данного числа.

**9.** Составить программу печати всех символов заданного текста, входящих в него по одному разу.

**10.** Составить программу для подсчета количества гласных и согласных букв в заданном тексте и определения, каких букв больше (гласных или согласных); учесть, что в строке могут быть и другие символы, кроме букв.

**11.** Составить программу печати всех первых вхождений в данный текст строчных латинских букв, сохраняя их взаимный порядок.

**12.** Составить программу поиска и печати в порядке убывания всех простых чисел из промежутка [2; 201], используя метод "решето Эратосфена".

**13.** Задано множество вычислительных машин. Известен набор машин, имеющихся в каждом из 10 техникумов города. Построить и распечатать множества, включающие в себя вычислительные машины:

a) которыми обеспечены все техникумы;
b) которые имеет хотя бы один техникум;
c) которых нет ни в одном техникуме.

**14.** Решите ребусы.

a)
```
  ЛОБ
+ ТРИ
-----
  САМ
```

d)
```
   ТОЧК
+  КРУГ
-------
  КОНУС
```

b)
```
  ИКС
+ ИСК
-----
  КСИ
```

e)
```
   VOLVO
+   FIAT
-------
  MONOR
```

c) ABC = AB + BC + CA.

## 2.7. КОМБИНИРОВАННЫЙ ТИП ДАННЫХ (ЗАПИСИ)

При использовании массивов основное ограничение заключается в том, что все элементы должны иметь один и тот же тип. Но при решении многих задач возникает необходимость хранить и обрабатывать совокупности данных различного типа.

### Пример

Известны фамилии и оценки (в баллах) по пяти дисциплинам каждого из двадцати пяти учеников класса. Требуется вычислить среднюю оценку каждого из учеников и выбрать человека, имеющего максимальный средний балл.

В данном случае фамилия может быть представлена символьной строкой, оценки — это целые числа, а средний балл должен быть представлен вещественным (действительным) числом. В Паскале для описания объектов, содержащих данные разных типов, используются записи.

*Запись* — это структурированный тип, описывающий набор данных разных типов. Составляющие запись объекты называются ее полями. Каждое поле имеет уникальное (в пределах записи) имя. Чтобы описать запись, необходимо указать ее имя, имена объектов, составляющих запись, и их типы. Общий вид описания записи следующий:

```
Type <имя записи> = Record
  <поле 1>:<тип 1>;
  <поле 2>:<тип 2>;
  ...
  <поле n>:<тип n>
                    End;
```

Применительно к рассматриваемой задаче запись можно описать следующим образом:

```
Type
  pupil = Record
  fam: String[15];   {поле фамилии ученика}
  b1,b2,b3,b4,b5: 2..5;
  {поля баллов по дисциплинам}
  sb: Real {поле среднего балла}
          End;
```

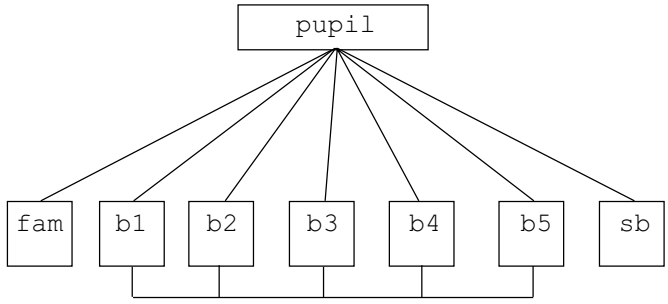



*Рис.* 8

Переменная типа `pupil` будет хранить структуру, содержащую информацию об одном ученике. Организация этой структуры показана на *рис.* 8.

Чтобы хранить в памяти ЭВМ информацию обо всех 25 учениках класса, необходимо ввести массив `klass`, представляющий массив записей:

```
Var klass: Array[1..25] Of pupil;
```

**Примечания.**

1. Имена полей, составляющих запись, не должны повторяться.
2. Каждое поле записи может иметь любой тип (кроме файлового), в частности, оно может быть снова записью.

Доступ к полям записи можно осуществить двумя способами:

*С указанием имени переменной и имени поля.* Например, `klass[2].fam, klass[3].sb, klass[1].b4`. Ввод фамилий и оценок учащихся, то есть элементов массива `klass`, можно записать так:

```
For i:=1 To 25 Do
Begin
  Readln(klass[i].fam);
  Readln(klass[i].b1);
  Readln(klass[i].b2);
  Readln(klass[i].b3);
  Readln(klass[i].b4);
  Readln(klass[i].b5);
End;
```

*С использованием оператора присоединения.* Имеется возможность осуществлять доступ к полям записи таким образом, как если бы они были простыми переменными. Общий вид оператора присоединения:

```
With <имя записи> Do <оператор>;
```

Внутри оператора присоединения к компонентам записи можно обращаться только с помощью имени соответствующего поля.

### Пример

```
For i:=1 To 25 Do
With klass[i] Do
Begin
  Readln(fam);
  ReadLn(b1,b2,b3,b4,b5);
End;
```

Программа для решения рассматриваемой задачи может быть записана следующим образом:

```
Program Example_54;
Type
  pupil = Record
  fam:String[15];
  b1,b2,b3,b4,b5:2..5;
  sb:Real;
          End;
Var klass: Array[1..25] Of pupil;
p: pupil;
i,m: Integer;
sbmax: Real;
Begin
  For i:=1 To 25 Do  {ввод исходных данных}
  With klass[i] Do
  Begin
    Writeln('Введите фамилию и пять оценок');
    Readln(fam);
    ReadLn(b1,b2,b3,b4,b5);
  End;
  For i:=1 To m Do
  {вычисление среднего балла}
  With klass[i] Do sb:=(b1+b2+b3+b4+b5)/5;
  sbmax:=0;
  {поиск максимального среднего балла}
  For i:=1 To m Do
  If klass[i].sb>=sbmax Then sbmax:=klass[i].sb;
   For i:=1 To m Do  {печать результатов}
     If klass[i].sb=sbmax Then
   With klass[i] Do Writeln(fam:20,' - ',sb:6:3);
    Readln;
End.
```

**Пример**

Определить дату завтрашнего дня.

*Решение*

Чтобы определить дату завтрашнего дня, надо знать не только дату сегодняшнего дня, но и количество дней в данном месяце (так как если это последний день месяца, то завтра будет первый день следующего), кроме того, надо знать, какой год — високосный или нет (от этого зависит количество дней в феврале).

Пусть дата вводится в формате *число — месяц — год* следующим образом:

*1 2 1997*

Опишем запись для хранения даты таким образом:

```
Type year=1500..2000;
  month=1..12;
  day=1..31;
  data=Record
  y: year;
  m: month;
  d: day;
       End;
```

Заметим, что:

- Если дата соответствует не последнему дню месяца, то год и месяц не изменяются, а число увеличивается на 1.
- Если дата соответствует последнему дню месяца, то:

а) если месяц не декабрь, то год не изменяется, месяц увеличивается на 1, а число устанавливается в 1;

б) если месяц — декабрь, то год увеличивается на 1, а месяц и число устанавливаются в 1.

```
Program Example_55;
Type year=1500..2000;
  month=1..12;
  day=1..31;
  data=Record
  y: year;
  m: month;
  d: day;
       End;
Var dat, next: data;
{dat - переменная для сегодняшней даты
next - переменная для определения даты
завтрашнего дня}
Function Leap( yy: year ): Boolean;
{функция, определяющая, високосный год или нет}
Begin
  {год называется високосным, если его номер
   делится на 4, но если это год столетия,
   то номер столетия високосного года не
   делится на 4, то есть не номер года
   делится на 400}
  Leap:=(yy Mod 4=0) And (yy Mod 400 <> 0);
End;
Function Dmonth(mm: month; yy: year): day;
{функция определения количества дней данного
месяца в данном году}
Begin
  Case mm Of
  1,3,5,7,8,10,12: Dmonth:=31;
  4,6,9,11: Dmonth:=30;
  2: If Leap(yy) Then Dmonth:=29 Else Dmonth:=28;
End;
End;
Procedure Tomorrow(td: data; Var nd: data);
     {процедура определения завтрашней даты}
Begin{если это не последний день месяца}
  If td.d<> Dmonth(td.m,td.y) Then
  With nd Do
  Begin
   d:=td.d+1;
   m:=td.m;
   y:=td.y;
  End
  Else    {если это последний день месяца}
  If td.m=12 Then    {если это декабрь}
  With nd Do
  Begin
   d:=1;
   m:=1;
   y:=td.y+1;
  End
  Else    {если это не декабрь}
  With nd Do
  Begin
   d:=1;
   m:=td.m+1;
   y:=td.y;
  End;
End;
```

```
Begin
  Writeln('Введите сегодняшнее число, месяц и год ');
  Readln(dat.d,dat.m,dat.y);
  Tomorrow(dat,next);
  Writeln('Завтра будет ');
  Writeln(next.d,'.',next.m,'.',next.y);
  Readln;
End.
```

***Решение задач***

**1.** Написать программу, определяющую:

a) дату следующего (предыдущего) дня;

b) дату, которая наступит через m дней;

c) дату, которая была за m дней до сегодняшнего дня;

d) количество суток, прошедших от даты t1 до t2;

e) день недели, выпадающий на дату t1, если известно, что первый день нашей эры был понедельником.

**2.** Запись для хранения времени описана следующим образом:

```
Type time = Record
  h: 0..23;
  m, s: 0..59
            End;
```

Описать:

a) логическую функцию для проверки, предшествует ли время t1 времени t2 (в пределах суток);

b) процедуру, присваивающую параметру t1 время, на 1 секунду большее времени t (учесть смену суток).

**3.**
```
Const n=300;
Type MyRecord = Record
     Key: Integer;
     Name : String;
                 End;
  Table=Array[1..n] Of MyRecord.
```

Считая, что записи в таблице имеют различные ключи, описать:

a) процедуру, упорядочивающую записи таблицы по убыванию значений поля Key;

b) логическую функцию поиск(T,K,H), определяющую, есть ли в таблице T (все записи которой уже упорядочены по возрастанию значений поля Key) запись со значением поля Key, равным K, и, если есть, присваивающую номер этой записи параметру H.

**4.** Дан массив, содержащий информацию об учениках некоторой школы.

a) Заполнить второй массив данными об учениках только девятых классов.

b) Выяснить, на сколько человек в восьмых классах больше, чем в девятых.

**5.** Багаж пассажира характеризуется количеством и общим весом вещей. Дан массив, 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).

a) Найти багаж, средний вес одной вещи в котором отличается не более чем на 0,3 кг от общего среднего веса одной вещи.

b) Найти число пассажиров, имеющих более двух вещей, и число пассажиров, количество вещей которых превосходит среднее число вещей.

c) Выяснить, имеется ли пассажир, багаж которого состоит из одной вещи весом менее 30 кг.

## 2.8. КОНТРОЛЬНЫЕ РАБОТЫ

### 2.8.1. Одномерные массивы. Работа с элементами

#### Контрольная работа № 1

**Вариант № 1**

**1.** Правильно ли описан массив A? Если нет, то что надо изменить?

```
Type myarray=Array[-10..n] Of Integer;
Var A : myarray;
```

**2.** Что будет выведено на печать в результате выполнения программы?

```
Program Variant1;
Const n=7;
Type myarray=Array[1..n] Of Integer;
Var C : myarray;
    i : Byte; p : Integer;
Begin
  p:=0;
  For i:=1 To n Do
  Begin
   C[i]:=-50+Random(151);
   If C[i]>50 Then p:=p+C[i];
  End;
  Writeln(p);
  Readln;
End.
```

**3.** Дан массив целых чисел, состоящий из 20 элементов. Заполнить его с клавиатуры. Найти:

- сумму элементов, имеющих нечетное значение;
- вывести индексы тех элементов, значения которых больше заданного числа A.

**4.** Определить, есть ли в данном массиве положительные элементы, кратные k (k вводится с клавиатуры).

**Вариант № 2**

**1.** Правильно ли описан массив C? Если нет, то что надо изменить?

```
Const n1=25;
Type m=Array[15..-n1] Of Integer;
Var C : m;
```

**2.** Что будет выведено на печать в результате выполнения программы?

```
Program Variant2;
Const n=10;
Type myarray=Array[1..n] Of Integer;
Var D : myarray;
    i : Byte; p : Integer;
Begin
  p:=0;
  For i:=1 To n Do
  Begin
   D[i]:=-25+Random(51);
   If D[i]<0 Then p:=p+D[i];
  End;
  Writeln(p);
  Readln;
End.
```

**3.** Дан массив целых чисел, состоящий из 25 элементов. Заполнить его с клавиатуры. Найти:

- сумму элементов, имеющих нечетные индексы;
- подсчитать количество элементов массива, значения которых больше заданного числа A и кратны 5.

**4.** Найти номер первого отрицательного элемента, делящегося на 5 с остатком 2.

### Вариант № 3

**1.** Правильно ли описан массив A? Если нет, то что надо изменить?

```
Type odmyarray=Array[1..n+20] Of Integer;
Var A : odmyarray;
```

**2.** Что получится в результате выполнения программы?

```
Program Variant3;
Const n=17;
Type myarray=Array[1..n] Of Integer;
Var B : myarray;
    i : Byte; p : Integer;
Begin
  p:=0;
  For i:=1 To n Do
  Begin
   B[i]:=-35+Random(121);
   If B[i] Mod 10=0 Then p:=p+1;
  End;
  Writeln(p);
  Readln;
End.
```

**3.** Дан массив целых чисел, состоящий из 15 элементов. Заполнить его с клавиатуры. Найти:

- сумму положительных элементов, значения которых меньше 10;
- вывести индексы тех элементов, значения которых кратны 3 и 5.

**4.** Определить, есть ли пара соседних элементов с суммой, равной заданному числу.

### Вариант № 4

**1.** Правильно ли описан массив D? Если нет, то что надо изменить?

```
Type odm=Array[-n..n] Of Integer;
Var D : odm;
```

**2.** Что получится в результате выполнения программы?

```
Program Variant4;
Const n=25;
Type myarray=Array[1..n] Of Integer;
Var A : myarray;
    i : Byte; p : Integer;
Begin
  p:=0;
  For i:=1 To n Do
  Begin
   A[i]:=-50+Random(151);
   If A[i]<=10 Then p:=p+A[i];
  End;
  Writeln(p);
  Readln;
End.
```

**3.** Дан массив целых чисел, состоящий из 10 элементов. Заполнить его с клавиатуры. Найти:

- удвоенную сумму положительных элементов;
- вывести индексы тех элементов, значения которых больше значения предыдущего элемента (начиная со второго).

**4.** Определить, есть ли 2 пары соседних элементов с одинаковыми знаками.

### Вариант № 5

**1.** Правильно ли описан массив A? Если нет, то что надо изменить?

```
Type myarray=Array[0..-n] Of Integer;
Var A : myarray;
```

**2.** Что получится в результате выполнения программы?

```
Program Variant5;
Const n=12;
Type myarray=Array[1..n] Of Integer;
Var C : myarray;
    i : Byte; p : Integer;
Begin
  For i:=1 To n Do
  Begin
   C[i]:=-25+Random(71);
   If C[i] Mod 3=0 Then p:=p+1;
  End;
  Writeln(p);
  Readln;
End.
```

**3.** Дан массив целых чисел, состоящий из 30 элементов. Заполнить его с клавиатуры. Найти:

- сумму отрицательных элементов;
- количество тех элементов, значения которых положительны и не превосходят заданного числа A.

**4.** Найти номер последней пары соседних элементов с разными знаками.

## 2.8.2. Одномерные массивы. Работа с элементами

### Контрольная работа № 2

#### Вариант № 1

**1.** Заменить максимальный по модулю отрицательный элемент массива нулем.

**2.** Заменить первые k элементов массива на противоположные по знаку.

**3.** Из элементов массива C сформировать массив A той же размерности по правилу: если номер i элемента четный, то $A_i=C_i^2$, если нечетный, то $A_i=2C_i$.

#### Вариант № 2

**1.** Заменить минимальный по модулю положительный элемент массива нулем.

**2.** Заменить элементы массива с k1-го по k2-й на те же элементы в обратном порядке.

**3.** Из элементов массива A сформировать массив D той же размерности по правилу: первые 10 элементов находятся по формуле $D_i=A_i+i$, остальные — по формуле $D_i=A_i-i$.

#### Вариант № 3

**1.** Заменить первый отрицательный элемент массива нулем.

**2.** Умножить все элементы массива, кратные 3, на третий элемент массива.

**3.** Из элементов массива P сформировать массив M той же размерности по правилу: если номер четный, то $M_i=i*P_i$, если нечетный, то $M_i=-P_i$.

#### Вариант № 4

**1.** Заменить максимальный элемент массива на противоположный по знаку.

**2.** Заменить нулями элементы массива между минимальным и максимальным, кроме их самих.

**3.** Из элементов массива C сформировать массив A той же размерности по правилу: элементы с 3-го по 12-й находятся по формуле $A_i=-C_i^2$, все остальные находятся по формуле $A_i=C_i-1$.

#### Вариант № 5

**1.** Заменить первый элемент массива, кратный 5, нулем.

**2.** Заменить элементы массива с нечетными номерами на квадраты их номеров.

**3.** Из элементов массива D сформировать массив A той же размерности по правилу: если номер четный, то значение элемента находится по формуле $A_i=D_i^2$, если нечетный, то по формуле $A_i=D_i/i$.

#### Вариант № 6

**1.** Заменить последний положительный элемент массива на второй элемент массива.

**2.** Разделить все элементы массива с четными номерами на первый элемент (первый элемент отличен от 0).

**3.** Из элементов массива C сформировать массив A той же размерности по правилу: если номер четный, то значение элемента находится по формуле $A_i=C_i^2$, если нечетный, то по формуле $A_i=2C_i$.

## 2.8.3. Одномерные массивы. Удаление, вставка и перестановка элементов

### Контрольная работа N º 3

#### Вариант № 1

Дан массив целых чисел из 15 элементов, заполненный случайным образом числами из промежутка [−20, 50].

**1.** Удалить из него все элементы, в записи которых есть цифра 5.

**2.** Вставить число k после всех элементов, кратных своему номеру (k вводится с клавиатуры).

**3.** Поменять местами первый положительный и последний отрицательный элементы.

#### Вариант № 2

Дан массив целых чисел из 10 элементов, заполненный случайным образом числами из промежутка [−40, 30].

**1.** Удалить из него все элементы, которые состоят из одинаковых цифр (включая однозначные числа).

**2.** Вставить число k перед всеми элементами, в которых есть цифра 1 (k вводится с клавиатуры).

**3.** Переставить первые три и последние три элемента местами, сохраняя порядок их следования.

#### Вариант № 3

Дан массив целых чисел из 12 элементов, заполненный случайным образом числами из промежутка [−10, 60].

**1.** Удалить из него все элементы, в которых последняя цифра четная, а само число делится на нее.

**2.** Вставить элемент со значением k до и после всех элементов, заканчивающихся на цифру k (k вводится с клавиатуры).

**3.** Переставить элементы массива следующим образом: a[1], a[12], a[2], a[11], ..., a[5], a[8], a[6], a[7].

#### Вариант № 4

Дан массив целых чисел из 25 элементов, заполненный случайным образом числами из промежутка [−35, 75].

**1.** Удалить из него все элементы, первая цифра которых четная.

**2.** Вставить число k1 после всех элементов, больших заданного числа, а число k2 — перед всеми элементами, кратными 3 (k1 и k2 вводятся с клавиатуры).

**3.** Перенести первые k элементов в конец: a[k+1], a[k+2], ..., a[n], a[1], a[2], ..., a[k].

**Вариант № 5**

Дан массив целых чисел из 20 элементов, заполненный случайным образом числами из промежутка [−45, 95].

**1.** Удалить из него все элементы, кратные 7 и принадлежащие промежутку [a; b] (a и b вводятся с клавиатуры).

**2.** Вставить число k между всеми соседними элементами, которые имеют одинаковые знаки (k вводится с клавиатуры).

**3.** Переставить в обратном порядке часть массива между элементами с номерами k1 и k2, включая их.

## 2.8.4. Двумерные массивы. Работа с элементами

### Контрольная работа № 4

**Вариант № 1**

**1.** Дан двумерный массив размерностью 5×6, заполненный целыми числами, введенными с клавиатуры. Сформировать одномерный массив, каждый элемент которого равен произведению четных положительных элементов соответствующего столбца.

**2.** Дан двумерный массив размером n×m, заполненный случайным образом. Определить, есть ли в данном массиве строка, в которой ровно два отрицательных элемента.

**3.** Заполнить массив размерностью 7×7 по правилу:

```
1 0 0 0 0 0 1
0 1 0 0 0 1 0
0 0 1 0 1 0 0
0 0 0 1 0 0 0
0 0 1 0 1 0 0
0 1 0 0 0 1 0
1 0 0 0 0 0 1
```

**Вариант № 2**

**1.** Дан двумерный массив размерностью 4×6, заполненный целыми числами, введенными с клавиатуры. Сформировать одномерный массив, каждый элемент которого равен количеству элементов соответствующей строки, бóльших данного числа.

**2.** Дан двумерный массив размером n×m, заполненный случайными числами. Определить, есть ли в данном массиве столбец, в котором имеются одинаковые элементы.

**3.** Заполнить массив размерностью 7×7 по правилу:

```
1 1 1 1 1 1 1
0 1 1 1 1 1 0
0 0 1 1 1 0 0
0 0 0 1 0 0 0
0 0 1 1 1 0 0
0 1 1 1 1 1 0
1 1 1 1 1 1 1
```

**Вариант № 3**

**1.** Дан двумерный массив размерностью 5×6, заполненный целыми числами, введенными с клавиатуры. Сформировать одномерный массив, каждый элемент которого равен наибольшему по модулю элементу соответствующего столбца.

**2.** Дан двумерный массив размером n×m, заполненный случайными числами. Определить, есть ли в данном массиве строка, в которой имеются два элемента массива, имеющие наибольшие значения.

**3.** Заполнить массив размерностью 6×6 по правилу:

```
1 2 3 4 5 6
2 3 4 5 6 1
3 4 5 6 1 2
4 5 6 1 2 3
5 6 1 2 3 4
6 1 2 3 4 5
```

**Вариант № 4**

**1.** Дан двумерный массив размерностью 4×5, заполненный целыми числами, введенными с клавиатуры. Сформировать одномерный массив, каждый элемент которого равен количеству отрицательных элементов, кратных 3 или 5, соответствующей строки.

**2.** Дан двумерный массив размером n×m, заполненный случайными числами. Определить, есть ли в данном массиве столбец, в котором равное количество положительных и отрицательных элементов.

**3.** Заполнить массив размерностью 6×6 по правилу:

```
1   1   1   1    1    1
1   2   3   4    5    6
1   3   6  10   15   21
1   4  10  20   35   56
1   5  15  35   70  126
1   6  21  56  126  252
```

**Вариант № 5**

**1.** Дан двумерный массив размерностью 6×5, заполненный целыми числами, введенными с клавиатуры. Сформировать одномерный массив, каждый элемент которого равен первому четному элементу соответствующего столбца, если такого нет, то равен нулю.

**2.** Дан двумерный массив размером n×m, заполненный случайными числами. Определить, есть ли в данном массиве строка, содержащая больше положительных элементов, чем отрицательных.

**3.** Заполнить массив размерностью 7×7 по правилу:

```
1 0 0 1 0 0 1
0 1 0 1 0 1 0
0 0 1 1 1 0 0
1 1 1 1 1 1 1
0 0 1 1 1 0 0
0 1 0 1 0 1 0
1 0 0 1 0 0 1
```

## 2.8.5. Двумерные массивы. Работа с элементами, вставка, удаление и перестановка строк

### Контрольная работа № 5

**Вариант № 1**

Дан двумерный массив размером 5×6, заполненный случайным образом.

**1.** Заменить максимальный элемент каждой строки на противоположный.

**2.** Вставить после столбцов, содержащих максимальный элемент массива, столбец из нулей.

**3.** Удалить среднюю строку.

**4.** Поменять местами средние столбцы.

**Вариант № 2**

Дан двумерный массив размером 8×7, заполненный случайным образом.

**1.** Заменить все элементы первых трех столбцов на их квадраты.

**2.** Вставить между средними строками первую строку.

**3.** Удалить все столбцы, в которых первый элемент больше последнего.

**4.** Поменять местами средние строки с первой и последней.

**Вариант № 3**

Дан двумерный массив размером 5×8, заполненный случайным образом.

**1.** Заменить в каждой строке все равные симметрично расположенные элементы на нули.

**2.** Вставить перед всеми строками, первый элемент которых делится на 3, строку из нулей.

**3.** Удалить самый левый столбец, в котором встретится четный отрицательный элемент.

**4.** Поменять местами средние столбцы со вторым и предпоследним.

**Вариант № 4**

Дан двумерный массив размером 6×7, заполненный случайным образом.

**1.** Заменить максимальный элемент каждой строки на противоположный.

**2.** Вставить после столбцов, содержащих максимальный элемент массива, столбец из нулей.

**3.** Удалить все столбцы, в которых первый элемент больше заданного числа A.

**4.** Поменять местами средние строки.

## 2.8.6. Строковый тип. Множественный тип

### Контрольная работа № 6

**Вариант № 1**

**1.** Дана последовательность слов. Напечатать все слова, предварительно заменив во всех словах первую букву заглавной.

**2.** Вывести общие русские буквы трех предложений.

**3.** Решить ребус:

```
  ОДИН
  ОДИН
+ ОДИН
  ОДИН
  ОДИН
-------
  ПЯТЬ
```

**Вариант № 2**

**1.** Дана последовательность слов. Напечатать все слова, предварительно удалив в словах наибольшей длины среднюю (средние) буквы.

**2.** Вывести из трех предложений русские буквы, которые встречаются только один раз (то есть такие, которые есть только в одном из них).

**3.** Решить ребус:

```
  VOLVO
- FIAT
-------
  MOTOR
```

**Вариант № 3**

**1.** Дана последовательность слов. Напечатать все слова, предварительно заменив в каждом слове первую встреченную букву “а” на “о”.

**2.** Вывести наибольшие цифры трех чисел (целых или действительных).

**3.** Решить ребус: $КУБ=(К+У+Б)^3$.

**Вариант № 4**

**1.** Дана последовательность слов. Проверить правильность написания сочетаний “жи”, “ши”, “ча”, “ща”, “чу” и “щу”. Исправить ошибки.

**2.** Даны три строки. Определить, можно ли из символов первых двух строк получить третью строку.

**3.** Решить ребус:

```
  ТРИ
+ ДВА
-------
  ПЯТЬ
```

## 2.8.7. Комбинированный тип данных

### Контрольная работа № 7

**Вариант № 1**

**1.** Дан текстовый файл, в котором хранятся данные об учениках класса: фамилия, имя, отчество, адрес (ули-

ца, дом, квартира) и домашний телефон (если есть). Вывести на экран строки с фамилиями, именами и адресами тех учеников, у которых нет домашнего телефона.

**2.** Имеется массив данных о работающих в фирме: фамилия, имя, отчество, адрес (улица, дом, квартира) и дата поступления на работу (месяц, год). Во второй массив записать только данные о тех из них, кто на сегодняшний день проработал уже не менее 5 лет.

**Вариант № 2**

**1.** Имеется текстовый файл, в котором хранятся данные об учениках нескольких школ: фамилия, имя, отчество, адрес (улица, дом, квартира), школа и класс. Вывести на экран строки с фамилиями, именами и адресами тех учеников, которые учатся в данной школе в старших классах (номер школы вводится с клавиатуры).

**2.** Имеется массив данных о клиентах пункта проката: фамилия, имя, отчество, адрес (улица, дом, квартира), название предмета, взятого напрокат (только одного). Во второй массив записать данные о клиентах, взявших телевизор.

**Вариант № 3**

**1.** Дан текстовый файл, в котором хранятся данные об учениках класса: фамилия, имя, отчество, дата рождения (число, месяц и год). Вывести на экран фамилии и имена тех учеников, у кого сегодня день рождения (сегодняшнюю дату вводить с клавиатуры).

**2.** Имеется массив данных о работающих на фабрике: фамилия, имя, отчество, адрес (улица, дом, квартира) и дата поступления на работу (месяц, год). Определить, есть ли в списке Ивановы (Иванов, Иванова), если есть, то вывести их адреса.

**Вариант № 4**

**1.** Имеется массив данных об учениках школы: фамилия, имя, адрес (улица, дом, квартира), класс. Записать все данные об учениках определенного класса во второй массив. Распечатать его, выделяя тех из них, кто живет на улице Ленина.

**2.** Имеется текстовый файл, в котором хранятся данные о расписании поездов: номер поезда, направление (откуда куда, например, Киров — Москва), время прибытия на станцию, время отправления (часы, минуты). Будем считать, что все поезда приходят каждый день. По данному времени определить, какие из поездов стоят сейчас на станции (время вводить с клавиатуры).

# 3. МЕТОДЫ СОРТИРОВКИ И ПОИСКА ДАННЫХ

## 3.1. АЛГОРИТМЫ СОРТИРОВКИ ИНФОРМАЦИИ

### 3.1.1. Повторение материала предыдущих глав

#### Вопросы для повторения

**1.** Что такое массив?

**2.** Как обратиться к элементу массива с заданным номером?

#### Пример

Даны две целочисленные переменные — x, y. Составить фрагмент программы, после выполнения которого значения этих переменных распределятся в порядке невозрастания.

*Решение*

Обмен значений переменных нужно производить лишь в том случае, если x < y. Для того чтобы не потерять начальное значение переменной x, введем дополнительную переменную t.

```
If x<y Then Begin t:=x; x:=y; y:=t; End;
```

#### Пример

Составить программу поиска максимального из трех введенных с клавиатуры чисел.

*Решение*

Пусть a, b, c — вводимые с клавиатуры числа, m — максимальное из них. Сначала предположим, что a — максимальное из чисел (т.е. m:=a). Затем сравним значение предполагаемого максимума со значениями переменных b и c. Если значение m окажется меньше, чем значение очередной переменной, то изменим значение максимума:

```
Begin
  Writeln('введите значения a, b, c');
  Readln(a,b,c);
  m:=a;
  If m<b Then m:=b;
  If m<c Then m:=c;
  Writeln('максимальное число - ',m)
End.
```

#### Пример

Дан массив a, состоящий из 10 элементов. Составить программу поиска максимального элемента массива.

*Решение*

Используем идею решения предыдущей задачи. Перед началом поиска выберем условно в качестве максимального первый элемент массива: m:=a[1]. Затем каждый элемент массива сравним со значением переменной m. Если он окажется больше, то изменим значение m. После сравнения со всеми элементами массива переменная m содержит значение максимального элемента массива.

```
Begin
  m:=a[1];
  For i:=2 To 10 Do
  If m<a[i] Then m:=a[i];
  Writeln('максимальный элемент массива - ', m);
End;
```

**Пример**

Написать программу вставки последнего элемента массива после первого отрицательного элемента этого же массива.

*Решение*

Запомним значение последнего элемента, найдем номер i первого отрицательного элемента, все элементы, стоящие после i-го, сдвинем на 1 позицию вправо, а на освободившееся i+1-е место запишем значение последнего элемента.

```
Program Example_56;
Const a: Array[1..10] Of Integer=
        (1,3,5,2,-7,4,-11,0,9,12);
Var i,k,m: Integer;
Begin
  Writeln('исходный массив:');
  {вывод исходного массива}
  For i:=1 To 10 Do Write(a[i]:3); Writeln;
  i:=1;
  {поиск первого отрицательного элемента}
  While (i<=10)And(a[i]>0) Do Inc(i);
  {сдвиг элементов, стоящих после первого
  i-го вправо на 1 позицию}
  m:=a[10];
  For k:=10 Downto i+2 Do a[k]:=a[k-1];
  a[i+1]:=m;
  For i:=1 To 10 Do Write(a[i]:3);
  {вывод обработанного массива}
End.
```

**Задания**

**1.** Модифицировать программу для поиска минимального элемента массива.

**2.** Дополнить программу так, чтобы в результате было получено не только значение максимального элемента, но и его индекс.

**3.** Составить программу, в результате выполнения которой максимальный и минимальный элементы массива меняются местами.

Для решения многих задач удобно сначала упорядочить данные по определенному признаку. Процесс упорядочения заданного множества объектов по заданному признаку называется **сортировкой.**

Данные, например, элементы массива, можно отсортировать:

- по возрастанию — каждый следующий элемент больше предыдущего: a[1]<a[2]<... <a[n];
- по неубыванию — каждый следующий элемент не меньше предыдущего: a[1]≤ a[2] ≤... ≤ a[n];
- по убыванию — каждый следующий элемент меньше предыдущего: a[1]> a[2]> ... >a[n];
- по невозрастанию — каждый следующий элемент не больше предыдущего: a[1]≥ a[2]≥ ... ≥a[n].

**Задача**

Составить программу сортировки элементов массива a, состоящего из n элементов, в порядке их возрастания.

Существует довольно много различных методов сортировки, отличающихся друг от друга степенью эффективности, под которой понимается количество сравнений и количество обменов, произведенных в процессе сортировки. Рассмотрим подробно некоторые из указанных методов.

## 3.1.2. Сортировка методом простого выбора

Этот метод сортировки обычно применяется для массивов, не содержащих повторяющихся элементов. Для достижения поставленной цели можно действовать следующим образом:

1) выбрать максимальный элемент массива;

2) поменять его местами с последним элементом (после этого наибольший элемент будет стоять на своем месте);

3) повторить пп. 1—2 с оставшимися n-1 элементами, то есть рассмотреть часть массива, начиная с первого элемента до предпоследнего, найти в ней максимальный элемент и поменять его местами с предпоследним (n-1)-м элементом массива и так далее, пока не останется один элемент, уже стоящий на своем месте.

Всего потребуется n-1 раз выполнить эту последовательность действий. В процессе сортировки будет увеличиваться отсортированная часть массива, а неотсортированная, соответственно, уменьшаться.

Рассмотрим этот процесс на примере. Пусть исходный массив a состоит из 10 элементов и имеет вид:

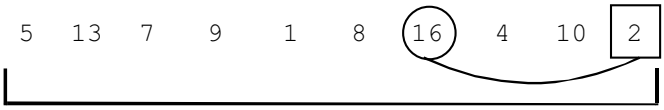
5 13 7 9 1 8 16 4 10 2

После сортировки массив должен выглядеть так:

1 2 4 5 7 8 9 10 13 16

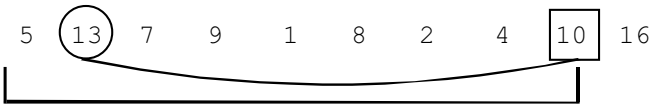
Процесс сортировки представлен ниже. Максимальный элемент текущей части массива заключен в кружок, а элемент, с которым происходит обмен, — в квадратик. Скобкой помечена рассматриваемая часть массива.

*1-й шаг:* рассмотрим весь массив и найдем в нем максимальный элемент — 16 (он стоит на седьмом месте); поменяем его местами с последним элементом.



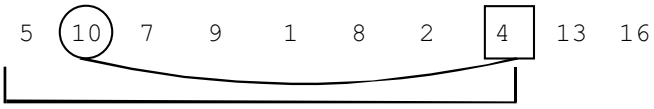

Максимальный элемент помещен на свое место.

*2-й шаг:* рассмотрим часть массива с первого до девятого элемента. Максимальный элемент этой части стоит на втором месте и имеет значение 13. Поменяем его местами с последним элементом этой части.



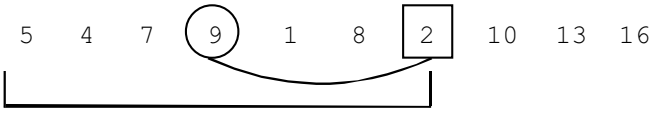

Отсортированная часть массива состоит теперь уже из двух элементов.

*3-й шаг:* снова уменьшим рассматриваемую часть массива на один элемент. Здесь нужно поменять местами второй элемент (его значение 10) и последний элемент этой части (его значение 4).

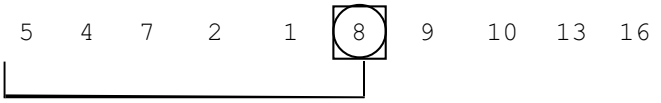



В отсортированной части массива теперь стало 3 элемента.

*4-й шаг:* аналогично.

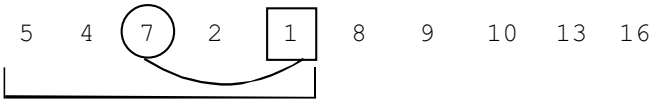



*5-й шаг:* максимальный элемент этой части массива является последним в ней, поэтому его нужно оставить на своем месте.




Далее действуем аналогично.

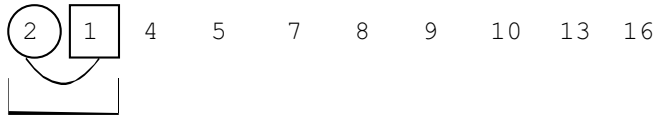
*6-й шаг:*




*7-й шаг:*


*8-й шаг:*


*9-й шаг:*




**Итог: 1 2 4 5 7 8 9 10 13 16**

Для программной реализации этого процесса необходимо организовать цикл по длине рассматриваемой части массива, которая изменяется от n до 2. В качестве начального значения максимума разумно взять значение последнего элемента рассматриваемой части.

Теперь можем записать алгоритм сортировки:

```
For i:=n Downto 2 Do
Begin
  найти максимальный элемент из a[1], ..., a[i];
  запомнить его индекс в переменной k;
  если i<>k, поменять местами a[i] и a[k]
End;
```

Опишем этот алгоритм подробно в виде процедуры:

```
Procedure sorting1(Var a:ar);
  {поскольку в процессе работы процедуры
  массив изменится, формальный параметр a
  описывается как параметр-переменная}
Var i, j, k: Integer;
    m: Integer;
   {значение максимального элемента
    рассматриваемой части массива}
Begin
  For i:=10 Downto 2 Do
  {цикл по длине рассматриваемой части массива}
  Begin
   {поиск максимального элемента и его
    номера в текущей части массива}
   k:=i; m:=a[i];
   {начальные значения макс. элемента
    и его индекса в рассматриваемой части
    массива }
   For j:=2 To i-1 Do
    If a[j]>m Then Begin k:=j; m:=a[k] End;
    If k<>i Then
     Begin {перестановка элементов}
      a[k]:=a[i];
      a[i]:=m
     End
  End
End;
```

***Решение задач***

**1.** Подсчитать количество произведенных сравнений.

**2.** Подсчитать количество произведенных перестановок.

**3.** Изменить процедуру сортировки так, чтобы значение параметра `i` с каждым шагом увеличивалось.

**4.** Изменить процедуру так, чтобы сортировка производилась по

a) убыванию элементов;

b) невозрастанию элементов;

c) неубыванию элементов.

**5.** Модифицировать программу — добавить проверку рассматриваемой части массива на упорядоченность: если она упорядочена, то сортировку завершить.

**6.** Отсортировать четные по значению элементы массива с помощью простого выбора.

**7.** Отсортировать с помощью простого выбора элементы массива, стоящие на нечетных местах.

**8.** Отсортировать отрицательные элементы массива с помощью простого выбора.

## 3.1.3. Сортировка методом простого обмена

### Повторение

**1.** Что понимается под сортировкой массива?

**2.** Чем отличается сортировка по убыванию от сортировки по невозрастанию?

**3.** Сформулировать идею сортировки массива методом простого выбора.

Сортировка методом простого обмена может быть применена для любого массива. Этот метод заключается в последовательных просмотрах массива слева направо (от начала к концу) и обмене местами **соседних** элементов, расположенных "неправильно", то есть таких, что `i<j`, а `a[i]>a[j]`. Опишем этот метод подробнее.

Начнем просмотр с первой пары элементов (`a[1]` и `a[2]`). Если первый элемент этой пары больше второго, то меняем их местами, иначе оставляем без изменения. Затем берем вторую пару элементов (`a[2]` и `a[3]`), если `a[2]>a[3]`, то меняем их местами; и так далее. На первом шаге будут просмотрены все пары элементов массива `a[i]` и `a[i+1]` для `i` от `1` до `n-1`. В результате максимальный элемент массива переместится в конец массива.

Поскольку самый большой элемент находится на своем месте, рассмотрим часть массива без него, то есть с первого до `(n-1)`-го элемента. Повторим предыдущие действия для этой части массива, в результате чего второй по величине элемент массива переместится на последнее место рассматриваемой части массива, то есть на `(n-1)`-е место во всем массиве. Эти действия продолжают до тех пор, пока количество элементов в текущей части массива не уменьшится до двух. В этом случае необходимо выполнить последнее сравнение и упорядочить последние два элемента. При сортировке выполняется `n-1` просмотр массива.

### Пример

Отсортируем по возрастанию методом простого обмена массив из 5 элементов: `5 4 8 2 9`.

Длина текущей части массива — `n-k+1`, где `k` — номер просмотра, `i` — номер проверяемой пары; номер последней пары — `n-k`. За вертикальной чертой располагаются отсортированные элементы.

***Первый просмотр:*** рассматривается весь массив.

**i=1** `5 > 4 8 2 9`

обмен

**i=2** `4 5 < 8 2 9`

нет обмена

**i=3** `4 5 8 > 2 9`

обмен

**i=4** `4 5 2 8 < 9`

нет обмена

**9 стоит на своем месте.**

***Второй просмотр:*** рассматриваем часть массива с первого до четвертого элемента.

**i=1** `4 < 5 2 8 | 9`

нет обмена

**i=2** `4 5 > 2 8 | 9`

обмен

**i=3** `4 2 5 < 8 | 9`

нет обмена

**8 стоит на своем месте.**

***Третий просмотр:*** рассматриваемая часть массива содержит три первых элемента.

**i=1** `4 > 2 5 | 8 9`

обмен

**i=2** `2 4 < 5 | 8 9`

нет обмена

**5 стоит на своем месте.**

***Четвертый просмотр:*** рассматриваем последнюю пару.

**i=1** `2 < 4 | 5 8 9`

нет обмена

**4 стоит на своем месте.**

Для самого маленького элемента (2) остается только одно место — первое.

Итак, наш массив отсортирован по возрастанию элементов методом простого обмена. Этот метод также называют методом "пузырька". Название это происходит от образной интерпретации, при которой в процессе выполнения сортировки более "легкие" элементы (элементы с заданным свойством) мало-помалу всплывают на "поверхность".

Опишем процедуру "пузырьковой" сортировки.

```
Procedure sorting2(Var a:ar);
Var k,i,t:Integer;
{k - номер просмотра (изменяется от 1 до n-1)
 i - номер рассматриваемой пары
 t - промежуточная переменная для
     перестановки местами элементов}
Begin
  For k:=1 To n-1 Do
  {цикл по номеру просмотра}
    For i:=1 To n-k Do
     If a[i]>a[i+1] Then
     {перестановка элементов}
     Begin
       t:=a[i];
       a[i]:=a[i+1];
       a[i+1]:=t
     End
End;
```

***Решение задач***

**1.** Подсчитать количество сравнений при "наилучшем" и "наихудшем" расположении элементов. Сравнить с количеством сравнений при сортировке методом простого выбора.

**2.** Подсчитать количество выполненных перестановок в данном случае, в самом лучшем случае, в самом худшем случае. Сравнить с методом простого выбора.

**3.** В нашем примере последний проход не влияют на порядок элементов, так как массив уже оказывается отсортированным. Следовательно, можно улучшить наш алгоритм, если запоминать, производились ли перестановки элементов в процессе очередного прохода. Если их не было, то сортировку можно закончить. Модифицировать процедуру с учетом этой возможности улучшения.

**4.** Если известен не только факт последнего обмена, но и его место, то нетрудно заметить, что все пары соседних элементов, расположенные правее этого места, уже находятся в нужном порядке. Поэтому просмотр можно закончить на этом индексе, а не продолжать до конца. Модифицировать процедуру с учетом этой возможности улучшения.

**5.** Описать процедуру "пузырьковой" сортировки по убыванию.

## 3.1.4. Сортировка методом прямого включения

Сортировка методом прямого включения, так же как и сортировка методом простого выбора, обычно применяется для массивов, не содержащих повторяющихся элементов.

Сортировка методом прямого включения, как и все описанные выше, производится по шагам. На k-м шаге считается, что часть массива, содержащая первые k−1 элементов, уже упорядочена, то есть

$$a[1] \le a[2] \le \ldots \le a[k-1].$$

Далее необходимо взять k-й элемент и подобрать для него такое место в отсортированной части массива, чтобы после его вставки упорядоченность не нарушилась, то есть надо найти такое $j$ ($1 \le j \le k-1$), что $a[j] \le a[k] < a[j+1]$. Затем надо вставить элемент a[k] на найденное место.

С каждым шагом отсортированная часть массива увеличивается. Для выполнения полной сортировки потребуется выполнить n−1 шаг.

Рассмотрим этот процесс на примере. Пусть требуется отсортировать массив из 10 элементов по возрастанию методом прямого включения:

*1-й шаг*

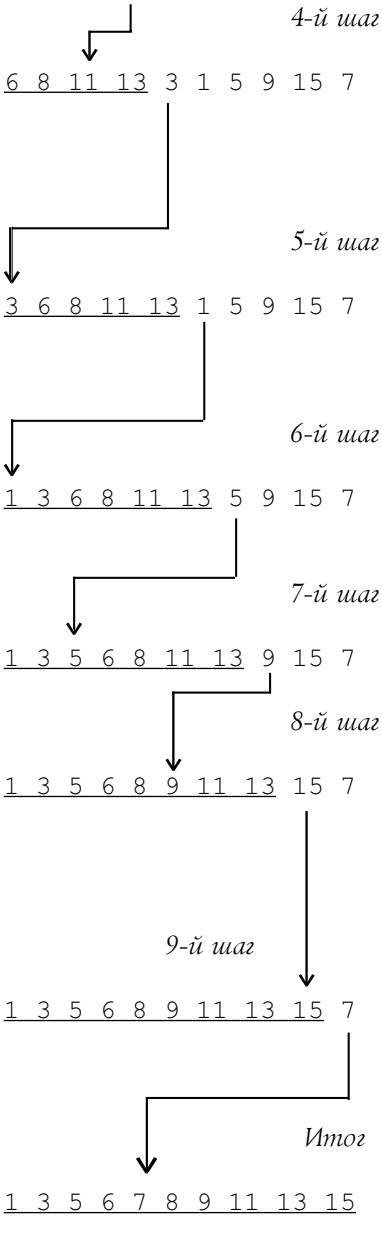
<u>13</u> 6 8 11 3 1 5 9 15 7 — Рассматриваем часть массива из одного элемента (13). Нужно вставить в нее второй элемент массива (6) так, чтобы упорядоченность сохранилась. Так как 6<13, вставляем 6 на первое место. Отсортированная часть массива содержит два элемента (6 13).

*2-й шаг*

<u>6 13</u> 8 11 3 1 5 9 15 7 — Возьмем третий элемент массива (8) и подберем для него место в упорядоченной части массива. 8>6 и 8<13, следовательно, его нужно вставить на второе место.

*3-й шаг*

<u>6 8 13</u> 11 3 1 5 9 15 7 — Следующий элемент — 11. Он записывается в упорядоченную часть массива на третье место, так как 11>8, но 11<13.




Сейчас можно коротко описать фрагмент алгоритма сортировки методом прямого включения:

```
For k:=2 To n Do
{так как начинаем сортировку с поиска
подходящего места для a[2],
i изменяется от 2 до n}
Begin
  x:=a[k];
  {вставить x на подходящее место
   в a[1],...,a[k]}
End;
```

Осталось ответить на вопрос, как осуществить поиск подходящего места для элемента x. Поступим следующим образом: будем просматривать элементы, расположенные левее x (то есть те, которые уже упорядочены), двигаясь к началу массива. Нужно просматривать элементы a[j], j изменяется от k-1 до 1. Такой просмотр должен закончиться при выполнении одного из следующих условий:

- найден элемент a[j]<x, что говорит о необходимости вставки x между a[j-1] и a[j];
- достигнут левый конец упорядоченной части массива, следовательно, нужно вставить x на первое место.

До тех пор, пока одно из этих условий не выполнится, будем смещать просматриваемые элементы на 1 позицию вправо, в результате чего в отсортированной части будет освобождено место под x.

Учитывая это, опишем процедуру сортировки:

```
Procedure Sorting3(Var a:ar);
Var k,j,x:Integer;
Begin
  For k:=2 To n Do
  Begin
   x:=a[k]; j:=k-1;
   While (j>0) And (x>=a[j]) Do
    Begin
     a[j+1]:=a[j];
     Dec(j)
    End;
    a[j+1]:=x
  End
End;
```

***Решение задач***

**1.** Подсчитать минимальное, максимальное, среднее количество выполненных сравнений.

**2.** Будет ли сортировка выполняться правильно, если в заголовке цикла while указать условие x>a[j]?

**3.** Изменить процедуру, чтобы сортировка производилась по возрастанию элементов.

## 3.1.5. Сортировка методом слияний

Существует еще один метод сортировки элементов массива, эффективность которого сравнительно велика, — метод слияний.

Этот метод состоит в разбиении данного массива на несколько частей, которые сортируются по отдельности и впоследствии "сливаются" в одну.

Пусть массив a[1..n] разбивается на части длиной k, тогда первая часть — a[1], a[2], ..., a[k], вторая — a[k+1], a[k+2], ..., a[2k]; и так далее. Если n не делится на k, то в последней части будет менее k элементов.

После того как массивы-части упорядочены, можно объединить их в упорядоченные массивы-части, состоящие не более чем из 2k элементов, которые далее объединить в упорядоченные массивы длиной не более 4k, и так далее, пока не получится один упорядоченный массив.

Таким образом, чтобы получить отсортированный массив этим методом, нужно многократно "сливать" два упорядоченных отрезка массива в один упорядоченный отрезок. При этом другие части массива не затрагиваются.

**Задача**

Дан массив a[1..n] целых чисел и числа k, m, n, такие, что k≤m≤n. Описать процедуру, которая сливает отрезки массива a[k+1], ...,a[m] и a[m+1], ..., a[n] в упорядоченный отрезок a[k+1], ..., a[n].

Основная идея решения состоит в сравнении очередных элементов каждой части, выяснении, какой из них меньше, переносе его в массив c, который является вспомогательным, и продвижении по тому массиву-части, из которого взят элемент. Когда одна из частей будет пройдена до конца, останется только перенести в c оставшиеся элементы второй части, сохраняя их порядок.

Рассмотрим пример.

Пусть первая часть (часть a) состоит из 5 элементов:

... 3 5 8 11 16...,

а вторая часть (часть b) — из 8 элементов:

... 1 5 7 9 12 13 18 20...

По условию оба массива-части упорядочены. Нужно сформировать массив c, который будет содержать 13 элементов.

Введем обозначения:

i — номер обрабатываемого элемента части a;

j — номер обрабатываемого элемента части b;

q — номер заполняемого элемента массива c.

Поскольку заполнение массива c ведется с его начала, на первом шаге значение q=1.

*1-й шаг:* на первое место в массиве c претендуют a[k+1]=3 и b[m+1]=1 (i=k+1, j=m+1). Так как 1<3, в массив c нужно занести 1 и перейти к следующему элементу части b, то есть увеличить j на 1 (значение j станет равно m+2), а также увеличить на 1 значение q (значением q будет 2).

*2-й шаг:* теперь нужно сравнить a[k+1]=3 и b[m+2]=5. 3<5, следовательно, на второе место в c надо занести a[k+1]=3. Затем нужно увеличить на 1 значения i и q.

*3-й шаг:* на третье место массива c претендуют два одинаковых элемента — a[k+2]=5 и b[m+2]=5. В этом и подобных случаях договоримся заносить в c первым элемент из части a. Таким образом, c[3]=a[k+2], а значение q=4, i=k+3, j остается равным m+2.

*4—11-й шаги:* рассуждая аналогичным образом, нужно занести в c элементы 5, 7, 8, 9, 11, 12, 13, 16. После выполнения 11-го шага первая часть будет занесена в c полностью, значение j равно m+7, значение q равно 12.

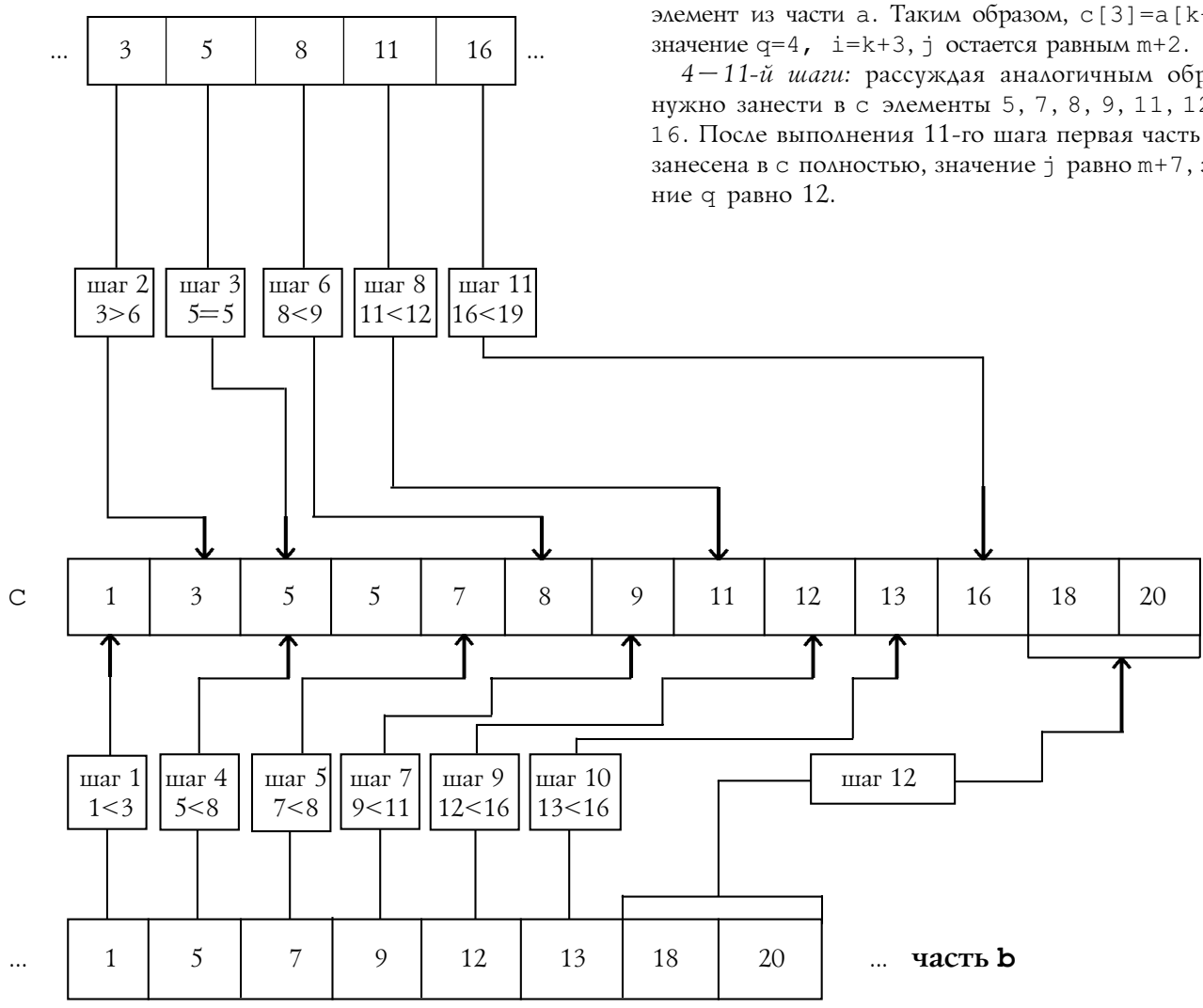



*Рис.* 9

*12-й шаг:* так как часть a закончилась, а "хвост" части b упорядочен по условию, то двенадцатым элементом массива c будет первый элемент "хвоста" b, то есть b[m+7]=18.

*13-й шаг:* последним элементом массива c будет последний элемент b — 20.

На *рис.* 9 схематично изображен процесс заполнения массива c.

Сейчас можем описать процедуру слияния двух упорядоченных массивов размерностей m–k и n–m соответственно в третий упорядоченный массив, размерности n–k.

```
Procedure sorting4(k,m,n:Integer);
Var i,j: integer
Begin
i:=k+1; j:=m+1; k:=1;
While (i<=m) And (j<=n) Do
  {пока не закончилась хотя бы одна часть}
Begin
  If a[i]<=b[j] Then
  Begin c[k]:=a[i]; Inc(i); End
  Else Begin c[k]:=b[j]; Inc(j); End;
  Inc(k);
End;
{один из массивов-частей обработан полностью}
{осталось перенести в с остаток другого
массива-части}
While i<=m Do
  Begin c[k]:=a[i]; Inc(i); Inc(k) End;
While j<=n Do
  Begin c[k]:=b[j]; Inc(j); Inc(k) End
End;
```

Далее остается лишь переписать результат слияния рассматриваемых частей — массив c — обратно в массив a.

***Решение задач***

Написать полную программу сортировки массива методом слияний.

## 3.1.6. Обменная сортировка с разделением (сортировка Хоара)

Сортировка методом простого обмена (методом "пузырька") является в среднем самой неэффективной. Это обусловлено самой идеей метода, который требует в процессе сортировки сравнивать и обменивать между собой только соседние элементы. Можно существенно улучшить метод сортировки, основанный на обмене. Это улучшение приводит к самому лучшему на сегодняшний день методу сортировки массивов, который можно назвать обменной сортировкой с разделением. Он основан на сравнениях и обменах элементов, стоящих на возможно больших расстояниях друг от друга. Предложил этот метод Ч.А.Р. Хоар в 1962 году. Поскольку производительность этого метода впечатляюща, автор назвал его "быстрой" сортировкой.

**Идея метода**

1. В исходном неотсортированном массиве выбрать некоторый элемент x (барьерный элемент).

2. Переставить элементы массива таким образом, чтобы слева от x оказались элементы массива, меньшие или равные x, а справа — элементы массива, большие x.

Пусть при этом элемент x попадет в позицию k, тогда массив будет иметь вид:

(a[1],a[2], ..., a[k-1]),a[k],(a[k+1], ..., a[n]).

Каждый из элементов a[1], a[2], ..., a[k-1] меньше либо равен a[k], каждый из элементов a[k+1], ..., a[n] больше a[k]. Отсюда можно сделать вывод, что элемент a[k] стоит на своем месте. А исходный массив при этом разделится на две неотсортированные части, барьером между которыми является элемент a[k].

Для дальнейшей сортировки необходимо применить пункты 1 и 2 для каждой из этих частей. И так до тех пор, пока не останутся подмассивы, состоящие из одного элемента, то есть пока не будет отсортирован весь массив.

Рассмотрим применение метода "быстрой" сортировки на примере.

Исходный массив состоит из 8 элементов:

8 12 3 7 19 11 4 16

В качестве барьерного элемента будем брать средний элемент массива.

Барьерный элемент — 7. Произведя необходимые перестановки для разделения, получим:

(4 3) 7 (12 19 11 8 16)

Теперь элемент со значением 7 стоит на своем месте. Далее сортируем подмассивы, элементы которых заключены в скобки. Этот процесс будем повторять до тех пор, пока не получим полностью отсортированный массив.

Левый подмассив:

```
(3) 4 7 (12 19 11 8 16)
3 4 7 (12 19 11 8 16)
```

Правый подмассив:

```
3 4 7 (8) 11 ( 19 12 16)
3 4 7 8 11 (19 12 16)
3 4 7 8 11 12 (19 16)
3 4 7 8 11 12 (16) 19
3 4 7 8 11 12 16 19
```

Массив отсортирован полностью.

Алгоритм "быстрой" сортировки можно определить как рекурсивную процедуру, параметрами которой являются нижняя и верхняя границы изменения индексов сортируемой части исходного массива:

```
Program Example_57;
Var a: Array[1..10] Of Integer;
Procedure Init;
{формирование массива из файла}
Var f: text;
```

```
   i: Integer;
Begin
  Assign(f,'c:\s.dat');
  Reset(f);
  For i:=1 To 10 Do Read(f,a[i]);
End;

Procedure Print;      {печать массива}
Var i: Integer;
Begin
  For i:=1 To 10 Do Write(a[i]:5);
  Writeln
End;

Procedure Quick_sorting(m,l: Integer);
Var i,j,x,w: Integer;
Begin
  i:=m; j:=l;
  x:=a[(m+l) Div 2];
  Repeat
  While a[i]<x Do Inc(i);
  While a[j]>x Do dec(j);
  If i<j Then
  Begin
   w:=a[i]; a[i]:=a[j]; a[j]:=w;
   Inc(i); dec(j);
  End
  Until i>j;
  If m<j Then quick_sorting(m,j);
  If i<l Then quick_sorting(i,l);
End;
Begin{основная программа}
  Writeln('массив:');
  init;print;
  quick_sorting(1,10);
  Writeln('отсортированный массив');
  print;
End.
```

### 3.1.7. Задания для контрольной работы

**1.** Написать программу сортировки элементов массива, имеющих нечетные индексы, методом простого выбора.

**2.** Написать программу сортировки нечетных элементов массива методом простого выбора.

**3.** Написать программу сортировки элементов массива, имеющих четные индексы, методом простых вставок.

**4.** Написать программу сортировки четных элементов массива методом простых вставок.

## 3.2. АЛГОРИТМЫ ПОИСКА ИНФОРМАЦИИ

### 3.2.1. Линейный поиск

Пусть имеется некоторый набор данных и требуется определить, где в этом наборе находится заданный элемент (и есть ли он в наборе). Такая задача называется *задачей поиска*. Большинство задач поиска сводится к поиску в таблице (массиве) элемента с заданным значением.

**Пример**

Написать программу поиска элемента x в массиве из n элементов. Значение элемента x вводится с клавиатуры.

**Решение**

Пусть имеются следующие описания:

```
Const n=10;     {размерность массива}
Var  a: Array[1..n] Of Integer;
{массив из n элементов целого типа}
x:Integer;{искомый элемент целого типа}
```

В данном случае известно только значение разыскиваемого элемента, никакой дополнительной информации о нем или о массиве, в котором его надо искать, нет. Проверка одного элемента не дает никакой информации об остальных. Поэтому для решения задачи разумно применить очевидный метод — последовательный просмотр массива и сравнение значения очередного рассматриваемого элемента с данным. Если значение очередного элемента совпадет с x, то запомним его номер в переменной k.

```
For i:=1 To n Do If a[i]=x Then k:=i;
```

Этот способ решения поставленной задачи, безусловно, приводит к цели, но обладает рядом существенных недостатков:

- если значение x встречается в массиве несколько раз, то найдено будет последнее из них;
- после того как нужное значение уже найдено, массив просматривается до конца, то есть всегда выполняется n сравнений.

Разумно прекратить просмотр сразу после обнаружения заданного элемента. Так как в этом случае число повторений неизвестно, необходимо использовать цикл с предусловием. В результате:

— либо будет найден искомый элемент, то есть найдется такой индекс i, что a[i]=x;

— либо будет просмотрен весь массив и искомый элемент не обнаружится.

Таким образом, поскольку нужно продолжать поиск до обнаружения искомого элемента или до конца массива, условие окончания цикла может выглядеть так:

```
While (i<=n) And (a[i]<>0) Do Inc(i);
```

**Примечание**. Необходимо обратить внимание на порядок простых условий в составном условии цикла. Здесь предполагается, что второе условие не проверяется, если результат логического выражения ясен после проверки первого условия. В противном случае возникнет отказ из-за выхода индекса за границы массива.

**Задание**

Оформить программу и проследить ее работу в режиме пошагового просмотра при различных значениях x.

Перед каждым увеличением индекса i выполняется следующее условие: для всех значений индекса, меньших i, совпадений нет, то есть после каждого шага та часть массива, где элемента x нет, увеличивается. Поскольку поиск заканчивается только в случае, когда i=n+1 или когда искомый элемент найден, то, если в массиве есть несколько элементов, совпадающих с элементом x, в результате работы программы будет найден первый из них, то есть элемент с наименьшим индексом.

**Задание**

Модифицировать программу для поиска элемента массива, равного x, с наибольшим индексом.

## 3.2.2. Линейный поиск с использованием барьера

Недостатком нашей программы является то, что в заголовке цикла записано сложное условие, которое проверяется перед каждым увеличением индекса, что замедляет поиск. Чтобы ускорить его, необходимо максимально упростить логическое выражение.

Постараемся оставить в заголовке цикла лишь простое условие a[i]<>x. Для этого используем следующий прием. В массив на (n+1)-е место запишем искомый элемент x, который назовем барьерным. Тогда если в процессе работы программы обнаружится такой индекс i, что a[i]=x, то элемент будет найден, а если условие a[i]=x выполнится только при i=n+1, то, значит, интересующего нас элемента в массиве нет. Таким образом, эта часть программы будет выглядеть так:

```
a[n+1]:=x; i:=1;
While a[i]<>x Do Inc(i);
```

При таком способе поиска в случае наличия в массиве нескольких элементов, удовлетворяющих заданному свойству, также будет найден элемент с наименьшим индексом.

**Задание**

Изменить программу так, чтобы был найден элемент с наибольшим индексом.

Таким образом, мы смогли сократить время поиска за счет сокращения времени проверки условия окончания работы. При данной постановке задачи других способов ускорения ее решения нет.

***Решение задач***

Подсчитать количество производимых сравнений
a) в лучшем случае;
b) в худшем случае;
c) в среднем.

## 3.2.3. Бинарный поиск

Как было отмечено выше, если никаких дополнительных сведений о массиве, в котором хранятся данные, нет, то ускорить поиск нельзя. Если же заранее известна некоторая информация о данных, среди которых ведется поиск, например, массив данных отсортирован, удается существенно сократить время работы, применяя другие методы поиска.

Одним из методов поиска, более эффективным, чем линейный, является бинарный (двоичный) поиск, называемый также методом половинного деления. При его использовании на каждом шаге область поиска сокращается вдвое. Рассмотрим применение этого метода для решения конкретной задачи.

**Задача**

Даны целое число x и массив a[1..n], отсортированный в порядке неубывания, то есть для любого k: 1≤k<n: a[k-1]≤a[k]. Найти такое i, что a[i]=x, или сообщить, что элемента x в массиве нет.

Идея бинарного метода состоит в том, чтобы проверить, является ли x средним элементом массива. Если да, то ответ получен. Если нет, то возможны два случая:

- x *меньше среднего элемента*, следовательно, в силу упорядоченности массива a можно исключить из рассмотрения все элементы массива, расположенные в нем правее среднего (так как они больше среднего элемента, который, в свою очередь, больше x), и применить этот метод к левой половине массива.
- x *больше среднего элемента*, следовательно, рассуждая аналогично, можно исключить из рассмотрения левую половину массива и применить этот метод к его правой части.

Средний элемент и в том, и в другом случае в дальнейшем не рассматривается. Таким образом, на каждом шаге отсекается та часть массива, где заведомо не может быть обнаружен элемент x.

**Рассмотрим пример**

Пусть x=6, а массив a состоит из 10 элементов:

3 5 6 8 12 15 17 18 20 25.

*1-й шаг:* найдем номер среднего элемента: $m = \left[\frac{1+10}{2}\right] = 5$. Так как 6<a[5], далее можем рассматривать только элементы, индексы которых меньше 5. Об остальных элементах можно сразу сказать, что они больше x вследствие упорядоченности массива, и среди них искомого элемента нет:

3 5 6 8 ~~**12** 15 17 18 20 25~~;

*2-й шаг:* рассматриваем лишь первые 4 элемента массива; находим индекс среднего элемента этой части: $m = \left[\frac{1+4}{2}\right] = 2$.

6>a[2], следовательно, первый и второй элементы из рассмотрения исключаем:

~~3~~ ~~**5**~~ 6 8 ~~12 15 17 18 20 25~~;

*3-й шаг:* рассматриваем два элемента; значение $m = \left[\frac{3+4}{2}\right] = 3$:

~~3~~ ~~5~~ ~~**6**~~ 8 ~~12 15 17 18 20 25~~;

a[3]=6! Элемент найден, его номер — 3.

**Примечание**. Вообще говоря, выбор значения m можно осуществлять случайным образом, корректность алгоритма от способа выбора m не зависит. Но на эффективность алгоритма выбор m влияет, так как мы хотим на каждом шаге исключить из дальнейшего рассмотрения как можно больше элементов. Оптимальным будет выбор среднего элемента, потому что при этом в любом случае будет исключаться половина массива.

В общем случае значение $m = \left[\frac{l+r}{2}\right]$, где l — индекс первого, а r — индекс последнего элемента рассматриваемой части массива.

Ниже приведен фрагмент программной реализации бинарного поиска.

```
Begin
  l:=1; r:=n;
  {на первом шаге  рассматривается весь массив}
  f:=False;
  {признак того, что x не найден}
  While (l<=r) And Not f Do
  Begin
    m:=(l+r) Div 2;
    If a[m]=x Then f:=True
    {элемент найден! Поиск надо прекратить.}
    Else If a[m]<x Then l:=m+1
    {отбрасывается левая часть}
    Else r:=m-1 {отбрасывается правая часть}
  End
End;
```

Можно также реализовать бинарный поиск с использованием фиктивного "барьерного" элемента. Ниже приведен фрагмент программы, в котором предлагаем разобраться самостоятельно:

```
Begin
  a[0]=x;
  l:=1; r:=n;
  Repeat
  m:=(l+r) Div 2;
  If l>r Then m:=0
         Else If a[m]<x Then l:=m+1
                        Else r:=m-1;
  Until a[m]=x;
  ans:=m;
End;
```

В отличие от последовательного поиска, который требует времени, пропорционального количеству элементов массива, бинарный поиск требует значительно меньших временных затрат. Например, при $n=2^{40}$ бинарный поиск выполняется достаточно быстро, в то время как последовательный поиск требует значительного времени. Кроме того, нет алгоритма, который требовал бы сравнений меньше, чем поиск методом половинного деления.

***Решение задач***

**1.** Подсчитать количество сравнений, производимых при бинарном поиске.

**2.** Ниже приведены три версии программ бинарного поиска:

```
Var i, j, k, x: Integer;
    a: Array [1..n] Of Integer;
a) i:=1; j:=n;
Repeat
  k:=(i+j) Div 2;
  If x<=a[k] Then j:=k-1;
  If a[k]<=x Then i:=k+1;
Until i>j;

b) i:=1; j:=n;
Repeat
  k:=(i+j)Div 2;
  If x<a[k] then j:=k Else i:=k+1;
Until i>=j;

c) i:=1; j:=n;
Repeat
  k:=(i+j) Div 2
  If a[k]<x Then i:=k Else j:=k;
Until (a[k]=x) Or (i>=j);
```

Какая из трех программ верная? Исправьте ошибки. Какая из них более эффективная?

**3.** Использование двоичного поиска позволяет значительно улучшить алгоритм сортировки массива методом простого включения. Учитывая, что готовая последовательность, в которую надо вставить элемент, является упорядоченной, можно методом деления пополам определять позицию включения нового элемента в нее. Такой модифицированный алгоритм сортировки называется методом двоичного включения. Написать программу, реализующую этот метод.

## 3.2.4. Поиск подстроки в строке. Прямой поиск

*Постановка задачи.* Заданы две строки — s и x. Длина первой строки — n, длина второй — m, причем $0 \le m \le n$. Требуется определить, является ли строка x подстрокой строки s, при этом требуется обнаружить первое вхождение x в s.

Самым простым методом поиска является метод прямого поиска. Рассмотрим его на примере. Пусть s='воротник', а x='рот'. Длина первой строки —

n=8, длина второй — m=3. В данном случае строка x короче s, следовательно, нужно найти такое значение индекса i, что для любого значения индекса k — 1≤k≤3 — будет выполняться равенство s[i+k]=x[k].

Начальное значение i равно 0.

*1-й шаг*

i=0, k=1 — сравниваем s[1] и x[1]: 'в'≠ 'р', значит, с первой позиции вхождения нет, нужно увеличить на 1 значение i.

*2-й шаг*

i=1, k=1 — сравниваем s[2] и x[1]: 'о'≠'р', снова надо перейти к следующему i.

*3-й шаг*

i=2, k=1 — сравниваем s[3] и x[1]: 'р'='р', следовательно, возможно совпадение, нужно увеличить k.

*4-й шаг*

i=2, k=2 — s[4]=x[2] ('о'='о') — снова надо увеличить k.

*5-й шаг*

i=2, k=3 — s[5]=x[3] ('т'='т') — полное совпадение! Далее поиск можно не продолжать, так как требовалось обнаружить лишь первое вхождение x в s.

Таким образом, прямой поиск подстроки в строке сводится к последовательным сравнениям отдельных символов. Поиск продолжается до тех пор, пока не обнаружится вхождение или пока не будет пройдена вся строка s. При этом можно закончить просмотр, когда i будет равно n-m, так как при следующих значениях i длина любого фрагмента строки s с позиции i меньше m.

Ниже приведена программа, реализующая метод прямого поиска подстроки.

```
Program Example_58;
Var s,x: String;
   i,j,n,m: Integer;
   f: Boolean;
Begin
  Writeln('введите s, x');
  Readln(s); Readln(x);
  n:=length(s); m:=length(x);
  {определение длин строк}
  i:=0;
  f:=False;
  {признак того, что подстрока найдена}
  Repeat
   j:=1;
   While (j<=m) And (s[i+j]=x[j]) Do Inc(j);
   If j=m+1 Then f:=True Else Inc(i);
  Until f Or (i>n-m);
  If f Then Writeln(x,' является подстрокой ',s,
                    ' с позиции - ',i)
 Else Writeln(x,' не является подстрокой ',s);
  Readln;
End.
```

Этот алгоритм требует достаточно больших временных затрат, поскольку, когда n значительно больше m, количество выполняемых сравнений — (n-m)*m~n*m.

***Решение задач***

Описать прямой поиск подстроки без использования логической переменной f.

## 3.2.5. Поиск подстроки в строке. Алгоритм Р.Бойера и Дж. Мура

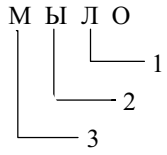
Алгоритм Р.Бойера и Дж. Мура по сравнению с линейным поиском требует значительно меньших временных затрат.

**Основная идея**

Поиск ведется от начала строки s, но с конца искомой подстроки x, для которой формируется таблица, размерность которой равна 256 — количеству всех символов в машинном алфавите. В таблице записываются расстояния от последнего символа искомой подстроки x до каждого ее символа. (Если в x встречаются одинаковые символы, то в таблицу заносится расстояние до ближайшего из них.) Если символ не входит в часть строки x без последнего символа, то в соответствующую ячейку таблицы заносится m — длина подстроки x. Когда очередной символ подстроки не совпадает с очередным символом строки s, для последнего из таблицы определяется расстояние, после чего x сдвигается вправо на соответствующее число позиций. Тем самым ряд позиций пропускается, время поиска сокращается.

**Пример**

Пусть s='МИЛА МАЛО МЫЛАСЬ МЫЛОМ', x='МЫЛО'. Длина x равна 4. Составим таблицу расстояний.




Расстояние до символа равно m-j, где j — индекс этого символа, а m — длина строки x.

Таблица расстояний (sd) выглядит так:

| Символ | Расстояние |
|---|---|
| ... | ... |
| А | 4 |
| ... | ... |
| К | 4 |
| Л | 1 |
| М | 3 |
| Н | 4 |

*Продолжение таблицы со стр. 29*

| Символ | Расстояние |
|---|---|
| О | 4 |
| П | 4 |
| ... | ... |
| Ъ | 4 |
| Ы | 2 |
| Ь | 4 |
| ... | ... |

Для всех символов, кроме выделенных, расстояние равно 4.

Будем выделять из строки s фрагменты длиной m и сравнивать их последовательно начиная с конца с соответствующими символами строки x. Пусть j — номер обрабатываемого символа строки x. Чтобы определить номер соответствующего символа s, нужно знать, с какой позиции начинается рассматриваемый фрагмент строки s. Обозначим через i номер символа, предшествующего первому символу обрабатываемого фрагмента. Тогда номер символа во фрагменте строки s, соответствующего j-му символу x, определяется как i+j.

*1-й шаг*

i=0, m=4. Рассмотрим первые 4 символа s и строку x. Начнем сравнивать их с конца. Для j=4 s[i+j] ≠ ≠ x[j] (s[4] ≠x[4]). Следовательно, далее можно не сравнивать, поскольку можно уже сказать, что этот фрагмент строки s не может являться искомой подстрокой (имеется по крайней мере одно несовпадение). Поэтому перейдем к следующему фрагменту.

*2-й шаг*

Теперь надо определить номер первого символа нового фрагмента. Для этого заметим, что так как последнего символа фрагмента s ('A') в x нет, то фрагменты со 2-го, 3-го, 4-го символов s можно не рассматривать, а нужно сразу перейти к фрагменту с позиции 5. В этом случае новое значение i получится прибавлением к старому значению sd['A'].

Таким образом, новый фрагмент строки s — ' МАЛ'. Начинаем новые сравнения с конца. Так как 'Л'≠'О', этот фрагмент снова можно далее не рассматривать.

*3-й шаг*

Поскольку символ 'Л' в искомой строке есть, то он может являться предпоследним. Следовательно, новое значение i=5 (определяется как 4+sd['Л']=4+1). Рассматриваем 'МАЛО' и 'МЫЛО': s[9]=x[4], s[8]=x[3], s[7]≠x[2] ('А'≠'Ы'), значит, и этот фрагмент строки s не является искомым.

*4-й шаг*

Четвертый фрагмент (i=9) ' МЫЛ' (см. 2-й шаг) снова не совпадает с x.

*5-й шаг*

Для следующего фрагмента значение i=9+sd['Л']=10. После первого сравнения можно сказать, что этот фрагмент тоже не подходит.

*6-й шаг*

i=14. Фрагмент 'СЬ М' пропускаем после первого сравнения.

*7-й шаг*

i=17 (14+sd['м']=14+3). Сравниваем 'МЫЛО' и 'МЫЛО':

Искомая подстрока найдена с позиции 18.

Задача решена.

Программа, реализующая этот алгоритм, выглядит так:

```
Program Example_59;
Var s,x: String;
    sd: Array[0..255] Of Integer;
Procedure search(s,x:String);
Var i,j,n,m: Integer;
    f: Boolean;
    h: Char;
Begin
  n:=Length(s); m:=Length(x);
  {определение длин строки и подстроки}
  {начальное заполнение массива расстояний}
   For i:=0 To 255 Do sd[i]:=m;
     For i:=1 To m-1 Do
    {заполнение массива расстояний}
     Begin
      h:=x[i];
      sd[Ord(h)]:=m-i;
    End;
   i:=0; f:=False;
   {признак того, что подстрока найдена}
   While (i<n-m+1) And (Not f) Do
   Begin
    j:=m;
    While (j>0) And (s[i+j]=x[j]) Do j:=j-1;
    If j=0 Then f:=True
    {полное совпадение!}
    Else i:=i+sd[Ord(s[i+j])];
  End;
  If f Then Writeln(x,'является подстрокой ',
                    s,' с позиции',i+1)
  Else Writeln('нет вхождения');
End;
  Begin   {Основная программа}
    Writeln('Введите строку s.'); Readln(s);
  Writeln('Введите подстроку x.'); Readln(x);
  search(s,x);
  Readln
End.
```

При анализе производительности этого алгоритма его авторы показали, что почти всегда, кроме специально построенных примеров, алгоритм требует значительно меньше n сравнений. В самых же благоприятных случаях, когда последний символ искомой строки x всегда попадает на несовпадающий символ строки s, число сравнений равно n/m.

ДОРОГИЕ ЧИТАТЕЛИ!

"Информатика" предлагает вам стать региональными представителями нашей газеты. Основная функция региональных представителей – распространение информационных материалов, содействие в обеспечении оперативной связи между учителями информатики и редакцией нашей газеты. (Распространение самих газет с помощью региональных представителей пока не предусматривается. Основной способ распространения "Информатики" остается прежним – почтовая подписка. Жители Москвы и Московской области также могут оформить редакционную подписку с получением газет в редакции.)

На первом этапе мы планируем иметь всего 15–20 представителей в регионах (кроме Москвы и Московской области). Региональные представители будут работать на договорной основе, они будут получать денежное вознаграждение.

Список региональных представителей будет периодически публиковаться в газете.

Замещение должностей региональных представителей конкурсное. Для участия в конкурсе необходимо выслать в наш адрес заполненную анкету. Победители конкурса будут извещены письменно.

Фамилия, имя, отчество (полностью) _____
_____.

Почтовый адрес (полностью) _____
_____
_____.

Электронный адрес (если есть) _____.

Место и стаж работы _____
_____.

С какого года вы выписываете "Информатику"? _____.

Участвуете ли вы в совещаниях учителей информатики (районных, городских)? Как часто они проводятся в вашем регионе? _____
_____.

Какие учебники и учебные пособия вы используете? Какие программы, учебники и учебные пособия рекомендованы к использованию в вашем регионе? _____
_____
_____
_____
_____.

Дополнительная информация (вы можете по желанию предоставить дополнительную информацию о себе).
_____
_____
_____
_____
_____.

Мы приглашаем всех подписчиков принять участие в конкурсе. Даже если на первом этапе вы не станете нашим региональным представителем, мы будем учитывать результаты первого конкурса при проведении последующих.

С уважением, **гл. ред. С.Л. Островский**

**Гл. редактор** С.Л.Островский
**Зам. гл. редактора** Е.Б.Докшицкая
**Редакция:** Н.Л.Беленькая, Н.П.Медведева
**Дизайн и компьютерная верстка:** Н.И.Пронская
**Корректоры:** Е.Л.Володина, С.М.Подберезина





121165, Киевская, 24
тел. 249 4896
Отдел рекламы
тел. 249 9870

**Учредитель: ООО "Чистые пруды"**
**Регистрационный номер 012868**
Отпечатано в типографии ОАО ПО "Пресса-1".
125865, ГСП, Москва, ул. Правды, 24.
Тираж 5000 экз.
Заказ №

**ИНДЕКС ПОДПИСКИ**
**для индивидуальных подписчиков 32291**
**комплекта приложений 32744**

**Тел. (095)249 3138, 249 3386. Факс (095)249 3184**

**Internet: inf@1september.ru**
**Fidonet: 2:5020/69.32**
**WWW: http://www.1september.ru**