Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ
И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

УТВЕРЖДАЮ
Заведующий кафедрой АОИ
д-р техн. наук, проф.
_____Ю.П. Ехлаков
«___»_____2016 г.

# Информатика и программирование

методические указания к лабораторным работам и практическим занятиям
для студентов направления
38.03.05 – «Бизнес-информатика»

Разработчик
ст. преподаватель каф. АОИ
_____Н.В. Пермякова
«___»_____2016 г.

## Оглавление

**ВВЕДЕНИЕ**

Основная цель курса – развитие теоретических представлений и практических навыков работы с информацией, хранящейся или обрабатываемой в вычислительных системах, способам представления данных и их обработки с помощью современных информационных технологий, формирование у студентов объектно-ориентированного мышления, обучение объектно-ориентированному (ОО) подходу к анализу предметной области и использованию объектно-ориентированной методологии программирования при разработке программных продуктов.

Для достижения указанной цели при выполнении лабораторных работ решаются следующие задачи:

формирование у студента знаний основных понятий, концепции, принципов и теорий, связанные с информатикой, понятия количества информации, типов систем счисления, структуры операционных систем, устройства файловых систем, основ архитектуры компьютера, способов представления алгоритмов, основных принципов структурного программирования;

получение студентами навыков осуществления операций преобразования и математических операций над данными, представленными в разных системах счисления, представления алгоритмов, программирования на языке высокого уровня;

обучение студентов владению языками структурного программирования, математическим аппаратом систем счисления, навыками использования прикладных программ, навыками разработки и отладки программ на алгоритмических языках программирования;

изучение техники объектно-ориентированного анализа;

изучение приемов объектно-ориентированного программирования.

**Лабораторная работа №1. *Системы счисления***

**Цель работы:** приобретение навыков перевода чисел в различные системы счисления, изучение правил выполнения основных арифметических операций над числами в разных системах счисления.

**Основные понятия систем счисления.**

Системой счисления называют систему приемов и правил, позволяющих устанавливать взаимно-однозначное соответствие между любым числом и его представлением в виде совокупности конечного числа символов. Множество символов, используемых для такого представления, называют цифрами.

Количество цифр, необходимых для записи числа в системе, называют основанием системы счисления. Основание системы записывается в справа числа в нижнем индексе: $5_{10}$; $111011_2$; $AF178_{16}$

Системы счисления разделяются на две группы: позиционные и непозиционные.

В *непозиционной* системе счисления смысл каждой цифры числа не зависит от занимаемой ею позиции. Примером такой системы счисления является римская система. В числе XXX, записанном в этой системе, цифра X в любой позиции означает 10 (десять).
В вычислительной технике непозиционные системы не применяются.
Систему счисления называют *позиционной*, если одна и та же цифра может принимать различные численные значения в зависимости от номера разряда этой цифры в совокупности цифр, представляющих заданное число. Пример такой системы – арабская десятичная система счисления.
В общем случае в позиционной системе счисления число N может быть представлено как:
$N_b = (p_n p_{n-1} \ldots p_1 p_0, p_{-1} p_{-2} \ldots)$, где:

$b$ – основание системы счисления (целое положительное число, равное числу цифр в данной системе);

$n$ – любые цифры из интервала от нуля до $(b-1)$.
Развернутой формулой записи числа называется запись в виде

$$A_q = \pm(a_{n-1} q^{n-1} + \ldots + a_0 q^0 + a_{-1} q^{-1} + \ldots + a_{-m} q^{-m}),$$

Здесь $A_q$ – само число, q – основание системы счисления, $a_i$ - цифры данной системы счисления, n – число разрядов целой части числа, m – число разрядов дробной части числа.
Для записи чисел в позиционной системе счисления с основанием n
снование позиционной системы счисления определяет ее название. В вычислительной технике применяются двоичная, восьмеричная, десятичная и шестнадцатеричная системы.

**Двоичная система счисления**

В компьютерах применяется, как правило, не десятичная, а позиционная двоичная нужно иметь алфавит из n цифр. Обычно для этого при n < 10 используют n первых арабских цифр, а при n > 10 к десяти арабским цифрам добавляют буквы. Вот примеры алфавитов нескольких систем:

| **Основание** | **Название** | **Алфавит** |
|---|---|---|
| **n=2** | **двоичная** | **0 1** |
| **n=3** | **троичная** | **0 1 2** |
| **n=8** | **восьмеричная** | **0 1 2 3 4 5 6 7** |
| **n=16** | **шестнадцатеричная** | **0 1 2 3 4 5 6 7 8 9 A B C D E F** |

Осистема счисления, т.е. система счисления с основанием 2.
В двоичной системе любое число записывается с помощью двух цифр 0 и 1 и называется двоичным числом.
Существуют специальные термины, широко используемые в вычислительной технике: бит, байт и слово.
Битом называют один двоичный разряд. Крайний слева бит числа называют старшим разрядом (он имеет наибольший вес), крайний справа – младшим разрядом (он имеет наименьший вес).
Восьмибитовая единица носит название байта.
Многие типы ЭВМ и дискретных систем управления перерабатывают информацию порциями (словами) по 8, 16 или 32 бита (1, 2 и 4 байта). Двоичное слово, состоящее из двух байт, показано на рисунке 1

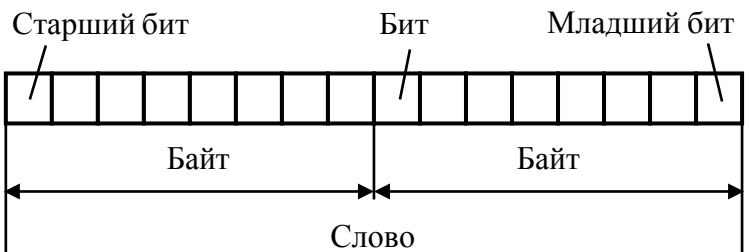


**Рис.** *Бит, байт и слово*

Как и десятичное число, любое двоичное число можно записать в виде суммы, явно отражающей различие весов цифр, входящих в двоичное число. В этой сумме в качестве основания используется число 2. Например, для двоичного числа 1010101,101 сумма примет вид.

$1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$.

В данном примере двоичное число имеет семизначную целую и трехзначную дробную части. Поэтому старшая цифра целой части, т.е. единица, умножается на $2^{7-1} = 2^6$, следующая цифра целой части, равная нулю, умножается на $2^5$ и т.д. по убывающим степеням двойки до младшей, третьей, цифры дробной части, которая будет умножена на $2^{-3}$. Выполняя в этой сумме арифметические операции по правилам десятичной системы, получим десятичное число 85,625. Таким образом, двоичное число 1010101,101 совпадает с десятичным числом 85,625, или

$1010101{,}101_2 = 85{,}625_{10}$.

**Правило перевода двоичного числа в десятичное.** Чтобы перевести число из двоичной системы в десятичную систему счисления, нужно двоичное число представить в виде суммы степеней двойки с коэффициентами-цифрами и найти эту сумму. При переводе удобно пользоваться таблицей степеней двойки:

Степени числа 2

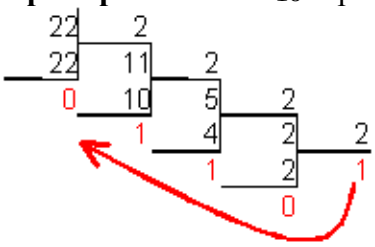
| n | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $2^n$ | 1 | 2 | 4 | 8 | 16 | 32 | 64 | 128 | 256 | 512 | 1024 |

**Пример.** Число $11101_2$ перевести в десятичную систему счисления.

**Перевод в двоичную систему счисления чисел из десятичной системы счисления**

При переводе десятичного числа в двоичное нужно это число делить на 2.

**Пример.** Число $22_{10}$ перевести в двоичную систему счисления.




$22_{10} = 10110_2$

**Правило перевода.** Чтобы перевести целое положительное десятичное число в двоичную систему счисления, нужно это число разделить на 2, полученное частное снова разделить на 2 и т.д. до тех пор, пока частное не окажется меньше двух. В результате записать в одну строку последнее частное и все остатки, начиная с последнего.

**Перевод десятичных дробей в двоичную систему счисления.** Перевод десятичных дробей в двоичную систему счисления заключается в поиске целых частей при умножении на 2.

**Пример**. Переведем десятичную дробь 0,625 в двоичную систему счисления.

Чтобы найти первую после запятой цифру двоичной дроби, нужно умножить заданное число на 2 и выделить целую часть произведения.

0,625 x 2 = 1,250, целая часть равна 1;
0,250 x 2 = 0,500, целая часть равна 0;
0,500 x 2 = 1,000, целая часть равна 1.
Дробная часть последнего произведения равна нулю. Перевод закончен. Записываем в одну строку полученное значение целой части, начиная с первой цифры. $0,625_{10} = 0,101_{2.}$
Каждый раз в умножении на 2 участвуют только дробная часть десятичного числа.
**Правило перевода**. Чтобы перевести положительную десятичную дробь в двоичную, нужно дробь умножить на 2. Целую часть произведения взять в качестве первой цифры после запятой в двоичной дроби, а дробную часть вновь умножить на 2. В качестве следующей цифры двоичной дроби взять целую часть этого произведения, а дробную часть произведения снова умножить на 2 т. д.
При переводе конечной десятичной дроби в двоичную может получиться бесконечная.
**Пример**. Переведем десятичную дробь 0,3 в двоичную систему счисления.

0.3 x 2 = 0,6, целая часть равна 0;
0,6 x 2 = 1,2, целая часть равна 1;
0,2 x 2 = 0,4, целая часть равна 0;
0,4 x 2 = 0,8, целая часть равна 0;
0,8 x 2 = 1,6, целая часть равна 1;
0,6 x 2 = 1,2, целая часть равна 1 и т.д.

Дробная часть 0,6 уже была на втором шаге вычислений. Поэтому вычисления начнут повторяться. Следовательно, в двоичной системе счисления число 0,3 представляется периодической дробью. $0,3_{10} = 0,0\ (1001)_2$. На практике эти операции продолжают до тех пор, пока после запятой не получится заданное количество цифр.
*Арифметические действия над двоичными числами.* ***Арифметика двоичной системы счисления основана на использовании таблиц сложения, вычитания и умножения. Эти таблицы чрезвычайно просты:***

| Таблица сложения | Таблица вычитания | Таблица умножения |
|---|---|---|
| 0 + 0 = 0 | 0 – 0 = 0 | 0 * 0 = 0 |
| 0 + 1 = 1 | 1 – 0 = 1 | 0 * 1 = 0 |
| 1 + 0 = 1 | 1 – 1 = 1 | 1 * 0 = 0 |
| 1 + 1 = 10 | 10 – 1 = 1 | 1 * 1 = 1 |

**Двоичное сложение. Двоичное сложение выполняется по тем же правилам, что и десятичное, с той лишь разницей, что перенос в следующий разряд производиться после того, как сумма достигнет не десяти, а двух.**
Пример. Сложение двоичных чисел $(101101)_2$ и $(111110)_2$

```
  101101
+ 111110
--------
  010011   – поразрядная сумма без учета переносов

  1011000  – переносы
+ 0010011
---------
  1001011  – поразрядная сумма без учета повторных переносов

  0100000 – повторные переносы
+ 1001011
---------
  1101011 – окончательный результат
```

Легко произвести проверку:

$(101101)_2 = 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 = 32 + 8 + 4 + 1 = (45)_{10}$,

$(111110)_2 = 1\cdot 2^5 + 1\cdot 2^4 + 1\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0 = 32 + 16 + 8 + 4 + 2 = (62)_{10}$,

$(1101011)_2 = 1\cdot 2^6 + 1\cdot 2^5 + 0\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 1\cdot 2^1 + 1\cdot 2^0 = 64 + 32 + 8 + 2 = (107)_{10}$,

$(45)_{10} + (62)_{10} = (107)_{10}$.

Пример. Сложение двоичных чисел $(110{,}1011)_2$ и $(10111{,}10101)_2$

```
    110,  1011
+ 10111,  10101
-----------------
  10001,  00011   – поразрядная сумма без учета переносов
   11 1    1      – переносы
+ 10001,  00011
-----------------
  11100,  01011   – поразрядная сумма без учета повторных переносов
      1           – повторные переносы
+ 11100,  01011
-----------------
  11110,  01011   – окончательный результат
```

Сложение нескольких чисел вызывает некоторые трудности, так как в результате поразрядного сложения могут получаться переносы, превышающие единицу.

Двоичное вычитание. **Вычитание в двоичной системе выполняется аналогично вычитанию в десятичной системе счисления. При необходимости, когда в некотором разряде приходится вычитать единицу из нуля, занимается единица из следующего старшего разряда. Если в следующем разряде нуль, то заем делается в ближайшем старшем разряде, в котором стоит единица. При этом следует понимать, что занимаемая единица равна двум единицам данного разряда, т. е. вычитание выполняется по следующему правилу:**

**Пример.** Вычитание двоичных чисел $(11010{,}1011)_2$ и $(1101{,}01111)_2$

```
  11010,  1011
- 1101,  01111
---------------
  1101,  00111
```

Двоичное умножение. **Умножение двух двоичных чисел выполняется так же, как и умножение десятичных. Сначала получаются частичные произведения и затем их суммируют с учетом веса соответствующего разряда множителя.**

Отличительной особенностью умножения в двоичной системе счисления является его простота, обусловленная простотой таблицы умножения. В соответствии с ней, каждое частичное произведение или равно нулю, если в соответствующем разряде множителя стоит нуль, или равно множимому, сдвинутому на соответствующее число разрядов, если в соответствующем разряде множителя стоит единица. Таким образом, операция умножения в двоичной системе сводится к операциям сдвига и сложения.

Умножение производится, начиная с младшего или старшего разряда множителя, что и определяет направление сдвига. Если сомножители имеют дробные части, то положение запятой в произведении определяется по тем же правилам, что и для десятичных чисел.

**Пример.** Умножение двоичных чисел $(101)_2$ и $(011)_2$

```
      1  0  1
  x      1  1
-------------
      1  0  1
+  1  0  1
-------------
   1  1  1  1(2) = 15
```

Двоичное деление. **Деление чисел в двоичной системе производится аналогично делению десятичных чисел. Рассмотрим деление двух целых чисел, так как делимое и делитель всегда могут быть приведены к такому виду путем перениесения запятой в делимом и делителе на одиноаковое число разрядов и дописывания необходимых нулей. Деление начинается с того, что от делимого слева отделяется минимальная группа разрядов, которая, рассматриваемая как число, превышает или равна делителю. Дальнейшие действия выполняются по обычным правилам, причем последняя целая цифра частного получается тогда, когда все цифры делимого исчерпаны.**

**Пример**. Деление двоичных чисел

**1) 18:2**

```
 1 0 0 1 0 | 1 0
 1 0       | 1 0 0 1
 ---
   0 0 1 0
     1 0
   -----
       0
```

**$1001_{(2)}$ = 9**

**2) 15,5:2**

```
1 1 1 1, 1 | 1 0
1 0        | 1 1 1, 1 1(10)  = 7,75(10)
-----
  1 1
  1 0
  ---
    1 1
    1 0
    ---
      1 1
      1 0
      ---
        1 0
        1 0
        ---
          0
```

**Восьмеричная система счисления.**

В восьмеричной системе, т.е. системе счисления с основанием 8, числа выражаются с помощью восьми цифр: 0, 1, 2, 3, 4, 5, 6, 7. Например, в восьмеричном числе 357 есть семь единиц, пять восьмерок и три восьмерки в квадрате, т.е. $357_8 = 3 \times 8^2 + 5 \times 8^1 + 7 \times 8^0$, где индекс «8» у числа 357 означает систему счисления. Выполняя в записанной сумме арифметические действия по правилам десятичной системы, получим, что $357_8 = 239_{10}$, т.е. восьмеричное число 357 совпадает с десятичным числом 239.

**Перевод десятичных чисел в восьмеричную систему счисления.** Для перевода числа из десятичной системы в восьмеричную применяется тот же прием, что и при переводе в двоичную систему.

Преобразуемое число делят на 8 по правилам десятичной системы с запоминанием остатка, который, конечно, не превышает 7. Если полученное частное больше 7, его тоже делят на 8,

сохраняя остаток. Новое частное, если оно больше 7, в свою очередь делят на 8. этот процесс деления на 8 продолжается до тех пор, пока полученное частное не станет меньше основания. Затем выписывают подряд все остатки, начиная с последнего полученного частного. Это и будет результирующее восьмеричное число.

**Например**. Переведем число 891 из десятичной системы счисления в восьмеричную систему.

891 делим на 8, получим 111 в остатке 3,
111 делим на 8, получим 13 в остатке 7,
13 делим на 8, получим 1 в остатке 5.
В итоге **1** – есть старшая цифра двоичного числа, далее 5,7,3.
$891_{10} = 1573_8$.

**Перевод двоичных чисел в восьмеричную систему счисления.** Чтобы перевести число из двоичной системы в восьмеричную, его нужно разбить на триады (тройки цифр), начиная с младшего разряда, в случае необходимости дополнив старшую триаду нулями, и каждую триаду заменить соответствующей восьмеричной цифрой.

**Пример:** $11101101_{(2)} \rightarrow 011101101_{(2)} \rightarrow 355_{(8)}$

**Сложение и вычитание в восьмеричной системе счисления.** При выполнении сложения и вычитания в восьмеричной системе счисления необходимо соблюдать следующие правила:
в записи результатов сложения и вычитания могут быть использованы только цифры восьмеричного алфавита;
десяток восьмеричной системы счисления равен 8, т.е. переполнение разряда наступает, когда результат сложения больше или равен 8.
В этом случае для записи результата надо вычесть 8, записать остаток, а к старшему разряду прибавить единицу переполнения;
3)если при вычитании приходится занимать единицу в старшем разряде, эта единица переносится в младший разряд в виде восьми единиц.

**$145_{(8)} + 367_{(8)}$**

| | | | | |
|---|---|---|---|---|
| **1** | **4** | **5** | | |
| **3** | **6** | **7** | | |
| **4** | **2** | **4** | - | **Результат без переносов** |
| **1** | **1** | | - | **Переносы** |
| **5** | **3** | **4** | - | **Общий результат** |

**$352_{(8)} - 165_{(8)}$**

| | | |
|---|---|---|
| **3** | **5** | **2** |
| **1** | **6** | **5** |
| **1** | **6** | **5** |

**Шестнадцатеричная система счисления**

Для сокращения записи двоичных чисел используют систему счисления с основанием 16. Эту систему называют шестнадцатеричной. В этой системе счисления для записи чисел используются цифры десятичной системы счисления 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 и для обозначения шести недостающих цифр используются первые прописные буквы латинского алфавита: A, B, C, D, E, F, имеющие значения десятичных чисел 10, 11, 12, 13, 14 и 15 соответственно. Таким образом, «цифрами» шестнадцатеричной системы являются все цифры десятичной сис-

темы и, кроме того, шесть латинских букв. За числом F следует число F + 1, что в десятичной системе соответствует 15 + 1 = 16.
Поэтому шестнадцатеричное число может иметь, например, вид 3E5A1. расписывая это число суммой с учетом основания 16, получим
$3E5A1_{16} = 3 \times 16^4 + E \times 16^3 + 5 \times 16^2 + A \times 16^1 + 1 \times 16^0$.
Выполняя арифметические операции по правилам десятичной системы и учитывая, что A = 10, E = 14, получим $3E5A1_{16} = 255393_{10}$. Заметьте, что число в шестнадцатеричной системе более компактно, чем в десятичной системе.

**Перевод десятичных чисел в шестнадцатеричную систему счисления**

Преобразуемое число делят на 16 по правилам десятичной системы с запоминанием остатка, который, конечно, не превышает 15. Если полученное частное больше 15, его тоже делят на 16, сохраняя остаток. Новое частное, если оно больше 15, в свою очередь делят на 16. этот процесс деления на 16 продолжается до тех пор, пока полученное частное не станет меньше 16. Затем выписывают подряд все остатки, начиная с последнего частного. Это и будет результирующее шестнадцатеричное число.

**Пример**. Число 891 перевести из десятичной системы в шестнадцатеричную систему счисления.
891 делим на 16, получим 55 в остатке 11,
55 делим на 16, получим 3 в остатке 7.
В итоге 3 – есть старшая цифра десятичного числа,
$891_{10}=37B_{16}$, т. к. B=11.

**Сложение и вычитание в шестнадцатеричной системе счисления.**

При выполнении этих действий в 16-ой с/с необходимо соблюдать следующие правила:
1)при записи результатов сложения и вычитания надо использовать цифры шестнадцатеричного алфавита: цифры, обозначающие числа от 10 до 15 записываются латинскими буквами, поэтому , если результат является числом из этого промежутка, его надо записывать соответствующей латинской буквой;
2)десяток шестнадцатеричной системы счисления равен 16, т.е. переполнение разряда поступает, если результат сложения больше или равен 16, и в этом случае для записи результата надо вычесть 16, записать остаток, а к старшему разряду прибавить единицу переполнения;
3)если приходится занимать единицу в старшем разряде, эта единица переносится в младший разряд в виде шестнадцати единиц.

**Задания к работе**

1. Перевести данное число из десятичной системы счисления в двоичную, восьмеричную и шестнадцатеричную системы счисления.
2. Перевести данное число в десятичную систему счисления.
3. Сложить числа.
4. Выполнить вычитание.
5. Выполнить умножение.
6. Выполнить деление.

Примечание. В заданиях 3–6 проверять правильность вычислений переводом исходных данных и результатов в десятичную систему счисления. В задании 1д получить пять знаков после запятой в двоичном представлении.

## Лабораторная работа №2. *Системы кодирования алгоритмов*

**Цель работы:** обучение навыкам представления алгоритмов с помощью блок-диаграмм, псевдокода, диаграмм Насси-Шнайдермана.

### Системы кодирования алгоритмов

*Система псевдокод*

Для описания алгоритмов существует множество способов, алгоритмы могут быть описаны естественным языком, формальным языком (языком программирования) или схемой. Псевдокод представляет собой систему обозначений и правил, предназначенную для единообразной записи алгоритмов. Псевдокод занимает промежуточное место между естественным и формальным языками. Основные служебные слова этого языка приведены в табл. 1.

Таблица 1. Служебные слова алгоритмического языка (псевдокода)

| | | | | |
|---|---|---|---|---|
| алг (алгоритм) | сим (символьный) | дано | для | да |
| арг (аргумент) | лит (литерный) | надо | от | нет |
| рез (результат) | лог (логический) | если | до | при |
| нач (начало) | таб(таблица) | то | знач | выбор |
| кон (конец) | нц (начало цикла) | иначе | и | ввод |
| цел (целый) | кц (конец цикла) | все | или | вывод |
| вещ (вещественный) | длин (длина) | пока | не | утв |

Запишем несколько алгоритмов на псевдокоде.

*Пример 1.* Даны два катета прямоугольного треугольника *k1 = 3, k2 = 4*. Найти гипотенузу. Записать решение задачи на псевдокоде.

алг Вычисление гипотенузы нач

дано $k1 = 3,\ k2 = 4$

$z := k1^2$ *// Вычислить квадрат* $k1$ *и запомнить в переменной* $z$

$z2 := k2^2$ *// Вычислить квадрат* $k2$ *и запомнить в переменной* $z2$

$g = \sqrt{z + z2}$ *// Вычислить гипотенузу и запомнить в переменной* $g$

рез $g$.

кон

Текст, выделенный курсивом, называется комментарием, пояснением к выполняемым действиям. Если в решаемой задаче требуется вычислить какое-либо значение (алгоритмы подобного рода называются вычислительными алгоритмами), то последним, завершающим действием алгоритма будет возвращение полученного результата.

*Пример 2.* Даны два произвольных числа. Найти максимальное число.

алг Поиск максимального нач

вещ $a, b$

ввод $a, b$

если $a \neq b$ то

если $a < b$ то рез «$b$ – максимальное число»

иначе рез «$a$ – максимальное число»

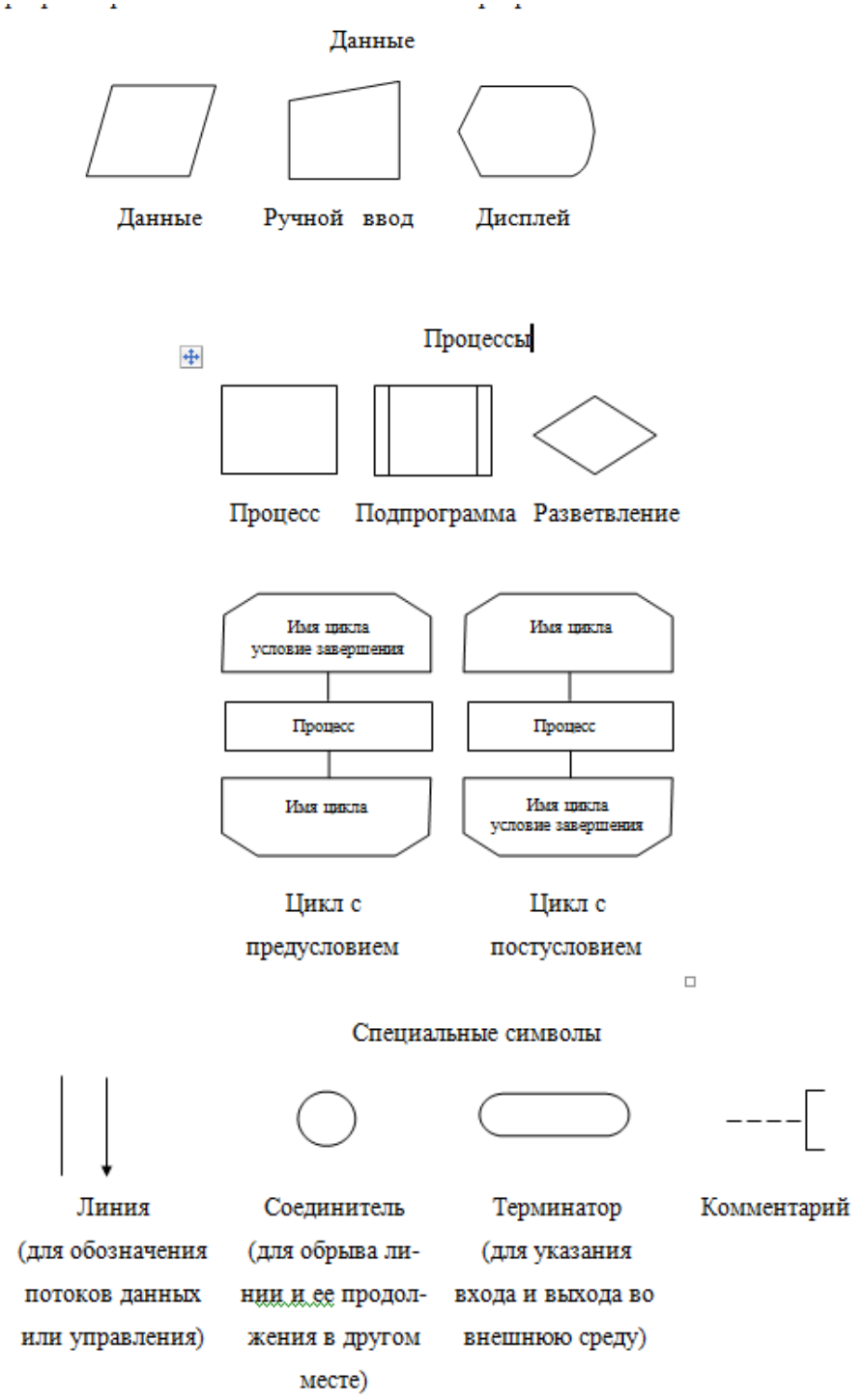
иначе рез «числа равны»

кон

*Блок-диаграммы*

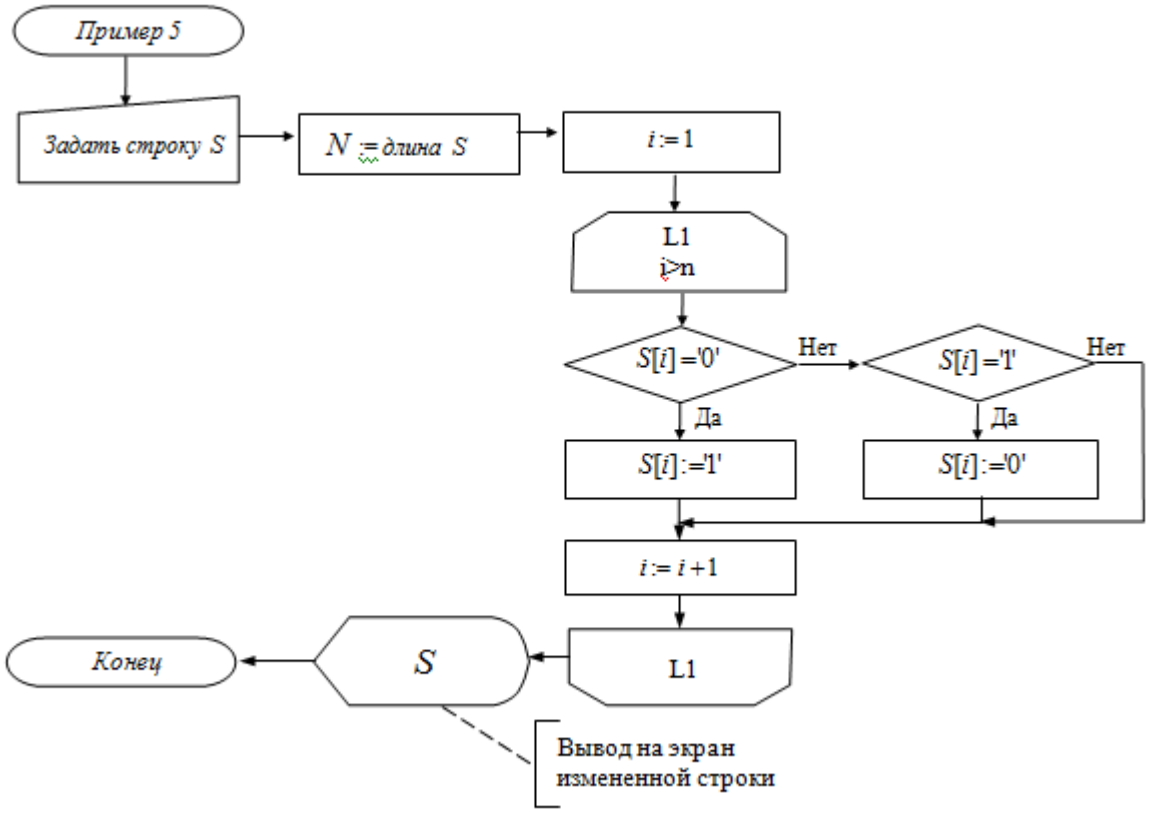
Алгоритм можно задать, используя схему, которая носит название блок-диаграммы. Правила начертания элементов, соответствующих основным конструкциям структурного программирования определены ГОСТом 19.701-90.

Приведем основные обозначения, соответствующие конструкциям структурного программирования и основным элементам программ:




Построим блок-диаграмму для примера 5.

*Пример 3.* Дана строка символов *S*. Заменить все вхождения символа “1” на “0”, а “0” на “1”.




*Диаграммы Насси-Шнайдермана*

Основные конструкции структурного программирования в диаграмме Насси-Шнайдермана обозначаются следующим образом:

Следование –

| *Действие 1* |
|---|
| *Действие 2* |

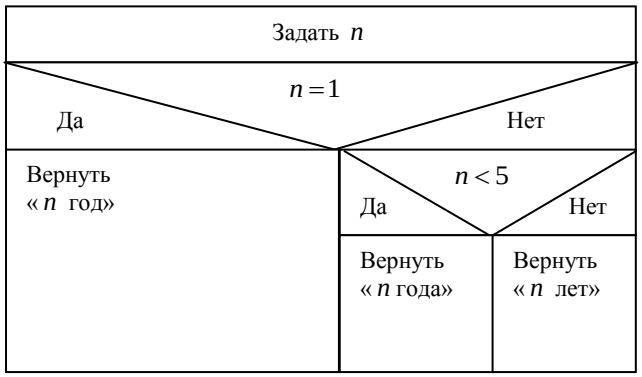
*Развилка* –

| Условие | |
|---|---|
| Да | Нет |
| Действия при истинном условии | Действия при ложном условии |

Цикл –

| Условие выполнения цикла | |
|---|---|
| | *Действия в цикле* |
| | *Действия в цикле* |
| | *Действия в цикле* |

*Пример 4*. Дано число натуральное число *n<10*. Вывести на экран грамматически верную фразу (*n* лет,  - 1 год, 2 года, 10 лет и т.д..)




**Задания к работе**

Представить алгоритм с помощью:

- псевдокода
- блок-диаграммы
- диаграммы Насси-Шнайдермана

**Лабораторная работа №3.** ***Основные конструкции структурного программирования. Проверка условий***

**Цель работы:** освоение условной конструкции языка Си

**Ветвление**

*Оператор проверки условия if <else>*

Для организации ветвления алгоритма в Си используется оператор проверки условия *if (*логическое выражение*) {*действия при истинном значении выражения*} else {*действия при ложном значении выражения*}.* Оператор *else* может отсутствовать, если это обусловлено алгоритмом.

Напомним, что ложью в Си считается нулевое значение, соответственно истиной – любое ненулевое значение. Поэтому, выражение *5>3* возвращает ненулевое значение, а выражение *3==0* – нулевое.

Логическое условие может быть сложным, т.е. может состоять из нескольких условий, связанных между собой логическими операциями:

– «И» (конъюнкция), в Си оператор &&, все логическое выражение считается истинным только в том случае, если истинны все простые выражения.

– «ИЛИ» (дизъюнкция), в Си оператор ||, все логическое выражение считается ложным только в том случае, если ложны все простые выражения.

Например, запишем следующее условие «Если переменная *x* меньше переменной *y* и переменная *x* меньше переменной z»: *x<y && x<z.* Следующее условие демонстрирует операцию дизъюнкции «Если переменная *m < 10* или переменная *m* равна переменной *x*» : *m<10 || m == x.*

Если в блоках программы, выполняющихся при истинности или ложности условия необходимо выполнить два или более действий, эти блоки определяются фигурными скобками. Приведем примеры.

– Если *x<y*, то вывести на экран значение суммы *x* и *y*, значение переменной *x* заменить на *10.*

```
if (x<y)
  { printf("Сумма ==> %d",x+y);
    x = 10;
  }
```

– Если *x* не равно *y* вывести на экран соответствующее сообщение, в противном случае вывести на экран значения переменных, переменную *x* увеличить в два раза.

```
if (x!=y)
  printf(“Переменные имеют неравные значения”);
else  {
        printf(“x = %d, y = %d\n”,x,y);
       x*=2;
      }
```

*Обратите внимание* на основные ошибки при работе с оператором *if*:

–вместо операции *«==»* (равно) используют операцию *«=»* (присвоить);

–не используются фигурные скобки:

```
if (x<y)
    printf(“Сумма ==> %d”,x+y);
   x = 10;
else ...
```

Дополним пример из предыдущего пункта проверкой корректности ввода данных. При таком условии ошибки выполнения могут быть следующие – вместо числовых данных введены символьные данные, или введены такие числовые данные, которые превращают выражение *x*x+z-y* в ноль. В случае таких исходных данных программа должна выдать на экран соответствующее сообщение и закончить свою работу.

*Пример 1.*

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
int main(int argc, char *argv[])
{
  system("chcp 1251");
  printf("      Программа вычисления по заданной формуле     \n");
 float x,y,z;
  printf("Введите значение x -->");
int m = scanf("%f",&x);
if (m !=1) {
                  printf("Введены некоректные данные. \n  ");
                        system("pause");
                        return 0; }
  printf("Введите значение y -->");
  m = scanf("%f",&y);
if (m !=1) {
                  printf("Введены некоректные данные. \n  ");
                        system("pause");
                        return 0; }
  printf("Введите значение z -->");
  m = scanf("%f",&z);
        if (m !=1) {
                  printf("Введены некоректные данные. \n  ");
                        system("pause");
                        return 0; }
  float f = x*x*x + y;
  float f1 = x*x+z-y;
  if (f1 == 0) { printf("Ошибка -  деление на 0!!!\n");
                        system("pause");
                        return 0; }
```

```
 f = f/f1;
 f +=x*z;
 f -=2*sin(y);
 printf("\n Введены значения  %5.2f  %5.2f   %5.2f \n "
"Значение f =  %8.3f \n", x,y,z,f);
system("PAUSE");      return 0; }
```

*Множественный выбор*

Предположим, необходимо решить следующую задачу: по заданному значению целочисленной переменной *k* выполнить одно из действий:

- если *k* равно 1 – найти сумму переменных *x* и *y*,
- если *k = 2* – найти произведение *x* и *y*,
- если *k = 3* – найти разность *x* и *y*,
- если *k = 4* – найти разность *y* и *x*.

Если для решения этой задачи использовать оператор *if* получается следующая структура с несколькими уровнями вложенности:

```
if (k==1){...}
else if (k==2){...}
      else if (k == 3) {...}
           else {...}
```

Си предлагает для решения таких задач оператор *switch.*

Синтаксис: *switch (выражение)*

```
{
case значение выражения1: операторы;
case значение выражения2: операторы;
…
default: операторы;
}
```

Выражение в *switch* может принимать целочисленные значения (типы *int, long int, char* и т.п.). Работа оператора множественного выбора происходит следующим образом - значение выражения сравнивается со значением, указанным в первом блоке *case.* Если значения совпали, выполняются операторы из первого блока *case.* Далее управление без проверки условия передается в последующие блоки *case.* Для того, чтобы пропустить их выполнение, в конце каждого блока должен быть выполнен оператор *break,* передающий управление на оператор, следующий за блоком *switch*. В блок *default* управление передается в случае, если не произошло ни одного совпадение выражения, указанного в *switch,* с выражениями, указанными в блоках *case.* Далее приведен текст программы, решающий поставленную задачу.

*Пример 2.*

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  system("chcp 1251");
  int x = 2;
  int y = 3;
  int k;
  printf("\n Введите 1, для вычисления суммы x + y \n");
  printf(" Введите 2, для вычисления произведения x*y \n");
  printf(" Введите 3, для вычисления  разности x - y \n");
  printf(" Введите 4, для вычисления разности y - x \n");
  scanf("%d",&k);
  switch (k)
```

```
{   case 1:
            printf("Сумма равна %d \n", x + y);break;
        case 2:
            printf("Произведение равно %d \n", x*y); break;
        case 3:
            printf("Разность равна %d \n",x - y); break;
        case 4:
            printf("Разность равна %d \n",y - x); break;
        default:
        printf("Введено число, не принадлежащее интервалу [1;4]! \n");
 }
system("PAUSE");
return 0;    }
```

**Задание к работе**

Напишите алгоритм предложенной задачи

Напишите программу, реализующую разработанный алгоритм

**Лабораторная работа №4.** ***Основные конструкции структурного программирования. Циклы с заданным числом итераций***

**Цель работы.**

Программирование циклических процессов. Использование конструкции цикла for языка Си.

**Конструкции циклов в языке Си**

*Цикл с фиксированным числом операций for*

Цикл, это конструкция структурного программирования, повторяющая определенные действия (итерации) несколько раз. При заданном количестве итераций в Си используется конструкция *for*. Синтаксис:

*for(секция инициализации значения; секция проверки условия; секция коррекции)*

Значение, инициализируемое в первой секции, называется счетчиком цикла. Повторяемые действия называются телом цикла. Если тело цикла состоит из двух и действий, тело цикла заключается в фигурные скобки.

Секция инициализации выполняется один раз, поэтому может содержать описание переменной. На каждом шаге значение счетчика подставляется в условное выражение второй секции, если выражение истинно, то управление передается в тело цикла, в противном случае, управление передается за цикл. После каждого выполнения тела цикла выполняется операция из секции коррекции.

Например, цикл

```
for(int i=0;i<3;i++)
 printf("%d \n",i);
```

будет работать следующим образом – счетчик *i* примет значение *0.* Условное выражение второй секции при таком значении счетчика истинно, на экран выведется значение переменной *i,* равное *0*. Управление передается в секцию коррекции и переменная *i* увеличивается на единицу. При этом условие все еще истинно, на экран выводится значение *1*. В ходе следующей итерации переменная-счетчик принимает значение *2*. Условное выражение остается истинным. На экран выводится значение счетчика. После этой итерации переменная *i* становится равной *3*. Условие при таком значении становится ложным, цикл заканчивает свою работу, управление передается следующей части программы.

*Циклы while и do while*

Существует множество задач, при выполнении которых циклические действия необходимо проводить до тех пор, пока истинно какое-либо условие. Для реализации таких алгоритмов возможно использовать циклы по условию *while* и *do while.*

Синтаксис: *while (условное выражение)*

*{ тело цикла*

*}*

Тело цикла *while* выполняется до тех пор, пока истинно условное выражение. Этот цикл называется циклом с предусловием. Цикл *while* может ни разу не выполниться (то есть, на входе в цикл условие ложно).

Синтаксис *do {*

*тело цикла*

*} while (условное выражение)*

Тело цикла *do while* выполняется пока условие истинно. Этот цикл называют циклом с постусловием. Такой цикл выполниться хотя бы один раз.

*Операторы безусловной передачи управления continue и break*

Оператор *break* досрочно завершает выполнение цикла. Управление передается оператору, следующему за циклом.

```
int n =15;
for(int i=0;i<n;i++)
{ int z = rand()%200;
if (z>100) break;}
```

Цикл должен выполняться *15* раз. В переменную *z* записывается случайное значение в интервале от *0* до *199* (функция *rand(), случайные числа от 0 до RAND_MAX (32767)*). Если получено случайное значение, большее *100*, цикл заканчивает свою работу.

Оператор *continue* пропускает все последующие операторы тела цикла и передает управление на начало цикла.

```
int f = 1;
do
{
int z = rand()%100;
if (z>30) continue;
if (z<10) f = 0;
printf("%d",z);
} while(f);
```

Описанный выше цикл работает следующим образом – цикл будет работать, пока переменная $f$ равна единице. Если полученное случайное значение будет больше *30*, то вторая проверка условия и функция печати полученного значения будут пропущены. Если полученное значение будет меньше *10*, то переменной $f$ будет присвоено значение *0*. Значение выведется на экран и цикл закончит свою работу.

В итоге, на экран будут выводиться числа, не больше тридцати и как только будет получено число, меньшее десяти, цикл закончит свою работу.

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Напишите и протестируйте программу.
3. Подготовьте отчет по лабораторной работе.

**Лабораторная работа №5.** ***Основные конструкции структурного программирования. Циклы по условию***

**Цель работы.** Программирование циклических процессов. Использование циклов по условию.

Циклы *while* и *do while*

Существует множество задач, при выполнении которых циклические действия необходимо проводить до тех пор, пока истинно какое-либо условие. Для реализации таких алгоритмов возможно использовать циклы по условию *while* и *do while.*

Синтаксис: *while (условное выражение)*

```
{ тело цикла
}
```

Тело цикла *while* выполняется до тех пор, пока истинно условное выражение. Этот цикл называется циклом с предусловием. Цикл *while* может ни разу не выполниться (то есть, на входе в цикл условие ложно).

Например:

```
int d=134;
int i=1;
while(d>1)
 { d=d/10;
  i++;
}
printf("Количество цифр числа =%d ", i);
```

Переменная-счетчик *i* изначально инициализируется значением 1 (число состоит хотя бы из одной цифры).

Пока значение исследуемого числа *d* больше единицы число *d* делится на 10 (по правилам преобразования типов выполняется целочисленное деление) и значение переменной-счетчика увеличивается на единицу.

Синтаксис *do {*

*тело цикла*

*} while (условное выражение)*

Тело цикла *do while* выполняется пока условие истинно. Этот цикл называют циклом с постусловием. Такой цикл выполнится хотя бы один раз.

Например:

```
int i=0;
do{
i++;
while(i<5);
```

Пока значение переменной *i* меньше пяти значение *i* увеличивается на единицу. Описанный цикл выполнится 5 раз и переменная *i* будет равна 5.

*Основные логические ошибки при использовании циклов.*

– После заголовка цикла ставиться точка с запятой. Такой цикл считается компилятором пустым, то есть, у него нет тела. Например:

```
for(int i=0;i<10;i++);
{ n+=10;
 y-=15;
}
```

При такой записи увеличение переменной *n* и уменьшение переменной *y* происходит за циклом, ровно один раз.

– Условие цикла заведомо ложно. Такой цикл никогда не выполниться.

– Условие цикла никогда не станет ложным. Такой цикл будет бесконечным. Как правило, эта ошибка возникает, если Вы не предусмотрели в теле цикла изменение переменной, от которой зависит условие цикла.

Операторы безусловной передачи управления *continue* и *break*

Оператор *break* досрочно завершает выполнение цикла. Управление передается оператору, следующему за циклом.

```
int n =15;
for(int i=0;i<n;i++)
{ int z = rand()%200;
if (z>100) break;
}
```

Цикл должен выполняться 15 раз. В переменную z записывается случайное значение в интервале от 0 до 199 (функция *rand(), stdlib.h*). Если получено случайное значение, большее 100, цикл заканчивает свою работу.

Оператор *continue* пропускает все последующие операторы тела цикла и передает управление в начало цикла.

```
int f = 1;
do
{
int z = rand()%100;
if (z>30) continue;
if (z<10) f = 0;
printf("%d",z);
} while(f);
```

Описанный выше цикл работает следующим образом – цикл будет работать, пока переменная *f* равна единице. Если полученное случайное значение будет больше *30*, то вторая проверка условия и функция печати полученного значения будут пропущены. Если полученное значение будет меньше *10*, то переменной *f* будет присвоено значение *0*. Значение выведется на экран и цикл закончит свою работу.

В итоге, на экран будут выводиться числа, не больше тридцати и как только будет получено число, меньшее десяти, цикл закончит свою работу.

*Пример 1.* Ввести с клавиатуры произвольное количество чисел и найти сумму введенных чисел. Ввод продолжать до первого отрицательного числа.

```
…
printf ("Вводите числа: \n");
   float S = 0,c=0;
   int n;
   int i = 1;
   do {
      printf("%d==> ",i);
      i++;
      n = scanf("%f",&c);
      if(n!=1) {fflush(stdin);
               i--;
               continue;}
      S+=c;
   }while (c>=0);
   printf(" Сумма - %f\n", S);
…
```

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Напишите и протестируйте программу.
3. Подготовьте отчет по лабораторной работе.

**Лабораторная работа №6.** ***Массивы. Поиск заданного значения в массиве***
**Цель работы:** формирование навыков работы с массивами. Разработка алгоритма поиска элемента с заданным значением.

Си поддерживает как одномерные, так и многомерные массивы. Под массивом будем понимать переменную, которая имея одно имя может сохранять множество значений. Поскольку это переменная, то перед использованием ее в программе ее необходимо объявить.
Общий синтаксис:

*Тип_данных Имя_массива*[<количество элементов в массиве>];

При объявлении массива может происходить и инициализация

*Тип_данных Имя_массива*[<количество элементов в массиве>]={<набор значений через запятую>};

Если количество значений элементов в наборе меньше, чем заявлено количество элементов, то в этом случае все элементы, не имеющие значений в указанном наборе будут иметь значение равное нулю.
Либо можно не указывать количество элементов массива (их количество будет определено из списка инициализации)

*Тип_данных Имя_массива*[]={<набор значений через запятую>};

Доступ к тому или иному значению элемента массива определяется следующим образом:

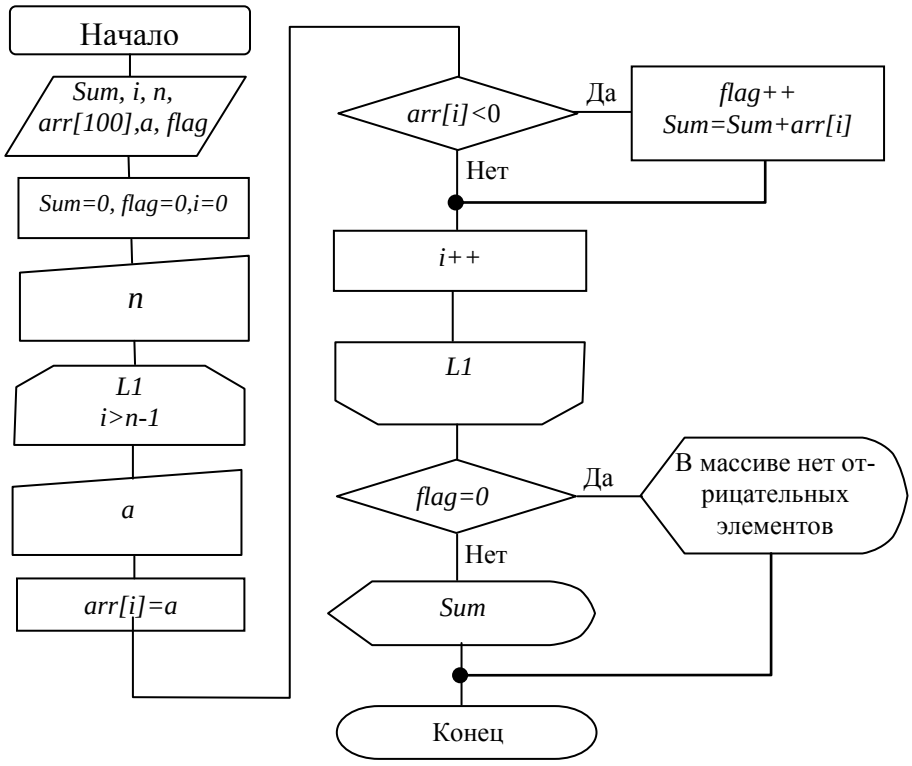
- сохранение значения в элементе массива

*Имя_массива*[*номер элемента*]=*значение*;

- присвоение значения элемента массива другой переменной

*Имя_переменной=Имя_массива*[*номер элемента*];

**Пример**. Вводится последовательность из *n* целых чисел. Сохранить все введенные значения в массиве и найти сумму всех отрицательных чисел.
**Решение**.



```
int main(int argc, char *argv[])
{
  system("chcp 1251");
  int Sum=0, a, n, arr[100];
  int flag = 0;
  printf("Введите количество членов последовательности не более 100 ");
  scanf("%d",&n);
 for(int i=0; i<n; i++)
    {
      printf("Введите значение члена последовательности");
      scanf("%d", &a);
      arr[i]=a;
      if (arr[i]<0)  {Sum+=arr[i]; flag++;}
    }
if (!flag) printf("В массиве нет отрицательных элементов\n"); else
printf("Значение суммы отрицательный членов последовательности равна %6d", Sum);
}
```

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

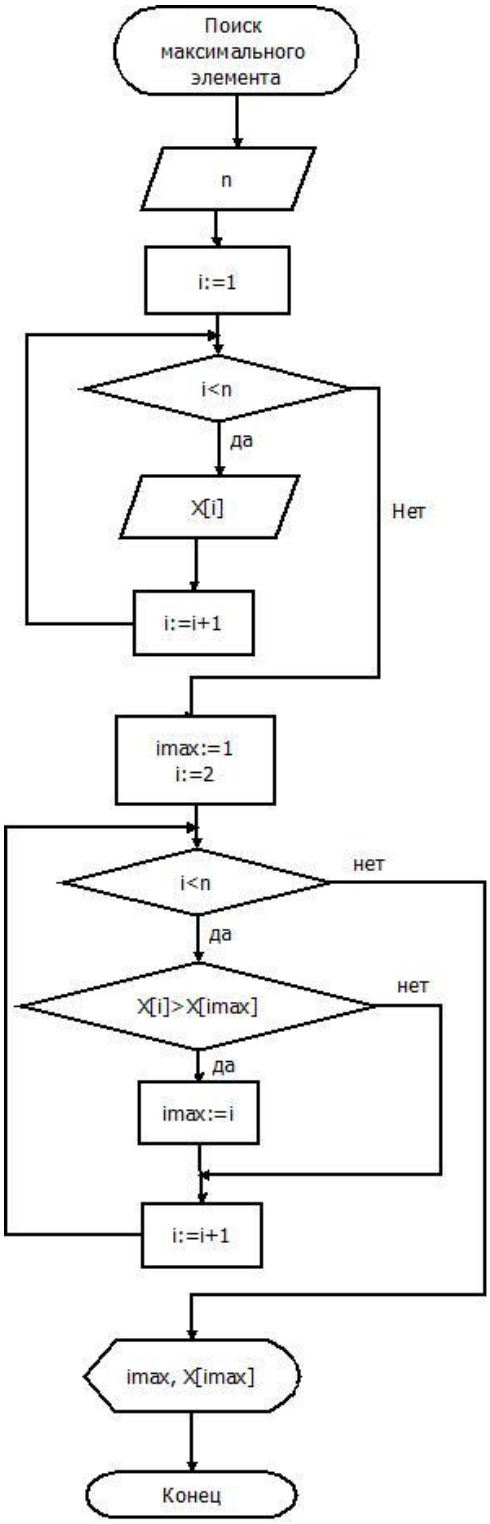
**Лабораторная работа №7.** ***Массивы. Поиск минимального и максимального значений***
**Цель работы:** формирование навыков работы с массивами. Разработка алгоритма поиска элемента с максимальным (минимальным) значением.

**Алгоритм поиска минимального элемента**

Схема поиска элемента с минимальным значением может быть следующей: определим как минимальный элемент первый элемент массива. Начнем просматривать массив со второго элемента и до последнего. Если текущее значение минимума стало больше просматриваемого элемента массива, то заменим текущее значение минимума на значение просматриваемого элемента массива.

алг Поиск минимального элемента в массиве нач
цел *i, n*
вещ *X[n], min*
ввод массив $X$ из $n$ элементов;
$\min = X[1];$
цикл для *i* от 2 до *n* нц
если $X[i] < \min$ то $\min := X[i]$
кц
рез $\min$;
кон

Предложенный алгоритм находит значение минимального элемента, но не указывает, где этот элемент находится. Если задача нахождения экстремального элемента сводится к поиску его местоположения в массиве, то алгоритм может быть представлен следующей блок-схемой:




**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа №8.** ***Массивы. Изменение массива***

**Цель работы:** формирование навыков работы с массивами. Разработка алгоритмов изменяющих массив.

**Динамические массивы**

При решении задач очень часто размер массива является переменной величиной. В этом случае используются динамические массивы. Инициализация таких массивов выполняется в два этапа. Первый этап - выделение памяти, второй этап – инициализация значений элементов массива. Элементы массива могут вводиться с клавиатуры:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  system("chcp 1251");
// Описание указателя на целое число
  int *x;
  int n,i;
  printf("Введите размерность массива: ");
  scanf("%d",&n);
// выделение памяти
  x = (int*)malloc(sizeof(int)*n);
  for(i=0;i<n;i++){
   printf("x[%d] -> ",i);
// чтение элементов с клавиатуры
   scanf("%d",&x[i]);
  }
// очистка экрана
  system("cls");
// печать элементов массива
  for(i=0;i<n;i++) printf("%5d",x[i]);
 // освобождение памяти
 free(x);
  printf("\n");
  system("PAUSE");
  return 0;}
```

Элементы массива могут вычисляться по формуле:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  system("chcp 1251");
// Описание указателя на целое число
  int *x;
  int n,i;
  printf("Введите размерность массива: ");
  scanf("%d",&n);
// выделение памяти
  x = (int*)malloc(sizeof(int)*n);
for(i=0;i<n;i++){
// вычисление значения элемента массива  x_i = i + 2i
        x[i] = i+2*i;
```

```
// печать элементов массива
        printf("%d ",x[i]);
}
 free(x);
 printf("\n");
 system("PAUSE");
 return 0;
}
```

Элементы массива можно задавать случайным образом. Для этого можно использовать функцию *rand ()*. Функция возвращает равномерно распределенную целую случайную величину в интервале от 0 до *MAXINT*. Прототип функции находится в заголовочном файле *stdlib.h*. Следующий фрагмент задает случайным образом элементы целочисленного массива *x*, значения элементов массива лежат в интервале от [20,50].

```
int i;
for( i=0;i<n;i++)
   {
         x[i] = rand()%31+20;
         printf("%d ",x[i]);
 }...
```

С помощью функции *rand* можно получать и отрицательные значения:

```
for(int i=0;i<n;i++){
        x[i] = rand()%31-rand()%31;
        printf("%d ",x[i]);
}...
```

А так же вещественные значения:

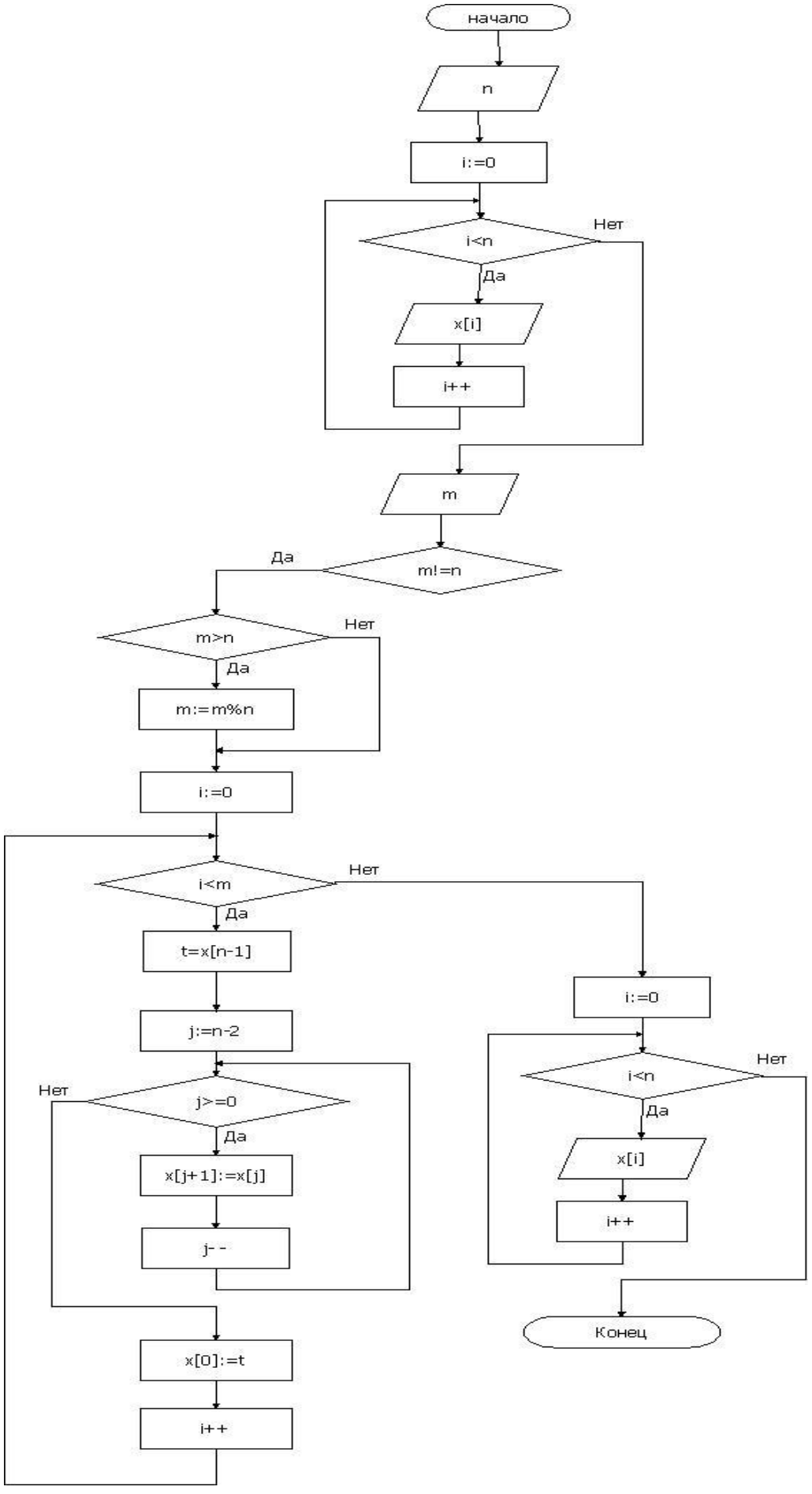
```
float *x;
...
x = (float*)malloc(sizeof(float)*n);
for(int i=0;i<n;i++){
x[i] = rand()%101/(rand()%31+1.);
printf("%7.2f ",x[i]);
}
...
```

Обратите внимание на знаменатель дроби. К полученному случайному числу прибавляется 1., для того, чтобы знаменатель не обратился в 0 (на ноль делить нельзя). Единица должна быть вещественной, тогда знаменатель по правилам преобразования типов считается вещественной величиной и при делении целого числа на вещественное число результат также считается вещественным.

Попробуйте изменить вещественную константу 1. на целую константу 1 и объяснить полученный результат.

Задачи на изменение массива могут быть сформулированы самыми разнообразными способами. Рассмотрим несколько подобных задач.

*Дан массив из n элементов. Необходимо сдвинуть элементы массива на m позиций вправо.*


начало
n
i:=0
i<n
Нет
Да
x[i]
i++
m
m!=n
Да
m>n
Нет
Да
m:=m%n
i:=0
i<m
Нет
Да
t=x[n-1]
j:=n-2
j>=0
Нет
Да
x[j+1]:=x[j]
j- -
x[0]:=t
i++
i:=0
i<n
Нет
Да
x[i]
i++
Конец

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа №9.** ***Сортировка массивов***

**Цель работы:** формирование навыков работы с массивами. Реализация алгоритмов сортировки элементов массива.

**Сортировка массивов**

*Сортировка обменом*

```
алг  Сортировка обменом нач
        ввод  n , X[n].
        цикл для i от 0 до n-1
                flag := 0; //  Обменов нет
           цикл для j от 0 до n-i-1
                если  X[j]>X[j+1] то
                                обмен   X[j]↔X[j+1]
                                flag := 1; // Обмен есть
           кц
// Если обменов не было
          если  flag==0 то
         Закончить выполнение цикла  по i.
 кц
рез X.
кон
```

*Сортировка выбором*

```
алг Сортировка выбором нач
ввод n , X[n]
цикл для i от 0 до n-1
// Считать минимальным элемент, расположенный на позиции k.
                k:=i
                цикл для j от i+1 до n
// Если элемент на позиции j меньше минимального, то изменить позицию
        если X[j]<X[k] то k:=j
                кц
// Если минимальный элемент не стоит на позиции i, то выполнить обмен
         если k ≠ i  то Обмен  X[i]↔X[k]
кц
рез X
кон
```

*Сортировка вставками*

```
алг Сортировка вставками нач
ввод n , X[n]
цикл для i от 1 до n
//  Сохранить значение вставляемого элемента во  временной переменной
     buf := X[i];
// начать просмотр элементов от j до 0
    j := i;
        // пока текущий элемент меньше предыдущего
    пока buf<X[j-1] И j>0
```

```
// заменить текущий элемент на предыдущий
                X[j] = X[j-1]
                j = j-1;
            кц
           X[j] := buf;
кц
рез X
кон
```

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа № 10. *Функции***

**Цель работы:** формирование навыков написания пользовательских функций.

**Синтаксис**

Синтаксически любая функция языка Си описывается следующим образом:

```
< тип возвращаемого результата> имя функции (<список формальных аргументов>)
{
  тело функции
}
```

**Объявление и вызов функций**

В языке Си объявить функцию можно несколькими способами. Первый способ – написать функцию до функции *main().*

Функция поиска минимального числа из трех заданных целых чисел.

```
#include <stdio.h>
#include <stdlib.h>
// Объявление функции
int min(int a1, int a2, int a3)
//  Тип возвращаемого результата – целый, формальные аргументы – три
// целых числа a1,a2,a3.
 {
   int m = a1;
   if (m>a2) m = a2;
   if (m>a3) m = a3;
   return m; // возвращение результата с помощью оператора return
 }
int main(int argc, char *argv[])
{
  system("chcp 1251");
  int  x,y,z;
  printf("Введите значение x - ");
  scanf("%d",&x);
  printf("Введите значение y - ");
  scanf("%d",&y);
  printf("Введите значение z - ");
  scanf("%d",&z);
  int k = min(x,y,z); // вызов функции с фактическим аргументами x,y,z.
// переменной k присваивается возвращаемое значение
  printf("\n Минимальное значение - %d \n",k);
system("pause");
```

```
return 0;
}
```

Сама функция может быть написана и после тела вызывающей функции, или даже в другом файле, но в этом случае необходимо использовать прототипы функции. Прототипом функции называется указание типа возвращаемого результата, имени функции и списка типов формальных аргументов. Например, для функции *min* прототип будет выглядеть следующим образом:

```
int min(int,int,int);
```

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа № 11. *Обработка матриц***

**Цель работы:** формирование навыков обработки двумерных массивов.

**Инициализация матриц**

```
int **x;
x=(int**)malloc(sizeof(int*)*n); // массив из указателей.
for( i=0;i<n;i++)
x[i] = (int)malloc(sizeof(int)*m); // массив из m элементов.
```

У матрицы заданной таким образом *n* строк и *m* столбцов.

Перед окончанием программы, работающей с динамической матрицей необходимо освободить используемую память. Освобождение памяти проходит так же в два этапа, но в обратном порядке:

```
for(i=n-1;i>=0;i--)
  free(x[i]);
free(x);
```

Для перебора элементов матрицы используются не один, а два цикла:

```
for(i=0;i<n;i++)
        for(j=0;j<m;j++)
                Обращение к элементу x[i][j]
```

Первый индекс элемента определяет номер строки, в которой он расположен, а второй – номер столбца.

Циклы, организованные выше, просматривают матрицу по строкам: элементы нулевой строки, элементы первой строки и т.д..

А такие циклы –

```
for(i=0;i<m;i++)
        for(j=0;j<n;j++)
                Обращение к элементу x[j][i]
```

просматривают элементы матрицы по столбцам, начиная с нулевого.

Задать элементы матрицы можно такими же способами, как и элементы массива: ввести с клавиатуры, задать случайным образом, рассчитать по формуле.

**Печать матриц**

Для вывода элементов матрицы на экран так же используются два цикла. Организуем эти два цикла следующим образом: внешний цикл перебирает строки, внутренний - столбцы, и как только на экран выведены все элементы одной строки, вывод следующей нужно организовать с новой строки экрана:

```
// печать элементов вещественной матрицы x
for(i=0;i<n;i++)
 {
```

```
 for( j=0;j<m;j++)
   printf("%8.3f ",x[i][j]);
// переход на новую строку экрана
 printf("\n");
 }
```

В целочисленной матрице размерности $n \times m$ упорядочить столбцы матрицы по элементам первой строки. Размерность матрицы задавать с клавиатуры. Элементы матрицы определить случайным образом. Решение оформить в виде функций.

```
#include <stdio.h>
#include <stdlib.h>
int ** Create(int n,int m){
srand();
int i,j;
int** x = (int**)malloc(sizeof(int)*n);
for(i=0;i<n;i++)
 x[i] = (int*)malloc(sizeof(int)*m);
for(i=0;i<n;i++)
 for(j=0;j<m;j++)
    x[i][j] = rand()%10;
return x;
}
void Print(int **x,int n,int m){
int i,j;
for( i=0;i<n;i++)
 {
 for( j=0;j<m;j++)
   printf("%6d ",x[i][j]);
 printf("\n");
 } }
// Функция сортировки столбцов матрицы.
int ** Sort(int **x, int n, int m){
int flag;
int i,j,k;
for( i=0;i<m-1;i++)
{
// Обменов не было.
  flag  = 0;
  for ( j=0;j<m-1-i;j++)
  {
// Сравнение двух рядом стоящих элементов первой
// строки. Если первый элемент пары больше второго
    if (x[0][j]>x[0][j+1])
    {
// зафиксируем обмен.
      flag = 1;
// выполним обмен столбцов.
      for (int k=0;k<n;k++)
       {
           int temp = x[k][j];
           x[k][j] = x[k][j+1];
           x[k][j+1] = temp;
       }
```

```
    }
   }
  // Если обменов не было, то закончить сортировку.
    if (!flag) break;
}
return x;
}
void Delete(int ** x,int n)
{
int i;
 for( i=n-1;i>=0;i--)
   free( x[i]);
 free(x);
}
int main(int argc, char *argv[])
{
int **I;
int n,m;
system("chcp 1251");
printf ("Введите количество строк матрицы: ");
scanf("%d",&n);
printf ("Введите количество столбцов матрицы: ");
scanf("%d",&m);
I = Create(n,m);
printf("Исходная матрица: \n ");
Print(I,n,m);
printf("Преобразованная матрица: \n ");
// Вызов функции сортировки.
I = Sort(I,n,m);
Print(I,n,m);
Delete(I,n);
system("PAUSE");
return 0;
}
```

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа № 12. *Обработка строк***

Цель работы: формирование навыков работы со строками.

**Инициализация и описание строк**

Строка с постоянной длиной:

*char str[25]; //* строка из 24 символов.

Динамическая строка:

*char *str = (char*)malloc(sizeof(char)*25); //* строка из 24 символов.

Начальная инициализация:

*char str[25] = "Пример строки";*

чтение с клавиатуры: *scanf("%s",str); str = gets(str);*

задание строки посимвольно – *str[i] = 'A';*

**Стандартные функции для работы со строками**

Прототипы ниже описанных функций находятся в заголовочном файле *string.h.*

```
char *strcat(char *dest, const char *src);
char *strchr(const char *s, int c);
int strcmp(const char *s1, const char *s2);
int strcmpi(const char *s1, const char *s2);
char *strcpy(char *dest, const char *src);
size_t strcspn(const char *s1, const char *s2);
size_t strlen(const char *s);
char *strlwr(char *s);
char *strncat(char *dest, const char *src, size_t maxlen);
int strncmp(const char *s1, const char *s2, size_t maxlen);
char *strncpy(char *dest, const char *src, size_t maxlen);
char *strnset(char *s, int ch, size_t n);
char *strpbrk(const char *s1, const char *s2);
char *strrchr(const char *s, int c);
char *strrev(char *s);
char *strset(char *s, int ch);
size_t strspn(const char *s1, const char *s2);
char *strstr(const char *s1, const char *s2);
double strtod(const char *s, char **endptr);
char *strtok(char *s1, const char *s2);
long strtol(const char *s, char **endptr, int radix);
unsigned long strtoul(const char *s, char **endptr, int radix);
char *strupr(char *s);
```

Дана произвольная строка символов, найти слово с максимальной длиной и вывести его на экран.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
char dest[200];
printf("Вводите строку символов: ");
gets(dest);
char *buf = ( char*)malloc(sizeof(char)*(strlen(dest)+1));
strcpy(buf,dest);
// Выделить первое слово строки
char *temp = strtok(buf," ,;.!?-");
// Принять это слово за слово с максимальной длиной
int max = strlen(temp);
char *strmax = ( char*)malloc(sizeof(char)*(max+1));
strcpy(strmax,temp);
// Выделять последующие слова и сравнивать их со
// словом с максимальной длиной
while (temp!=NULL)
{
 temp = strtok(NULL," ,;.!?-");
 if (!temp)break;
 if (max<strlen(temp))
  {
        free( strmax);
        max = strlen(temp);
        strmax = (char*)malloc(sizeof(char)*(max+1));
        strcpy(strmax,temp);
```

```
  } }
 printf("В строке // %s // \n слово с максимальной длиной // %s //",dest,strmax);
 printf("\n Его длина - %d",max);
 free( strmax);
 free( buf);
system("PAUSE");      return 0;
 }
```

В произвольной строке символов найти количество символов, не являющихся буквами.

Для решения этой задачи используется функция *isalpha()*, которая передает ненулевое значение, если проверяемый символ является буквой и нуль в противном случае. Прототип функции описан в заголовочном файле *ctype.h*

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
 char dest[200];
  printf("Вводите строку символов: ");
 gets(dest);
 int k = 0,i;
 for(i=0;i<strlen(dest);i++)
 {
  if(!isalpha(dest[i])) k++;
 }
 printf("Количество символов строки, не являющихся буквами - %d\n",k);
system("PAUSE");      return 0;
}
```

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа № 13. *Текстовые файлы***

**Цель работы:** формирование навыков работы с текстовыми файлами.

Функция, с помощью которой открывают файл:

*FILE* fopen (const char *filename, const char* mode);* -возвращает указатель на переменную типа *FILE*, *mode* – заданный режим открытия файла. При неуспешной работе функция возвращает *NULL*. Во избежание ошибок при открытии файла необходимо проверять результат выполнения функции, например, как в следующей программе:

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
 FILE *f;
 char name[] = "prim.txt";
if ((f = fopen(name,"rb"))==NULL) {
  printf( "Ошибка открытия файла: ");
system(“pause”);
  }
  else…
…
}
```

*Режимы доступа к файлу:*

*r* - открыть для чтения.

*w* - создать для записи. Если файл с таким именем уже существует, он будет перезаписан.

*a* – открыть файл для обновления, открывает файл для записи в конец файла или создает файл для записи, если файла не существует.

*r+*- открыть существующий файл для корректировки (чтения и записи).

*w+*- создать новый файл для корректировки (чтения и записи). Если файл с таким именем уже существует, он будет перезаписан.

*a+* - открыть для обновления; открывает для корректировки (чтения и записи) в конец файла, или создает, если файла не существует.

*t* – открыть файл в текстовом режиме.

*b*– открыть файл в двоичном режиме.

Два эти аргумента указываются вторыми символами в строковой переменной *mode,* например "*r+b*".

По умолчанию установлен текстовый режим доступа к файлу.

*int fclose(FILE *fp)* - закрывает файл *fp,* при успешной работе возвращает 0, при неуспешной *EOF).*

*int closeall(void)* – закрывает все файлы, открытые в программе, при успешной работе возвращает число закрытых потоков, при неуспешной - *EOF.*

Запись данных в текстовый файл

В текстовый файл можно записать данные различных типов. В дальнейшем, содержимое такого файла можно просмотреть в любом текстовом редакторе.

Создать вещественный массив случайным образом и сохранить его в текстовом файле.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
  char name[25];
 int n;
 FILE *f;
 int flag = 1,i;
do
 {
 printf( "Введите имя создаваемого файла: ");
 scanf("%s",name);
if ((f = fopen(name,"r"))!=NULL)
 {
  printf("Файл уже существует. Заменить? (y/n)");
  char ch = getch();
  if (ch == 'n') { continue;}
  }
  if ((f=fopen(name,"w"))==NULL)
   {
    printf("Ошибка создания файла");
    system("pause");
    break;
   }
  printf("\nВведите размерность массива: ");
  scanf("%d",&n);
  for( i=0;i<n;i++ )
   { float y = rand()%100/(rand()%50+1.)-rand()%30;
```

```
    if (i>=10&&i%10==0) fprintf(f,"\n");
    fprintf(f,"%8.3f",y);
   }
   fclose(f);
   flag = 0;
  } while (flag);
  printf("Файл создан");
 system("PAUSE");
 return 0;
 }
```

Чтение данных из текстового файла

На первой строке в файле записана размерность целочисленной матрицы. Далее – сама матрица. Считать матрицу в память и вывести ее на экран. Данные записаны в файле *my.txt.* Если не указан полный путь к файлу, то файл должен находиться в папке проекта.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
FILE *f;
f = fopen("my.txt", "r");
if (f==NULL) {
printf("Файл не найден... /n ");
system(“pause”);
return 0;
}
int n,m,i,j;
fscanf(f,"%d",&n);
fscanf(f,"%d",&m);
int **a;
a = (int**)malloc(sizeof(int*)*n);
for( i=0;i<n;i++)
 a[i] = (int*)malloc(sizeof(int)*m);
for(i=0;i<n;i++)
  for( j=0;j<m;j++)
    fscanf(f,"%d",&a[i][j]);
printf("Прочитана матрица: \n");
for(i=0;i<n;i++)
{
  for( j=0;j<m;j++)
     printf("%5d",a[i][j]);
  printf("\n");
}
system("PAUSE");
 return 0;
}
```

В текстовом файле записано произвольное количество целых чисел. Не считывая всю информацию в память, найти максимальное число и дописать в файл строку, содержащую найденное значение и его порядковый номер в файле. Гарантируется, что в файле записано хотя бы одно число.

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
```

```
{
system("chcp 1251");
FILE *f = fopen("text.txt","r");
FILE *f1 = fopen("text.txt","a");
if (f==NULL) {
        printf("Файл не найден. \n");
        system("PAUSE");
         return 0;
        }
int max, nmax = 1,a,i=1;
fscanf(f,"%d",&max);
while (!feof(f)){
 if ( fscanf(f,"%d",&a)!=1) break;
 i++;
 if (max<a) {max = a; nmax = i;}
}
fprintf(f1,"\nМаксимальное число - %d[%d]\n",max,nmax);
fclose(f);
fclose(f1);
system("pause");
return 0;
}
```

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа № 14.** ***Двоичные файлы***

Цель работы: формирование навыков работы с двоичными файлами. Организация прямого доступа к содержимому файла.

*Запись и чтение информации в двоичный файл.*

Рассмотрим сохранение и последующее чтение числовой информации в двоичном представлении.

Записать в двоичный файл *n* вещественных чисел, прочитать созданный файл и вывести на экран в виде матрицы с числом столбцов *m*. Решение оформить в виде функций.

```
void create_file(char * name, int n);
void read_file(char* name, int m);
#include <stdio.h>
#include <stdlib.h>
void create_file(char *name, int n)
{
  int i;
 FILE *f = fopen(name,"wb");
 if (f==NULL) {
        printf("Ошибка создания файла. \n");
        system("pause");
        return ;                }
 for( i=0;i<n;i++)
 {
  float z = rand()%200/(rand()%100+1.)-rand()%70;
  fwrite(&z,sizeof(float),1,f);
```

```
  }
  fclose(f);
}
void read_file(char *name, int m)
{
 FILE *f = fopen(name,"rb");
 if (f==NULL) {
          printf("Файл не найден. \n");
          system("pause");
          return ;
          }
 int i = 0;
 float z;
 while(!feof(f))
 {
  if (i>=m&&i%m==0)
     printf("\n");
  if(fread(&z,sizeof(float),1,f)!=1) break;
  printf("%8.3f",z);
  i++;
  }
  fclose(f);
}
int main(int argc, char *argv[])
{
char Fname[30];
int n,m;
printf("Введите имя создаваемого файла: ");
scanf("%s",Fname);
printf("Введите количество записываемых чисел: ");
scanf("%d",&n);
printf("Введите количество столбцов: ");
scanf("%d",&m);
create_file(Fname,n);
read_file(Fname,m);
system("pause");
return 0;
}
```

*Обратите внимание:* значение функции *feof* формируется только при попытке чтения из файла. Поэтому, внутри цикла выполнена проверка значения функции *fread:*

*if(fread(&z,sizeof(float),1,f)!=1) break;*

Если чтение на данном шаге проведено неуспешно (найден конец файла), то необходимо закончить работу цикла.

Реализация прямого доступа в двоичном файле

В некоторых задачах требуется читать только указанную информацию. Механизм работы с файлами в Си позволяет обращаться к элементам, записанным на заданных позициях файла, без чтения предыдущих элементов.

В двоичном файле сохранена следующая информация – размерность *n* квадратной матрицы и сама вещественная матрица. Вывести на экран элементы заданного столбца *k.*

В двоичном файле элементы матрицы сохранены следующим образом:

*x[0][0],x[0][1],…x[0][n],x[1][0],…,x[n-1][n-2],x[n-1][n-1].*

Очевидно, что позиция *k*-того элемента рассчитывается по формуле:

*k*sizeof(*элемента*)+i*sizeof(*элемента*),*

где *i* – номер строки.

Воспользуемся этой закономерностью для решения задачи:

```
#include <stdio.h>
#include <stdlib.h>
void create_file(char *name, int n)
{
  FILE *f = fopen(name,"wb");
  if (f==NULL) {
            printf("Ошибка создания файла. \n");
            system("pause");
            return ;
            }
  fwrite(&n,sizeof(n),1,f);
int i;
  for( i=0;i<n*n;i++)
  {
   float z = rand()%200/(rand()%100+1.)-rand()%70;
   fwrite(&z,sizeof(float),1,f);
   }
   fclose(f);
}
void read_file(char *name)
{
  int m;
  FILE *f = fopen(name,"rb");
  if (f==NULL) {
            printf("Файл не найден. \n");
            system("pause");
            return ;
            }
  int i = 0;
  float z;
  fread(&m,sizeof(int),1,f);
while(!feof(f))
  {
   if (i>=m&&i%m==0)
      printf("\n");
   if(fread(&z,sizeof(float),1,f)!=1) break;
   printf("%8.3f",z);
   i++;
   }
   fclose(f);
}
void read_k(char *name, int k)
{
  int m;
  FILE *f = fopen(name,"rb");
  if (f==NULL) {
            printf("Файл не найден. \n");
            system("pause");
            return ;
```

```
            }
  float z;
int i;
  fread(&m,sizeof(int),1,f);
  for( i=0;i<m;i++)
  {
    int l =i*m*sizeof(float) + k*sizeof(float)+sizeof(int);
    fseek(f, l,SEEK_SET);
    fread(&z,sizeof(z),1,f);
    printf("%8.3f",z);
   }
   fclose(f);

}
int main(int argc, char *argv[])
{
char Fname[30];
int n,m;
printf("Введите имя создаваемого файла: ");
scanf("%s",Fname);
printf("Введите количество строк матрицы: ");
scanf("%d",&n);
printf("Введите номер столбца: ");
scanf("%d",&m);
create_file(Fname,n);
printf("Матрица: \n");
read_file(Fname);
printf("Элементы столбца с номером %d \n",m);
read_k(Fname,m);
system("pause");
return 0;
} [.]
```

Аналогичным образом решаются задачи, требующие изменения уже существующих файлов.

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа № 15.** ***Рекурсивные функции***

**Цель работы**: формирование навыков оформления алгоритмов с помощью рекурсивных функций.

Рекурсивными называются функции, вызывающие сами себя. Запишем с помощью рекурсивной функции получение *n*-того числа Фибоначчи.

Числа *Fn* , образующие последовательность *1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233,* ... называются *«числами Фибоначчи»*, а сама последовательность - последовательностью Фибоначчи.

Суть последовательности Фибоначчи в том, что начиная с 1,1 следующее число получается сложением двух предыдущих.

$$x_i = x_{i-1} + x_{i-2}, i = \overline{2,\infty}, x_0 = 1, x_1 = 1$$

Параметром функции будет значение *n* – номер искомого числа Фибоначчи.

Написание рекурсивной функции, как правило, начинается с формулирования условия выхода из функции. Если правило выхода не предусмотрено, функция будет бесконечной.

По определению последовательности, функция должна возвращать 1, если *n* равно 0, или *n* равно 1. Во всех остальных случаях функция должна возвращать сумму результатов при *n-2* и *n-1*.

```
#include <stdio.h>
#include <stdlib.h>
int func(int n)
{
   if (n==0) return 1;
   if (n==1) return 1;
   return (func(n-1)+func(n-2));
  }
int main(int argc, char *argv[])
{
 system("chcp 1251");
int n;
printf("Введите целое n: ");
scanf("%d",&n);
int x  = func(n);
printf(" x [%d] = %d\n",n,x);
system("pause");
return 0;}
```

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа № 16.** ***Простые сортировки***

**Цель работы:** исследование простых алгоритмов сортировок.

Ниже рассмотрен пример, подсчитывающий количество сравнений и перестановок, выполняющихся алгоритмом сортировки обменом.

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
  // файл создается в текущей директории
    system("chcp 1251"); // смена кодировки консоли
    FILE *file; // Описание указателя на файл
    file = fopen("Data.txt","w"); // создание файла с именем Data.txt для записи
    // имя файла произвольное
    if (!file) {printf("Ошибка создания файла"); // проверка ошибки при открытии файла
              system("PAUSE");
              return 0;}
    while (1) { // Организуем бесконечный цикл
    printf("Введите n:");
    int n;
    scanf("%d",&n);
```

```
   int *x = (int*)malloc(sizeof(int)*n);
   fprintf(file,"Размерность: %d",n); // Печать размерности в файл
   float compare=0, shift=0;
   int temp,flag;
   // Печать исходного массива
   int i,j;
   srand (time(NULL));
   for(i=0;i<n;i++)
     { x[i] = rand()%20 - rand()%15;
       printf("%4d",x[i]);
           }
   //Сортировка
   for(i=0;i<n-1;i++){
      flag = 0;
      for(j=0;j<n-i-1;j++)
      {   compare++; // считаем сравнения
          if (x[j]>x[j+1])
            {temp = x[j];
             x[j] = x[j+1];
             x[j+1]= temp;
             shift+=3; // считаем перестановки
             flag++;
             }
             }
      if (!flag) break;
             }
   printf("\nОтсортированный массив: ");
    for(i=0;i<n;i++)
     printf("%4d",x[i]);
   fprintf(file, "\nКоличество сравнений : %.0f \n", compare); // Печать в файл
   fprintf(file, "Количество перестановок: %.0f \n",shift); //
   free( x);
   // Выполнить сортировку другого массива?
   printf("\nВыполнить сортировку другого массива? (1 - да, 0 - нет)");
   scanf("%d",&temp);
   if (!temp) break; // если введен 0, то выход из бесконечного цикла
    }
   fclose(file);    // закрытие файла
   printf("\n");
   system("PAUSE");
   return 0;
}
```

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм, подсчитывающий количество сравнений и перестановок, выполняемых заданным способом сортировки.
3. Напишите и протестируйте программу.
4. Выполните программу при различных значениях размерности массива (*n = 100, 1000, 10000* элементов) 5-10 раз.
5. Рассчитатйте среднее значение сравнений и перестановок при разных размерностях массивов.
6. Подготовьте отчет по лабораторной работе.

**Лабораторная работа № 17.** ***Улучшенные сортировки***

***Цель работы:*** ознакомиться и реализовать оптимизированные методы сортировок – шейкерную сортировку (оптимизация обмена), сортировку бинарными вставками и сортировку вставками со сторожевым элементом (оптимизация простых вставок).

***Шейкерная сортировка***

**Сртировка перемешиванием**, или Шейкерная сортировка, или двунаправленная (англ. *Cocktail sort*) — разновидность пузырьковой сортировки. Анализируя метод пузырьковой сортировки, можно отметить два обстоятельства.

***Во-первых***, если при движении по части массива перестановки не происходят, то эта часть массива уже отсортирована и, следовательно, её можно исключить из рассмотрения.

***Во-вторых***, при движении от конца массива к началу минимальный элемент "всплывает" на первую позицию, а максимальный элемент сдвигается только на одну позицию вправо.

Эти две идеи приводят к следующим модификациям в методе пузырьковой сортировки. Границы рабочей части массива (т.е. части массива, где происходит движение) **устанавливаются в месте последнего обмена** на каждой итерации. Массив просматривается поочередно справа налево и слева направо.

***Сортировка бинарными вставками***

Сортировка бинарными вставками некоторым образом улучшает алгоритм сортировки вставками, увеличивая скорость нахождения места для вставляемого элемента. Метод использует информацию о том, что часть выборки уже отсортирована. Тогда, можно организовать поиск места для вставки нового элемента следующим образом: пусть переменная *F* обозначает индекс начала упорядоченной последовательности, а *L* - индекс конца. Найдем индекс элемента, находящегося в центре отсортированной последовательности - $M = {(F+L)}/{2}$. Сравним вставляемый элемент *X[j]* с элементом *X[M]*. Если *X[j]> X[M]*, то изменим *F* на *M* (будем искать в правой части последовательности). Если *X[j]< =X[M]*, то изменим *L* на *M* (будем искать в левой части последовательности). Описанные действия будем проводить до тех пор, пока *L>F*. После выхода из цикла место для вставки элемента найдено.

Рассмотрим работу алгоритма на примере уже упорядоченной последовательности:

*1 2 3 5 6 7 8 9 10* ***4.***

*F=0, L = 8, M = 4 X[4]=6 > 4, следовательно:*

*F = 0 L = 4 M = 2 X[2]=3 < 4, следовательно:*

*F = 2 L = 4 M = 3 X[3]=5 >4, следовательно:*

*F = 2, L = 3 M = 2 ( по правилам деления целых чисел в Си 5/2 = 2) X[2]= 3 < 4, следовательно F = 3 L = 3.*

Так как *F* и *L* равны (условие выполнения цикла), то вставляемый элемент должен занимать в упорядоченной последовательности место с номером 3.

***Сортировка вставками со сторожевым элементом***

Еще одна оптимизация метода простых вставок. Очевидно, что на каждом шаге внутреннего цикла выполняется два сравнения – вставляемый элемент сравнивается с просматриваемым и отслеживается возможный выход за границы массива. Второго сравнения можно избежать, проведя перед сортировкой некоторые мероприятия.

1 вариант. Перед сортировкой можно найти минимальный элемент и поменять его местами с первым элементом массива. Таким образом, ни один из оставшихся элементов массива не будет больше первого элемента и второе сравнение во внутреннем цикле можно опустить.

2 вариант. При инициализации массива выделяется память для хранения n+1 элемента (n – размерность массива). Все элементы массива записываются, начиная со второго элемента. Далее, на каждом шаге внешнего цикла в первую позицию записывается вставляемый эле-

мент. После выполнения этих действий так же можно опустить второе сравнение во внутреннем цикле.

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм, подсчитывающий количество сравнений и перестановок, выполняемых заданным способом сортировки.
3. Напишите и протестируйте программу.
4. Выполните программу при различных значениях размерности массива (*n = 100, 1000, 10000* элементов) 5-10 раз.
5. Рассчитатйте среднее значение сравнений и перестановок при разных размерностях массивов.
6. Подготовьте отчет по лабораторной работе.

## Лабораторная работа № 18 *Код Грея*

**Алгоритм построения бинарного кода Грея**

Описанный далее алгоритм генерирует последовательность всех подмножеств *n*-элементного множества таким образом, что каждое последующее множество получается из предыдущего добавлением или удалением одного элемента.

Код Грея называется так же отраженным кодом. Рассмотрим построение кода на примере *n= 4.*

Будем считать старшим разрядом нулевой разряд. Он может принимать значения 0 и 1.

0000

0001 Далее старший разряд первый, который принимает значения 1, а младший разряд (нулевой) принимает значения в обратном порядке от предыдущего

0011

0010 Далее аналогичным образом: (старший разряд выделен размером, отражаемая часть жирным шрифтом)

0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001

Пронумеруем полученные наборы от 0 до 14.

| | | | |
|---|---|---|---|
| **0** | 0000 | **7** | 0100 |
| **1** | 0001 | **8** | 1100 |
| **2** | 0011 | **9** | 1101 |
| **3** | 0010 | **10** | 1111 |
| **4** | 0110 | **11** | 1110 |
| **5** | 0111 | **12** | 1010 |
| **6** | 0101 | **13** | 1011 |
| | | **14** | 1001 |

В первом наборе инвертировался разряд с номером 0, во втором – разряд с номером 1, в третьем – разряд с номером 0, в четвертом разряд с номером 2 и т.д.. Разложим числа от 1 до

14 на простые множители и подсчитаем количество двоек в разложении числа. В итоге получена та же самая последовательность**.**

| Число | Разложение | Количество двоек в разложении числа | Число | Разложение | Количество двоек в разложении числа |
|---|---|---|---|---|---|
| 1 | 1 | 0 | 8 | 2*2*2*2 | 3 |
| 2 | 2 | 1 | 9 | 3*3 | 0 |
| 3 | 3 | 0 | 10 | 2*5 | 1 |
| 4 | 2*2 | 2 | 11 | 11 | 0 |
| 5 | 5 | 0 | 12 | 2*2*3 | 2 |
| 6 | 2*3 | 1 | 13 | 13 | 0 |
| 7 | 7 | 0 | 14 | 2*7 | 1 |

1. Задать *A* – множество из *n* элементов.
2. Задать *M* = [000..]- подмножество булеана.
3. Вывести *M*;
4. Цикл $\left(i = \overline{1;2^n - 1}\right)$
   4.1. Найти *k* – количество двоек в разложении числа *i* ;
   4.2. Если *M*[*k*] = 0 То {*M[k] = 1*
   Иначе *M[k]* = 0;
   4.3. Вывести *M*;
5. Конец цикла
6. Конец

**Реализация кода Грея с помощью стека.**

1. СТЕК ← пустой стек
2. Цикл ( $j = \overline{n-1;0}$ )
   2.1. $g_j = 0$
   2.2. СТЕК ← *j*
3. Конец цикла
4. Печать $(g_{n-1}, g_{n-2}, \ldots, g_0)$
5. Пока (СТЕК не пуст);
   5.1. СТЕК → *a*
   5.2. $g_a := \overline{g_a}$ {Инвертировать $g_a$ }
   5.3.Печать $(g_{n-1}, g_{n-2}, \ldots, g_0)$
   5.4. Цикл ( $j = \overline{a-1;0}$ )
      5.4.1. СТЕК ← *j*
   5.5. Конец цикла
6. Конец цикла

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа № 19.** ***Машинное представление графов***

***Цель работы:*** ***р***азработка и реализация алгоритмов преобразования различных форм представления графов.

***Матрица смежности***

Матрицей смежности неориентированного графа *G=(X,U)*, $|X|=n, |U|=m$ называется квадратная матрица $A[n\times n]$, элементы которой задаются по правилу 1.

$$a_{i,j}=\begin{cases}1, & \text{если вершина } x_i \text{ смежна вершине } x_j;\\ 0, & \text{если вершина } x_i \text{ не смежна вершине } x_j;\end{cases} \quad (1)$$

Матрицей смежности ориентированного графа *G=(X,U)*, $|X|=n, |U|=m$ называется квадратная матрица $A[n\times n]$, элементы которой задаются по правилу 2.

$$a_{i,j}=\begin{cases}1, & \text{если вершина } x_i \text{ смежна вершине } x_j;\\ -1, & \text{если вершина } x_j \text{ смежна вершине } x_i;\\ 0, & \text{если вершина } x_i \text{ не смежна вершине } x_j;\end{cases} \quad (2)$$

***Матрица инцидентности***

Матрицей инцидентности неориентированного графа *G=(X,U)*, $|X|=n, |U|=m$ называется матрица $B[n\times m]$, элементы которой задаются по правилу 3.

$$b_{i,j}=\begin{cases}1, & \text{если вершина } x_i \text{ инцидентна ребру } u_j;\\ 0, & \text{если вершина } x_i \text{ не инцидентна ребру } u_j.\end{cases} \quad (3)$$

Матрицей инцидентности ориентированного графа *G=(X,U)*, $|X|=n, |U|=m$ называется матрица $B[n\times m]$, элементы которой задаются по правилу 4.

$$a_{i,j}=\begin{cases}1, & \text{если вершина } x_i \text{ начало дуги } u_j;\\ -1, & \text{если вершина } x_i \text{ конец дуги } u_j;\\ 2, & \text{если } u_j \text{ - петля при вершине } x_i;\\ 0, & \text{если вершина } x_i \text{ не инцидентна дуге } u_j.\end{cases} \quad (4)$$

***Список ребер***

Списком ребер неориентированного графа *G=(X,U)*, $|X|=n, |U|=m$ называются два массива $N[2m]$ и $K[2m]$, элементы которых задаются по правилу 5:

$n_i$ – вершина, являющаяся началом ребра с номером *i*.

$k_i$ – вершина, являющаяся концом ребра с номером *i*. (5)

Списком ребер ориентированного графа *G=(X,U)*, $|X|=n, |U|=m$ называются два массива $N[m]$ и $K[m]$, элементы которых задаются по правилу 6:

$n_i$ – вершина, являющаяся началом дуги с номером *i*.

$k_i$ – вершина, являющаяся концом дуги с номером *i*. (6)

***Структура смежности***

Структурой смежности графа *G=(X,U)*, $|X|=n, |U|=m$ называется массив списков $x_i : x_k, x_{k1}, .. x_{kz}$, $i=\overline{1,n}$,

где $x_k, x_{k1}, .. x_{kz}$ - все вершины, смежные вершине $x_i$.

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа №20.** ***Обходы графов***

***Цель работы:*** программная реализация основных алгоритмов на графах

***Алгоритмы обходов графа***

Дан граф *G=(X,U),* $|X|=n,|U|=m$.

1. Цикл ( $i=\overline{1,n}$ )
   1.1. *M[* $x_i$ *]=0; //* Все вершины не отмечены
2. Конец цикла
3. Выбрать произвольную вершину $v=x_i \in X$
4. $v \to T$ // Записать выбранную вершину в структуру T
5. $M[v]=1$// Отмечаем вершину, как пройденную
6. Пока (*T* не пуста)
   6.1. $u \leftarrow T$ // извлекаем вершину из структуры
   6.2. Печать *u*
   6.3. Цикл(для всех вершин *w*, смежных с *u*)
      6.3.1. Если (*M[w]=0)* То $w \to T$ ; *M[w]=1*
   6.4. Конец цикла
7. Конец цикла
8. Конец

Если структуру *T* определить как стек, то алгоритм обхода будет выполняться «в глубину». Если же *T* определена как очередь, будет выполняться обход «в ширину».

Рекурсивная реализация алгоритма обхода в глубину

```
нач Обход_в_глубину(G, Start, Finish)
      G - граф,
      Finish - начальная вершина,
      Start - конечная вершина
 пометить вершину Start как обработанную;
 выбрать дугу (Start, X), такую, что вершина X не обработана, если дуги нет - переход на 1.;
если в G есть дуга (Start, Finish) то  вернуть (true, (Start, Finish));
 (Res, Path) := Обход_в_глубину(G, X, Finish);
если Res = true то  вернуть (true, (Start, X) + Path);
вернуть Обход_в_глубину(G, Start, Finish);
1: конец - вернуть (false, ()).
```

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа №21.** ***Маршруты на графах***

***Цель работы:*** программная реализация основных алгоритмов на графах

*Алгоритм Дейкстры*

Алгоритм Дейкстры находит кратчайший путь между двумя заданными вершинами в орграфе.

Дан граф *G=(X,U),* $|X|=n,|U|=m$. Граф задан матрицей весов *C, s*– начальная вершина обхода, *z* – конечная вершина обхода.

На выходе алгоритма создаются два массива:

- *T* – если вершина *v* лежит на кратчайшем пути, то *T[v]*- длина кратчайшего пути от *s* к *v*.

- *H – H[v]* – вершина, непосредственно предшествующая *v* на кратчайшем пути.

1. Цикл( $x=\overline{1,n}$ )
   1.1. *T[x]=∞ //* кратчайший путь не известен
   1.2. *M[x] = 0 //* все вершины не отмечены
2. Конец цикла
3. *H[s]=0 //* *s* ничего не предшествует
4. *T[s]=0 //* кратчайший путь имеет длину 0
5. *M[s] = 1* // вершина *s* пройдена
6. *v = s //* зафиксируем текущую вершину
7. Цикл (1)
   7.1. Цикл(для $\forall u$ смежных с *v*)
      7.1.1. Если (*M[u]==0* и *T[u]>T[v]+C[v,u]*)
         То *T[u]=T[v]+C[v,u]; //* найден более
         // короткий путь из *s* в *u* через *v*
         *H[u]=v;*
   7.2. Конец цикла
   7.3. *t=∞*
   7.4. *v=0*
   7.5. Цикл( $x=\overline{1,n}$ )
      7.5.1. Если (*M[x]==0* и *T[x]<t*)
         То *v=x; t=T[x] //* вершина *v* заканчивает
         // кратчайший путь из *s.*
   7.6. Конец цикла
   7.7. Если (*v==0*) То «Нет пути из *s* в *z*»; Закончить выполнение цикла.
   7.8. Если (*v==z*) То «Путь найден»;
      Печать *T[v];*
      Печать *H[v]*.
   7.9. *M[v]=1, перейти на шаг 9.*
8. Конец цикла
9. Конец

*Алгоритм Форда*

Алгоритм Форда позволяет найти расстояние от заданной вершины *s* до всех вершин *T[v]=d(s,v),* ориентированного графа. Исходными данными для этого алгоритма является матрица весов дуг *C.* В отличие от алгоритма Дейкстры алгоритм может быть использован на графах с отрицательными длинами дуг.

Дан граф *G=(X,U),* $|X|=n, |U|=m$ . Граф задан матрицей весов *C. s* – вершина начала пути.

1. Цикл (для всех вершин *x* графа *G*)
   1.1. *D[x]:=C[s,x];*
2. Конец цикла
3. *D[s]:=0;*
4. Цикл ( $k=\overline{1,m-2}$ )

// последовательно перебрать все ребра

   4.1. Цикл (для всех вершин *v* графа, $v \neq s$ )
   // применить процесс поиска кратчайшего
   //пути для всех ребер
      4.1.1. Цикл (для всех вершин *u* графа, $u \neq v$ )
         4.1.1.1 Если (*D[v]>D[u]+C[u,v]*)
            То *D[v]=D[u]+C[u,v]*
      4.1.2. Конец цикла
   4.2. Конец цикла

// закончить алгоритм досрочно

5. Если нет изменений в *D*, То *k=n-2*
6. Конец цикла
7. Печать *D.*
8. Конец

*Алгоритм ближайшего соседа*

Существует другой метод получения кратчайшего остовного дерева, который не требует ни сортировки ребер, ни проверки на цикличность на каждом шаге, - так называемый *алгоритм ближайшего соседа*. Просмотр начинается с некоторой произвольной вершины *a* в заданном графе. Пусть *(a,b)*- ребро с наименьшим весом, инцидентное *a*; ребро *(a,b)* включается в остов. Затем среди всех ребер, инцидентных либо *a*, либо *b*, выбираем ребро с наименьшим весом и включаем его в частично построенное дерево. В результате этого в дерево добавляется новая вершина, например, *c*. Повторяя процесс, ищем наименьшее ребро, соединяющее *a, b* или *c* с некоторой другой вершиной графа. Процесс продолжается до тех пор, пока все вершины из *G* не будут включены в дерево, то есть пока дерево не станет остовным.

Дан граф *G=(X,U),* $|X|=n,|U|=m$. Граф задан матрицей весов *C*.

1. *T* – пустое множество ребер остовного дерева
2. *a* – произвольная вершина графа
3. $a \to T$
4. Пока ( $T \neq X$ )
   4.1. Найти ребро (*u,v*) с минимальным весом, такое, что $u \in T, v \notin T$ .
   4.2. $v \to T$
5. Конец цикла
6. Печать *T*
7. Конец

*Алгоритм Краскала*

Алгоритм Краскала относится к семейству «жадных» алгоритмов. Алгоритм ищет кратчайший остов в связном графе *G=(X,U),* $|X|=n,|U|=m$. Граф задан списком ребер *U* с длинами. На выходе алгоритма создается список *T* ребер кратчайшего остова.

1. *T* – пустой список
2. Упорядочить список *U* в порядке возрастания длин
3. *k = 1 //* номер рассматриваемого ребра
4. Цикл( $i=\overline{1,m-1}$ )
   4.1. Пока (добавление ребра *E[k]* образует цикл в *T*)
      4.1.1. *k=k+1*
   4.2. Конец цикла
   4.3. $E[k] \to T$
5. Конец цикла
6. Печать *T*
7. Конец

*Волновой алгоритм*

Волновой алгоритм является алгоритмом поиска кратчайшего пути между двумя вершинами в неориентированном ненагруженном графе.

Пусть дан граф *G=(X,U),* $|X|=n,|U|=m$. Вершина *a* – начало пути, вершина *b* – конец пути. Запишем вершину *a* во фронт волны нулевого уровня $FW_0$. Введем переменную *i =0.* Далее найдем все вершины, смежные вершинам $v \in FW_i$ и ранее не пройденные. Запишем эти

вершины во фронт волны следующего уровня $FW_{i+1}$. Увеличим *i - i = i +1.* Если среди найденных вершин есть вершина *b,* то длина кратчайшего пути найдена и равна *i*, если нет, то процесс продолжается до тех пор пока не будет найдена конечная вершина пути, либо, пока не будут просмотрены все вершины графа. В этом случае пути из *a* в *b* нет.

1. Цикл( $i = \overline{1,n}$ )
   1.1. $M[x_i] = -1$ // отметим все вершины, как не пройденные
2. Конец цикла
3. *M[a]=0; //* вершина пройдена
4. *i=1*
5. $a \cup FW_{i-1}$
6. Пока ( $b \notin FW_{i-1}$ И $\exists x \in X, M[x] = -1$)
   6.1. Цикл (для вершин $v \in X, M[v] = -1$ и смежных вершинам $\notin FW_{i-1}$)
      6.1.1. $v \cup FW_i$
   6.2. Конец цикла
   6.3. $i = i + 1$
7. Конец цикла
8. Если $b \in FW_{i-1}$ То «Путь найден», длина пути равна *i-1.*
   Иначе «Путь не существует»
9. Конец

**Задание к работе**

1. Получите индивидуальный вариант задания.
2. Разработайте алгоритм решения задания.
3. Напишите и протестируйте программу.
4. Подготовьте отчет по лабораторной работе.

**Лабораторная работа №22. *Описание математических объектов с помощью классов*. Лабораторная работа №23. *Конструкторы и деструкторы.* Лабораторная работа №24. *Наследование.* Лабораторная работа №25. *Полиморфизм. Л*абораторная работа №26. *Интерфейсы***

**Цель лабораторных работ:** получение навыков объектно-ориентированного программирования.

Структура простой программы

Язык *java* объектно-ориентированный, платформенно-независимый язык программирования. Программа на языке *java* состоит из классов, определенным образом взаимодействующих друг с другом. Один из классов обязательно должен содержать метод *main(),* который автоматически запускается интерпретатором *java.*

Простая программа на языке *java* может выглядеть следующим образом:

```
public class First{
 public static void main(String [] a){
 System.out.println("Первая программа на java");
}
```

Класс *First* содержит метод *main(),* который обязательно должен быть статическим и иметь аргументом массив строк. Вывод строки осуществляет метод *println* свойства *out* класса *System.* Класс *System* включается в пакет автоматически.

Для успешной компиляции класс *First* должен быть сохранен в файле *First.java.*

Компиляция и выполнение из командной строки

Любое приложение на *java* может быть скомпилировано из командной строки. Для этого переменная *PATH* должна быть проинициализирована строкой, указывающей месторасположение файлов *javac.exe* (компилятор) и *java.exe* (интерпретатор). Например,

*SET PATH = C:\Program Files\Java\jdk1.7.0_13\bin*

В различных версиях это расположение может быть различным.

Так же, необходимо задать корневой каталог для иерархии пакетов *java* с помощью переменной среды окружения *CLASSPATH:*

*SET CLASSPATH = .; C:\Program Files\Java\jdk1.7.0_13\src.zip*

Дополнительное значение - '.' переменной среды окружения задано для использования текущей директории в качестве рабочей.

Вызов строчного компилятора выглядит следующим образом:

*javac First.java*

При успешной компиляции создается виртуальный код, записывающийся в файл *First.class.*

Выполнить этот код можно с помощью интерпретатора, набрав в командной строке:

*java First*

Используйте *bat* – файлы для выполнения этого задания.

Компиляция и выполнение в среде eclipse

В помощь разработчикам приложений *java* существует множество интегрированных сред разработки. *Eclipse* - это расширяемая среда разработки с открытым исходным кодом.

При первом запуске загрузчика *Eclipse* перед появлением самой среды выполняется создание директории *workspase* для хранения файлов проектов.

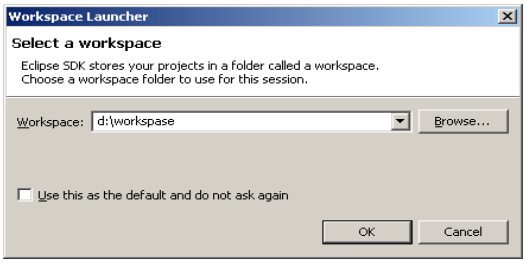



Рис.1. Создание директории *workspase* для хранения файлов проектов

Чтобы начать создание первого *Java*-проекта, выберите ***File->New->Project...*** В левом списке появившегося мастера выберите ***Java Project***. Затем нажимите кнопку ***Next.***

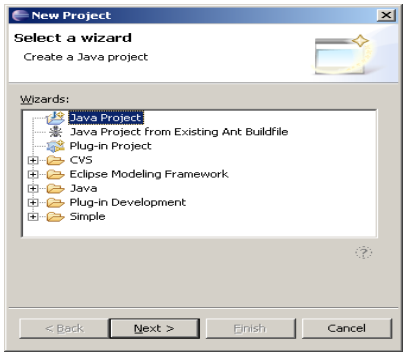



Рис. 2. Первая страница мастера нового проекта

Назовем проект *MyFirstProject.*

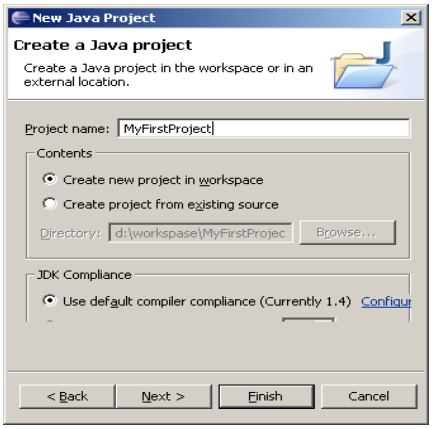
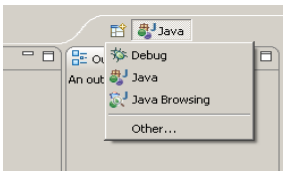



Рис. 3. Вторая страница мастера нового проекта

В этом же окне можно установить версию компилятора (по умолчанию 1.4) и место хранения откомпилированных файлов. После выполнения этих действий выберите кнопку *Finish*, после чего *Eclipse* создаст *java*-проект.

Новый внешний вид носит название Перспективы *Java*. Перспектива, в терминах *Eclipse*, это сохраняемый внешний вид окна, включающий любое число редакторов (editors) и представлений (views). В поставку *Eclipse* входит несколько перспектив по умолчанию (*Java*, *Debug*, *Resource* и так далее), которые можно настраивать. Перспективы управляются с помощью элементов меню ***Window*** или панели инструментов, которая обычно располагается вдоль левой границы окна *Eclipse*.




После создания проекта необходимо создать папки для хранения написанных классов. Эти папки носят название пакетов (*package*). Для создания пакета выберите *File->New->Package* или воспользуйтесь значком панели инструментов  Название пакета может быть произвольным.

После создания пакета необходимо создать класс, используя панель управления или меню *File->New->Class.* При создании класса можно установить атрибут доступа, наличие или отсутствие метода *main()* в классе. После выполнения этих действий будет создан файл <имя класса>*.java.*

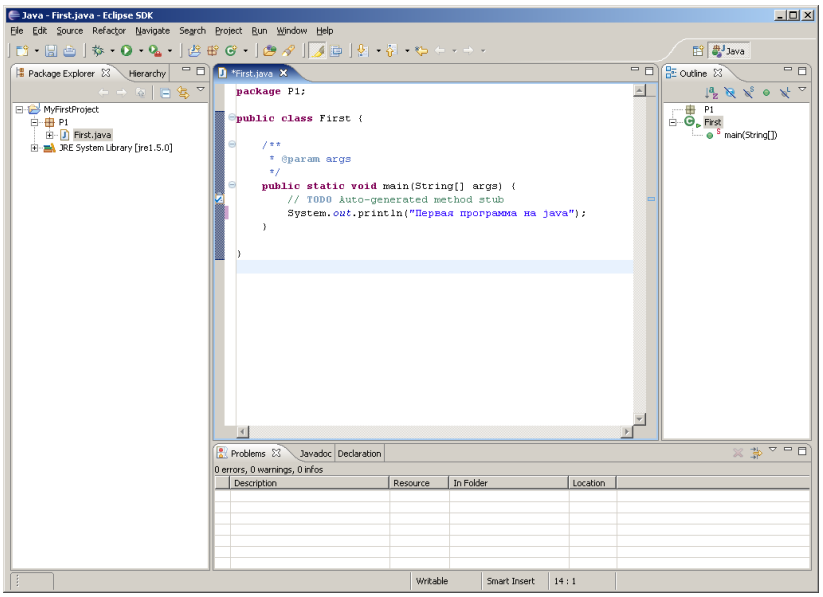



Рис. 4. Вид среды для перспективы *java*

Для перспективы *java* характерно следующее расположение окон:

- в левой части экрана открыт навигатор пакетов
- в средней части экрана открытые окна с текстами классов
- в правой части экрана окно, содержащее структуру активного на данный момент класса
- в нижней части экрана расположены окна сообщений об ошибках и консоль.

Для отладки и запуска программы  выберите значок или меню *Run->Run As->Java Application.*

При успешной компиляции на консоли, в нижней строке экрана появится выводимая строка.

Аргументы командной строки

Аргументы командной строки передаются программе при ее выполнении. Например, рассмотрим программу:

```
public class Two
{
 public static void main(String [] a){
// Обратите внимание на свойство length – длина массива a.
  for(int i=0;i<a.length;i++)
   System.out.println("аргумент "+i+ " " + a[i]);
}
}
```

При выполнении данной программы в командной строке

*java Two programming language java*

на консоль выведутся строки:

аргумент 0 *programming*

аргумент 1 *language*

аргумент 2 *java*

Типы данных

В *java* определены следующие базовые типы данных:

| Тип | Размер (бит) | По умолчанию | Значения (диапазон или максимум) |
|---|---|---|---|
| *boolean* | 8 | *false* | *true, false* |
| *byte* | 8 | 0 | -128…127 |
| *char* | 16 | ‘*u*0000’ | 0…65535 |
| *short* | 16 | 0 | -32768…32767 |
| *int* | 32 | 0 | -2147483648…2147483647 |
| *long* | 64 | 0 | 9223720368547758*07L* |
| *float* | 32 | 0.0*f* | 3.40282347E+38 |
| *double* | 64 | 0.0 | 1.797693134486231570E+308 |

Помимо базовых типов широко используются классы-оболочки *Boolean, Character, Integer, Byte, Short, Long, Float, Double.* Эти классы находятся в библиотеке *java.lang* и являются наследниками класса *Number.* Объекты классов могут быть преобразованы к любому базовому типу методами, описанными в классе.

Строка в *java* представляет объект класса *String.* При работе со строками можно использовать перегружаемую операцию «+» - объединение строк и методы класса *String.* Строковые константы заключаются в двойные кавычки.

Управляющие конструкции

**Проверка условий** в *java*-программе осуществляется посредством конструкции
*if (*<условие>*)* <действия при истинности условного выражения>
*[else* <действия при ложности условия>*]*
Конструкция множественного выбора
*switch (*<переменная, по которой происходит выбор>*){*
*case* <возможное значение переменной выбора 1>*:* <действия>*; break;*
*case* <возможное значение переменной выбора 2>*:* <действия>*; break;*
*…*
*default:* <действия, если переменная выбора не совпала ни с одним из возможных *case*-вариантов>
*}*

**Циклы** в *java* могут быть организованы тремя способами
*for(i=0;i<n;i++)*
*{<тело цикла>}*

Переменная *i* принимает значения от 0 до *n-1(*включительно*)* с шагом 1, соответственно тело цикла выполняется *n* раз.

В следующих двух циклах тело цикла выполняется до тех пор, пока истинно условное выражение.

*while (*<условное выражение>*)*
*{*
<тело цикла>
*}*
*do*
*{*
<тело цикла>
*}while(*<условное выражение>*)*

Если тело цикла содержит одно действие, фигурные скобки могут быть опущены.

Ввод и вывод информации на консоль

Вывод информации на консоль осуществляется с помощью метода *println()* или *print()* свойства *out* класса *System*. Параметрами методов могут быть строки, данные базовых типов и объекты классов-оболочек. При сложном выводе элементы соединяются перегруженным оператором «+».

```
…
Integer I = new Integer(25);
String S = “String”;
float f = (float) 23.1;
System.out.println(S+” I = “+I+” f=”+f );
…
```

На экран выведется строка

*String I* = 25 *f=23.1*

Ввод информации в *java* выполняется с помощью объектов классов *InputStreamReader* и *BufferedReader*.

На первом шаге объявляется объект класса *InputStreamReader*, который инициализируется как стандартный объект ввода *System.in* – клавиатура.

```
InputStreamReader is = new InputStreamReader(System.in);
```

На втором шаге создается объект буферизованного ввода

```
BufferedReader bis = new BufferedReader(is);
```

Ввод информации осуществляется в строковую переменную с помощью метода *readLine()*. Для перевода строкового типа в числовой тип используются методы соответствующих классов-оболочек *intValue(), floatValue()* и т.д.. Ввод информации обязательно сопровождается проверкой исключительной ситуации типа *IOException* – ошибка ввода-вывода.

Следующий фрагмент программы вводит с клавиатуры целое число *n*, размер массива строк, выделяет память под массив строк и вводит значения элементов массива строк.

```
InputStreamReader is = new InputStreamReader(System.in);
BufferedReader bis = new BufferedReader(is);
try{
     System.out.print("Enter n:");
     String nS = bis.readLine();
     n = Integer.valueOf(nS).intValue();
   } catch (IOException e){
                              System.out.print("Error"+e);
                           }
String My[]=new String[n];
for(int i=0;i<n;i++)
 {
  System.out.print("Enter a String: ");
  try{
      My[i] = bis.readLine();
    } catch (IOException e){
                              System.out.print("Error"+e);
                            }
}
```

Классы *InputStreamReader* и *BufferedReader* описаны в библиотеке *java.io.*

Для подключения библиотеки служит ключевое слово *import* <имя библиотеки>. Импорт классов из библиотеки выполняется перед описанием класса. Символ «*» означает, что из указанной библиотеки импортируются все имеющиеся в ней пакеты.

Например:
*import java.io.*;*

Апплеты

Апплет – небольшая программа, запускаемая *Web*-браузером. Апплеты, написанные пользователем являются наследниками (расширениями) класса *Applet* библиотеки *java.awt.** или класса *JApplet* библиотеки *javax.swing.**.

Для указания наследования в *java* используется ключевое слово *extends*.

Для запуска апплету не нужен метод *main()*. Код запуска апплета помещается в метод *paint()* или *init()*.

Метод *paint()* получает в качестве параметра объект класса *Graphics*. Вывод информации осуществляется методом *drawstring()* этого класса.

В классе *Graphics* определены методы:

*clearRect(int x, int y, int width, int height)* – очищает указанную прямоугольную область.

*copyArea(int x, int y, int width, int height, int dx, int dy)* - копирует указанную прямоугольную область в область с левым верхним улом *x+dx,y+dy*.

*draw3DRect(int x, int y, int width, int height, boolean raised)* - рисует 3D-прямоугольник указанного размера.

*drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)* – рисует круговую или эллиптическую дугу.

*drawLine(int x1, int y1, int x2, int y2)* – рисует линию по указанным координатам.

*drawOval(int x, int y, int width, int height)* - рисует эллипс.

*drawPolygon(Polygon p)* и *drawPolygon(int[] xPoints, int[] yPoints, int nPoints)* - рисуют замкнутую кривую по заданным точкам.

*drawPolyline(int[] xPoints, int[] yPoints, int nPoints)* - рисует произвольную кривую линию по заданным точкам.

*drawRect(int x, int y, int width, int height)* - рисует прямоугольник.

*drawString(String str, int x, int y)* - выводит в графическом окне заданную строку.

*fill3DRect(int x, int y, int width, int height, boolean raised)* - рисует закрашенный объемный прямоугольник.

*fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)* - рисует закрашенный круговой или эллиптический сектор.

*fillOval(int x, int y, int width, int height)* - рисует закрашенный эллипс.

*fillPolygon(int[] xPoints, int[] yPoints, int nPoints)* - рисует закрашенный многоугольник.

*fillRect(int x, int y, int width, int height)* - рисует закрашенный прямоугольник.

*getColor()* - возвращает текущий цвет рисования.

*getFont()* -возвращает текущий фонт.

*getFontMetrics()*- возвращает характеристики текущего фонта.

*setColor(Color c)* - устанавливает цвет рисования.

*setFont(Font font)* - устанавливает фонт.

А так же и другие методы для работы с клипами, изображениями и т.д..

Для работы с цветами в *java/awt* определен класс *Color*, полями которого являются цвета.

Для запуска апплета из среды *Eclipse* используйте меню *Run->Run As->Java Applet*.

Для того, чтобы запустить апплет с помощью браузера необходимо написать *html* – документ следующего вида:

*<html><body>*

*<applet code = <имя апплета>>*
*</applet>*
*</body></html>*

Если апплет находится в пакете, то к имени файла справа добавляется через / имя пакета. *html* документ при этом должен находиться в папке проекта. Например, в среде *Eclipse* создан проект *Progect*, в нем создан пакет *p1*, а в пакете создан апплет с именем *app.java*, скомпилированный в файл *app.class*, то имя апплета формируется следующим образом *p1/app.class*, сам *html*-документ должен находиться в папке *Progect.*

Если апплет находится не в пакете, то в имени апплета указывается только имя файла с расширением *class*, *html*-документ располагается в одной папке с этим файлом.

Пример java-приложения

Написать следующее консольное приложение: ввести с консоли размерность массива строк *n*. Ввести элементы массива с консоли. Найти количество элементов массива, длина которых не меньше заданной. Значение заданной длины ввести с консоли.

```
import java.io.*;
public class my1 {
public static void main(String [] args){
int n=0;
int len = 0;

InputStreamReader is = new InputStreamReader(System.in);
BufferedReader bis = new BufferedReader(is);
try{
System.out.print("Enter n:");
String nS = bis.readLine(); // чтение строки
n = Integer.valueOf(nS).intValue();// преобразование строки
в число
} catch (IOException e){
System.out.print("Error"+e);}
String My[]=new String[n]; // выделение памяти под массив
строк
for(int i=0;i<n;i++){
System.out.print("Enter a String " + i + ": ");
try{
My[i] = bis.readLine(); // чтение i-той строки с клавиатуры
} catch (IOException e){
System.out.print("Error"+e);
}}
for(int i=0;i<n;i++)
{System.out.println(My[i]); // печать полученного массива
строк
}
try{
System.out.print("Enter length:");
String nS = bis.readLine(); // чтение заданной длины
len = Integer.valueOf(nS).intValue(); // преобразование стро-
ки в число
} catch (IOException e){
System.out.print("Error"+e);}
int Sum = 0;
```

```
for (int i=0;i<n;i++){
if (My[i].length()>len) Sum++; // поиск количества слов,
длина которых
// больше заданной
}
// вывод результата
System.out.println(Sum + " string`s have length more then " +
len);}}
```

Пример апплета

Написать апплет с выводом в окно элементов массива. Размерность массива задается константой, элементы массива случайным образом. Выделите цветом не повторяющиеся элементы.

```
import java.awt.*;
import javax.swing.*;
public class App1 extends JApplet{
int mass[];
public void init(){
this.setBackground(Color.CYAN); // цвет фона
this.setForeground(Color.YELLOW);// цвет символов
mass = new int [10];
for(int i=0;i<mass.length;i++){
mass[i] = (int)(Math.random()*20);          }}
public void paint(Graphics G){
// вспомогательный массив меток
boolean metka []= new boolean [mass.length];
// обнулим все метки
for (int i=0;i<mass.length;i++)
 metka[i] = false;
for(int i=0;i<mass.length-1;i++)
for(int j=i+1;j<mass.length;j++)
// если i-тый  и  j-тый элементы равны
if (mass[i]==mass[j]){
// то изменить метки  этих элементов
                                                    metka[i]= true;
 metka[j] = true;
metka[i] = true;}
//  в зависимости от значений метки определять цвет вывода элемента
for(int i=0;i<mass.length;i++)
{if (metka[i]) G.setColor(Color.RED);
else G.setColor(Color.GREEN);
G.drawString(new Integer(mass[i]).toString(),i*20,40);}}}}
```

Абстрактные классы

Абстрактные классы содержат объявления абстрактных методов, которые не реализованы в этих классах, а будут реализованы в подклассах. Объекты абстрактных классов создать нельзя. Но можно создавать объекты подклассов, реализующих абстрактные методы. Синтаксис объявления абстрактного класса:

```
abstract class <имя класса>{
// абстрактный метод
```

*abstract* <тип возвращаемого значения><имя класса>*(*<параметры>*);*
*//* обыкновенный метод
*void Met(){*реализация*}*
*}*

Интерфейсы

Интерфейсы представляют собой полностью абстрактные классы, то есть ни один из объявленных методов не может быть реализован внутри интерфейса. Все поля интерфейса автоматически получают атрибуты доступа и спецификаторы *public, static, final,* а все методы как *public* и *abstract.*

Говорят, что класс реализует интерфейс, переопределяя его методы. При этом класс может реализовывать несколько интерфейсов. Если класс переопределяет не все методы интерфейса, он должен быть объявлен как абстрактный.

Синтаксис определения интерфейса:
*[public] interface* <имя> *[extends I1, I2, …, IN]*
*{* реализация интерфейса*}*

По приведенному синтаксису видно, что интерфейс может быть наследником одного или нескольких интерфейсов (для интерфейсов возможно множественное наследование).

Синтаксис реализации интерфейсов классами:
[доступ] *class* <имя класса> *implements I1, I2, …, IN*
*{* код класса
*}*

При этом *I1, I2, …, IN* - перечисление имен реализуемых классом интерфейсов.

Внутренние классы

Нестатические вложенные классы принято называть внутренними (*inner*). Из внешнего класса доступ к элементам внутреннего класса возможен только через объект внутреннего класса. Этот объект должен быть описан в коде внешнего класса. Методы внутреннего класса имеют прямой доступ к полям и методам внешнего класса.

При проектировании необходимо помнить, что во внутреннем классе нельзя определять статические поля и методы.

Внутренние классы могут быть наследниками и потомками других классов, реализовывать интерфейсы.

Наследование может быть организовано двумя способами:

а)
*class A{*
*//* поля и методы
*[*доступ*] class B [extends …][implements…]{*
*//* поля и методы
*}}*

*class A1 extends A{*
*class B1 extends B{}*
*//* инициализация объекта класса *B1*
*B obj = new B1()}*

б)
*class B2 extends A.B{*

/* при таком способе наследования класс *B2* не имеет доступа к полям внешнего класса *A*, поэтому в классе *B2* необходимо объявить конструктор с параметром-объектом внешнего класса, после этого, класс *B2* получит ссылку на класс *B* */

```
B2 (A obj){
  obj.super();
}}
```

Продемонстрируем пример объявления ссылок на объекты внутренних классов.

```
class A {
// защищенный внутренний класс
   protected class B {
      private int x;
      void putx(int x){
      this.x = x;}
      void show (){
      System.out.println("x = "+x);}}
// метод внешнего класса, возвращающий объект внутреннего класса
  B get(){
   return new B;
  }
}
class Demo{
public static void main(String [] a){
// создать объект внешнего класса
A obj = new A();
// создать объект внутреннего класса (обратите внимание на синтаксис)
A.B obj1 = obj.get();
// задать значение поля x
obj1.putx(17);
//  вывести на консоль значение поля x
obj1.show();}}
```

Вложенные классы

Если при описании внутреннего класса используется ключевое слово *static*, такой класс называют вложенным (*nested*) классом. Такой класс может обращаться к нестатическим полям и методам класса только через объект внешнего класса. К статическим полям и методам внешнего класса вложенный класс может обращаться напрямую.

```
class A{
  int x,y;
  static float f;
  void metod1(){x++;}
  void putxy(int x, int y){
  this.x = x; this.y = y;}
  static void show(){System.out.println("AB");}
  static class B{
  static void metod(){
  //обращение к нестатическим полям и методам класса A
  A obj = new A();
  obj.x =obj.x+2;
  obj.y = obj.y+1;
```

```
  obj.metod1();
  System.out.println("x = "+obj.x+"y = "+obj.y);}
  //обращение к статическим полям класса A
  void metod2(){
  f = 45;
  show();
  System.out.println(f);}}
//вызов нестатического метода вложенного класса
void metod3(){
new B().metod2();}

public static void main(String [] a){
A obj = new A();
obj.putxy(12,15);
obj.metod3();
// вызов статического метода вложенного класса
A.B.metod();}}
```

На консоль выведется:

AB
45.0
*x* = 3 *y* = 1

Абстрактный класс

Рассмотрим реализацию следующей заадчи: создать абстрактный класс **Учащийся** и его подклассы **Школьник** и **Студент.** Создать массив объектов абстрактного класса. Показать отдельно студентов и школьников.

```
// реализация абстрактного класса
import java.util.*;
import javax.swing.*;
abstract public class Learn {
String name;// имя
Date bday; // дата рождения
String scool; // название учебного учереждения
// абстрактный метод редактирования полей обекта
abstract void PutInfo(); }
```

В абстрактный класс включены общие для классов **Школьник** и **Студент** поля и методы.

```
// реализация класса Студент
import javax.swing.*;
import java.util.*;
import java.awt.*;
public class Student extends Learn{
String group; // номер группы
// конструктор
Student(String name, Date bday,String scool, String _class){
   this.name = name;
   this.bday = bday;
   this.scool = scool;
   this.group = _class;}
// конструктор без параметров
Student(){
```

```
    this.name  = "";
    this.bday  = new Date();
    this.scool = "";
    this.group = "";}
```

/* редактирование полей объекта – метод использует объекты класса *JOptionPane* и его статический метод *ShowInputDialog*, который вводит текстовую информацию во всплывающем окне, параметр метода – строка с названием вводимого поля */

```
void PutInfo(){
name = JOptionPane.showInputDialog("Name: ");
String day = JOptionPane.showInputDialog("Birth day: ");
bday = new Date(day);
scool = JOptionPane.showInputDialog("VUZ: ");
group = JOptionPane.showInputDialog("Group: ");}}
```

// реализация класса **Школьник**

```
import javax.swing.*;
import java.util.*;
import java.awt.*;
public class Scool extends Learn{
String _class; // класс
// конструктор
Scool(String name, Date bday,String scool, String _class){
    this.name = name;
    this.bday = bday;
    this.scool = scool;
    this._class = _class;}
// конструктор без параметров
Scool(){
this.name  = "";
this.bday  = new Date();
this.scool = "";
this._class = "";}
/* редактирование полей объекта с помощью объектов класса JOptionPane*/
void PutInfo(){
    name = JOptionPane.showInputDialog("Name: ");
    String day = JOptionPane.showInputDialog("Birth day: ");
    bday = new Date(day);
    scool = JOptionPane.showInputDialog("Scool: ");
    _class = JOptionPane.showInputDialog("Class: ");}}
//  управляющий класс
import javax.swing.*;
import java.awt.*;
import java.util.*;
public class MyFrame extends JFrame{
Object [][] cells;
```

/* конструкор, параметрами которого являются данные об объектах (*Object [][]c*), строка с названиями столбцов *(String []cN)*, строка заголовка (*String T*) */

```
MyFrame(Object [][]c, String []cN, String T){
setSize(new Dimension(500,400)); // установить размер окна
setTitle(T); // установить заголовок
setBackground(Color.WHITE); // установить цвет фона
setForeground(Color.BLUE); // установить цвет символов
int p = c.length/4;
```

```
 cells =new Object[c.length][4];
//  переписать данные об объекте в поле cells
for(int i=0;i<cells.length;i++)
{cells[i][0]=c[i][0];
cells[i][1]=((Date)c[i][1]).getDay()+"."
+((Date)c[i][1]).getMonth()+"."+
 ((Date)c[i][1]).getYear();
cells[i][2]=c[i][2];
cells[i][3]=c[i][3];}
JTable table = new JTable(cells,cN);//  создать таблицу
getContentPane().add(new JScrollPane(table), // добавить ее во фрейм
BorderLayout.CENTER); //  расположить ее по центру
} public static void main(String[] args) {
// TODO Auto-generated method stub
Learn [] S = new Learn[4]; // создать массив объектов Учащийся
/*создать диалоговое окно со списком, для этого определить возможные варианты выбора в переменной possibleValues */
Object [] possibleValues = {
"student",
"scool"};
int k = 0, k1 = 0;
/* для каждого вводимого объекта с помощью списка определить, к какому классу относится объект, для этого создается диалоговое окно с указанными параметрами: */
for (int i=0;i<S.length;i++)
{ Object selectedValue =
JOptionPane.showInputDialog(
null, // родительский объект
"Choice type: ", // текстовая метка в окне диалога
"Choice", // заголовок
JOptionPane.INFORMATION_MESSAGE, // тип сообщения
null, // иконка в окне диалога (null – по умолчанию)
possibleValues, // значения элементов списка
possibleValues[0]); // начальное значение
// если выбран объект Школьник
  if ((String)selectedValue == "scool")
  {S[i] = new Scool();// выделить память
  S[i].PutInfo(); // заполнить данные
  k++;}
else { // в противном случае создается и заполняется объект Студент
S[i] = new Student();
  S[i].PutInfo();
  k1++;}}
/* переписать данные в массивы cells и cells1
Object [][] cells = new Object[k][4];
Object [][] cells1 = new Object[k1][4];
for(int i=0,j=0,z=0;i<S.length;i++){
if (S[i] instanceof Scool){
cells[j][0] = S[i].name;
cells[j][1] = S[i].bday;
cells[j][2] = S[i].scool;
cells[j][3] = ((Scool)S[i])._class;
j++;}
```

```
else {
cells1[z][0] = S[i].name;
cells1[z][1] = S[i].bday;
cells1[z][2] = S[i].scool;
cells1[z][3] = ((Student)S[i]).group;
z++;}}
// заголовки столбцов таблицы для Школьников
String _scool [] = {
"Имя",
"Дата рождения",
"Школа",
"Класс"
};
// заголовки столбцов таблицы для Студентов
String _student [] = {
"Имя",
"Дата рождения",
"ВУЗ",
"Группа"};
// Создать фрейм, содержащий таблицу со Школьниками
MyFrame a = new MyFrame(cells,_scool,"Школьники");
a.setVisible(true);
// Создать фрейм, содержащий таблицу со Студентами
MyFrame b = new MyFrame(cells1,_scool,"Студенты");
b.setVisible(true);}}
```

Интерфейс

Создать интерфейс **Фигура** и его реализации для классов ***Окружность*** и **Квадрат.** Предусмотрим в интерфейсе методы вычисления площади фигуры и методы рисования.

```
import java.awt.*; // интерфейс
public interface Shape {
double pi = Math.PI;
public double Square();
public void Show (Graphics g);}

// класс окружность – реализация интерфейса
public class Circle implements Shape{
 int x,y,rad;
 Circle(int x, int y, int rad){ // конструктор
      this.x=x;
     this.y=y;
      this.rad=rad; }
 public double Square(){ // вычисление площади круга
      return 2*pi*rad*rad; }
 public void Show (Graphics g){//  рисование окружности
      g.drawOval(x,y,rad,rad); }}
// класс квадрат – реализация интерфейса
public class Quadro implements Shape{
int x,y,rad;
Quadro(int x, int y, int rad){ // конструктор
```

```
        this.x=x;
        this.y=y;
        this.rad=rad;}
 public double Square(){ // вычисление площади
        return 4*rad*rad; }
 public void Show (Graphics g){ // рисование
        g.drawRect(x,y,rad,rad); }}

import javax.swing.*; //  класс для работы с фреймом
import java.awt.*;
public class FrameI extends JFrame{
 Shape c1,c2; // описание ссылок на интерфейс
 FrameI(){ // конструктор
       this.setSize(new Dimension(500,500)); // установить размер,
       this.setBackground(Color.gray);          // цвет фона
       this.setForeground(Color.YELLOW);  // цвет рисования,
       this.setTitle("Shape");                    // заголовок
      c1 = new Circle(50,50,25);                // создание окружности
      c2 = new Quadro(50,140,25); }          // создание квадрата
 public void paint(Graphics g){
       double sq = c1.Square(); // расчет площади окружности
       double sq1 = c2.Square();// расчет площади квадрата
       c1.Show(g); // нарисовать окружность
       c2.Show(g);// нарисовать квадрат
      g.drawString("Square " + sq,25,100);     //  вывести значение
      g.drawString("Square " + sq1,25,195); }// площадей
 public static void main(String [] a){
       FrameI Fr = new FrameI(); // создать фрейм
       Fr.setVisible(true); // сделать его видимым
      Fr.repaint(); // вызвать метод рисования
 }}
```

Внутренний класс

Создать класс Записная книжка, с внутренним классом или классами, с помощью объектов которого могут храниться несколько записей на одну дату.

```
package p4;
import java.util.*;
import javax.swing.*;
public class Notepad {
int m;  // количество записей по датам
int n;  // количество записей с телефонами
Note [] N; // массив записей по датам
Telephone [] T; // массив записей с телефонами
// реализация класса «Запись по датам»
class Note {
Date d; //  дата
String [] note; // массив записей
int m; // количество записей
Note(Date d){ // конструктор
this.d = d;
```

```
note = new String [20];
m = 0;}
// создать новую запись
void SetNote(String str){
note[m]=str;
m++;}
Object []GetInfo(){// вернуть запись
Object [] c = new Object[m+1];
c[0]=d.getDay()+"."
      +(d.getMonth()+1)+"."+
       (d.getYear()+1900);
for(int i=0;i<m;i++)
 {c[i+1] = note[i];} return c;} }
// реализация класса «Запись телефона»
class Telephone {
String name;
String telephone;
void SetTelephone(String n, String t){ // создание новой записи
if (n!="") name = n;
if (t!="") telephone = t;}
Object [] GetInfo(){ // вернуть запись
Object [] c = new Object[2];
c[0] = name;
c[1] = telephone;
return c;}}
 Notepad(){ // конструктор
 m = 0; // количество записей обнуляется
 n = 0;
N = new Note [30];
T = new Telephone[30]; }
// создать новую запись по дате
 void SetNote(){
 boolean flag = false;
 String day = JOptionPane.showInputDialog("Date: ");
Date d = new Date(day);
 String n = JOptionPane.showInputDialog("Note: ");
 if (m==0){ N[m]=new Note(d); // если записная книжка пуста
              N[m].SetNote(n);
               m++; }
 else // сравнение веденной даты с уже имеющимися
 for(int i=0;i<m;i++)
 {if (d.equals(N[i].d)) {flag = true;
                         N[i].SetNote(n);
                         break;}
 else {
 if (!flag) {
 N[m]=new Note(d);
 N[m].SetNote(n);
 m++;
 break; }} } }
// ввод новой записи с телефоном
 void SetTelephone(){
```

```
boolean flag = false;
String name = JOptionPane.showInputDialog("Name: ");
 String t = JOptionPane.showInputDialog("Telephone: ");
 for(int i=0;i<n;i++){ // сравнение телефона с уже имеющимися
if (name.equals(((Telephone)T[i]).name)&&
!T.equals(((Telephone)T[i]).telephone)){
 int result = JOptionPane.showConfirmDialog(
 null,
 new String("Edit "+name),
 "Attention!",
 JOptionPane.YES_NO_OPTION);
 if (result==0) {T[i].SetTelephone("",t); flag = true; break;} }
 if (!name.equals(((Telephone)T[i]).name)&& // сравнение фамилии с
// уже имеющимися
T.equals(((Telephone)T[i]).telephone))
 {int result = JOptionPane.showConfirmDialog(
 null,
 new String("Edit "+t),
 "Attention!",
 JOptionPane.YES_NO_OPTION);
 if (result==0) {T[i].SetTelephone(name,""); flag = true; break;}}
 if (name.equals(((Telephone)T[i]).name)&&
T.equals(((Telephone)T[i]).telephone))
{int result = JOptionPane.showConfirmDialog(
 null,
 new String("Edit "+name + " "+t+ " ?"),
 "Attention!",
 JOptionPane.YES_NO_OPTION);
 if (result==0) {T[i].SetTelephone(name,t); flag = true; break;}} }
 if (!flag){T[n] = new Telephone();
  T[n].SetTelephone(name,t);
  n++;} }
Object [][] GetInfoT(){// вернуть информацию о телефонах
 Object [][] c = new Object[n][2];
for (int i=0;i<n;i++)
 {c [i]= T[i].GetInfo(); }
 return c; }
 Object [][] GetInfoN(){ // вернуть информацию о записях
Object [][] c = new Object[this.GetM()][2];
 int k = 0;
for (int i=0;i<m;i++)
 {   Object [] c1 = new Object[N[i].m+1];
 c1 = N[i].GetInfo();
for(int j=k,j1 = 1;j1<c1.length;j++,k++,j1++){
 c[j][0]=c1[0];
 c[j][1]=c1[j1];}}
 return c; }
 int GetN(){
return n; }
 int GetM(){
 int z = 0;
for(int i=0;i<m;i++){
```

```
 z+=N[i].m; }
 return z; }}

// исполняемый класс
package p4;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.util.Date;
import java.awt.event.*;
import javax.swing.*;
public class FrameN extends JFrame implements ActionListener{
JMenu menN;JMenu menT;JMenuItem TE;
JMenuItem NE;JMenuItem TS;JMenuItem NS;
JMenuItem E;
Notepad notepad;
FrameN(Object [][]c, String []cN, String T){// конструктор,
// использующийся для создания фрейма с таблицами
setSize(new Dimension(500,400));
setTitle(T);
setBackground(Color.WHITE);
setForeground(Color.BLUE);
JTable table = new JTable(c,cN);
getContentPane().add(new JScrollPane(table),
 BorderLayout.CENTER);}
FrameN(){// конструктор, использующийся для создания фрейма
// с меню
notepad = new Notepad();
setSize(new Dimension(500,400));
setTitle("Notebook");
setBackground(Color.WHITE);
setForeground(Color.BLUE);}
public void init(){ // инициализация
menT = new JMenu("Telephone"); // создание двух пунктов меню
menN = new JMenu("Note");
TE = new JMenuItem("New");// создание пунктов подменю
NE = new JMenuItem("New");
TS = new JMenuItem("Show");
NS = new JMenuItem("Show");
E = new JMenuItem("Exit");
JMenuBar menubar = new JMenuBar();
this.setJMenuBar(menubar);// создание поля для меню на фрейме
TE.addActionListener(this);// добавление пунктов подменю
NE.addActionListener(this);// в блок считывания событий
TS.addActionListener(this);
NS.addActionListener(this);
E.addActionListener(this);
menT.add(TE);menT.add(TS);// добавление пунктов подменю в
menT.add(E);menN.add(NE); // меню
menN.add(NS);
menubar.add(menT);
menubar.add(menN);}
```

```
public void actionPerformed(ActionEvent e){ //выбор действий
// определение заголовков таблиц
String [] _n = {"Date",
"Note"};
String [] _t = {"Name",
"Telephone"};
JMenuItem jm = (JMenuItem)e.getSource();
if (jm.equals(TE) ) notepad.SetTelephone();// ввод новой записи с
// телефоном
else if (jm.equals(NE) )notepad.SetNote();// ввод новой записи по дате
else if (jm.equals(TS)){
int n = notepad.GetN();// формирование и вывод таблицы с записями по
Object [][] cells1 = new Object[n][2]; // дате
cells1 = notepad.GetInfoT();
FrameN b = new FrameN(cells1,_t,"TB");
b.setVisible(true);
} else if (jm.equals(NS)){ // формирование и вывод таблицы с записями
int m = notepad.GetM();// телефонов
Object [][] cells = new Object[m][2];
cells = notepad.GetInfoN();
FrameN a = new FrameN(cells,_n,"NB");
a.setVisible(true);}
else if (jm.equals(E)) System.exit(0); // выход
}
public static void main(String[] args) {
FrameN f_menu = new FrameN(); // инициализация фрейма с меню
f_menu.init();
f_menu.setVisible(true);}}
```

Вложенный класс

Опишем класс «Лучи, с началом в одной точке координат» и вложенный класс «Линия». Объект класса Линия в этом случае, можно создавать напрямую, не используя объект внешнего класса.

```
package p44;
import java.awt.*;
public class Luch { // Класс «Лучи»
  static int x,y; // координаты начала лучей
  int n; // количество лучей
  int [][] xy; // координаты концов лучей
  Line [] l;  // массив объектов вложенного класса «Линия»
 Luch(int n1, int k, int p){ // конструктор класса
  n = n1;
  x = k;
  y = p;
  xy = new int [n][2];
  for (int i=0;i<n;i++) // случайное задание концов лучей
   for(int j=0;j<2;j++)
        xy[i][j] = (int)(1000*Math.random()/10);
   l = new Line[n];
for(int i=0;i<n;i++) // создание объектов класса «Линия»
l[i]=new Line(x,y,xy[i][0],xy[i][1],Color.blue);}
```

```
void Show(Graphics g){ // рисование
g.drawOval(x,y,5,5);
for(int i=0;i<n;i++)
  l[i].Show(g);  }
   static class Line { // вложенный класс
int px,py,px1,py1; // координаты концов линии
Color color; // цвет линии
Line(int x1,int y1, int x2, int y2, Color c){// конструктор
px = x1;
py = y1;
px1 = x2;
py1 = y2;
color = c;}
void Show(Graphics g){ // рисование
g.setColor(color);
g.drawLine(px,py,px1,py1);}}}
// описание фрейма для рисования лучей
package p44;
import java.awt.*;
import javax.swing.*;
public class FrameLuch extends JFrame{
Luch luch; // объект класса «Лучи»
FrameLuch(){ // конструктор
this.setTitle("Luch");
this.setBackground(Color.gray);
this.setSize(new Dimension(100,200));
luch = new Luch(10,50,100);}
public void paint(Graphics g){
luch.Show(g);}}
// описание фрейма для рисования линии
package p44;
import java.awt.*;
import javax.swing.*;
public class FrameLine extends JFrame{
Luch.Line line;
FrameLine(){
this.setTitle("Line");
this.setBackground(Color.gray);
this.setSize(new Dimension(100,200));
line = new Luch.Line(10,10,80,90,Color.yellow);}
public void paint(Graphics g){
line.Show(g);}}
// исполняемый класс
package p44;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

public class FrameL extends JFrame implements ActionListener{
  JMenu menuLuch; // описание меню и пунктов меню
  JMenu menuLine;
 JMenuItem LuchS;
```

```
    JMenuItem LineS;
    JMenuItem E;
FrameL(){ // конструктор
    setTitle("Nested class");
    setBackground(Color.WHITE);
    setForeground(Color.BLUE);
    setSize(new Dimension(200,200));
    menuLuch = new JMenu("Luch");
    menuLine = new JMenu("Line");
    LuchS = new JMenuItem("Show");
    LineS = new JMenuItem("Show");
    E = new JMenuItem("Exit");
    JMenuBar menubar = new JMenuBar();
    this.setJMenuBar(menubar);
    LuchS.addActionListener(this);
    LineS.addActionListener(this);
    E.addActionListener(this);
    menuLuch.add(LuchS);
    menuLuch.add(E);
    menuLine.add(LineS);
    menubar.add(menuLuch);
    menubar.add(menuLine);   }
    public void actionPerformed(ActionEvent e){// обработка событий
    JMenuItem jm = (JMenuItem)e.getSource();
    if (jm.equals(LuchS)) {
    FrameLuch FL = new FrameLuch(); // выбор рисования лучей
    FL.repaint();
    FL.setVisible(true);
    }else if (jm.equals(LineS)){// выбор рисования линии
    FrameLine fl = new FrameLine();
    fl.repaint();
    fl.setVisible(true);
    }else if (jm.equals(E))System.exit(0);// выход из меню}
    public static void main(String[] args) {
        FrameL Fr = new FrameL();
    Fr.setVisible(true);}}
```

**Практическое занятие №1.** ***Алгоритмы на графах***

План занятия

1. Повторение основных алгоритмов на графах.
2. Решение практических задач на разработку алгоритмов обхода, построение маршрутов на графах.

**Практическое занятие №4.** ***Знакомство с языком Java. Простые программы***

План занятия

3. Повторение синтаксиса языка Java.
4. Правила написания простой программы на Java.
5. Знакомство со средой программирования Eclipse. Ее настройка.
6. Решение практических задач.

**Практическое занятие №6.** ***Создание класса. Конструкторы***

План занятия

1. Определение класса.
2. Определение конструктора.
3. Решение практических задач.

Приложение А
Образец титульного листа к отчету по лабораторной работе

Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования

«ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ
И РАДИОЭЛЕКТРОНИКИ» (ТУСУР)

Кафедра автоматизации обработки информации (АОИ)

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ

Студент гр. 423-1
_____ И.П. Иванов
«___» _____ 20 __ г.

Преподаватель:
ст. преп. каф. АОИ
_____ Н.В. Пермякова
«___» _____ 20 __ г.

2015