

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ  
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ»  
ФИЛИАЛ «МИНСКИЙ РАДИОТЕХНИЧЕСКИЙ КОЛЛЕДЖ»

УТВЕРЖДАЮ  
Директор МРК  
\_\_\_\_\_ С.Н. Анкуда  
«\_\_\_\_\_» \_\_\_\_\_ 20\_\_ г.  
Регистрационный №\_\_\_\_\_

**СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ**

**МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ  
ПО ВЫПОЛНЕНИЮ ЛАБОРАТОРНЫХ РАБОТ**  
для учащихся специальности:  
2-39 03 02 «Программируемые мобильные системы»

Минск 2017 г.

**СОСТАВИТЕЛИ:**

С.А. Апанасевич, преподаватель 1 категории дисциплин специального цикла  
А.В. Яковлев, преподаватель 1 категории дисциплин специального цикла

**РЕКОМЕНДОВАНЫ К УТВЕРЖДЕНИЮ:**

Цикловой комиссией «Программное обеспечение информационных технологий»  
филиала БГУИР «Минский радиотехнический колледж»

Протокол №\_\_\_\_\_ от \_\_\_\_\_

Заседанием педагогического совета филиала БГУИР «Минский  
радиотехнический колледж»

Протокол №\_\_\_\_\_ от \_\_\_\_\_

Методическая экспертиза \_\_\_\_\_  
подпись \_\_\_\_\_ ФИО \_\_\_\_\_

## Содержание

Лабораторная работа № 1 .....	5
Лабораторная работа № 2 .....	19
Лабораторная работа № 3 .....	31
Лабораторная работа № 4 .....	40
Лабораторная работа № 5 .....	54
Лабораторная работа № 6 .....	70
Лабораторная работа № 7 .....	83
Лабораторная работа № 8 .....	96
Лабораторная работа № 9 .....	109
Лабораторная работа № 10 .....	120
Лабораторная работа № 11 .....	140
Лабораторная работа № 12 .....	152
Лабораторная работа № 13 .....	167
Лабораторная работа № 14 .....	183
Лабораторная работа № 15 .....	196
Лабораторная работа № 16 .....	205
Лабораторная работа № 17 .....	221
Лабораторная работа № 18 .....	229
Лабораторная работа № 19 .....	235
Лабораторная работа № 20 .....	256
Лабораторная работа № 21 .....	274
Лабораторная работа № 22 .....	287
Лабораторная работа № 23 .....	295

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Технология разработки программного обеспечения»

**Инструкция**  
по выполнению лабораторной работы  
«Построение функциональной модели программной системы»

Минск  
2017

# Лабораторная работа № 1

**Тема работы: «Построение функциональной модели программной системы»**

## 1. Цель работы

Изучить линейные алгоритмы.

## 2. Задание

Номер варианта соответствует вашему номеру по списку.

№	Функция
1	$y = \frac{5x + \sin x^2}{2x + \operatorname{tg} x} +  \sin x^2 $
2	$y = \frac{\log_3 x-10 }{2x-7} - \cos(3x+5)$
3	$y = \frac{\cos(x_1 + 5) + x_2}{3x_1 + 6x_2} *  \sin x_1^2 $
4	$y = \frac{\cos(3x+5) + 5^x}{4.35x} +  \cos x^2  + \frac{5}{x}$
5	$y = \frac{3x_1 + 2^{x_2}}{\sin(x_1 + x_2)} + \frac{5}{ \sin x_2 } + 6$
6	$y = \frac{\sin^2(x + 3.5) + \lg x}{3x + e^{x+1.35}}$
7	$y = \frac{x_1 + 5.5x_2}{\cos x_1 + \ln x_2} + \operatorname{tg}(x_1 + 4.3)$

<b>№</b>	<b>Функция</b>
8	$y = \frac{3^{x-1.4} + e^x}{4.5 + x} + \operatorname{tg} 3x$
9	$y = \frac{x_1^{x_2} + 3.5 \operatorname{tg} x_1}{3x_1 + 5x_2} + \frac{5x_1}{x_2 + 6} + 5$
10	$y = \frac{\cos^3(x+2.5) + 1.5x}{x^2 + 3.5} + \frac{3x^2}{8x} + 8.5$
11	$y = \frac{\ln(x+1.3) + 5}{1.35x^2 + 6} + \sqrt{\frac{x+1.3}{(35x)^2 + 6}}$
12	$y = \frac{2x_1 + 1.4^{x_2}}{\operatorname{tg} x_1 + 2x_2} + \sqrt[3]{6x_1 + 8^{x_2}}$
13	$y = \frac{3x + \operatorname{tg} x}{2.36x + 6} + 2x + 1.4^x$
14	$y = \frac{e^{x-1.3} + \sin x}{x + 3.5} + 2x + 1.4 \sin x$
15	$y = \frac{x + 3 \cos(x^2 + 1.5)}{\operatorname{tg} x + 4.56} + \sin x \frac{2+x}{\sqrt{5^x + 56x}}$
16	$y = \frac{x + 3 \cos(x^2 + 1.5)}{\operatorname{tg} x + 4.56} + \sin(2x) + \cos(x^2 + 5)$
17	$y = \frac{\cos^2(x_1 + 1.3) + x_2}{2x_1 + e^{x_2}} + \cos(x_2^2 + 1.5) + \frac{x_2}{\cos(x_1^2 + 1.5)}$

<b>№</b>	<b>Функция</b>
18	$y = \frac{5x_1 + 1.3^{x_2}}{\cos(x_1 + x_2)} + \cos(x_2^2 + 1.5) + \frac{5 + x_2}{\sqrt{x_1}}$
19	$t = \frac{2 \cos\left(x - \frac{\pi}{6}\right)}{0.5 + \sin^2 y} \left(1 + \frac{z^2}{3 - z^2/5}\right).$
20	$u = \frac{\sqrt[3]{8 +  x - y ^2 + 1}}{x^2 + y^2 + 2} - e^{ x-y } \left(g^2 z + 1\right).$
21	$v = \frac{1 + \sin^2 \left(x + y\right)}{\left x - \frac{2y}{1+x^2y^2}\right } + \cos^2 \left(\operatorname{arctg} \frac{1}{z}\right).$
22	$w =  \cos x - \cos y  + 2 \sin^2 y \left(1 + z + \frac{z^2}{2} + \frac{z^3}{3} + \frac{z^4}{4}\right).$
23	$\alpha = \ln\left(y^{-\sqrt{ x }}\right) \left(x - \frac{y}{2}\right) + \sin^2 \operatorname{arctg} z.$
24	$\beta = \sqrt{10} \left(\sqrt{x + x^{y+2}} \operatorname{arcsin}^2 z -  x - y \right).$
25	$\gamma = 5 \operatorname{arctg} z - \frac{1}{4} \operatorname{arccos} \frac{x + 3 x - y  + x^2}{ x - y z + x^2}.$
26	$\varphi = \frac{e^{ x-y }  x - y ^{x+y}}{\operatorname{arctg} z \operatorname{arctg} w} + \sqrt[3]{x^6 + \ln^2 y}.$
27	$\psi = \left x^{\frac{y}{x}} - 3\sqrt[3]{\frac{y}{x}}\right  + \left y - x\right  \frac{\cos y - \frac{z}{ y-x }}{1 + \left y - x\right ^2}.$
28	$a = 2^{-x} \sqrt{x + 4\sqrt{ y }} \sqrt[3]{e^{x-1/\sin z}}.$

№	Функция
29	$b = y^{\sqrt[3]{ x }} + \cos^3 \frac{ x-y  \left( 1 + \frac{\sin^2 z}{\sqrt{x+y}} \right)}{e^{ x-y } + \frac{x}{2}}.$
30	$c = 2 \lfloor x \rfloor + \lceil x \rceil - \frac{y \left( \operatorname{arctg}(z) - \frac{\pi}{6} \right)}{ x  + \frac{1}{y^2 + 1}}.$
31	$f = \frac{\sqrt[4]{y + \sqrt[3]{x-1}}}{ x-y  \lceil \ln^2 z + \operatorname{tg} z \rceil}.$
32	$g = \frac{y^{x+1}}{\sqrt[3]{ y-2 } + 3} + \frac{x + \frac{y}{2}}{2 x+y } \lceil c + 1 \rceil^{\sin z}.$
33	$h = \frac{x^{y+1} + e^{y-1}}{1 + x y-\operatorname{tg} z } +  y-x ^2 + \frac{ y-x ^2}{2} - \frac{ y-x ^3}{3}$

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

#### Простейшая программа на Си

Простейшая программа на языке Си состоит всего из 8 символов и имеет следующий вид:

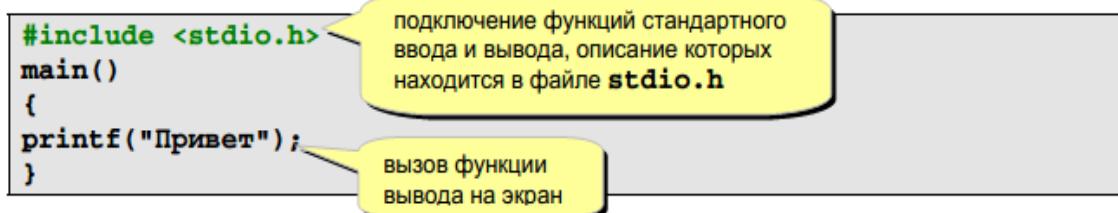
```
main()
{
}
```

Основная программа всегда называется именем **main** (будьте внимательны – Си различает большие и маленькие буквы, а все стандартные операторы Си записываются маленькими буквами). Пустые скобки означают,

что **main** не имеет аргументов. Фигурные скобки обозначают начало и конец основной программы – поскольку внутри них ничего нет, наша программа ничего не делает, она просто соответствует правилам языка Си, ее можно скомпилировать и полу-чить **exe**-файл.

### Вывод текста на экран

Составим теперь программу, которая делает что-нибудь полезное, например, выводит на экран слово «Привет».



Чтобы использовать стандартные функции, необходимо сказать транслятору, что есть функция с таким именем и перечислить тип ее аргументов – тогда он сможет определить, верно ли мы ее используем. Это значит, что надо включить в программу *описание* этой функции. Описания стандартных функций Си находятся в так называемых *заголовочных файлах* с расширением `*.h` (в каталоге `C:\Dev-Cpp\include`).

Для подключения заголовочных файлов используется директива (команда) препроцессора `#include`, после которой в угловых скобках ставится имя файла. Внутри угловых скобок не должно быть пробелов. Для подключения еще каждого нового заголовочного файла надо использовать новую команду `#include`.

Для вывода информации на экран используется функция `printf`. В простейшем случае она принимает единственный аргумент – строку в кавычках, которую надо вывести на эк-ран.

Каждый оператор языка Си заканчивается точкой с запятой.

Чтобы проверить эту программу, надо сначала «напустить» на нее транслятор, который переведет ее в машинные коды, а затем – компоновщик, который подключит стандартные функции и создаст исполняемый файл. Раньше все это делали, вводя команды в командной строке или с помощью так называемых пакетных файлов. На современном уровне все этапы создания, трансляции, компоновки, отладки и проверки программы объединены и выполняют-ся внутри специальной программы-оболочки, которую называют **интегрированная среда разработки** (*IDE* – *integrated development environment*).

В нее входят:

- ✓ редактор текста;
- ✓ транслятор;
- ✓ компоновщик;

- ✓ отладчик.

В среде *Dev-C++* вам достаточно набрать текст программы и нажать на одну клавишу, чтобы она выполнилась (если нет ошибок).

В оболочке *Dev-C++* для запуска программы надо нажать клавишу **F11**. Если в программе есть ошибки, вы увидите в нижней части экрана оболочки сообщения об этих ошибках (к сожалению, на английском языке). Если щелкнуть по одной из этих строчек, в тексте программы выделяется строка, в которой транслятору что-то не понравилось.

При поиске ошибок надо помнить, что

- ✓ часто ошибка сделана не в выделенной строке, а в предыдущей – проверяйте и ее тоже;
- ✓ часто одна ошибка вызывает еще несколько, и появляются так называемые наведенные ошибки.

Если запускать рассмотренную выше программу, то обнаружится, что программа сразу заканчивает работу и возвращается обратно в оболочку, не дав нам посмотреть результат ее работы на экране. Бороться с этим можно так – давайте скажем компьютеру, что в конце работы надо дождаться нажатия любой клавиши.

```
#include <stdio.h>
#include <conio.h>           подключение заголовочного
main()                         файла conio.h
{
    printf("Привет");        // вывод на экран
    getch();                  /* ждать нажатия клавиши */
}
```

Задержка до нажатия любой клавиши выполняется функцией **getch()**.

Описание этой функции находится в заголовочном файле **conio.h**.

Знаки **//** обозначают начало **комментария** — все правее них до конца строки не обрабатывается транслятором и служит нам для пояснения программы.

Комментарий также можно ограничивать парами символов **/\*** (начало комментария) и **\*/** (конец комментария). В этом случае комментарий может быть многострочный, то есть со-стоять из нескольких строк.

### Типы данных и переменные

Для обработки данных их необходимо хранить в памяти. При этом к этим данным надо как-то обращаться. Обычно люди обращаются друг к другу



**Пример 1.1** Ввести с клавиатуры два целых числа, и результат вывести на экран.

```
#include <stdio.h>
#include <conio.h>
main()
{
    int a, b, c; // объявление переменных
    printf ("Введите два целых числа \n"); // подсказка для ввода
    scanf ("%d%d", &a, &b); // ввод данных
    c = a + b; // вычисления (оператор присваивания)
    printf ("Результат: %d + %d = %d \n",
           a, b, c); // вывод результата
    getch();
}
```

Программа чаще всего содержит 4 части:

- ✓ объявление переменных;
- ✓ ввод исходных данных;
- ✓ обработка данных (вычисления);
- ✓ вывод результата.

Перед вводом данных необходимо вывести на экран подсказку (иначе компьютер будет ждать ввода данных, а пользователь не будет знать, что от него хочет машина).

Символы `\n` в функции `printf` обозначают переход в начало новой строки.

Для ввода данных используют функцию `scanf`.

```
scanf ( "%d%d", &a, &b );
```

Формат ввода – это строка в кавычках, в которой перечислены один или несколько форматов ввода данных:

<code>%d</code>	ввод целого числа (переменная типа <code>int</code> )
<code>%f</code>	ввод вещественного числа (переменная типа <code>float</code> )
<code>%c</code>	ввод одного символа (переменная типа <code>char</code> )

После формата ввода через запятую перечисляются адреса ячеек памяти, в которые надо записать введенные значения. Почувствуйте разницу:

- a** значение переменной **a**
- &a** адрес переменной **a**

Количество форматов в строке должно быть равно количеству адресов переменных в списке. Кроме того, тип переменных должен совпадать с указанным: например, если **a** и **b** – целые переменные, то следующие вызовы функций ошибочны:

```

scanf ( "%d%d", &a );           куда записывать второе введенное число?
scanf ( "%d%d", &a, &b, &c );    не задан формат для переменной c
scanf ( "%f%f", &a, &b );       нельзя вводить целые переменные по ве-
                                         щественному формату

```

Для вычислений используют **оператор присваивания**, в котором:

- ✓ справа от знака равенства стоит арифметическое выражение, которое надо вычислить;
- ✓ слева от знака равенства ставится имя переменной, в которую надо записать результат.

**c = a + b; // сумму a и b записать в c**

Для вывода чисел и значений переменных на экран используют функцию **printf**

```

printf ( "Результат: %d + %d = %d \n", a, b, c );

```

Содержание скобок при вызове функции **printf** очень похоже на функцию **scanf**. Сначала идет символьная строка — формат вывода — в которой можно использовать специальные символы:

<b>%d</b>	вывод целого числа
<b>%f</b>	вывод вещественного числа
<b>%c</b>	вывод одного символа
<b>%s</b>	вывод символьной строки
<b>\n</b>	переход в начало новой строки

Все остальные символы (кроме некоторых других специальных команд) просто выводятся на экран.

Одной строки формата недостаточно: в ней сказано, в какое место выводить данные, но не сказано, откуда их взять. Поэтому через запятую после формата вывода надо поставить список чисел или переменных, значения которых надо вывести, при этом можно сразу проводить вычисления.

```
printf ( "Результат: %d + %d = %d \n", a, 5, a+5 );
```

Так же, как и для функции **scanf**, надо следить за совпадением типов и количества переменных и форматов вывода.

Арифметические выражения, стоящие в правой части оператора присваивания, могут содержать:

- ✓ целые и вещественные числа (в вещественных числах целая и дробная часть разделяются **точкой**, а не запятой, как это принято в математике);
- ✓ знаки арифметических действий:
  - + сложение;
  - - вычитание;
  - \* умножение;

- / деление;
- % остаток от деления
- ✓ вызовы стандартных функций:
  - **abs(i)** модуль целого числа **i** ;
  - **fabs(x)** модуль вещественного числа **x** ;
  - **sqrt(x)** квадратный корень из вещественного числа **x** ;
  - **pow(x,y)** возведение **x** в степени **y**
- ✓ круглые скобки для изменения порядка действий.

При использовании деления надо помнить, что при делении целого числа на целое остаток от деления отбрасывается, таким образом, **7/4** будет равно 1. Если же надо получить вещественное число и не отбрасывать остаток, делимое или делитель надо преобразовать к вещественной форме.

Например:

```
int i, n;
float x;
i = 7;
x = i / 4;           // x=1, делится целое на целое
x = i / 4.;          // x=1.75, делится целое на дробное
x =(float) i / 4;   // x=1.75, делится дробное на целое
n = 7. / 4.;          // n=1, результат записывается в
                      // целую переменную
```

Наибольшие сложности из всех действий вызывает взятие остатка. Если надо вычислить остаток от деления переменной **a** на переменную **b** и результат записать в переменную **ostatok**, то оператор присваивания выглядит так:

```
ostatok = a % b;
```

В языках программирования арифметические выражения записываются в одну строчку, поэтому необходимо знать **приоритет** (старшинство) операций, то есть последовательность их выполнения. Сначала выполняются

- ✓ операции в скобках;
- ✓ вызовы функций;
- ✓ умножение, деление и остаток от деления, слева направо;
- ✓ сложение и вычитание, слева направо.

Например:

2	1	5	4	3	8	6	7
<b>x = ( a + 5 * b ) * fabs ( c + d ) - ( 3 * b - c );</b>							

Для изменения порядка выполнения операций используются круглые скобки. Выражение

$$y = \frac{4x+5}{(2x-15z)(3z-3)} - \frac{5x}{x+z+3}$$

в компьютерном виде примет следующую форму

$$y = \frac{(4*x + 5) / ((2*x - 15*z)*(3*z - 3)) - 5 * x}{(x + z + 3)};$$

В программировании часто используются несколько странные операторы присваивания, например:

$$i = i + 1;$$

Если считать это уравнением, то оно бессмысленно с точки зрения математики. Однако с точки зрения информатики этот оператор служит для увеличения значения переменной **i** на единицу.

Буквально это означает: взять старое значение переменной **i**, прибавить к нему единицу и за-писать результат в ту же переменную **i**.

В языке Си определены специальные операторы быстрого увеличения на единицу (*инкремента*)

$$\begin{array}{l} i++; \\ // или... \\ ++i; \end{array}$$

что равносильно оператору присваивания

$$i = i + 1;$$

и быстрого уменьшения на единицу (*декремента*)

$$\begin{array}{l} i--; \\ // или... \\ --i; \end{array}$$

что равносильно оператору присваивания

$$i = i - 1;$$

Между первой и второй формами этих операторов есть некоторая разница, но только тогда, когда они входят в состав более сложных операторов или условий.

Если мы хотим изменить значение какой-то переменной (взять ее старое значение, что-то с ним сделать и записать результат в эту же переменную), то удобно использовать сокращенную запись арифметических выражений:

Сокращенная запись	Полная запись
$x += a;$	$x = x + a;$
$x -= a;$	$x = x - a;$
$x *= a;$	$x = x * a;$
$x /= a;$	$x = x / a;$
$x %= a;$	$x = x \% a;$

## **5. Порядок выполнения работы**

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Из каких разделов состоит программа на языке Си?
2. Что такое оператор?
3. Какие операторы языка Си вам известны?
4. Зачем нужен оператор присваивания? Какой вид он имеет?
5. Что может быть записано в правой части оператора присваивания?
6. Что такое переменная?
7. Что такое константа?
8. Какие правила применяются для создания имен переменных?
9. Что такое идентификатор?
10. Почему знак умножения всегда записывают явно (например, пишут  $a*t$ , а не  $at$ )?
11. Как описываются переменные в Си?
12. Какие стандартные числовые типы языка Си вам известны?
13. Что вам известно о соответствии типов переменных в языке Си?
14. Какие арифметические операции можно выполнять в Си?
15. Что вам известно о приоритете арифметических действий в Си?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программированием для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание ветвящихся алгоритмов и программ»

Минск  
2017

## Лабораторная работа № 2

**Тема работы:** «Разработка, отладка и испытание ветвящихся алгоритмов и программ»

### 1. Цель работы

Изучить различные виды условного оператора

### 2. Задание

Задачи первого и второго уровня обязательны для всех. Номер варианта соответствует вашему номеру по списку.

#### Задачи первого уровня

1. Две окружности заданы координатами центров и радиусами. Составьте программу, которая находит координаты точек пересечения данных окружностей или выдает сообщение об отсутствии таковых.
2. Окружность задана координатами центра и радиусом. Составьте программу, которая определяет координаты точек пересечения данной окружности с прямой  $ax + by + c = 0$ .
3. Прямоугольник задан координатами двух противоположных своих вершин. Определите, принадлежит ли точка с заданными координатами области прямоугольника, если стороны прямоугольника параллельны осям координат.
4. Прямоугольник задан координатами своих вершин. Определите, принадлежит ли окружность с заданным радиусом и координатами центра области прямоугольника.
5. Даны длины сторон треугольника. Определите, является ли данный треугольник прямоугольным.
6. Даны радиус круга и сторона квадрата. Проверьте, пройдет ли квадрат в круг?
7. Введите три числа, найдите наименьшее отношение пар этих чисел.
8. Даны радиус круга и сторона квадрата. Проверьте, пройдет ли круг в квадрат?
9. Функция  $sign(x)$  задана аналитически следующим образом:

$$sign(x) = \begin{cases} 1, & \text{если } x > 0, \\ 0, & \text{если } x = 0, \\ -1, & \text{если } x < 0. \end{cases}$$

10. Введите значение  $x$ , выведите значение функции  $sign(x)$ .
11. На плоскости расположена окружность радиусом  $R$  с центром в начале координат. Введите заданные координаты точки и определите, находится ли она на окружности. Результат присвойте символьной переменной. Точка находится на окружности, если длина радиус-вектора, соединяющего начало координат с заданной точкой, равна  $R$  (проверку на

равенство осуществите с точностью до  $e = 1.0E-3$ .

12. Даны координаты точки  $M(x, y)$ . Определите, принадлежит ли данная точка замкнутому множеству  $D$  (рис. 2.1).
13. Даны координаты точки  $M(x, y)$ . Определите, принадлежит ли данная точка замкнутому множеству  $D$  (рис. 2.2).  $\Delta ABC$  – равносторонний.

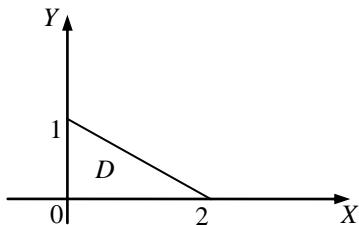


Рис. 2.1

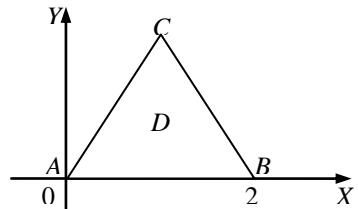


Рис. 2.2

14. Даны координаты точки  $M(x, y)$ . Определите, принадлежит ли данная точка замкнутому множеству  $D$  (рис. 2.3).
15. Даны координаты точки  $M(x, y)$ . Определите, принадлежит ли данная точка замкнутому множеству  $D$  (рис. 2.4).

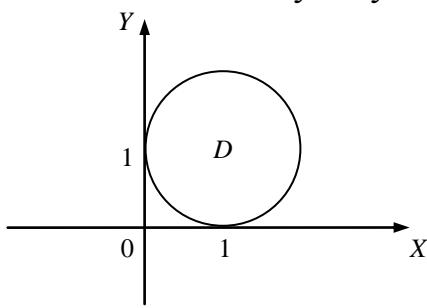


Рис. 2.3

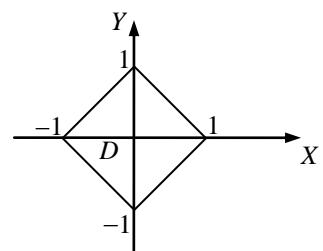


Рис. 2.4

16. Даны координаты точки  $M(x, y)$ . Определите, принадлежит ли данная точка замкнутому множеству  $D$ , заданному системой ограничений:
- $$\begin{cases} x + y \leq 1, \\ 2x - y \leq 1, \\ y \geq 0. \end{cases}$$

17. Даны координаты точки  $M(x, y)$ . Определите, принадлежит ли данная точка замкнутому множеству  $D$  (рис. 2.5).

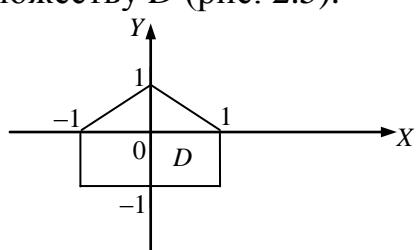


Рис. 2.5

18. Введите два положительных числа и покажите, что среднее

арифметическое этих чисел не меньше их среднего геометрического.

19. Введите два положительных числа и покажите, что среднее геометрическое этих чисел не меньше их среднего гармонического.
20. Даны длины трех отрезков. Определите, можно ли из этих отрезков сложить треугольник.
21. Даны длины сторон треугольника. Определите, является ли данный треугольник равнобедренным.
22. Даны длины сторон треугольника. Определите, является ли данный треугольник равносторонним.
23. Даны длины сторон треугольника. Определите, является ли данный треугольник разносторонним.
24. Даны в градусах величины двух углов треугольника. Определите, является ли данный треугольник равнобедренным.
25. Найдите наибольшее значение из трех  $f(1)$ ,  $f(2)$  и  $f(3)$ , где  $f(x) = \sin(5x)$ .
26. Даны в градусах величины двух углов треугольника. Определите, является ли данный треугольник остроугольным.
27. Вывести какой-либо вопрос и несколько возможных ответов на него, один из которых верный. После ввода пользователем номера ответа, который он считает верным, выдать сообщение о правильности его выбора.
28. Вывести какой-либо вопрос и несколько возможных ответов на него, некоторые из которых верные. После ввода пользователем номеров ответов, которые он считает верными, выдать сообщение о правильности его выбора.
29. Треугольник задан координатами вершин. Определите, принадлежит ли точка с заданными координатами области треугольника?
30. Треугольник задан координатами вершин. Определите находится ли отрезок внутри данного треугольника.

### **Задачи второго уровня**

1. Два треугольника заданы координатами своих вершин. Определите, можно ли из этих треугольников составить прямоугольник.
2. Два треугольника заданы координатами своих вершин. Определите радиус, описанной окружности, какого треугольника больше.
3. Два треугольника заданы координатами своих вершин. Определите радиус, вписанной окружности, какого треугольника больше.
4. Треугольник задан координатами своих вершин. Определите длину большей стороны треугольника.
5. Треугольник задан координатами своих вершин. Определите длину большей медианы треугольника.
6. Треугольник задан координатами своих вершин. Определите длину меньшей стороны треугольника.
7. Треугольник задан координатами своих вершин. Определите длину меньшей высоты треугольника.

8. Треугольник задан координатами своих вершин. Определите величину в градусах большего угла треугольника.
9. Треугольник задан координатами своих вершин. Определите величину в градусах меньшего угла треугольника.
10. Четырехугольник задан координатами своих вершин. Определите, является ли данный четырехугольник параллелограммом.
11. Четырехугольник задан координатами своих вершин. Определите, является ли данный четырехугольник ромбом.
12. Четырехугольник задан координатами своих вершин. Определите, является ли данный четырехугольник прямоугольником.
13. Четырехугольник задан координатами своих вершин. Определите, является ли данный четырехугольник квадратом.
14. Дано трехзначное натуральное число. Определите, является ли сумма цифр данного числа четной.
15. Дано трехзначное натуральное число. Определите, является ли вторая цифра числа наибольшей.
16. Дано трехзначное натуральное число. Получите наименьшее число, составленное из цифр данного числа.
17. Даны два натуральных числа. Определите, является ли одно из них делителем другого.
18. На шахматной доске стоят две шахматных фигуры. Их координаты заданы номерами столбцов и строк, на которых они находятся. Одна из них - шахматный конь. Определите, может ли конь побить другую фигуру.
19. На шахматной доске стоят две шахматных фигуры. Их координаты заданы номерами столбцов и строк, на которых они находятся. Одна из них - шахматный слон. Определите, может ли слон побить другую фигуру.
20. На шахматной доске стоят две шахматных фигуры. Их координаты заданы номерами столбцов и строк, на которых они находятся. Одна из них - шахматный король. Определите, может ли король побить другую фигуру.
21. На шахматной доске стоят две шахматных фигуры. Их координаты заданы номерами столбцов и строк, на которых они находятся. Одна из них - шахматная ладья. Определите, может ли ладья побить другую фигуру.
22. На шахматной доске стоят две шахматных фигуры. Их координаты заданы номерами столбцов и строк, на которых они находятся. Одна из них - шахматная пешка. Определите, может ли пешка побить другую фигуру.
23. На шахматной доске стоят две шахматных фигуры. Их координаты заданы номерами столбцов и строк, на которых они находятся. Одна из них - шахматная пешка. Определите, может ли пешка стать следующим

ходом ферзем.

24. На шахматной доске стоят две шахматных фигуры. Их координаты заданы номерами столбцов и строк, на которых они находятся. Одна из них - шахматный конь. Определите, за какое число ходов конь может побить другую фигуру.
25. На шахматной доске стоят две шахматных фигуры. Их координаты заданы номерами столбцов и строк, на которых они находятся. Одна из них - шахматный ферзь. Определите, за какое число ходов ферзь может побить другую фигуру.
26. На доске игры в шашки стоят шашки разного цвета: первая белая, вторая черная. Их координаты заданы номерами столбцов и строк, на которых они находятся. Определите, может ли белая шашка следующим ходом стать дамкой.
27. На доске игры в шашки стоят шашки разного цвета: первая белая, вторая черная. Их координаты заданы номерами столбцов и строк, на которых они находятся. Определите, может ли белая шашка следующим ходом побить черную шашку.
28. На доске игры в шашки стоят две шашки разного цвета: первая белая, вторая черная. Их координаты заданы номерами столбцов и строк, на которых они находятся. Определите, может ли белая шашка при наилучших ходах черной шашки стать дамкой.
29. В каждой больничной палате четыре койки. Введите количество палат, количество больных мужчин и количество больных женщин. Определите, сколько всего свободных мест.
30. В каждой больничной палате четыре койки. Введите количество палат, количество больных мужчин и количество больных женщин. Сколько больных мужского пола можно еще положить в больницу?

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

В простейших программах все команды выполняются одна за другой последовательно. Так реализуются *линейные алгоритмы*. Однако часто надо выбрать тот или иной вариант действий в зависимости от некоторых условий: если условие верно, поступать одним способом, а если неверно — другим. Для этого используют **разветвляющиеся алгоритмы**, которые в языках программирования представлены в виде **условных операторов**. В языке Си существует два вида условных операторов:

- ✓ оператор **if – else** для выбора из двух вариантов;
- ✓ оператор множественного выбора **switch** для выбора из нескольких вариантов.

**Пример 2.1** Ввести с клавиатуры два вещественных числа и определить наибольшее из них.

По условию задачи нам надо вывести один из двух вариантов ответа: если первое число больше второго, то вывести на экран его, если нет — то второе число. Ниже показаны два варианта решения этой задачи: в первом результат сразу выводится на экран, а во втором наибольшее из двух чисел сначала записывается в третью переменную **Max**.

```
#include <stdio.h>
#include <conio.h>
main()
{
    float A, B;
    printf ("Введите A и B :");
    scanf ( "%f%f", &A, &B );
    if ( A > B )
    {
        printf ( "Наибольшее %f",
                 A );
    }
    else
    {
        printf ( "Наибольшее %f",
                 B );
    }
    getch();
}
```

```
#include <stdio.h>
#include <conio.h>
main()
{
    float A, B, Max;
    printf("Введите A и B : ");
    scanf ( "%f%f", &A, &B );
    if ( A > B ) // заголовок
    {
        Max = A; // блок «если»
    }
    else
    {
        Max = B; // блок «иначе»
    }
    printf ( "Наибольшее %f",
             Max );
    getch();
}
```

Условный оператор имеет следующий вид:

```
if ( условие ) // заголовок с условием
{
    ...
    ... // блок «если» – операторы, которые выполняются,
         // если условие в заголовке истинно
}
else
{
    ...
    ... // блок «иначе» – операторы, которые выполняются,
         // если условие в скобках ложно
}
```

Эта запись представляет собой единый оператор, поэтому между скобкой, завершающей блок «если» и словом **else** не могут находиться никакие операторы.

После слова **else** никогда **НЕ** ставится условие — блок «иначе» выполняется тогда, когда основное условие, указанное в скобках после **if**, ложно.

Если в блоке «если» или в блоке «иначе» только один оператор, то фигурные скобки можно не ставить.

В условии можно использовать знаки логических отношений:

> <	больше, меньше
>= <=	больше или равно, меньше или равно
==	равно
!=	не равно

В языке Си любое число, не равное нулю, обозначает истинное условие, а ноль — ложное условие.

Если в блоке «иначе» не надо ничего делать (например: «если в продаже есть мороженое, купи мороженое», а если нет ...), то весь блок «иначе» можно опустить и использовать сокращенную форму условного оператора:

```
if ( условие )
{
    ...
    // что делать, если условие истинно
}
```

Например, решение предыдущей задачи могло бы выглядеть так:

```
#include <stdio.h>
#include <conio.h>
main()
{
    float A, B, Max;
    printf("Введите A и B : ");
    scanf ( "%f%f", &A, &B );
    Max = A;
    if ( B > A )
        Max = B;
    printf ( "Наибольшее %f", Max );
    getch();
}
```

В блоки «если» и «иначе» могут входить любые другие операторы, в том числе и другие вложенные условные операторы; при этом оператор **else** относится к ближайшему предыдущему **if**:

```
if ( A > 10 )
    if ( A > 100 )
        printf ( "У вас очень много денег." );
    else
        printf ( "У вас достаточно денег." );
else
    printf ( "У вас маловато денег." );
```

Чтобы легче разобраться в программе, все блоки «если» и «иначе» (вместе с ограничивающими их скобками) сдвигаются вправо на 2-3 символа (запись «лесенкой»).

Простейшие условия состоят из одного отношения (больше, меньше и т.д.). Иногда надо написать условие, в котором объединяются два или более простейших отношений. Например, фирма отбирает сотрудников в возрасте от 25 до 40 лет (включительно). Тогда простейшая про-грамма могла бы выглядеть так:

```

#include <stdio.h>
#include <conio.h>
main()
{
int age;
printf ( "\nВведите ваш возраст: " );
scanf ( "%d", &age );
if ( 25 <= age && age <= 40 ) // сложное условие
    printf ( "Вы нам подходите." );
else
    printf ( "Извините, Вы нам не подходите." );
getch();
}

```

Сложное условие состоит из двух или нескольких простых отношений, которые объединяются с помощью знаков *логических операций*:

- ✓ операция ***И*** — требуется одновременное выполнение двух условий **условие\_1 && условие\_2**  
эту операцию можно описать следующей таблицей (она называется *таблицей истинности*)

условие_1	условие_2	условие_1 && условие_2
ложно (0)	ложно (0)	ложно(0)
ложно (0)	истинно (1)	ложно(0)
истинно (1)	ложно (0)	ложно(0)
истинно (1)	истинно (1)	истинно (1)

- ✓ операция ***ИЛИ*** — требуется выполнение хотя бы одного из двух условий (или обоих сразу)  
**условие\_1 || условие\_2**  
эту операцию можно описать следующей таблицей (она называется *таблицей истинности*)

условие_1	условие_2	условие_1    условие_2
ложно (0)	ложно (0)	ложно(0)
ложно (0)	истинно (1)	истинно (1)
истинно (1)	ложно (0)	истинно (1)
истинно (1)	истинно (1)	истинно (1)

- ✓ в сложных условиях иногда используется операция ***НЕ*** — отрицание условия (или об-ратное условие)  
**! условие**

например, следующие два условия равносильны

**A > B**

**! ( A <= B )**

Порядок выполнения (*приоритет*) логических операций и отношений:

- ✓ операции в скобках;
- ✓ операция ***НЕ***;
- ✓ логические отношения **>, <, >=, <=, ==, !=;**
- ✓ операция ***И***;
- ✓ операций ***ИЛИ***.

Для изменения порядка действий используются круглые скобки.

Если надо выбрать один из нескольких вариантов в зависимости от значения некоторой целой или символьной переменной, можно использовать несколько вложенных операторов **if**, но значительно удобнее использовать специальный оператор **switch**.

**Пример 2.2** Составить программу, которая вводит с клавиатуры русскую букву и выводит на экран название животного на эту букву.

```
#include <stdio.h>
#include <conio.h>
main()
{
    char c;
    printf("\nВведите первую букву:");
    scanf("%c", &c); // ввести букву
    switch ( c )      // заголовок оператора выбора
    {
        case 'а': printf("\nАнтилопа"); break;
        case 'б': printf("\nБарсук"); break;
        case 'в': printf("\nВолк"); break;
        default:   printf("\nНе знаю я таких!"); // по умолчанию
    }
    getch();
}
```

Оператор множественного выбора **switch** состоит из заголовка и тела оператора, заключенного в фигурные скобки.

В заголовке после ключевого слова **switch** в круглых скобках записано имя переменной (целой или символьной). В зависимости от значения этой переменной делается выбор между несколькими вариантами.

Каждому варианту соответствует метка **case**, после которой стоит одно из возможных значений этой переменной и двоеточие; если значение переменной совпадает с одной из меток, то программа переходит на эту метку и выполняет все последующие операторы.

Оператор **break** служит для выхода из тела оператора **switch**. Если убрать все операторы **break**, то, например, при нажатии на букву *a* будет напечатано:

Антилопа  
Барсук  
Волк  
Не знаю таких!

Если значение переменной не совпадает ни с одной из меток, программа переходит на метку **default** (по умолчанию, то есть если ничего другого не указано).

Можно ставить две метки на один оператор. Например, чтобы программа реагировала как на большие, так и на маленькие буквы, надо в теле оператора **switch** написать так:

```
case 'a':  
case 'A':  
    printf("\nАнтилопа"); break;  
case 'б':  
case 'Б':  
    printf("\nБарсук"); break;
```

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## 6. Форма отчета о работе

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_

\_\_\_\_\_

## 7. Контрольные вопросы и задания

1. Что такое алгоритм с ветвлением?
2. Как записывается условный оператор и оператор выбора в Си?
3. Что такое полная и сокращенная записи условного оператора?
4. Что используется в качестве условий в операторе ветвления?
5. Какие знаки отношений можно использовать при составлении условий?
6. Что такое составное условие?
7. Каковы правила записи составных условий?
8. Какие вы знаете логические операции?
9. Какие действуют правила старшинства логических операций?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ на циклы с  
известным числом повторений»

Минск  
2017

## Лабораторная работа № 3

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ на циклы с известным числом повторений»

### 1. Цель работы

Научить применять операторы циклов с известным числом повторений, при решении задач разного класса

### 2. Задание

Задачи первого и второго уровня обязательны для всех. Номер варианта соответствует вашему номеру по списку.

Что бы определить вариант задачи второго уровня, необходимо воспользоваться следующей формулой:  $(N \% 17) + 1$ , где  $N$  – это номер по списку,  $\%$  - остаток от деления.

#### Задачи первого уровня

1. Напечатайте таблицу стоимости порций сыра весом 50, 100, 150, ..., 1000 г (цена 1 кг 280 руб.)
  2. Вычислите приближенно площадь фигуры, ограниченной функцией  $y = x^2$  и прямой  $y = 25$ , разбивая интервал изменения  $x$  на 100 частей и суммируя площади прямоугольников с основанием, равным 0,10 и высотой, определяемой значением функции в середине основания (высота прямоугольника в точке  $x$  равна  $25 - x^2$ ).
  3. Начав тренировки, спортсмен в первый день пробежал 10 км. Каждый следующий день он увеличивал дневную дистанцию на 10 % от дистанции предыдущего дня. Какой суммарный путь пробежит спортсмен за  $N$  дней?
  4. Определите суммарный объем (в литрах) 12-ти вложенных друг в друга шаров со стенками 5 мм. Внутренний диаметр внутреннего шара равен 10 см. Считайте, что шары вкладываются друг в друга без зазоров.
  5. Вычислите сумму  $1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n}$ .
  6. Пусть  $n$  – натуральное число и пусть  $n!!$  означает  $1 \cdot 3 \cdot \dots \cdot n$  для нечетного  $n$  и  $2 \cdot 4 \cdot \dots \cdot n$  для четного  $n$ . Для заданного натурального  $n$  вычислите  $n!!$ .
  7. Вычислите сумму  $\sum_{i=1}^n \frac{1}{i!}$ .
  8. Найдите  $n$ -й член ряда Фибоначчи, элементы которого вычисляются по формулам:  $a_1 = a_2 = 1$ ;  $a_i = a_{i-1} + a_{i-2}$ , ( $i > 2$ ).
- П р и м е ч а н и е. Для нахождения членов ряда используйте только две переменные  $a$  и  $b$ .
9. Найдите сумму  $n$ -членов ряда Фибоначчи, используя также только две переменные для нахождения суммы ряда.

10. Выведите на экран таблицу значений:  $1, 1 + h, 1 + 2h, \dots, 1 + nh$ .
11. Проверьте формулу  $1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$ .
12. Проверьте формулу  $1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(n+2)}{6}$ .
13. Вычислите сумму  $\sum_{i=1}^n \frac{1}{2^i}$ .
14. Вычислите число сочетаний из  $n$  по  $m$   $C_n^m = \frac{n!}{m!(n-m)!}$ .
15. Вычислите число размещений из  $n$  по  $m$
- $$A_n^m = \frac{n!}{(n-m)!} = n \cdot (n-1) \cdot \dots \cdot (n-m+1)$$
16. Выведите на экран таблицу значений функции  $y=a^x$  для  $x$ , изменяющегося от  $a$  до  $b$  с шагом  $h$ .
17. Вычислите сумму  $\sum_{i=1}^n \frac{1}{i^3}$ .
18. Вычислите сумму  $\sum_{i=1}^n (-1)^i 2^i$ .
19. Вычислите сумму  $\sum_{i=1}^n \frac{(-1)^i}{2i}$ .
20. Вычислите произведение  $P = 1 \cdot 3 \cdot 5 \cdot 7 \cdot \dots \cdot (2n+1)$  для заданного  $n$ .
21. Вычислите произведение  $P = 2 \cdot 4 \cdot 6 \cdot 8 \cdot \dots \cdot 2n$  для заданного  $n$ .
22. Вычислите сумму  $S = 1 + 3 + 3 + 5 + 5 + 7 + \dots + (2n-1) (2n+1)$  для заданного  $n$ .
23. Вычислите произведение  $S = (1+3) \cdot (3+5) \cdot \dots \cdot ((2n-1)+(2n+1))$  для заданного  $n$ .
24. Выведите на экран элементы последовательности  $a_n = a_{n-1} + nd$  для  $n$ , изменяющегося от 1 до  $k$ ,  $a_0 = 0$ ;  $k$  и  $d$  заданные натуральные числа.
25. Вычислите сумму  $A = 1 + (1+2) + (2+3) + (3+4) + \dots + ((n-1)+n)$
26. Составьте таблицу умножения для заданного числа  $N$ , которая содержит результаты умножения  $1 \cdot N, 2 \cdot N, \dots, N \cdot N$ .
27. Напечатайте таблицу значений функции  $y = x^3$  при изменении  $x$  в интервале от  $A$  до  $B$  с шагом  $H$ .
28. Проверьте формулу
- $$1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{(n(n+1))^2}{4}$$

### Задачи второго уровня

1. Проверьте справедливость неравенства для заданного  $n$ .
- $$4^n > 7n - 5$$
2. Проверьте справедливость неравенства для заданного  $n$ .
- $$2^n > 5n + 1$$
3. Вычислите значение интеграла  $\int_0^{\pi/2} \sin(x) dx$  по общей формуле трапеций с

заданным числом разбиений.

П р и м е ч а н и е. Общая формула трапеций имеет вид:

$$\int_a^b f(x)dx \approx \frac{b-a}{n} \left[ \frac{f(a)+f(b)}{2} + f(a+h) + f(a+2h) + \dots + f(a+(n-1)h) \right].$$

4. Вычислите значение интеграла  $\int_0^{\pi/2} \sin(x)dx$  по общей формуле Симпсона с заданным числом разбиений.

П р и м е ч а н и е. Общая формула Симпсона имеет вид:

$$\int_a^b f(x)dx \approx \frac{b-a}{3n} \left[ f(a) + 2[f(a+2h) + f(a+4h)] + \dots + f(a+2(n-1)h) \right]$$

5. Покажите, что при всех натуральных  $n$  число  $n^3 + 2n$  кратно 3.

6. Вычислите сумму  $A = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \dots + \frac{1}{(n-1) \cdot n}$ .

7. Вычислите сумму

$$A = \frac{1}{1 \cdot 4} + \frac{1}{4 \cdot 7} + \frac{1}{7 \cdot 10} + \dots + \frac{1}{(3 \cdot (n-1) + 1) \cdot (3 \cdot n + 1)}.$$

8. Вычислите сумму

$$A = \frac{1}{1 \cdot 3 \cdot 5} + \frac{1}{3 \cdot 5 \cdot 7} + \dots + \frac{1}{(2n-1) \cdot (2n+1) \cdot (2n+3)}.$$

9. Вычислите сумму

$$A = \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{2 \cdot 3 \cdot 4} + \dots + \frac{1}{n \cdot (n+1) \cdot (n+2)}.$$

10. Вычислите произведение  $\left(1 - \frac{1}{2}\right) \left(1 - \frac{1}{3}\right) \dots \left(1 - \frac{1}{n+1}\right)$ .

11. Вычислите сумму

$$A = \frac{1}{1 \cdot 2 \cdot 3 \cdot 4} + \frac{1}{2 \cdot 3 \cdot 4 \cdot 5} + \dots + \frac{1}{n \cdot (n+1) \cdot (n+2) \cdot (n+3)}.$$

12. Проверьте справедливость неравенства для заданного  $k$ .

$$\frac{1 \cdot 3 \cdot 5 \cdot \dots \cdot (2k-1)}{2 \cdot 4 \cdot 6 \cdot \dots \cdot 2k} < \frac{1}{\sqrt{3k-1}}.$$

13. Покажите, что при всех натуральных  $n$  число  $n^3 + 2n$  кратно 3.

14. Покажите, что при всех натуральных  $n$  число

$$3^{3n+2} + 5 \cdot 2^{3n+1}$$

- кратно 19.

15. Покажите, что при всех натуральных  $n$  число

$$2^{n+5} \cdot 3^{4n} + 5^{3n+1}$$

- кратно 37.

16. Вычислите значение интеграла  $\int_0^{\pi/2} \sin(x)dx$  по общей формуле левых

прямоугольников с заданным числом разбиений.

П р и м е ч а н и е. Формула левых прямоугольников имеет вид:

$$\int_a^b f(x)dx \approx \frac{b-a}{n} (f(a) + f(a+h) + f(a+2h) + \dots + f(a+(n-1)h))$$

17. Вычислите значение интеграла  $\int_0^{\pi/2} \sin(x)dx$  по общей формуле правых

прямоугольников с заданным числом разбиений.

П р и м е ч а н и е. Формула правых прямоугольников имеет вид:

$$\int_a^b f(x)dx \approx \frac{b-a}{n} (f(a+h) + f(a+2h) + f(a+3h) + \dots + f(a+nh))$$

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

Теперь посмотрим, как вывести на экран это самое приветствие 10 раз. Конечно, можно написать 10 раз оператор **printf**, но если надо вывести строку 200 раз, то программа значительно увеличится. Поэтому надо использовать **циклы**.

**Цикл** - это последовательность команд, которая выполняется несколько раз.

Часто мы заранее знаем заранее (или можем рассчитать), сколько раз нам надо выполнить какую-то операцию. В некоторых языках программирования для этого используется цикл **repeat** – «повтори заданное количество раз». Подумаем, как выполнять такой цикл. В памяти выделяется ячейка и в нее записывается число повторений. Когда программа выполняет тело цикла один раз, содержимое этой ячейки (*счетчик*) уменьшается на единицу. Выполнение цикла заканчивается, когда в этой ячейке будет ноль.

В языке Си цикла **repeat** нет, а есть цикл **for**. Он не скрывает ячейку-счетчик, а требует явно объявить ее (выделить под нее память), и даже позволяет использовать ее значение в теле цикла.

**Пример 3.1** Вывести на экран приветствие 10 раз.

```
#include <stdio.h>
#include <conio.h>
main()
{
    int i;                                // объявление переменной цикла
    for ( i = 1; i <= 10; i ++ )           // заголовок цикла
    {
        printf("Привет");                // начало цикла (открывающая скобка)
    }                                       // тело цикла
                                         // конец цикла (закрывающая скобка)
    getch();
}
```

Цикл **for** используется тогда, когда количество повторений цикла заранее известно или может быть вычислено.

Цикл **for** состоит из заголовка и тела цикла.

В заголовке после слова **for** в круглых скобках записываются через точку с запятой три выражения:

- ✓ **начальные значения:** операторы присваивания, которые выполняются один раз перед выполнением цикла;
- ✓ **условие,** при котором выполняется следующий шаг цикла, если условие неверно, работа цикла заканчивается; если оно неверно в самом начале, цикл не выполняется ни одного раза (говорят, что это цикл с предусловием, то есть условие проверяется перед выполнением цикла);
- ✓ **действия в конце каждого шага цикла** (в большинстве случаев это операторы присваивания).

В каждой части заголовка может быть несколько операторов, разделенных запятыми. Примеры заголовков:

```
for ( i = 0; i < 10; i ++ ) { ... }
for ( i = 0, x = 1.; i < 10; i += 2, x *= 0.1 ) { ... }
```

Тело цикла заключается в фигурные скобки; если в теле цикла стоит всего один оператор, скобки можно не ставить.

В тело цикла могут входить любые другие операторы, в том числе и другие циклы (такой прием называется «вложенные циклы»).

Для того, чтобы легче разобраться в программе, все тело цикла и ограничивающие его скобки сдвигаются вправо на 2-3 символа (запись «лесенкой»).

**Пример 3.2** Написать программу, которая вводит с клавиатуры натуральное число  $N$  и выводит на экран квадраты всех целых чисел от 1 до  $N$  таком виде:

```
Квадрат числа 1 равен 1
Квадрат числа 2 равен 4
```

...

```
#include <stdio.h>
#include <conio.h>
main()
{
    int i, N; // i - переменная цикла
    printf ("Введите число N: "); // подсказка для ввода
    scanf ("%d", &N); // ввод N с клавиатуры
    for ( i = 1; i <= N; i ++ ) // цикл: для всех i от 1 до N
    {
        printf ("Квадрат числа %d равен %d\n", i, i*i);
    }
    getch();
}
```

Мы объявили две переменные: **N** — максимальное число, **i** — вспомогательная переменная, которая в цикле принимает последовательно все значения от **1** до **N**. Для ввода значения **N** мы напечатали на экране подсказку и использовали функцию **scanf** с форматом **%d** (ввод целого числа).

При входе в цикл выполняется оператор **i = 1**, и затем переменная **i** с каждым шагом увеличивается на единицу (**i ++**). Цикл выполняется пока истинно условие **i <= n**. В теле цикла единственный оператор вывода печатает на экране само число и его квадрат по заданному формату. Для возведения в квадрат или другую невысокую степень лучше использовать умножение.

**Пример 3.3** Найти сумму первых 20 элементов последовательности

$$S = \frac{1}{2} - \frac{2}{4} + \frac{3}{8} - \frac{4}{16} + \dots$$

Чтобы решить эту задачу, надо определить закономерность в изменении элементов. В данном случае можно заметить, что

- ✓ каждый элемент представляет собой дробь;
- ✓ числитель дроби при переходе к следующему элементу возрастает на единицу;
- ✓ знаменатель дроби с каждым шагом увеличивается в 2 раза;
- ✓ знаки перед дробями чередуются (плюс, минус и т.д.).

Любой элемент последовательности можно представить в виде

$$a_i = \frac{zc}{d},$$

где изменение переменных **z**, **c** и **d** описываются следующей таблицей (для первых пяти элементов)

<b>i</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>z</b>	1	-1	1	-1	1
<b>c</b>	1	2	3	4	5
<b>d</b>	2	4	8	16	32

У переменной **z** меняется знак (эту операцию можно записать в виде **z = -z**), значение переменной **c** увеличивается на единицу (**c ++**), а переменная **d** умножается на 2 (**d = d \* 2**). Алгоритм решения задачи можно записать в виде следующих шагов:

- ✓ записать в переменную **S** значение **0**; в этой ячейке будет накапливаться сумма;
- ✓ записать в переменные **z**, **c** и **d** начальные значения (для первого элемента): **z = 1**, **c = 1**, **d = 2**.
- ✓ сделать 20 раз:
  - добавить к сумме значение очередного элемента;
  - изменить значения переменных **z**, **c** и **d** для следующего элемента.

```

#include <stdio.h>
main()
{
float S, z, c, d;
int i;
S = 0; z = 1; c = 1; d = 2; // начальные значения
for ( i = 1; i <= 20; i ++ )
{
    S = S + z*c/d;           // добавить элемент к сумме
    z = - z;                 // изменить переменные z, c, d
    c++;
    d = d * 2;
}
printf("Сумма S = %f", S);
}

```

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## 6. Форма отчета о работе

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

## 7. Контрольные вопросы и задания

1. Для чего предназначен оператор цикла?
2. Какие виды циклов есть в Си?
3. Какой формат записи имеет оператор **for**? Какие существуют варианты этого цикла?
4. Как работает оператор **for**? В каких случаях применяется?
5. Что является телом цикла?
6. Как в теле цикла выполнить несколько операторов?
7. Почему перед выполнением цикла некоторым переменным нужно задавать

начальные значения?

8. При каких условиях оператор **for** не выполнится ни разу?

### **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ на циклы с  
неизвестным числом повторений»

Минск  
2017

## Лабораторная работа № 4

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ на циклы с неизвестным числом повторений»

### 1. Цель работы

Научить применять операторы циклов с неизвестным числом повторений, при решении задач разного класса

### 2. Задание

Задачи первого и второго уровня обязательны для всех. Номер варианта соответствует вашему номеру по списку. Задачу нужно решить двумя способами: с помощью цикла **while** и цикла **do while**

#### Задачи первого уровня

**1.** Значение функции  $\sin^2(x)$  можно вычислить с помощью разложения ее в ряд Маклорена

$$\sin^2 x = x^2 - \frac{x^4}{3} + \frac{2x^6}{45} - \dots + (-1)^{n-1} \frac{2^{2n-1}}{(2n-1)!} x^{2n} + \dots$$

Вычислите  $\sin^2(x)$  с точностью EPS, т.е. вычисление суммы ряда нужно продолжать до тех пор, пока абсолютная величина очередного члена ряда не станет меньше EPS. Определите количество членов ряда, которое для этого понадобилось.

**2.** Введите два натуральных числа  $m$  и  $n$ . Проверьте, являются ли данные числа взаимно-простыми.

**3.** Изменяя  $x$  от  $a$  с шагом  $h$ , определите, при каком значении  $x$   $\text{SIN}(x)$  станет больше  $\text{COS}(x)$ .

**4.** Найдите наименьшее общее кратное натуральных чисел  $k, m, n$ .

**5.** Напишите программу сложения двух рациональных дробей. Если полученный результат является сократимой дробью, то сократите эту дробь.

**6.** Дано целое число  $m > 10$ . Получите наибольшее целое  $k$ , при котором  $4^k < m$ .

**7.** Напишите программу умножения двух рациональных дробей. Если полученный результат является сократимой дробью, то сократите эту дробь.

**8.** Вычислите значение корня  $n$ -ой степени  $y = \sqrt[n]{x}$  с точностью EPS с использованием итерационной формулы Ньютона:

$$Y_0 = 1$$

$$Y_i = Y_{i-1} - \frac{X - Y_{i-1}^{n-1}}{n(Y_{i-1}^{n-1})} \quad (i = 1, 2, 3, \dots).$$

Вычисления производить пока  $|Y_i - Y_{i-1}|$  не станет меньше EPS. Определите количество итераций, за которое достигается эта точность.

**9.** Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} \left(1 - \frac{n}{1+n^3}\right) \times \frac{n}{1+n^3}, \quad \alpha = 0,001$$

**10.** Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} \left(1 - \frac{1}{3n^2}\right), \quad \alpha = 0,0001$$

**11.** Вычислите и выведите на экран значения функции  $y = \frac{1}{x}$ , превосходящие  $\varepsilon$  для  $x$ , принимающего значения 1, 2, ... .

Значение  $\varepsilon$  введите с клавиатуры.

**12.** Введите натуральное число  $n$ . Определите количество цифр в этом числе.

**13.** Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} \left(1 - \frac{1}{n+1}\right) \times \frac{1}{n}, \quad \alpha = 0,0001$$

**14.** Вычислите значение функции с помощью ряда Маклорена с заданной точностью  $S(x)$  и с помощью стандартных функций Delphi  $Y(x)$ :

$$S(x) = x - \frac{x^3}{3} + \dots + (-1)^n \frac{x^{2n+1}}{2n+1} + \dots; \quad Y(x) = \arctg x$$

**15.** Вычислите значение числа  $\pi$  с заданной точностью, используя формулу

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots \quad \text{Выполните вычисления.}$$

**16.** Вычислите значение числа  $\pi$  с заданной точностью, используя формулу

$$\frac{\pi}{8} = \frac{1}{1 \cdot 3} + \frac{1}{5 \cdot 7} + \frac{1}{9 \cdot 11} + \dots$$

**17.** Вычислите значение функции с помощью ряда Маклорена с заданной точностью  $S(x)$  и с помощью стандартных функций Delphi  $Y(x)$ :

$$S(x) = 1 + \frac{2x}{1!} + \dots + \frac{(2x)^n}{n!} + \dots; \quad Y(x) = e^{2x}$$

**18.** Вычислите значение функции с помощью ряда Маклорена с заданной точностью  $S(x)$  и с помощью стандартных функций Delphi  $Y(x)$ :

$$S(x) = \frac{x^3}{3} - \frac{x^5}{15} + \dots + (-1)^{n+1} \frac{x^{2n+1}}{4n^2 - 1} + \dots;$$

$$Y(x) = \frac{1+x^2}{2} \arctg x - \frac{x}{2}$$

**19** Вычислите значение функции с помощью ряда Маклорена с заданной точностью  $S(x)$  и с помощью стандартных функций Delphi  $Y(x)$ :

$$S(x) = 1 - \frac{x^2}{2!} + \dots + (-1)^n \frac{x^{2n}}{(2n)!} + \dots; \quad Y(x) = \cos x$$

**20.** Вычислите значение квадратного корня  $y = \sqrt{x}$  с точностью EPS с использованием итерационной формулы Ньютона:

$$Y_0 = 1$$

$$Y_i = \frac{1}{2} (Y_{i-1} + X/Y_{i-1}) \quad (i = 1, 2, 3, \dots).$$

Вычисления производить пока  $|Y_i - Y_{i-1}|$  не станет меньше EPS. Определите количество итераций, за которое достигается эта точность.

**21.** Найдите сумму членов ряда  $S = 1 + 1/2 + 1/4 + 1/8 + \dots$

Сумму вычислять, пока очередной член ряда не станет меньше EPS.

**22.** Вычислите значение кубического корня  $y = \sqrt[3]{x}$  с точностью EPS с использованием итерационной формулы Ньютона:

$$Y_0 = 1$$

$$Y_i = 1/3 (2Y_{i-1} + X/Y_{i-1}^2) \quad (i = 1, 2, 3, \dots).$$

Вычисления производить пока  $|Y_i - Y_{i-1}|$  не станет меньше EPS. Определите количество итераций, за которое достигается эта точность.

**23.** Алгоритм Евклида нахождения НОД( $m, n$ ) основан на следующих свойствах этой величины: пусть  $m$  и  $n$  – два натуральных числа и пусть  $m \geq n$ . Тогда для чисел  $m, n$  и  $r$ , где  $r$  – остаток от деления  $m$  на  $n$ , выполняется равенство  $\text{НОД}(m, n) = \text{НОД}(n, r)$ . Используя алгоритм Евклида, найдите наибольший общий делитель  $m$  и  $n$ .

**24.** Значение функции  $LN(1 + X)$  можно вычислить с помощью разложения ее в ряд Маклорена

$$LN(1 + X) = X - X^2/2 + X^3/3 - X^4/4 + \dots$$

Вычислите  $LN(1 + X)$  с точностью EPS, т.е., вычисление суммы ряда нужно продолжать до тех пор, пока абсолютная величина очередного члена ряда не станет меньше EPS. Определите количество членов ряда, которое для этого понадобилось.

**25.** Значение функции  $y(x) = (e^x + e^{-x})/2$  можно вычислить с помощью разложения ее в ряд Маклорена

$$y(x) = \sum_{n=0}^{\infty} \frac{x^{2n}}{(2n)!}.$$

Вычислите  $y(x)$  с точностью EPS, т.е., вычисление суммы ряда нужно продолжать до тех пор, пока абсолютная величина очередного члена ряда не станет меньше EPS. Определите количество членов ряда, которое для этого понадобилось.

**26.** Значение функции  $y(x) = \cos(x)$  можно вычислить с помощью разложения ее в ряд Маклорена

$$y(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}.$$

Вычислите  $y(x)$  с точностью EPS, т.е., вычисление суммы ряда нужно продолжать до тех пор, пока абсолютная величина очередного члена ряда не станет меньше EPS. Определите количество членов ряда, которое для этого понадобилось.

**27.** Значение функции  $y(x) = (e^x - e^{-x})/2$  можно вычислить с помощью разложения ее в ряд Маклорена

$$y(x) = \sum_{n=0}^{\infty} \frac{x^{2n+1}}{(2n+1)!}.$$

Вычислите  $y(x)$  с точностью EPS, т.е., вычисление суммы ряда нужно продолжать до тех пор, пока абсолютная величина очередного члена ряда не станет меньше EPS. Определите количество членов ряда, которое для этого понадобилось.

**28.** Значение функции  $y(x)=\arctg(x)$  можно вычислить с помощью разложения ее в ряд Маклорена

$$y(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{2n+1}.$$

Вычислите  $y(x)$  с точностью EPS, т.е., вычисление суммы ряда нужно продолжать до тех пор, пока абсолютная величина очередного члена ряда не станет меньше EPS. Определите количество членов ряда, которое для этого понадобилось.

**29.** Значение функции  $y(x)=2(\cos^2(x)-1)$  можно вычислить с помощью разложения ее в ряд Маклорена

$$y(x) = \sum_{n=0}^{\infty} (-1)^n \frac{(2x)^{2n}}{(2n)!}.$$

Вычислите  $y(x)$  с точностью EPS, т.е., вычисление суммы ряда нужно продолжать до тех пор, пока абсолютная величина очередного члена ряда не станет меньше EPS. Определите количество членов ряда, которое для этого понадобилось.

**30.** Значение функции  $y(x)=\sin(x)/x$  можно вычислить с помощью разложения ее в ряд Маклорена

$$y(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n+1)!}.$$

Вычислите  $y(x)$  с точностью EPS, т.е., вычисление суммы ряда нужно продолжать до тех пор, пока абсолютная величина очередного члена ряда не станет меньше EPS. Определите количество членов ряда, которое для этого понадобилось.

**31.** Значение функции  $y(x)=x\arctg(x)-\ln\sqrt{1+x^2}$  можно вычислить с помощью разложения ее в ряд Маклорена

$$y(x) = \sum_{n=0}^{\infty} (-1)^{n+1} \frac{x^{2n}}{2n(2n+1)}.$$

Вычислите  $y(x)$  с точностью EPS, т.е., вычисление суммы ряда нужно продолжать до тех пор, пока абсолютная величина очередного члена ряда не станет меньше EPS. Определите количество членов ряда, которое для этого понадобилось.

**32.** Даны два натуральных числа  $m$  и  $n$  ( $1 < m < n$ ). Найдите наименьшее  $k$ , при котором  $m^k > n$ .

**33.** Найдите приближенно с точностью до 0,01 наибольшее значение функции  $y = \frac{ax^2 + bx + c}{dx + e}$  на отрезке  $[x_1; x_2]$ . Значения  $a, b, c, d, e, x_1, x_2$  введите с клавиатуры.

## Задачи второго уровня

1. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} (-1)^n \times \frac{n}{1+n^3}, \quad \alpha = 0,001$$

2. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{1}{3n^2}, \quad \alpha = 0,0001$$

3. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^{n+1} \times \frac{1}{n}, \quad \alpha = 0,0001$$

4. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^{n-1} \times \frac{1}{(2n)^3}, \quad \alpha = 0,001$$

5. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} (-1)^n \times \frac{1}{n(2n+1)}, \quad \alpha = 0,001$$

6. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{1}{(2n+1)}, \quad \alpha = 0,0001$$

7. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{n}{2^n}, \quad \alpha = 0,001$$

8. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{n^2}{3^n}, \quad \alpha = 0,001$$

9. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{n}{(2n-1)^2(2n+1)^3}, \quad \alpha = 0,001$$

10. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{1}{(2n-1)n}, \quad \alpha = 0,0001$$

11. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} (-1)^{n+1} \times \left(\frac{-2}{3}\right)^n, \quad \alpha = 0,001$$

12. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{n}{7^n}, \quad \alpha = 0,0001$$

13. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^{n+1} \times \left(\frac{-2}{3}\right)^{n+1}, \quad \alpha = 0,01$$

14. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{1}{2n}, \quad \alpha = 0,001$$

15. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} (-1)^n \times \frac{1}{3n+1}, \quad \alpha = 0,01$$

16. Вычислите сумму ряда с заданной степенью точности

$$\alpha: \sum_{n=1}^{\infty} (-1)^n \times \frac{1}{(2n)^2}, \quad \alpha = 0,00001$$

17. Вычислите сумму ряда с заданной степенью точности

$$\alpha: \sum_{n=1}^{\infty} (-1)^n \times \frac{(2n+1)}{2n^2}, \quad \alpha = 0,001$$

18. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{1}{2^n \times n}, \quad \alpha = 0,001$$

19. Вычислите сумму ряда с заданной степенью точности

$$\alpha: \sum_{n=0}^{\infty} (-1)^n \times \frac{1}{(3^n + 1) \times n}, \quad \alpha = 0,001$$

20. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{1}{2n^3}, \quad \alpha = 0,0001$$

21. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} \frac{\cos \pi n}{3^n(n+1)}, \quad \alpha = 0,001$$

22. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} (-1)^n \times \frac{\cos \pi n}{4^n(2n+1)}, \quad \alpha = 0,001$$

23. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} \frac{\sin(\pi/2 + \pi n)}{n^3}, \quad \alpha = 0,01$$

24. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} (-1)^n \times \frac{2^n}{(n+1)^n}, \quad \alpha = 0,001$$

25. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} (-1)^n \times \frac{1}{(n+1)^n}, \quad \alpha = 0,001$$

26. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} \frac{\sin(\pi/2 + \pi n)}{n^3 + 1}, \quad \alpha = 0,01$$

27. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} (-1)^n \times \frac{1}{n^3(n+3)}, \quad \alpha = 0,01$$

28. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} \frac{\cos \pi n}{(n^3 + 1)^2}, \quad \alpha = 0,001$$

29. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=0}^{\infty} (-1)^n \times \frac{1}{1+n^3}, \quad \alpha = 0,001$$

30. Вычислите сумму ряда с заданной степенью точности  $\alpha$ :

$$\sum_{n=1}^{\infty} (-1)^n \times \frac{2n+1}{n^3(n+1)}, \quad \alpha = 0,01$$

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

Очень часто заранее невозможно сказать, сколько раз надо выполнить какую-то операцию, но можно определить условие, при котором она должна заканчиваться. Такое задание на русском языке может выглядеть так: делай эту работу до тех пор, **пока** она не будет закончена (или бревно, пока оно не будет распилено; иди вперед, пока не дойдешь до двери). Слово «пока» на английском языке записывается как `while`, и так же называется еще один вид цикла.

**Пример 4.1** Ввести натуральное число и определить, сколько в нем цифр.

Для решения этой задачи обычно применяется такой алгоритм. Число делится на 10 и отбрасывается остаток, и так до тех пор, пока результат деления не равен нулю. С помощью специальной переменной (она называется *счетчиком*) считаем, сколько раз выполнялось деление – столько цифр было в числе. Понятно, что нельзя заранее определить, сколько раз надо разделить число, поэтому надо использовать цикл с условием.

```

#include <stdio.h>
#include <conio.h>
main()
{
int N;           // число, с которым работаем
int count=0;    // переменная-счетчик

printf ( "\nВведите число N: " ); // подсказка
scanf ( "%d", &N );           // ввод N с клавиатуры

while ( N > 0 ) // заголовок цикла «пока N > 0»
{
    // начало цикла (открывающая скобка)
    N /= 10;          // отсекаем последнюю цифру
    count++;          // увеличиваем счетчик цифр
}                   // конец цикла (закрывающая скобка)

printf ( "В этом числе %d цифр\n", count );
getch();
}

```

тело цикла

Цикл **while** используется тогда, когда количество повторений цикла заранее неизвестно и не может быть вычислено.

Цикл **while** состоит из заголовка и тела цикла.

В заголовке после слова **while** в круглых скобках записывается условие, при котором цикл продолжает выполняться. Когда это условие нарушается (становится ложно), цикл заканчивается.

В условии можно использовать знаки логических отношений и операций:

- ✓ > больше;
- ✓ < меньше;
- ✓ >= больше или равно;
- ✓ <= меньше или равно;
- ✓ == равно;
- ✓ != не равно.

Если условие неверно в самом начале, то цикл не выполняется ни разу (это цикл с *предусловием*).

Если условие никогда не становится *ложным* (неверным), то цикл никогда не заканчивается; в таком случае говорят, что программа «зациклилась» — это серьезная логическая ошибка.

В языке Си любое число, не равное нулю, обозначает истинное условие, а ноль — ложное условие:

while ( 1 ){ ... } // бесконечный цикл
while ( 0 ){ ... } // цикл не выполнится ни разу

Тело цикла заключается в фигурные скобки; если в теле цикла стоит всего один оператор, скобки можно не ставить.

В тело цикла могут входить любые другие операторы, в том числе и другие циклы (такой прием называется «вложенные циклы»).

Для того, чтобы легче разобраться в программе, все тело цикла и ограничивающие его скобки сдвигаются вправо на 2-3 символа (запись «лесенкой»).

Существуют также случаи, когда надо выполнить цикл хотя бы один раз, затем на каждом шагу делать проверку некоторого условия и закончить цикл, когда это условие станет ложным. Для этого используется **цикл с постусловием** (то есть условие проверяется не в начале, а в конце цикла). Не рекомендуется применять его слишком часто, поскольку он напоминает такую ситуацию: прыгнул в бассейн, и только потом посмотрел, есть ли в нем вода. Рассмотрим случай, когда его использование оправдано.

**Пример 4.2** Ввести натуральное число и найти сумму его цифр. Организовать ввод числа так, чтобы нельзя было ввести отрицательное число или ноль.

Любая программа должна обеспечивать защиту от неверного ввода данных (иногда такую защиту называют «защитой от дурака» — *fool proof*). Поскольку пользователь может вводить данные неверно сколько угодно раз, то надо использовать цикл с условием. С другой стороны, один раз обязательно надо ввести число, поэтому нужен цикл с постусловием.

В отличие от предыдущей программы, теперь надо при каждом делении определять остаток (последняя цифра числа равна остатку от деления его на 10) и суммировать все остатки в специальной переменной.

```
#include <stdio.h>
#include <conio.h>
main()
{
    int N, sum; // sum - сумма цифр числа
    sum = 0; // сначала сумму обнуляем
    do { // начало цикла
        printf ( "\nВведите натуральное число: " );
        scanf ( "%d", &N );
    }
    while ( N <= 0 ); // условие цикла «пока N <= 0»
    while ( N > 0 ) {
        sum += N % 10;
        N /= 10;
    }
    printf ( "Сумма цифр этого числа равна %d\n", sum );
    getch();
}
```

Цикл **do—while** используется тогда, когда количество повторений цикла заранее неизвестно и не может быть вычислено.

Цикл состоит из заголовка **do**, тела цикла и завершающего условия.

Условие записывается в круглых скобках после слова **while**, цикл продолжает выполняться, пока условие верно; когда условие становится неверно, цикл заканчивается.

Условие проверяется только в конце очередного шага цикла (это **цикл с постусловием**), таким образом, **цикл всегда выполняется хотя бы один раз**.

Если условие никогда не становится ложным (неверным), то цикл никогда не заканчивается; в таком случае говорят, что программа «зациклилась» — это серьезная логическая ошибка.

Тело цикла заключается в фигурные скобки; если в теле цикла стоит всего один оператор, скобки можно не ставить.

В тело цикла могут входить любые другие операторы, в том числе и другие циклы (такой прием называется «вложенные циклы»).

Для того, чтобы легче разобраться в программе, все тело цикла и ограничивающие его скобки сдвигаются вправо на 2-3 символа (запись «лесенкой»).

Иногда надо выйти из цикла и перейти к следующему оператору, не дожидаясь окончания очередного шага цикла. Для этого используют специальный оператор **break**. Можно также сказать компьютеру, что надо завершить текущий шаг цикла и сразу перейти к новому шагу (не выходя из цикла) — для этого применяют оператор **continue**.

**Пример 4.3** Написать программу, которая вычисляет частное и остаток от деления двух введенных целых чисел. Программа должна работать в цикле, то есть запрашивать значения делимого и делителя, выводить результат, снова запрашивать данные и т.д. Если оба числа равны нулю, надо выйти из цикла и завершить работу программы. Предусмотреть сообщение об ошибке в том случае, если второе число равно нулю, а первое — нет.

Особенность этой задачи состоит в том, что при входе в цикл мы не можем определить, надо ли будет выполнить до конца очередной шаг. Необходимая информация поступает лишь при вводе данных с клавиатуры. Поэтому здесь используется бесконечный цикл **while(1) { . . . }** (напомним, что в языке Си единица считается истинным условием). Выйти из такого цикла можно только с помощью специального оператора **break**.

В то же время, если второе число равно нулю, то оставшуюся часть цикла не надо выполнять. Для этого служит оператор **continue**.

```
#include <stdio.h>
#include <conio.h>
main()
{
    int A, B;
    while ( 1 ) // бесконечный цикл, выход только по break!
    {
        printf ( "\nВведите делимое и делитель:" );
        scanf ( "%d%d", &A, &B );
        if ( A == 0 && B == 0 ) break; // выход из цикла
        if ( B == 0 ) {
            printf ( "Деление на ноль" );
            continue; // досрочный переход к новому шагу цикла
        }
    }
}
```

```

    printf ( "Частное %d, остаток %d", A/B, A%B );
}
getch();
}

```

Если только внутри цикла можно определить, надо ли делать вычисления в теле цикла и надо ли продолжать цикл (например, при вводе исходных данных), часто используют бесконечный цикл, внутри которого стоит оператор выхода **break**:

```

while ( 1 ) {
...
if ( надо выйти ) break;
...
}

```

С помощью оператора **break** можно досрочно выходить из любых циклов: **for**, **while**, **do-while**.

Чтобы досрочно завершить текущий шаг цикла и сразу перейти к следующему шагу, используют оператор **continue**.

Рассмотрим более сложную задачу.

**Пример 4.4** Найти сумму всех элементов последовательности, которые по модулю не меньше, чем 0,001:

$$S = \frac{1}{2} - \frac{2}{4} + \frac{3}{8} - \frac{4}{16} + \dots$$

Эта задача имеет решение только тогда, когда элементы последовательности убывают по модулю и стремятся к нулю. Поскольку мы не знаем, сколько элементов войдет в сумму, надо использовать цикл **while** (или **do-while**). Один из вариантов решения показан ниже.

```

#include <stdio.h>
main()
{
float S, z, c, d, a;
S = 0; z = 1; c = 1; d = 2; // начальные значения
a = 0.5; // первый элемент последовательности

while ( a >= 0.001 )
{
    S = S + z*a; // добавить элемент к сумме
    z = - z; // изменить переменные z, c, d
    c++;
    d = d * 2;
    a = c / d; // модуль следующего элемента
}
printf("Сумма S = %f", S);
}

```

Цикл закончится тогда, когда переменная **a** (она обозначает модуль очередного элемента последовательности) станет меньше 0,001. Чтобы программа вошла в цикл на первом шаге, в эту переменную надо записать любое значение, большее, чем 0,001.

Очевидно, что если переменная **a** не будет уменьшаться, то условие в заголовке цикла всегда будет истинно и программа «зациклится».

## **5. Порядок выполнения работы**

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_  
\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Для чего предназначен оператор цикла?
2. Какие виды циклов есть в Си?
3. Какой формат записи имеет оператор **while**? Как он работает? В каких случаях применяется?
4. Чем отличается оператор **while** от оператора **do -while**?
5. Что является телом цикла?
6. Как в теле цикла выполнить несколько операторов?
7. Почему перед выполнением цикла некоторым переменным нужно задавать начальные значения?
8. Что такое зацикливание? Как его избегать?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.

5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ обработки массивов»

Минск  
2017

## Лабораторная работа № 5

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ обработки массивов»

### 1. Цель работы

Научить применять массивы при решении задач различных видов и уровней сложности

### 2. Задание

Задачи первого и второго уровня обязательны для всех. Номер варианта соответствует вашему номеру по списку. Обязательно реализовать вывод исходного и конечного массивов.

*Примечание!!! В Си нумерация элементов идет с 0, в примерах с 1. Сделайте соответствующие поправки.*

#### Задачи первого уровня

1. Вычислите  $A_0 + A_1x + A_2x^2 + A_3x^3 + \dots + A_Nx^N$ , используя схему Горнера. В соответствии со схемой Горнера данный многочлен преобразуется к виду:  $\dots((A_N x + A_{N-1})x + A_{N-2})x + \dots + A_1)x + A_0$ .
2. Вычислите  $A_1x + A_2x^2 + A_3x^3 + \dots + A_Nx^N$ , не используя схему Горнера.
3. Вычислите  $A_1 + A_3 + A_5 + \dots + A_{2N-1}$ .
4. Вычислите  $A_1 - A_2 + A_3 - \dots + (-1)^{N-1}A_N$ .
5. Вычислите  $A_1A_2A_3 \dots A_N$ .
6. Вычислите  $A_1 + 2A_2 + 3A_3 + \dots + NA_N$ .
7. Вычислите  $-\frac{A_1}{1!} + \frac{A_2}{2!} - \dots + (-1)^N \frac{A_N}{N!}$ .
8. Вычислите  $(|A_1| - A_1) + \dots + (|A_N| - A_N)$ .
9. Цилиндр объемом единица имеет высоту  $h$ . Определите радиус основания цилиндра для значений  $h$ , равных 0,5, 1, 1,5, ..., 5.
10. «Переверните» последовательность  $A_1, A_2, A_3, \dots, A_N$ , т.е. поменяйте местами  $A_1$  с  $A_N$ ,  $A_2$  с  $A_{N-1}$  и т.д.
11. Получите таблицу температур от 0 до 100 °C и их эквивалентов по шкале Фаренгейта, используя для перевода формулу:  
$$tF = 1,8tC + 32$$
12. Вычислите таблицу значений функции  $y = ax^2 + bx + c$  для значений  $x$ , изменяющихся от  $x_0$  до  $x_n$ , с шагом  $h$  ( $x_0 < x_n$ ).
13. Значения  $C_1, \dots, C_N$  являются емкостями  $N$ -конденсаторов. Определите емкости систем конденсаторов, которые получаются последовательным и параллельным соединением исходных конденсаторов.
14. Найдите периметр  $N$ -угольника, заданного координатами вершин на плоскости  $\{(X_i; Y_i)\}, (i = 1, \dots, N)$ .
15. Даны координаты  $\{(x_i; y_i)\}, (i = 1, \dots, n)$   $n$  заводов потребителей сырья и координаты места добычи сырья  $(x_C; y_C)$ . Найдите расстояния от места

добычи сырья до каждого завода, а также среднее арифметическое этих расстояний.

16. Дан массив  $A$ , состоящий из  $n$  элементов. Сформируйте «сглаженный» массив, заменив в исходном все элементы, кроме крайних, по формуле

$$A_i = \frac{A_{i-1} + A_i + A_{i+1}}{3}, \quad i = 2, 3, \dots, n-1.$$

При сглаживании используются лишь старые значения элементов массива.

17. Дан массив  $a$ , состоящий из  $n$  элементов. Вычислите  $a_1, a_1 + a_2, a_1 + a_2 + a_3, \dots, a_1 + a_2 + a_3 + \dots + a_n$ .

18. Дан массив  $a$ , состоящий из  $n$  элементов. Получите  $a_1, -a_2, a_3, \dots, (-1)^{n-1} a_n$ .

19. Дан массив  $a$ , состоящий из  $n$  элементов. Получите массив  $b$ , где  $b_k = a_k + k!$ .

20. Дан массив  $a$ , состоящий из  $n$  элементов. Получите массив  $b$ , где  $b_k = 2a_k + k$ .

21. Даны массивы  $a$  и  $b$ , состоящие из  $n$  элементов каждый. Получите массив  $c$ , где  $c_k = a_k + b_k$ .

22. Даны массивы  $a$  и  $b$ , состоящие из  $n$  элементов каждый. Получите массив  $c$ , где  $c_k = a_k \cdot b_k$ .

23. Дан массив  $a$ , состоящий из  $n$  элементов. Найдите сумму элементов массива, стоящих на нечетных местах.

24. Дан массив  $a$ , состоящий из  $n$  элементов, и число  $x$ . Вычислите значение полинома  $P = x^n + a_1x^{n-1} + \dots + a_{n-1}x + a_n$ .

25. Дан массив  $a$ , состоящий из  $n$  элементов. Получите новый массив, поменяв элементы, стоящие на четных местах, с элементами, стоящими на нечетных местах, т.е.  $a_1$  с  $a_2, a_3$  с  $a_4, a_5$  с  $a_6$  и т.д.

26. Даны массивы  $a$  и  $b$ , состоящие из  $n$  элементов каждый. Получите новые массивы  $a$  и  $b$ , элементы которых вычисляются по правилу:  $a_i = a_i + b_i, b_i = a_i - b_i$ .

27. Даны массивы  $a$  и  $b$ , состоящие из  $n$  элементов каждый. Получите новые массивы  $a$  и  $b$ , элементы которых вычисляются по правилу:  $a_i = b_i, b_i = -a_i$ .

28. Дан массив  $a$ , состоящий из  $n$  элементов. Найдите среднее арифметическое значение элементов массива, стоящих на четных местах.

29. Дан массив  $a$ , состоящий из  $n$  элементов. Найдите сумму  $a_1 + 2a_2 + 3a_3 + \dots + na_n$ .

30. Дан массив  $a$ , состоящий из  $n$  элементов. Найдите сумму  $a_1 + 2!a_2 + 3!a_3 + \dots + n!a_n$ .

## Задачи второго уровня

1. Проверьте, является ли данная числовая последовательность  $a_1, a_2, \dots, a_n$  возрастающей.
2. Информация о температуре воздуха за месяц задана в виде массива. Определите, сколько раз температура опускалась ниже  $0^{\circ}\text{C}$ . Число дней конкретного месяца введите с клавиатуры.
3. Информация о среднесуточной температуре воздуха за месяц задана в виде массива. Определите, сколько дней температура была ниже среднесуточной.
4. Дан числовой массив  $A$ , состоящий из  $n$  элементов. Найдите среднее арифметическое положительных элементов этого массива.
5. Дан числовой массив  $A$ , состоящий из  $n$  элементов. Все положительные элементы этого массива уменьшите на 0,5.
6. Дан числовой массив  $A$ , состоящий из  $n$  натуральных чисел. Определите количество элементов массива, которые больше заданного числа.
7. Дан числовой массив  $A$ , состоящий из  $n$  натуральных чисел. Определите количество элементов массива, являющихся нечетными числами.
8. Дан числовой массив  $A$ , состоящий из  $n$  натуральных чисел. Определите количество элементов массива, являющихся кратными 7.
9. Дан числовой массив  $A$ , состоящий из  $n$  натуральных чисел. Определите количество элементов массива кратных 3, но не кратных 5.
10. Дан числовой массив  $A$ , состоящий из  $n$  натуральных чисел. Определите количество элементов массива, удовлетворяющих условию  $A_i < (A_{i-1} + A_{i+1})/2$ .
11. Дан числовой массив  $A$ , состоящий из  $n$  натуральных чисел. Определите количество элементов массива, удовлетворяющих условию  $2k < A_k < 3k$ .
12. Дан числовой массив  $A$ , состоящий из  $n$  натуральных чисел. Определите количество элементов массива, имеющих четные порядковые номера и являющихся нечетными числами.
13. Дан числовой массив  $A$ , состоящий из  $n$  натуральных чисел. Определите количество элементов массива, которые при делении на 7 дают остаток 1, 2 или 5.
14. Имеется  $n$  итоговых оценок студента. Определите, является ли студент отличником (все оценки не ниже 9).
15. Имеется  $n$  итоговых оценок студента. Определите, является ли студент неуспевающим (имеются оценки ниже 4).
16. Имеется  $n$  итоговых оценок студента. Определите количество шестерок, семерок, восьмерок, девяток и десяток.
17. Дана последовательность  $x_1, x_2, \dots, x_n$ , упорядоченная в порядке возрастания, и вещественное  $y$ . Найдите такое  $k$ , что  $x_k < y \leq x_{k+1}$ .
18. Имеется  $n$  итоговых оценок студента. Расположите эти оценки в следующем порядке: десятки, восьмерки, шестерки. Остальные оценки в

произвольном порядке.

19. Данна последовательность  $x_1, x_2, \dots, x_n$ . Определите количество таких троек, что  $x_{i-1} < x_i < x_{i+1}$ , ( $i = 2, n - 1$ ).
20. Данна последовательность  $x_1, x_2, \dots, x_n$ . Найдите номер элемента, который отличается от среднего арифметического значения элементов последовательности на минимальную величину.
21. Данна последовательность  $x_1, x_2, \dots, x_n$ . Найдите наибольшую сумму подряд идущих элементов.
22. Данна последовательность  $x_1, x_2, \dots, x_n$ . Определите количество элементов последовательности, больших среднего арифметического значения положительных элементов последовательности.
23. Данна последовательность натуральных чисел  $x_1, x_2, \dots, x_n$ . Измените данную последовательность так, чтобы в начале стояли все четные, а затем – нечетные элементы последовательности.
24. Данна последовательность целых чисел  $x_1, x_2, \dots, x_n$ . Измените данную последовательность так, чтобы в начале стояли все нулевые элементы, затем – отрицательные, а затем – положительные элементы последовательности.
25. Даны две последовательности чисел  $a_1, a_2, \dots, a_n$  и  $b_1, b_2, \dots, b_n$ . Определите, какое число раз встречается ситуация, когда  $a_i > b_i$  и  $a_{i+1} < b_{i+1}$ , ( $i = 1, n - 1$ ).
26. Даны две последовательности чисел  $a_1, a_2, \dots, a_n$  и  $b_1, b_2, \dots, b_n$ . Найдите  $i$ , при котором  $a_i + b_i$  – наибольшая из всех таких пар.
27. Даны две последовательности чисел  $a_1, a_2, \dots, a_n$  и  $b_1, b_2, \dots, b_n$ . Найдите количество пар  $a_i, b_i$ , ( $i = 1, n$ ) таких, что оба числа в паре четные.
28. Данна последовательность  $x_1, x_2, \dots, x_n$ . Найдите наибольший по модулю элемент последовательности с указанием его номера.
29. Данна последовательность  $x_1, x_2, \dots, x_n$ . Поменяйте местами самый большой элемент с самым маленьким.
30. Данна последовательность  $x_1, x_2, \dots, x_n$ . Найдите самый большой по модулю отрицательный элемент.

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Builder C++**.

### 4. Основные теоретические сведения

Основное предназначение компьютеров не вычисления, как считают многие, а **обработка больших объемов данных**. При размещении большого количества данных в памяти возникает такая проблема: надо научиться обращаться к каждой ячейке с данными отдельно. При этом очень сложно дать каждой ячейке собственное имя и при этом не запутаться. Выкручиваются из этой ситуации так: дают имя не ячейке, а группе ячеек, в которой каждая ячейка имеет номер. Такая область памяти называется массивом.

**Массив** – это группа ячеек памяти одинакового типа, расположенных рядом и имеющих общее имя. Каждая ячейка в группе имеет уникальный номер.

При работе с массивами надо научиться решать три задачи:

- ✓ выделять память нужного размера под массив;
- ✓ записывать данные в нужную ячейку;
- ✓ читать данные из ячейки.

Чтобы использовать массив, надо его объявить – выделить место в памяти. Типом массива называется тип массива это тип входящих в него элементов. Массивы могут быть разных типов — **int**, **float**, **char**, и т.д. Массив объявляют так же, как и обычные переменные, но после имени массива в квадратных скобках записывается его размер.

```
int A[10], B[20]; // 2 массива на 10 и 20 целых чисел  
float C[12]; // массив из 12 вещественных чисел
```

При объявлении массива можно сразу заполнить его начальными значениями, перечисляя их внутри фигурных скобок:

```
int A[4] = { 2, 3, 12, 76 };
```

Если в списке в фигурных скобках записано меньше чисел, чем элементов в массиве, то оставшиеся элементы заполняются нулями. Если чисел больше, чем надо, транслятор сообщает об ошибке. Например,

```
int A[4] = { 2 }; // последние три элементы равны 0
```

Для повышения универсальности программы размер массива лучше определять через константу. В этом случае для переделки программы для массива другого размера надо только поменять значение этой константы:

```
const int N = 20; // константа  
main()  
{  
    int A[N]; // размер массива задан через константу  
    ...  
}
```

В таблице показаны примеры правильного и неправильного объявления массива.

правильно		неправильно	
<b>int A[20];</b>	размер массива указан явно	<b>int A[];</b>	размер массива неизвестен
<b>const int N = 20;</b> <b>int A[N];</b>	размер массива – постоянная величина	<b>int N = 20;</b> <b>int A[N];</b>	размер массива не может быть переменной

Каждый элемент массива имеет свой порядковый номер. Чтобы обратиться к элементу массива, надо написать имя массива и затем в квадратных скобках номер нужного элемента. Важно запомнить одно важное правило:

Элементы массивов в языке Си нумеруются с нуля. Таким образом, если в массиве 10 элементов, он содержит элементы:

**A[0], A[1], A[2], ..., A[9]**

Номер элемента массива также называется его **индексом**. Вот примеры обращения к массиву **A**:

```
x = (A[3] + 5)*A[1]; // прочитать значения A[3] и A[1]
A[0] = x + 6; // записать новое значение в A[0]
```

В языке Си не контролируется **выход за границы массива**, то есть формально вы можете записать что-то в элемент с несуществующим индексом, например в **A[345]** или в **A[-12]**. Однако при этом вы стираете какую-то ячейку в памяти, не относящуюся к массиву, поэтому последствия такого шага непредсказуемы и во многих случаях программа «зависает».

Как же ввести данные в массив? Существует много способов в зависимости от вашей задачи:

- ✓ элементы массива вводятся с клавиатуры вручную;
- ✓ массив заполняется случайными числами (например, для моделирования случайных процессов);
- ✓ элементы массива читаются из файла;
- ✓ элементы массива поступают через порт с внешнего устройства (например, сканера, модема и т.п.);
- ✓ массив заполняется в процессе вычислений.

**Пример 5.1** Ввести с клавиатуры массив из 10 элементов, умножить все элементы на 2 и вывести полученный массив на экран.

К сожалению, невозможно просто сказать компьютеру: «введи массив **A**». Мы должны каждый элемент прочитать отдельно.

Чтобы ввести массив в память, надо каждый его элемент обработать отдельно (например, вызвав для него функцию ввода **scanf**).

Ввод с клавиатуры применяется в простейших программах, когда объем вводимой информации невелик. Для ввода массива будем использовать цикл **for**. Напомним, что массив надо предварительно *объявить*, то есть выделить под него память.

Вводить можно столько элементов массива, сколько ячеек памяти выделено. Помните, что элементы массива нумеруются с нуля, поэтому если массив имеет всего 10 элементов, то последний элемент имеет номер 9. Если пытаться записывать в 10-ый элемент, произойдет выход за границы массива, и программа может работать неверно (а, возможно, и «зависнет»). При вводе массива желательно выдать на экран общую подсказку для ввода всего массива и подсказки для каждого элемента.

Для умножения элементов массива на 2 надо снова использовать цикл, в котором за один раз обрабатывается 1 элемент массива.

Вывод массива на экран выполняется также в цикле **for**. Элементы выводятся по одному. Если в конце строки-формата в операторе **printf** поставить пробел, то элементы массива будут напечатаны в строчку, а если символ **\n** – то в столбик.

```

#include <stdio.h>
const int N = 10;           // размер массива
main()
{
    int i, A[N];           // объявление массива
    printf("Введите массив A\n"); // подсказка для ввода

    for ( i = 0; i < N; i ++ ) {      // цикл по всем элементам
        printf("Введите A[%d]> ", i ); // подсказка для ввода A[i]
        scanf ("%d", &A[i]);          // ввод A[i]
    }

    for ( i = 0; i < N; i ++ ) // цикл по всем элементам
        A[i] = A[i] * 2;       // умножить A[i] на 2

    printf("\nРезультат:\n");
    for ( i = 0; i < N; i ++ ) // цикл по всем элементам
        printf("%d ", A[i]);   // вывести A[i]
}

```

Этот прием используется для моделирования случайных процессов, например, броуновского движения частиц. Пусть требуется заполнить массив равномерно распределенными случайными числами в интервале **[a,b]**. Поскольку для целых и вещественных чисел способы вычисления случайного числа в заданном интервале отличаются, рассмотрим оба варианта. Здесь и далее предполагается, что в начале программы есть строчка

```
const int N = 10;
```

Описание функции-датчика случайных чисел находится в заголовочном файле **stdlib.h**.

Удобно также добавить в свою программу функцию **random**:

```
int random (int N) { return rand() % N; }
```

которая выдает случайные числа с равномерным распределением в интервале **[0,N-1]**.

Для получения случайных чисел с равномерным распределением в интервале **[a,b]** надо использовать формулу:

```
k = random ( b - a + 1 ) + a;
```

Для вещественных чисел формула несколько другая:

```
x = rand()*(b - a)/RAND_MAX + a;
```

Здесь константа **RAND\_MAX** – это максимальное случайное число, которое выдает стандартная функция **rand**.

**Пример 5.2** Необходимо массив **A** заполнить случайными *целыми* числами в интервале **[-5,10]**, а массив **X** – случайными *вещественными* числами в том же интервале.

```

#include <stdlib.h>
const int N = 10;
main()
{
int i, A[N], a = -5, b = 10;
float X[N];

for ( i = 0; i < N; i ++ )
    A[i] = random(b-a+1) + a;

for ( i = 0; i < N; i ++ )
    X[i] = (float)rand()*(b-a)/RAND_MAX + a;
// здесь что-то делаем с массивами
}

```

Возможно, в этом примере не вполне ясно, зачем перед вызовом функции **rand** поставлено слово (**float**). Это связано с тем, что у нас **a** и **b** – целые числа. Результат функции **rand** – тоже целое число. Здесь возможны две проблемы:

- ✓ при умножении результата функции **rand** на **b-a** может получиться очень большое число, которое не поместится в переменную типа **int**;
- ✓ в языке Си при делении целого числа на целое остаток отбрасывается, поэтому при делении результат будет неверным.

Когда массив заполняется случайными числами, обязательно вывести на экран исходный массив.

**Пример 5.3** Заполнить массив случайными целыми числами в интервале [-10,15], умножить все элементы на 2 и вывести на экран исходный массив и результат.

```

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

const int N = 10;

main()
{
int i, A[N];

for ( i = 0; i < N; i ++ ) // заполнение массива сл. числами
    A[i] = random(26) - 10;

printf("n Исходный массив:\n"); // вывод исходного массива
for ( i = 0; i < N; i ++ )
    printf("%d ", A[i]);

for ( i = 0; i < N; i ++ ) // умножить все элементы на 2
    A[i] = A[i] * 2;

printf("n Результат:\n");
for ( i = 0; i < N; i ++ ) // вывод результата
    printf("%d ", A[i]);
}

```

Во многих задачах требуется последовательно перебрать все элементы массива и найти нужные нам. Мы рассмотрим четыре таких задачи:

- ✓ поиск одного заданного элемента;

- ✓ вывод всех элементов, которые удовлетворяют заданному условию;
- ✓ формирование нового массива из всех отобранных элементов;
- ✓ поиск минимального (максимального) элемента.

Все эти задачи решаются с помощью цикла, в котором перебираются все элементы массива от начального (**0**-ого) до конечного (**N-1**-ого) элемента. Такой поиск называется *линейным*, поскольку все элементы просматриваются последовательно один за другим.

**Пример 5.4** Определить, есть ли в массиве элемент с заданным значением **x**, и, если он есть, найти его номер.

Если нет никакой информации о расположении элементов массива, то применяется линейный поиск, основная идея которого – последовательно просматривать массив, пока не будет обнаружено совпадение или не будет достигнут конец массива. Это реализует следующая простая программа:

```
#include <stdio.h>
const int N = 10;
main()
{
    int i, x, A[N];
    int success = 0;           // переменная-флаг, флаг сброшен
    // здесь нужно ввести массив и X
    for ( i=0; i<N; i++ )
        if ( A[i] == x ) {   // если нашли, то...
            success = 1;      // установить флаг
            break;             // выйти из цикла
        }
    if ( success )
        printf ( "A[%d] = %d", i, A[i] );
    else
        printf ( "Элемент %d не найден.", x );
}
```

Чтобы определить ситуацию, когда элемент не найден, нам надо ввести специальную переменную **success**, которая устанавливается в 1, если элемент найден, и остается равной нулю, если в массиве нет нужного элемента. Такая переменная называется **флагом**, флаг может быть установлен (равен 1) или сброшен (равен нулю).

Для линейного поиска в худшем случае мы имеем **N** сравнений. Понятно, что для ускорения поиска надо сначала как-то упорядочить данные, в этом случае можно сделать поиск эффективным.

**Пример 5.5** Определить, сколько в массиве положительных элементов и вывести их на экран.

Для решения этой задачи вводим **счетчик** – специальную переменную, значение которой будет увеличиваться на единицу, когда мы нашли очередной положительный элемент.

```

#include <stdio.h>
const int N = 10;
main()
{
int i, A[N], count = 0; // count - счетчик положительных элементов
// здесь нужно ввести массив

for ( i = 0; i < N; i ++ ) // цикл по всем элементам массива
    if ( A[i] > 0 ) { // если нашли положительный, ...
        count++;
        printf ("%d ", A[i]); // увеличиваем счетчик и ...
    }
printf ("\n В массиве %d положительных чисел", count);
}

```

**Пример 5.6** Сформировать новый массив **B**, включив в него все положительные элементы исходного массива **A**, и вывести его на экран.

Пусть есть массив **A[N]**. Надо выбрать из него все положительные элементы и записать их в новый массив, который и будет дальше использоваться.

Сначала надо определить, сколько места в памяти надо выделить для массива **B**. В «худшем» случае все элементы в массиве **A** будут положительными и войдут в массив **B**, поэтому массив **B** должен иметь такой же размер, что и массив **A**.

Можно предложить такой способ: просматривать весь массив **A**, и если для очередного элемента **A[i]>0**, его значение копируется в **B[i]**.



Однако в этом случае использовать такой массив **B** очень сложно, потому что нужные элементы стоят не подряд.

Есть более красивый способ. Объявляем временную переменную-счетчик **count**, в которой будем хранить количество найденных положительных элементов. Сначала она равна нулю. Если нашли очередной положительный элемент, то ставим его в ячейку **B[count]** и увеличиваем счетчик. Таким образом, все нужные элементы стоят в начале массива **B**.



```

#include <stdio.h>
const int N = 10;
main()
{
    int i, A[N], B[N], count = 0;
    // здесь нужно ввести массив A

    for ( i = 0; i < N; i ++ )
        if ( A[i] > 0 ) {
            B[count] = A[i];
            count++;
        }

    printf("\n Результат:\n");
    for ( i=0; i<count; i++ )
        printf("%d ", B[i]);
}

```

Можно сделать и так:  
`if ( A[i] > 0 )  
 B[count++] = A[i];`

Обратите внимание, что переменная в последнем цикле изменяется от **0** до **count-1** (а не до **N-1**) так что на экран выводятся только реально используемые (а не все) элементы массива **B**.

**Пример 5.7** Найти и вывести на экран минимальный элемент в массиве **A**.

Для решения задачи надо выделить в памяти ячейку (переменную) для хранения найденного минимального значения. Сначала мы записываем в эту ячейку первый элемент (**A[0]**). Затем берем следующий элемент и сравниваем его с минимальным . Если он меньше минимального, записываем его значение в ячейку минимального элемента. И так далее. Когда мы рассмотрим последний элемент в массиве, в дополнительной ячейке памяти будет минимальное значение из всех элементов массива.

Заметим, что перебор в цикле начинается с элемента с номером 1 (а не 0), поскольку начальный элемент мы рассмотрели отдельно.

```

#include <stdio.h>
const int N = 10;
main()
{
    int i, A[N],min;
    // здесь нужно ввести массив A

    min = A[0]; // предполагаем, что A[0] - минимальный
    for ( i=1; i<N; i++ ) // цикл по всем элементам с A[1] до A[N-1]
        if ( A[i] < min ) // если A[i] меньше min, ...
            min = A[i]; // запомнить A[i] как минимальный

    printf("\n Минимальный элемент %d", min);
}

```

Чтобы найти максимальный элемент, достаточно изменить условие в заголовке условного оператора на обратное (**A[i] > min**). Конечно, вспомогательную переменную в этом случае лучше (но не обязательно!) назвать **max**.

Теперь можно усложнить задачу и найти еще и номер минимального элемента.

**Пример 5.8** Найти и вывести на экран минимальный элемент в массиве **A** и его номер.

Напрашивается такое решение: завести еще одну переменную, в которой хранить номер минимального элемента. Если мы нашли новый минимальный элемент, то в одну переменную записали его значение, а во вторую – его номер.

Тем не менее, можно обойтись одной дополнительной переменной. Дело в том, что по номеру элемента можно легко найти его значение в массиве. На этом основана приведенная ниже программа. Теперь мы запоминаем (в переменной **nMin**) не значение минимального элемента, а только его номер.

```
#include <stdio.h>
const int N = 10;
main()
{
    int i, A[N], nMin; // nMin - номер минимального элемента
    // здесь нужно ввести массив A

    nMin = 0;           // считаем, что A[0] - минимальный
    for ( i = 1; i < N; i ++ ) // цикл по всем остальным элементам
        if ( A[i] < A[nMin] ) // если A[i]<A[nMin], то...
            nMin = i;        // запомнить i

    printf("\n Минимальный элемент A[%d]=%d", nMin, A[nMin]);
}
```

Представьте, что в вазе для цветов налито молоко, а в кувшине – вода с удобрениями. Как привести все в порядок? Надо использовать третью емкость такого же (или большего) объема. Сначала переливаем в нее воду из кувшина (или молоко из вазы, все равно), затем в пустой кувшин переливаем молоко (или в вазу – воду), а затем из третьей емкости переливаем воду в вазу (или, соответственно, молоко в кувшин).

Так же и в программировании: чтобы поменять местами значения двух ячеек в памяти, надо использовать временную переменную. Пусть даны ячейки **a** и **b**, содержащие некоторые значения. После выполнения следующих команд их значения поменяются:

```
a = 4; b = 6;
c = a; // a = 4; b = 6; c = 4;
a = b; // a = 6; b = 6; c = 4;
b = c; // a = 6; b = 4; c = 4;
```

Эта цепочка операторов присваивания особая:

- ✓ она начинается и заканчивается временной переменной **c**;
- ✓ следующий оператор начинается с той переменной, на которую закончился предыдущий.

*Инверсия* (от английского *inverse* – обратный) – это такая перестановка, когда первый элемент становится последним, второй – предпоследним и т.д.

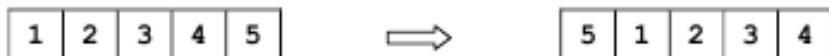


Эта простая задача имеет один подводный камень. Пусть размер массива **N**. Тогда элемент **A[0]** надо переставить с **A[N-1]**, **A[1]** с **A[N-2]** и т.д. Заметим, что в любом случае сумма индексов переставляемых элементов равна **N-1**, поэтому хочется сделать цикл от **0** до **N-1**, в котором переставить элементы **A[i]** с **A[N-1-i]**. Однако при этом вы обнаружите, что массив не изменился.

Обратим внимание, что перестановка затрагивает одновременно и первую, и вторую половину массива. Поэтому сначала все элементы будут переставлены правильно, а затем (когда **i > N/2**) будут возвращены на свои места. Правильное решение – делать перебор, при котором переменная цикла доходит только до середины массива.

```
#include <stdio.h>
const int N = 10;
main()
{
    int i, A[N], c;
    // здесь нужно ввести массив A
    for ( i = 0; i < N/2; i ++ ) // перебор только до середины!
    {
        c = A[i];                // цепочка для перестановки
        A[i] = A[N-1-i];          // A[i] и A[N-1-i]
        A[N-1-i] = c;
    }
    printf("\n Результат:\n");
    for ( i = 0; i < N; i ++ )
        printf("%d ", A[i]);
}
```

При циклическом сдвиге (вправо) первый элемент переходит на место второго, второй на место третьего и т.д., а последний элемент – на место первого.



Для выполнения циклического сдвига нам будет нужна одна временная переменная – в ней мы сохраним значение последнего элемента, пока будем переставлять остальные. Обратите внимание, что мы начинаем с конца массива, иначе массив просто заполнится первым элементом. Первый элемент ставится отдельно – копированием из временной переменной.

```
#include <stdio.h>
const int N = 10;
main()
{
int i, A[N], c;
    // здесь нужно ввести массив A
c = A[N-1];    // запомнить последний элемент
for ( i=N-1; i>0; i-- ) // цикл с уменьшением i
    A[i] = A[i-1];
A[0] = c;        // первый элемент ставим отдельно
printf("\n Результат:\n");
for ( i=0; i<N; i++ )
    printf("%d ", A[i]);
}
```

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## 6. Форма отчета о работе

Лабораторная работа № \_\_\_\_

Номер учебной группы \_\_\_\_\_

Фамилия, инициалы учащегося \_\_\_\_\_

Дата выполнения работы \_\_\_\_\_

Тема работы: \_\_\_\_\_

Цель работы: \_\_\_\_\_

Оснащение работы: \_\_\_\_\_

Результат выполнения работы: \_\_\_\_\_  
\_\_\_\_\_  
\_\_\_\_\_

## 7. Контрольные вопросы и задания

1. Что такое массив?
2. В каких случаях необходимо использовать массивы?
3. Что такое размер массива?
4. Что такое элемент массива, индекс массива?
5. Какие типы данных могут использоваться в качестве индексов для массивов?
6. Как ввести массив чисел?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ с использованием  
процедур»

Минск  
2017

# Лабораторная работа № 6

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ с использованием процедур»

## 1. Цель работы

Научить применять процедуры в конкретных задачах (алгоритмах)

## 2. Задание

Задачи первого и второго уровня обязательны для всех. Номер варианта соответствует номеру по списку. Основное решение должно быть реализовано с помощью методов. В методе `main` производится ввод данных, вызов вспомогательных методов, и вывод ответа на экран. Задачу нужно решить двумя способами: с помощью **методов возвращающих значение и методов не возвращающих значение** (через указатели).

Что бы определить вариант задачи второго уровня, необходимо воспользоваться следующей формулой:  $(N \% 19) + 1$ , где  $N$  – это номер по списку,  $\%$  - остаток от деления.

### Задачи первого уровня

**1.** Дано натуральное число  $n$ . Выясните, можно ли представить данное число в виде произведения трех последовательных натуральных чисел.

**2.** Дано натуральное число  $m$ . Укажите все тройки натуральных чисел  $x, y$  и  $z$ , удовлетворяющие следующему условию:  $m = x^3 + y^3 + z^3$ .

**3.** Дано натуральное число  $P$ . Определите, какая цифра в этом числе встречается чаще других.

**4.** Определите количество номеров машин, содержащих три одинаковые цифры (номер машины четырехзначный).

**5.** Определите количество номеров машин, содержащих две одинаковые цифры (номер машины четырехзначный).

**6.** Напишите программу нахождения следующего за данным совершенного числа.

Совершенным называется число, сумма делителей которого, не считая самого числа, равна этому числу. Первое совершенное число 6 ( $6 = 1 + 2 + 3$ ).

**7.** Проверьте, является ли данное натуральное число простым.

**8.** Дано натуральное число  $P$ . Напишите программу нахождения всех натуральных чисел, не превосходящих  $P$ , которые можно представить в виде произведения двух простых чисел.

**9.** Дано натуральное число  $P$ , заданное в восмеричной системе счисления. Напишите программу перевода этого числа в двоичную систему счисления.

**10.** Дано натуральное число  $P$ , заданное в шестнадцатеричной системе счисления. Переведите его в двоичную систему счисления.

**11.** Дано натуральное число  $P$ . Найдите все «совершенные» числа, не

превосходящие  $P$ .

«Совершенными» называются натуральные числа, сумма делителей, не включая самого числа, которых равна самому числу.

**12.** Дано натуральное  $n$ -значное число  $P$ . Проверьте, является ли данное число палиндромом (перевертышем).

**13.** Дано натуральное  $n$ -значное число  $P$ . Верно ли, что данное число содержит три одинаковые цифры?

**14.** Дано натуральное  $n$ -значное число  $P$ . Верно ли, что данное число содержит две одинаковые цифры?

**15.** Дано натуральное число  $P$ . Проверьте, кратно ли  $P$  трем, используя признак делимости на 3.

**16.** Дано натуральное число  $P$ . Проверьте, кратно ли  $P$  одиннадцати, используя признак делимости на 11 (знакопеременная сумма его цифр делится на 11).

**17.** Дано натуральное число  $P$ . Разложите данное число на простые множители.

**18.** Дано натуральное число  $P$ . Найдите все простые числа, не превосходящие числа  $P$ .

**19.** Дано натуральное число  $P$ . Найдите все натуральные числа, не превосходящие  $P$ , которые нельзя представить в виде суммы двух простых чисел.

**20.** Дано натуральное число  $P$ . Найдите все делители числа  $P$ .

**21.** Дано натуральное число  $P$ . Найдите сумму цифр числа  $P$ .

**22.** Дано натуральное число  $P$ . Выбросите из записи числа  $P$  цифры 0, оставив прежним порядок остальных цифр.

**23.** Дано натуральное число  $P$ . Проверьте, кратно ли число  $P$  девяти, используя признак делимости на 9.

**24.** Найдите все простые делители данного натурального числа .

**25.** Дано натуральное число  $n$ . Вычислите  $S_n = 1 \cdot 2 + 2 \cdot 3 \cdot 4 + \dots + n \cdot (n + 1) \cdot \dots \cdot 2n$ .

**26.** Даны обыкновенная дробь  $\frac{m}{n}$ . Сократите данную дробь.

**27.** Напишите программу сложения двух обыкновенных несократимых дробей  $\frac{m}{n}$  и  $\frac{p}{q}$ . Результат представить в виде несократимой дроби.

**28.** Напишите программу вычитания двух обыкновенных несократимых дробей  $\frac{m}{n}$  и  $\frac{p}{q}$ . Результат представить в виде несократимой дроби.

**29.** Напишите программу умножения двух обыкновенных несократимых дробей  $\frac{m}{n}$  и  $\frac{p}{q}$ . Результат представить в виде несократимой дроби.

**30.** Напишите программу деления двух обыкновенных несократимых

дробей  $\frac{m}{n}$  и  $\frac{p}{q}$ . Результат представить в виде несократимой дроби.

**31.** Данное натуральное число  $N$  переведите из десятичной системы счисления в двоичную.

**32.** Данную десятичную дробь переведите в двоичную систему счисления

**33.** Данную двоичную дробь переведите в десятичную систему счисления.

## Задачи второго уровня

1. Если последнюю цифру некоторого натурального числа  $n$  перенести и поставить перед первой цифрой этого числа, то получится число, в два раза больше, чем  $n$ . Найдите самое маленькое из таких чисел.
2. Дано натуральное число  $n$ . Выясните, можно ли представить данное число в виде произведения трех последовательных натуральных чисел.
3. Результатом деления натурального числа  $m$  на натуральное число  $n$  является либо конечная десятичная дробь, либо бесконечная периодическая дробь. Напишите программу нахождения этого результата. Если дробь бесконечная, то найти все цифры до первого периода и цифры периода
4. Определите количество «счастливых» талонов для проезда в городском транспорте.  
«Счастливым» считается талон, у которого сумма первых трех цифр равна сумме трех других цифр.
5. Найдите все натуральные числа, которые в  $k$  раз больше суммы своих цифр.
6. Данное натуральное число  $N$  замените суммой квадратов его цифр. Произведите  $K$  таких замен.
7. Данное натуральное число  $N$  переведите из десятичной системы счисления в двоичную.
8. Данную десятичную дробь переведите в двоичную систему счисления.
9. Данную двоичную дробь переведите в десятичную систему счисления.
10. Дано натуральное число  $n$ . Найдите все меньшие  $n$  числа Мерсена.  
Простое число называется числом Мерсена, если оно может быть представлено в виде  $2^p - 1$ , где  $p$  – тоже простое число.
11. Найдите все простые делители данного натурального числа.
12. Для данного натурального числа найдите произведение его простых делителей, взятых по одному разу.
13. На данном отрезке  $[a;b]$  найдите все числа «близнецы». Два простых числа называют «близнецами», если разность между ними равна 2.
14. Даны два натуральных числа  $N$  и  $M$ . Найдите все меньшие  $N$

- натуральные числа, квадрат суммы цифр которых равны M.
15. Дано натуральное число  $N \leq 50$ . Найдите все способы выплаты данной суммы с помощью монет достоинством 1, 5, 10.
16. Дано натуральное число N. Найдите все составные натуральные числа, меньшие N.
17. Проверьте, является ли данное натуральное число палиндромом.
18. Дано натуральное число N. Найдите все автоморфные натуральные числа, меньшие N. Автоморфными называются числа, последние цифры квадрата которых совпадают с самим числом. Например:  $5^2 = 25$ ,  $25^2 = 625$ .
19. На данном отрезке  $[a;b]$  найдите все пары дружелюбных чисел. Два натуральных числа называют дружелюбными, если каждый из них равен сумме всех делителей другого, за исключением самого числа.

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

**Функция (процедура)** — это самостоятельная единица программы, которая спроектирована для реализации конкретной подзадачи.

Функция является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно (в библиотеке). Каждая функция выполняет в программе определенные действия.

**Сигнатура** функции определяет правила использования функции. Обычно сигнатурой представляет собой описание функции, включающее имя функции, перечень формальных параметров с их типами и тип возвращаемого значения.

**Семантика** функции определяет способ реализации функции. Обычно представляет собой тело функции.

#### Определение функции

Каждая функция в языке Си должна быть определена, то есть должны быть указаны:

- ✓ тип возвращаемого значения;
- ✓ имя функции;
- ✓ информация о формальных аргументах;
- ✓ тело функции.

Определение функции имеет следующий синтаксис:

```
ТипВозвращаемогоЗначения ИмяФункции(СписокФормальныхАргументов)
{
    ТелоФункции;
    ...
    return(ВозвращаемоеЗначение);
}
```

**Пример 6.1** Функция сложения двух вещественных чисел

```
1 float function(float x, float z)
2 {
3     float y;
4     y=x+z;
5     return(y);
6 }
```

В указанном примере возвращаемое значение имеет тип **float**. В качестве возвращаемого значения в вызывающую функцию передается значение переменной **y**. Формальными аргументами являются значения переменных **x** и **z**.

Если функция не возвращает значения, то тип возвращаемого значения для нее указывается как **void**. При этом операция **return** может быть опущена. Если функция не принимает аргументов, в круглых скобках также указывается **void**.

Различают системные (в составе систем программирования) и собственные функции.

Системные функции хранятся в стандартных библиотеках, и пользователю не нужно вдаваться в подробности их реализации. Достаточно знать лишь их сигнатуру. Примером системных функций, используемых ранее, являются функции **printf()** и **scanf()**.

Собственные функции – это функции, написанные пользователем для решения конкретной подзадачи.

Разбиение программ на функции дает следующие преимущества:

- ✓ функцию можно вызвать из различных мест программы, что позволяет избежать повторения программного кода;
- ✓ одну и ту же функцию можно использовать в разных программах;
- ✓ функции повышают уровень модульности программы и облегчают ее проектирование;
- ✓ использование функций облегчает чтение и понимание программы и ускоряет поиск и исправление ошибок.

С точки зрения вызывающей программы функцию можно представить как некий "черный ящик", у которого есть несколько входов и один выход. С точки зрения вызывающей программы неважно, каким образом производится обработка информации внутри функции. Для корректного использования функции достаточно знать лишь ее сигнатуру.

## Вызов функции

Общий вид вызова функции

```
Переменная = ИмяФункции(СписокФактическихАргументов);
```

**Фактический аргумент** — это величина, которая присваивается формальному аргументу при вызове функции. Таким образом, **формальный аргумент** — это переменная в вызываемой функции, а **фактический аргумент** — это конкретное значение, присвоенное этой переменной

вызывающей функцией. Фактический аргумент может быть константой, переменной или выражением. Если фактический аргумент представлен в виде выражения, то его значение сначала вычисляется, а затем передается в вызываемую функцию. Если в функцию требуется передать несколько значений, то они записываются через запятую. При этом формальные параметры заменяются значениями фактических параметров в порядке их следования в сигнатуре функции.

### Возврат в вызывающую функцию

По окончании выполнения вызываемой функции осуществляется возврат значения в точку ее вызова. Это значение присваивается переменной, тип которой должен соответствовать типу возвращаемого значения функции. Функция может передать в вызывающую программу только одно значение. Для передачи возвращаемого значения в вызывающую функцию используется оператор **return** в одной из форм:

```
return(ВозвращаемоеЗначение);
```

```
return ВозвращаемоеЗначение;
```

Действие оператора следующее: значение выражения, заключенного в скобки, вычисляется и передается в вызывающую функцию. Возвращаемое значение может использоваться в вызывающей программе как часть некоторого выражения.

Оператор **return** также завершает выполнение функции и передает управление следующему оператору в вызывающей функции. Оператор **return** не обязательно должен находиться в конце тела функции.

Функции могут и не возвращать значения, а просто выполнять некоторые вычисления. В этом случае указывается пустой тип возвращаемого значения **void**, а оператор **return** может либо отсутствовать, либо не возвращать никакого значения:

```
return;
```

## Пример 6.2 Посчитать сумму двух чисел.

```
1 #define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
2 #include <stdio.h>
3 // Функция вычисления суммы двух чисел
4 int sum(int x, int y) // в функцию передаются два целых числа
5 {
6     int k = x + y; // вычисляем сумму чисел и сохраняем в k
7     return k; // возвращаем значение k
8 }
9 int main()
10 {
11     int a, r; // описание двух целых переменных
12     printf("a= ");
13     scanf("%d", &a); // вводим a
14     r = sum(a, 5); // вызов функции: x=a, y=5
15     printf("%d + 5 = %d", a, r); // вывод: a + 5 = r
16     getchar(); getchar(); // мы использовали scanf(),
17     return 0; // поэтому getchar() вызываем дважды
18 }
```

В языке Си нельзя определять одну функцию внутри другой.

В языке Си нет требования, чтобы семантика функции обязательно предшествовало её вызову. Функции могут определяться как до вызывающей функции, так и после нее. Однако если семантика вызываемой функции описывается ниже ее вызова, необходимо до вызова функции определить прототип этой функции, содержащий:

- ✓ тип возвращаемого значения;
- ✓ имя функции;
- ✓ типы формальных аргументов в порядке их следования.

Прототип необходим для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических аргументов типам формальных аргументов. Имена формальных аргументов в прототипе функции могут отсутствовать.

Если в примере выше тело функции сложения чисел разместить после тела функции **main**, то код будет выглядеть следующим образом:

```

1 #define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
2 #include <stdio.h>
3 int sum(int, int); // сигнатура
4 int main()
5 {
6     int a, r;
7     printf("a= ");
8     scanf("%d", &a);
9     r = sum(a, 5); // вызов функции: x=a, y=5
10    printf("%d + 5 = %d", a, r);
11    getchar(); getchar();
12    return 0;
13 }
14 int sum(int x, int y) // семантика
15 {
16     int k;
17     k = x + y;
18     return(k);
19 }
```

**Пример 6.3** Написать программу, которая вводит целое число и определяет сумму его цифр. Использовать функцию, вычисляющую сумму цифр числа.

Вспомним, что для того чтобы найти последнюю цифру числа, надо взять остаток от его деления на 10. Затем делим число на 10, отбрасывая его последнюю цифру, и т.д. Сложив все эти остатки-цифры, мы получим сумму цифр числа.

```

#include <stdio.h>
#include <conio.h>
int SumDigits ( int N ) // заголовок функции
{ // начало функции
int d, sum = 0;
while ( N != 0 )
{
    d = N % 10; // тело функции
    sum = sum + d;
    N = N / 10;
}
return sum; // функция возвращает значение sum
} // конец функции

main()
{
int N, s;
printf ( "\nВведите целое число " );
scanf ( "%d", &N );
s = SumDigits ( N ); // вызов функции
printf ( "Сумма цифр числа %d равна %d\n", N, s );
getch();
}
```

Желательно выбирать осмысленные имена параметров — это позволяет легче разобраться в программе потом.

Очень часто надо составить функцию, которая просто решает какой-то вопрос и отвечает на вопрос «Да» или «Нет». Такие функции называются **логическими**. Вспомним, что в Си ноль означает ложное условие, а единица – истинное.

**Логическая функция** – это функция, возвращающая **1** (если ответ «Да») или **0** (если ответ «Нет»).

Логические функции используются, главным образом, в двух случаях:

- ✓ если надо проанализировать ситуацию и ответить на вопрос, от которого зависят дальнейшие действия программы;
- ✓ если надо выполнить какие-то сложные операции и определить, была ли при этом какая-то ошибка.

**Пример 6.4** Ввести число  $N$  и определить, простое оно или нет. Использовать функцию, которая отвечает на этот вопрос.

Теперь расположим тело функции ниже основной программы. Чтобы транслятор знал об этой функции во время обработки основной программы, надо объявить её заранее.

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

int Prime ( int N ); // объявление функции

main()
{
    int N;
    printf ( "\nВведите целое число " );
    scanf ( "%d", &N );

    if ( Prime(N) )      // вызов функции
        printf ( "Число %d - простое\n", N );
    else printf ( "Число %d - составное\n", N );
    getch();
}

int Prime ( int N ) // описание функции
{
    for ( int i = 2; i*i <= N; i++ )
        if ( N % i == 0 ) return 0; // нашли делитель - составное!
    return 1; // не нашли ни одного делителя - простое!
}
```

По определению функция может вернуть только одно значение-результат. Если надо вернуть два и больше результатов, приходится использовать специальный прием — **передачу параметров по ссылке**.

**Пример 6.5** Написать функцию, которая определяет максимальное и минимальное из двух целых чисел.

В следующей программе используется достаточно хитрый прием: мы сделаем так, чтобы функция изменяла значение переменной, которая принадлежит основной программе. Один результат (минимальное из двух чисел) функция вернет как обычно, а второй – за счет изменения переменной, которая передана из основной программы.

```

#include <stdio.h>
#include <conio.h>
int MinMax ( int a, int b, int &Max )
{
if ( a > b ) { Max = a; return b; }
else           { Max = b; return a; }
}
main()
{
int N, M, min, max;
printf ( "\nВведите 2 целых числа " );
scanf ( "%d%d", &N, &M );
min = MinMax ( N, M, max ); // вызов функции
printf ( "Наименьшее из них %d, наибольшее — %d\n", min, max );
getch();
}

```

параметр-результат

Обычно при передаче параметра в процедуру или функцию в памяти создается копия переменной, и функция работает с этой копией. Это значит, что все изменения переменной-параметра, сделанные в функции, не отражаются на значении этой переменной в вызывающей программе.

Если перед именем параметра в заголовке функции поставить знак **&** (вспомним, что он также используется для определения адреса переменной), то функция работает прямо с переменной из вызывающей программы, а не с ее копией. Поэтому в нашем примере функция изменит значение переменной **max** из основной программы и запишет туда максимальное из двух чисел.

Этот приём можно использовать и для процедур: хотя формально они не возвращают никакого значения-результата, можно всё-таки передавать данные в вызывающую программу через изменяемые параметры.

Если надо, чтобы функция вернула два и более результатов, поступают следующим образом:

- ✓ один результат передается как обычно с помощью оператора **return**;
  
- ✓ остальные возвращаемые значения передаются через изменяемые параметры.

Обычные параметры не могут изменяться подпрограммой, потому что она работает с *копиями* параметров (например, если менять значения **a** и **b** в функции **MinMax**, соответствующие им переменные **N** и **M** в основной программе не изменятся).

Любая процедура и функция может возвращать значения через изменяемые параметры.

Изменяемые параметры (или параметры, передаваемые по ссылке) объявляются в заголовке подпрограммы специальным образом: перед их именем ставится знак **&** — в данном случае он означает ссылку, то есть

подпрограмма может менять значение параметра (в данном случае функция меняет значение переменной **max** в основной программе).

При вызове таких функций и процедур вместо каждого фактического изменяемого параметра надо подставлять только имя переменной (не число и не арифметическое выражение — в этих случаях транслятор выдает предупреждение и формирует в памяти временную переменную).

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## 6. Форма отчета о работе

*Лабораторная работа №* \_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_  
\_\_\_\_\_

## 7. Контрольные вопросы и задания

1. Что такое подпрограмма?
2. С помощью каких структур реализуются подпрограммы в языке Си?
3. Как указать тип возвращаемого значения из функции?
4. За что отвечает ключевое слово **void**?
5. За что отвечает ключевое слово **return**?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ с использованием  
функций»

Минск  
2017

## Лабораторная работа № 7

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ с использованием функций»

### 1. Цель работы

Научить применять функции в конкретных задачах (алгоритмах).

### 2. Задание

Номер варианта соответствует номеру по списку. Основное решение должно быть реализовано с помощью методов. В методе main производится ввод данных, вызов вспомогательных методов, и вывод ответа на экран. Задачу нужно решить двумя способами: с помощью **методов возвращающих значение и методов не возвращающих значение** (через указатели).

1. Треугольник задан координатами своих вершин. Составить программу для вычисления его площади.

2. Составить программу для нахождения наибольшего общего делителя и наименьшего общего кратного двух натуральных чисел

$$(\text{НОК}(A, B) = \frac{A \cdot B}{\text{НОД}(A, B)})$$

3. Составить программу для нахождения наибольшего общего делителя четырех натуральных чисел.

4. Составить программу для нахождения наименьшего общего кратного трех натуральных чисел.

5. Написать программу для нахождения суммы большего и меньшего из трех чисел.

6. Вычислить площадь правильного шестиугольника со стороной A, используя подпрограмму вычисления площади треугольника.

7. Составить программу, проверяющую, являются ли данные три числа взаимно простыми.

8. Написать программу для вычисления суммы факториалов всех нечетных чисел от 1 до 9.

9. Составить программу для вычисления суммы факториалов всех четных чисел от 1 до n.

10. Дано простое число. Составить функцию, которая будет находить следующее за ним простое число.

11. Составить функцию для нахождения наименьшего нечетного натурального делителя k ( $k > 1$ ) любого заданного натурального числа n.

12. Составить программу, определяющую, в каком из данных двух чисел больше цифр.

13. Заменить данное натуральное число на число, которое получается из исходного записью его цифр в обратном порядке (например, дано число 156, нужно получить 651).

14. Написать программу, которая находит и выводит на печать все четырехзначные числа вида , для которых выполняется:  
А) **A, B, C, D** — разные цифры;  
Б) **AB - CD = A + B + C + D**.
15. Найти все простые натуральные числа, не превосходящие **n**, двоичная запись которых представляет собой палиндром, т.е. читается одинаково слева направо и справа налево.
- 16.Найти все натуральные **N**-значные числа, цифры в которых образуют строго возрастающую последовательность (например, 1234, 5789).
- 17.Найти все натуральные числа, не превосходящие заданного **n**, которые делятся на каждую из своих цифр.
- 18.Составить программу для нахождения чисел из интервала [m, N], имеющих наибольшее количество делителей.
- 19.Дано натуральное число **n**. Выяснить, можно ли представить его в виде произведения трех последовательных натуральных чисел.
20. Написать программу, определяющую Сумму **N**-значных чисел, содержащих только нечетные цифры. Определить также, сколько четных цифр в найденной Сумме.
- 21.Из заданного числа вычли сумму его цифр. Из результата вновь вычли Сумму его цифр и т.д. Сколько таких действий надо произвести, чтобы получился нуль?
- 22.Составить программу для разложения данного натурального числа на простые множители. Например,  $200 = 2^3 \cdot 5^2$ .
- 23.Дано четное число **n**. Проверить для него гипотезу Гольдбаха: каждое четное **n** представляется в виде суммы двух простых чисел.
- 24.Необходимо действительное число **a** возвести в целую степень **n**. Встроенной функцией не пользоваться.
- 25.Определить, в каком из данных двух целых чисел больше цифр.
- 26.Вычисляющую площадь кольца, зная внешней и внутренней радиусы окружностей.
- 27.Написать подпрограмму, вычисляющую сумму делителей данного натурального числа.
- 28.Написать подпрограмму, которая возвращает максимальную из двух обыкновенных дробей вида  $\frac{a}{b}$  и  $\frac{c}{d}$ .

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++** .

## 4. Основные теоретические сведения

*Функция (процедура)* — это самостоятельная единица программы, которая спроектирована для реализации конкретной подзадачи.

Функция является подпрограммой, которая может содержаться в основной программе, а может быть создана отдельно (в библиотеке). Каждая функция выполняет в программе определенные действия.

*Сигнатура* функции определяет правила использования функции. Обычно сигнатура представляет собой описание функции, включающее имя функции, перечень формальных параметров с их типами и тип возвращаемого значения.

*Семантика* функции определяет способ реализации функции. Обычно представляет собой тело функции.

### Определение функции

Каждая функция в языке Си должна быть определена, то есть должны быть указаны:

- ✓ тип возвращаемого значения;
- ✓ имя функции;
- ✓ информация о формальных аргументах;
- ✓ тело функции.

Определение функции имеет следующий синтаксис:

```
ТипВозвращаемогоЗначения ИмяФункции(СписокФормальныхАргументов)
{
    ТелоФункции;
    ...
    return(ВозвращаемоеЗначение);
}
```

**Пример 7.1** Функция сложения двух вещественных чисел

```
1   float function(float x, float z)
2   {
3       float y;
4       y=x+z;
5       return(y);
6   }
```

В указанном примере возвращаемое значение имеет тип **float**. В качестве возвращаемого значения в вызывающую функцию передается значение переменной **y**. Формальными аргументами являются значения переменных **x** и **z**.

Если функция не возвращает значения, то тип возвращаемого значения для нее указывается как **void**. При этом операция **return** может быть опущена. Если функция не принимает аргументов, в круглых скобках также указывается **void**.

Различают системные (в составе систем программирования) и собственные функции.

Системные функции хранятся в стандартных библиотеках, и пользователю не нужно вдаваться в подробности их реализации. Достаточно

знать лишь их сигнатуру. Примером системных функций, используемых ранее, являются функции **printf()** и **scanf()**.

Собственные функции – это функции, написанные пользователем для решения конкретной подзадачи.

Разбиение программ на функции дает следующие преимущества:

- ✓ функцию можно вызвать из различных мест программы, что позволяет избежать повторения программного кода;
- ✓ одну и ту же функцию можно использовать в разных программах;
- ✓ функции повышают уровень модульности программы и облегчают ее проектирование;
- ✓ использование функций облегчает чтение и понимание программы и ускоряет поиск и исправление ошибок.

С точки зрения вызывающей программы функцию можно представить как некий "черный ящик", у которого есть несколько входов и один выход. С точки зрения вызывающей программы неважно, каким образом производится обработка информации внутри функции. Для корректного использования функции достаточно знать ее сигнатуру.

## Вызов функции

Общий вид вызова функции

```
Переменная = ИмяФункции(СписокФактическихАргументов);
```

*Фактический аргумент* — это величина, которая присваивается формальному аргументу при вызове функции. Таким образом, *формальный аргумент* — это переменная в вызываемой функции, а *фактический аргумент* — это конкретное значение, присвоенное этой переменной вызывающей функцией. Фактический аргумент может быть константой, переменной или выражением. Если фактический аргумент представлен в виде выражения, то его значение сначала вычисляется, а затем передается в вызываемую функцию. Если в функцию требуется передать несколько значений, то они записываются через запятую. При этом формальные параметры заменяются значениями фактических параметров в порядке их следования в сигнатуре функции.

## Возврат в вызывающую функцию

По окончании выполнения вызываемой функции осуществляется возврат значения в точку ее вызова. Это значение присваивается переменной, тип которой должен соответствовать типу возвращаемого значения функции. Функция может передать в вызывающую программу только одно значение. Для передачи возвращаемого значения в вызывающую функцию используется оператор **return** в одной из форм:

```
return(ВозвращаемоеЗначение);
```

```
return ВозвращаемоеЗначение;
```

Действие оператора следующее: значение выражения, заключенного в скобки, вычисляется и передается в вызывающую функцию. Возвращаемое значение может использоваться в вызывающей программе как часть некоторого выражения.

Оператор **return** также завершает выполнение функции и передает управление следующему оператору в вызывающей функции. Оператор **return** не обязательно должен находиться в конце тела функции.

Функции могут и не возвращать значения, а просто выполнять некоторые вычисления. В этом случае указывается пустой тип возвращаемого значения **void**, а оператор **return** может либо отсутствовать, либо не возвращать никакого значения:

```
return;
```

### Пример7.2 Посчитать сумму двух чисел.

```
1 #define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
2 #include <stdio.h>
3 // Функция вычисления суммы двух чисел
4 int sum(int x, int y) // в функцию передаются два целых числа
5 {
6     int k = x + y; // вычисляем сумму чисел и сохраняем в k
7     return k; // возвращаем значение k
8 }
9 int main()
10 {
11     int a, r; // описание двух целых переменных
12     printf("a= ");
13     scanf("%d", &a); // вводим a
14     r = sum(a, 5); // вызов функции: x=a, y=5
15     printf("%d + 5 = %d", a, r); // вывод: a + 5 = r
16     getchar(); getchar(); // мы использовали scanf(),
17     return 0; // поэтому getchar() вызываем дважды
18 }
```

В языке Си нельзя определять одну функцию внутри другой.

В языке Си нет требования, чтобы семантика функции обязательно предшествовало её вызову. Функции могут определяться как до вызывающей функции, так и после нее. Однако если семантика вызываемой функции описывается ниже ее вызова, необходимо до вызова функции определить прототип этой функции, содержащий:

- ✓ тип возвращаемого значения;
- ✓ имя функции;
- ✓ типы формальных аргументов в порядке их следования.

Прототип необходим для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических аргументов типам формальных аргументов. Имена формальных аргументов в прототипе функции могут отсутствовать.

Если в примере выше тело функции сложения чисел разместить после тела функции **main**, то код будет выглядеть следующим образом:

```
1 #define _CRT_SECURE_NO_WARNINGS // для возможности использования scanf
2 #include <stdio.h>
3 int sum(int, int); // сигнатура
4 int main()
5 {
6     int a, r;
7     printf("a= ");
8     scanf("%d", &a);
9     r = sum(a, 5); // вызов функции: x=a, y=5
10    printf("%d + 5 = %d", a, r);
11    getchar(); getchar();
12    return 0;
13 }
14 int sum(int x, int y) // семантика
15 {
16     int k;
17     k = x + y;
18     return(k);
19 }
```

**Пример 7.3** Написать программу, которая вводит целое число и определяет сумму его цифр. Использовать функцию, вычисляющую сумму цифр числа.

Вспомним , что для того чтобы найти последнюю цифру числа, надо взять остаток от его деления на 10. Затем делим число на 10, отбрасывая его последнюю цифру, и т.д. Сложив все эти остатки-цифры, мы получим сумму цифр числа.

```
#include <stdio.h>
#include <conio.h>
int SumDigits ( int N ) // заголовок функции
{ // начало функции
int d, sum = 0;
while ( N != 0 )
{
    d = N % 10; // тело функции
    sum = sum + d;
    N = N / 10;
}
return sum; // функция возвращает значение sum
} // конец функции
main()
{
int N, s;
printf ( "\nВведите целое число " );
scanf ( "%d", &N );
s = SumDigits ( N ); // вызов функции
printf ( "Сумма цифр числа %d равна %d\n", N, s );
getch();
}
```

Желательно выбирать осмысленные имена параметров — это позволяет легче разобраться в программе потом.

Очень часто надо составить функцию, которая просто решает какой-то вопрос и отвечает на вопрос «Да» или «Нет». Такие функции называются **логическими**. Вспомним, что в Си ноль означает ложное условие, а единица — истинное.

**Логическая функция** — это функция, возвращающая **1** (если ответ «Да») или **0** (если ответ «Нет»).

Логические функции используются, главным образом, в двух случаях:

- ✓ если надо проанализировать ситуацию и ответить на вопрос, от которого зависят дальнейшие действия программы;
- ✓ если надо выполнить какие-то сложные операции и определить, была ли при этом какая-то ошибка.

**Пример 7.4** Ввести число  $N$  и определить, простое оно или нет. Использовать функцию, которая отвечает на этот вопрос.

Теперь расположим тело функции ниже основной программы. Чтобы транслятор знал об этой функции во время обработки основной программы, надо объявить её заранее.

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

int Prime ( int N ); // объявление функции

main()
{
    int N;
    printf ( "\nВведите целое число " );
    scanf ( "%d", &N );

    if ( Prime(N) )      // вызов функции
        printf ( "Число %d - простое\n", N );
    else printf ( "Число %d - составное\n", N );
    getch();
}

int Prime ( int N ) // описание функции
{
    for ( int i = 2; i*i <= N; i ++ )
        if ( N % i == 0 ) return 0; // нашли делитель - составное!
    return 1; // не нашли ни одного делителя - простое!
}
```

По определению функция может вернуть только одно значение-результат. Если надо вернуть два и больше результатов, приходится использовать специальный прием — **передачу параметров по ссылке**.

**Пример 7.5** Написать функцию, которая определяет максимальное и минимальное из двух целых чисел.

В следующей программе используется достаточно хитрый прием: мы сделаем так, чтобы функция изменяла значение переменной, которая принадлежит основной программе. Один результат (минимальное из двух чисел) функция вернет как обычно, а второй — за счет изменения переменной, которая передана из основной программы.

```

#include <stdio.h>
#include <conio.h>
int MinMax ( int a, int b, int &Max )
{
if ( a > b ) { Max = a; return b; }
else           { Max = b; return a; }
}
main()
{
int N, M, min, max;
printf ( "\nВведите 2 целых числа " );
scanf ( "%d%d", &N, &M );
min = MinMax ( N, M, max ); // вызов функции
printf ( "Наименьшее из них %d, наибольшее — %d\n", min, max );
getch();
}

```

параметр-результат

Обычно при передаче параметра в процедуру или функцию в памяти создается копия переменной, и функция работает с этой копией. Это значит, что все изменения переменной-параметра, сделанные в функции, не отражаются на значении этой переменной в вызывающей программе.

Если перед именем параметра в заголовке функции поставить знак **&** (вспомним, что он также используется для определения адреса переменной), то функция работает прямо с переменной из вызывающей программы, а не с ее копией. Поэтому в нашем примере функция изменит значение переменной **max** из основной программы и запишет туда максимальное из двух чисел.

Этот приём можно использовать и для процедур: хотя формально они не возвращают никакого значения-результата, можно всё-таки передавать данные в вызывающую программу через изменяемые параметры.

Если надо, чтобы функция вернула два и более результатов, поступают следующим образом:

- ✓ один результат передается как обычно с помощью оператора **return**;
  
- ✓ остальные возвращаемые значения передаются через изменяемые параметры.

Обычные параметры не могут изменяться подпрограммой, потому что она работает с *копиями* параметров (например, если менять значения **a** и **b** в функции **MinMax**, соответствующие им переменные **N** и **M** в основной программе не изменятся).

Любая процедура и функция может возвращать значения через изменяемые параметры.

Изменяемые параметры (или параметры, передаваемые по ссылке) объявляются в заголовке подпрограммы специальным образом: перед их именем ставится знак **&** — в данном случае он означает ссылку, то есть

подпрограмма может менять значение параметра (в данном случае функция меняет значение переменной **max** в основной программе).

При вызове таких функций и процедур вместо каждого фактического изменяемого параметра надо подставлять только имя переменной (не число и не арифметическое выражение — в этих случаях транслятор выдает предупреждение и формирует в памяти временную переменную).

При оформлении функций и процедур рекомендуется придерживаться следующих правил:

Однократные операции в разных частях программы оформляются в виде подпрограмм.

Имена функций и процедур должны быть информативными, то есть нести информацию о том, что делает эта подпрограмма. К сожалению, транслятор не понимает русские имена, поэтому приходится писать по-английски. Если вам сложно писать имена на английском языке, можно писать русские слова английскими буквами. Например, процедуру, рисующую квадрат, можно объявить так:

```
void Square ( int x, int y, int a );
```

или так

```
void Kvadrat ( int x, int y, int a );
```

Перед заголовком подпрограммы надо вставлять несколько строк с комментарием (здесь можно писать по-русски). В комментарий записывается самая важная информация: что означают параметры подпрограммы, что она делает, какое значение она возвращает (если это функция), её особенности.

Не рекомендуется делать подпрограммы длиной более 25-30 строк, так как при этом они становятся сложными и запутанными. Если подпрограмма получается длинной, ее надо разбить на более мелкие процедуры и функции.

Чтобы отделить одну часть подпрограммы от другой, используют пустые строки. Крупный смысловой блок функции можно выделять строкой знаков минус в комментариях.

Приведем пример оформления на примере функции, которая рисует на экране закрашенный ромб с заданными параметрами, если весь ромб умещается на экран (при этом результат функции равен 1), или возвращает признак ошибки (число 0).

```

//*****
// ROMB - рисование ромба в заданной позиции
//      (x,y)    - координаты центра ромба
//      a, b - ширина и высота ромба
//      color, colorFill - цвета границы и заливки
//      Возвращает 1, если операция выполнена, и 0 если
//      ромб выходит за пределы экрана
//*****
int Romb ( int x, int y, int a, int b, int color,
           int colorFill )
{
    if ( (x < a) || (x > 640-a) || (y < b) || (y > 480-b) )
        return 0;
    -----
    setcolor ( color );
    line ( x-a, y, x, y-b ); line ( x-a, y, x, y+b );
    line ( x+a, y, x, y-b ); line ( x+a, y, x, y+b );

    setfillstyle ( SOLID_FILL, colorFill );
    floodfill ( x, y, color );
    return 1;
}

```

заголовок

обработка ошибок

крупный блок

пустая строка

Отступы используются для выделения структурных блоков программы (подпрограмм, циклов, условных операторов). Отступы позволяют легко искать лишние и недостающие скобки, понимать логику работы программы и находить в ней ошибки. При расстановке отступов рекомендуется соблюдать следующие правила:

Величина отступа равна 2-4 символа.

Дополнительным отступом выделяются:

- ✓ тело циклов **for**, **while**, **do-while**;
- ✓ тело условного оператора **if** и блока **else**;
  
- ✓ тело оператора множественного выбора **switch**.

Вот пример записи программы с отступами (запись «лесенкой»):

```
main()
{
    int a = 1, b = 4, i;
    for (i=1; i<2; i++)
    {
        if ( a > b )
        {
            b = a + i;
        }
        else
        {
            a = b + i;
        }
    }
    printf ( "%d %d\n", a, b );
}
```

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## 6. Форма отчета о работе

Лабораторная работа № \_\_\_\_

Номер учебной группы \_\_\_\_\_

Фамилия, инициалы учащегося \_\_\_\_\_

Дата выполнения работы \_\_\_\_\_

Тема работы: \_\_\_\_\_

Цель работы: \_\_\_\_\_

Оснащение работы: \_\_\_\_\_

Результат выполнения работы: \_\_\_\_\_

\_\_\_\_\_

## 7. Контрольные вопросы и задания

1. Что такое подпрограмма?
2. С помощью каких структур реализуются подпрограммы в языке Си?
3. Как указать тип возвращаемого значения из функции?

4. За что отвечает ключевое слово **void**?
5. За что отвечает ключевое слово **return**?

### **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ с использованием  
указателей»

Минск  
2017

## **Лабораторная работа № 8**

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ с использованием указателей»

### **1. Цель работы**

Изучить механизмы применения и использования указателей.

### **2. Задание**

Реализовать одномерные массивы с помощью указателей, и решить задачу по варианту. Вариант соответствует номеру по списку.

1. В одномерном массиве, состоящем из **n** вещественных элементов необходимо вычислить сумму отрицательных элементов массива.
2. В одномерном массиве, состоящем из **n** вещественных элементов необходимо вычислить произведение элементов стоящих между минимальным и максимальным элементами массива.
3. В одномерном массиве, состоящем из **n** вещественных элементов необходимо вычислить сумму положительных элементов массива.
4. В одномерном массиве, состоящем из **n** вещественных элементов необходимо вычислить произведение элементов стоящих между минимальным по модулю и максимальным по модулю элементами массива.
5. В одномерном массиве, состоящем из **n** целых элементов необходимо вычислить произведение элементов массива с четными номерами.
6. В одномерном массиве, состоящем из **n** целых элементов необходимо вычислить произведение элементов массива с нечетными номерами.
7. В одномерном массиве, состоящем из **n** целых элементов необходимо вычислить сумму элементов массива расположенных между первым и последними нулевыми элементами.
8. В одномерном массиве, состоящем из **n** целых элементов необходимо вычислить сумму элементов массива расположенных до первого нулевого элемента.
9. В одномерном массиве, состоящем из **n** целых элементов необходимо вычислить сумму элементов массива расположенных после последнего нулевого элемента.
10. В одномерном массиве, состоящем из **n** целых элементов необходимо вычислить произведение элементов массива расположенных до первого нулевого элемента.
11. В одномерном массиве, состоящем из **n** целых элементов необходимо вычислить произведение элементов массива расположенных после последнего нулевого элемента.

12. В одномерном массиве, состоящем из **n** вещественных элементов необходимо вычислить сумму элементов массива, расположенных между первым и последним отрицательными элементами.
13. В одномерном массиве, состоящем из **n** вещественных элементов необходимо вычислить произведение элементов до минимального значения массива.
14. В одномерном массиве, состоящем из **n** вещественных элементов необходимо вычислить произведение элементов массива после максимального значения массива.
15. В одномерном массиве, состоящем из **n** вещественных элементов необходимо вычислить сумму между первым и последним положительными элементами массива.
16. В одномерном массиве, состоящем из **n** целых элементов необходимо вычислить сумму элементов между первым и вторым положительными элементами.
17. В одномерном массиве, состоящем из **n** вещественных элементов необходимо найти максимальный по модулю элемент массива.
18. В одномерном массиве, состоящем из **n** вещественных элементов необходимо найти минимальный по модулю элемент массива.
19. В одномерном массиве, состоящем из **n** целых элементов необходимо найти среднее арифметическое всех элементов массива.
20. В одномерном массиве, состоящем из **n** целых элементов необходимо найти среднее арифметическое положительных элементов массива.
21. В одномерном массиве, состоящем из **n** целых элементов необходимо найти среднее арифметическое всех отрицательных элементов массива.
22. В одномерном массиве, состоящем из **n** целых элементов необходимо найти количество элементов массива больших среднего арифметического всех элементов массива.
23. В одномерном массиве, состоящем из **n** целых элементов необходимо найти количество элементов массива меньших среднего арифметического всех элементов массива.
24. В одномерном массиве, состоящем из **n** целых элементов необходимо найти элемент наиболее близкий к среднему арифметическому всех элементов массива.
25. В одномерном массиве, состоящем из **n** целых элементов необходимо найти среднее гармоническое всех элементов массива.
26. В одномерном массиве, состоящем из **n** натуральных элементов необходимо найти среднее геометрическое всех элементов массива.
27. В одномерном массиве, состоящем из **n** целых элементов необходимо найти количество элементов массива кратных натуральному числу **k**.
28. В одномерном массиве, состоящем из **n** целых элементов необходимо найти количество цифр, в сумме всех элементов массива.

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

Указатель — переменная, содержащая адрес объекта. Указатель не несет информации о содержимом объекта, а содержит сведения о том, где размещен объект.

Указатели широко используются в программировании на языке Си.

Указатели часто используются при работе с массивами.

Память компьютера можно представить в виде последовательности пронумерованных однобайтовых ячеек, с которыми можно работать по отдельности или блоками.

Каждая переменная в памяти имеет свой адрес - номер первой ячейки, где она расположена, а также свое значение. Указатель — это тоже переменная, которая размещается в памяти. Она тоже имеет адрес, а ее значение является адресом некоторой другой переменной. Переменная, объявленная как указатель, занимает 4 байта в оперативной памяти (в случае 32-битной версии компилятора).

Указатель, как и любая переменная, должен быть объявлен.

Общая форма объявления указателя

тип \*ИмяОбъекта;

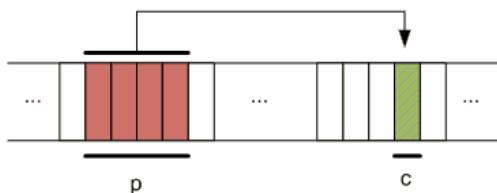
**Тип указателя** — это тип переменной, адрес которой он содержит.

Для работы с указателями в Си определены две операции:

- ✓ операция \* (звездочка) — позволяет получить значение объекта по его адресу - определяет значение переменной, которое содержится по адресу, содержащемуся в указателе;
- ✓ операция & (амперсанд) — позволяет определить адрес переменной.

Например,

```
char c; // переменная
char *p; // указатель
p = &c; // p = адрес c
```



Для указанного примера обращение к одним и тем же значениям переменной и адреса представлено в таблице

	Переменная	Указатель
Адрес	&c	p
Значение	c	*p

Два основных оператора для работы с указателями – это оператор & взятия адреса, и оператор \* разыменования.

**Пример 8.1** Изменить значение переменной с помощью указателя.

```

1 #include <conio.h>
2 #include <stdio.h>
3
4 void main() {
5     int A = 100;
6     int *p;
7
8     //Получаем адрес переменной A
9     p = &A;
10
11    //Выводим адрес переменной A
12    printf("%p\n", p);
13
14    //Выводим содержимое переменной A
15    printf("%d\n", *p);
16
17    //Меняем содержимое переменной A
18    *p = 200;
19
20    printf("%d\n", A);
21    printf("%d", *p);
22
23    getch();
24 }
```

Рассмотрим код внимательно, ещё раз.

В строке 5 была объявлена переменная с именем *A*. Она располагается по какому-то адресу в памяти. По этому адресу хранится значение 100.

В 6 строке создали указатель типа *int* и в 9 присваиваем адрес переменной *A*.

Теперь переменная *p* хранит адрес переменной *A*. Используя оператор \* мы получаем доступ до содержимого переменной *A*. В 18 строке изменяем содержимое переменной *A*.

После этого значение *A* также изменено, так как она указывает на ту же область памяти.

В результате на экран будет выведено:

200

200

**Пример 8.2** Вывести на экран размер выделенной памяти под различные статические типы, и под указатели различных типов данных.

```
1 #include <conio.h>
2 #include <stdio.h>
3
4 void main() {
5     int A = 100;
6     int *a = &A;
7     double B = 2.3;
8     double *b = &B;
9
10    printf("%d\n", sizeof(A));
11    printf("%d\n", sizeof(a));
12    printf("%d\n", sizeof(B));
13    printf("%d\n", sizeof(b));
14
15    getch();
16 }
```

Будет выведено

```
4
4
8
4
```

Несмотря на то, что переменные имеют разный тип и размер, указатели на них имеют один размер. Действительно, если указатели хранят адреса, то они должны быть целочисленного типа. Так и есть, указатель сам по себе хранится в переменной типа *size\_t* (а также *ptrdiff\_t*), это тип, который ведёт себя как целочисленный, однако его размер зависит от разрядности системы. В большинстве случаев разницы между ними нет. Зачем тогда указателю нужен тип?

### Арифметика указателей

Во-первых, указателю нужен тип для того, чтобы корректно работала операция разыменования (получения содержимого по адресу). Если указатель хранит адрес переменной, необходимо знать, сколько байт нужно взять, начиная от этого адреса, чтобы получить всю переменную.

Во-вторых, указатели поддерживают арифметические операции. Для их выполнения необходимо знать размер. Операция **+ N** сдвигает указатель вперёд на **N\*sizeof(тип)** байт.

Например, если указатель **int \*p;** хранит адрес **CC02**, то после **p += 10;** он будет хранить адрес **CC02 + sizeof(int)\*10 = CC02 + 28 = CC2A** (Все операции выполняются в шестнадцатиричном формате). Пусть мы создали указатель на начало массива. После этого мы можем "двигаться" по этому массиву, получая доступ до отдельных элементов.

```

1 #include <conio.h>
2 #include <stdio.h>
3
4 void main() {
5     int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
6     int *p;
7
8     p = A;
9
10    printf("%d\n", *p);
11    p++;
12    printf("%d\n", *p);
13    p = p + 4;
14    printf("%d\n", *p);
15
16    getch();
17 }

```

Обратите внимание, каким образом мы получили адрес первого элемента массива (строка 8).

Массив, по сути, сам является указателем, поэтому не нужно использовать оператор **&**. Мы можем переписать пример по-другому (**p = &A[0]**).

Получить адрес первого элемента и относительно него двигаться по массиву. Кроме операторов **+** и **-** указатели поддерживают операции сравнения. Если у нас есть два указателя **a** и **b**, то **a > b**, если адрес, который хранит **a**, больше адреса, который хранит **b**.

```

1 #include <conio.h>
2 #include <stdio.h>
3
4 void main() {
5     int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
6     int *a, *b;
7
8     a = &A[0];
9     b = &A[9];
10
11    printf("&A[0] == %p\n", a);
12    printf("&A[9] == %p\n", b);
13
14    if (a < b) {
15        printf("a < b");
16    } else {
17        printf("b > a");
18    }
19
20    getch();
21 }

```

Если же указатели равны, то они указывают на одну и ту же область памяти.

### Указатель на указатель

Указатель хранит адрес области памяти. Можно создать указатель на указатель, тогда он будет хранить адрес указателя и сможет обращаться к его содержимому. Указатель на указатель определяется как

**<тип> \*\*<имя>;**

Очевидно, ничто не мешает создать и указатель на указатель на указатель, и указатель на указатель на указатель на указатель и так далее. Это нам понадобится при работе с двумерными и многомерными массивами.

**Пример 8.3** Создание указателя на указатель.

```

1 #include <conio.h>
2 #include <stdio.h>
3
4 #define SIZE 10
5
6 void main() {
7     int A;
8     int B;
9     int *p;
10    int **pp;
11
12    A = 10;
13    B = 111;
14    p = &A;
15    pp = &p;
16
17    printf("A = %d\n", A);
18    *p = 20;
19    printf("A = %d\n", A);
20    *(*pp) = 30; //здесь скобки можно не писать
21    printf("A = %d\n", A);
22
23    *pp = &B;
24    printf("B = %d\n", *p);
25    **pp = 333;
26    printf("B = %d", B);
27
28    getch();
29 }

```

## Указатели и приведение типов

Так как указатель хранит адрес, можно доставать его до другого типа. Это может понадобиться, например, если мы хотим взять часть переменной, или если мы знаем, что переменная хранит нужный нам тип.

### Пример 8.4 Приведение типов через указатель

```

1 #include <conio.h>
2 #include <stdio.h>
3
4 #define SIZE 10
5
6 void main() {
7     int A = 10;
8     int *intPtr;
9     char *charPtr;
10
11     intPtr = &A;
12     printf("%d\n", *intPtr);
13     printf("-----\n");
14     charPtr = (char*)intPtr;
15     printf("%d ", *charPtr);
16     charPtr++;
17     printf("%d ", *charPtr);
18     charPtr++;
19     printf("%d ", *charPtr);
20     charPtr++;
21     printf("%d ", *charPtr);
22
23     getch();
24 }

```

В этом примере мы пользуемся тем, что размер типа *int* равен 4 байта, а *char* 1 байт. За счёт этого, получив адрес первого байта, можно пройти по остальным байтам числа и вывести их содержимое.

## NULL pointer - нулевой указатель

Указатель до инициализации хранит мусор, как и любая другая переменная. Но в то же время, этот "мусор" вполне может оказаться валидным адресом. Пусть, к примеру, у нас есть указатель. Каким образом узнать, инициализирован он или нет? В общем случае никак. Для решения этой проблемы был введён макрос **NULL** библиотеки **stdlib**. Принято при определении указателя, если он не инициализируется конкретным значением, делать его равным **NULL**.

```
int *ptr = NULL;
```

По стандарту гарантировано, что в этом случае указатель равен **NULL**, и равен нулю, и может быть использован как булево значение *false*. Хотя в зависимости от реализации **NULL** может и не быть равным 0 (в смысле, не равен нулю в побитовом представлении, как например, *int* или *float*).

Это значит, что в данном случае

```
int *ptr = NULL;
if (ptr == 0) {
    ...
}
```

вполне корректная операция, а в случае

```
int a = 0;
if (a == NULL) {
    ...
}
```

поведение не определено. То есть указатель можно сравнивать с нулём, или с **NULL**, но нельзя **NULL** сравнивать с переменной целого типа или типа с плавающей точкой.

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <conio.h>
4
5 void main() {
6     int *a = NULL;
7     unsigned length, i;
8
9     printf("Enter length of array: ");
10    scanf("%d", &length);
11
12    if (length > 0) {
13        //При выделении памяти возвращается указатель.
14        //Если память не была выделена, то возвращается NULL
15        if ((a = (int*) malloc(length * sizeof(int))) != NULL) {
16            for (i = 0; i < length; i++) {
17                a[i] = i * i;
18            }
19        } else {
20            printf("Error: can't allocate memory");
21        }
22    }
23
24    //Если переменная была инициализирована, то очищаем её
25    if (a != NULL) {
26        free(a);
27    }
28    getch();
29 }
```

**Пример 8.5** Пройдём по массиву и найдём все чётные элементы.

```

1 #include <conio.h>
2 #include <stdio.h>
3
4 void main() {
5     int A[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
6     int even[10];
7     int evenCounter = 0;
8     int *iter, *end;
9
10    //iter хранит адрес первого элемента массива
11    //end хранит адрес следующего за последним "элемента" массива
12    for (iter = A, end = &A[10]; iter < end; iter++) {
13        if (*iter % 2 == 0) {
14            even[evenCounter++] = *iter;
15        }
16    }
17
18    //Выводим задом наперёд чётные числа
19    for (--evenCounter; evenCounter >= 0; evenCounter--) {
20        printf("%d ", even[evenCounter]);
21    }
22
23    getch();
24}

```

**Пример 8.6** Когда мы сортируем элементы часто приходится их перемещать. Если объект занимает много места, то операция обмена местами двух элементов будет дорогостоящей. Вместо этого можно создать массив указателей на исходные элементы и отсортировать его. Так как размер указателей меньше, чем размер элементов целевого массива, то и сортировка будет происходить быстрее. Кроме того, массив не будет изменён, часто это важно.

```

1 #include <conio.h>
2 #include <stdio.h>
3
4 #define SIZE 10
5
6 void main() {
7     double unsorted[SIZE] = {1.0, 3.0, 2.0, 4.0, 5.0, 6.0, 8.0, 7.0, 9.0, 0.0};
8     double *p[SIZE];
9     double *tmp;
10    char flag = 1;
11    unsigned i;
12
13    printf("unsorted array\n");
14    for (i = 0; i < SIZE; i++) {
15        printf("%.2f ", unsorted[i]);
16    }
17    printf("\n");
18
19    //Сохраняем в массив p адреса элементов
20    for (i = 0; i < SIZE; i++) {
21        p[i] = &unsorted[i];
22    }

```

```

-- 24
25     do {
26         flag = 0;
27         for (i = 1; i<SIZE; i++) {
28             //Сравниваем СОДЕРЖИМОЕ
29             if (*p[i] < *p[i-1]) {
30                 //обмениваем местами АДРЕСА
31                 tmp = p[i];
32                 p[i] = p[i-1];
33                 p[i-1] = tmp;
34                 flag = 1;
35             }
36         } while(flag);
37
38         printf("sorted array of pointers\n");
39         for (i = 0; i < SIZE; i++) {
40             printf("%.2f ", *p[i]);
41         }
42         printf("\n");
43
44         printf("make sure that unsorted array wasn't modified\n");
45         for (i = 0; i < SIZE; i++) {
46             printf("%.2f ", unsorted[i]);
47         }
48
49         getch();
50     }

```

**Пример 8.7** Более интересный пример. Так как размер типа **char** всегда равен 1 байт, то с его помощью можно реализовать операцию **swap** – обмена местами содержимого двух переменных.

```

1 #include <conio.h>
2 #include <conio.h>
3 #include <stdio.h>
4
5 void main() {
6     int length;
7     char *p1, *p2;
8     char tmp;
9     float a = 5.0f;
10    float b = 3.0f;
11
12    printf("a = %.3f\n", a);
13    printf("b = %.3f\n", b);
14
15    p1 = (char*) &a;
16    p2 = (char*) &b;
17    //Узнаём сколько байт перемещать
18    length = sizeof(float);
19    while (length--) {
20        //Обмениваем местами содержимое переменных побайтно
21        tmp = *p1;
22        *p1 = *p2;
23        *p2 = tmp;
24        //не забываем перемещаться вперёд
25        p1++;
26        p2++;
27    }
28
29    printf("a = %.3f\n", a);
30    printf("b = %.3f\n", b);
31
32    getch();
33}

```

В этом примере можно поменять тип переменных *a* и *b* на **double** или любой другой (с соответствующим изменением вывода и вызова **sizeof**), всё равно мы будем обменивать местами байты двух переменных.

**Пример 8.8** Найдём длину строки, введённой пользователем, используя указатель

```

1 #include <conio.h>
2 #include <stdio.h>
3
4 void main() {
5     char buffer[128];
6     char *p;
7     unsigned length = 0;
8
9     scanf("%127s", buffer);
10    p = buffer;
11    while (*p != '\0') {
12        p++;
13        length++;
14    }
15
16    printf("length = %d", length);
17    getch();
18}

```

Обратите внимание на участок кода, строки 11-14. Его можно переписать одним из следующих способов:

11   <b>while</b> (*p != 0) { 12          p++; 13          length++; 14      }	11   <b>while</b> (*p) { 12          p++; 13          length++; 14      }	11   <b>while</b> (*p++) { 12          length++; 13      }
---	--	--

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## 6. Форма отчета о работе

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_

\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Что такое указатель?
2. Сколько памяти занимает указатель?
3. Для чего необходимо указывать тип для указателя?
4. Как объявить указатель?
5. Каким образом можно получить значение, хранимое в памяти, на которую ссылается указатель?
6. Как получить доступ к адресу, записанному в указатель?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ обработки  
многомерных массивов»

Минск  
2017

## **Лабораторная работа № 9**

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ обработки многомерных массивов»

### **1. Цель работы**

Научить применять многомерные массивы при решении задач различных видов и уровней сложности.

### **2. Задание**

Задачи первого и второго уровня обязательны для всех. Номер варианта соответствует вашему номеру по списку. Основное решение должно быть реализовано с помощью методов. В методе `main` производится ввод данных, вызов вспомогательных методов, и вывод ответа на экран. Обязательно реализовать методы для ввода и вывода массива.

#### **Задачи первого уровня**

1. Найдите произведение вектора на матрицу.
2. Даны две матрицы одинаковой размерности. Найдите сумму и разность этих матриц.
3. К данной матрице  $A$  порядка  $n$  добавьте нулевую и  $n + 1$ -ю строки с элементами, равными 1.
4. К данной матрице  $A$  порядка  $n$  добавьте нулевой и  $n + 1$ -й столбцы с элементами, равными 0.
5. Определите количество строк заданной матрицы, которые упорядочены по возрастанию.
6. Сложите две треугольные матрицы порядка  $n$ , у которых только элементы над главной диагональю отличны от нуля.
7. В данной матрице определите количество столбцов, у которых элементы расположены в порядке возрастания.
8. Данна матрица  $A_{m \times n}$ , содержащая оценки группы за первый семестр. Найдите количество «хорошистов» в группе (оценки не ниже шестерки, но не выше восьмерки).
9. Данна действительная квадратная матрица  $A$  порядка  $n$ . Найдите количество строк матрицы, сумма модулей элементов которых больше 1.
10. Данна квадратная матрица  $A$  порядка  $n$ . Найдите среднее арифметическое положительных элементов каждого столбца матрицы.
11. Данна квадратная матрица  $A$  порядка  $n$ . Найдите номер строки матрицы, в которой больше всего единичных элементов.
12. Данна квадратная матрица  $A$  порядка  $n$ . Проверьте, равны ли суммы элементов матрицы по строкам, столбцам, главной и побочной диагоналям между собой, т.е. является ли матрица магическим квадратом.
13. Данна квадратная матрица  $A$  порядка  $n$ . Определите норму заданной

матрицы. Одна из норм матрицы равна наибольшей из сумм модулей элементов, стоящих в одной строке.

14. Данна квадратная матрица  $A$  порядка  $n$ . Постройте вектор, элементы которого являются наибольшими числами каждой строки матрицы.
15. Данна квадратная матрица  $A$  порядка  $n$ . Найдите сумму положительных элементов матрицы, стоящих под главной диагональю.
16. Данна квадратная матрица  $A$  порядка  $n$ . Транспонируйте данную матрицу.
17. Данна квадратная матрица  $A$  порядка  $n$ . Проверьте, является ли данная матрица симметричной.
18. Данна квадратная матрица  $A$  порядка  $n$ . Проверьте, является ли матрица единичной.
19. Данна матрица  $A$ , имеющая  $n$  строк и  $m$  столбцов, содержащая оценки группы за первый семестр. Выведите номера отличников (оценки не ниже 8).
20. Данна матрица  $A$ , имеющая  $n$  строк и  $m$  столбцов, содержащая оценки группы за первый семестр. Определите средний балл каждого студента.
21. Данна матрица  $A$ , имеющая  $n$  строк и  $m$  столбцов, содержащая оценки группы за первый семестр. Найдите количество единиц, двоек и троек у каждого студента.
22. Данна матрица  $A$ , имеющая  $n$  строк и  $m$  столбцов, содержащая оценки группы за первый семестр. Определите средний балл студентов группы по каждому предмету.
23. Данна матрица  $A$ , имеющая  $n$  строк и  $m$  столбцов, содержащая оценки группы за первый семестр. Определите количество неуспевающих студентов (имеются оценки 0, 1, 2 или 3).
24. Если все элементы какой-либо строки данной матрицы равны между собой, то все элементы такой строки замените нулями.
25. Данна матрица  $A$ , имеющая  $n$  строк и  $m$  столбцов, содержащая оценки группы за первый семестр. Выведите номера предметов, по которым имеются нулевые оценки.
26. Данна матрица  $A$  порядка  $n$ . Определите количество строк матрицы, элементы которых представляют перестановки чисел от 1 до  $n$ .
27. Найдите наибольшую сумму модулей элементов строк заданной матрицы.
28. Найдите произведение матрицы на вектор.
29. Данна матрица  $A$  порядка  $n$ . Найдите наибольший среди отрицательных элементов матрицы.
30. Данна матрица  $A$  порядка  $n$ . Расставьте элементы строк с четными номерами матрицы в порядке убывания.

### **Задачи второго уровня**

1. Дан массив  $A$ , состоящий из  $n$  натуральных чисел. Найдите элемент массива, сумма цифр которого наибольшая.

- Дана действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 9.1).
- Дана действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 9.2).
- Дана действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 9.3).

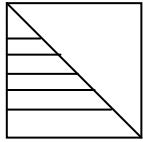


Рис. 9.1

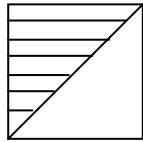


Рис. 9.2

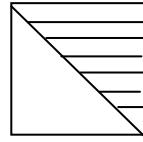


Рис. 9.3

- Даны две последовательности  $a_1, a_2, \dots, a_n$  и  $b_1, b_2, \dots, b_n$  целых чисел. Среди членов каждой последовательности нет повторяющихся чисел. Получите все члены последовательности  $b_1, b_2, \dots, b_n$ , которые не входят в последовательность  $a_1, a_2, \dots, a_n$ .
- Дана действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 9.4).
- Дана действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 9.5).
- Дана действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 9.6).

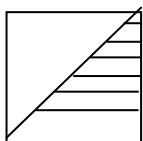


Рис. 9.4

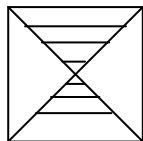


Рис. 9.5

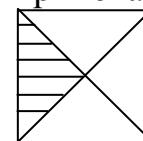


Рис. 9.6

- Дана действительная квадратная матрица порядка  $2n$  (рис. 9.7). Цифрами обозначены подматрицы порядка  $n$ .
- Дана матрица  $A$  порядка  $n$ . Поменяйте местами наибольший и наименьший элементы матрицы.
- Дана матрица  $A$  порядка  $n$ . Расставьте элементы строк с четными номерами матрицы в порядке убывания.
- Дана матрица  $A$  порядка  $n$ . Найдите два наибольших элемента матрицы с указанием номеров строк и столбцов, в которых они находятся.
- Дана матрица  $A$  порядка  $n$ . Поменяйте местами строки: первую с последней, вторую с предпоследней и т.д.
- Дана матрица  $A$  порядка  $n$ . Отсортируйте строки матрицы в порядке возрастания наибольших элементов в каждой строке.
- Дана квадратная матрица  $A$  порядка  $n$ . Найдите седловую точку матрицы, т.е. элемент, который является наименьшим в своей строке и наибольшим в своем столбце.
- Дана квадратная матрица  $A$  порядка  $n$ . Найдите суммы элементов тех строк матрицы, на главной диагонали которой стоят отрицательные элементы.

17. Данна матрица  $A$  порядка  $n$ . Расставьте элементы каждой строки в порядке возрастания.
18. Данна матрица  $A$  порядка  $n$ . Расставьте строки матрицы в порядке возрастания количества нулевых элементов.
19. Данна матрица  $A$  порядка  $n$ . Удалите строки, содержащие нулевые элементы.
20. Данна действительная квадратная матрица порядка  $2n$ . Получите новую матрицу (рис. 9.8).
21. Данна действительная квадратная матрица порядка  $2n$ . Получите новую матрицу (рис. 9.9).
22. Данна действительная квадратная матрица порядка  $2n$ . Получите новую матрицу (рис. 9.10).
23. Данна действительная квадратная матрица порядка  $2n$ . Получите новую матрицу (рис. 9.11).

1	2
2	3
3	4

Рис. 9.7

3	4
2	1

Рис. 9.8

1	4
2	3

Рис. 9.9

4	3
1	2

Рис. 9.10

1	2
4	3

Рис. 9.11

24. Даны две последовательности  $a_1, a_2, \dots, a_n$  и  $b_1, b_2, \dots, b_n$  целых чисел. Среди членов каждой последовательности нет повторяющихся чисел. Постройте пересечение последовательностей (т.е. получите в каком-нибудь порядке все числа, принадлежащие обеим последовательностям одновременно).
25. Даны две последовательности  $a_1, a_2, \dots, a_n$  и  $b_1, b_2, \dots, b_n$  целых чисел. Среди членов каждой последовательности нет повторяющихся чисел. Постройте объединение данных последовательностей (равные члены разных последовательностей должны входить только один раз).
26. Постройте  $n$  строк треугольника Паскаля (рис. 9.12).

1
1    1
1    2    1
1    3    3    1

Рис. 9.12

27. Постройте матрицу порядка  $n$ , записав в нее числа от 1 до  $n^2$  согласно схеме (рис. 9.13).

$$\left( \begin{array}{ccccc} 1 & 2 & 3 & \dots & n \\ 2n & 2n-1 & 2n-2 & \dots & n+1 \\ 2n+1 & \dots & & & 3n \\ \vdots & & & & \\ n^2-n+1 & \dots & & & n^2 \end{array} \right)$$

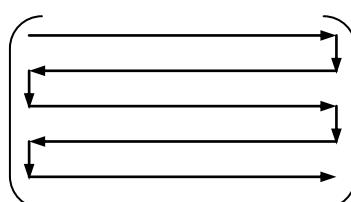


Рис. 9.13

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

Вспомните, что из себя представляет адрес любого человека (или фирмы). Вы можете сказать: "Я живу на Невском проспекте в доме 20 в квартире 45", но никому в голову не придет сказать: "Я живу в 13678 квартире от начала Невского проспекта". Почему? Потому что неудобно. Первая часть адреса – улица, вторая – дом, третья – квартира.

Часто обычный массив неудобен для хранения данных из-за своей линейной структуры. Например, пусть нам надо обрабатывать информацию о количестве выпавших осадков за несколько лет, причем известны осадки по месяцам. Если обрабатывать данные вручную, то удобнее всего использовать таблицу - по горизонтали откладывать годы, а по вертикали - месяцы (или наоборот). В принципе в программе можно использовать и одномерный массив, но тогда требуется пересчитывать индексы – по известному году и месяцу получать смещение ячейки от начала линейного массива:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
янв	фев	март	апр	май	июнь	июль	авг	сен	окт	ноя	дек	янв	фев	март	апр	май	июнь	
2008												2009						

Для такого пересчета можно было бы использовать формулу

$$i = (\text{год} - 2008) * 12 + \text{месяц} - 1;$$

где **i** – индекс нужного элемента массива. Тем не менее, так не делают (почти никогда), потому что для работы с табличными данными во всех современных языках программирования существуют *двухмерные массивы* или *матрицы*.

**Матрица** – это прямоугольная таблица элементов (например, чисел или символов). В информатике матрица представляется в виде двухмерного массива, то есть массива, все элементы которого имеют два индекса.

Матрица, как и таблица, состоит из строк и столбцов. Два индекса элемента – это и есть номера строки и столбца, на пересечении которых этот элемент находится.



В языке Си каждый индекс записывается отдельно в квадратных скобках. Каждую строку и каждый столбец матрицы можно рассматривать как обычный одномерный массив. Поэтому можно сказать, что матрица – это *массив из массивов*. Существует только два ограничения:

- ✓ все элементы матрицы должны быть **одинакового типа**;
- ✓ все строки матрицы должны быть одинаковой длины.

Первый индекс элемента матрицы – это *строка*, второй – *столбец*. Поэтому когда говорят о «матрице 4 на 5», это означает, что матрица имеет 4 строки и 5 столбца.

И Матрицы, у которых число строк равно числу столбцов, называют *квадратными*. В квадратных матрицах можно выделить *главную диагональ* – это все элементы, у которых номер строки равен номеру столбца, то есть

**A[0][0], A[1][1], ..., A[N-1][N-1]**

для матрицы размером **N** на **N**.

Матрицы объявляются также, как и простые массивы, но у них не один индекс, а два. При объявлении в отдельных квадратных скобках указывается количество строк и количество столбцов. Например, оператор

**int Bmv[20][10];**

выделит место в памяти под матрицу целых чисел, имеющую 20 строк и 10 столбцов (всего 200 элементов). Если матрица глобальная (объявляется выше всех процедур и функций), то она в самом начале заполняется нулями. Локальные матрицы (объявленные внутри процедуры или функции) содержат «мусор» – неизвестные значения.

При объявлении можно сразу задать все или часть ее элементов, например так

**float X[2][3] = {{1., 2., 3.}, {4., 5., 6.}};**

Как видно из примера, элементы каждой строки заключаются в отдельные фигурные скобки.

Если задать не все элементы, то остальные заполняются нулями:

**float X[2][3] = {{1., 3.}, {6.}};**

Здесь элементы **X[1][2]**, **X[2][1]** и **X[2][2]** будут нулевыми.

Иногда бывает полезно знать, как матрицы располагаются в памяти ЭВМ. Оказывается во всех современных языках программирования (кроме **Фортрана**) элементы матрицы располагаются *по строкам*, то есть сначала изменяется последний индекс. Объявленная выше матрица **X** расположена так:

<b>X[0][0]</b>	<b>X[0][1]</b>	<b>X[0][2]</b>	<b>X[1][0]</b>	<b>X[1][1]</b>	<b>X[1][2]</b>
----------------	----------------	----------------	----------------	----------------	----------------

Как и для одномерных массивов, матрицы могут быть введены с клавиатуры, из файла, и заполнены с помощью случайных чисел. Общий принцип – для каждого элемента функция чтения вызывается отдельно. Представим себе матрицу как массив строк равной длины. Для ввода одной строки требуется цикл, и таких строк несколько. Поэтому для работы с матрицами требуется *двойной* или *вложенный* цикл, то есть цикл в цикле.

## Пример 9.1 Заполнение двумерного массива с клавиатуры.

```
#include <stdio.h>
const int M = 5; // число строк
const int N = 4; // число столбцов
main()
{
    int i, j, A[M][N];
    for ( i = 0; i < M; i ++ )           // цикл по строкам
        for ( j = 0; j < N; j ++ )       // цикл по столбцам строки
        {
            printf ("A[%d][%d]=", i, j); // подсказка для ввода
            scanf ("%d", & A[i][j]);     // ввод A[i][j]
        }
    // работа с матрицей
}
```

Заметьте, что при изменении порядка циклов (если поменять местами два оператора `for`) изменится и порядок ввода элементов в память.

Выполняется также в двойном цикле аналогично одномерным массивам. В примере показано заполнение целой матрицы случайными числами в интервале `[a, b]` (для вещественных чисел формула изменится – см. одномерные массивы). Функция

```
int random ( int N ) { return rand() % N; }
```

возвращающая случайное целое число в интервале `[0, N-1]`, была рассмотрена выше, когда мы говорили о массивах (ее нужно добавить в программу).

В этой и следующей программах мы будем считать, что объявлена целая матрица `M` на `N`, где `M` и `N` — целые константы (объявленные через `const`), а также целые переменные `i` и `j`.

```
for ( i = 0; i < M; i ++ )
    for ( j = 0; j < N; j ++ )
        A[i][j] = random(b-a+1) + a;
```

При выводе матрицы ее элементы желательно расположить в привычном виде – по строкам. Напрашивается такой прием: вывели одну строку матрицы, перешли на новую строку экрана, и т.д. Надо учитывать, что для красивого вывода на каждый элемент матрицы надо отвести равное количество символов (иначе столбцы будут неровные). Делается это с помощью форматирования – цифра после знака процента задает количество символов, отводимое на данное число.

## Пример 9.2 Вывод содержимого двумерного массива на экран.

```
printf("Матрица A\n");
for ( i = 0; i < M; i ++ ) {           // цикл по строкам
    for ( j = 0; j < N; j ++ )         // вывод одной строки (в цикле)
        printf ( "%4d", A[i][j] );      // 4 символа на число
    printf("\n");                      // переход на другую строку
}
```

В отличие от одномерных массивов, для перебора всех элементов матрицы надо использовать двойной цикл. Ниже показано, как найти минимальный элемент в массиве и его индексы. Сначала считаем, что

минимальным является элемент **A[0][0]** (хотя можно начать и с любого другого), а затем проходим все элементы, проверяя, нет ли где еще меньшего. Так же, как и для одномерного массива, запоминаются только индексы, а значение минимального элемента «вытаскивается» прямо из массива.

**Пример 9.3** Нахождение минимального значения в матрице.

```

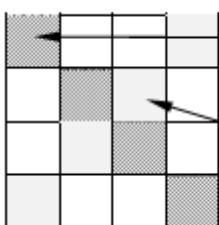
float A[M][N], i, j, row, col;
...
row = col = 0; // сначала считаем, что A[0][0] - минимальный
for ( i = 0; i < M; i ++ ) // просмотр всех строк
    for ( j = 0; j < N; j ++ ) // просмотр всех столбцов
        if ( A[i][j] < A[row][col] ) {
            row = i;           // запомнили новые индексы
            col = j;
        }
printf ("Минимальный элемент A[%d][%d]=%d",
       row, col, A[row][col]);

```

Рассмотрим квадратную матрицу **N** на **N**. Выведем на экран обе ее диагонали (главную диагональ и перпендикулярную ей). С главной диагональю все просто – в цикле выводим все элементы, у которых номера строки и столбца равны, то есть **A[i][i]** для всех **i** от 0 до **N-1**. Вторую диагональ формируют такие элементы:

**A[0][N-1], A[1][N-2], A[2][N-3], ..., A[N-1][0]**

Обратим внимание, что каждый следующий элемент имеет номер строки на 1 больше, а номер столбца – на 1 меньше. Таким образом, **сумма номеров строки и столбца постоянна** и равна **N-1**. Тогда, зная номер строки **i** можно сразу сказать, что на второй диагонали стоит ее элемент **A[i][N-1-i]**.



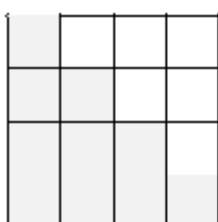
```

printf("Главная диагональ:\n");
for ( i = 0; i < N; i ++ )
    printf ("%d ", A[i][i]);

printf("\nВторая диагональ:\n");
for ( i = 0; i < N; i ++ )
    printf ("%d ", A[i][N-1-i]);

```

Теперь можно рассмотреть более сложный случай. Обнулим все элементы, кроме тех, которые стоят выше главной диагонали. В первой строке это единственный элемент **A[0][0]**, во второй – два элемента: **A[1][0]** и **A[1][1]**, в третьей – три: **A[2][0]**, **A[2][1]** и **A[2][2]**. Заметим, что в строке **i** обнуляются все элементы, у которых номера столбцов от 0 до **i**.



```

for ( i = 0; i < N; i ++ )
    for ( j = 0; j < i; j ++ )
        A[i][j] = 0;

```

Еще более сложно обработать все элементы, стоящие не выше второй диагонали. В первой строке это единственный элемент **A[0][N-1]**, во второй – два элемента: **A[1][N-2]** и **A[1][N-1]**, в третьей – три: **A[2][N-3]**, **A[2][N-2]** и **A[2][N-1]**. Заметим, что в строке **i** обнуляются все элементы, у которых номера столбцов от **N-1-i** до **N-1**.


```
for ( i = 0; i < N; i ++ )
    for ( j = N-1-i; j < N; j ++ )
        A[i][j] = 0;
```

Пусть надо переставить две строки с индексами **i1** и **i2**. Это значит, что для каждого столбца **j** надо поменять местами элементы **A[i1][j]** и **A[i2][j]** через временную переменную (она называется **temp**).

```
for ( j = 0; j < N; j ++ ) {
    temp = A[i1][j];
    A[i1][j] = A[i2][j];
    A[i2][j] = temp;
}
```

Иногда надо скопировать матрицу **A** размером **M** на **N** в одномерный массив **B** размером **M\*N**. Очевидно, что при копировании по строкам (сначала первая строка, затем вторая и т.д.) элемент первой строки **A[0][j]** надо скопировать в **B[j]**, элементы второй строки **A[1][j]** в **B[N+j]** и т.д. Отсюда следует, что для любой строки **i** элемент **A[i][j]** копируется в **B[i\*N+j]**. Теперь осталось только в двойном цикле перебрать все элементы матрицы.

```
for ( i = 0; i < M; i ++ )
    for ( j = 0; j < N; j ++ )
        B[i*N+j] = A[i][j];
```

Заметим, что если надо провести какую-то операцию со всеми или некоторыми (стоящими подряд) элементами матрицы и для одномерного массива уже есть соответствующая процедура, ее можно использовать, учитывая, что имя строки массива (например, **A[0]**) является указателем на начальный элемент этой строки. При этом надо учитывать, что в памяти элементы матрицы расположены по строкам. Например, функция вычисления суммы элементов (см. массивы) может применяться для матрицы так:

```
s0 = Sum(A[0], N);      // сумма строки 0
s14 = Sum(A[1], 4*N);   // сумма строк 1-4
sAll = Sum(A[0], M*N); // сумма всех строк
```

## **5. Порядок выполнения работы**

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_  
\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Как задается размер многомерного массива?
2. Каким образом двумерный массив задается в памяти компьютера?
3. Как инициализируются многомерные массивы?
4. Сколько памяти занимает  $N$ -мерный массив конкретного типа?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ обработки строк»

Минск  
2017

# **Лабораторная работа № 10**

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ обработки строк»

## **1. Цель работы**

Научить создавать строки и выполнять над ними типичные операции.

## **2. Задание**

Разработайте программу решения задачи с использованием процедур и функций над строками и операций над множествами.

Общая задача для всех: построить таблицу аски.

Номер варианта соответствует вашему номеру по списку.

1. Для каждого слова заданного предложения укажите долю согласных.

Определите слово, в котором доля согласных максимальна.

2. Найдите самое длинное симметричное слово заданного предложения и укажите номер позиции, с которого оно начинается.

3. Даны строка, представляющая собой запись числа в десятеричной системе счисления. Преобразуйте ее в строку, представляющую собой запись числа в восьмеричной системе счисления.

4. Даны строка, представляющая собой запись числа в восьмеричной системе счисления. Преобразуйте ее в строку, представляющую собой запись числа в двоичной системе счисления.

5. Даны строка символов, состоящая из произвольных десятичных чисел, разделенных пробелами. Выведите на экран числа этой строки в порядке возрастания их значений.

6. Даны строка, состоящая из групп нулей и единиц, разделенных пробелами. Найдите и выведите на экран самую короткую группу.

7. Даны строка, состоящая из групп нулей и единиц. Каждая группа отделяется от другой одним или несколькими пробелами. Подсчитайте количество символов в самой длинной группе.

8. Даны строка, состоящая из групп нулей и единиц. Каждая группа отделяется от другой одним или несколькими пробелами. Найдите и выведите на экран группы с четным количеством символов.

9. Даны строка, состоящая из групп нулей и единиц. Каждая группа отделяется от другой одним или несколькими пробелами. Подсчитайте количество нулей и единиц в группах с нечетным количеством символов.

10. Даны строка символов, состоящая из натуральных чисел, разделенных пробелами. Выведите четные числа этой строки.

11. Даны строка, представляющая собой запись числа в двоичной системе счисления. Преобразуйте ее в строку, представляющую собой запись числа в шестнадцатеричной системе счисления.

**12.** Данна строка символов, состоящая из произвольного текста, слова разделены пробелами. Выведите на экран порядковый номер слова, накрывающего k-ю позицию (если на k-ю позицию попадает пробел, то номер предыдущего слова), и найдите в нем количество повторяющихся символов.

**13.** Данна строка символов, состоящая из произвольного текста, слова разделены пробелами. Разбейте исходную строку на две подстроки, причем первая длиной k-символов (если на k-ю позицию попадает слово, то его следует отнести ко второй строке, дополнив первую пробелами до k-позиций).

**14.** Данна строка символов, состоящая из произвольного текста, слова разделены пробелами. Выведите на экран порядковый номер слова максимальной длины и номер позиции строки с которой оно начинается.

**15.** Данна строка символов, состоящая из произвольного текста, слова разделены пробелами. Выведите на экран порядковый номер слова минимальной длины и количество неповторяющихся символов в этом слове.

**16.** Найдите самое длинное симметричное слово заданного предложения и укажите номер позиции, с которой оно начинается.

**17.** Напишите программу, вычеркивающую из данного слова все буквы «а» так, чтобы, например, из слова «заноза» получилось «зноз».

**18.** Напишите программу, проверяющую, является ли частью данного слова слово «сок».

**19.** Напишите программу, подсчитывающую, сколько раз в данном слове встречается сочетание «со».

**20.** Напишите программу, заменяющую в тексте все прописные латинские буквы на строчные.

**21.** Напишите программу, заменяющую в тексте все строчные латинские буквы на прописные.

**22.** Данна строка символов, состоящая из произвольного текста на английском языке, слова отделены пробелами. После каждого гласного символа вставьте символ «\*».

**23.** Выведите все строчные гласные латинские буквы, встречающиеся в данной строке ровно один раз.

**24.** В заданном тесте найдите количество четырехбуквенных слов и каждое четное из них замените на сочетание «SsSs». Слова отделены друг от друга пробелом.

**25.** Отредактируйте предложение, удаляя из него лишние пробелы, оставляя по одному пробелу между словами.

**26.** В заданном предложении укажите слово, в котором доля гласных «А» максимальна.

**27.** Проверьте, имеется ли в заданном тексте баланс открывающих и закрывающих скобок, имея в виду, что балансом, например, будет комбинация (...), в то время как комбинация )..(..)..( балансом не является.

**28.** Выведите на экран все символы кодовой таблицы ПЭВМ.

**29.** Введите слово и все буквы «О» заменить в нем на «А».

**30.** Напечатайте в столбик числовые коды букв введенного слова.

31. Напишите программу подсчета количества гласных в тексте на английском языке.
32. Напишите программу обращения слова.
33. Напишите программу замены в тексте всех букв «а» на «о» и наоборот.

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

Понятно, что символьная строка – это последовательность символов. Мы будем рассматривать строки, в которых на каждый символ отводится 1 байт. В этом случае можно использовать  $2^8=256$  различных символов. Каждый символ имеет свой код (от 0 до 255), эти коды определяются по специальной таблице.

Строка, как и другие переменные, записывается в память, причем компьютеру все равно, какие данные записаны – для него это набор байтов. Как же определить, где заканчивается строка? Есть два решения:

- ✓ хранить длину строки в отдельной ячейке (как в языке Паскаль);
- ✓ выбрать один особый символ, который будет обозначать конец строки, причем в середине строки этот символ не может встречаться.

В языке Си принят второй подход.

**Символьная строка** – это последовательность символов, которая заканчивается **символом с кодом 0**.

Символ с кодом ноль не имеет никакого изображения, в программе его записывают как '`\0`'.

Символ с кодом ноль (обозначается как '`\0`') и цифра ноль (обозначается '`0`', имеет код 48) это два разных символа.

Строка представляет собой массив символов, поэтому и объявляется она именно как массив:

```
char s[80];
```

Однако строка отличается от массива тем, что она заканчивается символом с кодом 0 – признаком окончания строки, поэтому: если массив символов будет использоваться как строка, надо выделять на 1 байт больше памяти.

При выделении памяти глобальные переменные заполняются нулями, а локальные содержат «мусор». Начальное значение строки можно задать при объявлении в двойных кавычках после знака равенства:

```
char s[80] = "Привет, Вася!";
```

Символы в кавычках будут записаны в начало массива **s**, а затем – признак окончания строки '`\0`'. Оставшиеся символы не меняются, и в локальных строках там будет «мусор». Можно написать и так

```
char s[] = "Привет, Вася!";
```

В этом случае компилятор подсчитает символы в кавычках, выделит памяти на 1 байт больше и занесет в эту область саму строку и завершающий ноль. Аналогично можно выделить память на указатель:

```
char *s = "Привет, Вася!";
```

Результат – тот же самый, что и в предыдущем случае, но теперь **s** – это указатель (переменная, в которой хранится адрес ячейки в памяти), и с ним можно работать так же, как с обычным указателем (присваивать, изменять и т.п.). Если строка не будет изменяться во время работы программы, то можно объявить константу (постоянную строку) так:

```
const char PRIVET[] = "Привет, Вася!";
```

Для ввода и вывода строк с помощью функций **scanf** и **printf** используется специальный формат "%s":

```
#include <stdio.h>
main()
{
    char Name[50];
    printf("Как тебя зовут? ");
    scanf("%s", Name);
    printf("Привет, %s!", Name);
}
```

Заметьте, что в функцию **scanf** надо передать просто имя строки (без знака **&**), ведь имя массива является одновременно адресом его начального элемента.

Однако у функции **scanf** есть одна особенность: она заканчивает ввод, встретив первый пробел. Если вы на вопрос в предыдущем примере ввели "**Вася Пупкин**", то увидите надпись "**Привет, Вася!**" вместо ожидаемого "**Привет, Вася Пупкин!**". Если надо ввести всю строку целиком, включая пробелы (то есть до нажатия на клавишу **Enter**), придется делать иначе, заменив вызов **scanf** на более простой:

```
gets ( s );
```

Название этой функции происходит от английских слов *get string* – получить строку.

Для вывода строки на экран можно (кроме **printf**) использовать и функцию **puts**, которая после вывода строки еще и дает команду перехода на новую строку. В примере значение строки **Name** будет напечатано на следующей строчке экрана.

```
#include <stdio.h>
main()
{
    char Name[50] = "Вася!";
    puts( "Привет," );
    puts ( Name );
}
```

**Пример 10.1** Ввести символьную строку и заменить в ней все буквы '**А**' на буквы '**Б**'.

Будем рассматривать строку как массив символов. Надо перебрать все элементы массива, пока мы не встретим символ '\0' (признак окончания строки) и, если очередной символ – это буква '**А**', заменить его на '**Б**'. Для этого используем цикл **while**, потому что мы заранее не знаем длину строки. Условие цикла можно сформулировать так: «пока не конец строки».

```
#include <stdio.h>
main()
{
    char a[80];
    int i;

    printf( "\n Введите строку \n" );
    gets ( s );

    i = 0; // начать с первого символа, s[0]
    while ( s[i] != '\0' ) // пока не достигли конца строки
    {
        if ( s[i] == 'A' ) // если очередной символ – 'А', ...
            s[i] = 'B'; // меняем его на 'Б'
        i++; // переходим к следующему символу
    }
    puts ( "Результат:\n" );
    puts ( a );
}
```

Заметьте, что одиночный символ записывается в апострофах, а символьная строка – в кавычках.

При выводе строк с помощью функции **printf** часто применяется форматирование. После знака % в формате указывается размер поля для вывода строки. Перед этим числом можно также поставить знак минус, что означает «прижать к левому краю поля».

Пример вывода	Результат	Комментарий
printf( "[%s]", "Вася");	[Вася]	Минимальное число позиций.
printf( "%6s", "Вася");	[ Вася]	6 позиций, выравнивание вправо.
printf( "%-6s", "Вася");	[Вася ]	6 позиций, выравнивание влево.
printf( "%2s", "Вася");	[Вася]	Строка не помещается в заданные 2 позиции, поэтому область вывода расширяется.

В языке Си есть достаточно много специальных функций, которые работают со строками – последовательностями символом с нулевым символом на конце. Для использования этих функций надо включить в программу заголовочный файл

```
#include <string.h>
```

Многие из этих функций достаточно опасны при неправильном использовании, ведь они не проверяют, достаточно ли выделено памяти для

копирования, перемещения или другой операции, единственным признаком окончания строки для них является символ '\0'.

### Длина строки – `strlen`

Это самая простая функция, которая определяет, сколько символов в переданной ей строке (не считая '\0'). Ее имя происходит от английских слов *string length* (длина строки).

```
#include <stdio.h>
#include <string.h>
main()
{
    int len;
    char s[] = "Prodigy";
    len = strlen(s);
    printf ("Длина строки %s равна %d", s, len );
}
```

В этом примере функция определит, что длина строки равна 7. Теперь рассмотрим более сложную задачу.

**Пример 10.2** В текстовом файле `input.dat` записаны строки текста. Вывести в файл `output.dat` в столбик длины этих строк.

```
#include <stdio.h>
#include <string.h>
main()
{
    char s[80];
    FILE *fin, *fout;
    fin = fopen ( "input.dat", "r" );
    fout = fopen ( "output.dat", "w" );
    while ( NULL != fgets(s, 80, fin) ) // читаем строку s
    {
        fprintf(fout, "%d\n", strlen(s)); // выводим ее длину в файл
    }
    fclose ( fin );
    fclose ( fout );
}
```

Несмотря на то, что с первого взгляда программа написана верно, числа в файле будут на единицу больше, чем длины строк (кроме последней строки). Далее будет показано, как получить точный результат.

### Сравнение строк – `strcmp`

Для сравнения двух строк используют функцию `strcmp` (от английских слов *string comparison* – сравнение строк). Функция возвращает ноль, если строки равны (то есть «разность» между ними равна нулю) и ненулевое значение, если строки различны. Сравнение происходит по кодам символов, поэтому функция различает строчные и заглавные буквы – они имеют разные коды.

```

#include <stdio.h>
#include <string.h>
main()
{
    char s1[] = "Вася",
        s2[] = "Петя";
    if ( 0 == strcmp(s1,s2) )
        printf("Строки %s и %s одинаковы", s1, s2);
    else printf("Строки %s и %s разные", s1, s2);
}

```

Если строки не равны, функция возвращает «разность» между первой и второй строкой, то есть *разность кодов первых различных символов*. Эти числа можно использовать для сортировки строк – если «разность» отрицательна , значит первая строка «меньше» второй, то есть стоит за ней в алфавитном порядке. В таблице показано несколько примеров (код буквы 'A' равен 65, код буквы 'B' – 66, код буквы 'C' – 67).

s1	s2	результат strcmp(s1, s2)
AA	AA	0
AB	AAB	'B' – 'A' = 66 – 65 = 1
AB	CAA	'A' – 'C' = 65 – 67 = -2
AA	AAA	'\0' – 'A' = -65

**Пример 10.3** Ввести две строки и вывести их в алфавитном порядке.

```

#include <stdio.h>
#include <string.h>
main()
{
    char s1[80], s2[80];
    printf ("Введите первую строку");
    gets(s1);
    printf ("Введите вторую строку");
    gets(s2);

    if ( strcmp(s1,s2) <= 0 )
        printf("%s\n%s", s1, s2);
    else printf("%s\n%s", s2, s1);
}

```

Иногда надо сравнить не всю строку, а только первые несколько символов. Для этого служит функция **strncpy** (с буквой **n** в середине). Третий параметр этой функции – количество сравниваемых символов. Принцип работы такой же – она возвращает нуль, если заданное количество первых символов обеих строк одинаково.

```

#include <stdio.h>
#include <string.h>
main()
{
    char s1[80], s2[80];
    printf ("Введите первую строку");
    gets(s1);
    printf ("Введите вторую строку");
    gets(s2);
    if ( 0 == strncmp(s1, s2, 2) )
        printf("Первые два символа %s и %s одинаковы", s1, s2);
    else
        printf("Первые два символа %s и %s разные", s1, s2);
}

```

Один из примеров использования функции **strcmp** – проверка пароля. Составим про-граммму, которая спрашивает пароль и, если пароль введен неверно, заканчивает работу, а если верно – выполняет какую-нибудь задачу.

**Пример 10.4** Составить программу, которая определяет, сколько цифр в символьной строке. Программа должна работать только при вводе пароля «куку».

```

#include <stdio.h>
#include <string.h>
main()
{
    char pass[] = "куку", // правильный пароль
          s[80];           // вспомогательная строка
    int i, count = 0;

    printf ("Введите пароль ");
    gets(s);
    if ( strcmp ( pass, s ) != 0 )
    {
        printf ( "Неверный пароль" );
        return 1;           // выход по ошибке, код ошибки 1
    }

    printf ("Введите строку");
    gets(s);

    i = 0;
    while ( s[i] != '\0' ) {
        if ( s[i] >= '0' && s[i] <= '9' )
            count++;
    }
    printf("\nНашли %d цифр", count);
}

```

В этой программе использован тот факт, что коды цифр расположены в таблице символов последовательно от '**'0'**' до '**'9'**'. Поэтому можно использовать двойное неравенство, а не сравнивать текущий символ **s[i]** с каждой из цифр. Обратите внимание на разницу между символами '**\0**' (символ с кодом 0, признак конца строки) и '**'0'**' (символ с кодом 48, цифра 0). Переменная **count** работает как счетчик.

## Копирование строк

Часто надо записать новое значение в строку или скопировать информацию из одной строки в другую. Функции копирования принадлежат к числу «опасных» – они могут вызвать серьезную ошибку, если произойдет **выход за границы массива**. Это бывает в том случае, если строка, в которую копируется информация, имеет недостаточный **размер** (под нее выделено мало места в памяти).

В копировании участвуют две строки, они называются «источник» (строка, откуда копируется информация) и «приемник» (куда она записывается или добавляется).

При копировании строк надо проверить, чтобы для строки-приемника было выделено достаточно места в памяти.

Простое копирование выполняет функция **strcpy**. Она принимает два аргумента: сначала строка-приемник, потом – источник (порядок важен!).

```
char s1[50], s2[10];
gets(s1);
strcpy ( s2, s1); // s2 (приемник) <- s1 (источник)
puts ( s2 );
```

Этот фрагмент программы является «опасным» с точки зрения выхода за границы строки. В строку **s1** можно безопасно записать не более 49 символов (плюс завершающий ноль). Поэтому если с клавиатуры будет введена строка длиннее 49 символов, при записи ее в память произойдет выход за границы строки **s1**. Стока **s2** может принять не более 9 символов, поэтому при большем размере **s1** произойдет выход за границы строки **s2**.

Поскольку реально функции передается адрес начала строки, можно заставить функцию начать работу любого символа, а не только с начала строки. Например, следующая строка скопирует строку **s2** в область памяти строки **s1**, которая начинается с ее 6-ого символа, оставив без изменения первые пять:

```
strcpy ( s1+5, s2 );
```

При этом надо следить, чтобы не выйти за границу массива. Кроме того, если до выполнения этой операции в строке **s1** меньше 5 символов, фокус не удастся.

Еще одна функция позволяет скопировать только заданное количество символов, она называется **strncpy** и принимает в третьем параметре количество символов, которые надо скопировать. Важно помнить, что эта функция **НЕ записывает завершающий нуль**, а только копирует символы (в отличие от нее **strcpy** всегда копирует завершающий нуль). Функция **strncpy** особенно полезна тогда, когда надо по частям собрать строку из кусочков.

```

#include <stdio.h>
#include <string.h>
main()
{
    char s1[] = "Ку-ку", s2[10];
    strncpy ( s2, s1, 2 ); // скопировать 2 символа из s1 в s2
    puts ( s2 );           // ошибка! нет последнего '\0'
    s2[2] = '\0';          // добавляем символ окончания строки
    puts (s2);             // вывод
}

```

При копировании стандартные функции **strcpy** и **strncpy** поступают так: определяют количество символов, которые надо скопировать и затем переписывают их, начиная с первого символа до последнего. Если области источника и приемника не перекрываются, то все проходит нормально. Но попробуем «раздвинуть» строку, например для того, чтобы вставить что-то в ее середину. Пусть в строке **s1** записано имя и фамилия человека, а в строке **s2** – его отчество. Надо получить в строке **s1** полностью имя, фамилию и отчество.

```

#include <stdio.h>
#include <string.h>
main()
{
    int n;
    char s1[80] = "Иван Рождественский",
         s2[] = "Петрович ";
    n = strlen(s2);           // длина второй строки
    strcpy (s1+5+n, s1+5);   // пытаемся раздвинуть на n символов
    strncpy(s1+5, s2, n);    // вставляем отчество в середину
    puts ( s1 );
}

```

При первом вызове **strcpy** мы хотели скопировать конец строки, начиная с символа с номером 5 (он шестой с начала, так как нумерация идет с нуля) вправо на **n** символов (объявленная длина массива символов – 80 – позволяет это сделать). Однако из-за того, что копирование выполнялось с начала блока данных, скопировав на новое место первый символ фамилии ('Р') функция стерла букву 'н' (справа на 9 символов) и т.д. В результате в строке **s1** получаем

"Иван Рождественский..."

Многоточие обозначает, что при копировании стирается завершающий символ с кодом 0, который обозначает окончание строки, и копирование продолжается бесконечно, пока программа не завершится аварийно из-за неверного обращения к памяти.

Таким образом, вся задумка не удалась из-за того, что функция копирования работает в данном случае неверно. Выход из этой ситуации такой – написать свою функцию копирования, которая копирует не с начала блока, а с конца (однако она будет неверно работать в обратной ситуации – при сжатии строки). Например, так:

```

void strcpy1 ( char s1[], char s2[] )
{
    int n = strlen(s2);
    while ( n >= 0 )
    {
        s1[n] = s2[n];
        n--;
    }
}

```

Заметьте, что завершающий нуль строки **s2** также копируется. Если использовать в нашем примере эту функцию вместо **strcpy**, то получим желаемый результат.

Возникает вопрос: можно ли сделать функцию, которая всегда правильно копирует? Конечно, можно, хотя это и не просто. Для этого в самой функции надо использовать вспомогательную строку (ее называют *буфером*) и в ней формировать результат так, чтобы в процессе копирования не портилась исходная строка. Когда результат в буфере готов, его останется скопировать в то место, где была исходная строка и удалить память, выделенную под буфер. Попробуйте написать такую функцию, если известно, что ее будут использовать для копирования строк длиной не более 80 символов.

### Объединение строк

Еще одна функция – **strcat** (от *string concatenation* – сцепка строк) позволяет добавить строку источник в конец строки-приемника (завершающий нуль записывается автоматически). Надо только помнить, что приемник должен иметь достаточный размер, чтобы вместить обе исходных строки. Функция **strcat** автоматически добавляет в конец строки-результата завершающий символ '\0'.

```

#include <stdio.h>
#include <string.h>
main()
{
    char s1[80] = "Могу, ";
    s2[] = "хочу, ", s3[] = "надо!";
    strcat ( s1, s2 ); // дописать s2 в конец s1
    puts ( s1 ); // "Могу, хочу,
    strcat ( s1, s3 ); // дописать s3 в конец s1
    puts ( s1 ); // "Могу, хочу, надо!"
}

```

Заметьте, что если бы строка **s1** была объявлена как **s1[]** (или с длиной меньше 18), произошел бы выход за границы массива с печальными последствиями.

**Пример 10.5** Ввести с клавиатуры имя файла. Изменить его расширение на ".exe".

Алгоритм решения:

- ✓ найти в имени файла точку '.' или признак конца строки '\0';

- ✓ если нашли точку, скопировать начиная с этого места новое расширение ".exe" (используем функцию **strcpy**);
- ✓ если нашли конец строки (точки нет), добавить в конец расширение ".exe" (используем функцию **strcat**).

```
#include <stdio.h>
#include <string.h>
main()
{
    char s[80];
    int n;          // номер символа '.'
    printf("Введите имя файла ");
    gets ( s );

    n = 0;
    while ( (s[n] != '.')      // ищем первую точку
            && (s[n] != '\0') )   // или конец строки
        n++;

    if ( s[n] == '.' )         // если нашли точку, то...
        strcpy ( s+n, ".exe" ); // меняем расширение,
    else strcat ( s, ".exe" ); // иначе добавляем
    puts ( s );
}
```

**Пример 10.6** Ввести с клавиатуры фамилию и имя в одну строку (допустим, "Иванов Вася"). Построить в памяти строку «Привет, Вася Иванов!».

Алгоритм решения этой задачи можно записать так:

- ✓ Ввести строку с клавиатуры (назовем ее **s**).

**s** И в а н о в    в а с я \0

- ✓ Найти длину первого слова (фамилии), обозначим ее через **n**. Тогда символ с номером **n** – это пробел между именем и фамилией.

**s** И в а н о в    в а с я \0

- ✓ Запишем в новую строку первую часть фразы – «Привет, », используя функцию **strcpy**.

**a** П р и в е т , \0

- ✓ Добавим в конец этой строки второе слово (имя), используя функцию **strcat**. Для этого надо скопировать в конец строки **a** все символы строки **s**, начиная с **n+1**-ого:

**a** П р и в е т ,    в а с я \0

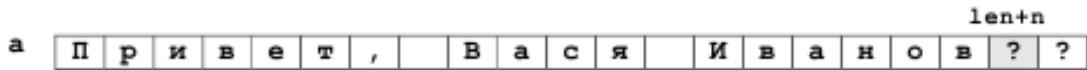
- ✓ Добавим в конец строки пробел, используя функцию **strcat**.

**a** П р и в е т ,    в а с я \0

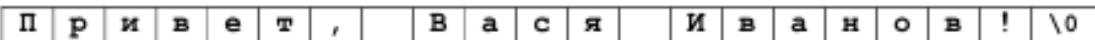
- ✓ Определим длину полученной строки и обозначим ее через **len**. Символ с номером **len** – это символ с кодом '\0'.

**a** П р и в е т ,    в а с я \0

- ✓ Добавим в конец строки первое слово (фамилию), используя функцию **strncpy**. Для этого скопируем в конец строки **a** первые **n** символов строки **s**. Обратите внимание, что в конце строки теперь нет завершающего символа '\0'.

**a**  **len+n**

- ✓ Осталось добавить в конец строки символ '!' и '\0'. Для этого используется функция **strcpy**. Для копирования нужно использовать адрес **s+len+n**, так как к строке длиной **len** мы добавили фамилию длиной **n**.



В программе используется важное свойство массивов в языке Си (в том числе и символьных строк, которые являются массивами символов): если массив называется **s**, то запись **s+i** обозначает адрес элемента **s[i]** (так же, как и **&s[i]**).

Так как функциям **strcpy** и **strcat** надо передать адреса в памяти, куда и откуда переместить данные, мы можем использовать запись **a+len** вместо **&a[len]** и т.д.

```
#include <stdio.h>
#include <string.h>
main()
{
    char s[80], a[80] = "Привет, ";
    int n, len;
    printf("Введите фамилию и имя ");
    gets ( s );

    n = 0;
    while ( (s[n] != ' ')      // ищем первый пробел
            && (s[n] != '\0') ) // или конец строки
        n++;
    if ( s[n] != ' ' ) { // если нет пробела, ...
        printf( "Неверная строка" );
        return 1; // выход по ошибке, код ошибки 1
    }
    strcat ( a, s+n+1 );           // добавить имя
    strcat ( a, " " );           // добавить пробел
    len = strlen ( a );          // найти длину строки
    strncpy ( a + len, s, n );   // добавить фамилию
    strcpy ( a + len + n, "!" ); // добавить "!"

    puts ( a );
}
```

## Поиск в строках

Когда говорят о поиске в строках, обычно рассматривают две задачи: найти первый заданный символ с начала (или с конца), или также найти заданную подстроку (если она есть). Первую задачу выполняют функции **strchr** (поиск с начала строки) и  **strrchr** (поиск с конца строки), а вторую – функция **strstr**.

Все эти функции возвращают указатель на найденный символ (или на первый символ найденной подстроки). Это значит, что переменная, в которую записывается это значение, должна быть объявлена как **указатель** на символьную переменную. Мы уже встречались с указателями, когда работали с файлами. Указатель – это ячейка в памяти, в которую можно записывать *адрес* другой переменной.

Структура вызова функций такая: на первом месте – где искать (строка), на втором – что искать (один символ для функций **strchr** и **strrchr** или строка для **strstr**). Чтобы получить номер символа с начала строки, надо вычесть из полученного указателя адрес начала массива. Если поиск завершился неудачно, функции возвращают **NULL**.

```
#include <stdio.h>
#include <string.h>
main()
{
    char s1[] = "Мама мыла раму",
         s2[] = "Война и мир", *p;

    p = strchr(s1, 'а');
    if ( p != NULL ) {
        printf("Первая буква а: номер %d", p - s1);
        p = strrchr(s1, 'а');
        printf("\nПоследняя буква а: номер %d", p - s1);
    }

    p = strstr( s2, "мир");
    if ( p != NULL )
        printf("\nНашли мир в %s", s2);
    else printf("\nНет слова мир в %s", s2);
}
```

Вспомните, что при чтении строк из файла с помощью функции **fgets** на конце иногда остается символ перехода на новую строку '**\n**'. Чаще всего он совсем не нужен и надо его удалить – поставить на его месте нуль (признак конца строки). Делается это так:

```
char s[80], *p;
...
p = strrchr (s, '\n'); // ищем символ '\n'
if ( p != NULL )        // если нашли, ...
    *p = '\0';           // записываем в это место '\0'
```

**Пример 10.7** С клавиатуры вводится предложение и слово. Надо определить, сколько раз встречается это слово в предложении.

Функция **strstr** может определить только первое вхождение слова в строку, поэтому в одну строчку эту задачу не решить.

Попробуем использовать такую идею: если мы нашли адрес первого данного слова в строке и записали его в указатель **p**, то искать следующее слово нужно не сначала строки, а с адреса **p+длина\_слова**. Повторяем эту операцию в цикле, пока функция **strstr** может найти слово в оставшейся части строки. Поскольку начало области поиска постоянно смещается с

каждым новым найденным словом, адрес оставшейся части надо хранить в отдельной переменной типа «указатель на символ». Реализация может выглядеть так:

```
#include <stdio.h>
#include <string.h>
main()
{ int len, count;
  char s[80], word[20],
        *p,           // указатель на найденное слово
        *start;        // указатель на начало зоны поиска
  puts ( "Введите предложение" );
  gets ( s );
  puts ( "Введите слово для поиска" );
  gets ( word );
  len = strlen ( word ); // находим длину слова
  count = 0; // счетчик найденных слов
  start = s; // в первый раз ищем с начала строки
  while ( 1 ) {
    p = strstr ( start, word ); // есть ли еще слова?
    if ( p == NULL ) break; // если нет, то выход
    count++;
    start = p + len; // увеличить счетчик
                      // сместили начало поиска
  }
  printf ( "В этом предложении %d слов %s", count, word );
}
```

В конце работы цикла в переменной **count**, будет записано количество данных слов в предложении. Заметьте, что вместо переменной **start** можно везде использовать **p**, результат от этого не изменится.

### Форматирование строк

В программах часто требуется перед выводом информации сформировать всю строку для вывода целиком, включив в нее все необходимые данные. Например, сообщение об ошибке выводится стандартной функцией, и в это сообщение надо включить числовые данные. Другой пример – вывод текста в графическом режиме, для которого нет аналога функции **printf**.

В этих случаях необходимо использовать функцию **sprintf**, которая поддерживает те же форматы данных, что и **printf**, но записывает результат не на экран и не в файл, а в *символьную строку* (под нее надо заранее выделить память). Вот как выглядит вывод на экран значения переменных **x** и **y** в графическом режиме:

```

#include <stdio.h>
#include <conio.h>
#include <graphics.h>
main()
{
    char s[80]; // вспомогательная строка
    int x, y;
        // здесь нужно открыть окно для графики
    x = 1;
    y = 5;
    sprintf (s, "X=%d, Y=%d", x, y); // вывод в строку s
    outtextxy ( 100, 100, s );           // вывод строки s на экран
    getch();
    closegraph();
}

```

Не забудьте, что для использования функции **outtextxy** надо открыть окно для работы с графикой (с помощью функции **initwindow**).

### Чтение из строки

Иногда, особенно при чтении данных из файлов, возникает обратная задача: есть символьная строка, в которой записаны данные. Необходимо ввести их в соответствующие ячейки памяти.

В этом случае используется функция **sscanf**, которая читает данные по указанному формату не с клавиатуры (как **scanf**) и не из файла (как **fscanf**), а из символьной строки. В приведенном ниже примере мы ищем в файле строчку, которая начинается с символа **#** и считываем из нее значения **x** и **y**.

Сложность задачи заключается в том, что мы не знаем точно, какая по счету эта строчка в файле. Если не использовать функцию **sscanf**, то пришлось бы сначала найти номер нужной строки в файле, затем начать просмотр с начала, отсчитать нужное количество строк и использовать **fscanf**.

```

#include <stdio.h>
main()
{
    char s[80]; // вспомогательная строка
    int x, y;
    FILE *fp;
    fp = fopen ( "input.dat", "r" );
    while ( fgets ( s, 80, fp ) )
        if ( s[0] == '#' ) { // если строка начинается с #, ...
            sscanf ( s+1, "%d%d", &x, &y ); // читаем данные
            break;                      // выход из цикла
        }
    fclose ( fp );
    printf ( "x = %d, y = %d", x, y );
}

```

Стандартные и ваши собственные функции могут принимать строки в качестве параметров. Здесь, однако, есть некоторые тонкости.

Вы знаете, что если параметр функции объявлен как `int a`, то изменение переменной `a` в функции никак не влияет на ее значение в вызывающей программе – функция создает копию переменной и работает с ней (для того, чтобы функция могла это сделать, надо объявить параметр как `int &a`). Это относится ко всем простым типам.

Когда вы передаете в функция или процедуру строку, вы в самом деле передаете *адрес начала строки*, никаких копий строки не создается. Поэтому всегда надо помнить, что при изменении строки-параметра в функции или процедуре меняется и соответствующая строка в основной программе.

Как же объявить строку-параметр в своей функции? В простейшем варианте – так же, как и массив символов: `char s[]`. А можно также и как указатель (все равно в функцию передается адрес строки `char *s`). Между этими двумя способами есть некоторая разница. В первом случае `s` – это имя массива символов, поэтому нельзя его изменять, а во втором случае – указатель на символ, который можно сдвигать, как хочется.

Все стандартные функции языка Си объявляют символьные строки как указатели – это дает большую свободу действий. Сравните, например, две реализации функции, которая копирует строки (аналогично `strcpy`). Первая выполнена с помощью параметра-массива, а вторая – с помощью указателя.

```
void copy1 ( char s1[],  
            char s2[] )  
{  
    int i = 0;  
    while ( s2[i] ) {  
        s1[i] = s2[i];  
        i++;  
    }  
    s1[i] = '\0';  
}
```

```
void copy2 ( char *s1, char *s2 )  
{  
    while ( *s1++ = *s2++ );  
}
```

- 1) скопировать `*s2` по адресу `s1`  
2) увеличить `s1` и `s2`  
3) делать так пока `*s2` не ноль

Как видите, вторая функция получилась более компактной. Применение указателей позволило не вводить дополнительную переменную, хотя и сделала программу менее ясной. Итак, в условии цикла `while` стоит оператор присваивания. Если не обращать внимания на плюсы, он означает «взять символ по адресу `s2` и записать его по адресу `s1`». Двойные плюсы ПОСЛЕ `s1` и `s2` означают, что ПОСЛЕ выполнения присваивания оба **указателя** надо увеличить на единицу, то есть перейти к следующему символу.

Что же является условием цикла? Оказывается условие – это величина `*s1`, то есть код символа по адресу `s1`. Когда же происходит проверка? Это зависит от расположения знаков `++`.

В данном случае они стоят ПОСЛЕ имен переменных, поэтому операция инкремента выполняется ПОСЛЕ проверки условия. Проверка выполняется так: скопировали очередной символ, посмотрели на него, и если он – ноль (признак конца строки), то вышли из цикла. После этого увеличили указатели `s1` и `s2`. Обратите внимание, что после выхода из цикла увеличение указателей также

происходит, и они будут указывать не на нули, завершающие строки, а на следующие байты в памяти.

### **5. Порядок выполнения работы**

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

### **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

### **7. Контрольные вопросы и задания**

1. Как объявляется строковая переменная?
2. Сколько памяти занимает строковая переменная?
3. Как символом заканчивается любая строковая переменная?
4. С помощью каких команд можно занести данные с клавиатуры в строку?

### **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.

7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ с использованием  
внутренних методов сортировки»

Минск  
2017

# **Лабораторная работа № 11**

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ с использованием внутренних методов сортировки»

## **1. Цель работы**

Научить применять внутренние методы сортировки.

## **2. Задание**

Номер варианта соответствует номеру по списку.

1. Отсортировать строки массива целых чисел по убыванию. Сортировка включением.
2. Отсортировать столбцы массива целых чисел по возрастанию. Шейкерная сортировка.
3. Отсортировать нечетные столбцы массива по возрастанию. Сортировка прямой выбор.
4. Отсортировать элементы нечетных строк массива целых чисел по убыванию. Сортировка разделением.
5. Отсортировать элементы квадратной вещественной матрицы размерности  $n$ , применив сортировку бинарным включением.
6. Отсортировать элементы квадратной вещественной матрицы размерности  $n$ , применив пузырьковую сортировку.
7. Отсортировать положительные элементы одномерного массива, отрицательные оставить на местах. Пузырьковая сортировка.
8. Отсортировать строки массива целых чисел по убыванию. Шейкерная сортировка.
9. Отсортировать столбцы массива целых чисел по возрастанию. Сортировка включением.
10. Отсортировать нечетные столбцы массива по возрастанию. Сортировка разделением.
11. Отсортировать элементы нечетных строк массива целых чисел по убыванию. Сортировка прямой выбор.
12. Отсортировать элементы квадратной вещественной матрицы размерности  $n$ , применив пузырьковую сортировку слева направо.
13. Заданный одномерный массив отсортировать по возрастанию цифры десятков каждого элемента. Сортировка прямой выбор.
14. Отсортировать элементы квадратной вещественной матрицы размерности  $n$ , стоящие на побочной диагонали, применив сортировку бинарным включением.
15. Заданный одномерный массив отсортировать по возрастанию цифры десятков каждого элемента. Сортировка включением.
16. В одномерном массиве упорядочить отрицательные элементы, оставив положительные на местах. Сортировка включением.

17. В одномерном массиве упорядочить нечетные элементы, оставив четные на местах. Сортировка прямой выбор.
18. Упорядочить одномерный массив так, чтобы в начале располагались четные элементы в порядке возрастания их значений, а затем нечетные – в порядке убывания их значений.
19. В одномерном массиве упорядочить нечетные элементы, оставив четные на местах. Сортировка шейкерная.
20. Отсортировать положительные элементы одномерного массива, отрицательные оставить на местах. Сортировка прямой выбор.
21. Отсортировать строки массива целых чисел по убыванию. Шейкерная сортировка.
22. Отсортировать столбцы массива целых чисел по возрастанию. Сортировка включением.
23. Отсортировать нечетные столбцы массива по возрастанию. Сортировка разделением.
24. Отсортировать элементы нечетных строк массива целых чисел по убыванию. Сортировка прямой выбор.
25. Отсортировать элементы квадратной вещественной матрицы размерности  $n$ , стоящие на главной диагонали, применив пузырьковую сортировку слева направо.
26. Заданный одномерный массив отсортировать по возрастанию цифры десятков каждого элемента. Сортировка прямой выбор.
27. Отсортировать элементы квадратной вещественной матрицы размерности  $n$ , стоящие на побочной диагонали, применив сортировку бинарным включением.
28. Заданный одномерный массив отсортировать по возрастанию цифры десятков каждого элемента. Сортировка включением.
29. В одномерном массиве упорядочить отрицательные элементы, оставив положительные на местах. Сортировка включением.
30. В одномерном массиве упорядочить нечетные элементы, оставив четные на местах. Сортировка прямой выбор.

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

**Сортировка** – процесс упорядочения множества объектов по заданному признаку.

Обычно сортировку подразделяют на два класса: **внутреннюю** и **внешнюю**. При внутренней сортировке все элементы хранятся в оперативной памяти, таким образом, как правило, это сортировка массивов. При внешней

сортовке — элементы хранятся на внешнем запоминающем устройстве, это сортировка содержимого файлов.

Одно из основных требований к методам сортировки — экономное использование памяти. Это означает, что переупорядочение нужно выполнять «на том же месте», то есть методы пересылки элементов из одного массива в другой не представляют интереса.

Сортировать массив можно:

- ✓ **по возрастанию** — каждый следующий элемент больше предыдущего:  $a[1] < a[2] < \dots < a[n]$ ;
- ✓ **по неубыванию** — каждый следующий элемент не меньше предыдущего:  $a[1] \leq a[2] \leq \dots \leq a[n]$ ;
- ✓ **по убыванию** — каждый следующий элемент меньше предыдущего:  $a[1] > a[2] > \dots > a[n]$ ;
- ✓ **по невозрастанию** — каждый следующий элемент не больше предыдущего:  $a[1] \geq a[2] \geq \dots \geq a[n]$ .

Рассмотрим некоторые методы внутренней сортировки.

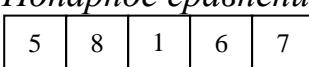
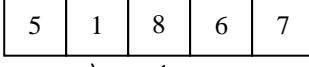
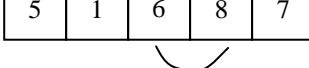
### Метод «пузырька» — метод простого обмена

Производится последовательное упорядочение смежных пар элементов массива:  $x[1]$  и  $x[2]$ ,  $x[2]$  и  $x[3]$ , ...  $x[N-1]$  и  $x[N]$ . Если в паре первый элемент больше второго, то элементы меняются местами. Далее, таким же образом, обрабатываем следующую пару. В итоге, после  $N-1$  сравнения максимальное значение переместится на место элемента  $X[N]$ , т.е. "вверху" окажется самый "легкий" элемент — отсюда аналогия с пузырьком. Следующий проход делается аналогичным образом до второго сверху элемента ( $X[N-1]$ ), в результате второй по величине элемент поднимется на правильную позицию и т.д. Для сортировки всего массива нужно выполнить  $N-1$  проход по массиву. При первом прохождении нужно сравнить  $N-1$  пар элементов, при втором —  $N-2$  пары, при  $k$ -м прохождении —  $N-k$  пар.

Например, дан массив из 5 элементов: 5 8 1 6 7. Упорядочить его по возрастанию элементов.

1 прохождение:  $5-1=4$  сравнения (см. Таблица 11.1. Трассировка методом пузырька. Первый проход по массиву).

**Таблица 11.1. Трассировка методом пузырька. Первый проход по массиву.**

	<i>Попарное сравнение элементов массива</i>	<i>Действие</i>
1.		Нет обмена
2.		$1 \leftrightarrow 8$
3.		$6 \leftrightarrow 8$

4.

5	1	6	7	8
			7↔8	

2 прохождение:  $5-2=3$  сравнения

5 1 6 7 8

3 прохождение:  $5-3=2$  сравнения

1 5 6 7 8

4 прохождение:  $5-4=1$  сравнение

1 5 6 7 8

```
void sort_bubble (int *m, int n) // Сортировка пузырьком
{
    int i, j, t;
    for (i=0; i<n-1; i++)
        for (j=0; j<n-i-1; j++)
            if (m[j] > m[j+1])
            {
                t=m[j];
                m[j]=m[j+1];
                m[j+1]=t;
            }
}
```

Среднее число сравнений и обменов имеют квадратичный порядок роста:  $O(n^2)$ , отсюда можно заключить, что алгоритм пузырька очень медленен и малоэффективен.

### Модифицированный «пузырек»

При сортировке методом простого обмена («пузырька») возможна ситуация, когда на каком-либо из проходов не произошло ни одного обмена. Это значит, что все пары расположены в правильном порядке, так что массив уже отсортирован. И продолжать процесс не имеет смысла (особенно, если массив был отсортирован с самого начала).

Можно улучшить эффективность этого метода, просматривая массив только до тех пор, пока существует обмен между элементами. Как только при очередном просмотре не происходит ни одного обмена между элементами, это означает, что массив отсортирован. При таком подходе может потребоваться один просмотр массива, если массив уже был отсортирован.

Поэтому лучше использовать «модифицированный пузырек», который отсортирует за один просмотр.

```
void modif_bubble(int *m, int n) // Модифицированный
пузырек
{
    int i=n; // Длина неотсортированной части
 массива
```

```

int f, k, buf;
do {
    f=0;      //Предположим, что массив является
отсортированным
    for (k=0;k<i-1;k++)
        if (m[k]>m[k+1])
    {
        // Обмен
        buf =m[k];
        m[k]=m[k+1];
        m[k+1]= buf;
        f=1;      // Массив был неотсортированным
    }
    i--;
} while (f && i>1);
}

```

### Шейкер-сортировка

Использование «модифицированного пузырька» отсортирует заданный массив за 5 просмотров ( $n-1$ ). УстраниТЬ такое «неравноправие» можно путем смены направлений просмотров, т.е первоначально в направлении слева направо получаем 5, 7, 9, 10, 3, 12, а затем справа налево результат – 3, 5, 7, 9, 10, 12.

Этот алгоритм уменьшает количество перемещений, действуя следующим образом: за один проход из всех элементов выбирается минимальный и максимальный, минимальный элемент помещается в начало массива, а максимальный, соответственно, в конец. Далее алгоритм выполняется для остальных данных.

```

void sort_sheiker (int *m, int n) // Шейкер-сортировка
{
    int left=1;                      // левая граница
    int right=n-1;                   // правая граница
    int j,buf,k=n-1;
    do
    {
        // Обратный проход "Пузырька" от правой границы до
левой
        for (j=right;j>=left; j--)
            if (m[j-1]>m[j])
            {
                buf =m[j-1];
                m[j-1]=m[j];
                m[j]= buf;
            }
    }
}

```

```

        k=j;      // фиксирование индекса последнего
        обмена
    }
left=k+1;           // Левая граница
//Прямой проход "Пузырька" от левой границы до правой
for (j=left; j<=right; j++)
    if (m[j-1]>m[j])
    {
        buf =m[j-1];
        m[j-1]=m[j];
        m[j]= buf;
        k=j;      // фиксирование индекса последнего
        обмена
    }
right=k-1;          // правая граница
}   while (left<=right); // До тех пор, пока левая
граница не //станет больше правой границы
}

```

Фиксирование индексов последнего обмена при проходах слева направо и справа налево ускоряет «шейкер»-сортировку.

### Сортировка вставками (метод прямого включения)

Алгоритм имеет следующий вид:

- ✓ Начиная со 2-го элемента, каждый текущий элемент с номером  $i$  запоминается в промежуточной переменной.
- ✓ Затем просматриваются предыдущие  $i-1$  элемент и ищется место вставки  $i$ -го элемента таким образом, чтобы не нарушить упорядоченности, при этом все элементы, превышающие  $i$ -й, сдвигаются вправо.
- ✓ На освободившееся место вставляется  $i$ -й элемент.

**Таблица 11.2. Пример сортировки прямыми включениями.**

$i=2$	5	4	3	1
$i=3$	4	5	3	1
$i=4$	3	4	5	1
Отсортированный массив	1	3	4	5

```

void sort_insert (int *m, int n)
{
    int j, i, r;
    for (i=1;i<n;i++)
    {

```

```

        r=m[i]; // Запоминаем текущий элемент в
промежуточной переменной
        j=i-1;
        while (j>=0 && m[j]>r) // Ищем новое место
вставки,
        { m[j+1]=m[j]; j--; } // сдвигая на 1 элемент
вправо
        m[j+1]=r; // На освободившееся место
вставляется элемент
    }
}

```

### **Сортировка бинарными вставками**

Алгоритм сортировки простыми включениями можно легко улучшить, пользуясь тем, что готовая последовательность  $a[1], \dots, a[i-1]$ , в которую нужно включить новый элемент, уже упорядочена. Поэтому место включения можно найти значительно быстрее, применив бинарный поиск, который исследует средний элемент готовой последовательности и продолжает деление пополам, пока не будет найдено место включения. Модифицированный алгоритм сортировки называется *сортировкой бинарными включениями*.

```

void sort_bin_insert (int *a, int n) // Сортировка
бинарными //вставками
{ int i,j,x,left,right,sred;
  for (i=1; i<n; i++)
  {
    if (a[i-1]>a[i])
    {
      x=a[i]; // x - включаемый элемент
      left=0; // левая граница отсортированной
части //массива
      right=i-1; // правая граница отсортированной части
//массива
      do {
        sred = (left+right)/2; // sred - новая "середина"
//последовательности
        if (a[sred]<x) left= sred+1;
        else right=sred-1;
      } while (left<=right); // поиск ведется до тех пор,
пока //левая граница не окажется правее правой границы
      for (j=i-1; j>=left;j--) a[j+1]= a[j];
      a[left]= x;
    }
  }
}

```

}

## Сортировка методом простого выбора

Обычно применяется для массивов, не содержащих повторяющихся элементов.

Алгоритм:

- ✓ выбрать максимальный элемент массива;
- ✓ поменять его местами с последним элементом (после этого самый большой элемент будет стоять на своём месте);
- ✓ повторить пункты 1-2 с оставшимися  $n-1$  элементами.

Порядок сортировки показан в таблице

**Таблица 11.3. Сортировка массива по возрастанию методом простого выбора.**

	Элементы массива					Примечание
Исходный массив	5	2	7	4	6	max=7
Шаг 1	5	2	<b>6</b>	4	7	$7 \leftrightarrow 6$ ; max=6
Шаг 2	<b>5</b>	2	4	6	7	$4 \leftrightarrow 6$ ; max=5
Шаг 3	<b>4</b>	2	5	6	7	$4 \leftrightarrow 5$ ; max=4
Шаг 4	2	4	5	6	7	$4 \leftrightarrow 2$

```
void sort_vybor (int *a, int n) // Сортировка простым
выбором
{
    int i, j, k, m;
    for (i=n-1; i>0; i--)
    {
        k=i;                                // Запоминаем номер и
        m=a[i];                             // значение текущего
элемента
        for (j=0; j<i; j++)                  // Поиск максимального
элемента
            if (a[j]>m) {k=j; m=a[k];}
        if (k!=i)
        {
            // Меняем местами с
последним
            a[k]=a[i];
            a[i]=m;
        }
    }
}
```

## Сортировка Шелла

При сортировке Шелла сначала сравниваются и сортируются между собой значения, стоящие один от другого на некотором расстоянии  $d$  (о выборе значения  $d$ ). После этого процедура повторяется для некоторых меньших значений  $d$ , а завершается сортировка Шелла упорядочиванием элементов при  $d = 1$  (то есть обычной сортировкой вставками). Эффективность сортировки Шелла в определённых случаях обеспечивается тем, что элементы «быстрее» встают на свои места (в простых методах сортировки, например, пузырьковой, каждая перестановка двух элементов уменьшает количество инверсий в списке максимум на 1, а при сортировке Шелла это число может быть больше).

Невзирая на то, что сортировка Шелла во многих случаях медленнее, чем быстрая сортировка, она имеет ряд преимуществ:

- отсутствие потребности в памяти под стек;
- отсутствие деградации при неудачных наборах данных — быстрая сортировка легко деградирует до  $O(n^2)$ , что хуже, чем худшее гарантированное время для сортировки Шелла.

Пусть дан список  $A = (32, 95, 16, 82, 24, 66, 35, 19, 75, 54, 40, 43, 93, 68)$  и выполняется его сортировка методом Шелла, а в качестве значений  $d$  выбраны 5, 3, 1.

На первом шаге сортируются подсписки  $A$ , составленные из всех элементов  $A$ , отличающихся на 5 позиций, то есть подсписки  $A_{5,1} = (32, 66, 40)$ ,  $A_{5,2} = (95, 35, 43)$ ,  $A_{5,3} = (16, 19, 93)$ ,  $A_{5,4} = (82, 75, 68)$ ,  $A_{5,5} = (24, 54)$ .

В полученном списке на втором шаге вновь сортируются подсписки из отстоящих на 3 позиции элементов.

Процесс завершается обычной сортировкой вставками получившегося списка.

Среднее время работы алгоритма зависит от длин промежутков —  $d$ , на которых будут находиться сортируемые элементы исходного массива ёмкостью  $N$  на каждом шаге алгоритма.

```
void ShellSort(int data[], int rzm_data)
{ int step,i,j,c;
  step = rzm_data; // Шаг поисков и вставки
  do
  { // Вычисляем новый шаг
    step = step / 3 + 1;
    // Производим сортировку простыми вставками с
    заданным шагом
    int k=0;
    while (k<step) //сортируем с шагом step
последовательности, первый элемент которых имеет номер от
0 до step-1
```

```

{
    for      (i=k+step;      i<rzm_data;      i=i+step)
//перемещаемся      с      шагом      step      по      теккущей
последовательности
    {
        c = data[i];
        j = i-step;
        while (j >= 0 && data[j] > c) //ищем место
вставки элемента, если он меньше предыдущего
        {   data[j+step] = data[j];
            j = j-step;
        }
        data[j+step] = c; //выполняем вставку
    }
    k++; //переходим к следующей последовательности
для сортировки
}
}

while (step !=1);
}

```

## **5. Порядок выполнения работы**

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Дайте определение понятиям «поиск» и «сортировка».
2. Что такое ключ и основные требования к нему?
3. Назовите принципы действия сортировки выбором.
4. Назовите принципы действия обменной сортировки. Приведите примеры.
5. Назовите принципы действия шейкерной сортировки.
6. Назовите принципы действия сортировки вставками.

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ обработки  
динамических массивов»

Минск  
2017

## Лабораторная работа № 12

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ обработки динамических массивов»

### 1. Цель работы

Отработать навык обработки динамических массивов.

### 2. Задание

Вариант соответствует номеру по списку. Задачи необходимо решить с помощью динамических массивов.

28. Дан массив  $A$ , состоящий из  $n$  натуральных чисел. Найдите элемент массива, сумма цифр которого наибольшая.
29. Данна действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 12.1).
30. Данна действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 12.2).
31. Данна действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 12.3).

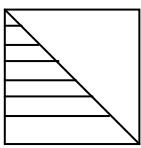


Рис. 12.1

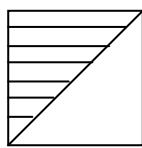


Рис. 12.2

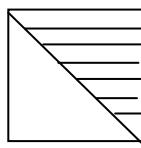


Рис. 12.3

32. Даны две последовательности  $a_1, a_2, \dots, a_n$  и  $b_1, b_2, \dots, b_n$  целых чисел. Среди членов каждой последовательности нет повторяющихся чисел. Получите все члены последовательности  $b_1, b_2, \dots, b_n$ , которые не входят в последовательность  $a_1, a_2, \dots, a_n$ .
33. Данна действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 12.4).
34. Данна действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 12.5).
35. Данна действительная квадратная матрица порядка  $n$ . Найдите сумму элементов, расположенных в заштрихованной части матрицы (рис. 12.6).

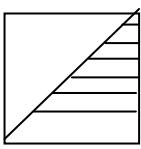


Рис. 12.4

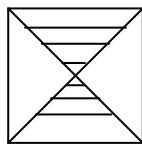


Рис. 12.5

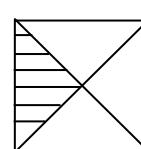


Рис. 12.6

36. Данна действительная квадратная матрица порядка  $2n$  (рис. 12.7). Цифрами обозначены подматрицы порядка  $n$ .
37. Данна матрица  $A$  порядка  $n$ . Поменяйте местами наибольший и наименьший элементы матрицы.
38. Данна матрица  $A$  порядка  $n$ . Расставьте элементы строк с четными номерами матрицы в порядке убывания.
39. Данна матрица  $A$  порядка  $n$ . Найдите два наибольших элемента матрицы с указанием номеров строк и столбцов, в которых они находятся.
40. Данна матрица  $A$  порядка  $n$ . Поменяйте местами строки: первую с последней, вторую с предпоследней и т.д.
41. Данна матрица  $A$  порядка  $n$ . Отсортируйте строки матрицы в порядке возрастания наибольших элементов в каждой строке.
42. Данна квадратная матрица  $A$  порядка  $n$ . Найдите седловую точку матрицы, т.е. элемент, который является наименьшим в своей строке и наибольшим в своем столбце.
43. Данна квадратная матрица  $A$  порядка  $n$ . Найдите суммы элементов тех строк матрицы, на главной диагонали которой стоят отрицательные элементы.
44. Данна матрица  $A$  порядка  $n$ . Расставьте элементы каждой строки в порядке возрастания.
45. Данна матрица  $A$  порядка  $n$ . Расставьте строки матрицы в порядке возрастания количества нулевых элементов.
46. Данна матрица  $A$  порядка  $n$ . Удалите строки, содержащие нулевые элементы.
47. Данна действительная квадратная матрица порядка  $2n$ . Получите новую матрицу (рис. 12.8).
48. Данна действительная квадратная матрица порядка  $2n$ . Получите новую матрицу (рис. 12.9).
49. Данна действительная квадратная матрица порядка  $2n$ . Получите новую матрицу (рис. 12.10).
50. Данна действительная квадратная матрица порядка  $2n$ . Получите новую матрицу (рис. 12.11).

1	2
3	4

3	4
2	1

1	4
2	3

4	3
1	2

1	2
4	3

Рис.  
12.7

Рис.  
12.8

Рис.  
12.9

Рис.  
12.10  
12.11

51. Даны две последовательности  $a_1, a_2, \dots, a_n$  и  $b_1, b_2, \dots, b_n$  целых чисел. Среди членов каждой последовательности нет повторяющихся чисел. Постройте пересечение последовательностей (т.е. получите в каком-нибудь порядке все числа, принадлежащие обеим последовательностям

одновременно).

52. Даны две последовательности  $a_1, a_2, \dots, a_n$  и  $b_1, b_2, \dots, b_n$  целых чисел.

Среди членов каждой последовательности нет повторяющихся чисел.

Постройте объединение данных последовательностей (равные члены разных последовательностей должны входить только один раз).

53. Постройте  $n$  строк треугольника Паскаля (рис. 12.12).

$$\begin{array}{ccccccc} & & & & 1 & & \\ & & & & 1 & 1 & \\ & & & & 1 & 2 & 1 \\ & & & & 1 & 3 & 3 & 1 \\ & & & & \dots & & \\ \end{array}$$

Рис. 12.12

54. Постройте матрицу порядка  $n$ , записав в нее числа от 1 до  $n^2$  согласно схеме (рис. 12.13).

$$\left( \begin{array}{ccccc} 1 & 2 & 3 & \dots & n \\ 2n & 2n-1 & 2n-2 & \dots & n+1 \\ 2n+1 & \dots & & & 3n \\ \dots & & & & \\ n^2-n+1 & \dots & & & n^2 \end{array} \right)$$

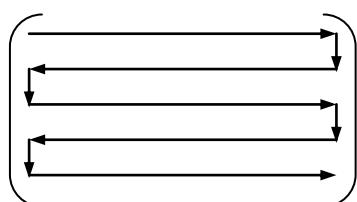


Рис. 12.13

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

Очень часто возникают задачи обработки массивов данных, размерность которых заранее неизвестна. В этом случае возможно использование одного из двух подходов:

- ✓ выделение памяти под статический массив, содержащий максимально возможное число элементов, однако в этом случае память расходуется не рационально;
- ✓ динамическое выделение памяти для хранение массива данных.

Для использования функций динамического выделения памяти необходимо описать указатель, представляющий собой начальный адрес хранения элементов массива.

`int *p; // указатель на тип int`

Начальный адрес статического массива определяется компилятором в момент его объявления и не может быть изменен.

Для динамического массива начальный адрес присваивается объявлениюному указателю на массив в процессе выполнения программы.

## Стандартные функции динамического выделения памяти

Функции динамического выделения памяти находят в оперативной памяти непрерывный участок требуемой длины и возвращают начальный адрес этого участка.

Функции динамического распределения памяти:

```
void* malloc(РазмерМассиваВБайтах);  
void* calloc(ЧислоЭлементов, РазмерЭлементаВБайтах);
```

Для использования функций динамического распределения памяти необходимо подключение библиотеки `<malloc.h>`:

```
#include <malloc.h>
```

Поскольку обе представленные функции в качестве возвращаемого значения имеют указатель на пустой тип `void`, требуется явное приведение типа возвращаемого значения.

Для определения размера массива в байтах, используемого в качестве аргумента функции `malloc()` требуется количество элементов умножить на размер одного элемента. Поскольку элементами массива могут быть как данные простых типов, так и составных типов (например, структуры), для точного определения размера элемента в общем случае рекомендуется использование функции

```
int sizeof(тип);
```

которая определяет количество байт, занимаемое элементом указанного типа.

Память, динамически выделенная с использованием функций `calloc()`, `malloc()`, может быть освобождена с использованием функции

```
free(указатель);
```

"Правилом хорошего тона" в программировании является освобождение динамически выделенной памяти в случае отсутствия ее дальнейшего использования. Однако если динамически выделенная память не освобождается явным образом, она будет освобождена по завершении выполнения программы.

## Динамическое выделение памяти для одномерных массивов

Форма обращения к элементам массива с помощью указателей имеет следующий вид:

```
int a[10], *p; // описываем статический массив и указатель  
int b;  
p = a; // присваиваем указателю начальный адрес массива  
... // ввод элементов массива  
b = *p; // b = a[0];  
b = *(p+i) // b = a[i];
```

**Примеры 12.1** Организация динамического одномерного массива и ввод его элементов.

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <malloc.h>
4 #include <stdlib.h>
5 int main()
6 {
7     int *a; // указатель на массив
8     int i, n;
9     system("chcp 1251");
10    system("cls");
11    printf("Введите размер массива: ");
12    scanf("%d", &n);
13    // Выделение памяти
14    a = (int*)malloc(n * sizeof(int));
15    // Ввод элементов массива
16    for (i = 0; i<n; i++)
17    {
18        printf("a[%d] = ", i);
19        scanf("%d", &a[i]);
20    }
21    // Вывод элементов массива
22    for (i = 0; i<n; i++)
23        printf("%d ", a[i]);
24    free(a);
25    getchar(); getchar();
26    return 0;
27 }

```

Результат выполнения программы:

```

C:\MyProgram\Debug\MyProgram.exe
Введите размер массива: 5
a[0] = 4
a[1] = 7
a[2] = 3
a[3] = 2
a[4] = 1
4 7 3 2 1

```

### Динамическое выделение памяти для двумерных массивов

Пусть требуется разместить в динамической памяти матрицу, содержащую **n** строк и **m** столбцов. Двумерная матрица будет располагаться в оперативной памяти в форме ленты, состоящей из элементов строк. При этом индекс любого элемента двумерной матрицы можно получить по формуле

**index = i\*m+j;**

где **i** - номер текущей строки; **j** - номер текущего столбца.

Рассмотрим матрицу 3x4

			j=2
0	1	2	3
i=1	4	5	6
	8	9	10
			11

Индекс выделенного элемента определится как

$$\text{index} = 1 * 4 + 2 = 6;$$

Объем памяти, требуемый для размещения двумерного массива, определится как

$$n \cdot m \cdot (\text{размер элемента})$$

Однако поскольку при таком объявлении компилятору явно не указывается количество элементов в строке и столбце двумерного массива, традиционное обращение к элементу путем указания индекса строки и индекса столбца является некорректным: **a[i][j]** – некорректно.

Правильное обращение к элементу с использованием указателя будет выглядеть как

$$\ast(p + i \ast m + j),$$

где

- ✓ **p** - указатель на массив,
- ✓ **m** - количество столбцов,
- ✓ **i** - индекс строки,
- ✓ **j** - индекс столбца.

**Пример 12.2** Ввод и вывод значений динамического двумерного массива

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <malloc.h>
4 #include <stdlib.h>
5 int main()
6 {
7     int *a; // указатель на массив
8     int i, j, n, m;
9     system("chcp 1251");
10    system("cls");
11    printf("Введите количество строк: ");
12    scanf("%d", &n);
13    printf("Введите количество столбцов: ");
14    scanf("%d", &m);
15    // Выделение памяти
16    a = (int*)malloc(n*m * sizeof(int));

```

```

17 // Ввод элементов массива
18 for (i = 0; i<n; i++) // цикл по строкам
19 {
20     for (j = 0; j<m; j++) // цикл по столбцам
21     {
22         printf("a[%d][%d] = ", i, j);
23         scanf("%d", (a + i*m + j));
24     }
25 }
26 // Вывод элементов массива
27 for (i = 0; i<n; i++) // цикл по строкам
28 {
29     for (j = 0; j<m; j++) // цикл по столбцам
30     {
31         printf("%5d ", *(a + i*m + j)); // 5 знакомест
32     }
33     printf("\n");
34 }
35 free(a);
36 getchar(); getchar();
37 return 0;
38 }

```

### Результат выполнения

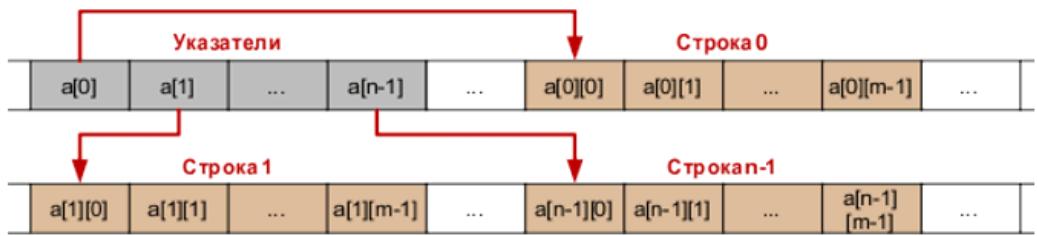
```

C:\MyProgram\Debug\MyProgram.exe
Введите количество строк: 3
Введите количество столбцов: 4
a[0][0] = 1
a[0][1] = 2
a[0][2] = 3
a[0][3] = 4
a[1][0] = 5
a[1][1] = 6
a[1][2] = 7
a[1][3] = 8
a[2][0] = 9
a[2][1] = 10
a[2][2] = 11
a[2][3] = 12
    1    2    3    4
    5    6    7    8
    9   10   11   12

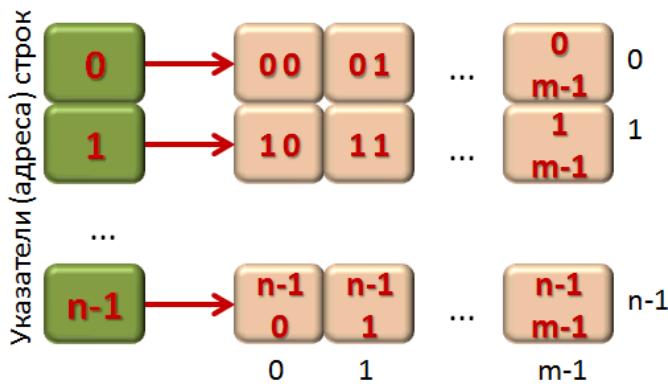
```

Возможен также другой способ динамического выделения памяти под двумерный массив - с использованием массива указателей. Для этого необходимо:

- ✓ выделить блок оперативной памяти под массив указателей;
- ✓ выделить блоки оперативной памяти под одномерные массивы, представляющие собой строки искомой матрицы;
- ✓ записать адреса строк в массив указателей.



Графически такой способ выделения памяти можно представить следующим образом.



При таком способе выделения памяти компилятору явно указано количество строк и количество столбцов в массиве.

**Пример 12.3** Выделение памяти под двумерный динамический массив, с помощью массива указателей.

```

1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <malloc.h>
4 #include <stdlib.h>
5 int main()
6 {
7     int **a; // указатель на указатель на строку элементов
8     int i, j, n, m;
9     system("chcp 1251");
10    system("cls");
11    printf("Введите количество строк: ");
12    scanf("%d", &n);
13    printf("Введите количество столбцов: ");
14    scanf("%d", &m);
15    // Выделение памяти под указатели на строки
16    a = (int**)malloc(n * sizeof(int*));

```

```
17 // Ввод элементов массива
18 for (i = 0; i<n; i++) // цикл по строкам
19 {
20     // Выделение памяти под хранение строк
21     a[i] = (int*)malloc(m * sizeof(int));
22     for (j = 0; j<m; j++) // цикл по столбцам
23     {
24         printf("a[%d][%d] = ", i, j);
25         scanf("%d", &a[i][j]);
26     }
27 }
28 // Вывод элементов массива
29 for (i = 0; i < n; i++) // цикл по строкам
30 {
31     for (j = 0; j < m; j++) // цикл по столбцам
32     {
33         printf("%5d ", a[i][j]); // 5 знакомест под элемент ма-
34     }
35     printf("\n");
36 }
37 // Очистка памяти
38 for (i = 0; i < n; i++) // цикл по строкам
39     free(a[i]); // освобождение памяти под строку
40 free(a);
41 getchar(); getchar();
42 return 0;
43 }
```

Результат выполнения программы аналогичен предыдущему случаю.

С помощью динамического выделения памяти под указатели строк можно размещать свободные массивы. **Свободным** называется двухмерный массив (матрица), размер строк которого может быть различным. Преимущество использования свободного массива заключается в том, что не требуется отводить память компьютера с запасом для размещения строки максимально возможной длины. Фактически свободный массив представляет собой одномерный массив указателей на одномерные массивы данных.

Для размещения в оперативной памяти матрицы со строками разной длины необходимо ввести дополнительный массив **m**, в котором будут храниться размеры строк.

## Пример 12.4 Создание свободного массива

```
1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <stdlib.h>
4 int main()
5 {
6     int **a;
7     int i, j, n, *m;
8     system("chcp 1251");
9     system("cls");
10    printf("Введите количество строк: ");
11    scanf("%d", &n);
12    a = (int**)malloc(n * sizeof(int*));
13    m = (int*)malloc(n * sizeof(int)); // массив кол-ва элем
14                                // Ввод элементов массива
15    for (i = 0; i<n; i++)
16    {
17        printf("Введите количество столбцов строки %d: ", i);
18        scanf("%d", &m[i]);
19        a[i] = (int*)malloc(m[i] * sizeof(int));
20        for (j = 0; j<m[i]; j++) {
21            printf("a[%d][%d]= ", i, j);
22            scanf("%d", &a[i][j]);
23        }
24    }
25    // Вывод элементов массива
26    for (i = 0; i<n; i++)
27    {
28        for (j = 0; j<m[i]; j++)
29        {
30            printf("%3d ", a[i][j]);
31        }
32        printf("\n");
33    }
34    // Освобождение памяти
35    for (i = 0; i < n; i++)
36    {
37        free(a[i]);
38    }
39    free(a);
40    free(m);
41    getchar(); getchar();
42    return 0;
43 }
```

Результат выполнения

```
C:\MyProgram\Debug\MyProgram.exe
Введите количество строк: 3
Введите количество столбцов строки 0: 10
a[0][0]= 1
a[0][1]= 2
a[0][2]= 3
a[0][3]= 4
a[0][4]= 5
a[0][5]= 6
a[0][6]= 7
a[0][7]= 8
a[0][8]= 9
a[0][9]= 10
Введите количество столбцов строки 1: 2
a[1][0]= 11
a[1][1]= 12
Введите количество столбцов строки 2: 5
a[2][0]= 13
a[2][1]= 14
a[2][2]= 15
a[2][3]= 16
a[2][4]= 17
 1  2  3  4  5  6  7  8  9  10
 11 12
 13 14 15 16 17
```

### Перераспределение памяти

Если размер выделяемой памяти нельзя задать заранее, например при вводе последовательности значений до определенной команды, то для увеличения размера массива при вводе следующего значения необходимо выполнить следующие действия:

- ✓ Выделить блок памяти размерности **n+1** (на 1 больше текущего размера массива)
- ✓ Скопировать все значения, хранящиеся в массиве во вновь выделенную область памяти
- ✓ Освободить память, выделенную ранее для хранения массива
- ✓ Переместить указатель начала массива на начало вновь выделенной области памяти
- ✓ Дополнить массив последним введенным значением

Все перечисленные выше действия (кроме последнего) выполняет функция

```
void* realloc (void* ptr, size_t size);
```

- ✓ **ptr** - указатель на блок ранее выделенной памяти функциями **malloc()**, **calloc()** или **realloc()** для перемещения в новое место. Если этот параметр равен **NULL**, то выделяется новый блок, и функция возвращает на него указатель.
- ✓ **size** - новый размер, в байтах, выделяемого блока памяти. Если **size = 0**, ранее выделенная память освобождается и функция возвращает нулевой указатель, **ptr** устанавливается в **NULL**.
- ✓ Размер блока памяти, на который ссылается параметр **ptr** изменяется на **size** байтов. Блок памяти может уменьшаться или увеличиваться в размере. Содержимое блока

памяти сохраняется даже если новый блок имеет меньший размер, чем старый. Но отбрасываются те данные, которые выходят за рамки нового блока. Если новый блок памяти больше старого, то содержимое вновь выделенной памяти будет неопределенным.

**Пример 12.5** Выделить память для ввода массива целых чисел. После ввода каждого значения задавать вопрос о вводе следующего значения.

```
1 #define _CRT_SECURE_NO_WARNINGS
2 #include <stdio.h>
3 #include <malloc.h>
4 int main()
5 {
6     int *a = NULL, i = 0, elem;
7     char c;
8     do {
9         printf("a[%d]= ", i);
10        scanf("%d", &elem);
11        a = (int*)realloc(a, (i + 1) * sizeof(int));
12        a[i] = elem;
13        i++;
14        getchar();
15        printf("Next (y/n)? ");
16        c = getchar();
17    } while (c == 'y');
18    for (int j = 0; j < i; j++)
19        printf("%d ", a[j]);
20    if (i>2) i -= 2;
21    printf("\n");
22    a = (int*)realloc(a, i * sizeof(int)); // уменьшение размера массива на 2
23    for (int j = 0; j < i; j++)
24        printf("%d ", a[j]);
25    getchar(); getchar();
26    return 0;
27 }
```

Результат выполнения

```
a[0]= 1
Next (y/n)? y
a[1]= 2
Next (y/n)? y
a[2]= 3
Next (y/n)? y
a[3]= 4
Next (y/n)? y
a[4]= 5
Next (y/n)? y
a[5]= 6
Next (y/n)? y
a[6]= 7
Next (y/n)? y
a[7]= 8
Next (y/n)? y
a[8]= 9
Next (y/n)? y
a[9]= 10
Next (y/n)? n
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8
```

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## 6. Форма отчета о работе

Лабораторная работа № \_\_\_\_\_

Номер учебной группы \_\_\_\_\_

Фамилия, инициалы учащегося \_\_\_\_\_

Дата выполнения работы \_\_\_\_\_

Тема работы: \_\_\_\_\_

Цель работы: \_\_\_\_\_

Оснащение работы: \_\_\_\_\_

Результат выполнения работы: \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Назначение динамической памяти. В чем состоит отличие динамической памяти от статической?
2. Дайте определение понятию «указатель». Формат описания.
3. Как выполняется обращение к выделенной динамической памяти?
4. Где размещается динамическая область? Как устанавливается размер «кучи»? Единица измерения размера «кучи».
5. Как создать динамический массив?
6. Как передать динамический массив в функцию?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Бондарев В.М. Основы программирования/ В.И. Рублинецкий, Е.Г. Качко. – Харьков: Фолио; Ростов н/Д: Феникс, 1997. –368с.
3. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
4. Гриффитс Д. Изучаем программирование на С / Д. Гриффитс – М.: Эксмо, Айдиономикс, 2013. – 625 с.
5. Касьянов В.Н. Сборник заданий по практикуму на ЭВМ / В.К. Сабельфельд– М.:Наука, 1986. –272 с.
6. Клеменс Б. Язык С в XXI веке. – М.: ДМК Пресс, 2015. – 376 с.
7. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
8. Майк М.К. С программирование для начинающих – М.: Эксмо, 2016. – 192 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание программ обработки односвязного и  
двусвязного списков»

Минск  
2017

# **Лабораторная работа № 13**

**Тема работы:** «Разработка, отладка и испытание программ обработки односвязного и двусвязного списков»

## **1. Цель работы**

Отработать навык создания обработки информации на базе односвязного и двусвязного списков.

## **2. Задание**

В задании 1 приведены задачи по теме «Односвязный список», в задании 2 приведены задачи по теме «Двусвязный список». Оба задания обязательны для выполнения. Вариант соответствует вашему номеру по списку.

### **Задание 1: «Односвязный список»**

1. Счет в банке представляет собой структуру с полями: номер счета, код счета, фамилия владельца, сумма на счете, дата открытия счета, годовой процент начисления. Реализовать поиск и сортировку по номеру счета, дате его открытия и фамилии владельца.
2. Запись о товаре на складе представляет собой структуру с полями: номер склада, код товара, наименование товара, дата поступления на склад, срок хранения в днях, количество единиц товара, цена за единицу товара. Поиск и сортировка - по номеру склада, наименованию товара. Вывести список просроченных товаров (поиск всех товаров, у которых на текущую дату истек срок хранения).
3. Запись о преподаваемой дисциплине представляется структурой: код дисциплины в учебном плане, наименование дисциплины, фамилия преподавателя, код группы, количество студентов в группе, количество часов лекций, количество часов практических занятий, вид итогового контроля (зачет или экзамен), дата начала занятий. Поиск и сортировка - по фамилии преподавателя, количеству часов, дате начала занятий.
4. Информационная запись о книге, выданной на руки абоненту, представляет собой структуру следующего вида: номер читательского билета, фамилия абонента, дата выдачи, количество дней, автор, название, год издания, цена. Поиск и сортировка - по номеру читательского билета, автору книги. Вывести список всех просроченных книг (поиск всех книг, которые на текущую дату должны быть сданы).
5. Информационная запись о файле содержит следующие поля: каталог, имя файла, расширение, дата и время создания, атрибуты «только для чтения», «скрытый», «системный», количество выделенных секторов (размер сектора принять равным 512 байтам). Поиск и сортировка - по каталогу, дате создания файла. Выяснить, поместится ли файл на носитель с некоторым количеством секторов.
6. Разовый платеж за телефонный разговор является структурой с полями: фамилия плательщика, номер телефона, дата разговора, тариф за минуту

разговора, время начала разговора, время окончания разговора. Поиск и сортировка - по фамилии владельца, дате разговора. Найти все разговоры со временем разговора больше заданного.

7. Модель компьютера характеризуется кодом и маркой компьютера, типом процессора (может содержать цифры и буквы), частотой работы процессора, объемом оперативной памяти, объемом жесткого диска, датой выпуска на рынок, стоимостью компьютера в рублях и количеством экземпляров, имеющихся в наличии. Поиск и сортировка - по типу процессора, объему ОЗУ, дате выпуска компьютера на рынок.
8. Список абонентов сети кабельного телевидения состоит из элементов следующей структуры: фамилия, район, адрес, телефон, номер договора, дата заключения договора, оплата установки, дата последнего платежа. Поиск и сортировка - по району, номеру договора, дате последнего платежа.
9. Сотрудник представлен структурой Person с полями: табельный номер, номер отдела, фамилия, оклад, дата поступления на работу, процент надбавки, процент налоговых сборов, количество отработанных дней в месяце, количество рабочих дней в месяце, начислено, удержано. Поиск и сортировка - по номеру отдела, дате поступления на работу, фамилии.
10. Запись о багаже пассажира авиарейса содержит следующие поля: номер рейса, дата и время вылета, пункт назначения, фамилия пассажира, количество мест багажа, суммарный вес багажа. Поиск и сортировка - по дате вылета, пункту назначения. Найти всех пассажиров, у которых масса багажа выше максимально допустимой.
11. Учетная запись посещения спорткомплекса имеет структуру: фамилия клиента, код и вид спортивного занятия, фамилия тренера, дата и время начала тренировки, количество минут, тариф. Поиск и сортировка - по фамилии клиента, дате начала и количеству минут тренировки (больше или меньше введенного).
12. Одна запись о медикаменте содержит следующие поля: номер аптеки, название лекарства, количество упаковок, имеющиеся в наличии в данной аптеке, стоимость одной упаковки, дата поступления в аптеку, срок хранения (в днях). Поиск и сортировка - по номеру аптеки, наименованию препарата, дате поступления.
13. Одна запись журнала учета содержит поля: код игрушки, название игрушки, тип игрушки, возрастные ограничения (например, от 6 лет), цена за единицу, количество в наличии, дата поступления в магазин, поставщик. Поиск и сортировка - по дате поступления, поставщику, возрастным ограничениям.
14. Один элемент (автомобиль) представляет собой структуру с полями: фамилия владельца, марка автомобиля, требуемая марка бензина, мощность двигателя, объем бака, остаток бензина, объем масла, дата техосмотра. Даны фиксированная цена литра бензина и заливки масла.

Поиск и сортировка - по марке автомобиля, мощности двигателя, дате техосмотра.

- 15.Запись в журнале зимней экзаменационной сессии представляет собой структуру с полями: курс, код группы, фамилия студента, дата поступления, номер зачетной книжки, дисциплина, оценка за экзамен по дисциплине. Поиск и сортировка - по номеру курса, номеру зачетной книжки, дате поступления.
- 16.Структура одной записи оплаты за коммунальные услуги содержит поля: номер дома, номер квартиры, фамилия владельца, вид платежа (квартплата, газ, вода, электричество), дата платежа, сумма платежа, процент пени, на сколько дней просрочен , платеж. Поиск и сортировка - по номеру дома, виду платежа, дате платежа.
- 17.Одна запись счета за ремонтные работы содержит поля: название фирмы, вид работ, единица измерения, стоимость единицы выполненных работ, дата исполнения, количество выполненных работ. Поиск и сортировка - по названию фирмы, виду работ, дате исполнения.
- 18.Одна учетная запись журнала стоянки автомобилей имеет структуру: номер автомобиля, фамилия владельца, дата и время начала, дата и время окончания, тариф за час. Поиск и сортировка - по номеру автомобиля, дате начала стоянки, фамилии владельца.
- 19.Структура записи о сельскохозяйственном продукте со держит поля: наименование района (где выращивают), наименование продукта, площадь (га), урожайность (кг/га), цена за 1 кг, потери при транспортировке (%), стоимость продукта, предполагаемая дата сбора. Поиск и сортировка - по наименованию района, урожайности, предполагаемой дате сбора.
- 20.В туристической фирме учетная запись о проданном туре содержит следующие поля: наименование тура, фамилия клиента, цена одного дня (в рублях), дата заезда, количество дней, стоимость проезда, курс валюты, количество валюты, стоимость проезда. Поиск и сортировка - по наименованию тура, стоимости проезда, дате заезда.
- 21.Сотовый телефон характеризуется названием производителя, номером модели (может содержать цифры и буквы), временем работы аккумулятора, наличием и максимальной емкостью карты памяти, датой выпуска на рынок, стоимостью в рублях и количеством экземпляров, имеющихся в наличии. Поиск и сортировка - по номеру модели, объему памяти на карте, дате выхода на рынок.
- 22.Одна запись о предмете мебели содержит следующие поля: артикул (может содержать цифры и буквы), наименование, цвет, стоимость, дата изготовления, количество имеющихся в наличии экземпляров. Поиск и сортировка - по артикулу, количеству экземпляров, дате изготовления.
- 23.Список абонентов сети кабельного телевидения состоит из элементов следующей структуры: фамилия, район, адрес, телефон, номер договора, дата заключения договора, оплата установки, дата последнего платежа.

Поиск и сортировка - по району, номеру договора, дате последнего платежа.

24. Сотрудник представлен структурой Person с полями: табельный номер, номер отдела, фамилия, оклад, дата поступления на работу, процент надбавки, процент налоговых сборов, количество отработанных дней в месяце, количество рабочих дней в месяце, начислено, удержано. Поиск и сортировка - по номеру отдела, дате поступления на работу, фамилии.
25. Запись о багаже пассажира авиарейса содержит следующие поля: номер рейса, дата и время вылета, пункт назначения, фамилия пассажира, количество мест багажа, суммарный вес багажа. Поиск и сортировка - по дате вылета, пункту назначения. Найти всех пассажиров, у которых масса багажа выше максимально допустимой.
26. Учетная запись посещения спорткомплекса имеет структуру: фамилия клиента, код и вид спортивного занятия, фамилия тренера, дата и время начала тренировки, количество минут, та риф. Поиск и сортировка - по фамилии клиента, дате начала и количеству минут тренировки (больше или меньше введенного).
27. Одна запись о медикаменте содержит следующие поля: номер аптеки, название лекарства, количество упаковок, имеющиеся в наличии в данной аптеке, стоимость одной упаковки, дата поступления в аптеку, срок хранения (в днях). Поиск и сортировка - по номеру аптеки, наименованию препарата, дате поступления.
28. Одна запись журнала учета содержит поля: код игрушки, название игрушки, тип игрушки, возрастные ограничения (например, от 6 лет), цена за единицу, количество в наличии, дата поступления в магазин, поставщик. Поиск и сортировка - по дате поступления, поставщику, возрастным ограничениям.
29. Один элемент (автомобиль) представляет собой структуру с полями: фамилия владельца, марка автомобиля, требуемая марка бензина, мощность двигателя, объем бака, остаток бензина, объем масла, дата техосмотра. Даны фиксированная цена литра бензина и заливки масла. Поиск и сортировка - по марке автомобиля, мощности двигателя, дате техосмотра.
30. Запись в журнале зимней экзаменационной сессии представляет собой структуру с полями: курс, код группы, фамилия студента, дата поступления, номер зачетной книжки, дисциплина, оценка за экзамен по дисциплине. Поиск и сортировка - по номеру курса, номеру зачетной книжки, дате поступления.

### **Задание 2: «Двусвязный список»**

1. В созданном списке вычислить среднее арифметическое и заменить им первый элемент.

2. В созданном списке вычислить среднее арифметическое и заменить им все четные значения элементов.
3. В созданном списке определить количество и удалить все элементы, находящиеся между минимальным и максимальным элементами.
4. В созданном списке определить количество элементов, имеющих значения, меньше среднего значения от всех элементов, и удалить эти элементы.
5. В созданном списке определить максимальное значение и удалить его.
6. В созданном списке поменять местами крайние элементы.
7. В созданном списке поменять местами элементы, содержащие максимальное и минимальное значения.
8. Из созданного списка удалить каждый второй элемент.
9. Из созданного списка удалить каждый нечетный элемент.
10. Из созданного списка удалить элементы, заканчивающиеся на цифру 5.
11. Магазин. Список с товарами (название, цена). Список покупателей (номер, название товара, который хочет купить, количество товара). Просмотреть список покупателей, подсчитать количество денег, полученных за обслуживание всех покупателей.
12. Очередь в магазине. Узел очереди: номер, индикатор — является ли человек в очереди ветераном. Сначала извлечь из очереди всех ветеранов, затем всех остальных в порядке очереди.
13. Очередь с ветеранами (см. условие задачи 7). Переставить ветеранов в начало очереди. Распечатать очередь.
14. Перенести из созданного списка в новый список все элементы, находящиеся между вершиной и максимальным элементом.
15. Перенести из созданного списка в новый список все элементы, находящиеся между вершиной и элементом с минимальным значением.
16. Разделить созданный список на два: в первом — положительные числа, во втором — отрицательные.
17. Созданный список разделить на два: в первый поместить четные, а во второй — нечетные числа.
18. Список (односвязный). Узел — информация об ученике (фамилия, имя, список отметок по нескольким предметам). Построить отдельный список из двоичников (упорядоченный по алфавиту). Распечатать полученный список.
19. Список планет солнечной системы. Узел — название, удаление от Солнца, масса планеты. Упорядочить планеты в списке по возрастанию масс.
20. Список с товарами (название товара, цена, количество товара в магазине). Список с покупателями. Элементы списка покупателей удаляются после обслуживания. Вывести суммарную стоимость проданных товаров, а также список оставшихся товаров.

21. Список. Узел — запись о книге в библиотеке: автор, название, имеется ли в наличии (если нет, то фамилия читателя, которому выдана). Создать отдельный список для книг свободных. Распечатать новый список.
22. Список. Узел — информация о летчиках: имя, год рождения, звание, число полетов, число сбитых самолетов, отметка о жизни (сбит или нет). Удалить из списка летчиков, погибших в бою. Распечатать оставшийся список.
23. Список. Узел — информация о пациенте в больнице. Пациентов с нормальной температурой и давлением перенести в новый список, удаляя из старого списка, распечатать новый список.
24. Список. Узел — информация о пациенте в больнице: имя, возраст, пол, диагноз. Сделать два списка: мужчины и женщины (упорядочить по возрасту). Распечатать их.
25. Список. Узел — информация о студенте: имя, дата рождения, отметки по алгебре, музыке, информатике. Создать 4 списка: в соответствии с отметками по информатике (2, 3, 4, 5).
26. Список. Узел — информация о товаре: название, цена, дата поступления, срок хранения. Вводится сегодняшняя дата. Удалить товары, срок хранения которых истек.
27. Список. Узел — информация об ученике. Получить два списка: успевающие и неуспевающие ученики. Распечатать оба списка.
28. Список. Узел списка — информация о студенте: фамилия, год рождения, год поступления, оценки по предметам. Студентов, поступивших в нечетном году, занести в другой список (с удалением из первого). Распечатать оба списка.
29. Список. Узел списка — целое число. Упорядочить элементы списка по возрастанию.
30. Удалить из созданного списка отрицательные элементы.
31. Удалить из созданного списка элементы с четными числами.

### **3. Оснащение работы**

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### **4. Основные теоретические сведения**

Часто в серьезных программах надо использовать данные, размер и структура которых должны меняться в процессе работы. Динамические массивы здесь не выручают, поскольку заранее нельзя сказать, сколько памяти надо выделить — это выясняется только в процессе работы. Например, надо проанализировать текст и определить, какие слова и в каком количестве в нем встречаются, причем эти слова нужно расставить по алфавиту.

В таких случаях применяют данные особой структуры, которые представляют собой отдельные элементы, связанные с помощью **ссылок**. Каждый элемент (узел) состоит из двух областей памяти: **поля данных** и **ссылок**. Ссылки – это адреса других узлов этого же типа, с которыми данный элемент логически связан. В языке Си для организации ссылок используются переменные-указатели. При добавлении нового узла в такую структуру выделяется новый блок памяти и (с помощью ссылок) устанавливаются связи этого элемента с уже существующими. Для обозначения конечного элемента в цепи используются **нулевые ссылки (NULL)**.



### Линейный список

В простейшем случае каждый узел содержит всего одну ссылку. Для определенности будем считать, что решается задача частотного анализа текста – определения всех слов, встречающихся в тексте и их количества. В этом случае область данных элемента включает строку (длиной не более 40 символов) и целое число.



Каждый элемент содержит также ссылку на **следующий** за ним элемент. У последнего в списке элемента поле ссылки содержит **NULL**. Чтобы не потерять список, мы должны где-то (в переменной) хранить адрес его первого узла – он называется «головой» списка. В программе надо объявить два новых типа данных – узел списка **Node** и указатель на него **PNode**. Узел представляет собой структуру, которая содержит три поля - строку, целое число и указатель на такой же узел. Правилами языка Си допускается объявление

```

struct Node {
    char word[40]; // область данных
    int count;
    Node *next; // ссылка на следующий узел
};

typedef Node *PNode; // тип данных: указатель на узел
  
```

В дальнейшем мы будем считать, что указатель **Head** указывает на начало списка, то есть, объявлен в виде

```
PNode Head = NULL;
```

Первая буква «Р» в названии типа **PNode** происходит от слова *pointer* – указатель (англ.) В начале работы в списке нет ни одного элемента, поэтому в указатель **Head** записывается нулевой адрес **NULL**.

### Создание элемента списка

Для того, чтобы добавить узел к списку, необходимо создать его, то есть выделить память под узел и запомнить адрес выделенного блока. Будем считать, что надо добавить к списку узел, соответствующий новому слову, которое записано в переменной **NewWord**. Составим функцию, которая создает

новый узел в памяти и возвращает его адрес. Обратите внимание, что при записи данных в узел используется обращение к полям структуры через указатель.

```
PNode CreateNode ( char NewWord[] )  
{  
    PNode NewNode = new Node; // указатель на новый узел  
    strcpy(NewNode->word, NewWord); // записать слово  
    NewNode->count = 1; // счетчик слов = 1  
    NewNode->next = NULL; // следующего узла нет  
    return NewNode; // результат функции - адрес узла  
}
```

После этого узел надо добавить к списку (в начало, в конец или в середину).

### Добавление узла

#### Добавление узла в начало списка

При добавлении нового узла **NewNode** в начало списка надо 1) установить ссылку узла **NewNode** на голову существующего списка и 2) установить голову списка на новый узел.



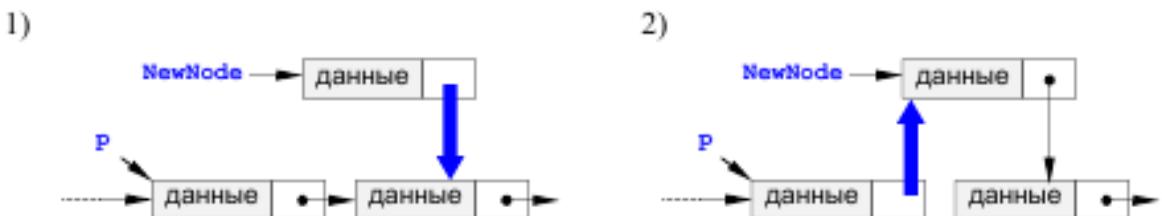
По такой схеме работает процедура **AddFirst**. Предполагается, что адрес начала списка хранится в **Head**. Важно, что здесь и далее адрес начала списка передается *по ссылке*, так как при добавлении нового узла он изменяется внутри процедуры.

```
void AddFirst (PNode &Head, PNode NewNode)  
{  
    NewNode->next = Head;  
    Head = NewNode;  
}
```

#### Добавление узла после заданного

Дан адрес **NewNode** нового узла и адрес **p** одного из существующих узлов в списке. Требуется вставить в список новый узел после узла с адресом **p**. Эта операция выполняется в два этапа:

- 1) установить ссылку нового узла на узел, следующий за данным;
- 2) установить ссылку данного узла **p** на **NewNode**.



Последовательность операций менять нельзя, потому что если сначала поменять ссылку у узла **p**, будет потерян адрес следующего узла.

```
void AddAfter (PNode p, PNode NewNode)
{
    NewNode->next = p->next;
    p->next = NewNode;
}
```

### Добавление узла перед заданным

Эта схема добавления самая сложная. Проблема заключается в том, что в простейшем линейном списке (он называется *односвязным*, потому что связи направлены только в одну сторону) для того, чтобы получить адрес предыдущего узла, нужно пройти весь список сначала. Задача сводится либо к вставке узла в начало списка (если заданный узел – первый), либо к вставке после заданного узла.

```
void AddBefore(PNode &Head, PNode p, PNode NewNode)
{
    PNode q = Head;

    if (Head == p) {
        AddFirst(Head, NewNode); // вставка перед первым узлом
        return;
    }

    while (q && q->next!=p) // ищем узел, за которым следует p
        q = q->next;
    if ( q )                // если нашли такой узел,
        AddAfter(q, NewNode); //     добавить новый после него
}
```

Такая процедура обеспечивает «защиту от дурака»: если задан узел, не присутствующий в списке, то в конце цикла указатель **q** равен **NULL** и ничего не происходит.

Существует еще один интересный прием: если надо вставить новый узел **NewNode** **до** заданного узла **p**, вставляют узел **после** этого узла, а потом выполняется обмен данными между узлами **NewNode** и **p**. Таким образом, по адресу **p** в самом деле будет расположен узел с новыми данными, а по адресу **NewNode** – с теми данными, которые были в узле **p**, то есть мы решили задачу. Этот прием не сработает, если адрес нового узла **NewNode** запоминается где-то в программе и потом используется, поскольку по этому адресу будут находиться другие данные.

### Добавление узла в конец списка

Для решения задачи надо сначала найти последний узел, у которого ссылка равна **NULL**, а затем воспользоваться процедурой вставки после заданного узла. Отдельно надо обработать случай, когда список пуст.

```

void AddLast(PNode &Head, PNode NewNode)
{
    PNode q = Head;
    if (Head == NULL) {           // если список пуст,
        AddFirst(Head, NewNode); // вставляем первый элемент
        return;
    }
    while (q->next) q = q->next; // ищем последний элемент
    AddAfter(q, NewNode);
}

```

### Проход по списку

Для того, чтобы пройти весь список и сделать что-либо с каждым его элементом, надо начать с головы и, используя указатель **next**, продвигаться к следующему узлу.

```

PNode p = Head;           // начали с головы списка
while ( p != NULL ) {    // пока не дошли до конца
    // делаем что-нибудь с узлом p
    p = p->next;         // переходим к следующему узлу
}

```

### Поиск узла в списке

Часто требуется найти в списке нужный элемент (его адрес или данные). Надо учесть, что требуемого элемента может и не быть, тогда просмотр заканчивается при достижении конца списка. Такой подход приводит к следующему алгоритму:

- ✓ начать с головы списка;
- ✓ пока текущий элемент существует (указатель – не **NULL**), проверить нужное условие и перейти к следующему элементу;
- ✓ закончить, когда найден требуемый элемент или все элементы списка просмотрены.

Например, следующая функция ищет в списке элемент, соответствующий заданному слову (для которого поле **word** совпадает с заданной строкой **NewWord**), и возвращает его адрес или **NULL**, если такого узла нет.

```

PNode Find (PNode Head, char NewWord[])
{
    PNode q = Head;
    while (q && strcmp(q->word, NewWord))
        q = q->next;
    return q;
}

```

Вернемся к задаче построения алфавитно-частотного словаря. Для того, чтобы добавить новое слово в нужное место (в алфавитном порядке), требуется найти адрес узла, *перед* которым надо вставить новое слово. Это будет первый от начала списка узел, для которого «его» слово окажется «больше», чем новое слово. Поэтому достаточно просто изменить условие в цикле **while** в функции **Find**., учитывая, что функция **strcmp** возвращает «разность» первого и второго слова.

```

PNode FindPlace (PNode Head, char NewWord[])
{
    PNode q = Head;
    while (q && (strcmp(q->word, NewWord) > 0))
        q = q->next;
    return q;
}

```

### Удаление узла

Эта процедура также связана с поиском заданного узла по всему списку, так как нам надо поменять ссылку у предыдущего узла, а перейти к нему непосредственно невозможно. Если мы нашли узел, за которым идет удаляемый узел, надо просто переставить ссылку.



Отдельно обрабатывается случай, когда удаляется первый элемент списка. При удалении узла освобождается память, которую он занимал.

Отдельно рассматриваем случай, когда удаляется первый элемент списка. В этом случае адрес удаляемого узла совпадает с адресом головы списка **Head** и надо просто записать в **Head** адрес следующего элемента.

```

void DeleteNode(PNode &Head, PNode OldNode)
{
    PNode q = Head;
    if (Head == OldNode)
        Head = OldNode->next; // удаляем первый элемент
    else {
        while (q && q->next != OldNode) // ищем элемент
            q = q->next;
        if (q == NULL) return; // если не нашли, выход
        q->next = OldNode->next;
    }
    delete OldNode;           // освобождаем память
}

```

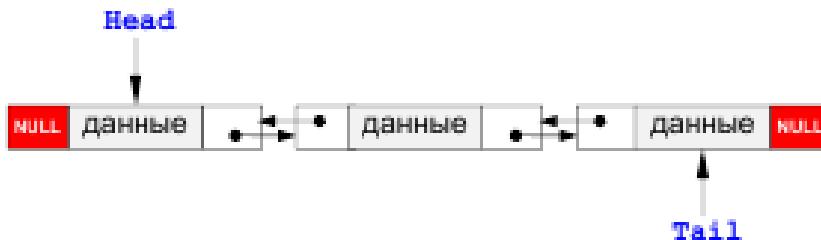
### Барьеры

Вы заметили, что для рассмотренного варианта списка требуется отдельно обрабатывать граничные случаи: добавление в начало, добавление в конец, удаление одного из крайних элементов. Можно значительно упростить приведенные выше процедуры, если установить два барьера – фиктивные первый и последний элементы. Таким образом, в списке всегда есть хотя бы два элемента-барьера, а все рабочие узлы находятся между ними.

### Двусвязный список

Многие проблемы при работе с односвязным списком вызваны тем, что в них невозможно перейти к предыдущему элементу. Возникает естественная

идея – хранить в памяти ссылку не только на следующий, но и на предыдущий элемент списка. Для доступа к списку используется не одна переменная-указатель, а две – ссылка на «голову» списка (**Head**) и на «хвост» - последний элемент (**Tail**).



Каждый узел содержит ( кроме полезных данных) также ссылку на **следующий** за ним узел (поле **next**) и предыдущий (поле **prev**). Поле **next** у последнего элемента и поле **prev** у первого содержат **NULL**. Узел объявляется так:

```
struct Node {
    char word[40];      // область данных
    int count;
    Node *next, *prev; // ссылки на соседние узлы
};

typedef Node *PNode; // тип данных «указатель на узел»
```

В дальнейшем мы будем считать, что указатель **Head** указывает на начало списка, а указатель **Tail** – на конец списка:

```
PNode Head = NULL, Tail = NULL;
```

Для пустого списка оба указателя равны **NULL**.

### Операции с двусвязным списком

#### Добавление узла в начало списка

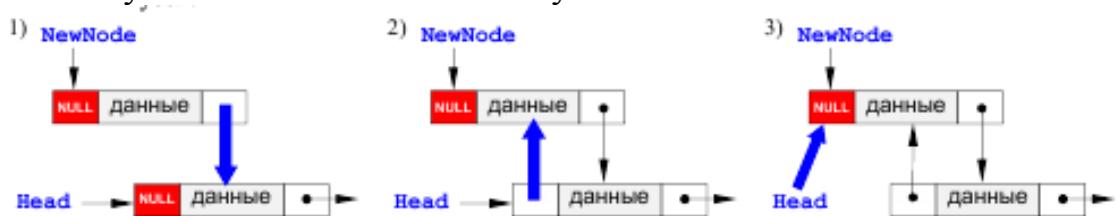
При добавлении нового узла **NewNode** в начало списка надо

1) установить ссылку **next** узла **NewNode** на голову существующего списка и его ссылку **prev** в **NULL**;

2) установить ссылку **prev** бывшего первого узла (если он существовал) на **NewNode**;

3) установить голову списка на новый узел;

4) если в списке не было ни одного элемента, хвост списка также устанавливается на новый узел.



По такой схеме работает следующая процедура:

```

void AddFirst(PNode &Head, PNode &Tail, PNode NewNode)
{
    NewNode->next = Head;
    NewNode->prev = NULL;
    if ( Head ) Head->prev = NewNode;
    Head = NewNode;
    if ( ! Tail ) Tail = Head; // этот элемент - первый
}

```

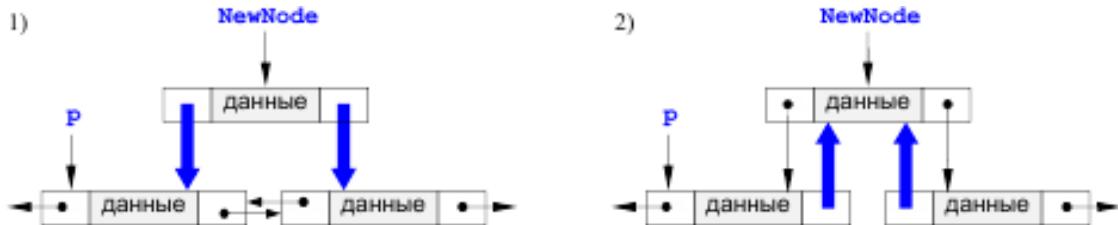
### Добавление узла в конец списка

Благодаря симметрии добавление нового узла **NewNode** в конец списка проходит совершенно аналогично, в процедуре надо везде заменить **Head** на **Tail** и наоборот, а также поменять **prev** и **next**.

### Добавление узла после заданного

Дан адрес **NewNode** нового узла и адрес **p** одного из существующих узлов в списке. Требуется вставить в список новый узел после **p**. Если узел **p** является последним, то операция сводится к добавлению в конец списка (см. выше). Если узел **p** – не последний, то операция вставки выполняется в два этапа:

- 1) установить ссылки нового узла на следующий за данным (**next**) и предшествующий ему (**prev**);
- 2) установить ссылки соседних узлов так, чтобы включить **NewNode** в список.



Такой метод реализует приведенная ниже процедура (она учитывает также возможность вставки элемента в конец списка, именно для этого в параметрах передаются ссылки на голову и хвост списка):

```

void AddAfter (PNode &Head, PNode &Tail,
               PNode p, PNode NewNode)
{
    if ( ! p->next )
        AddLast (Head, Tail, NewNode); // вставка в конец списка
    else {
        NewNode->next = p->next; // меняем ссылки нового узла
        NewNode->prev = p;
        p->next->prev = NewNode; // меняем ссылки соседних узлов
        p->next = NewNode;
    }
}

```

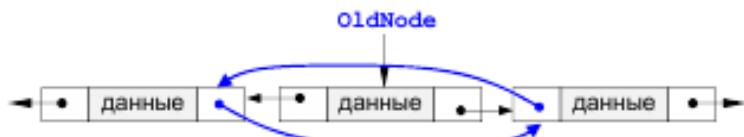
Добавление узла перед заданным выполняется аналогично.

### Поиск узла в списке

Проход по двусвязному списку может выполняться в двух направлениях – от головы к хвосту (как для односвязного) или от хвоста к голове.

### Удаление узла

Эта процедура также требует ссылки на голову и хвост списка, поскольку они могут изменяться при удалении крайнего элемента списка. На первом этапе устанавливаются ссылки соседних узлов (если они есть) так, как если бы удаляемого узла не было бы. Затем узел удаляется и память, которую он занимает, освобождается. Эти этапы показаны на рисунке внизу. Отдельно проверяется, не является ли удаляемый узел первым или последним узлом списка.



```
void Delete(PNode &Head, PNode &Tail, PNode OldNode)
{
    if (Head == OldNode) {
        Head = OldNode->next;      // удаляем первый элемент
        if (Head)
            Head->prev = NULL;
        else Tail = NULL;          // удалили единственный элемент
    }
    else {
        OldNode->prev->next = OldNode->next;
        if (OldNode->next)
            OldNode->next->prev = OldNode->prev;
        else Tail = NULL;          // удалили последний элемент
    }
    delete OldNode;
}
```

### Циклические списки

Иногда список (односвязный или двусвязный) замыкают в кольцо, то есть указатель **next** последнего элемента указывает на первый элемент, и (для двусвязных списков) указатель **prev** первого элемента указывает на последний. В таких списках понятие «хвоста» списка не имеет смысла, для работы с ним надо использовать указатель на «голову», причем «головой» можно считать любой элемент.

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_  
\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Что такое ссылка?
2. Линейный (односвязный) список, что это?
3. Какие операции можно выполнять с односвязным списком?
4. Двусвязный список, что это?
5. Какие операции можно выполнять с двусвязным списком?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
3. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
4. Окулов С.М. Программирование в алгоритмах – М.: БИНОМ. Лаборатория знаний, 2006. - 383 с.
5. Прата С. Язык программирования С. - СПб.: Вильямс, 2013. – 960 с.
6. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400
7. Хьюз Дж. Структурный подход к программированию / Мичтом Дж. – М.: Мир, 1980. –278 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание программ обработки стеков и очередей»

Минск  
2017

## **Лабораторная работа № 14**

**Тема работы:** «Разработка, отладка и испытание программ обработки стеков и очередей»

### **1. Цель работы**

Закрепить навык работы с стоком о очередью.

### **2. Задание**

В задании 1 приведены задачи по теме «Стек», в задании 2 приведены задачи по теме «Очередь». Оба задания обязательны для выполнения. Вариант соответствует вашему номеру по списку.

#### **Задание 1: «Стек»**

1. Заполнить стек **N** случайными числами из интервала [-10;20].  
Просмотреть содержимое стека. Найти сумму положительных чисел, хранящихся в стеке.
2. Сформировать стек из **N** чисел. Извлечь элементы из стека, увеличить каждое из них на 1 и снова поместить в стек в том же порядке.
3. Сформировать стек из **10** чисел. Извлечь элементы из стека, положительные заменить на 1, а отрицательные на -1 и снова поместить в стек в том же порядке.
4. Сформировать стек из **N** чисел. Найти количество элементов стека равных заданному числу **K**.
5. Сформировать стек из **N** чисел. Извлечь элементы из стека, найти их сумму и произведение. Результат поместить в стек.
6. Заполнить стек **N** случайными числами из интервала [-10;80]. Найти количество чисел, которые при делении на 5 дают в остатке 3.
7. Сформировать стек из **N** чисел. Найти сумму первых **K** чисел и результат поместить в стек.
8. Сформировать стек из **N** чисел. Найти произведение второго и третьего чисел. Результат поместить в стек.
9. Сформировать стек из **N** чисел. Найти сумму нечетных чисел из стека. Результат поместить в стек.
- 10.Сформировать стек из **N** чисел. Найти произведение элементов стека кратных 3 и 4. Результат вывести.
- 11.Заполнить стек **N** случайными числами из интервала [-10;20]. Найти максимальное число.
- 12.Сформировать стек из **N** чисел. Найти среднее геометрическое элементов стека.
- 13.Сформировать стек из **N** чисел. Найти количество элементов стека меньших заданного числа **K**.

- 14.Сформировать стек из **N** чисел. Извлечь элементы из стека, найти их среднее арифметическое. Результат поместить в стек.
- 15.Заполнить стек **N** случайными числами из интервала [10; 100]. Найти количество чисел, которые при делении на 7 дают в остатке 2.
- 16.Сформировать стек из **N** чисел. Найти сумму последних **K** чисел и результат поместить в стек.
- 17.Сформировать стек из **N** чисел. Найти среднее арифметическое второго и третьего чисел. Результат поместить в стек.
- 18.Заполнить стек **N** случайными числами из интервала [-50; 70]. Просмотреть содержимое стека. Найти сумму отрицательных чисел, хранящихся в стеке. Результат поместить в стек.
- 19.Заполнить стек **N** случайными числами из интервала [10;50]. Найти минимальное число.
- 20.Сформировать стек из **N** чисел. Извлечь элементы из стека, найти их среднее арифметическое. Результат поместить в стек.
- 21.Сформировать стек из **N** чисел. Найти сумму первых **N** чисел и результат поместить в стек.
- 22.Заполнить стек **N** случайными числами из интервала [10; 100]. Найти количество чисел, которые при делении на 7 дают в остатке 2.
- 23.Заполнить стек **N** случайными числами из интервала [10;80]. Найти сумму четных элементов стека. Результат поместить в стек.
- 24.Сформировать стек из **N** чисел. Найти среднее арифметическое элементов стека.
- 25.Сформировать стек из **N** чисел. Найти среднее геометрическое элементов стека.
- 26.Сформировать стек из **N** чисел. Найти количество элементов стека больших заданного числа **K**.
- 27.Сформировать стек из **N** чисел. Найти элемент стека наиболее близкий по значению, к заданному числу **K**.
- 28.Сформировать стек из **N** чисел. Найти количество элементов стека некратных заданному числу **K**.
- 29.Сформировать стек из **N** чисел. Найти количество элементов стека кратных 3 и не кратных 5. Подумайте, что сделать с числом 15?
- 30.Найти среднее арифметическое значение элементов целочисленной очереди и удалить элементы, меньшие среднего значения.
- 31.Найти среднее арифметическое значение элементов очереди вещественных чисел. Поместить ближайший к среднему значению элемент очереди на первую позицию.
- 32.Дан стек целых чисел. Удалить наибольший элемент стека.
- 33.Дан стек символов. Удалить каждый n-ый элемент стека.
- 34.Дан стек целых чисел. Удалить все четные числа из стека.
- 35.Дан стек символов. Удалить каждый второй четный элемент стека.

## Задание 2: «Очередь»

1. Дано очередь вещественных чисел. Удалить числа, меньшие среднего арифметического.
2. Дано очередь целых чисел. Удалить наименьший элемент очереди.
3. Дано очередь символов. Удалить каждый нечетный элемент очереди.
4. Дано очередь целых чисел. Удалить все нечетные числа из очереди.
5. Дано очередь вещественных чисел. Удалить из очереди числа из заданного пользователем диапазона.
6. Дано очередь вещественных чисел, упорядоченных по убыванию. Добавить в очередь среднее арифметическое элементов, не нарушая упорядоченности.
7. Дано очередь символов. Преобразовать очередь, оставив в нем из группы подряд идущих одинаковых элементов только один.
8. Дано очередь целых чисел. Удалить все локальные минимумы, при этом первый и последний элемент не обрабатывать.
9. Дан набор из чисел. Создать две очереди: первый должен содержать числа из исходного набора с нечетными номерами (1, 3, ...), а второй — с четными (2, 4, ...).
10. Разделить созданную очередь вещественных чисел на две: в первый — положительные числа, во второй — отрицательные.
11. В созданной очереди символов поменять местами крайние элементы.
12. Из очереди целых чисел удалить элементы, заканчивающиеся на цифру 5.
13. В очереди вещественных чисел поменять местами элементы, содержащие максимальное и минимальное значения.
14. Перенести из созданной очереди целых чисел в новую очередь все элементы, находящиеся между первым и максимальным элементом.
15. Перенести из созданной очереди вещественных чисел в новую очередь все элементы, находящиеся между первым и минимальным элементом.
16. В очереди целых чисел определить количество и удалить все элементы, находящиеся между минимальным и максимальным элементами.
17. В очереди вещественных чисел определить количество элементов, имеющих значения, меньше среднего арифметического значения всех элементов, и удалить эти элементы.
18. В очереди вещественных чисел вычислить среднее арифметическое и заменить им первый элемент.
19. В очереди вещественных чисел вычислить среднее арифметическое и заменить им все четные значения элементов.

20. В очереди целых чисел заменить все положительные цифры, отрицательными. И наоборот.
21. Объединить две целочисленных очереди в одну новую.
22. Удалить из очереди целых чисел все четные числа.
23. Удалить из очереди вещественных чисел все отрицательные числа.
24. Поменять местами крайние элементы очереди целых чисел.
25. Поменять местами минимальный и максимальный элементы очереди вещественных чисел.
26. Удалить из очереди символов каждый второй элемент.
27. Удалить из очереди целых чисел все элементы, расположенные до минимального элемента очереди.
28. Поменять местами наибольший среди отрицательных и наименьший среди положительных элементов целочисленной очереди.
29. Поместить максимальный элемент очереди вещественных чисел на первую позицию.
30. Поменять местами минимальный и первый элементы очереди целых чисел.
31. Поменять местами первый и последний элементы очереди вещественных чисел.
32. Удалить первый и последний элементы очереди целых чисел.
33. Удалить из очереди вещественных чисел все элементы, расположенные между минимальным и максимальным элементами.
34. Удалить из очереди целых чисел все элементы, расположенные после максимального элемента.

### **3. Оснащение работы**

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### **4. Основные теоретические сведения**

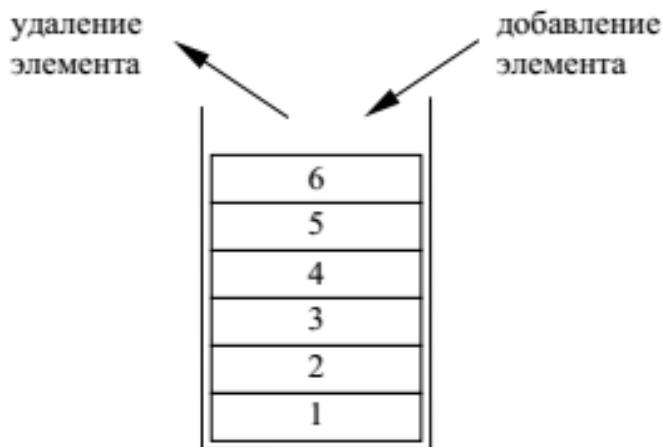
Структуры, перечисленные в заголовке раздела, представляют собой списки, для которых разрешены только операции вставки и удаления первого и/или последнего элемента.

#### **Стек**

Стек – это упорядоченный набор элементов, в котором добавление новых и удаление существующих элементов допустимо только с одного конца, который называется **вершиной стека**.

Стек называют структурой типа **LIFO** (*Last In – First Out*) – последним пришел, первым ушел. Стек похож на стопку с подносами,ложенными один на другой – чтобы достать какой-то поднос надо снять все подносы, которые

лежат на нем, а положить новый поднос можно только сверху всей стопки. На рисунке показан стек, содержащий 6 элементов.



В современных компьютерах стек используется для:

- ✓ размещения локальных переменных;
- ✓ размещения параметров процедуры или функции;
- ✓ сохранения адреса возврата (по какому адресу надо вернуться из процедуры);
- ✓ временного хранения данных, особенно при программировании на Ассемблере.

На стек выделяется ограниченная область памяти. При каждом вызове процедуры в стек добавляются новые элементы (параметры, локальные переменные, адрес возврата). Поэтому если вложенных вызовов будет много, стек переполнится. Очень опасной в отношении переполнения стека является **рекурсия**, поскольку она как раз и предполагает вложенные вызовы одной и той же процедуры или функции. При ошибке в программе рекурсия может стать бесконечной, кроме того, стек может переполниться, если вложенных вызовов будет слишком много.

### Реализация стека с помощью массива

Если максимальный размер стека заранее известен, его можно реализовать в программе в виде массива. Удобно объединить в одной структуре сам массив и его размер. Объявим новый тип данных – стек на 100 элементов-символов.

```
const int MAXSIZE = 100;
struct Stack {
    char data[MAXSIZE];
    int size;
};
```

Для работы со стеком надо определить, как выполняются две операции – добавление элемента на вершину стека (**Push**) и снятие элемента с вершины стека (**Pop**).

```

void Push ( Stack &S, char x )
{
    if ( S.size == MAXSIZE ) {
        printf ("Стек переполнен");
        return;
    }
    S.data[S.size] = x;
    S.size++;
}

```

Поскольку нумерация элементов массива **data** начинается с нуля, надо сначала записать новый элемент в **S.data[S.size]**, а затем увеличить размер стека. В процедуре предусмотрена обработка ошибки «переполнение стека». В этом случае на экран будет выдано сообщение «Стек переполнен». Можно также сделать функцию **Push**, которая будет возвращать 1 в случае удачного добавления элемента и 0 в случае ошибки.

Обратите внимание, что стек **S** передается в процедуру по ссылке, то есть, фактически передается адрес этого стека в памяти. Поэтому все операции со стеком в процедуре выполняются непосредственно со стеком вызывающей программы.

```

char Pop ( Stack &S )
{
    if ( S.size == 0 ) {
        printf ("Стек пуст");
        return char(255);
    }
    S.size--;
    return S.data[S.size];
}

```

Функция **Pop** возвращает символ, «снятый» с вершины стека, при этом размер стека уменьшается на единицу. Если стек пуст, функция возвращает символ с кодом 255 (который никогда не может находиться в стеке по условию задачи и сигнализирует об ошибке).

**Пример 14.1 Задача.** Ввести символьную строку, которая может содержать три вида скобок: (), [] и {}. Определить, верно ли расставлены скобки (символы между скобками не учитывать). Например, в строках () [{ }] и [{ } ([ ]) ] скобки расставлены верно, а в строках ([ ]) и ]]] (( - неверно.

Для одного вида скобок решение очень просто – ввести счетчик «вложенности» скобок, про-смотреть всю строку, увеличивая счетчик для каждой открывающей скобки и уменьшая его для каждой закрывающей. Выражение записано верно, если счетчик ни разу не стал отрицательным и после обработки всей строки оказался равен нулю.

Если используются несколько видов скобок, счетчики не помогают. Однако эта задача имеет красивое решение с помощью стека. Вначале стек пуст. Проходим всю строку от начала до символа с кодом 0, который обозначает конец строки. Если встретили открывающую скобку, заносим ее в стек. Если встретили закрывающую скобку, то на вершине стека должна быть

соответствующая ей открывающая скобка. Если это так, снимаем ее со стека. Если стек пуст или на вершине стека находится скобка другого вида, выражение неверное. В конце прохода стек должен быть пуст.

В приведенной ниже программе используются написанные ранее объявление структуры **Stack** и операции **Push** и **Pop**.

```
void main()
{
    char br1[3] = { '(', '[', '{' }; // открывающие скобки
    char br2[3] = { ')', ']', '}' }; // закрывающие скобки
    char s[80], upper;
    int i, k, OK;
    Stack S;           // стек символов

    printf("Введите выражение со скобками> ");
    gets ( s );

    S.size = 0;        // сначала стек пуст
    OK = 1;

    for (i = 0; OK && (s[i] != '\0'); i++)
        for (k = 0; k < 3; k++) // проверить 3 вида скобок
        {
            if (s[i] == br1[k]) { // открывающая скобка
                Push (S, s[i]); break;
            }
            if (s[i] == br2[k]) { // закрывающая скобка
                upper = Pop (S);
                if (upper != br1[k]) OK = 0;
                break;
            }
        }

    if (OK && (S.size == 0) )
        printf("\nВыражение правильное\n");
    else printf("\nВыражение неправильное \n");
}
```

Открывающие и закрывающие скобки записаны в массивах **br1** и **br2**. В самом начале стек пуст и его размер равен нулю (**S.size = 0**). Переменная **OK** служит для того, чтобы выйти из внешнего цикла, когда обнаружена ошибка (и не рассматривать оставшуюся часть строки). Она устанавливается в нуль, если в стеке обнаружена скобка другого типа или стек оказался пуст.

### Реализация стека с помощью списка

Рассмотрим пример стека, в котором хранятся символы (это простейший вариант, элементом стека могут быть любые типы данных или структур, так же, как и для списка). Реализуем стек на основе двусвязного списка. При этом количество элементов стека ограничивается только доступным объемом памяти.

```
struct Node {
    char data;
    Node *next, *prev;
};

typedef Node *PNode;
```

Чтобы не работать с отдельными указателями на хвост и голову списка, объявим структуру, в которой будет храниться вся информация о стеке:

```
struct Stack {  
    PNode Head, Tail;  
};
```

В самом начале надо записать в обе ссылки стека **NULL**.

### Добавление элемента на вершину стека

Фактически это добавление нового элемента в начало двусвязного списка. Эта процедура уже была написана ранее, теперь ее придется немного переделать, чтобы работать не с отдельными указателями, а со структурой типа **Stack**. В параметрах процедуры указывается не новый узел, а только *данные* для этого узла, то есть целое число. Память под новый узел выделяется в процедуре, то есть, скрыта от нас и снижает вероятность ошибки.

```
void Push ( Stack &S, char x )  
{  
    PNode NewNode;  
    NewNode = new Node;      // создать новый узел...  
    NewNode->data = x;     // и заполнить его данными  
    NewNode->next = S.Head;  
    NewNode->prev = NULL;  
    if ( S.Head )           // добавить в начало списка  
        S.Head->prev = NewNode;  
    S.Head = NewNode;  
    if ( ! S.Tail ) S.Tail = S.Head;  
}
```

### Получение верхнего элемента с вершины стека

Функция **Pop** удаляет верхний элемент и возвращает его данные.

```
char Pop ( Stack &S )  
{  
    PNode TopNode = S.Head;  
    char x;  
    if ( ! TopNode )      // если стек пуст, то  
        return char(255); // вернуть символ с кодом 255  
    x = TopNode->data;  
    S.Head = TopNode->next;  
    if ( S.Head ) S.Head->prev = NULL; // переставить ссылки  
    else          S.Tail = NULL;  
    delete TopNode; // освободить память  
    return x;  
}
```

### Системный стек в программах

При выполнении программ определенная область памяти отводится на стек программы. Более того, в процессоре есть специальная ячейка (регистр), в которой хранится адрес вершины стека. Программа использует стек для хранения

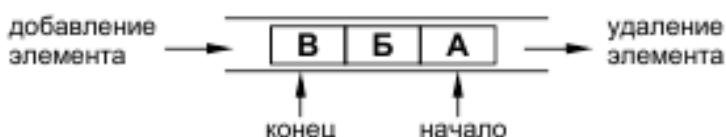
- ✓ адресов возврата из процедур и функций (это адреса, на которые переходит программа после выполнения процедуры или функции);
- ✓ параметров, передаваемых в процедуры и функции;
- ✓ локальных переменных в процедурах и функциях;
- ✓ временных данных (в основном в программах на ассемблере).

Больше всего места занимают в стеке локальные переменные. Поэтому память под большие массивы надо выделять динамически. Кроме того, желательно не передавать в процедуры большие структуры, вместо этого можно передать их адрес или использовать передачу по ссылке (при этом перед именем параметра должен стоять знак &).

### Очередь

**Очередь** – это упорядоченный набор элементов, в котором добавление новых элементов допустимо с одного конца (он называется **концом очереди**), а удаление существующих элементов – только с другого конца, который называется **началом очереди**.

Хорошо знакомой моделью является очередь в магазине. Очередь называют структурой типа **FIFO** (*First In – First Out*) – первым пришел, первым ушел. На рисунке изображена очередь из 3-х элементов.



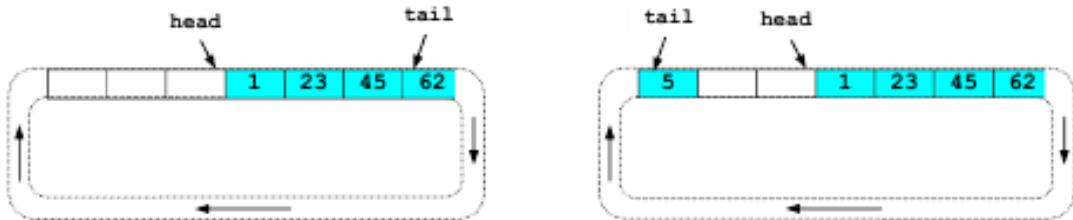
Наиболее известные примеры применения очередей в программировании – очередь событий системы *Windows* и ей подобных. Очереди используются также для моделирования в **задачах массового обслуживания** (например, обслуживания клиентов в банке).

### Реализация очереди с помощью массива

Если максимальный размер очереди заранее известен, его можно реализовать в программе в виде массива. Удобно объединить в одной структуре сам массив и его размер. Объявим новый тип данных – очередь на 100 элементов (целых чисел).

```
const int MAXSIZE = 100;
struct Queue {
    int data[MAXSIZE];
    int size, head, tail;
};
```

Если у стека один конец «закреплен» (не двигается), то у очереди «подвижны» оба конца. Чтобы не сдвигать все элементы в массиве при удалении или добавлении элемента, обычно использую две переменные **head** и **tail** – первая из них обозначает номер первого элемента в очереди, а вторая – номер последнего. Если они равны, то в очереди всего один элемент. Массив как бы замыкается в кольцо – если массив закончился, но в начале массива есть свободные места, то новый элемент добавляется в начало массива, как показано на рисунках.



Для работы с очередью надо определить, как выполняются две операции – добавление элемента в конец очереди (**PushTail**) и удаление элемента с начала очереди (**Pop**).

```
void PushTail ( Queue &Q, int x )
{
    if ( Q.size == MAXSIZE ) {
        printf ("Очередь переполнена\n");
        return;
    }
    Q.tail++;
    if ( Q.tail >= MAXSIZE ) // замыкание в кольцо
        Q.tail = 0;
    Q.data[Q.tail] = x;
    Q.size++;
}
```

Поскольку очередь может начинаться не с начала массива (за счет того, что некоторые элементы уже «выбраны»), после увеличения **Q.tail** надо проверить, не вышли ли мы за границу массива. Если это случилось, новый элемент записывается в начало массива (хотя является хвостом очереди). В процедуре предусмотрена обработка ошибки «переполнение очереди». В этом случае на экран будет выдано сообщение «Очередь переполнена». Можно также сделать функцию **PushTail**, которая будет возвращать 1 в случае удачного добавления элемента и 0 в случае ошибки.

Обратите внимание, что очередь **Q** передается в процедуру по ссылке, то есть, фактически передается адрес этой структуры в памяти.

```
int Pop ( Queue &Q )
{
    int temp;
    if ( Q.size == 0 ) {
        printf ("Очередь пуста\n");
        return 32767;      // сигнал об ошибке
    }
    temp = Q.data[Q.head];
    Q.head++;
    if ( Q.head >= MAXSIZE ) Q.head = 0;
    Q.size--;
    return temp;
}
```

Функция **Pop** возвращает число, полученное с начала очереди, при этом размер очереди уменьшается на единицу. Если стек пуст, функция возвращает число 32767 (предполагается, что оно не может находиться в очереди по условию задачи и сигнализирует об ошибке).

## Реализация очереди с помощью списка

Если максимальный размер заранее неизвестен или требуется сделать его динамическим, для реализации используют список. Рассмотрим пример очереди, элементами которой являются целые числа. При этом количество ее элементов ограничивается только доступным объемом памяти. Новые типы данных (узел и указатель на него) объявляются так же, как для списка:

```
struct Node {  
    int data;  
    Node *next, *prev;  
};  
typedef Node *PNode;
```

Чтобы не работать с отдельными указателями на хвост и голову списка, объявим структуру, в которой будет храниться вся информация об очереди:

```
struct Queue {  
    PNode head, tail;  
};
```

В самом начале надо записать в обе ссылки **NULL**. Заметим, что такая же структура использовалась и для стека. Более того, функция для получения первого элемента очереди (**Pop**) совпадает с функцией снятия элемента с вершины стека (напишите ее в качестве упражнения).

### Добавление элемента в конец очереди

Фактически это добавление нового элемента в конец двусвязного списка. В параметрах процедуры указывается не новый узел, а только *данные* для этого узла, то есть целое число. Память под новый узел выделяется в процедуре, то есть, скрыта от нас и снижает вероятность ошибки.

```
void PushTail ( Queue &Q, int x )  
{  
    PNode NewNode;  
    NewNode = new Node;           // создать новый узел  
    NewNode->data = x;          // заполнить узел данными  
    NewNode->prev = Q.tail;  
    NewNode->next = NULL;  
  
    if ( Q.tail )               // добавить узел в конец списка  
        Q.tail->next = NewNode;  
    Q.tail = NewNode;  
  
    if ( ! Q.head ) Q.head = Q.tail;  
}
```

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Дайте определение стеку.
2. Где применяются стеки?
3. Расскажите о внутренней структуре стека.
4. Дайте определение очереди.
5. Где применяются очереди?
6. Расскажите о внутренней структуре очереди.

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
3. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
4. Окулов С.М. Программирование в алгоритмах – М.: БИНОМ. Лаборатория знаний, 2006. - 383 с.
5. Прата С. Язык программирования С. - СПб.: Вильямс, 2013. – 960 с.
6. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400
7. Хьюз Дж. Структурный подход к программированию / Мичтом Дж. – М.: Мир, 1980. –278 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание программ обработки бинарных деревьев»

Минск  
2017

## **Лабораторная работа № 15**

**Тема работы:** «Разработка, отладка и испытание программ обработки бинарных деревьев»

### **1. Цель работы**

Закрепить навык обработки бинарных деревьев.

### **2. Задание**

Вариант соответствует вашему номеру по списку.

1. Определите, есть ли в данном бинарном дереве два одинаковых элемента (дерево не является бинарным деревом поиска).
2. Выведите номера уровней данного бинарного дерева, на которых имеются листья.
3. Выведите номера вершин, у которых количество потомков в левом поддереве не равно количеству потомков в правом поддереве.
4. Выведите номера вершин, для которых высота левого поддерева не равна высоте правого поддерева.
5. Выведите номера вершин, у которых количество потомков в левом поддереве отличается от количества потомков в правом поддереве на 1.
6. Найдите высоту дерева  $H$  и удалите из него (с перестройкой) все вершины на уровне  $H/2$ .
7. Найдите минимальный путь между двумя произвольными листьями.
8. Найдите минимальный путь между двумя произвольными вершинами дерева.
9. Найдите высоту дерева  $H$  и удалите в нем все вершины (с перестройкой) на глубине  $H/2$ , у которых высота левого поддерева равна высоте правого поддерева.
10. Найдите путь максимальной длины и отразите дерево зеркально относительно этого пути.
11. Найдите путь максимальной длины между двумя произвольными вершинами с разным числом потомков.
12. Найдите путь максимальной длины между двумя произвольными вершинами разной высоты.
13. Найдите пути минимальной длины между корнем и листьями.
14. Определите, являются ли два дерева зеркальным отражением друг друга.
15. Найдите среднюю по значению вершину в дереве (вершину, у которой значение ближе всего по модулю к среднему арифметическому значений всех вершин).
16. Найдите вершины, у которых высоты поддеревьев равны, а количество потомков в правом и левом поддеревьях не равны.
17. Найдите вершины, у которых высоты поддеревьев не равны, а количество потомков в правом и левом поддеревьях равны.

18. Удалите все вершины, для которых количество потомков в левом поддереве отличается от количества вершин в правом поддереве на 2 и более. i
19. Удалите все вершины, у которых высота левого поддерева отличается от высоты правого поддерева на 2.
20. Выясните, является ли дерево симметричным.
21. Вычислите количество вершин, для которых высота левого поддерева равна высоте правого поддерева.
22. Вычислите количество вершин, у которых равны или высоты поддеревьев, или количество потомков в правом и левом поддеревьях.
23. Найти среднее арифметическое элементов бинарного дерева.
24. Определить среднее арифметическое, какого поддерева (относительно корня) больше.
25. Заменить каждое значение узла дерева на его квадрат, бес перестройки дерева (не меняя его структуру).
26. Найти сумму элементов расположенных в листьях бинарного дерева.
27. Найти количество узлов расположенных на последнем уровне бинарного дерева.
28. В бинарном дереве поменять места максимальный и минимальный элементы.
29. Найти количество элементов бинарного дерева, которые имеют два потомка.
30. Найти сумму элементов бинарного дерева, которые имеют только одного потомка.
31. Найти количество элементов бинарного дерева, которые имею только левого потомка.

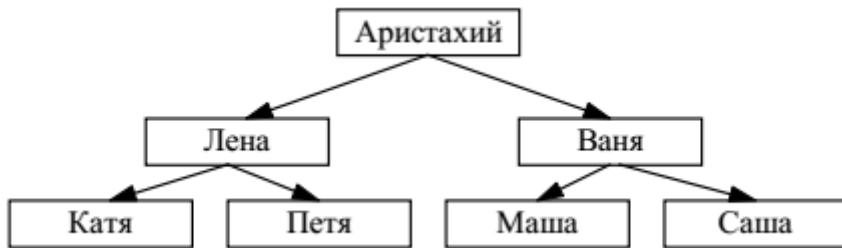
### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

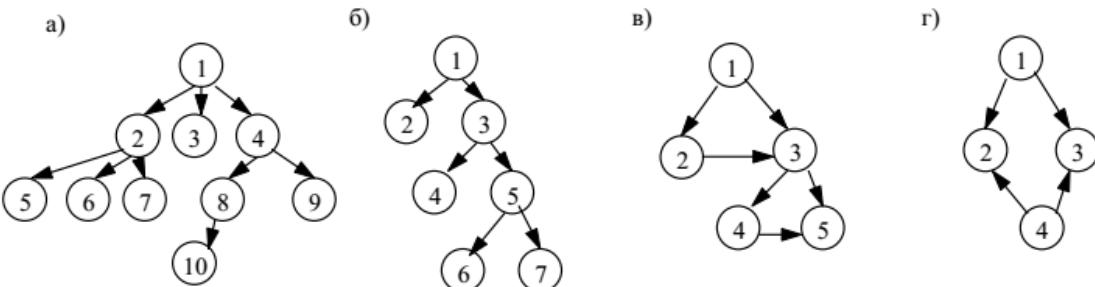
### 4. Основные теоретические сведения

**Дерево** – это совокупность **узлов** (**вершин**) и соединяющих их направленных **ребер** (**дуг**), причем в каждый узел (за исключением одного - **корня**) ведет ровно одна дуга. **Корень** – это начальный узел дерева, в который не ведет ни одной дуги.

Примером может служить **генеалогическое дерево** - в корне дерева находитесь вы сами, от вас идет две дуги к родителям, от каждого из родителей - две дуги к их родителям и т.д.



Например, на рисунке структуры а) и б) являются деревьями, а в) и г) - нет.



**Предком** для узла  $x$  называется узел дерева, из которого существует путь в узел  $x$ .

**Потомком** узла  $x$  называется узел дерева, в который существует путь (по стрелкам) из узла  $x$ .

**Родителем** для узла  $x$  называется узел дерева, из которого существует непосредственная дуга в узел  $x$ .

**Сыном** узла  $x$  называется узел дерева, в который существует непосредственная дуга из узла  $x$ .

**Уровнем** узла  $x$  называется длина пути (количество дуг) от корня к данному узлу. Считается, что корень находится на уровне 0.

**Листом дерева** называется узел, не имеющий потомков.

**Внутренней вершиной** называется узел, имеющий потомков.

**Высотой дерева** называется максимальный уровень листа дерева.

**Упорядоченным деревом** называется дерево, все вершины которого упорядочены (то есть имеет значение последовательность перечисления потомков каждого узла).

Например, два упорядоченных дерева на рисунке ниже – разные.



### Рекурсивное определение

Дерево представляет собой типичную рекурсивную структуру (определяемую через саму себя). Как и любое рекурсивное определение, определение дерева состоит из двух частей – первая определяет условие окончания рекурсии, а второе – механизм ее использования.

- ✓ пустая структура является деревом;

- ✓ дерево – это корень и несколько связанных с ним деревьев (поддеревьев).

Таким образом, размер памяти, необходимый для хранения дерева, заранее неизвестен, потому что неизвестно, сколько узлов будет в нем входить.

### Двоичные деревья

На практике используются главным образом деревья особого вида, называемые **двоичными (бинарными)**.

**Двоичным деревом** называется дерево, каждый узел которого имеет не более двух сыновей.

Можно определить двоичное дерево рекурсивно:

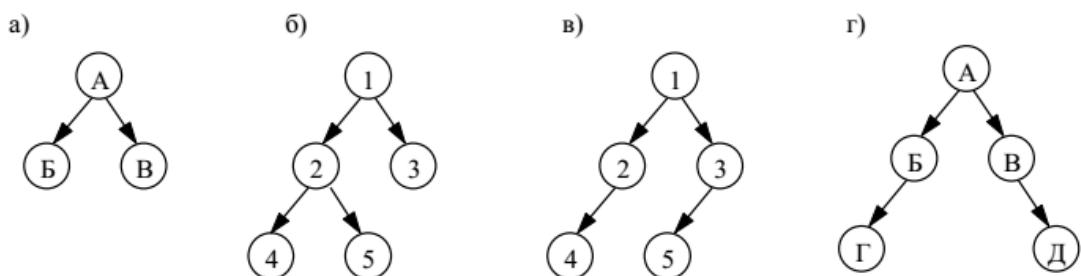
- ✓ пустая структура является двоичным деревом;
- ✓ дерево – это корень и два связанных с ним двоичных дерева, которые называют **левым и правым** поддеревом .

Двоичные деревья **упорядочены**, то есть различают левое и правое поддеревья. Типичным примером двоичного дерева является генеалогическое дерево (родословная). В других случаях двоичные деревья используются тогда, когда на каждом этапе некоторого процесса надо принять одно решение из двух возможных. В дальнейшем мы будем рассматривать только двоичные деревья.

**Строго двоичным деревом** называется дерево, у которого каждая внутренняя вершина имеет непустые левое и правое поддеревья.

Это означает, что в строго двоичном дереве нет вершин, у которых есть только одно поддерево.

На рисунке даны деревья а) и б) являются строго двоичными, а в) и г) – нет.



**Полным двоичным деревом** называется дерево, у которого все листья находятся на одном уровне и каждая внутренняя вершина имеет непустые левое и правое поддеревья.

На рисунке выше только дерево а) является полным двоичным деревом.

### Реализация двоичных деревьев в языке Си

#### Описание вершины

Вершина дерева, как и узел любой динамической структуры, имеет две группы данных: полезную информацию и ссылки на узлы, связанные с ним. Для двоичного дерева таких ссылок будет две – ссылка на **левого сына** и ссылка на **правого сына**. В результате получаем структуру, описывающую

вершину (предполагая, что полезными данными для каждой вершины является одно целое число):

```
struct Node {  
    int key;           // полезные данные (ключ)  
    Node *left, *right; // указатели на сыновей  
};  
typedef Node *PNode;      // указатель на вершину
```

### Деревья минимальной высоты

Для большинства практических задач наиболее интересны такие деревья, которые имеют минимально возможную высоту при заданном количестве вершин  $n$ . Очевидно, что минимальная высота достигается тогда, когда на каждом уровне (кроме, возможно, последнего) будет максимально возможное число вершин.

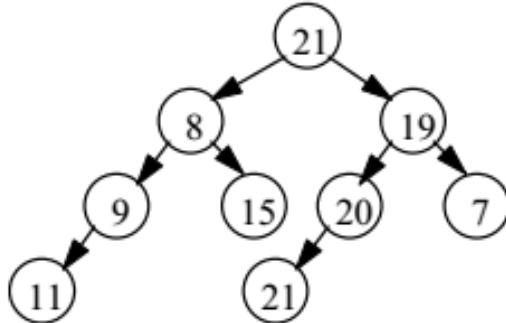
Предположим, что задано  $n$  чисел (их количество заранее известно). Требуется построить из них дерево минимальной высоты. Алгоритм решения этой задачи предельно прост.

- ✓ Взять одну вершину в качестве корня и записать в нее первое нерассмотренное число.
- ✓ Построить этим же способом левое поддерево из  $n_1 = n/2$  вершин (деление нацело!).
- ✓ Построить этим же способом правое поддерево из  $n_2 = n - n_1 - 1$  вершин.

Заметим, что по построению левое поддерево всегда будет содержать столько же вершин, сколько правое поддерево, или на 1 больше. Для массива данных

**21, 8, 9, 11, 15, 19, 20, 21, 7**

по этому алгоритму строится дерево, показанное на рисунке снизу.



Как будет выглядеть эта программа на языке Си? Надо сначала разобраться, что означает «взять одну вершину в качестве корня и записать туда первое нерассмотренное число». Поскольку вершины должны создаваться динамически, надо **выделить память** под вершину и записать в поле данных нужное число. Затем из оставшихся чисел построить левое и правое поддеревья.

В основной программе нам надо объявить указатель на корень нового дерева, задать массив данных (в принципе можно читать данные из файла) и вызвать функцию, возвращающую указатель на построенное дерево.

```

int data[] = { 21, 8, 9, 11, 15, 19, 20, 21, 7 };
PNode Tree; // указатель на корень дерева
n = sizeof(data) / sizeof(int) - 1; // размер массива
Tree = MakeTree (data, 0, n); // использовать п элементов,
// начиная с номера 0

```

Сама функция MakeTree принимает три параметра: массив данных, номер первого неиспользованного элемента и количество элементов в новом дереве. Возвращает она указатель на новое дерево (типа PNode).

```

PNode MakeTree (int data[], int from, int n)
{
PNode Tree;
int n1, n2;
if ( n == 0 ) return NULL; // ограничение рекурсии
Tree = new Node; // выделить память под вершину
Tree->key = data[from]; // записать данные (ключ)

n1 = n / 2; // размеры поддеревьев
n2 = n - n1 - 1;
Tree->left = MakeTree(data, from+1, n1);
Tree->right = MakeTree(data, from+1+n1, n2);
return Tree;
}

```

Выделенные строчки программы содержат рекурсивные вызовы. При этом левое поддерево содержит **n1** элементов массива начиная с номера **from+1**, тогда как правое – **n2** элементов начиная с номера **from+1+n1**.

### Обход дерева

Одной из необходимых операций при работе с деревьями является **обход дерева**, во время которого надо посетить каждый узел по одному разу и (возможно) вывести информацию, содержащуюся в вершинах.

Пусть в результате обхода надо напечатать значения поля данных всех вершин в определенном порядке. Существуют **три варианта обхода**:

- ✓ **КЛП (корень – левое – правое)**: сначала посещается корень (выводится информация о нем), затем левое поддерево, а затем – правое;
- ✓ **ЛКП (левое – корень – правое)**: сначала посещается левое поддерево, затем корень, а за-тем – правое;
- ✓ **ЛПК (левое – правое – корень)**: сначала посещается левое поддерево, затем правое, а за-тем – корень.

Для примера ниже дана рекурсивная процедура просмотра дерева в порядке ЛКП. Обратите внимание, что поскольку дерево является рекурсивной структурой данных, при работе с ним естественно широко применять рекурсию.

```

void PrintLKP(PNode Tree)
{
if ( ! Tree ) return; // пустое дерево – окончание рекурсии
PrintLKP(Tree->left); // обход левого поддерева
printf("%d ", Tree->key); // вывод информации о корне
PrintLKP(Tree->right); // обход правого поддерева
}

```

Остальные варианты обхода программируются аналогично.

### **5. Порядок выполнения работы**

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

### **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_  
\_\_\_\_\_

### **7. Контрольные вопросы и задания**

1. Дать определение структуре дерева?
2. Что такое двоичное дерево?
3. Корень – это?
4. Высота дерева – это?
5. Потомок – это?
6. Лист – это?
7. Какие можно реализовать обходы дерева?

### **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
3. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
4. Окулов С.М. Программирование в алгоритмах – М.: БИНОМ. Лаборатория знаний, 2006. - 383 с.
5. Прата С. Язык программирования С. - СПб.: Вильямс, 2013. – 960 с.
6. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400

7. Хьюз Дж. Структурный подход к программированию / Мичтом Дж.  
– М.: Мир, 1980. –278 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание программ обработки деревьев поиска»

Минск  
2017

# **Лабораторная работа № 16**

**Тема работы:** «Разработка, отладка и испытание программ обработки деревьев поиска»

## **1. Цель работы**

Отработать навык обработки деревьев поиска.

## **2. Задание**

В данных задачах используются только бинарные деревья поиска.  
Вариант соответствует вашему номеру по списку.

- 1.** Написать рекурсивную числовую функцию, подсчитывающую сумму элементов дерева.
- 2.** Написать функцию, которая находит наибольший элемент дерева.
- 3.** Написать функцию, которая находит наименьший элемент дерева.
- 4.** Напишите процедуру, которая удаляет из дерева все четные элементы.
- 5.** Написать рекурсивную процедуру, которая определяет число вхождений заданного элемента в дерево.
- 6.** Написать рекурсивную процедуру, которая печатает элементы из всех листьев дерева.
- 7.** Написать рекурсивную функцию, которая определяет глубину заданного элемента на дереве и возвращает  $-1$ , если такого элемента нет.
- 8.** Написать процедуру, которая печатает (по одному разу) все вершины дерева.
- 9.** Написать процедуру, которая по заданному  $n$  считает число всех вершин глубины  $n$  в заданном дереве.
- 10.** Написать процедуру, которая считает глубину дерева.
- 11.** Отсортировать массив  $A$  путем включения его элементов в дерево и скопировать отсортированные данные обратно в  $A$ .
- 12.** Задана последовательность слов. Определить частоту вхождения каждого из слов в последовательность.
- 13.** Указание. Для решения задачи любое слово ищется в дереве, которое на начальном этапе пусто. Если слово найдено, то счетчик его вхождений увеличивается на единицу, если нет, то слово включается в дерево с единичным значением счетчика.
- 14.** Подсчитать длину максимального пути и выдать его трассировку.
- 15.** Выдать метки всех вершин с их кратностями
- 16.** Выделить метку вершины дерева, имеющую наибольшее число вхождений
- 17.** В указанном ярусе дерева добавить вершину с указанной меткой

- 18.** Составить программу, реализующую сортировку линейного числового списка на основе бинарного отсортированного дерева
- 19.** Проверить дерево на изоморфизм
- 20.** Выделить в дереве все поддеревья с указанной меткой
- 21.** Реализовать обход дерева в глубину с выдачей меток
- 22.** Реализовать поиск в дереве данного поддерева
- 23.** Подсчитать количество вхождений в дерево данной метки
- 24.** Выделить в дереве максимальное поддерево с заданной меткой
- 25.** Подсчитать в дереве количество совпадающих вершин
- 26.** Выдать элемент бинарного дерева, который имеет в нем наибольшее число вхождений
- 27.** Сравнить два дерева на изоморфизм
- 28.** Выдать метки всех вершин с их кратностями
- 29.** Реализовать поиск в дереве данного поддерева
- 30.** Выделить в дереве все поддеревья с указанной меткой
- 31.** Подсчитать количество вхождений в дерево данной метки
- 32.** Подсчитать длину максимального пути и выдать его трассировку
- 33.** Подсчитать количество вершин дерева. Выдать наиболее длинный путь дерева.

### **3. Оснащение работы**

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### **4. Основные теоретические сведения**

#### **Поиск с помощью дерева**

#### **Как быстрее искать?**

Деревья очень удобны для поиска в них информации. Однако для быстрого поиска требуется предварительная подготовка – дерево надо построить специальным образом.

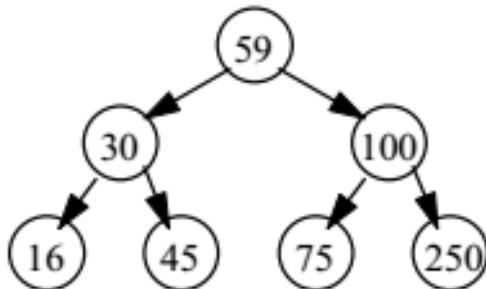
Предположим, что существует массив данных и с каждым элементом связан ключ - число, по которому выполняется поиск. Пусть ключи для элементов таковы:

**59, 100, 75, 30, 16, 45, 250**

Для этих данных нам надо много раз проверять, есть ли среди ключей заданный ключ **x**, и если есть, то вывести всю связанную с этим элементом информацию.

Если данные организованы в виде массива (без сортировки) то для поиска в худшем случае надо сделать **n** сравнений элементов (сравнивая

последовательно с каждым элементом пока не найдется нужный или пока не закончится массив).



Теперь предположим, что данные организованы в виде дерева, показанного на рисунке.

Такое дерево (оно называется **дерево поиска**) обладает следующим важным свойством:

Значения ключей всех вершин левого поддерева вершины **x** меньше ключа **x**, а значения ключей всех вершин правого поддерева **x** больше или равно ключу вершины **x**.

Для поиска нужного элемента в таком дереве требуется не более 3 сравнений вместо 7 при поиске в списке или массиве, то есть поиск проходит значительно быстрее. С ростом количества элементов эффективность поиска по дереву растет.

#### Построение дерева поиска

Как же, имея массив данных, построить такое дерево?

- ✓ Сравнить ключ очередного элемента массива с ключом корня.
- ✓ Если ключ нового элемента меньше, включить его в левое поддерево, если больше или равен, то в правое.
- ✓ Если текущее дерево пустое, создать новую вершину и включить в дерево.

Программа, приведенная ниже, реализует этот алгоритм:

```
void AddToTree (PNode &Tree, // указатель на корень (ссылка)
                int data) // добавляемый ключ
{
    if ( ! Tree ) {
        Tree = new Node; // создать новый узел
        Tree->key = data;
        Tree->left = NULL;
        Tree->right = NULL;
        return;
    }

    if ( data < Tree->key ) // добавить в нужное поддерево
        AddToTree ( Tree->left, data );
    else AddToTree ( Tree->right, data );
}
```

Важно, что указатель на корень дерева надо передавать по ссылке, так как он может измениться при создании новой вершины.

Надо заметить, что в результате работы этого алгоритма не всегда получается дерево минимальной высоты – все зависит от порядка выбора элементов. Для оптимизации поиска используют так называемые **сбалансированные** или **АВЛ-деревья** деревья, у которых для любой вершины высоты левого и правого поддеревьев отличаются не более, чем на 1. Добавление в них нового элемента иногда сопровождается некоторой перестройкой дерева.

### Поиск по дереву

Теперь, когда дерево сортировки построено, очень легко искать элемент с заданным ключом. Сначала проверяем ключ корня, если он равен искомому, то нашли. Если он меньше искомого, ищем в левом поддереве корня, если больше – то в правом . Приведенная функция возвращает адрес нужной вершины, если поиск успешный, и **NULL**, если требуемый элемент не найден.

```
PNode Search (PNode Tree, int what)
{
    if ( ! Tree ) return NULL; // ключ не найден
    if ( what == Tree->key ) return Tree; // ключ найден!
    if ( what < Tree->key ) // искать в поддеревьях
        return Search ( Tree->left, what );
    else return Search ( Tree->right, what );
}
```

### Сортировка с помощью дерева поиска

Если дерево поиска построено, очень просто вывести отсортированные данные. действительно, обход типа ЛКП ( *левое поддерево – корень – правое поддерево* ) даст ключи в порядке возрастания, а обход типа ПКЛ ( *правое поддерево – корень – левое поддерево* ) – в порядке убывания.

### Поиск одинаковых элементов

Приведенный алгоритм можно модифицировать так, чтобы быстро искать одинаковые элементы в массиве чисел. Конечно, можно перебрать все элементы массива и сравнить каждый со всеми остальными. Однако для этого требуется очень большое число сравнений. С помощью двоичного дерева можно значительно ускорить поиск. Для этого надо в структуру вершины включить еще одно поле – счетчик найденных дубликатов **count**.

```
struct Node {
    int key;
    int count; // счетчик дубликатов
    Node *left, *right;
};
```

При создании узла в счетчик записывается единица (найден один элемент). Поиск дубликатов происходит по следующему алгоритму:

- ✓ сравнить ключ очередного элемента массива с ключом корня;
- ✓ если ключ нового элемента равен ключу корня, то увеличить счетчик корня и стоп;
- ✓ если ключ нового элемента меньше, чем у корня, включить его в левое поддерево, если больше или равен – в правое;

- ✓ Если текущее дерево пустое, создать новую вершину (со значением счетчика 1) и включить в дерево.

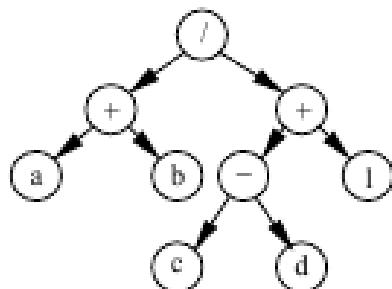
## Разбор арифметического выражения

### Дерево для арифметического выражения

Вы задумывались над тем, как транслятор обрабатывает и выполняет арифметические и логические выражения, которые он встречает в программе? Один из вариантов - представить это выражение в виде двоичного дерева. Например, выражению

$$(a + b) / (c - d + 1)$$

соответствует дерево, показанное на рисунке слева. Листья содержат числа и имена переменных (операндов), а внутренние вершины и корень – арифметические действия и вызовы функций. Вычисляется такое выражение снизу, начиная с листьев. Как видим, скобки отсутствуют, и дерево полностью определяет порядок выполнения операций.



### Формы записи арифметического выражения

Теперь посмотрим, что получается при прохождении таких двоичных деревьев. Прохождение дерева в ширину (корень – левое – правое) дает

$$/ + a \ b + - c \ d \ 1$$

то есть знак операции (корень) предшествует своим operandам. Такая форма записи арифметических выражений называется **префиксной**. Проход в прямом порядке (левое – корень – правое) дает **инфиксную форму**, которая совпадает с обычной записью, но без скобок:

$$a + b / c - d + 1$$

Поскольку скобок нет, по инфиксной записи невозможно восстановить правильный порядок операций.

В трансляторах широко используется **постфиксная запись** выражений, которая получается в результате обхода в порядке ЛПК (левое – правое – корень). В ней знак операции стоит **после** обоих operandов:

$$a \ b \ + \ c \ d \ - \ 1 \ /$$

Порядок выполнения такого выражения однозначно определяется следующим алгоритмом, который использует стек:

Пока в постфиксной записи есть невыбранные элементы,

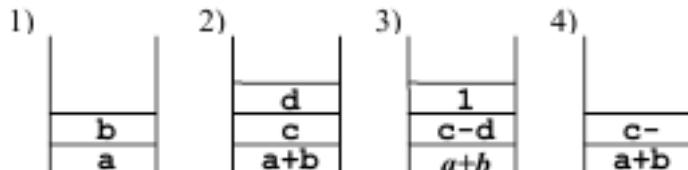
- ✓ взять очередной элемент;
- ✓ если это operand (не знак операции), то записать его в стек;
- ✓ если это знак операции, то
  - выбрать из стека второй operand;

- выбрать из стека первый операнд;
- выполнить операцию с этими данными и результат записать в стек.

Проиллюстрируем на примере вычисление выражения в постфиксной форме

**a b + c d - 1 /**

Согласно алгоритму, сначала запишем в стек **a**, а затем **b** (рисунок 1).



Результат выполнения операции **a+b** запишем обратно в стек, а сверху – выбранные из входного потока значения переменных **c** и **d** (рисунок 2). Дальнейшее развитие событий показано на рисунках 3 и 4. Выполнение последней операции (деления) для стека на рисунке 4 дает искомый результат.

### Алгоритм построения дерева

Пусть задано арифметическое выражение. Надо построить для него дерево синтаксического разбора и различные формы записи.

Чтобы не слишком усложнять задачу, рассмотрим самый простой вариант, введя следующие упрощения.

- В выражении могут присутствовать только однозначные целые числа знаки операций **+ - \* /**.
- Запрещается использование вызовов функций, скобок, унарных знаков плюс и минус (например, запрещено выражение **-a+5**, вместо него надо писать **0-a+5**).
- Предполагается, что выражение записано верно, то есть не делается проверки на правильность.

Вспомним, что порядок выполнения операций в выражении определяется **приоритетом операций** – первыми выполняются операции с более высоким приоритетом. Например, умножение и деление выполняются раньше, чем сложение и вычитание.

Будем правильное арифметическое выражение записано в виде символьной строки **Expr** длиной **N**. Построим дерево для элементов массива с номерами от **first** до **last** (полное дерево дает применение этого алгоритма ко всему массиву, то есть при **first=0** и **last=N-1**). В словесном виде алгоритм выглядит так:

- ✓ Если **first=last** (остался один элемент – переменная или число), то создать новый узел и записать в него этот элемент. Иначе...
- ✓ Среди элементов от **first** до **last** включительно найти **последнюю** операцию с наибольшим приоритетом (пусть найденный элемент имеет номер **k**).

- ✓ Создать новый узел (корень) и записать в него знак операции **Expr[k]**.
- ✓ Рекурсивно применить этот алгоритм два раза:
  - построить левое поддерево, разобрав выражение из элементов массива с номерами от **first** до **k-1**
  - построить правое поддерево, разобрав выражение из элементов массива с номерами от **k+1** до **last**

Объявим структуру, описывающую узел такого дерева. Так как мы используем только однозначные целые числа и знаки, область данных может содержать один символ.

```
struct Node {
    char data;
    Node *left, *right;
};
typedef Node *PNode;
```

Далее надо определить функцию, возвращающую приоритет операции, которая ей передана.

Определим приоритет 1 для сложения и вычитания и приоритет 2 для умножения и деления.

```
int Priority ( char c )
{
    switch ( c ) {
        case '+': case '-': return 1;
        case '*': case '/': return 2;
    }
    return 100; // это не арифметическая операция
}
```

Приведенная ниже процедура строит требуемое дерево, используя эту функцию, и возвращает адрес построенного дерева в памяти. Обратите внимание, что при сравнении приоритета текущей операции с минимальным предыдущим используется условие **<=**. За счет этого мы ищем именно **последнюю** операцию с минимальным приоритетом, то есть, операцию, которая будет выполняться самой последней. Если бы мы использовали знак **<**, то нашли бы  **первую** операцию с наименьшим приоритетом, и дерево было бы построено неверно (вычисления дают неверный результат, если встречаются два знака вычитания или деления).

```
PNode MakeTree (char Expr[], int first, int last)
{
    int MinPrt, i, k, prt;
    PNode Tree = new Node; // создать в памяти новую вершину
    if ( first == last ) { // конечная вершина: число или
        Tree->data = Expr[first]; // переменная
```

```

Tree->left = NULL;
Tree->right = NULL;
return Tree;
}

MinPrt = 100;
for ( i = first; i <= last; i ++ ) {
    prt = Priority ( Expr[i] );
    if ( prt <= MinPrt ) { // ищем последнюю операцию
        MinPrt = prt;      // с наименьшим приоритетом
        k = i;
    }
}
Tree->data = Expr[k]; // внутренняя вершина (операция)
Tree->left = MakeTree (Expr,first,k-1); // рекурсивно строим
Tree->right = MakeTree (Expr,k+1,last); // поддеревья
return Tree;
}

```

Теперь обход этого дерева разными способами дает различные формы представления соответствующего арифметического выражения.

### Вычисление выражения по дереву

Пусть для некоторого арифметического выражения построено дерево и известен его адрес **Tree**. Напишем функцию, которая возвращает целое число – результат вычисления этого выражения. Учтем, что деление выполняется нацело (остаток отбрасывается).

```

int CalcTree (PNode Tree)
{
    int num1, num2;
    if ( ! Tree->left )           // если нет потомков,
        return Tree->data - '0';   // вернули число
    num1 = CalcTree(Tree->left); // вычисляем поддеревья
    num2 = CalcTree(Tree->right);
    switch ( Tree->data ) {       // выполняем операцию
        case '+': return num1+num2;
        case '-': return num1-num2;
        case '*': return num1*num2;
        case '/': return num1/num2;
    }
    return 32767; // неизвестная операция, ошибка!
}

```

Если дерево не имеет потомков, значит это число. Чтобы получить результат как целое число, из кода этой цифры надо вычесть код цифры '**0**'. Если потомки есть, вычисляем левое и правое поддеревья (рекурсивно !) и выполняем операцию, записанную в корне дерева. Основная программа может выглядеть так, как показано ниже.

```

void main()
{
    char s[80];
    PNode Tree;
    printf("Введите выражение > ");
}

```

```
    gets(s);
    Tree = MakeTree(s, 0, strlen(s)-1);
    printf ("= %d \n", CalcTree ( Tree ) );
    getch();
}
```

## Разбор выражения со скобками

Немного усложним задачу, разрешив использовать в выражении скобки одного вида (допустим, круглые). Тогда при поиске в заданном диапазоне операции с минимальным приоритетом не надо брать во внимание выражения в скобках (они выделены на рисунке).

1 + ((2 + 3) \* 5 + 3) \* 7

Самый простой способ добиться этого эффекта – ввести счетчик открытых скобок **nest**. В начале он равен нулю, с каждой найденной открывающей скобкой будем увеличивать его на 1, а с каждой закрывающей – уменьшать на 1. Рассматриваются только те операции, которые найдены при **nest=0**, то есть, расположены вне скобок.

Если же ни одной такой операции не найдено, то мы имеем выражение, ограниченное скобками, поэтому надо вызвать процедуру рекурсивно для диапазона **from+1..last-1** (напомним, что мы предполагаем, что выражение корректно). Для сокращения записи показаны только те части процедуры, которые изменяются:

```

PNode MakeTree (char Expr[], int first, int last)
{
    int MinPrt, i, k, prt;
    int nest = 0; // счетчик открытых скобок
    PNode Tree = new Node;
    ...
    MinPrt = 100;
    for ( i = first; i <= last; i ++ ) {
        if ( Expr[i] == '(' ) // открывающая скобка
            { nest++; continue; }
        if ( Expr[i] == ')' ) // закрывающая скобка
            { nest--; continue; }
        if ( nest > 0 ) continue; // пропускаем все, что в скобках
        prt = Priority ( Expr[i] );
        if ( prt <= MinPrt ) {
            MinPrt = prt;
            k = i;
        }
    }
    if ( MinPrt == 100 && // все выражение взято в скобки
        Expr[first]== '(' && Expr[last]== ')' ) {
        delete Tree;
        return MakeTree(Expr, first+1, last-1);
    }
    ...
    return Tree;
}

```

Поскольку новый узел создается в самом начале функции, его надо удалить, если все выражение взято в скобки.

### Многозначные числа и переменные

Для хранения многозначных чисел и имен переменных надо использовать массив символов в области данных узла.

```

struct Node {
    char data[40];
    Node *left, *right;
};
typedef Node *PNode;

```

Будем по-прежнему считать, что выражение не содержит ошибок. Тогда, если в строке нет ни одного знака арифметической операции (вне скобок) и нет скобок по краям, все выражение представляет собой единый элемент (он называется **операндом**) – число или имя переменной. Для записи этого элемента в область данных узла используется функция **strncpy** , которая копирует заданное количество символов. Она не ставит символ конца строки, поэтому приходится делать это вручную.

```

PNode MakeTree (char Expr[], int first, int last)
{
    int MinPrt, i, k, prt;
    PNode Tree = new Node;

    MinPrt = 100;
    for ( i = first; i <= last; i ++ ) {
        prt = Priority ( Expr[i] );
        if ( prt <= MinPrt ) {
            MinPrt = prt;
            k = i;
        }
    }
    if ( MinPrt == 100 )
        if ( Expr[first]=='(' && Expr[last]==')' ) {
            delete Tree;
            return MakeTree(Expr, first+1, last-1);
        }
    else {           // число или переменная
        k = last - first + 1;
        strncpy(Tree->data, Expr+first, k);
        Tree->data[k] = '\0';
        Tree->left = NULL;
        Tree->right = NULL;
        return Tree;
    }

    Tree->data[0] = Expr[k];      //знак операции
    Tree->data[1] = '\0';
    Tree->left = MakeTree (Expr,first,k-1);
    Tree->right = MakeTree (Expr,k+1,last);

    return Tree;
}

```

Если обнаружено число или переменная, сначала вычисляем ее длину и записываем в переменную **k**.

Для вычисления такого выражения по дереву надо несколько изменить функцию **CalcTree** с учетом того, что поле данных узла – символьная строка. Для преобразования числа из символьного вида в числовой используем стандартную функцию **atoi** (для этого надо подключить заголовочный файл **stdlib.h**).

```

int CalcTree (PNode Tree)
{
    int num1, num2;
    if ( ! Tree->left )           // если нет потомков,
        return atoi(Tree->data); // раскодировали число
    // ... дальше все то же самое...
}

```

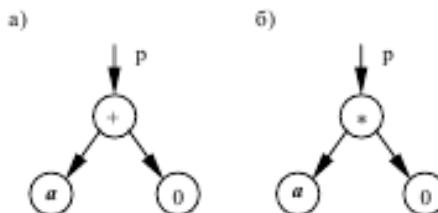
Конечно, для того, чтобы выражение можно было вычислить, оно не должно содержать имен переменных.

### Упрощение выражения с помощью дерева

Некоторые выражения можно сразу значительно упростить, используя очевидные тождества, верные для любого *x*:

$0 + x = x$	$x + 0 = x$	$0 * x = 0$
$0 - x = -x$	$x - 0 = x$	$1 * x = x$

Пусть, например, мы нашли такую структуру, как показано на рисунке *a*. Значение всего первого выражения равно **a**, поэтому нам надо сделать следующее: указатель **p** поставить на вершину **a**, а две ненужные вершины удалить из памяти.



В случае б) аналогично надо указатель **p** переставить на вершину со значением 0. при этом надо учесть, что второй узел (со значением **a**) может иметь потомков, которых также надо корректно удалить. Это делается рекурсивно:

```
void DeleteNode ( PNode Tree )
{
    if ( Tree == NULL ) return;
    DeleteNode ( Tree->left );
    DeleteNode ( Tree->right );
    delete Tree;
}
```

Кроме того, если оба сына какой-то вершины являются листьями и содержат числа, такое выражение можно сразу посчитать, также удалив два ненужных узла. Один из вариантов реализации этой операции приведен ниже. Здесь используется функция **IsNumber**, которая возвращает 1, если узел является листом и содержит число, и 0 в противном случае:

```
int IsNumber ( PNode Tree )
{
    int i = 0;
    if ( ! Tree ) return 0; // пустое дерево

    while ( Tree->data[i] ) // пока не дошли до конца строки
        if ( ! strchr("0123456789", Tree->data[i++]) )
            return 0;           // если не нашли цифру, выход

    return 1;
}
```

Сама процедура вычисления выражения выглядит так:

```

void Calculate(PNode Tree)
{
    int num1, num2, result = 0;

    if ( ! Tree || // если нельзя вычислить, выход
        ! IsNumber(Tree->left) ||
        ! IsNumber(Tree->right) ) return;

    num1 = atoi(Tree->left->data); // получить данные от сыновей
    num2 = atoi(Tree->right->data);

    switch ( Tree->data[0] ) { // выполнить операцию
        case '+': result = num1 + num2; break;
        case '-': result = num1 - num2; break;
        case '*': result = num1 * num2; break;
        case '/': result = num1 / num2; break;
    }

    delete Tree->left; // удалить ненужные поддеревья
    delete Tree->right;
    sprintf(Tree->data, "%d", result);
    Tree->left = NULL;
    Tree->right = NULL;
}

```

## Дерево игр

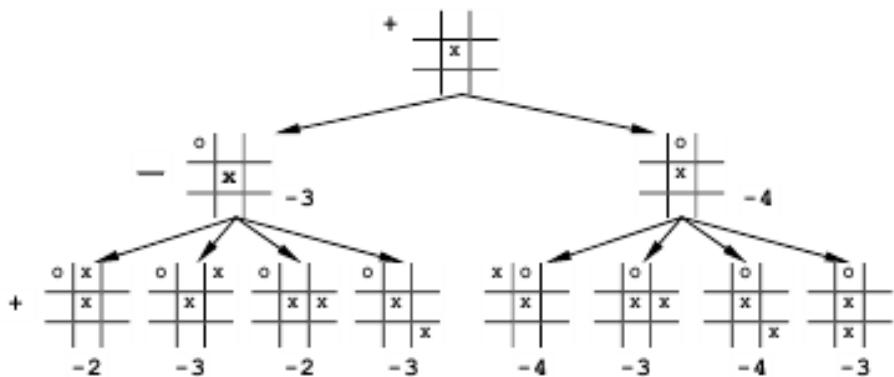
Одно из применений деревьев - игры с компьютером. Рассмотрим самый простой пример

– игру в крестики-нолики на поле 3 на 3.

Программа должны анализировать позицию и находить лучший ход. Для этого нужно определить **оценочную функцию**, которая получая позицию на доске и указание, чем играет игрок (крестики или нолики) возвращает число – оценку позиции. Чем она выше, тем более выгодна эта позиция для игрока . Примером такой функции может служить сумма строк, столбцов и диагоналей, которые может занять игрок минус такая же сумма для его противника.

Однако, в этой ситуации программа не ведет просчет вперед и не оценивает позиции, которые могут возникнуть из текущей. Это недостаточно для предсказания исхода игры. Хотя для крестиков-ноликов можно перебрать все варианты и найти выигрышную позицию, большинство игр слишком сложно, чтобы допускать полный перебор.

Выбор хода может быть существенно улучшен, если просматривать на несколько ходов вперед. **Уровнем просмотра** называется число будущих рассматриваемых ходов. Начиная с любой позиции можно построить дерево возможных позиций, получающихся после каждого хода игры. Для крестиков-ноликов ниже приведено дерево с уровнем просмотра 3 для того случая, когда крестики сделали первый ход в центр доски.



Обозначим игрока, который ходит в корневой позиции (в данном случае – нолики) знаком «плюс», а его соперника – знаком «минус». Попытаемся найти лучший ход для игрока «плюс» в этой позиции. Пусть все варианты следующих ходов были оценены для игрока «плюс». Он должен выбрать такой, в котором оценка максимальная для него.

С другой стороны, как только игрок «плюс» сделает свой ход, игрок «минус» из всех возможных ходов сделает такой, чтобы его оценка с позиции игрока «плюс» была минимальной. Поэтому значение минусового узла для игрока «плюс» равно минимальному из значений сыновей этого узла. Это означает, что на каждом шаге соперники делают наилучшие возможные ходы.

Для того, чтобы выбрать оптимальный ход в корне дерева, надо оценить позицию в его листьях. После этого каждому плюсовому узлу присваивается **максимальное** из значений его сыновей, а каждому минусовому – **минимальное**. Такой метод называется методом **минимакса**, так как по мере продвижения вверх используются попеременно функции максимума и минимума. Общая идея метода состоит в том, чтобы выбрать лучший ход на случай худших (для нас) действий противника. Таким образом, лучшим для ноликов в корневой позиции будет ход в угол.

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_  
\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Дерево поиска – это?
2. В чем отличаются двоичное дерево и дерево поиска?
3. Область применения деревьев поиска?
4. Как связаны между собой дерево и арифметическое выражение?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
3. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
4. Окулов С.М. Программирование в алгоритмах – М.: БИНОМ. Лаборатория знаний, 2006. - 383 с.
5. Прата С. Язык программирования С. - СПб.: Вильямс, 2013. – 960 с.
6. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400
7. Хьюз Дж. Структурный подход к программированию / Мичтом Дж. – М.: Мир, 1980. –278 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ основанных на  
поиске в ширину и глубину в графе»

Минск  
2017

## **Лабораторная работа № 17**

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ основанных на поиске в ширину и глубину в графе»

### **1. Цель работы**

Закрепить навык поиска в графе в ширину и глубину.

### **2. Задание**

Номер варианта соответствует вашему номеру по списку.

1. Найти самый длинный простой путь в графе (путь, все ребра которого попарно различны).
2. Найти медиану взвешенного графа, т.е. такую вершину, сумма расстояний от которой до всех других вершин минимальна.
3. Задана система односторонних дорог. Найти путь, соединяющий города А и В и не проходящий через заданное множество городов.
4. Определить, изоморфен ли заданный граф своему дополнению.
5. Мостом графа назовем такое ребро, удаление которого увеличивает число компонент связности графа. Найти все мосты для заданного графа.
6. Найти длину самого длинного простого пути от города А до города В в заданной системе односторонних дорог.
7. В заданном графе найти максимальный по количеству вершин полный подграф.
8. Задан ориентированный граф с  $N$  ( $1 < N < 10$ ) вершинами, пронумерованными целыми числами от 1 до  $N$ . Напишите программу, которая подсчитывает количество различных путей между всеми парами вершин графа.
9. Необходимо добраться из города А в город В при условии, что между ними нет прямого авиационного сообщения, затратив при этом минимальные средства. Заданы возможные промежуточные аэропорты. Для каждой пары аэропортов известно, существует ли между ними прямой маршрут, и если да, то известна минимальная стоимость перелета по этому маршруту.
10. Даны два числа:  $N$  и  $M$ . Построить граф из  $N$  вершин и  $M$  ребер. Каждой вершине ставится в соответствие число ребер, входящих в нее. Граф должен быть таким, чтобы сумма квадратов этих чисел была минимальна.
11. По заданной системе односторонних дорог определить, есть ли в ней город, куда можно попасть из любого другого города, проезжая не более 100 км.
12. В графе найти максимальное (по количеству ребер) подмножество попарно несмежных ребер.
13. Определить, является ли заданный граф двудольным.

14. По системе двусторонних дорог определить, можно ли, построив какие-нибудь три новые дороги, из заданного города добраться до каждого из остальных городов, проезжая не более 100 км.
15. Некоторые школы связаны компьютерной сетью. Между школами заключены соглашения: каждая школа имеет список школ-получателей, которым она рассыпает программное обеспечение всякий раз, получив новое бесплатное программное обеспечение (извне сети или из другой школы). При этом, если школа В есть в списке получателей школы А, то школа А может не быть в списке получателей школы В. Требуется написать программу, определяющую минимальное количество школ, которым надо передать по экземпляру нового программного обеспечения, чтобы распространить его по всем школам сети в соответствии с соглашениями.
16. Известно, что заданный граф - не дерево. Проверить, можно ли из него получить дерево путем удаления п вершин (каждая вершина удаляется вместе с инцидентными ей ребрами, п вводится с клавиатуры).
17. Задан неориентированный граф. При прохождении по некоторым ребрам некоторые (определенные заранее) ребра могут исчезать или появляться. Найти кратчайший путь из вершины с номером q в вершину с номером w.
18. Дан ориентированный граф с N вершинами ( $N < 50$ ). Вершины и дуги окрашены в цвета с номерами от 1 до M ( $M < 6$ ). Указаны две вершины, в которых находятся фишечки игрока, и конечная вершина. Правила перемещения фишечек: игрок может передвигать фишку по дуге, если ее цвет совпадает с цветом вершины, в которой находится другая фишка; ходы можно делать только в направлении дуг графа; поочередность ходов необязательна. Игра заканчивается, если одна из фишечек достигает конечной вершины. Написать программу поиска кратчайшего пути до конечной вершины, если он существует.
19. Заданы два числа: N и M ( $20 < M < N < 150$ ), где N - количество точек на плоскости. Требуется построить дерево из M точек так, чтобы оно было оптимальным. Дерево называется оптимальным, если сумма всех его ребер минимальна. Все ребра - это расстояния между вершинами, заданными координатами точек на плоскости.
20. Треугольником в неориентированном графе называется тройка вершин, попарно соединенных дугами. Склейванием треугольника называется операция замены вершин треугольника новой вершиной с сохранением всех связей с остальными вершинами графа. Дан неориентированный граф. Склейте все треугольники графа.
21. Проверить, является ли заданный ориентированный граф связным.

### 3. Оснащение работы

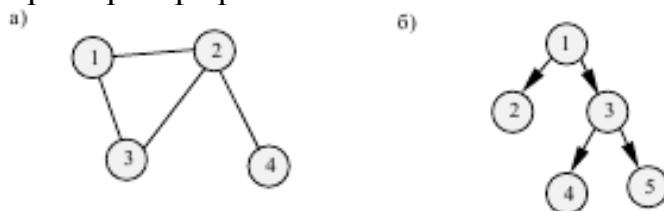
Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

#### 4. Основные теоретические сведения

Во многих жизненных ситуациях старая привычка толкает нас рисовать на бумаге точки, обозначающие людей, города, химические вещества, и показывать линиями (возможно со стрелками) связи между ними. Описанная картинка называется **графом**.

**Граф** - это совокупность **узлов** (**вершин**) и соединяющих их **ребер** (**дуг**).

Ниже показаны примеры графов



Если дуги имеют направление (вспомните улицы с односторонним движением), то такой граф называется **направленным** или **ориентированным** графом (**орграфом**).

**Цепью** называется последовательность ребер, соединяющих две (возможно не соседние) вершины **u** и **v**. В направленном графе такая последовательность ребер называется «**путь**».

Граф называется **связным**, если существует цепь между любой парой вершин. Если граф не связный, то его можно разбить на **k** связных компонент – он называется **k-связным**.

В практических задачах часто рассматриваются **взвешенные графы**, в которых каждому ребру приписывается **вес** (или длина). Такой граф называют **сетью**.

**Циклом** называется цепь из какой-нибудь вершины **v** в нее саму.

**Деревом** называется граф без циклов.

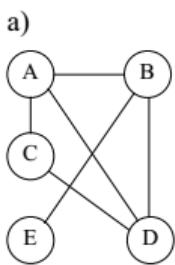
**Полным** называется граф, в котором проведены все возможные ребра (для графа, имеющего **n** вершин таких ребер будет  $n(n-1)/2$ ).

#### Описание графов

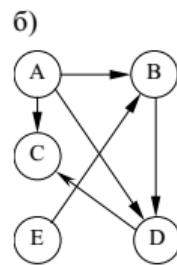
Для описания графов часто используют два типа матриц – **матрицу смежности** (для не-взвешенных графов) и **весовую матрицу** (для взвешенных).

**Матрица смежности** графа с **N** вершинами – это матрица размером **N** на **N**, где каждый элемент с индексами **(i, j)** является логическим значением и показывает, есть ли дуга из вершины **i** в вершину **j**.

Часто вместо логических значений (истина/ложь) используют целые числа (1/0). Для неориентированных графов матрица смежности всегда симметрична относительно главной диагонали (рисунок а). Для ориентированных графов (рисунок б) это не всегда так, потому что может существовать путь из вершины **i** в вершину **j** и не существовать обратного пути.



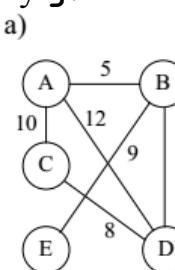
	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	0	0
E	0	1	0	0	0



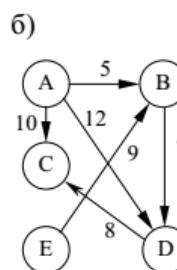
	A	B	C	D	E
A	0	1	1	1	0
B	0	0	0	1	0
C	0	0	0	0	0
D	0	0	1	0	0
E	0	1	0	0	0

Для взвешенных графов недостаточно просто указать, есть ли связь между вершинами. Требуется еще хранить в памяти «весы» каждого ребра, например, стоимость проезда или длину пути. Для этого используется **весовая матрица**.

**Весовая матрица** графа с **N** вершинами – это матрица размером **N** на **N**, где каждый элемент с индексами  $(i, j)$  равен «весу» ребра из вершины  $i$  в вершину  $j$ .



	A	B	C	D	E
A	0	5	10	12	0
B	5	0	0	7	9
C	10	0	0	8	0
D	12	7	8	0	0
E	0	9	0	0	0



	A	B	C	D	E
A	0	5	10	12	0
B	0	0	0	7	0
C	0	0	0	0	0
D	0	0	8	0	0
E	0	9	0	0	0

### Задача Прима-Краскала

#### Формулировка задачи

**Задача.** Дано плоская страна и в ней **n** городов с известными координатами. Нужно соединить все города телефонной сетью так, чтобы длина телефонных линий была минимальная.

Город будем изображать узлом (точкой). Телефонные линии могут разветвляться только на телефонных станциях, а не в чистом поле. Поскольку требуется линия минимальной общей длины, в ней не будет циклов, потому что иначе можно было бы убрать одно звено цикла и станции по-прежнему были бы связаны. В терминах теории графов эта задача звучит так: дан граф с **n** вершинами; длины ребер заданы матрицей  $\{d_{ij}\}$ ,  $i, j = 1 \dots n$ . Найти набор ребер, соединяющий все вершины графа (он называется **остовным деревом**) и имеющий минимальную длину.

Эта задача – одна из тех немногих, для которых известно точное и несложное решение, причем алгоритм предельно прост.

#### Жадные алгоритмы

Представим себе зимовщика, которому предоставили некоторый запас продуктов на всю зиму. Конечно, он может сначала съесть все самое вкусное – шоколад, мясо и т.п., но за такой подход придется жестоко расплачиваться в конце зимовки, когда останется только соль и маргарин.

Подобным образом, если оптимальное решение строится по шагам, обычно нельзя выбирать на каждом этапе «самое вкусное» – за это придется расплачиваться на последних шагах. Такие алгоритмы называют **жадными**.

Удивительно, но для непростой задачи Прима-Краскала жадный алгоритм дает точное оптимальное решение. Алгоритм формулируется так: в цикле **n-1** раз выбрать из оставшихся ребер самое короткое ребро, которое не образует цикла с уже выбранными.

Как же проследить, чтобы не было циклов? Оказывается очень просто: в самом начале покрасим все вершины в разные цвета и затем, выбрав очередное ребро между вершинами **i** и **j**, где **i** и **j** имеют разные цвета, перекрасим вершину **j** и все соединенные с ней (то есть имеющие ее цвет) в цвет **i**. Таким образом, при выборе ребер, соединяющих вершины разного цвета, цикл не возникнет никогда, а после **n-1** шагов все вершины будут иметь один цвет.

Для записи информации о ребрах введем структуру

```
struct rebro { int i, j; } // ребро соединяет вершины i и j
```

причем будем считать, что в паре номер первой вершины **i** меньше, чем номер второй **j**. Приведенная ниже программа действует по следующему «жадному» алгоритму:

- ✓ Покрасить все вершины в разные цвета.
- ✓ Сделать **n-1** раз в цикле
  - выбрать ребро (**i, j**) минимальной длины, соединяющее вершины разного цвета;
  - запомнить его в массиве ребер;
  - перекрасить все вершины, имеющие цвет **j**, в цвет **i**.
- ✓ Вывести результат.

```

const int N = 5;

void main()
{
    int D[N][N], Col[N], i, j, k, Dmin, jMin, iMin, col_j;
    rebro Reb[N-1];
    // здесь надо ввести матрицу D

    for ( i = 0; i < N; i ++ ) // покрасить все вершины
        Col[i] = i;           // в разные цвета

    for ( k = 0; k < N-1; k ++ ) {
        Dmin = 30000;
        for ( i = 0; i < N-1; i ++ )
            for ( j = i+1; j < N; j ++ )
                if ( Col[i] != Col[j] && // ищем самое короткое ребро,
                    D[i][j] < Dmin ) { // не образующее цикла
                    Dmin = D[i][j];
                    iMin = i;
                    jMin = j;
                }
        Reb[k].i = iMin; // запомнить найденное ребро
        Reb[k].j = jMin;
        col_j = Col[jMin];
        for ( i = 0; i < N; i ++ ) // перекрасить все вершины, цвет
            if ( Col[i] == col_j ) // которых совпал с цветом
                Col[i] = Col[iMin]; // вершины jMin
    }
    // здесь надо вывести найденные ребра
}

```

Дополнительно можно рассчитывать общую длину выбранных ребер, но это не меняет принципиально алгоритм. Этот алгоритм требует памяти порядка  $n^2$  (обозначается  $O(n^2)$ ), то есть при увеличении  $n$  в 2 раза объем требуемой памяти увеличивается в 4 раза). Его времененная сложность –  $O(n^3)$ , то есть при увеличении  $n$  в 2 раза время вычислений увеличивается в 8 раз (надо просмотреть  $O(n^3)$  чисел и сделать это  $n-1$  раз).

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_

\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Дайте определение графу.
2. Как можно представить граф в памяти компьютера?
3. Ребро (дуга) графа – это?
4. Вершина графа – это?
5. Вес ребра – это?
6. Какие существуют обходы графа?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
3. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
4. Окулов С.М. Программирование в алгоритмах – М.: БИНОМ. Лаборатория знаний, 2006. - 383 с.
5. Прата С. Язык программирования С. - СПб.: Вильямс, 2013. – 960 с.
6. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400
7. Хьюз Дж. Структурный подход к программированию / Мичтом Дж. – М.: Мир, 1980. –278 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ поиска кратчайших  
расстояний»

Минск  
2017

# Лабораторная работа № 18

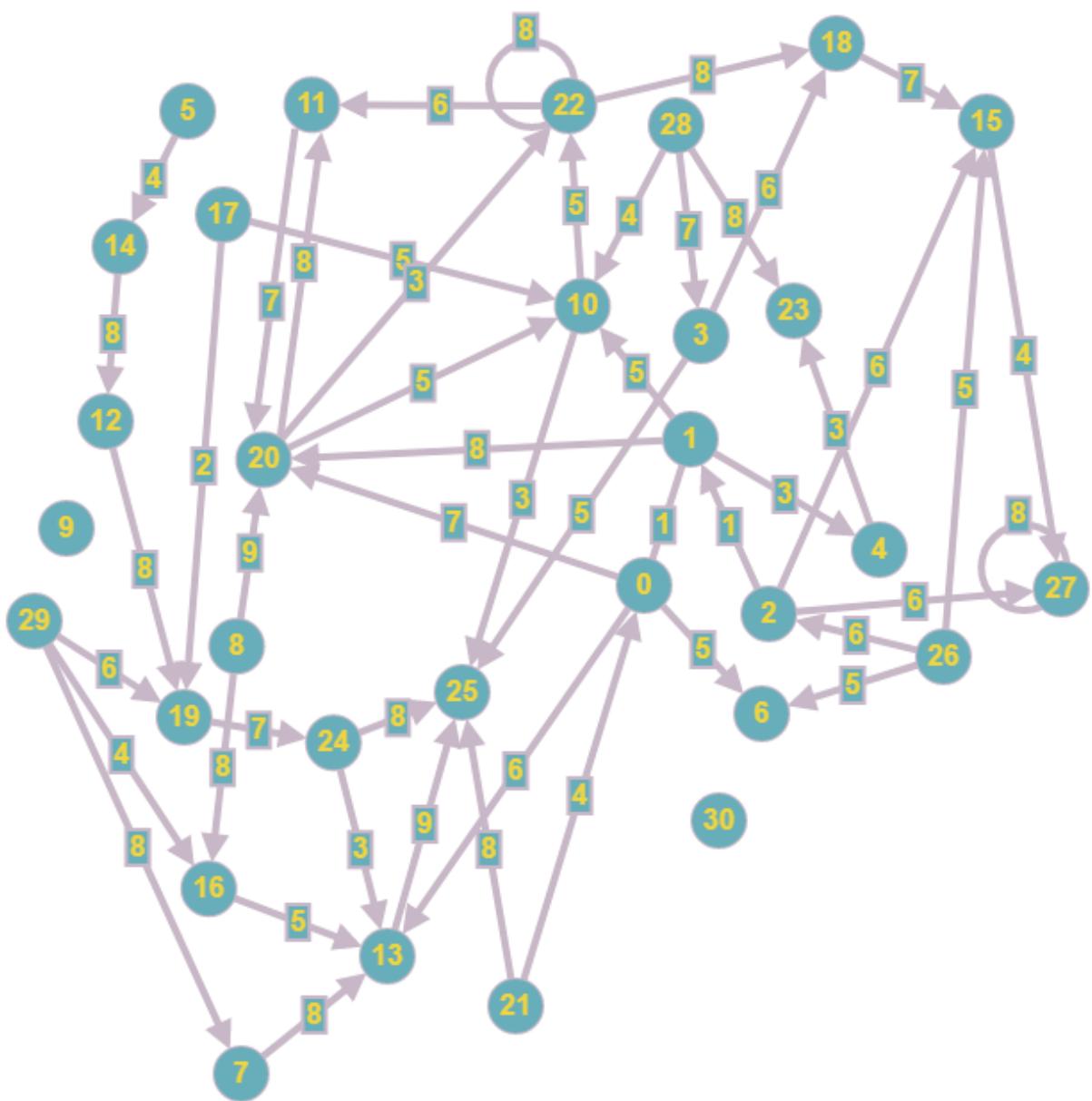
**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ поиска кратчайших расстояний»

## 1. Цель работы

Отработка навыка поиска кратчайших путей.

## 2. Задание

В предложенном графе найти кратчайшие пути от начальной вершины ко всем остальным, одним из известных алгоритмов. Номер начальной вершины соответствует вашему номеру по списку.



### **3. Оснащение работы**

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### **4. Основные теоретические сведения**

**Задача.** Задана сеть дорог между населенными пунктами (часть из них могут иметь одностороннее движение). Требуется найти кратчайший путь между двумя заданными пунктами.

Обычно задачу несколько расширяют и находят сразу кратчайшие пути от заданной вершины ко всем остальным. В терминах теории графов задача выглядит так: в сети, где часть дорог имеет одностороннее движение, найти кратчайшие пути от заданной вершины ко всем остальным.

#### **Алгоритм Дейкстры**

Ниже описывается алгоритм, предложенный Дейкстрой в 1959 г. Дано матрица  $\{d_{ij}\}$  длин дуг между вершинами, если дуги между вершинами  $i$  и  $j$  нет, то  $d_{ij} = \infty$ . Если сеть образуют  $n$  вершин, то для реализации алгоритма требуется три массива длиной  $n$ :

- ✓ Массив  $\{a_i\}$ , в котором  $a_i=0$ , если вершина  $i$  еще не рассмотрена, и  $a_i=1$ , если вершина  $i$  уже рассмотрена.
- ✓ Массив  $\{b_i\}$ , в котором  $b_i$  – текущее кратчайшее расстояние от выбранной стартовой вершины  $x$  до вершины  $i$ .
- ✓ Массив  $\{c_i\}$ , в котором  $c_i$  – номер предпоследней вершины в текущем кратчайшем пути из выбранной стартовой вершины  $x$  до вершины  $i$ .

Сам алгоритм состоит из трех этапов: инициализации, основного цикла и вывода результата.

#### **Инициализация**

Пусть  $x$  – номер выбранной стартовой вершины.

- ✓ Заполним весь массив  $a$  значением 0 (пока ни одна вершина не рассмотрена).
- ✓ В  $i$ -ый элемент массива  $b$  запишем расстояние от вершины  $x$  до вершины  $i$  (если пути нет, то оно равно  $\infty$ , в программе укажем очень большое число).
- ✓ Заполним весь массив  $c$  значением  $x$  (пока рассматриваем только прямые пути от  $x$  к  $i$ ).
- ✓ Рассмотрим стартовую вершину: присвоим  $a[x]=1$  и  $c[x]=0$  (начало всех путей).

#### **Основной цикл**

Среди нерассмотренных вершин (для которых  $a[i]=0$ ) найти такую вершину  $j$ , что расстояние  $b_j$  – минимальное. Рассмотреть ее:

- ✓ установить  $a[j]=1$  (вершина рассмотрена);

✓ сделать для всех вершин  $k$ :

- если путь от вершины  $x$  к вершине  $k$  через  $j$  короче, чем уже найденный кратчайший путь (то есть  $b_j + d_{jk} < b_k$ ), то запомнить его:  $c[k] = j$  и  $b_k = b_j + d_{jk}$ .

### **Выход результата**

Можно доказать, что в конце такой процедуры массив  $b$ , будет содержать кратчайшие расстояния от стартовой вершины  $x$  до вершины  $i$  (для всех  $i$ ). Однако хотелось бы еще получить сам оптимальный путь.

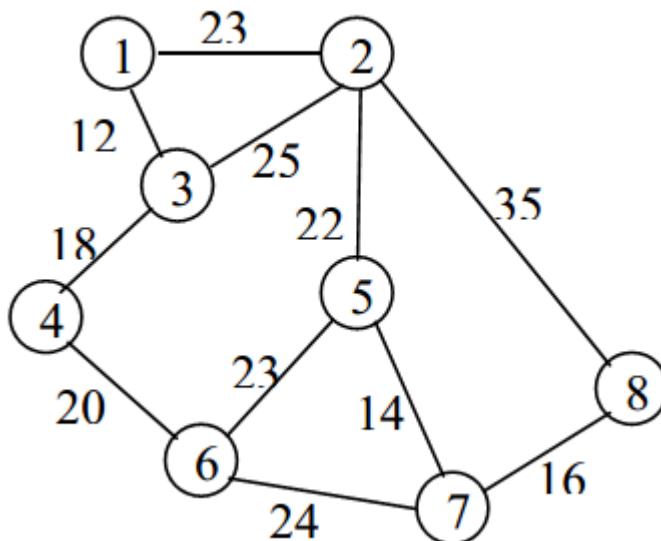
Оказывается, массив  $c$  содержит всю необходимую информацию для решения этой задачи. Предположим, что надо вывести оптимальный путь из вершины  $x$  в вершину  $i$ . По построению предпоследняя вершина в этой цепи имеет номер  $z=c[i]$ . Теперь надо найти оптимальный путь в вершину  $z$  тем же способом. Таким образом, путь «раскручивается» с конца. Когда мы получили  $z=0$ , путь закончен, мы вернулись в стартовую вершину. В виде алгоритма это можно записать так:

- ✓ установить  $z=i$ ;
- ✓ пока  $c[z]$  не равно нулю,
  - $z=c[z]$ ;
  - вывести  $z$ .

Для выполнения алгоритма надо  $n$  раз просмотреть массив  $b$  из  $n$  элементов, поэтому он имеет квадратичную сложность  $O(n^2)$ .

### **Пример**

Пусть дана сеть дорог, показанная на рисунке справа. Найдем кратчайшие расстояния от вершины 3 до всех остальных вершин. Покажем ход выполнения алгоритма Дейкстры по шагам.



**Инициализация.** В результате этого шага получаем такие матрицы:

	1	2	3	4	5	6	7	8
<b>a</b>	0	0	1	0	0	0	0	0
<b>b</b>	12	25	0	18	$\infty$	$\infty$	$\infty$	$\infty$
<b>c</b>	3	3	0	3	3	3	3	3

**Основной цикл.** Последовательно массивы преобразуются к виду :

1)  $\min b_j = 12$  для  $j = 1$

	1	2	3	4	5	6	7	8
<b>a</b>	1	0	1	0	0	0	0	0
<b>b</b>	12	25	0	18	$\infty$	$\infty$	$\infty$	$\infty$
<b>c</b>	3	3	0	3	3	3	3	3

2)  $\min b_j = 18$  для  $j = 4$

	1	2	3	4	5	6	7	8
<b>a</b>	1	0	1	1	0	0	0	0
<b>b</b>	12	25	0	18	$\infty$	38	$\infty$	$\infty$
<b>c</b>	3	3	0	3	3	4	3	3

3)  $\min b_j = 25$  для  $j = 2$

	1	2	3	4	5	6	7	8
<b>a</b>	1	1	1	1	0	0	0	0
<b>b</b>	12	25	0	18	47	38	$\infty$	60
<b>c</b>	3	3	0	3	2	4	3	2

4)  $\min b_j = 38$  для  $j = 6$

	1	2	3	4	5	6	7	8
<b>a</b>	1	1	1	1	1	0	1	0
<b>b</b>	12	25	0	18	47	38	62	60
<b>c</b>	3	3	0	3	2	4	6	2

5)  $\min b_j = 47$  для  $j = 5$

	1	2	3	4	5	6	7	8
<b>a</b>	1	1	1	1	1	1	0	0
<b>b</b>	12	25	0	18	47	38	61	60
<b>c</b>	3	3	0	3	2	4	5	2

6)  $\min b_j = 60$  для  $j = 8$

	1	2	3	4	5	6	7	8
<b>a</b>	1	1	1	1	1	1	0	1
<b>b</b>	12	25	0	18	47	38	61	60
<b>c</b>	3	3	0	3	2	4	5	2

**Вывод кратчайшего пути.** Найдем, например, кратчайший путь из вершины 3 в вершину 8. «Раскручивая» массив **c**, получаем  $8 \leftarrow 2 \leftarrow 3$ .

### Алгоритм Флойда-Уоршелла

Если надо только вычислить кратчайшие расстояния между всеми вершинами, но не требуется знать сами кратчайшие маршруты, можно использовать весьма элегантный и простой алгоритм Флойда-Уоршелла:

```

for (k = 0; k < n; k++)
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            if ( d[i][j] > d[i][k]+d[k][j] ) {
                d[i][j] = d[i][k]+d[k][j];
                p[i][j] = p[k][j];
            }
    }
}

```

Сначала в матрице  $\{d_{ij}\}$  записаны расстояние между вершинами напрямую. Затем рассмотрим все пути, которые проходят через первую вершину. Если путь через нее короче, чем напрямую, то заменяем значение в матрице на новый кратчайший путь. В конце элемент матрицы  $\{d_{ij}\}$  содержит длину кратчайшего пути из  $i$  в  $j$ . Каждая строка матрицы  $\{p_{ij}\}$  сначала заполняется так, как массив **c** в алгоритме Дейкстры, а в конце элемент  $\{p_{ij}\}$  равен номеру предпоследней вершины для кратчайшего пути из вершины  $i$  в вершину  $j$ .

## **5. Порядок выполнения работы**

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_  
\_\_\_\_\_

## **7. Контрольные вопросы и задания**

7. Что такое вес ребра?
8. Для чего нужно находить кратчайшие пути в графе?
9. Принцип работы алгоритма Дейкстры.
10. Где применяется алгоритмы нахождения кратчайших путей?
11. Сформулировать задачу о кратчайшем пути. Область применения.

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
3. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
4. Окулов С.М. Программирование в алгоритмах – М.: БИНОМ. Лаборатория знаний, 2006. - 383 с.
5. Прата С. Язык программирования С. - СПб.: Вильямс, 2013. – 960 с.
6. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400
7. Хьюз Дж. Структурный подход к программированию / Мичтом Дж. – М.: Мир, 1980. –278 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ нахождения  
максимальных потоков в графе»

Минск  
2017

## **Лабораторная работа № 19**

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ нахождения максимальных потоков в графе»

### **1. Цель работы**

Закрепить навык поиска максимальных потоков в графах.

### **2. Задание**

Пропускные способности дуг заданы матрицей. Построить максимальный поток от **s** к **t** и указать минимальный разрез, отделяющий **s** от **t**.

Номер матрицы соответствует вашему номеру по списку.

1)

0	6	42	14	0	16	0	36	0	0	5	0	0
0	0	0	3	27	12	0	45	0	23	2	0	0
0	21	0	21	25	0	0	0	29	0	13	9	0
0	32	31	0	0	0	30	0	31	0	27	0	0
0	0	0	0	0	21	0	2	22	44	44	41	33
35	0	0	2	8	0	0	29	5	0	42	0	0
0	11	0	48	0	0	0	0	42	0	0	0	45
0	0	29	0	0	48	1	0	0	41	0	0	34
24	16	7	0	0	0	0	21	0	8	0	23	47
0	31	44	0	0	41	0	0	24	0	0	0	0
12	25	0	0	7	0	0	39	0	0	0	0	0
0	0	25	30	0	17	0	0	0	47	4	0	0
38	0	0	35	6	0	3	0	0	36	14	0	0

2)

0	31	0	35	38	2	0	31	26	4	20	0	7
0	0	27	0	14	0	0	29	0	19	6	38	14
0	0	0	0	0	0	0	15	15	0	0	0	33
0	0	2	0	0	32	0	45	0	0	47	47	0
0	49	24	0	0	42	27	0	24	0	0	0	0
19	0	0	0	0	45	0	0	0	0	45	0	0
45	0	0	11	47	26	0	29	0	27	0	28	45
0	9	0	4	0	0	2	0	0	32	0	39	0
27	29	0	10	0	0	38	2	0	8	0	0	17
8	35	24	0	0	49	18	17	0	0	0	0	24
23	21	44	0	0	38	16	30	0	0	0	8	0

0	0	0	30	28	2	0	0	44	29	42	0	14
36	16	42	17	12	0	12	0	14	0	0	0	0

3)

0	0	47	0	0	26	0	40	24	0	1	0	0
0	0	33	0	0	0	35	0	0	16	21	48	6
3	0	0	10	47	0	0	31	46	0	20	40	15
35	42	0	0	38	0	34	0	0	0	26	0	0
0	48	5	0	0	0	0	44	0	48	0	0	6
0	0	0	0	0	0	30	0	8	22	0	0	0
0	6	0	21	0	0	0	8	48	27	0	0	0
0	5	19	6	0	3	47	0	0	0	0	41	14
44	1	40	0	0	0	0	21	0	10	7	0	19
13	0	23	20	23	0	5	36	0	0	0	0	42
28	0	0	22	47	43	13	0	11	44	0	0	0
0	4	32	0	38	44	0	45	0	27	32	0	0
17	0	14	49	0	38	0	0	0	37	0	0	0

4)

0	14	29	0	0	5	0	0	14	0	0	28	36
0	0	0	31	4	24	0	24	0	15	16	0	18
0	0	0	0	18	29	0	12	0	0	25	0	0
0	0	0	0	3	0	34	26	0	0	23	38	17
37	0	35	0	0	30	1	0	0	41	0	0	0
0	12	0	35	0	0	0	12	0	19	0	47	18
21	0	0	27	0	0	0	0	0	20	0	47	0
0	23	3	40	0	0	0	0	42	34	38	32	0
0	31	22	0	0	0	25	19	0	36	10	0	0
11	12	10	0	0	0	10	0	1	0	12	0	0
35	0	31	0	20	40	13	6	0	38	0	0	0
0	0	0	0	1	0	15	0	29	0	0	0	48
47	0	29	0	0	0	0	19	0	0	17	38	0

5)

0	0	9	0	0	0	32	23	0	0	0	0	0
10	0	6	0	7	0	15	22	0	24	14	35	19
16	48	0	0	30	0	0	0	5	25	48	0	32
2	0	0	0	0	8	0	0	0	0	35	10	0

0	40	0	0	0	0	23	0	5	19	0	0	0
0	0	16	40	0	0	29	0	0	0	8	19	0
40	22	18	25	48	29	0	0	0	0	39	0	0
0	0	0	48	0	0	0	0	43	0	22	0	0
0	47	4	0	9	13	12	1	0	0	43	0	0
0	0	7	0	0	29	0	42	0	0	47	0	3
0	45	10	9	19	47	32	0	0	0	0	47	0
14	0	36	0	0	32	30	0	26	0	0	0	49
0	0	0	10	0	48	0	0	28	0	2	49	0

6)

0	0	22	19	3	28	0	0	40	0	0	8	0
39	0	38	0	0	38	14	0	41	10	30	40	0
0	0	0	0	0	0	0	19	35	0	30	16	29
0	0	0	0	16	0	0	45	34	14	0	0	0
0	15	22	0	0	7	25	0	0	0	29	0	44
0	0	28	0	0	0	0	41	0	18	30	0	0
0	0	41	26	0	0	0	13	0	18	0	0	37
0	0	39	0	0	28	38	0	0	21	13	33	29
0	16	0	28	0	0	0	0	0	0	28	16	37
0	0	37	0	0	18	0	29	39	0	0	13	8
42	0	35	21	0	27	0	0	1	0	0	0	33
42	0	49	20	0	0	19	0	0	35	40	0	0
13	0	27	0	0	10	25	23	0	0	15	0	0

7)

0	0	0	3	0	0	0	20	0	3	44	31	15
40	0	0	0	0	0	0	0	0	38	18	0	41
0	0	0	0	38	0	25	16	0	0	20	30	14
41	0	0	0	22	18	0	0	3	0	7	37	37
0	40	26	44	0	37	8	10	0	0	48	0	0
0	0	36	47	28	0	13	15	11	0	18	27	0
0	32	0	43	0	8	0	0	0	35	0	0	45
0	0	48	16	0	23	24	0	0	0	33	0	9
42	0	5	22	0	0	0	0	0	0	0	48	18
0	21	28	0	31	0	0	5	0	0	0	27	0
0	0	31	18	39	0	0	15	0	4	0	0	47
0	0	31	30	0	0	17	0	27	48	16	0	18
0	31	4	0	0	36	0	0	0	2	39	0	0

8)

0	36	10	0	2	27	0	0	0	0	0	0	24
0	0	47	21	35	0	42	21	6	10	10	0	26
9	15	0	0	33	44	0	0	31	9	43	0	12
9	19	41	0	26	0	0	0	13	0	0	0	0
39	0	0	0	0	49	0	0	0	0	0	0	0
0	0	26	0	0	0	0	32	7	0	0	0	0
2	28	20	22	14	0	0	36	0	40	39	0	34
20	0	0	0	0	39	0	0	0	38	0	25	29
0	0	0	42	0	0	36	0	0	27	0	0	0
0	32	0	38	0	0	32	0	0	0	0	0	45
10	44	0	29	6	1	33	0	18	0	0	0	43
8	0	0	31	0	8	0	0	39	0	29	0	16
14	33	0	11	29	0	22	0	8	27	0	0	0

9)

0	0	0	13	14	0	0	23	21	15	45	47	0
0	0	0	35	8	0	48	42	39	40	10	0	0
0	27	0	18	13	0	2	0	0	0	0	38	39
32	20	46	0	32	0	0	10	16	15	32	0	29
0	0	0	3	0	0	0	37	0	0	36	28	47
26	0	0	0	48	0	38	45	0	0	0	18	0
0	0	0	29	0	0	0	0	0	5	0	22	14
30	21	0	28	37	0	0	0	0	7	9	13	49
41	0	39	0	0	17	0	0	0	0	0	0	0
6	40	10	19	0	41	0	0	9	0	1	40	8
0	46	21	0	20	0	0	32	0	38	0	0	20
0	0	0	0	0	0	0	23	25	0	0	0	0
0	0	27	0	0	19	0	0	0	0	5	38	0

10)

0	0	0	0	18	0	3	0	6	0	1	0	0
42	0	0	0	0	20	0	0	48	31	0	7	0
0	36	0	29	0	0	0	0	0	0	5	14	31
5	0	15	0	20	0	7	0	34	0	0	36	40
0	17	0	41	0	0	46	19	0	48	4	19	24
0	21	0	48	0	0	0	15	26	17	41	33	0
15	1	0	36	0	0	0	0	0	32	0	0	0

0	0	0	0	0	27	20	0	0	4	10	0	11
0	21	28	3	0	0	0	0	0	0	24	0	0
44	0	23	0	18	0	43	17	0	0	43	26	0
10	1	47	0	45	0	29	0	0	43	0	23	33
0	10	13	35	39	18	0	37	22	0	0	0	33
32	44	6	0	0	0	0	20	0	0	0	0	0

11)

0	0	7	8	45	11	0	23	43	17	0	0	3
33	0	26	0	0	17	0	0	45	0	0	0	0
0	0	0	0	0	42	0	8	0	0	17	0	32
12	38	0	0	0	20	0	0	0	10	0	29	0
24	40	0	0	0	0	0	15	0	0	0	0	0
0	10	47	33	1	0	48	0	0	0	27	0	0
6	0	35	48	0	0	0	0	3	0	0	12	1
46	0	0	0	8	0	0	0	13	14	48	0	0
0	0	2	47	0	0	0	0	0	0	48	0	0
0	0	0	0	45	48	48	48	45	0	9	44	15
17	48	16	29	0	0	34	32	0	0	0	45	0
19	0	30	0	44	21	2	37	34	0	0	0	0
0	45	34	23	37	0	19	23	0	42	23	38	0

12)

0	1	34	0	38	0	27	12	0	0	16	0	0
27	0	28	0	0	0	0	31	0	24	0	0	18
4	0	0	0	0	7	0	32	0	5	0	42	0
0	31	0	0	33	0	42	11	0	0	0	0	12
0	0	0	18	0	23	0	0	0	0	17	0	10
0	0	41	35	40	0	0	28	0	25	33	0	32
31	0	18	15	0	0	0	0	49	0	28	4	27
0	0	33	29	0	6	0	0	0	30	0	0	0
39	25	0	0	30	0	0	19	0	0	0	0	0
0	26	24	0	0	0	0	0	0	0	0	35	0
20	6	0	29	22	0	22	0	1	40	0	0	0
45	23	38	26	0	0	0	0	11	24	20	0	0
0	0	1	2	0	38	0	0	0	7	0	0	0

13)

0	0	0	0	0	41	18	0	29	0	38	0	47
0	0	32	32	0	0	0	0	46	0	23	0	47
0	36	0	0	0	0	6	13	0	0	42	38	28
0	0	0	0	0	10	0	0	31	31	0	0	0
0	48	28	9	0	33	0	23	36	0	0	0	0
0	12	0	0	21	0	0	0	0	44	0	0	0
24	0	34	6	0	2	0	24	14	0	0	43	0
0	0	16	29	0	43	0	0	12	34	0	41	0
0	0	6	26	49	14	42	47	0	12	32	33	0
42	35	0	0	0	46	10	22	0	0	0	13	41
0	16	35	0	40	0	0	0	12	36	0	0	0
0	0	9	19	20	0	48	0	0	0	0	0	0
11	49	0	0	0	0	0	44	0	30	31	0	0

14)

0	38	0	43	0	4	49	0	0	41	46	46	6
0	0	0	30	0	0	37	24	0	0	0	28	10
0	23	0	0	6	0	0	28	0	5	0	0	25
30	0	45	0	0	0	0	0	34	31	0	38	0
0	0	0	0	0	0	18	0	11	0	24	0	0
0	46	0	0	0	0	16	0	0	9	0	0	0
15	6	0	0	4	0	0	0	41	37	29	2	30
3	0	0	0	36	0	46	0	0	0	39	15	0
36	0	0	0	0	2	0	0	0	1	0	0	49
0	0	10	0	25	38	44	46	43	0	21	0	0
0	0	0	43	0	46	0	36	49	0	0	29	0
0	0	0	15	0	37	42	0	0	5	0	0	40
0	0	28	0	0	28	15	22	0	0	13	12	0
0	0	0	0	35	47	0	29	46	1	0	0	12
0	20	44	12	2	5	39	23	0	21	0	0	27

15)

0	0	0	4	0	36	22	29	0	0	49	0	0
26	0	15	0	0	48	0	9	11	43	0	0	6
0	4	0	0	0	25	39	0	14	0	0	40	0
0	20	19	0	0	0	8	1	0	0	0	0	48
0	37	46	0	0	16	0	6	0	0	28	0	31
26	0	0	0	0	0	0	22	8	0	0	44	46

0	12	0	0	0	0	0	6	0	0	0	0	0
0	0	2	48	0	0	0	0	7	34	41	0	34
17	0	0	0	0	0	0	49	0	0	10	0	0
39	0	0	0	0	0	0	0	0	0	0	0	48
0	35	16	0	21	0	22	0	7	34	0	0	33
0	0	2	22	0	4	0	19	0	0	0	0	0
0	43	5	47	0	0	0	14	32	48	0	0	0

16)

0	27	0	0	0	45	21	0	0	0	33	0	0
0	0	34	0	37	11	0	0	3	0	0	38	27
0	26	0	0	20	0	9	0	4	36	45	23	14
0	0	0	0	37	14	0	0	21	43	0	0	34
30	0	7	18	0	36	42	0	0	0	23	33	10
0	2	14	0	0	0	0	0	40	12	6	0	49
0	0	0	14	8	0	0	49	17	13	0	25	0
0	28	0	0	4	0	0	0	0	0	0	0	7
0	41	49	34	0	36	45	24	0	0	0	43	0
41	37	14	28	14	13	34	0	19	0	0	1	20
0	0	0	0	0	0	0	0	23	0	0	0	0
47	7	3	0	0	21	16	45	0	19	0	0	0
0	9	0	0	0	33	0	0	0	49	7	32	0

17)

0	0	0	0	0	10	0	0	0	10	41	15	0
3	0	0	0	0	0	45	0	9	49	0	15	37
0	0	0	18	0	23	25	17	48	37	48	0	0
42	0	0	0	35	0	0	29	0	0	4	16	0
12	0	0	46	0	19	13	0	43	0	14	47	0
34	11	34	49	0	0	0	9	0	16	0	19	8
17	0	0	0	0	11	0	0	23	45	19	43	0
17	0	0	24	29	0	0	0	23	0	1	0	0
0	0	0	49	3	14	0	4	0	0	0	0	0
19	0	42	33	0	9	0	0	23	0	0	48	0
0	0	5	15	0	0	8	0	0	1	0	0	0
22	0	2	45	49	0	0	35	29	0	0	0	16
0	5	0	9	21	2	1	0	49	0	0	0	0

18)

0	0	43	6	10	0	0	14	36	37	0	46	20
35	0	16	0	0	35	0	0	0	0	34	0	0
0	0	0	0	32	8	0	0	0	0	7	30	17
0	0	0	0	0	0	0	0	34	16	21	0	0
0	0	0	0	0	0	0	0	0	0	0	14	0
9	33	0	0	0	0	0	0	0	0	10	23	12
24	15	0	0	0	0	0	23	17	23	47	0	0
0	0	0	23	35	20	0	0	26	1	0	0	0
0	49	25	0	0	0	0	4	0	21	19	30	0
36	0	0	0	0	0	0	0	23	0	0	5	0
16	0	0	47	19	0	36	0	0	0	0	20	19
17	4	0	0	41	0	0	0	11	0	0	0	6
24	33	11	4	0	0	49	35	0	0	10	0	0

19)

0	10	0	1	0	0	0	0	2	0	0	0	34
25	0	5	37	0	42	0	18	48	0	4	0	0
0	0	0	0	0	0	14	47	0	13	0	0	0
17	0	0	0	29	7	0	34	0	18	48	0	28
0	0	41	0	0	0	15	0	0	0	5	27	0
20	35	30	8	42	0	33	9	0	16	13	0	17
0	0	0	0	22	25	0	0	0	27	3	24	8
0	3	29	6	0	3	0	0	8	0	0	0	0
33	0	21	0	29	13	0	18	0	0	0	0	0
32	0	26	14	0	0	0	36	0	0	8	0	0
0	5	37	0	31	11	30	0	0	2	0	41	0
0	0	0	13	0	0	36	49	3	20	0	0	0
0	8	3	0	0	0	42	1	0	0	7	0	0

20)

0	30	23	46	39	2	0	0	0	41	41	28	35
32	0	15	0	0	14	0	0	17	0	29	7	0
24	14	0	2	0	5	25	0	49	28	6	11	18
14	11	0	0	45	0	0	0	0	7	0	0	17
0	43	11	0	0	0	0	0	0	0	5	0	46
24	0	0	0	16	0	0	5	0	19	7	38	0
0	25	20	18	46	0	0	0	0	30	0	0	45
0	0	0	34	42	0	16	0	34	0	0	0	8

0	10	27	0	0	0	27	0	0	0	0	0	40
0	0	0	6	20	0	35	42	8	0	0	0	49
3	28	19	3	18	0	28	18	17	0	0	0	0
0	0	44	0	0	46	0	0	30	0	9	0	6
0	0	0	6	0	0	43	15	0	0	40	4	0
31	0	32	41	46	0	0	0	49	13	0	0	1
0	13	18	39	37	0	15	43	31	15	30	0	32

21)

0	27	48	5	29	46	0	30	0	8	0	0	43
2	0	46	14	0	8	0	0	0	3	23	37	0
7	0	0	12	46	12	0	0	0	0	46	0	0
31	0	32	0	0	0	34	37	0	26	0	0	22
0	31	0	0	0	21	0	26	0	0	0	0	41
0	0	0	13	0	0	22	0	48	0	37	0	21
0	30	0	24	33	0	0	36	0	0	0	35	0
0	0	6	0	0	0	9	0	34	0	41	0	35
34	0	21	0	0	7	12	0	0	2	5	4	0
9	0	0	0	32	0	0	0	0	0	0	0	26
7	15	28	0	26	0	0	0	0	14	0	0	0
7	37	0	32	0	0	36	0	0	42	0	0	0
0	31	0	41	37	4	37	0	2	0	22	9	0

22)

0	49	24	11	19	0	0	0	35	0	0	24	
0	0	0	22	0	15	9	39	0	4	0	0	12
0	0	0	0	0	0	3	35	0	0	33	33	28
0	24	0	0	21	28	0	6	0	37	0	0	6
21	9	0	18	0	0	0	0	0	0	3	36	0
0	0	0	0	0	0	0	16	48	0	0	22	16
0	0	9	22	16	6	0	0	31	15	0	0	34
44	0	0	0	3	41	0	0	0	0	0	0	6
0	0	32	0	0	33	2	44	0	0	0	0	0
16	19	0	0	4	12	0	0	0	0	0	0	0
0	43	0	22	0	0	8	2	0	49	0	24	0
0	19	0	0	0	0	0	0	0	10	0	0	0
28	49	39	13	14	0	0	0	0	5	19	0	0

23)

0	0	0	0	0	0	0	30	4	0	11	0	7
0	0	25	32	0	0	21	0	5	0	0	0	46
8	31	0	0	0	48	42	0	40	6	0	0	1
0	0	0	0	43	38	11	28	49	0	0	34	34
46	28	0	0	0	0	0	45	0	0	0	0	0
10	0	43	0	0	0	0	2	0	41	0	0	0
18	12	26	0	0	0	0	0	48	9	0	0	0
0	29	19	0	9	0	0	0	0	4	34	22	0
45	0	31	0	0	0	0	0	0	0	0	0	0
40	45	0	7	0	0	0	49	0	0	18	36	13
0	8	48	2	41	0	0	2	0	4	0	0	0
0	5	0	48	33	11	46	27	0	0	0	0	0
38	16	0	17	0	49	44	24	35	22	0	0	0

24)

0	7	0	0	0	0	23	0	35	0	0	0	34
0	0	0	12	0	0	11	46	36	0	0	0	0
7	0	0	0	28	35	11	49	20	27	30	48	0
0	34	13	0	0	0	0	0	12	0	0	0	0
0	0	0	0	0	0	0	0	9	0	0	19	0
28	0	0	0	0	0	1	0	11	0	0	24	0
5	28	0	0	46	0	0	0	0	0	28	0	9
0	0	44	0	0	0	0	0	0	44	16	0	0
3	0	39	0	0	0	0	40	0	0	27	2	0
0	14	18	0	0	21	0	47	0	0	0	27	0
31	20	0	28	28	0	2	0	18	0	0	0	0
47	0	0	34	42	0	18	0	0	0	30	0	0
30	9	19	36	0	0	46	6	36	0	0	8	0

25)

0	0	45	7	27	0	0	0	0	1	44	0	0
0	0	13	0	0	35	0	44	0	29	0	0	37
0	27	0	14	19	0	5	0	10	27	11	0	37
38	0	0	0	0	0	2	1	1	0	0	9	0
41	0	0	0	0	0	0	15	12	0	0	0	48
0	34	0	0	28	0	0	0	0	0	0	27	0
0	0	0	27	0	2	0	10	0	30	43	34	16
0	21	13	0	0	0	0	0	9	0	0	2	0
0	33	0	0	25	0	38	0	0	37	0	0	4

0	9	9	0	28	7	0	0	17	0	5	0	0
16	27	49	0	0	10	0	16	3	0	0	15	4
0	0	0	0	0	0	0	0	0	0	32	0	0
0	0	0	26	0	0	0	0	0	39	11	20	0

26)

0	0	0	18	32	0	0	0	0	0	0	0	0
48	0	5	0	22	31	0	0	39	0	28	12	28
0	0	0	0	36	0	28	41	13	16	0	32	0
0	0	0	0	46	1	23	0	0	0	0	25	0
36	0	3	0	0	0	41	0	29	0	0	42	0
10	0	25	0	29	0	5	0	0	0	0	4	0
17	0	0	0	0	39	0	0	2	0	0	6	22
0	38	1	2	16	0	0	0	0	0	0	0	0
2	0	0	35	24	0	30	0	0	0	0	0	46
0	0	0	0	47	34	32	0	29	0	0	37	6
0	0	0	0	0	0	0	26	36	0	0	0	4
44	33	8	0	0	0	0	0	0	0	0	0	41
0	38	0	0	0	15	0	33	4	0	21	27	0

27)

0	0	0	0	36	0	40	25	47	42	0	38	0
41	0	0	0	26	47	15	0	0	13	6	0	30
0	0	0	25	42	0	42	20	0	31	29	0	6
11	42	33	0	40	14	0	14	0	35	0	0	47
0	0	31	20	0	4	0	21	0	0	0	3	18
0	4	0	12	0	0	2	0	0	0	0	0	0
0	0	13	0	31	48	0	0	0	0	0	0	27
0	0	45	0	0	0	20	0	0	0	0	18	0
17	0	14	32	0	39	5	0	0	48	0	29	12
0	46	25	0	0	19	0	40	20	0	0	0	16
23	22	6	0	5	37	15	34	25	28	0	0	0
35	3	26	35	0	0	17	0	8	45	31	0	0
0	12	0	0	0	0	0	0	46	0	30	0	0

28)

0	0	0	21	25	39	0	27	0	2	0	0	27
0	0	0	9	8	18	2	0	0	0	0	37	0

41	46	0	0	7	46	44	3	27	0	0	0	10
0	0	0	0	0	21	19	4	25	0	0	42	6
31	0	0	19	0	45	26	22	1	0	0	40	0
49	0	48	40	33	0	40	0	9	42	46	0	42
48	4	0	0	0	0	0	48	35	0	41	0	39
38	0	25	24	38	7	14	0	25	21	0	0	0
10	0	49	0	0	0	0	0	0	2	8	30	0
0	30	26	0	0	0	0	0	0	0	32	3	8
2	0	20	0	0	20	31	7	47	8	0	0	0
34	13	34	17	37	0	0	9	0	45	0	0	0
0	35	0	41	37	0	0	40	38	0	39	0	0

29)

0	18	27	23	0	42	0	39	0	0	21	0	0
7	0	0	0	41	27	0	6	0	0	8	0	0
0	0	0	0	0	0	47	0	0	44	0	27	4
25	48	0	0	0	0	0	37	0	0	0	0	14
45	8	33	0	0	0	0	0	44	18	41	0	0
0	19	0	7	23	0	11	0	35	0	0	0	13
0	39	0	0	0	43	0	3	26	16	33	0	38
10	21	2	42	30	15	0	0	0	0	0	48	0
27	29	11	36	47	0	0	8	0	0	12	0	0
0	35	0	0	0	0	19	0	0	0	0	0	0
0	0	19	0	3	0	0	0	49	36	0	31	40
43	0	0	0	0	25	21	0	0	21	0	0	0
0	19	0	0	4	5	0	0	21	7	0	15	0

30)

0	0	35	45	37	0	43	0	31	24	49	0	0
42	0	0	36	0	0	17	44	27	48	30	0	7
0	0	0	12	0	0	0	0	0	22	33	0	0
39	0	0	0	9	9	11	0	0	0	9	0	13
0	0	0	37	0	9	0	0	0	10	0	0	0
0	6	30	14	0	0	0	0	18	42	0	11	0
27	43	29	0	0	22	0	0	0	0	22	0	0
0	0	0	0	18	0	0	0	0	0	0	23	11
37	0	0	0	26	0	34	36	0	30	0	31	44
7	3	0	19	0	0	24	0	41	0	0	0	16
12	0	0	38	0	19	0	3	45	0	0	0	44
0	37	0	0	13	26	7	0	0	0	0	0	27

6	42	0	38	1	0	32	0	45	0	0	37	0
---	----	---	----	---	---	----	---	----	---	---	----	---

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

Рассмотрим задачу определения максимального потока между двумя выделенными узлами связной сети. Каждая дуга сети обладает пропускными способностями в обоих направлениях, которые определяют максимальное количество потока, проходящего по данной дуге. Ориентированная (односторонняя) дуга соответствует нулевой пропускной способности в запрещенном направлении.

Пропускные способности  $c_{ij}$  сети можно представить в матричной форме. Для определения максимального потока из источника  $s$  в сток  $t$  применяются следующие шаги.

*Шаг 1.* Найти цепь, соединяющую  $s$  с  $t$ , по которой поток принимает положительное значение в направлении  $s \rightarrow t$ . Если такой цепи не существует, перейти к шагу 3. В противном случае перейти к шагу 2.

*Шаг 2.* Пусть  $c_{ij}^-$  ( $c_{ij}^+$ ) — пропускные способности дуг цепи  $\langle s, t \rangle$  в направлении  $s \rightarrow t$  ( $t \rightarrow s$ ) и

$$\theta = \min c_{ij}^- > 0.$$

Матрицу пропускных способностей  $c_{ij}$  изменить следующим образом:

- вычесть  $\theta$  из всех  $c_{ij}^-$ ;
- прибавить  $\theta$  ко всем  $c_{ij}^+$ .

Заменить текущую  $c_{ij}^-$  матрицу на вновь полученную и перейти к шагу 1.

Операция «а» дает возможность использовать остатки пропускных способностей дуг выбранной цепи в направлении  $s \rightarrow t$ . Операция «б» восстанавливает исходные пропускные способности сети, поскольку уменьшение пропускной способности дуги в одном направлении можно рассматривать как увеличение ее пропускной способности в противоположном направлении.

*Шаг 3.* Найти максимальный поток в сети. Пусть  $C = \|c_{ij}\|$  — исходная матрица пропускных способностей, и пусть  $C^* = \|c_{ij}^*\|$  — последняя матрица, получившаяся в результате модификации исходной матрицы (шаги 1 и 2). Оптимальный поток  $X = \|x_{ij}\|$  в дугах задается как

$$x_{ij} = \begin{cases} c_{ij} - c_{ij}^*, & c_{ij} > c_{ij}^* \\ 0, & c_{ij} \leq c_{ij}^* \end{cases}.$$

Максимальный поток из  $s$  в  $t$  равен

$$z = \sum_i x_{si} = \sum_j x_{tj}.$$

Заметим, что  $z$  есть сумма всех положительных 0, определенных на шаге 2. Таким образом можно объяснить, почему используются положительные элементы матрицы  $C - C^*$  для определения результирующего потока в направлении  $s \rightarrow t$ .

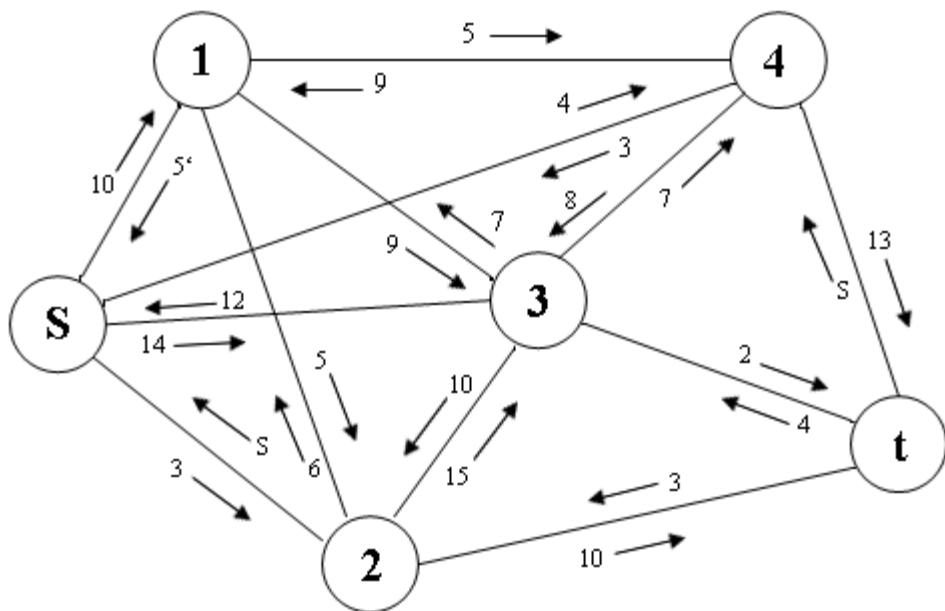


Рис. 19. 1

Пример. Рассмотрим сеть на рис. 19.1 с данными пропускными способностями. Соответствующая матрица пропускных способностей  $C$  приведена в табл. 19.1.

Таблица 19. 1

		$s$	1	2	3	4	$t$
		$s$	10	3	14	4	
		$1$	+		5	9	5-
		$2$	5	6	15		10
		$3$	12	7	10	7	2
		$4$	3	9+		8	-
		$t$		3	4	5+	

Цепь  $s \rightarrow 1 \rightarrow 4 \rightarrow t$   $\theta = \min \{10, 5, 13\}$ .

В качестве исходной цепи можно выбрать  $s \rightarrow 1 \rightarrow 4 \rightarrow t$ . Таким образом, ячейки  $(s, 1)$ ,  $(4, t)$  и  $(4, 1)$  помечаются знаком (-), ячейки  $(s, 4)$ ,  $(4, 1)$  и  $(4, 4)$  — знаком (+). Для данной цепи максимальный поток определяется как

$$\theta = \min c_{s1}, c_{14}, c_{4t} = \min 10, 5, 13 = 5.$$

Заметим, что можно выбирать различные исходные цепи. Очевидно, что хороший выбор (вначале и на каждой итерации) должен давать наибольшее значение  $\theta$ . Однако при этом, возможно, понадобится перебрать несколько вариантов, что в конечном итоге оказывается малоэффективным. При программировании алгоритма цепь удобно определять непосредственно из матрицы С, начиная с первой строки ( $s$ -строки) и выбирая следующий узел среди тех, которые соединены с  $s$  положительным потоком. Далее рассматривается строка, соответствующая выбранному узлу, и выбирается следующий узел, соединенный с предыдущим положительной дугой. Процесс продолжается до тех пор, пока не будет достигнут узел  $t$ .

Матрица С в табл. 1 корректируется путем вычитания  $\theta = 5$  из всех элементов, помеченных знаком (-), и сложения со всеми элементами, имеющими знак (+). Результаты приведены в табл. 19.2.

Таблица 19.2

	$s$	1	2	3	4	$t$
$s$		5	3	14	4	
1	10		5	9	0	
2	5	6		15		10
3	12	7	10		7	2
+ 4	3	14		8		8
$t$			3+	4	10	

Цепь  $s \rightarrow 3 \rightarrow 2 \rightarrow t$   $\theta = \min \{14, 10, 10\} = 10$ .

Результаты последующих итераций приведены в табл. 19.3—7. Из табл. 19.2 следует, что между  $s$  и  $t$  нельзя построить цепей с положительным потоком, поскольку все элементы в столбце  $t$  равны нулю. Таким образом, табл. 19.7 дает матрицу  $C^*$ .

Таблица 19.3

	s	1	2	3	4	t
s		5-	3	4	4	
1	10		5	9-	0	
+						
2	5	6		25		0
3	22	7+	0		7-	2
4	3	14		8+		8-
t			13	4	10	
					+	

Цепь  $| \rightarrow 1 \rightarrow 3 \rightarrow 4 \rightarrow t^- | \quad \theta = \min 5|9,7,8 \not\rightarrow 5.$

В табл. 19.1 (матрица  $C$ ) и 3.3 (матрица  $C^*$ ) приведены данные, характеризующие оптимальный поток, которые получаются вычислением  $X = C - C^*$  и заменой отрицательных величин нулями. Табл. 19.8 дает матрицу  $X$ .

Таблица 19.4

	s	1	2	3	4	t
s		0	3	4	4-	
1	15		5	4	0	
2	5	6		25		0
3	22	7+	0		2	2-
4	3+	14		13		3-
t			13	4	15	
					+	

Цепь  $| \rightarrow 4 \rightarrow t^- | \quad \theta = \min 4|3 \not\rightarrow 3.$

Таблица 19.5

	s	1	2	3	4	t
s		0	3	4-	1	
1	15		5	4	0	
2	5	6		25		0
3	22	12	0		2	2-
+						
4	6	14		13		0
t			13	4	18	
					+	

Цепь  $| \rightarrow 3 \rightarrow t^- | \quad \theta = \min 4|2 \not\rightarrow 2.$

Таблица 19. 6

	s	1	2	3	4	t
s	0	3	2	1		
1	15		5	4	0	
2	5	6		25		0
3	24	12	0		2	0
4	6	14		13		0
t			13	4	20	

(Между  $s$  и  $t$  нельзя построить цепь)

Таблица 19.7

	s	1	2	3	4	t
s	10		12	3		
1			5	5		
2						10
X			10		5	2
3						
4						13
t						

Из табл. 3. 7 видно, что

$$z = \sum x_{si} = 10 + 12 + 3 = \sum x_{ti} = 10 + 2 + 13 = 25.$$

Сумма всех  $\theta \leftarrow 5 + 10 + 5 + 3 + 2 = 25$  также дает максимальный поток.

Графическое решение представлено на рис. 3.2. Здесь уместно ввести понятие **минимального разреза**. **Разрез** в связной сети представляет собой такое множество дуг, которое определяет нулевой поток из  $s$  в  $t$ , если пропускные способности этих дуг полагаются равными нулю. Пропускная способность разреза равна сумме пропускных способностей его дуг. В сети на рис. 3. 1 можно выделить следующие разрезы:

Разрез	Пропускная способность
$(s, 1), (s, 2), (s, 3), (s, 4)$ $(4, t), (3, t), (2, t)$ $(1, 4), (s, 4), (3, 4), (3, t), (2, t)$	$10 + 3 + 14 + 4 = 31$ $13 + 2 + 10 = 25$ $5 + 4 + 7 + 2 + 10 = 28$

Интуитивно очевидно, что максимальный поток можно найти, перебирая *все* разрезы сети. Разрез минимальной пропускной способности даст решение. Это интуитивное соображение на самом деле можно доказать, используя **теорему о максимальном потоке — минимальном разрезе**,

согласно которой максимальный поток в сети равен пропускной способности минимального разреза.

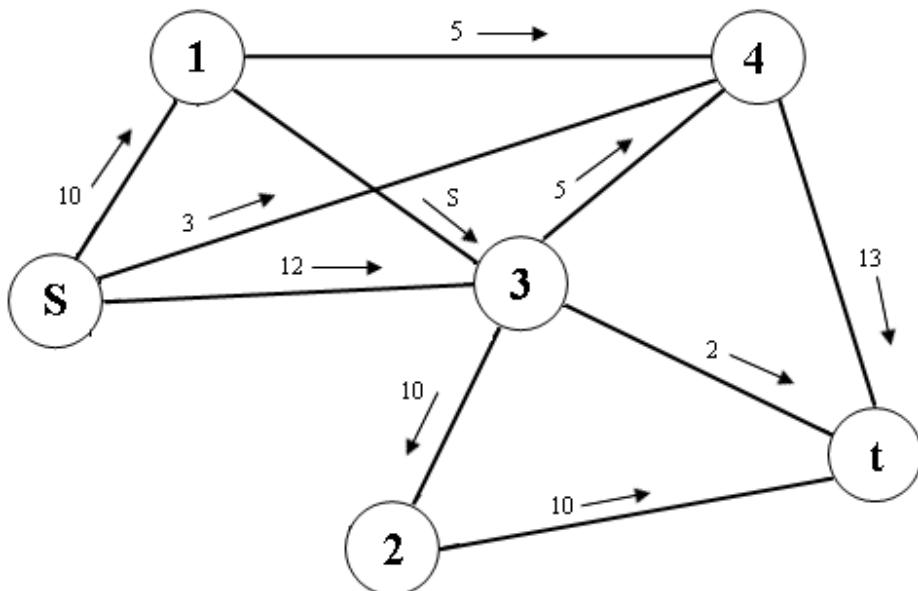


Рис 19.2

Задачи о кратчайшем пути и максимальном потоке можно сформулировать как задачи линейного программирования. Следует, однако, подчеркнуть, что решение сетевых задач симплекс-методом едва ли целесообразно. С другой стороны, изучение формулировок сетевых задач как задач ЛП помогает идентифицировать модели ЛП, которые на первый взгляд не являются сетевыми, но которые либо непосредственно, либо с некоторыми модификациями можно свести к сетевым. Очевидное преимущество такого подхода состоит в том, что при использовании сетевых постановок эффективность вычислений может значительно увеличиться.

Модель линейного программирования для задачи о кратчайшем пути строится следующим образом.

1. Каждая переменная соответствует дуге.
2. Каждое ограничение соответствует узлу.

Пусть  $x_{ij}$  представляет величину потока по дуге  $\langle i, j \rangle$ . Тогда задача о кратчайшем пути в сети с  $n$  узлами формулируется как

$$\text{минимизировать } z = \sum_{i,j} d_{ij} x_{ij}$$

при ограничениях

$$\sum_{1,j} x_{1j} = 1 \text{ (исходный пункт)},$$

$$\sum_{i,k} x_{ik} = \sum_{k,j} x_{kj} \text{ (для всех } k \neq 1 \text{ или } n\text{)},$$

$$\sum_{i,n} x_{in} = 1 \text{ (пункт назначения),}$$

$$x_{ij} \geq 0 \text{ для всех } i \text{ и } j.$$

Ограничения модели линейного программирования соответствуют формулировке задачи о кратчайшем пути как транспортной задачи с промежуточными пунктами Единица потока доставляется из узла 1 в узел  $n$ . Первым и последним ограничениями устанавливается, что суммарный поток (сумма переменных), выходящий из узла 1, равен 1, как и суммарный поток, поступающий в узел  $n$ . В любом промежуточном узле суммарный входящий поток равен суммарному выходящему потоку. Целевая функция требует, чтобы общее расстояние, пройденное единицей потока, было минимальным.

Следует подчеркнуть, что данная постановка имеет реальный смысл, если  $x_{ij} = 0$  или 1, т. е. дуга  $\langle j \rangle$  принадлежит кратчайшему пути, только если  $x_{ij} = 1$ . Если  $x_{ij} = 0$ , то  $\langle j \rangle$  не входит в кратчайший путь. Несмотря на то, что условия  $x_{ij} = 0$  или 1 не отражены в модели в явном виде, ее специальная структура всегда приводит к оптимальному решению, которое удовлетворяет этим требованиям. В самом деле, модель обладает свойством абсолютной унимодулярности, согласно которому в решении задачи линейного программирования всегда  $x_{ij} = 0$  или 1

Таким же образом к задаче линейного программирования можно свести задачу о максимальном потоке. Пусть  $y$  — поток из источника 1 в сток  $n$ . Обозначив поток в дуге  $\langle j \rangle$  через  $x_{ij}$ , получим следующую модель линейного программирования:

максимизировать  $z = y$

при ограничениях

$$\sum_{1,j} x_{1j} = y \text{ (источник),}$$

$$\sum_{i,k} x_{ik} = \sum_{k,j} x_{kj} \text{ (для всех } k \neq 1 \text{ или } n\text{),}$$

$$\sum_{i,n} x_{in} = y \text{ (сток),}$$

$$0 \leq x_{ij} \leq c_{ij} \text{ для всех } i \text{ и } j,$$

где  $c_{ij}$  обозначает пропускную способность дуги  $\langle j \rangle$ . Заметим, что

ограничения строятся по той же схеме, которая использовалась для построения модели ЛП задачи о кратчайшем пути.

## **5. Порядок выполнения работы**

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_  
\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Что значит, найти в графе максимальный поток?
2. Где применяются задачи поиска максимального потока?
3. Как найти в графе минимальный разрез?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
3. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
4. Окулов С.М. Программирование в алгоритмах – М.: БИНОМ. Лаборатория знаний, 2006. - 383 с.
5. Прата С. Язык программирования С. - СПб.: Вильямс, 2013. – 960 с.
6. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400
7. Хьюз Дж. Структурный подход к программированию / Мичтом Дж. – М.: Мир, 1980. –278 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ нахождения  
максимальных потоков минимальной стоимости в графе»

Минск  
2017

## Лабораторная работа № 20

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ нахождения максимальных потоков минимальной стоимости в графе»

### 1. Цель работы

Определить минимальную стоимость доставки товара из точки **s** в точку **t**, при этом что бы маршрут имел максимальную пропускную способность.

### 2. Задание

Граф задан матрицей смежности стоимости доставки товара. Определить минимальную стоимость доставки товара из точки **s** в точку **t**.

Номер матрицы соответствует вашему номеру по списку.

1)

0	0	0	25	0	0	0	0	0	0	0	0	0
18	0	0	0	37	0	0	0	3	0	12	0	0
8	0	0	28	0	33	0	0	48	0	0	38	0
0	0	0	0	16	21	0	34	0	0	40	40	0
2	0	0	10	0	0	0	0	31	4	24	36	0
15	5	46	0	0	0	0	0	7	32	37	28	37
0	0	19	21	41	0	0	30	0	0	0	0	38
0	36	0	0	0	27	2	0	0	15	0	0	0
0	0	0	44	35	48	26	0	0	0	0	21	0
33	32	0	0	0	0	12	44	0	0	27	29	2
0	31	38	0	8	0	0	0	0	3	0	5	48
0	39	13	26	0	33	21	34	0	0	0	0	0
0	0	45	0	26	26	22	0	0	14	0	46	0

2)

0	27	0	0	0	41	49	0	0	0	0	0	22
0	0	20	0	0	37	0	3	0	30	16	26	0
0	0	0	0	22	36	0	42	0	0	30	5	0
0	0	5	0	0	40	47	15	30	0	0	0	46
18	47	46	46	0	0	0	3	12	0	26	3	9
0	0	0	13	0	0	36	0	2	0	43	0	0
37	46	39	47	39	0	0	0	2	1	30	0	42
26	0	0	24	0	24	0	0	0	21	0	0	34
8	17	45	11	0	0	0	0	0	28	0	4	0
0	0	0	0	0	0	0	0	7	0	0	25	0

0	25	0	0	0	0	0	22	22	0	0	31	25
0	0	0	13	24	0	3	0	0	49	0	0	39
40	0	16	0	0	10	0	0	0	0	0	17	0

3)

0	0	0	33	42	48	0	44	2	11	0	49	1
0	0	20	0	0	0	45	0	48	0	0	0	32
0	0	0	0	0	22	22	0	0	0	23	19	1
0	38	0	0	0	0	0	42	2	0	3	43	23
0	0	6	0	0	0	0	0	37	0	16	0	0
0	0	0	0	17	0	26	9	0	0	0	0	0
1	21	43	5	0	0	0	0	23	28	36	13	27
27	5	35	0	23	0	0	0	16	0	0	43	18
0	29	26	0	19	33	0	12	0	0	0	0	30
0	7	0	48	0	34	47	0	0	0	0	6	0
0	10	0	0	11	0	0	40	0	10	0	6	0
0	0	8	0	0	0	0	0	0	0	0	0	6
28	38	14	0	0	31	0	19	42	0	10	35	0

4)

0	0	42	34	0	17	18	10	0	0	32	0	42
26	0	0	0	0	0	0	13	13	0	0	0	27
22	26	0	0	1	0	0	0	30	13	0	8	4
0	0	35	0	0	18	28	0	22	43	0	0	31
10	31	0	28	0	0	0	0	13	0	0	37	0
0	0	32	0	0	0	47	0	12	0	0	14	22
0	40	0	0	10	0	0	0	0	49	0	0	0
24	0	26	0	0	0	0	0	18	40	31	23	35
33	9	28	0	7	41	0	4	0	0	22	0	31
0	30	0	0	5	0	0	0	0	0	43	0	41
24	45	37	17	25	0	0	0	37	8	0	3	0
0	0	44	35	27	29	0	0	24	43	0	0	0
6	0	0	36	42	0	0	0	0	43	0	0	0

5)

0	13	22	10	1	0	6	31	0	1	0	39	8
0	0	0	0	0	0	41	0	28	0	26	0	29
0	0	0	0	0	0	0	40	45	2	0	15	49

0	0	0	0	0	26	41	0	0	0	0	0	2
0	4	16	17	0	0	48	0	0	34	19	0	27
0	0	40	0	0	0	14	0	18	31	5	0	0
9	16	11	0	20	42	0	0	0	24	0	0	0
0	0	30	0	0	0	4	0	0	12	3	0	39
0	6	0	0	37	0	30	0	0	15	3	48	15
46	42	0	0	0	0	0	46	43	0	23	48	9
0	6	0	15	9	28	20	0	0	0	0	20	27
0	44	0	0	45	6	0	0	0	22	0	0	0
0	12	0	0	0	20	2	0	0	0	21	32	0

6)

0	0	0	0	49	30	0	35	0	0	0	31	27
0	0	0	18	0	0	20	0	10	0	0	17	0
0	0	0	0	0	0	0	0	48	0	0	1	0
0	0	37	0	0	37	0	0	0	13	0	0	35
0	0	0	28	0	0	21	48	0	0	21	40	0
0	24	41	5	0	0	0	0	0	32	0	0	7
0	0	30	45	4	0	0	0	15	41	32	0	2
0	34	0	0	22	0	20	0	0	37	33	10	0
0	0	21	0	12	22	0	28	0	0	21	0	0
41	0	17	1	0	0	13	0	0	0	18	0	29
43	0	0	0	6	38	0	33	44	49	0	22	44
7	0	12	31	0	33	0	48	41	0	0	0	41
0	0	46	42	6	29	41	3	9	0	0	26	0

7)

0	0	22	33	37	0	27	0	0	47	0	0	0
0	0	0	0	45	25	38	8	19	14	41	0	0
0	0	0	31	0	0	11	0	11	5	17	12	7
0	23	0	0	15	44	0	44	0	28	48	40	0
0	2	16	0	0	47	20	0	0	48	0	0	0
7	35	0	0	0	0	0	0	0	0	0	0	1
0	27	4	11	0	0	0	4	40	0	30	0	48
7	0	33	14	0	45	26	0	0	0	9	9	0
0	0	0	0	37	0	0	32	0	33	19	0	0
0	0	0	0	0	15	21	45	41	0	43	0	31
12	0	37	0	28	0	9	28	0	39	0	0	0
0	17	30	0	0	0	27	0	0	0	0	0	0
19	27	31	10	0	46	0	23	44	0	15	0	0

8)

0	45	1	0	42	24	16	0	0	0	0	0	0
32	0	0	10	25	2	0	38	0	0	0	0	49
20	0	0	6	25	0	29	47	0	27	0	7	0
0	47	0	0	0	16	21	0	39	0	33	48	3
42	0	0	0	0	0	0	47	47	0	0	0	0
25	8	0	22	0	0	0	6	32	16	0	48	0
0	0	0	0	31	0	0	0	48	0	5	37	0
0	20	23	0	0	20	0	0	39	0	15	0	0
0	0	0	0	27	0	0	0	0	18	0	0	0
20	20	31	39	0	11	0	0	0	0	0	32	41
0	8	0	6	16	0	0	40	2	13	0	0	0
39	0	0	32	0	0	7	0	23	22	27	0	0
0	0	30	23	0	0	0	7	38	0	5	43	0

9)

0	22	0	41	12	32	13	0	25	19	11	0	0
6	0	0	44	34	33	1	16	48	0	0	40	18
0	0	0	0	0	13	29	0	21	24	0	45	0
0	27	0	0	21	27	21	0	0	49	0	0	8
0	24	10	0	0	48	0	0	0	0	0	0	0
17	0	12	0	18	0	17	0	0	0	0	0	0
0	12	0	25	0	14	0	0	0	13	0	0	45
28	0	0	0	0	0	0	0	0	9	22	0	42
26	14	0	41	13	25	0	0	0	0	22	45	6
5	0	0	42	17	0	0	24	3	0	17	0	0
26	0	0	0	29	11	0	0	33	22	0	46	0
0	39	37	0	15	0	13	42	33	0	46	0	0
0	28	46	0	0	26	0	3	1	34	30	0	0

10)

0	0	0	38	22	0	0	0	0	0	0	0	0
33	0	0	34	0	0	0	41	37	0	0	37	0
0	0	0	42	13	0	0	2	39	0	0	0	8
0	0	16	0	0	0	0	4	34	41	0	30	0
47	0	42	13	0	35	1	0	48	0	9	16	47
0	17	46	34	0	0	49	0	12	0	23	0	0

0	2	0	0	0	33	0	0	0	4	9	42	44
0	0	39	25	0	9	0	0	39	0	26	0	0
45	38	30	4	21	27	16	0	0	0	39	2	2
40	17	33	19	0	0	25	42	17	0	0	13	11
0	44	0	25	41	33	0	8	0	9	0	0	43
0	15	41	0	17	26	47	20	17	47	0	0	41
0	13	37	0	0	29	0	38	3	0	7	0	0

11)

0	0	0	0	7	34	0	0	40	22	0	0	0
0	0	13	1	0	0	0	22	47	0	0	30	42
21	29	0	0	0	0	0	13	24	48	48	46	0
0	0	0	0	7	43	33	48	45	0	0	0	0
14	12	34	0	0	0	0	5	0	31	46	37	5
39	38	36	0	0	0	0	30	12	0	28	30	0
0	0	0	0	19	10	0	32	0	0	0	0	0
7	0	0	15	36	21	0	0	6	44	0	3	0
0	0	0	9	0	0	0	38	0	23	40	34	36
0	0	0	38	12	0	0	0	7	0	0	0	0
47	0	3	22	0	0	0	41	16	6	0	0	3
7	2	0	0	8	31	0	0	28	0	0	0	0
0	0	0	6	0	0	0	20	17	18	15	28	0

12)

0	0	9	0	0	17	32	36	41	0	0	28	43
34	0	13	0	0	0	1	0	42	0	0	38	0
3	0	0	0	32	49	49	0	1	31	0	20	31
7	0	17	0	39	45	0	14	41	21	0	0	0
24	46	46	30	0	34	0	0	23	0	18	0	0
0	0	0	36	0	0	0	0	0	21	0	44	0
0	0	49	24	0	0	0	0	0	0	17	0	0
19	0	43	36	48	28	0	0	12	0	43	0	11
34	40	32	16	0	0	0	0	0	45	0	0	3
0	0	6	39	7	0	0	0	5	0	0	0	0
0	19	0	0	39	9	31	0	0	36	0	0	0
39	0	0	0	3	0	11	0	19	47	0	0	0
0	25	38	0	0	41	6	43	0	0	47	0	0

13)

0	3	14	31	11	0	34	0	4	0	20	0	0
24	0	0	23	0	0	0	29	19	36	0	41	4
0	0	0	0	34	0	18	0	0	16	5	0	23
0	0	24	0	0	0	8	0	0	27	9	35	0
11	39	43	22	0	0	19	39	21	7	21	0	8
14	0	23	0	0	0	0	0	0	0	0	4	0
0	0	49	4	22	25	0	0	0	0	11	28	0
0	0	22	19	10	0	0	0	0	4	43	29	44
0	0	0	0	0	0	0	0	0	0	49	23	41
4	0	0	26	43	0	0	0	32	0	0	0	30
44	0	0	5	11	47	46	0	18	0	0	0	0
0	0	0	0	4	49	0	19	42	38	0	0	16
0	41	39	0	0	0	0	31	0	3	25	14	0

14)

0	0	17	0	47	0	0	0	0	16	42	0	0
37	0	0	7	0	0	28	0	30	43	0	0	40
0	0	0	45	38	0	2	27	0	0	40	32	40
0	26	1	0	0	0	7	21	0	0	0	38	46
0	0	34	0	0	29	48	47	33	13	0	0	19
35	0	0	3	16	0	0	48	0	0	0	24	0
0	4	0	0	22	17	0	41	0	0	32	0	0
4	0	0	23	0	0	25	0	10	0	36	20	29
0	1	13	5	28	0	42	6	0	35	37	0	0
0	47	0	0	0	0	0	0	36	0	24	0	0
44	3	0	16	47	31	22	0	2	0	0	0	0
20	33	47	20	0	0	26	0	37	3	0	0	32
0	5	0	0	31	0	9	19	2	0	25	0	0

15)

0	0	0	37	7	14	26	16	39	25	14	0	0
0	0	0	0	0	0	43	0	31	38	0	0	0
0	48	0	12	0	0	0	9	0	0	0	0	0
18	39	15	0	26	0	13	0	3	41	0	0	0
0	0	37	11	0	26	0	0	25	2	0	32	33
27	0	0	10	33	0	0	26	0	45	0	0	32
15	0	28	0	0	49	0	0	0	40	5	0	33
21	29	0	0	11	39	0	0	41	46	0	13	29

0	6	4	41	29	23	21	0	0	0	0	0	0
18	30	41	0	15	0	0	23	0	0	42	12	44
46	0	46	0	41	0	27	0	0	0	0	40	34
19	46	10	0	21	32	0	0	0	18	18	0	25
23	0	0	24	17	8	48	15	23	0	40	0	0
13	12	0	42	0	0	0	7	17	0	0	0	0
28	0	0	0	0	0	14	0	13	35	30	23	37

16)

0	32	0	0	12	0	13	0	27	1	25	43	0
0	0	0	0	0	38	0	0	49	0	37	0	23
0	14	0	30	16	0	36	0	0	48	17	19	0
32	3	8	0	46	0	0	0	2	0	0	0	0
6	47	26	0	0	0	20	0	0	29	9	0	44
0	5	0	4	0	0	0	0	45	0	0	0	40
0	31	20	26	18	0	0	24	44	3	0	5	0
0	10	15	0	45	26	2	0	37	0	4	37	43
0	0	13	10	24	0	0	10	0	0	0	0	0
0	28	19	0	40	21	0	0	0	0	0	0	14
0	0	38	0	0	0	0	0	0	46	0	0	0
0	3	15	49	0	25	0	12	42	31	0	0	0
0	0	0	0	47	0	0	0	0	4	0	0	0

17)

0	0	22	0	0	32	36	0	0	0	5	25	0
0	0	0	16	12	0	0	0	0	0	0	38	0
0	42	0	0	34	48	0	8	0	21	4	16	14
7	44	36	0	28	6	0	0	0	0	0	25	34
2	27	10	0	0	1	0	32	10	0	2	34	0
4	0	0	39	38	0	0	0	19	0	44	10	48
3	0	18	0	0	0	0	0	20	0	33	26	17
0	0	0	0	13	0	42	0	0	34	10	0	20
38	0	0	39	0	0	0	0	0	0	0	31	0
0	0	40	0	0	0	36	0	14	0	0	0	0
12	8	43	26	38	13	0	35	48	23	0	43	0
36	20	1	0	0	36	0	0	13	42	0	0	0
45	23	0	9	0	19	41	29	0	0	28	0	0

18)

0	0	0	47	0	0	8	0	46	0	4	34	30
40	0	43	38	0	24	0	0	0	13	6	31	0
0	0	0	25	0	21	21	0	33	0	0	38	0
24	3	28	0	15	33	8	31	20	23	0	0	0
33	48	0	1	0	36	37	8	0	7	17	16	0
20	40	26	1	0	0	0	49	1	0	0	46	8
0	0	0	22	25	0	0	0	0	24	0	0	0
18	7	4	0	0	0	0	0	49	0	0	21	7
0	44	23	0	21	4	0	0	0	4	0	0	0
11	0	48	6	23	17	0	37	0	0	0	32	11
0	0	0	0	40	0	12	10	0	0	0	3	0
0	31	0	0	24	0	45	0	0	0	0	0	46
0	0	40	9	11	0	0	22	1	0	0	41	0
0	0	0	0	36	0	0	0	0	0	17	0	0
30	39	0	27	0	46	48	25	0	39	0	0	0

19)

0	40	0	7	9	27	0	0	0	0	42	0	21
0	0	0	0	42	47	0	18	29	18	0	0	47
19	0	0	0	0	1	35	0	0	0	38	0	19
0	49	0	0	37	0	0	0	39	1	18	0	14
46	0	26	32	0	0	0	0	0	0	0	4	19
0	4	42	0	0	0	8	3	21	0	4	32	0
0	35	23	36	46	0	0	0	0	0	19	43	11
14	0	49	37	30	0	0	0	45	0	30	4	0
0	0	0	0	24	11	0	37	0	0	3	38	0
34	0	0	0	0	5	29	26	12	0	41	0	46
0	8	0	35	0	0	0	47	44	10	0	0	2
0	0	0	4	21	0	0	15	42	11	16	0	0
0	45	0	0	15	49	35	49	0	0	15	0	0

20)

0	7	28	0	45	13	0	4	0	0	0	0	37
0	0	49	0	0	0	0	0	29	0	0	0	0
0	3	0	0	0	0	0	33	6	0	0	22	0
0	0	13	0	0	31	0	6	0	0	0	0	0
0	40	0	0	0	0	0	6	0	34	23	0	0
33	0	41	3	43	0	0	0	30	32	20	10	0
35	0	0	0	0	0	0	0	0	41	0	0	22

0	0	16	0	46	0	26	0	30	0	0	15	0
37	11	0	0	20	31	0	0	0	5	0	30	0
0	12	0	9	3	0	0	3	0	0	0	0	26
47	49	0	23	0	0	12	0	39	0	0	0	0
42	7	27	48	9	0	0	18	0	0	0	0	47
0	2	0	0	0	0	2	30	44	0	45	47	0

21)

0	0	10	0	16	24	0	16	0	16	0	4	0
27	0	0	0	0	0	0	0	0	28	4	0	36
6	0	0	33	15	16	36	21	25	41	38	0	43
0	4	21	0	0	0	8	0	0	27	24	0	41
1	6	0	0	0	6	0	6	0	34	0	8	23
12	27	10	0	0	0	30	11	0	21	48	8	32
9	0	0	0	43	0	0	0	6	36	0	0	0
4	43	0	40	22	0	8	0	0	0	0	14	16
0	0	12	17	0	0	0	0	0	0	0	0	0
0	31	0	36	23	0	0	0	0	0	13	41	0
0	20	42	22	0	0	31	0	34	0	0	0	16
6	3	0	0	45	0	0	0	0	0	0	0	0
0	0	0	5	0	16	0	1	41	0	0	0	0

22)

0	35	48	0	0	11	0	0	0	0	0	4	0
36	0	42	41	1	0	31	39	0	28	0	0	0
0	31	0	25	23	0	0	0	0	0	43	0	29
47	0	0	0	0	40	0	46	0	16	0	40	10
0	40	0	2	0	0	0	0	38	0	0	0	0
33	0	0	38	0	0	4	37	0	0	43	0	0
2	0	6	26	0	46	0	40	28	39	4	18	0
0	34	0	7	49	11	22	0	0	0	3	0	26
5	0	0	22	0	10	29	0	0	12	48	22	0
0	0	13	0	0	29	23	0	0	0	10	17	0
0	0	0	0	31	0	0	20	0	34	0	14	26
4	0	0	0	39	16	0	13	0	0	31	0	41
0	0	45	49	0	10	0	22	0	46	0	0	0

23)

0	23	0	0	0	0	45	22	40	6	39	0	0
4	0	47	10	30	43	0	5	19	0	0	0	0
0	0	0	0	0	0	11	0	35	0	0	0	17
29	6	0	0	8	11	0	2	20	0	0	24	0
23	43	0	17	0	24	0	46	25	12	15	12	0
0	30	0	0	36	0	41	23	18	0	3	35	17
0	0	25	2	0	0	0	30	19	24	0	44	17
0	0	19	0	20	29	30	0	29	0	3	4	19
0	2	0	24	37	0	31	0	0	0	0	42	0
25	0	31	43	0	0	0	3	0	0	0	0	30
49	0	16	0	0	37	24	0	0	37	0	0	0
48	0	15	0	0	0	0	11	13	0	0	0	9
0	0	0	6	0	0	7	0	0	0	3	42	0

24)

0	23	24	0	0	0	46	30	0	24	41	19	2
2	0	10	17	0	21	41	0	0	44	26	0	32
0	0	0	0	3	0	28	0	0	5	0	24	0
28	0	0	0	0	28	0	0	0	23	0	0	0
20	31	37	0	0	47	36	26	0	25	37	34	48
0	0	0	16	15	0	0	0	0	10	0	0	0
0	0	33	0	49	24	0	42	43	41	0	37	0
41	43	31	43	5	7	0	0	0	0	0	0	32
11	0	0	0	0	19	36	0	0	30	6	37	0
48	0	0	8	1	0	5	6	2	0	0	1	34
0	0	27	0	0	0	0	18	0	44	0	16	0
47	49	11	27	4	0	0	0	21	0	0	0	0
0	0	0	32	0	0	0	0	45	37	0	0	0

25)

0	2	30	26	3	46	43	0	0	27	48	0	10
9	0	0	0	22	0	0	3	0	25	0	17	1
17	18	0	19	36	0	0	11	14	30	31	18	0
0	14	46	0	3	29	49	42	0	17	48	0	0
48	0	0	0	0	0	4	29	13	13	0	0	0
0	0	4	0	41	0	3	39	0	0	22	0	0
40	29	0	24	37	36	0	36	0	0	20	36	47
25	9	3	0	3	19	0	0	3	0	31	0	34

37	0	0	0	0	0	8	10	0	13	0	31	35
37	37	0	0	0	0	0	0	0	0	28	9	16
0	11	18	0	5	0	49	49	0	8	0	9	0
43	0	15	0	0	0	0	23	44	0	0	0	18
8	0	0	4	33	0	0	0	0	0	47	18	0

26)

0	20	0	12	7	0	0	13	30	9	0	0	0
46	0	0	34	26	7	0	0	0	0	0	0	4
0	3	0	28	0	0	35	0	5	0	36	0	23
20	0	0	0	0	23	19	0	0	17	0	40	25
40	0	0	29	0	21	12	43	2	0	25	40	2
20	0	12	0	0	0	0	0	14	0	37	0	10
0	0	37	43	0	0	0	30	17	0	31	39	40
0	33	36	0	0	0	0	0	29	0	0	44	0
0	0	0	41	0	0	9	30	0	48	25	45	4
28	20	6	22	4	34	0	0	0	0	0	0	0
36	34	0	0	22	29	42	0	0	0	0	32	0
0	0	1	6	0	0	25	0	37	0	0	0	15
0	20	0	4	47	49	0	0	0	23	8	0	0

27)

0	24	0	0	0	0	0	0	0	0	0	0	0
20	0	13	7	42	0	0	0	7	0	0	0	0
0	0	0	15	42	26	3	0	24	0	0	0	12
0	43	4	0	8	0	0	47	0	11	0	7	5
0	22	0	0	0	7	47	28	0	21	6	0	0
0	22	0	0	0	0	0	0	0	0	26	0	22
30	28	31	0	0	8	0	0	0	44	47	0	25
0	4	45	19	0	13	0	0	0	0	0	23	0
0	31	26	32	0	0	0	0	0	18	12	0	8
0	45	0	0	0	29	31	0	38	0	0	3	11
0	0	0	0	0	0	30	0	0	0	29	0	0
0	0	40	23	0	16	35	0	28	34	0	0	0
0	29	37	0	0	21	0	0	20	0	0	0	0

28)

0	8	0	32	0	0	43	21	0	33	0	18	0
0	0	0	0	0	0	22	0	0	0	0	0	0
14	0	0	0	33	14	12	2	0	49	28	0	28
0	31	10	0	0	0	0	40	0	0	44	0	35
0	22	0	28	0	0	27	2	15	44	0	37	25
0	8	15	9	10	0	12	32	34	21	33	29	0
0	0	0	0	0	0	0	0	0	0	5	0	0
0	0	38	30	33	18	0	0	0	48	45	0	0
0	15	18	36	14	19	2	0	0	0	13	0	0
0	0	16	0	0	33	0	47	0	0	24	0	0
0	7	0	24	27	26	0	19	0	0	0	0	0
0	29	0	40	47	47	0	0	0	0	20	0	0
0	30	4	0	39	49	8	0	22	0	0	0	0

29)

0	0	9	0	0	0	0	0	0	0	0	0	25
0	0	0	0	35	0	22	0	0	21	0	28	0
0	0	0	0	0	0	16	10	0	25	25	0	0
31	43	0	0	41	0	0	41	10	0	0	0	0
31	0	3	44	0	49	26	5	0	0	0	0	0
5	0	33	0	0	0	14	44	18	37	4	0	0
0	19	36	0	0	0	0	0	13	0	46	0	0
23	0	0	38	0	35	41	0	0	0	0	24	0
40	0	38	0	24	7	0	13	0	0	0	0	45
1	0	18	0	0	46	14	0	30	0	24	0	0
40	35	17	35	0	19	0	29	2	0	0	0	49
0	10	0	0	43	14	0	0	8	14	2	0	30
0	0	0	0	13	45	0	36	36	8	11	7	0

30)

0	33	5	20	0	0	0	0	0	0	36	0	11
0	0	48	38	0	33	0	0	43	0	0	0	14
44	46	0	40	10	0	0	0	10	39	7	0	0
15	0	7	0	18	1	26	0	46	48	0	0	0
0	0	23	19	0	0	3	7	46	0	0	9	0
47	0	0	38	0	0	0	21	0	0	0	0	0
32	4	0	5	41	0	0	0	24	45	49	0	25
48	0	29	0	4	0	23	0	0	44	0	12	32

0	0	48	39	0	45	0	0	0	0	0	5	0
46	0	29	30	36	7	0	0	0	0	0	0	0
34	11	35	29	13	0	23	10	19	0	0	25	0
0	0	46	0	10	0	0	19	0	0	47	0	0
19	0	0	0	0	19	3	42	0	20	20	0	0

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

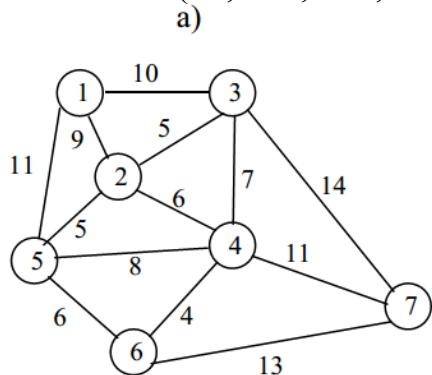
### 4. Основные теоретические сведения

#### Задача на минимум суммы

**Задача.** Имеется  $n$  населенных пунктов, в каждом из которых живет  $p_i$  школьников ( $i=1, \dots, n$ ). Надо разместить школу в одном из них так, чтобы общее расстояние, проходимое всеми учениками по дороге в школу, было минимальным.

Эта задача просто решается на основе алгоритма Флойда-Уоршелла. Сначала получаем матрицу минимальных путей из каждой вершины в каждую  $\{d_{ij}\}$ . Пусть школа размещается в вершине с номером  $k$ . Тогда общее расстояние, которое должны пройти ученики из пункта  $j$  по дороге в школу, равно произведению расстояния между вершинами  $i$  и  $j$  на количество учеников в вершине  $j$ , то есть  $d_{ij} p_j$ . Суммируя эти произведения для всех вершин, найдем общее расстояние, которое будут проходить ученики, если школа будет в пункте  $i$ . Естественно, надо выбрать такую вершину  $k$ , для которой это число будет наименьшим.

Рассмотрим сеть, показанную на рисунке а). Цифры у дуг показывают расстояние между населенными пунктами. Количество учеников в пунктах 1..7 задано числами (80, 100, 140, 90, 60, 50, 40).



а)

б)

	1	2	3	4	5	6	7	
1	0	9	10	15	11	17	24	6120
2	9	0	5	6	5	10	17	3440
3	10	5	0	7	10	11	14	3640
4	15	6	7	0	8	4	11	3800
5	11	5	10	8	0	6	19	4560
6	17	10	11	4	6	0	13	4930
7	24	17	14	11	19	13	0	8360

На рисунке б) показана матрица длин кратчайших путей. В последнем столбце сосчитано общее расстояние, проходимое учениками, если школа будет в

данной вершине. Поскольку надо выбрать наименьшее расстояние, то школу надо размещать в вершине 2.

### Минимаксная задача размещения

Имеется  $n$  населенных пунктов, в одном из которых надо разместить спецподразделение, которое должно выезжать на вызовы при срабатывании сигнализации на одном из охраняемых объектов. Надо расположить его так, чтобы самый дальний охраняемый объект достигался за минимальное время.

Прежде всего, находится матрица кратчайших путей. Затем в каждой строке  $i$  выбирается максимальное число – это будет расстояние до самого удаленного охраняемого объекта, если пункт полиции разместить в вершине  $i$ . Затем выбирается такой пункт  $j$ , для которого это число наименьшее.

Более сложной является минимаксная задача, в которой допускается размещать объект **на ребрах сети** (между пунктами). Для решения этой задачи можно использовать следующие идеи.

Рассмотрим ребро между вершинами  $i$  и  $j$ . Если расположить пункт полиции на нем, то расстояние от него до другой вершины  $k$  не может быть меньше, чем

$$\delta_k = \min(d_{ik}, d_{jk})$$

Вычислив эти величины для всех вершин  $k$ , определим максимальную из них. Это число  $\delta_0$  называется **нижней оценкой ребра  $(i, j)$** , поскольку при любом выборе места для пункта полиции на этом ребре расстояние до самого дальнего объекта не будет меньше  $\delta_0$ . Мы уже знаем минимально возможное расстояние до самого дальнего пункта при размещении отряда в одном из городов. Очевидно, имеет смысл рассматривать только те ребра, для которых нижняя оценка не превышает это расстояние.

Для каждого ребра  $(i, j)$  с нижней оценкой, меньше уже полученной, сделать следующие операции:

- ✓ найти самую дальнюю вершину  $k$  из всех тех вершин, в которые надо ехать через вершину (то есть  $d_{ik} > d_{jk}$ ) ;
- ✓ найти самую дальнюю вершину  $l$  из всех тех вершин, в которые надо ехать через вершину (то есть  $d_{il} < d_{jl}$ ) ;
- ✓ вычислить середину расстояния между вершинами  $k$  и  $l$ ; если она попадает на ребро  $(i, j)$  и расстояние до самых дальних вершин меньше уже найденного, запомнить это место.

Ниже этот алгоритм записан на языке Си:

```

minDist = 32767; // большое число
for (i = 0; i < N-1; i++)
    for (j = i+1; j < N; j++)
        { // перебрать все ребра (i,j)
            maxDi = maxDj = 0;
            for (k = 0; k < N; k++)
                { // перебрать все вершины
                    if (k == i || k == j) continue;
                    if (D[j,k] < D[i,k])
                        if (D[j,k] < maxDj) maxDj = D[j,k];
                    else
                        if (D[j,k] < maxDi) maxDi = D[j,k];
                }
            dist = (maxDi + D[i][j] + maxDj) / 2;
            if (dist < minDist && dist > maxDi &&
                dist < maxDi+D[i][j])
                {
                    minDist = dist; // запомнить новый вариант
                    ...
                }
        }
}

```

### Задача коммивояжера

#### Формулировка задачи

Эта знаменитая задача была сформулирована в 1934 г. В области дискретной оптимизации она стала испытательным полигоном, на котором проверяются новые методы. Проблема заключается в том, что эта задача относится к многочисленному классу задач, которые теоретически не решаются иначе, чем перебором.

**Задача коммивояжера.** Коммивояжер (бродячий торговец) должен выйти из первого города и, посетив по разу в неизвестном порядке города **2, 3, ..., n**, вернуться обратно в первый город. В каком порядке надо обходить города, чтобы замкнутый путь (тур) коммивояжера был кратчайшим?

Этот метод предусматривает просто перебор всех вариантов. Сначала нам надо получить некоторую допустимую последовательность обхода вершин. Поскольку первый город (1) зафиксирован, всего таких перестановок будет **(n-1)!**. Получив очередную перестановку надо найти длину пути и, если она окажется меньше уже найденного кратчайшего пути, ее надо запомнить и перейти к следующей перестановке.

Поскольку нами уже написана процедура получения всех перестановок из целых чисел от 1 до **N** (см. пункт *Комбинации и перестановки* раздела *Рекурсия* в главе III), имеет смысл ее использовать. Удобно ввести глобальные переменные

```

int Pmin[N], // лучшая перестановка
P[N], // текущая перестановка
Lmin, // минимальная длина
L, // текущая длина
D[N][N]; // матрица расстояний

```

Основная программа будет инициализировать глобальные переменные, вызывать рекурсивную процедуру построения оптимального пути и распечатывать результат:

```

void main()
{
    Lmin = 32767; // большое число
    L = 0;
    P[0] = 1; // начальная вершина 1
    Commi(1); // построить тур
    for ( int i = 0; i < N; i ++ ) // вывести результат
        printf("%d ", Pmin[i]);
}

```

Ниже приведена рекурсивная процедура, решающая задачу коммивояжера, приводится ниже.

Текущая длина пути вычисляется последовательно с добавление каждого нового звена.

```

void Commi( int q ) // q - число уже поставленных вершин
{
    int i, temp;

    if ( q == N ) { // перестановка получена
        if ( L < Lmin ) {
            Lmin = L;
            for ( i = 0; i < N; i ++ ) // запомнить новый минимальный тур
                Pmin[i] = P[i];
        }
        return;
    }
    for ( i = q; i < N; i ++ ) {
        temp = P[q]; P[q] = P[i]; P[i] = temp; // P[q] <-> P[i]
        L += D[P[q-1]][P[q]]; // добавить ребро
        Commi ( q+1 ); // рекурсивный вызов
        L -= D[P[q-1]][P[q]]; // убрать ребро
        temp = P[q]; P[q] = P[i]; P[i] = temp; // P[q] <-> P[i]
    }
}

```

Главная трудность такого переборного способа заключается в том, что для больших **n** (например, для **n>20**) число комбинаций настолько велико, что перебор нереален.

Однако, часто можно найти такую стратегию перебора, при которой сразу отбрасывается целая группа вариантов, которые заведомо не лучше уже найденного минимального решения. Одним из таких алгоритмов является метод ветвей и границ.

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_  
\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Что значит, найти в графе максимальный поток минимальной стоимости?
2. Где применяются задачи поиска максимального потока минимальной стоимости?
3. Какие знаете алгоритмы поиска максимального потока минимальной стоимости?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
3. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
4. Окулов С.М. Программирование в алгоритмах – М.: БИНОМ. Лаборатория знаний, 2006. - 383 с.
5. Прата С. Язык программирования С. - СПб.: Вильямс, 2013. – 960 с.
6. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400
7. Хьюз Дж. Структурный подход к программированию / Мичтом Дж. – М.: Мир, 1980. –278 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ методом ветвей и  
границ»

Минск  
2017

# Лабораторная работа № 21

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ методом ветвей и границ»

## 1. Цель работы

Закрепить навык работы с алгоритмом ветвей и границ.

## 2. Задание

Решите задачу из лабораторной работы № 20 методом ветвей и границ.  
Вариант соответствует вашему номеру по списку.

## 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

## 4. Основные теоретические сведения

Впервые метод ветвей и границ был предложен в 1960 г. А. Лэндом и А. Дойгом применительно к задаче целочисленного линейного программирования. Фактически «второе рождение» этого метода связано с работой Дж. Литтла, К. Мурти, Д. Суини и С. Кэрела. В ней же было предложено и принятое теперь название метода: «метод ветвей и границ». Он применим как к полностью, так и частично целочисленным задачам.

Приведем описание идеи данного метода. Рассматривается целочисленная (частично целочисленная) задача линейного программирования: минимизировать

$$z = F(x) = \sum_{j=1}^n c_j x_j \quad (21.26)$$

при условиях

$$\sum_{j=1}^n a_{ij} x_j \begin{cases} \leq \\ = \\ \geq \end{cases} b_i, i = \overline{1, m}, \quad (21.27)$$

$$0 \leq x_j \leq d_j, j = \overline{1, n}, \quad (21.28)$$

$$x_j - \text{целое}, j = 1, 2, \dots, n_1; n_1 \leq n. \quad (21.29)$$

Многогранное множество (21.27) – (21.28) предполагается ограниченным. Как в методах отсечения, процесс начинается с решения задачи линейного программирования (21.26) – (21.28). Если полученный при этом оптимальный план  $X^0 = (x_1^0, x_2^0, \dots, x_n^0)$  не удовлетворяет условиям целочисленности (21.29), то значение целевой функции  $z_0 = \xi^0$  на этом плане дает нижнюю границу для

исходного оптимума, т.е.  $\min z \geq \xi^0$ . Пусть  $x_{i_0}$  – целочисленная переменная, значение  $x_{i_0}^0$  которой в оптимальном плане  $\mathbf{X}^0$  задачи  $(D_0, F)$  является дробным ( $1 \leq i_0 \leq n_1$ ). Интервал

$$\lfloor x_{i_0}^0 \rfloor < x_{i_0} < \lfloor x_{i_0}^0 \rfloor + 1$$

не содержит допустимых целочисленных компонентов решения. Поэтому в оптимальном плане значение  $x_{i_0}$  должно удовлетворять одному из неравенств:  $x_{i_0} \leq \lfloor x_{i_0}^0 \rfloor$  или  $x_{i_0} \geq \lfloor x_{i_0}^0 \rfloor + 1$ .

Если границы изменения  $x_{i_0}$  заранее не заданы, то их можно вычислить, решив для этого две вспомогательные задачи линейного программирования, заключающиеся в максимизации и минимизации  $x_{i_0}$  при условиях (21.27) – (21.28). Введение их в задачу без учета ограничения (21.29) порождает две не связанные между собой задачи. В этом случае говорят, что задача разветвляется (или разбивается) на две подзадачи. Осуществляемый в процессе ветвления учет ограничения (21.29) позволяет исключить части многогранника допустимых решений, не содержащие точки с целыми координатами. Теперь каждая подзадача решается как задача линейного программирования (с целевой функцией исходной задачи) с условиями (21.27) и дополнительным

ограничением  $x_{i_0} = k_{i_0}$ . Если полученный оптимум оказывается допустимым для целочисленной задачи, такое решение следует зафиксировать как наилучшее. При этом нет необходимости продолжать «ветвление» подзадачи, поскольку улучшить найденное решение, очевидно, не удастся. В противном случае подзадача вновь разбивается на две подзадачи при условии целочисленности переменных, значения которых в оптимальном решении не являются целыми. Как только полученное допустимое целочисленное решение одной из подзадач оказывается "лучше" имеющегося, оно фиксируется вместо зарегистрированного ранее. Процесс ветвления продолжается до тех пор, пока каждая подзадача не приведет к целочисленному решению или пока не будет установлена возможность улучшения имеющегося решения. Зафиксированное допустимое решение и будет оптимальным.

Эффективность вычислений можно повысить, введя в рассмотрение понятие границы, на основании которого делается вывод о необходимости дальнейшего дробления каждой из подзадач. Если оптимальное решение подзадачи без требования целочисленности обеспечивает "худшее" значение целевой функции по сравнению с имеющимся, эту подзадачу далее рассматривать не следует.

Вышеизложенное можно представить в виде некоторого дерева задач, в котором вершина  $\mathbb{O}$  отвечает исходному плану  $\mathbf{X}^0$ , а каждая из соединенных с

ней ветвью вершина отвечает оптимальному плану задачи (21.26) – (21.28) при дополнительном условии  $x_{i_0} = k_{i_0}$ . Каждой из вершин приписывается граница  $\xi^k = \xi(i_0, k)$ , равная минимальному (максимальному – в случае максимизации) значению  $z$  для соответствующей задачи. Ясно, что  $\xi^0 \leq \xi(i_0, k)$  для всех  $k$ .

Опишем формально данный алгоритм.

**1. Задание множества  $D^0$ .** Множество  $D^0 \equiv D$  задается условиями (21.27) – (21.29).

**2. Задание множества  $D_v^k$ .** Множество  $D_v^k$  ( $v = 1, 2, \dots, r_k; k = 1, 2, \dots$ ) определяется условиями (21.27) – (21.28) и дополнительным условием

$$h_j \binom{k}{v} \leq x_j \leq d_j \binom{k}{v} \quad (j = 1, 2, \dots, n). \quad (13.37)$$

**3. Вычисление границ (оценок).** Для

множества  $D^0$  оценка  $\xi(D^0) = F(X^0)$ , где  $X^0$  – оптимальный план задачи линейного программирования (21.26) – (21.28). Для

множества  $D_v^k$  оценка  $\xi(D_v^k) = F(X_v^k)$ , где  $X_v^k$  – оптимальный план задачи линейного программирования (21.26), (21.27), (21.30). Если

множество  $D_v^k$  оказывается пустым, то полагаем  $\xi(D_v^k) = +\infty$  ( $-\infty$  – в случае максимизации).

**4. Вычисление планов.** Если  $X^0$  удовлетворяет условию целочисленности (21.29), то  $X^0$  – оптимальный план задачи (21.26) – (21.29).

Если  $X_v^k$  удовлетворяет условию целочисленности (21.29), то  $X_v^k$  – оптимальный план задачи (21.26), (21.27), (21.29), (21.30) и исходной задачи (21.27) – (21.29).

**5. Ветвление.** Оно необходимо в том случае, когда план  $X_v^{k(k)}$  не

удовлетворяет условию целочисленности (21.29). Пусть  $X_r^{k(k)}$  – один из нецелочисленных компонентов этого плана ( $1 \leq r \leq n$ ). Тогда

множество  $D_{v(k)}^k$  разбивается на два множества:  $D_{v(k)}^k = D_{v(k),1}^k \cup D_{v(k),2}^k$ , где

$$D_{\mathbf{v}(k),1}^k = \{X \mid X \in D_{\mathbf{v}(k)}^k, x_r \leq [x_r \binom{k}{\mathbf{v}(k)}]\},$$

$$D_{\mathbf{v}(k),2}^k = \{X \mid X \in D_{\mathbf{v}(k)}^k, x_r \geq [x_r \binom{k}{\mathbf{v}(k)}] + 1\}.$$

**Замечание 1.** Если все коэффициенты  $c_j$  в соотношении (21.26) нецелые при  $j \leq n_1$  и равны нулю при  $j > n_1$ , то оценку  $\xi(D_v^k)$  можно заменить более сильной:

$$\xi(D_{\mathbf{v}}^k) = JF(X_{\mathbf{v}}^{(k)}),$$

где  $\lceil a \rceil$  – наименьшее целое число, не меньшее чем  $a$ .

**З а м е ч а н и е 2 .** Вопрос о наилучшем способе выбора переменной ветвления или последовательности решения конкретных задач в настоящее время не решен.

### **Пример. Максимизировать**

$$F(x) = 7x_1 + 9x_2 \quad (21.31)$$

## при условиях

$$\begin{cases} -x_1 + 3x_2 \leq 6, \\ 7x_1 + x_2 \leq 35, \\ x_1 \geq 0, x_2 \geq 0, \end{cases} \quad (21.32)$$

$x_1, x_2$  – целые. (21.33)

$$0\text{-й шаг. Оптимальный план задачи (21.31) - (21.33)}$$

**следующий:**  $X^0 = (9/2; 7/2)$ .      **Имеем**  $\xi(D^0) = F(X^0) = 63$ .

**План  $X^0$  не удовлетворяет условию целочисленности (21.33). Возьмем его**

$$D^0 = D_1^1 \cup D_2^1$$

$$D_1^1 = \{X \mid X \in D^0, x_1 \leq 4\} \quad D_2^1 = \{X \mid X \in D^0, x_1 \geq 5\}$$

1-й шаг. Решим две ЗЛП, состоящие из в максимизации функции (21.31) по  $D_1^1$  и  $D_2^1$

## Решение задачи $(D_1^1, F)$ .

	$x_3$		1						
0	$x_4$		5						5
	$x_5$								
	$F$			7	9				
	$x_2$								
$I$			1/3			/3			
	$x_4$	3	2/3			1/3			/2
	$x_5$								
	$F$		8	10					
	$x_2$	0/3							
$II$	$x_4$	1/3							
	$x_1$								
	$F$	8						0	

Максимум достигается при  $x_1 = 4, x_2 = 10/3$  и  $\xi'(D_1^1) = 58$ .

Решение задачи  $(D_2^1, F)$ :

Но мер итерации	Базис ные переменные		$a_{11}$	$a_{12}$	$a_{13}$	$a_{14}$	$a_{15}$	$a_{21}$	$a_{22}$
0	$x_3$		1				0	0	-
	$x_4$	5					1	0	5
	$x_5$						0	-	5
	$F$		7	9			0	0	
	$x_2$	1					0	-	1
	$x_4$						1	7	0

I	$x_1$					0	-	-
	$F$	5		9		0	7	
II	$x_3$	1				3	-	10
	$x_2$					1	7	
	$x_1$					0	-	
	$F$	5				9	5	

Максимум достигается при  $x_1 = 5$ ,  $x_2 = 0$  и  $\xi'(D_2^1) = 35 [= 35]$ .

Производим разветвление по  $D_1^1$ :

$$D_1^1 = D_{1,1}^1 \cup D_{1,2}^1,$$

$$\text{где } D_{1,1}^1 = \{X \mid X \in D_1^1, x_2 \leq 3\}, \quad D_{1,2}^1 = \{X \mid X \in D_1^1, x_2 \geq 4\}.$$

$$D_{1,1}^1 \equiv D_1^2, D_{1,2}^1 \equiv D_2^2, D_2^1 \equiv D_3^2$$

Переобозначения:  
2-й шаг. Решаем две задачи линейного программирования, заключающиеся в максимизации функции (21.31) по множествам  $D_1^2$  и  $D_2^2$ .

Решение задачи  $(D_1^2, F)$ :

Но мер итерации	Базис ные переменные							
0	$x_3$			1				
	$x_4$							
	$x_5$							
	$F$		7	9				
I	$x_3$	1			2/7		/7	/2
	$x_1$				/7		/7	5
	$x_5$							

	$F$	5	8					
$\Pi$	$x_3$	1/7						
	$x_1$	2/7						
	$x_2$							
	$F$	9						

Максимум достигается при  $x_1 = 32/7, x_2 = 3$  и  $\xi(D_1^2) = \lceil 59 \rceil = 59$ .

Рассмотрим следующий шаг решения задачи.

Исходная система (0-я итерация) оказалась приведенной к единичному базису  $(\bar{a}_3, \bar{a}_4, \bar{a}_5)$  ( $\bar{a}_3 = (1; 0; 0), \bar{a}_4 = (0; 1; 0), \bar{a}_5 = (0, 0, 1)$ ). Однако свободный член в третьем уравнении отрицательный. Поэтому данной системе соответствует исходное базисное, но не опорное решение  $\bar{X} = (0; 0; 6; 35; -4)$ .

Выделенное уравнение (третью строку) перепишем, умножив все коэффициенты на  $-1$ . Все свободные члены теперь положительные. Первое и второе уравнения (кроме выделенного) разрешены относительно базисных неизвестных  $x_3$  и  $x_4$ . Дальнейшие преобразования системы будем проводить согласно правилам симплексных преобразований, выбирая разрешающий столбец из такого условия, чтобы он имел в выделенной (третьей) строке положительный элемент. Разрешающая строка выбирается из условия

$$\min_i \left\{ \frac{a_{i0}}{a_{ip}} \right\}$$

положительный элемент.

Решение задачи  $(D_2^2, F)$ .

Но мер итерации	Базис ные переменные	$x_1$	$x_2$	$x_3$	$x_4$	$x_5$	$F$	$C$
0	$x_3$						0	
	$x_4$			1			1	
	$x_5$				1		0	
	$F$		7	9			0	

	$x_3$		1			0	
I	$x_4$					1	
		5					5
	$x_5$					0	1
	$F$					0	
			7	9			
<hr/>							
II	$x_2$					0	
			1/3		/3		
	$x_4$	3		<span style="border: 1px solid black; padding: 2px;">2/3</span>	1/3	1	/2
	$x_5$		/3		1/3	0	1
	$F$		8	10		3	
<hr/>							
III	$x_2$					1	
		/2			/22	/22	
	$x_1$		/2		1/22	/22	3
	$x_5$		/3		7/22	1/22	1
	$F$		3		5/11	5/11	1

В полученной после II итерации таблице выделенное третье уравнение снова оказалось не разрешенным относительно базисного неизвестного (первое и второе уравнения разрешены относительно  $x_2$  и  $x_4$ ), но свободный член в этом уравнении уменьшился (2 вместо 4). Поэтому выполним III итерацию. Разрешающей строкой является вторая, разрешающим может служить только один столбец.

После III итерации приходим к таблице, в которой выделенная третья строка не имеет ни одного положительного элемента, кроме свободного члена. В этом случае исходная система уравнений не имеет ни одного решения с неотрицательными значениями неизвестных, в том числе и опорного решения.

Итак, множество  $D_2^2$  оказывается пустым и для него имеем  $\xi(D_2^2) = -\infty$ .  
Получаем:

$$1) \quad X \begin{pmatrix} 2 \\ 1 \end{pmatrix} = (32/7; 3); \quad \xi(D_1^2) = 59;$$



	$F$	5						
--	-----	---	--	--	--	--	--	--

Максимум достигается при  $x_1 = 4, x_2 = 3$  и  $\xi'(D_1^3) = 55$ .

Решение задачи  $(D_2^3, F)$ :

Но мер итерации	Базис ные переменные							
0	$x_3$		1					
	$x_4$		5					
	$x_5$							
	$x_6$							1
	$F$		7	9				
$I$	$x_3$	1					1	$1/3$
	$x_4$							
	$x_5$							
	$x_1$						1	
	$F$	5		9			7	
$II$	$x_3$	1			3		22	
	$x_2$							
	$x_5$				1		7	
	$x_1$						1	
	$F$	5					6	

Максимум достигается при  $x_1 = 5, x_2 = 0$  и  $\xi'(D_2^3) = 35$ .  
Итак,

1)  $X\begin{pmatrix} 3 \\ 1 \end{pmatrix} = (4, 3), \xi(D_1^3) = 55;$

2)  $X\begin{pmatrix} 3 \\ 2 \end{pmatrix} = (5, 0), \xi(D_2^3) = 35;$

3)  $D_3^3 = \square, \xi(D_4^3) = -\infty;$

4)  $X\begin{pmatrix} 3 \\ 4 \end{pmatrix} = (5, 0), \xi(D_4^3) = 35.$

Получим

целочисленный  $\bar{X} = X\begin{pmatrix} 3 \\ 1 \end{pmatrix} = (4; 3)$ , причем  $\xi' = \max\{\xi(D_1^3), \xi(D_2^3), \xi(D_3^3), \xi(D_4^3)\} = \max\{55, 35, -\infty, 35\} = 55 \geq F(\bar{X}) = 55$ .

Поэтому план  $\bar{X} = (4; 3)$  оптимальный.

**Главный недостаток алгоритма метода ветвей и границ** заключается в необходимости полностью решать задачи линейного программирования, ассоциированные с каждой из вершин многогранника допустимых решений. Для задач большой размерности это требует значительных и, в известной степени, неоправданных с практической точки зрения затрат времени.

Таким образом, несмотря на отмеченные недостатки данного метода, можно утверждать, что в настоящее время алгоритмы метода являются наиболее надежным средством решения целочисленных задач, встречающихся в практических исследованиях. Это не означает, однако, что любая целочисленная задача может быть решена с помощью рассматриваемого метода. Тем не менее проблема выбора между методом отсечений и методом ветвей и границ в подавляющем большинстве случаев обоснованно разрешается в пользу последнего.

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Назовите основные принципы метода ветвей и границ.
2. Какое множество называется рекордом?
3. Сформулируйте условие задачи коммивояжера.
4. Что означает выражение привести матрицу по строкам?
5. Как строится дерево перебора для расшифровки криптограмм?
6. Что такое функция штрафа?
7. Как исключается досрочное завершение тура?
8. Что такая нижняя граничная оценка?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
3. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
4. Окулов С.М. Программирование в алгоритмах – М.: БИНОМ. Лаборатория знаний, 2006. - 383 с.
5. Прата С. Язык программирования С. - СПб.: Вильямс, 2013. – 960 с.
6. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400
7. Хьюз Дж. Структурный подход к программированию / Мичтом Дж. – М.: Мир, 1980. –278 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание программ «жадных» алгоритмов»

Минск  
2017

## Лабораторная работа № 22

**Тема работы:** «Разработка, отладка и испытание программ “жадных” алгоритмов»

### 1. Цель работы

Отработка навыка реализации «жадных» алгоритмов.

### 2. Задание

Решите задачу из лабораторной работы № 19 методом ветвей и границ. Вариант соответствует вашему номеру по списку.

### 3. Оснащение работы

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### 4. Основные теоретические сведения

#### Жадные алгоритмы

Жадными (градиентными) называют алгоритмы, действующие по принципу "максимальный выигрыш на каждом шаге". Такая стратегия не всегда ведет к конечному успеху - иногда выгоднее сделать не наилучший, казалось бы, выбор на очередном шаге с тем, чтобы в итоге получить оптимальное решение. Но для некоторых задач применение жадных алгоритмов оказывается оправданным. Одной из самых известных задач такого рода является задача об *оптимальном каркасе*. Существует общая теория, обосновывающая применимость жадных алгоритмов к задачам определенного типа, к которому относится и рассмотренный в "предыдущей лекции" *алгоритм Крускала*. В этой лекции будет дан набросок этой теории. Затем будет рассмотрено применение жадного алгоритма в сочетании с методом увеличивающих путей для решения задачи о *паросочетании наибольшего веса в двудольном графе* со взвешенными вершинами.

#### Матроиды

Когда возник вопрос о том, каковы обстоятельства, при которых жадный алгоритм результативен, или иначе - что особенного в тех задачах, для которых он дает точное решение, оказалось, что математическая теория, с помощью которой что-то в этом можно прояснить, уже существует. Это теория *матроидов*, основы которой были заложены Уитни в работе 1935 г. Целью создания теории *матроидов* было изучение комбинаторного аспекта линейной независимости, а в дальнейшем обнаружились разнообразные применения понятия *матроида*, первоначально совсем не имевшиеся в виду.

**Определение.** Матроидом называется пара  $M = (E, \Phi)$ , где  $E$  - конечное непустое множество,  $\Phi$  - семейство подмножеств множества  $E$ , удовлетворяющее условиям:

- (1) если  $X \in \Phi$  и  $Y \subseteq X$ , то  $Y \in \Phi$ ;

- (2) если  $X \in \Phi$ ,  $Y \in \Phi$  и  $|X| < |Y|$ , то существует такой элемент  $a \in Y - X$ , что  $X \cup (a) \in \Phi$ .

Элементы множества  $E$  называются элементами *матроида*, а множества из семейства  $\Phi$  - *независимыми множествами матроида*. Максимальное по включению независимое множество называют *базой матроида*. Из аксиомы (2) следует, что все базы матроида состоят из одинакового количества элементов.

Если  $E$  - множество строк некоторой матрицы, а  $\Phi$  состоит из всех линейно независимых множеств строк этой матрицы, то пара  $(E, \Phi)$  образует *матроид*, называемый *матричным матроидом*.

Другим важным типом матроидов являются *графовые матроиды*. Пусть  $G = (V, E)$  - обычный *граф*. Подмножество множества  $E$  назовем *ациклическим*, если подграф, образованный ребрами этого подмножества, не содержит циклов, т.е. является лесом.

**Теорема 1.** Если  $G = (V, E)$  - обычный *граф*,  $\Phi$  - семейство всех ациклических подмножеств множества  $E$ , то пара  $M_G = (E, \Phi)$  является *матроидом*.

**Доказательство.** Аксиома (1) выполняется, так как всякое подмножество ациклического множества, очевидно, является ациклическим. Докажем, что выполняется и (2). Пусть  $X$  и  $Y$  - два ациклических множества и  $|X| < |Y|$ . Допустим, что не существует такого ребра  $e \in Y$ , что множество  $X \cup \{e\}$  является ациклическим. Тогда при добавлении любого ребра из множества  $Y$  к множеству  $X$  образуется цикл. Это означает, что концы каждого такого ребра принадлежат одной компоненте связности остовного подграфа, образованного ребрами множества  $Y$ . Тогда каждая область связности подграфа  $X$  содержится в какой-нибудь области связности подграфа  $Y$ . Но компоненты связности каждого из этих подграфов - деревья, а дерево с  $k$  вершинами содержит ровно  $k - 1$  ребер. Следовательно, в этом случае число ребер в  $Y$  не превосходило бы числа ребер в  $X$ , что противоречит условию  $|X| < |Y|$ .

Базами *графового матроида* являются все *каркасы графа*. Для связного графа это будут все его *остовные деревья*.

### Теорема Радо-Эдмондса

Рассмотрим общий тип оптимизационных задач, формулируемых следующим образом. Дано произвольное конечное множество  $E$  и некоторое семейство  $\Phi$  его подмножеств. Для каждого элемента  $x \in E$  задан его *вес* - положительное число  $w(x)$ . Весмножество  $X \subseteq E$  определяется как сумма весов его элементов. Требуется найти множество наибольшего веса, принадлежащее  $\Phi$ . В эту схему укладываются многие известные задачи,

например задача о независимом множестве графа ( $E$  - множество вершин графа, вес каждой вершины равен 1, а  $\Phi$  состоит из всех независимых множеств) или задача о паросочетании ( $E$  - множество ребер графа, вес каждого ребра равен 1,  $\Phi$  состоит из всех паросочетаний). В задаче об оптимальном каркасе, рассмотренной в "предыдущей лекции", требуется найти каркас минимального, а не максимального веса, ее можно свести к задаче на максимум. Действительно, пусть  $w$  - заданная весовая функция на множестве ребер графа и требуется найти каркас минимального веса. Пусть  $a$  - число, большее, чем веса всех ребер в данном графе. Рассмотрим новую весовую функцию  $w'$ , полагая что  $w'(e) = a - w(e)$  для каждого ребра  $e$ . Очевидно, что ациклическое множество наибольшего веса относительно весовой функции  $w'$  будет каркасом наименьшего веса относительно весовой функции  $w$  и обратно. Таким образом, задача об оптимальном каркасе эквивалентна задаче построения ациклического множества максимального веса. Последнюю будем называть задачей об оптимальном каркасе на максимум. Она тоже является задачей рассматриваемого общего типа:  $E$  - множество ребер графа,  $\Phi$  состоит из всех ациклических множеств. К этой задаче применимы рассмотренные в "предыдущей лекции" алгоритмы Прима и Крускала, если в первом из них на каждом шаге выбирать ребро не минимального, а максимального веса, а во втором упорядочивать ребра не по возрастанию, а по убыванию весов. Модифицированный таким образом алгоритм Крускала будем называть максимизирующим алгоритмом Крускала.

Сформулируем теперь жадный алгоритм для решения этой общей задачи. Чтобы отличать этот алгоритм от других жадных алгоритмов, назовем его СПО (Сортировка и Последовательный Отбор).

### Алгоритм 1. Алгоритм СПО

1. Упорядочить элементы множества  $E$  по убыванию весов:  $E = \{e_1, e_2 \dots e_m\}$ ,  $w(e_1) \geq w(e_2) \geq \dots \geq w(e_m)$ .
2.  $A := \emptyset$
3. **for**  $i := 1$  **to**  $m$  **do**
4. **if**  $A \cup \{e_i\} \in \Phi$  **then**  $A := A \cup \{e_i\}$

Максимизирующий алгоритм Крускала является примером алгоритма этого типа. Уместен вопрос: каким условиям должно удовлетворять семейство  $\Phi$  для того, чтобы при любой весовой функции  $w$  алгоритм СПО находил оптимальное решение? Исчерпывающий ответ дает следующая теорема Радо-Эдмондса.

**Теорема 2.** Если  $M = (E, \Phi)$  - матроид, то для любой весовой функции  $w: E \rightarrow R^+$  множество  $A$ , найденное алгоритмом СПО, будет множеством наибольшего веса из  $\Phi$ . Если же  $M = (E, \Phi)$  не

является матроидом, то найдется такая функция  $w: E \rightarrow R^+$ , что  $A$  не будет множеством наибольшего веса из  $\Phi$ .

**Доказательство.** Предположим, что  $M = (E, \Phi)$  является матроидом и  $A = \{a_1, a_2 \dots a_n\}$  - множество, построенное алгоритмом СПО, причем  $w(a_1) \geq w(a_2) \geq \dots \geq w(a_n)$ . Очевидно,  $A$  является базой матроида. Пусть  $B = \{b_1, b_2 \dots b_k\}$  - любое другое независимое множество и  $w(b_1) \geq w(b_2) \geq \dots \geq w(b_k)$ . Так как  $A$  - база, то  $k \leq n$ . Покажем, что  $w(a_i) \geq w(b_i)$  для каждого  $i \in \{1 \dots k\}$ . Действительно, положим  $X = \{a_1 \dots a_{i-1}\}$ ,  $Y = \{b_1 \dots b_{i-1}, b_i\}$  для некоторого  $i$ . Согласно условию (2) определения матроида, в множестве  $Y$  имеется такой элемент  $b_j$ , что  $b_j \notin X$  и множество  $X \cup \{b_j\}$  - независимое. В соответствии с алгоритмом, элементом наибольшего веса, который может быть добавлен к  $X$  так, чтобы получилось независимое множество, является  $a_i$ . Следовательно,  $w(a_i) \geq w(b_j) \geq w(b_i)$ .

Теперь предположим, что  $M = (E, \Phi)$  не является матроидом. Допустим сначала, что нарушается условие (1), т.е. существуют такие подмножества  $X$  и  $Y$  множества  $E$ , что  $X \in \Phi$ ,  $Y \subset X$  и  $Y \notin \Phi$ . Определим функцию  $w$  следующим образом:

$$w(x) = \begin{cases} 1, & \text{если } x \in Y, \\ 0, & \text{если } x \notin Y. \end{cases}$$

Алгоритм СПО сначала будет рассматривать все элементы множества  $Y$ . Так как  $Y \notin \Phi$ , то не все они войдут в построенное алгоритмом множество  $A$ . Следовательно,  $w(A) < |Y|$ . В то же время имеется множество  $X \in \Phi$ , такое, что  $w(X) = |Y|$ . Таким образом, в этом случае алгоритм СПО строит не оптимальное множество. Если же условие (1) выполнено, а не выполняется условие (2), то существуют такие подмножества  $X$  и  $Y$  множества  $E$ , что  $X \in \Phi$ ,  $Y \in \Phi$ ,  $|X| < |Y|$  и  $X \cup \{x\} \notin \Phi$  для каждого  $x \in Y$ . Выберем такое  $\varepsilon$ , что  $0 < \varepsilon < \frac{|Y|}{|X|} - 1$ , и определим функцию  $w$  следующим образом:

$$w(x) = \begin{cases} 1 + \varepsilon, & \text{если } x \in X, \\ 1, & \text{если } x \in Y - X, \\ 0, & \text{если } x \notin X \cup Y. \end{cases}$$

*Алгоритм СПО* сначала выберет все элементы множества  $X$ , а затем отвергнет все элементы из  $Y - X$ . В результате будет построено множество  $A$  с весом  $w(A) = (1 + \varepsilon)|X| < |Y|$ , которое не является оптимальным, так как  $W(Y) = |Y|$ .

Максимизирующий алгоритм Крускала - это алгоритм СПО, применяемый к семейству ациклических множеств ребер графа. Из теорем 1 и 2 следует, что он действительно решает задачу об *оптимальном каркасе*. В то же время существует много жадных алгоритмов, не являющихся алгоритмами типа СПО. Примером может служить алгоритм Прима. Эти алгоритмы не попадают под действие теоремы Радо-Эдмондса, для их обоснования нужна иная аргументация.

Если для некоторой конкретной задачи удалось установить применимость к ней алгоритма СПО, это не значит, что все проблемы позади.

### Взвешенные паросочетания

Рассмотрим следующую задачу. Дан двудольный граф  $G = (A, B, E)$  и для каждой вершины  $x \in A$  задан положительный вес  $w(x)$ . Требуется найти такое паросочетание в этом графе, чтобы сумма весов вершин из доли  $A$ , инцидентных ребрам паросочетания, была максимальной. Эту задачу иногда интерпретируют следующим образом.  $A$  - это множество *работ*, а  $B$  - множество *работников*. Ребро в графе  $G$  соединяет вершину  $a \in A$  с вершиной  $b \in B$ , если квалификация работника  $b$  позволяет ему выполнить работу  $a$ . Каждая работа выполняется одним работником. Выполнение работы  $a$  принесет прибыль  $w(a)$ . Требуется так распределить обязанности работников, чтобы максимизировать общую прибыль. Покажем, что эта задача может быть решена алгоритмом СПО в сочетании с методом чередующихся цепей.

Множество  $X \subseteq A$  назовем *отображаемым*, если в графе  $G$  существует паросочетание  $M$ , насыщающее все вершины из  $X$ .  $M$  в этом случае будем называть *отображением* для  $X$ . Пусть  $\Phi$  - семейство всех отображаемых множеств.

**Теорема 3.** Пара  $(A, \Phi)$  является матроидом.

**Доказательство.** Условие (1) определения матроида, очевидно, выполняется. Докажем, что выполняется и условие (2). Пусть  $X \in \Phi$ ,  $Y \in \Phi$ ,  $|X| < |Y|$ . Рассмотрим подграф  $H$  графа  $G$ , порожденный всеми вершинами из  $X \cup Y$  и всеми смежными с ними вершинами из доли  $B$ . Пусть  $M_X$  - отображение для  $X$ . Так как  $M_X$  не является наибольшим паросочетанием в графе  $H$ , то по теореме 6 относительно него в этом графе существует увеличивающая цепь. Одним из концов этой цепи является свободная относительно  $M_X$  вершина  $a \in Y$ . После

увеличения паросочетания  $M_X$  с использованием этой цепи, как было описано выше, получим паросочетание  $M'$ , отображающее множество  $X \cup \{a\}$ . Следовательно,  $X \cup \{a\} \in \Phi$ .

Даже если бы в задаче требовалось найти только отображаемое множество наибольшего веса, проверка принадлежности множества  $\Phi$  требовала бы и нахождения соответствующего отображения, т.е. паросочетания. На самом же деле построение паросочетания входит в условие задачи. Комбинируя СПО с алгоритмом поиска увеличивающих цепей, получаем следующий алгоритм.

**Алгоритм 2.** Построение паросочетания наибольшего веса в двудольном графе  $G = (A, B, E)$  с заданными весами вершин доли  $A$ .

1. Упорядочить элементы множества  $A$  по убыванию весов:  $A = \{a_1, a_2, \dots, a_k\}$ ,  $w(a_1) \geq w(a_2) \geq \dots \geq w(a_k)$
2.  $X := \emptyset$
3.  $M := \emptyset$
4. **for**  $i := 1$  **to**  $k$  **do**
5. **if** в  $G$  существует увеличивающая цепь  $P$  относительно  $M$ , начинающаяся в вершине  $a_i$
6. **then**  $X := X \cup \{a_i\}$ ;  $M := M \otimes P$

Если для поиска увеличивающей цепи применить метод поиска в ширину, как описано выше, то время поиска будет пропорционально числу ребер. Общая трудоемкость алгоритма будет  $O(mk)$ , где  $k$  - число ребер в доле  $A$ .

## 5. Порядок выполнения работы

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## 6. Форма отчета о работе

Лабораторная работа № \_\_\_\_\_

Номер учебной группы \_\_\_\_\_

Фамилия, инициалы учащегося \_\_\_\_\_

Дата выполнения работы \_\_\_\_\_

Тема работы: \_\_\_\_\_

Цель работы: \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Изложите принципы структурного подхода к разработке программного обеспечения.
2. Охарактеризуйте методологию IDEF0-моделирования.
3. Что такое функциональная декомпозиция?
4. Что такое восходящее и нисходящее проектирование?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Котов В.М., Методы алгоритмизации / И.А. Волков, Харитонович А.И. – Мн.: Нар.асвета, 1996. –127 с.
3. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400
4. Хьюз Дж. Структурный подход к программированию / Мичтом Дж. – М.: Мир, 1980. –278 с.

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет  
информатики и радиоэлектроники»  
Филиал  
«Минский радиотехнический колледж»

Учебная дисциплина «Структуры и алгоритмы обработки данных»

**Инструкция**  
по выполнению лабораторной работы  
«Разработка, отладка и испытание алгоритмов и программ генерирования  
случайных чисел»

Минск  
2017

## **Лабораторная работа № 23**

**Тема работы:** «Разработка, отладка и испытание алгоритмов и программ генерирования случайных чисел»

### **1. Цель работы**

Закрепление навыка реализации алгоритмов генерирования случайных чисел.

### **2. Задание**

Номер варианта соответствует номеру по списку.

1. Используя генератор случайных чисел, смоделировать процесс заполнения зрительного зала кинотеатра на 300 мест. Зрительный зал имеет вид прямоугольника  $N L; x M$ .
2. Используя генератор случайных чисел, смоделировать процесс парковки автомобилей коммерческой фирмы у тротуара. Длина тротуара  $D$ , длина автомобиля  $N$ .
3. Используя метод середины квадрата, получить 10 случайных чисел.
4. Используя линейный конгруэнтный метод, получить 10 случайных чисел.
5. Используя полярный метод, получить 10 случайных чисел.

### **3. Оснащение работы**

Задание по варианту, ЭВМ, среда разработки **Dev C++** или **Bolder C++**.

### **4. Основные теоретические сведения**

Многие явления в природе, технике, экономике и других областях носят случайный характер. В этом случае величина, принимающая свои значения в зависимости от исходов некоторого испытания (опыта), называется случайной величиной.

Пусть  $X$  - дискретная случайная величина, возможными значениями которой являются числа  $x_1, x_2, \dots, x_n$ .

Обозначим через  $p_i = P(X = x_i)$  (где  $i = 1, 2, \dots, n$ ), т.е.  $p_i$  - вероятность события, состоящего в том, что  $X$  принимает значение  $x_i$ .

Закон распределения полностью задает дискретную случайную величину. Однако часто закон распределения случайной величины неизвестен. В таких ситуациях ее описывают числовыми характеристиками.

#### **Числовые характеристики случайных величин**

Случайные величины характеризуются следующими числовыми параметрами:

- ✓ *математическое ожидание* – статистическое среднее случайной величины:

$$M(X) = \sum_{i=1}^n x_i p_i$$

где  $x_i$  - значение случайной величины;  $p_i$  - вероятность появления этой величины;

сумма вероятностей равна единице:

$$\sum_{i=1}^n p_i = 1$$

для равновероятных событий математическое ожидание равно:

$$M(X) = \frac{\sum_{i=1}^n x_i}{n}$$

- ✓ *дисперсия* – математическое ожидание квадрата отклонения случайной величины от ее математического ожидания:

$$D(x) = \sum_{i=1}^n (x_i - M(x))^2 \cdot p_i$$

- ✓ корень квадратный из дисперсии - *среднее квадратичное отклонение* случайной величины:

$$\sigma(x) = \sqrt{D(x)}$$

- ✓ *коэффициент корреляции* определяет зависимость между потоками случайных величин:

$$R(x, y) = \frac{M(x \cdot y) - M(x) \cdot M(y)}{\sigma(x) \cdot \sigma(y)}$$

Потоки случайных величин, для которых  $R(x, y) = 0$ , называются *некоррелированными (независимыми)*. При этом

$$-1 \leq R(x, y) \leq 1$$

Рассмотрим алгоритмы для детерминированной выборки случайных чисел.

### **Метод середины квадрата**

Один из ранних генераторов случайных чисел, принадлежащий Джону фон Нейману (1946), известен как *метод середины квадрата*.

Метод используется для генерации  $k$ -разрядных псевдослучайных чисел. Должно быть задано  $k$ -разрядное начальное число  $x_0$  (для удобства предполагаем, что  $k$  четно). Это число возводится в квадрат и получается число  $y$ . Число  $y$  должно иметь  $2k$  разрядов. Если число разрядов меньше  $2k$ , то число

$y$  дополняется слева нулями. Затем в  $y$  выделяют средние  $k$  разрядов, которые и дают новое случайное число.

Алгоритм сводится к выполнению следующих шагов.

**Шаг 0** [Инициализация]:  $x := x_0$ .

**Шаг 1** [Основной цикл]:  $FOR j := 1 \ TO \ m \ DO \ шаг 2; Stop$ .

**Шаг 2** [Генерация нового случайного числа  $x_j$ ]:

$y := x^2; \ x_j := (\text{средние } k \text{ разрядов числа } y); \ x := x_j$ .

(Число  $y$  будет иметь  $2k$  разрядов, а следующее случайное число  $x_j$  получается, если удалить по  $k/2$  разрядов с каждого конца  $y$ . Десятичная точка помещается перед первым разрядом числа  $x_j$  до поступления его на выход случайного генератора.)

**Пример 23.1.** Получить три случайных числа методом середины квадрата.  $k = 4; x_0 = 3167; y = 10029889$ .

$$x_1 = 0298; \quad y = 000\cancel{8}8804;$$

$$x_2 = 0888; \quad y = 00\cancel{7}88544;$$

$$x_3 = 7885.$$

**Пример 23.2.** Для  $k = 4$  и  $x_0 = 2134$  получить первые восемь псевдослучайных чисел, выработанных алгоритмом:

$x_0^2 = 04\ 5539\ 56$	$x_1 = 0,5539$
$x_1^2 = 30\ 6805\ 21$	$x_2 = 0,6805$
$x_2^2 = 46\ 3080\ 25$	$x_3 = 0,3080$
$x_3^2 = 09\ 4864\ 00$	$x_4 = 0,4864$
$x_4^2 = 23\ 6584\ 96$	$x_5 = 0,6584$
$x_5^2 = 43\ 3490\ 56$	$x_6 = 0,3490$
$x_6^2 = 12\ 1801\ 00$	$x_7 = 0,1801$
$x_7^2 = 03\ 2436\ 01$	$x_8 = 0,2436$

Несмотря на видимую случайность чисел, генерируемых алгоритмом, ему свойственны недостатки. В самом деле, если в последовательности когда-нибудь появится число 0,0000, то все следующие за ним числа будут также равны 0,0000.

Таким образом, многое зависит от начального выбора  $k$  и  $x_0$ .

### Линейный конгруэнтный метод

Метод используется для генерации последовательности  $x_1, x_2, \dots, x_m$  из  $m$  псевдослучайных чисел. Должны быть заданы следующие входные значения:

- ✓  $b$  - целочисленный множитель,  $b \geq 1$ ;
- ✓  $x_0$  - начальное случайное целое число,  $x_0 \geq 1$ ;

- ✓  $k$  - шаг;  $k \geq 0$  - целое;
- ✓  $m$  - целочисленный модуль,  $m \geq x_0, b, k$ .

Случайные числа генерируются по рекуррентной формуле:

$$x_j = (b \cdot x_{j-1} + k) \bmod m.$$

Алгоритм сводится к выполнению следующих шагов.

**Шаг 1** [Основной цикл]:

*FOR*  $j := 1$  *TO*  $m$  *DO* шаг2, шаг3; *Stop*.

**Шаг 2** [Генерация нового необработанного случайного числа]:

$$x_j := (b \cdot x_{j-1} + k) \bmod m.$$

(Число  $x_j$  должно лежать в интервале  $0 \leq x_j < m$ . По определению, если  $a \bmod m = x$  для целых  $a$  и  $m$ , то  $x$  есть целый остаток от деления  $a$  на  $m$ . Например,  $5 \bmod 3 = 2$ ,  $7 \bmod 3 = 1$ ,  $9 \bmod 3 = 0$ ).

**Шаг 3** [Генерация следующего случайного числа]:

$y_j := x_j / m$ . ( $y_j$  будет лежать в интервале  $0 \leq y_j < 1$  и обладать требуемым распределением.)

**Пример 23.3.** Получить три случайных числа линейным конгруэнтным методом.  $x_0 = 27$ ;  $b = 5$ ;  $k = 10$ ;  $m = 40$ ;

$$x_0 = 27; b = 5; k = 10; m = 40;$$

$$x_1 = (5 \cdot 27 + 10) \bmod 40 = 145 \bmod 40 = 25;$$

$$x_2 = (5 \cdot 25 + 10) \bmod 40 = 15;$$

$$x_3 = (5 \cdot 15 + 10) \bmod 40 = 5.$$

**Пример 23.4** Для  $k=0$ ,  $m = 2^{10}$ ,  $b = 101$ ,  $x_0 = 432$  получить первые восемь псевдослучайных чисел, выбранных алгоритмом:

$x_1 = 624$	$y_1 = 624/1024 = 0,610$
$x_2 = 560$	$y_2 = 560/1024 = 0,546$
$x_3 = 240$	$y_3 = 240/1024 = 0,234$
$x_4 = 688$	$y_4 = 688/1023 = 0,673$
$x_5 = 880$	$y_5 = 880/1024 = 0,859$
$x_6 = 816$	$y_6 = 816/1024 = 0,796$
$x_7 = 496$	$y_7 = 496/1024 = 0,486$
$x_8 = 944$	$y_8 = 944/1024 = 0,923$

Существует такой выбор параметров  $k$ ,  $m$ ,  $b$ ,  $x_0$  при котором алгоритм будет генерировать числа на отрезке  $[0; 1]$ , представляющиеся непредсказуемыми и удовлетворяющие определенным статистическим критериям. Для всех практических целей эти числа оказываются

последовательностью наблюдений равномерно распределенной случайной переменной.

### **Полярный метод генерации случайных чисел с нормальным распределением**

Метод используется для генерации двух независимых случайных чисел с распределением  $N(0,1)$  из двух независимых равномерно распределенных случайных чисел. Распределение  $N(0,1)$  преобразуется в  $N(\mu, \sigma_2)$  с использованием центральной предельной теоремы.

Алгоритм сводится к выполнению следующих шагов.

#### **Шаг 1** [Генерация двух равномерно распределенных случайных чисел].

Генерируются два независимых случайных числа  $u_1$  и  $u_2$  с распределением  $U(0,1)$ :

$$v_1 := 2u_1 - 1; v_2 := 2u_2 - 1$$

( $v_1$  и  $v_2$  имеют распределение  $U(-1, +1)$ ).

#### **Шаг 2** [Вычисление и проверка $s$ ]:

$$s := v_1^2 + v_2^2; \text{ IF } s \geq 1 \text{ THEN GOTO шаг 1.}$$

#### **Шаг 3** [Вычисление $n_1$ и $n_2$ ]:

$$n_1 := v_1 \sqrt{(-2 \ln s) / s}; \quad n_2 := v_2 \sqrt{(-2 \ln s) / s};$$

Stop ( $n_1$  и  $n_2$  распределены нормально).

Алгоритм имеет вычислительное преимущество перед другими методами: он обычно генерирует по одному числу с нормальным распределением на каждое число с равномерным распределением. Правда, при этом следует убедиться, что компенсируется дополнительное время, затрачиваемое на вычисление натуральных логарифмов и квадратных корней.

## **5. Порядок выполнения работы**

1. выделить необходимые структуры данных для решения данной задачи;
2. построить алгоритм решения данной задачи;
3. запрограммировать полученный алгоритм;
4. провести тестирование полученной программы.

## **6. Форма отчета о работе**

*Лабораторная работа №* \_\_\_\_\_

*Номер учебной группы* \_\_\_\_\_

*Фамилия, инициалы учащегося* \_\_\_\_\_

*Дата выполнения работы* \_\_\_\_\_

*Тема работы:* \_\_\_\_\_

*Цель работы:* \_\_\_\_\_

*Оснащение работы:* \_\_\_\_\_

*Результат выполнения работы:* \_\_\_\_\_  
\_\_\_\_\_

## **7. Контрольные вопросы и задания**

1. Какие типы числовых характеристик случайных величин существуют?
2. Что показывает коэффициент корреляции?
3. В чем суть метода середины квадрата?
4. Какие исходные данные используются в линейном конгруэнтном методе?
5. Какие случайные числа получают полярным методом?

## **8. Рекомендуемая литература**

1. Ахо А. Построение и анализ вычислительных алгоритмов / Хопкрофт Дж., Ульман Дж. - М.: Мир, 1979. – 325 с.
2. Вирт Н. Алгоритмы и структуры данных – М.: ДМК Пресс, 2016. – 272 с.
3. Окулов С.М. Программирование в алгоритмах – М.: БИНОМ. Лаборатория знаний, 2006. - 383 с.
4. Хаггарти Р. Дискретная математика для программистов – М.: Техносфера, 2016. - 400