А.В. Лямин, Е.Н. Череповская

ЯЗЫКИ ПРОГРАММИРОВАНИЯ C/C++ КОМПЬЮТЕРНЫЙ ПРАКТИКУМ

Санкт-Петербург
2017

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

УНИВЕРСИТЕТ ИТМО

А.В. Лямин, Е.Н. Череповская

ЯЗЫКИ ПРОГРАММИРОВАНИЯ C/C++ КОМПЬЮТЕРНЫЙ ПРАКТИКУМ

РЕКОМЕНДОВАНО К ИСПОЛЬЗОВАНИЮ В УНИВЕРСИТЕТЕ ИТМО
по направлениям подготовки 11.03.02 «Инфокоммуникационные технологии и системы связи», 09.03.02 «Информационные системы и технологии» и другим информационным направлениям в качестве учебно-методического пособия для реализации основных профессиональных образовательных программ бакалавриата

Санкт-Петербург

2017

Лямин А.В., Череповская Е.Н. Языки программирования C/C++. Компьютерный практикум. – СПб: Университет ИТМО, 2017. – 71 с.

Рецензенты:
Чежин М.С., к.т.н., доцент
Меженин А.В., к.т.н., доцент

Компьютерный практикум содержит краткие справочные материалы и задания для изучения языков программирования C/C++, включая операторы ветвления и циклов, форматированный ввод и вывод данных, работу со структурами данных, функции и работу с файлами. Закрепление полученных навыков осуществляется на задачах разного уровня сложности, в том числе из области вычислительной математики. Предназначается для студентов бакалавриата, обучающихся по направлениям 11.03.02 «Инфокоммуникационные технологии и системы связи», 09.03.02 «Информационные системы и технологии» и другим информационным направлениям.

УНИВЕРСИТЕТ ИТМО

Университет ИТМО – ведущий вуз России в области информационных и фотонных технологий, один из немногих российских вузов, получивших в 2009 году статус национального исследовательского университета. С 2013 года Университет ИТМО – участник программы повышения конкурентоспособности российских университетов среди ведущих мировых научно-образовательных центров, известной как проект «5 в 100». Цель Университета ИТМО – становление исследовательского университета мирового уровня, предпринимательского по типу, ориентированного на интернационализацию всех направлений деятельности.

Содержание

Лабораторная работа № 1: Среда разработки

1. Запустите онлайн среду разработки "Tutorials Point" с рабочим пространством для языка программирования Си по ссылке: http://www.tutorialspoint.com/compile_c_online.php. Интерфейс среды разработки состоит из трех областей: с левой стороны располагается каталог текущих файлов проекта, с правой стороны – редактор кода программы, под редактором кода находится терминал для ввода команд. В ходе выполнения лабораторных работ Вам нужно будет создавать файлы программ и запускать их путем ввода команд в терминале. Для сохранения файлов проекта можно использовать внешний носитель ("Project" → "Download project") или диск Google ("Project" → "Save project" → "Save on Google Drive").

2. Создайте программу **hello-c-world.c** по следующему шаблону:

```
#include <stdio.h>
int main() {
printf("Hello World!\n");
return 0;
}
```

Первая строка программы (**#include <stdio.h>**) служит для подключения базовой библиотеки для ввода-вывода информации в терминал, использующейся в языке программирования Си – **stdio.h**. Все выполняемые команды должны располагаться в специальных блоках – функциях. При этом программа обязательно должна содержать функцию **main(){}**, которая будет запущена по умолчанию при запуске программы. Ключевое слово **int** означает, что данная функция должна возвращать значение целочисленного типа. Для простых программ чаще всего в конце функции, имеющей тип **int**, возвращаемым значением ставится 0 после ключевого слова **return**, служащем для возвращения значения. Функция **printf()** служит для непосредственного вывода информации в терминал (консоль, командную строку).

3. Скомпилируйте и запустите программу, поочередно введя в терминале следующие команды:

```
gcc hello-c-world.c -o "helloWorld"
./helloWorld
```

В терминале будет выведено сообщение "Hello World!".

Компиляция и запуск программ

Перед запуском программ необходимо их скомпилировать, то есть перевести на машинно-ориентированный язык, близкий машинному коду, который будет понятен компьютеру. Эту функцию выполняют компиляторы, осуществляющие компиляцию программ, т.е. их трансляцию с исходного языка на машинно-ориентированный. Созданное рабочее пространство поддерживает компилятор GCC (GNU Compiler Collection) для языков программирования Си и C++. Команда **gcc** запускает компиляцию файла программы на языке Си (**hello-c-world.c**) в исполняемый файл. Задание имени исполняемого файла происходит с помощью использования параметра "**-o**", т.е. в приведенном примере именем исполняемого файла является **helloWorld**. Поскольку наш файл лежит напрямую в корневой папке проекта, запуск программы осуществляется из текущего каталога командой **./helloWorld**.

4. Создайте новую программу **lab_01_01.c** по следующему шаблону (комментарии можно опустить):

```
#include <stdio.h>
/*
   многострочный
   комментарий
*/
int main() {
    // комментарий до конца строки
int x = 13;
int y = 4;
printf("x + y = %d \n", x + y); // сложение целых чисел
printf("x - y = %d \n", x - y); // вычитание целых чисел
printf("x * y = %d \n", x * y); // умножение целых чисел
printf("x %% y = %d \n", x % y); // остаток от деления
целых чисел
printf("x / y = %d \n", x / y); // деление целых чисел
double a = 11.7,
       b = 3.5;
printf("a / b = %lf \n", a / b); // деление вещ. чисел
printf("x + a = %lf \n", x + a); // сложение целых и вещ.
чисел
char c1 = 10,
     c2 = 100;
printf("c1 + c2 = %d \n", c1 + c2);
// пример взаимозаменяемых операторов
     int i = 15;
     printf("i = %d \n", i);
```

```
     i = i + 1;
     printf("i=i+1 : %d \n", i);
     i++;
     printf("i++ : %d \n", i);
     i += 1;
     printf("i+=1 : %d \n", i);
// экспоненциальный формат
double d = -5.438754e3,
       f = 1.3e-7;
int dInt = (int) d;
printf("d= %lf \n", d);
     printf("dInt = %d \n", dInt);
    return 0;
}
```

Сохраните файл. Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией.

Простые типы данных

В языке программирования Си используются следующие типы данных:

Название	Размер в байтах	Значение
`char`	1	от -128 до 127
`signed char`	1	от -128 до 127
`unsigned char`	1	от 0 до 255
`int`	2	от -32 768 до 32 767
`signed int`	2	от -32 768 до 32 767
`unsigned int`	2	от 0 до 65535
`short int`	2	от -32 768 до 32 767
`signed short int`	2	от -32 768 до 32 767
`unsigned short int`	2	от 0 до 65535
`long int`	4	от -2147483648 до 2147483647
`signed long int`	4	от -2147483648 до 2147483647
`unsigned long int`	4	от 0 до 4294967295
`float`	4	от $3.4e^{-38}$ до $3.4e^{+38}$
`double`	8	от $1.7e^{-308}$ до $1.7e^{+308}$
`long double`	10	от $3.4e^{-4932}$ до $3.4e^{+4932}$

Переменные символьного типа **char** предназначены для хранения символьных данных. Строки в Си представляются в виде массива символов и объявляются как **char a[]**, где в квадратных скобках указывается количество символов, содержащихся в строке (длина строки). Если значение строковой переменной указывается сразу при ее создании, указание длины строки в скобках можно опустить. Вещественный тип **float** и вещественный тип двойной точности **double** и их расширения предназначены для хранения значений с плавающей точкой. Для объявления константы перед типом переменной необходимо указать ключевое слово **const**, запрещающее возможное переопределение значения переменной.

Для числовых типов данных поддерживаются следующие операции:

- **+** (сложение)
- **-** (вычитание)
- ***** (умножение)
- **/** (деление)
- **%** (остаток от деления)

Одним из основных операторов является оператор присваивания: **=**. Для присваивания переменной результата выполнения над ней математических операций можно использовать сокращенные операции: **+=**, **-=**, ***=**, **/=**, **%=**. Например, результат выражения **a = a + b** будет полностью идентичен результату выражения **a += b**. Также для прибавления к значению переменной единицы или вычитания из него единицы возможно использование унарных операторов **++** и **--**. Например, результат выражения **a = a + 1** будет полностью идентичен результату выражения **a++**.

Язык Си поддерживает запись чисел в экспоненциальном формате, например, **5e8** или **3.5e-2**. Для них доступны те же операции, что и для других вещественных чисел.

При необходимости привести целый тип к вещественному, вещественный к целому и т.п. используется приведение типов. Для этого перед значением необходимо в круглых скобках указать тип, к которому нужно его привести:

```
int main() {
    int a=5, b=3;
    double c = (double) a/b;
    return 0;
}
```

5. Модифицируйте программу **lab_01_01.c**, изменив значения переменных **x**, **y**, **a** и **b**. Ознакомьтесь с результатом.

6. Модифицируйте программу **lab_01_01.c**, изменив значение переменной **c2** на 200. Ознакомьтесь с результатом и подумайте, почему так произошло.

7. Модифицируйте программу **lab_01_01.c**, изменив тип переменной **c2** на **unsigned char**, оставив значение переменной равным 200. Ознакомьтесь с результатом и объясните, почему так произошло.

8. Дополните код программы **lab_01_01.c**. Вычтите из переменной **i** единицу, используя все возможные формы записи выражения. Ознакомьтесь с результатом.

9. Дополните код программы **lab_01_01.c**. Создайте целочисленные переменные **q** со значением 100 и **w** со значением 34, и вещественные переменные двойной точности **res1** и **res2**. Запишите результат деления переменной **q** на переменную **w** без приведения типов в **res1** и с использованием приведения типов в **res2**. Ознакомьтесь с результатом.

Форматированный вывод

Функция **printf()** служит для вывода информации в консоль. При вставке переменной определенного типа в строку вывода необходимо учитывать различные виды команд вывода (спецификаторов) для разных типов, в частности:

- **%i** или **%d** для типа **int**
- **%ld** для типа **long**
- **%c** для типа **char**
- **%f** для типа **float**
- **%lf** для типа **double**
- **%s** для строковых данных

Для строкового аргумента функции **printf()** знак ‘**%**’ является специальным символом. Для стандартного вывода знака ‘**%**’ для значения процентов в терминал используется форма записи **printf("%%")**. ‘**\n**’ используется при выводе информации для обозначения перевода строки. При попытке вывода переменной символьного типа с форматированием как целочисленной, будет выведен код символа, содержащегося в переменной.

При выводе целочисленных и вещественных числовых значений возможно

изменение базового спецификатора с целью округления выводимого числа:

- **%d** – вывод числа типа **int** без форматирования
- **%3d** – вывод числа типа **int** шириной как минимум 3 знака
- **%f** – вывод числа типа **float** без форматирования
- **%4f** – вывод числа типа **float** шириной как минимум 4 знака
- **%.5f** – вывод числа типа **float** с точностью до 5 знака после запятой
- **%3.2f** – вывод числа типа **float** шириной как минимум 3 знака с точностью до 2 знака после запятой

Пример наиболее часто используемых форматов округления:

Команда	Результат
`printf("%d",101)`	`101`
`printf("%4d",10)`	`  10`
`printf("%f",5.7)`	`5.7`
`printf("%.1f",10.87)`	`10.9`
`printf("%3.2f",3.14159)`	`3.14`

10. Создайте новую программу **lab_01_02.c** по следующему шаблону (комментарии можно опустить):

```
#include <stdio.h>
int main() {
char c1 = 110,
     c2 = 't';
printf("c1 = %d \n", c1);
printf("c1 = %c \n", c1);
printf("c2 = %d \n", c2);
const int coef = 2016;
double a = 1005.23125478965,
       b = 5.347;
printf("%8d \n", coef);
     printf("%10.2lf \n", b);
    return 0;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с результатом выполнения программы.

11. Дополните код программы **lab_01_02.c**. Осуществите вывод в терминал значений переменных **a** и **b**, подставив вместо имен переменных их

значения, в формате: "A = **a**; B = **b**". При этом значение **a** должно быть выведено с точностью до 7 знаков после запятой, а значение **b** с точностью до 3 знаков после запятой.

12. Дополните код программы **lab_01_02.c**. Осуществите вывод в терминал значения переменной **a** с шириной не менее 5 знаков с точностью до 7 знаков после запятой.

13. Дополните код программы **lab_01_02.c**. Осуществите вывод в терминал значения переменной **b** с шириной не менее 7 знаков с точностью до 1 знака после запятой.

14. Дополните код программы **lab_01_02.c**. Осуществите вывод в терминал значения переменной **coef** с шириной не менее 7 знаков.

Лабораторная работа № 2: Операторы ветвления и циклы

1. Создайте новую программу **lab_02_01.c** по следующему шаблону:

```
#include <stdio.h>
int main(){
    int a = 3,
        b = 7,
        c = 21;
    if ( a == 3 ) {
        printf("Значение переменной a равно трем. \n");
    }

    if ( b > 0 && (c % a == 0) ) {
        printf("Значение переменной b положительное.
Значение переменной c кратно %d. \n", a);
    }
    else {
        printf("Значение переменной b меньше либо равно
нулю или значение переменной c не кратно %d. \n", a);
    }

    if ( b % 10 == 0 ) {
        printf("Число %d кратно 10. \n", b);
    }
    else if (b % 2 == 0) {
        printf("Число %d кратно 2. \n", b);
    }
    else {
        printf("Значение числа b не кратно 2 и 10. \n");
    }

    int place = 1;
    switch (place) {
        case 1:
            printf("Первое место! \n");
            break;
        case 2:
            printf("Второе место! \n");
            break;
        case 3:
            printf("Третье место! \n");
            break;
        default:
            printf("Вы не заняли призового места. \n");
    }
```

```
    return 0;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией.

Операторы ветвления

В языке программирования Си существует два типа операторов ветвления. Первым типом является конструкция **if** … **else if** … **else**:

```
if ( <условие> ) {
    <блок операций>
}
[else if ( <условие> ) {
    <блок операций>
}]
[else {
    <блок операций>
}]
```

В скобках **[]** обозначены необязательные блоки. В качестве проверяемого условия (**<условие>**) может выступать простое или составное логическое выражение, использующее следующие логические операции:

- == (равенство)
- != (неравенство)
- > (больше)
- < (меньше)
- >= (больше либо равно)
- <= (меньше либо равно)
- ! (отрицание)
- && (И / $\wedge$)
- || (ИЛИ / $\vee$)

<блок операций> может содержать неограниченное количество различных конструкций. В случае, когда **<блок операций>** содержит только одно выражение, фигурные скобки вокруг блока операций могут быть опущены. В языке Си нет базового логического типа. Результат проверки выполнения условия представляется в целочисленном виде, где **1** свидетельствует о выполненном условии, а **0** о ложном результате.

Вторым типом операторов ветвления является switch:

```
switch (<переменная>) {
    case <значение_1>: <блок операций_1> [break;]
    case <значение_2>: <блок операций_2> [break;]
```

```
    . . .
    default: <блок операций> [break;]
}
```

Данный тип операторов ветвления служит для проверки равенства значения переменной (**<переменная>**) одному из значений **(<значение_1>, <значение_2>**, **...**) с целью выполнения соответствующего блока операций (**<блок операций_1>**, **<блок операций_2>**, …) или же базового блока операций (**<блок операций>**), выполняющегося при переходе по базовой ветке **default**. Оператор **[break;]** не является обязательным и служит для прекращения выполнения блоков операций разных блоков **case** при соблюдении какого-либо из них.

2. Модифицируйте программу **lab_02_01.c**, изменив значения переменных **a**, **b** и **c**. Запустите программу. Ознакомьтесь с выведенной информацией.

3. Модифицируйте программу **lab_02_01.c**, последовательно изменив значение переменной **place** на **0**, **3**, **2**. Запустите программу. Ознакомьтесь с выведенной информацией.

4. Модифицируйте программу **lab_02_01.c**, убрав оператор **break** из **case** со значением **1**. Запускайте программу, последовательно изменяя значение переменной **place** на **0**, **3**, **2**. Ознакомьтесь с выведенной информацией.

5. Создайте новую программу **lab_02_02.c** по следующему шаблону:

```
#include <stdio.h>
int main(){
    int i = 0,
        j = 0;
    for (i = 0; i < 5; i++) {
        printf("Цикл for: i = %d \n", i);
    }

    i = 0;
    while ( i < 5 && j == 0 ) {
        printf("Цикл while: i = %d \n", i);
        i = i + 1;
    }

    i = 0;
    do {
```

```
        printf("Цикл do: i = %d \n", i);
        i++;
    } while ( i < 5 );
    return 0;

    int i = 0;
    for(i=0; i<200; i++) {
   if (i%2==0) continue;
   else if (i>150) break;
   else {
       printf("%d ",i);
   }
    }
    printf("\n");

    return 0;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией.

Циклы

В языке программирования Си существует три типа циклов: цикл со счетчиком, цикл с предусловием, цикл с постусловием.

Цикл со счетчиком представляется в следующем виде:

```
for (<переменная>; <условие>; <счетчик>) {
    <блок операций>
}
```

Для данного типа циклов: **<переменная>** – итерируемая переменная, для которой установлен счетчик, **<условие>** – простое или составное условие, которое должно быть выполнено для завершения цикла, **<счетчик>** – счетчик, изменяющий значение переменной, **<блок операций>** – блок операций, повторяющийся на каждой итерации цикла.

Цикл с предусловием представляется в следующем виде:

```
while (<условие>) {
    <блок операций>
}
```

Для данного типа циклов: **<условие>** – простое или составное условие, которое должно быть выполнено для завершения цикла, **<блок операций>** – блок операций, повторяющийся на каждой итерации цикла.

Цикл с постусловием представляется в следующем виде:

```
do {
    <блок операций>
} while (<условие>);
```

Для данного типа циклов: **<блок операций>** – блок операций, повторяющийся на каждой итерации цикла, **<условие>** – простое или составное условие, которое должно быть выполнено для завершения цикла.

Основное отличие цикла с постусловием от других типов циклов в том, что в цикле с постусловием **<блок операций>** обязательно выполнится хотя бы один раз.

В теле цикла (**<блок операций>**) могут быть использованы ключевые слова, позволяющие опустить выражения, указанные после ключевого слова, и перейти к следующей итерации (**continue**) или полностью остановить выполнение цикла (**break**).

6. Модифицируйте программу **lab_02_02.c**, изменив значения переменной **i** в условиях циклов. Запустите программу. Ознакомьтесь с выведенной информацией.

7. Модифицируйте программу **lab_02_02.c**, поочередно закомментировав обнуление значения переменной **i** перед циклом с предусловием и циклом с постусловием. Запустите программу. Ознакомьтесь с выведенной информацией. Почему выводимые значения циклов изменились?

8. Дополните код программы **lab_02_02.c**. С использованием цикла с предусловием на 150 итераций осуществите вывод в терминал значений итерируемой переменной, кратных трем, остановив цикл по достижении 130-й итерации.

9. Дополните код программы **lab_02_02.c**. Осуществите вывод на терминал значений логической функции, заданной формулой **A∧B∨¬C**, реализовав расчет выражения для всех возможных комбинаций **A**, **B** и **C** сначала с использованием циклов со счетчиком, затем с использованием циклов с предусловием, и, наконец, используя циклы с постусловием. Запустите программу. Убедитесь, что во всех трех случаях выведенные значения будут полностью совпадать.

Лабораторная работа № 3: Массивы и строки

1. Создайте новую программу **lab_03_01.c** по следующему шаблону:

```
#include <stdio.h>
int main() {
    // создание одномерного целочисленного массива
    int a[10];
    int i, j;
    // задание элементов массива
    a[0] = 1;
    a[1] = 2;
    a[2] = 3;
    a[3] = 4;
    a[4] = 5;
    for (i = 5; i < 10; i++) {
        a[i] = i + 1;
    }
    // задание всех элементов массива одной строкой
    int b[10] = { 10, 9, 8, 7, 6, 5, 4, 3, 2, 1 };
    // вывод элементов массива
    for (i = 0; i < 10; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
    // задание и вывод многомерного массива
    int arr[5][3];
    int c[2] = {-1, -1};
    for (i = 0; i < 5; i++) {
        for (j = 0; j < 3; j++) {
            arr[i][j] = i * j;
            // поиск позиции ненулевого элемента кратного
6
            if (arr[i][j] != 0 && arr[i][j] % 6 == 0) {
                c[0] = i;
                c[1] = j;
            }
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
    printf("Элемент находится в %d строке %d столбце \n",
c[0], c[1]);
    // получение длины массива
    long int rows = sizeof(arr)/sizeof(arr[0]);
```

```
    long int columns = sizeof(arr[0])/sizeof(arr[0][0]);
    printf("%ld \n", rows);
    printf("%ld \n", columns);
    return 0;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией.

Массивы

Массивы представляют собой последовательность значений определенного типа. Объявление массива происходит путем указания типа и имени переменной, а также длины массива (т.е. максимального количества хранимых значений). В языке программирования Си поддерживаются как одномерные, так и многомерные массивы, например:

- **`int a[10]`** – целочисленный одномерный массив **`a`**, содержащий 10 элементов с индексом ***i***;
- **`double b[2][3]`** – двумерный массив **`b`** с вещественными числами двойной точности, состоящий из 6 элементов с двумя индексами;
- **`int c[2][4][5]`** – трехмерный целочисленный массив из 40 элементов с тремя индексами ***i, j, k***.

Первый элемент массива должен иметь индекс 0, иначе в качестве значения данного элемента будет установлено значение 0. Задать значения элементов массива можно тремя способами:

- поэлементно вручную (**`a[0] = 1; a[1] = 2; a[2] = 3;`** …);
- поэлементно в цикле;
- одной строкой при объявлении (**`int a[3] = {1, 2, 3}`**).

Обращение к определенному элементу происходит указанием его индекса в массиве, например:

- **`a[3]`** – обращение к элементу одномерного массива с индексом 3,
- **`b[0][2]`** – обращение к элементу двумерного массива **`b`** с индексами 0, 2.

Вывод массивов осуществляется поэлементно в любом удобном виде.

В языке Си не предусмотрена функция получения длины массива, однако для этой цели может быть использована стандартная функция – **`sizeof(<имя_переменной>)`**. Данная функция предназначена для получения объема памяти (в байтах), занимаемого указанной переменной или типом, если вместо имени переменной указывается название типа. Таким образом, для получения количества элементов одномерного массива **`a`** необходимо разделить значение общего объема памяти, занимаемого

массивом **a**, на объем памяти, занимаемый одним его элементом: **sizeof(a)/sizeof(a[0])**.

Функция **sizeof** с параметром в виде наименования типа возвращает объем памяти, занимаемый переменной данного типа, т.е. **sizeof(int)** эквивалентно **sizeof(variable)**, где **variable** является целочисленной переменной.

2. Дополните код программы **lab_03_01.c**. Создайте одномерный целочисленный массив **d**, состоящий из 30 элементов. Заполните массив значениями в цикле по **i**, используя следующую формулу: **d[i] = d[i-1] + d[i-2]**, где **d[0]=0** и **d[1]=1**.

3. Дополните код программы **lab_03_01.c**. Для созданного на предыдущем шаге массива **d** добавьте фрагмент кода, находящего сумму элементов массива с четными индексами. Результат запишите в переменную **sum**.

4. Дополните код программы **lab_03_01.c**. Отсортируйте созданный массив **d** по убыванию. Осуществите вывод отсортированного массива.

5. Дополните код программы **lab_03_01.c**. Создайте три одномерных целочисленных массива **t1**, **t2**, **t3**, состоящих из 5 элементов. Массив **t1** заполните поэлементно вручную, массив **t2** поэлементно в цикле, а для массива **t3** укажите значения одной строкой.

6. Дополните код программы **lab_03_01.c**. Создайте трехмерный целочисленный массив **arr[2][4][5]**. Заполните массив значениями в цикле, используя следующую формулу: **arr[i][j][z] = i*j*z**. Рассчитайте длину массива по всем трем индексам.

7. Создайте новую программу **lab_03_02.c** по следующему шаблону:

```
#include <stdio.h>
#include <string.h>
int main() {
char str1[7];
str1[0] = 'W';
str1[1] = 'o';
str1[2] = 'r';
str1[3] = 'l';
str1[4] = 'd';
```

```
str1[5] = '!';
printf("%s \n", str1);
char str2[20] = "Hello, ";
printf("%s \n", str2);
char str3[7];
strcpy(str3,str1); // копирование строки
printf("%s \n", str3);
if (strcmp(str1,str3) == 0) { // сравнение строк
    printf("Строка 1 полностью совпадает со строкой 3
\n");
}
else {
    printf("Строка 1 не совпадает со строкой 3 \n");
}
strcat(str2,str1); // конкатенация строк
printf("%s \n", str2);
// получение значения длины строки
int i = 0;
for (i = 0; i <strlen(str2); i++) {
    printf("%c", str2[i]);
}
printf("\n");
return 0;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией.

Строки

Строки в языке Си представляют собой массивы символьного типа (**char**), объявление которых так же, как и для целочисленных массивов, может быть произведено различными способами:

- поэлементно (**char arr[30]; arr[0] = 'a'; a[1] = 'b';** ...),
- одной строкой (**char arr[20] = "Hello, World!"**).

Обратите внимание, что в случае, когда строка объявляется целиком, необходимо использовать двойные кавычки. Также при указании длины символьного массива необходимо учитывать, что в языке Си для определения конца строки используется символ ‘\0’, поэтому к вашему предполагаемому значению длины строки следует прибавлять единицу данного терминального символа.

Для операций над строками используют библиотеку **string.h** или

strings.h в зависимости от используемой операционной системы. Одними из основных функций данной библиотеки являются:

- Копирование строк: **strcpy(*строка_для_вставки*, *строка_для_копирования*)**.
- Сравнение строк: **strcmp(*строка_1*, *строка_2*)**.Функция сравнивает коды символов переданных в качестве аргументов строк. Возвращает 0, если строки одинаковые, положительное значение, если первая строка больше, отрицательное – если меньше.
- Конкатенация (сложение) строк: **strcat(*строка_1*, *строка_2*)**. Дописывает значение второй строки в конец первой.
- Получение длины строки: **strlen(*строка*)**.
- Функция **strncpy(*строка_для_вставки*, *строка_для_копирования, количество_байт*)** – скопирует не более чем ***количество_байт*** байт из строки для копирования в строку для вставки.
- Функция **strncmp(*строка_1*, *строка_2, количество_байт*)** – сравнит не более чем ***количество_байт*** байт строки 1 и строки 2.
- Функция **strncat(*строка_1*, *строка_2, количество_байт*)** – осуществит конкатенацию не более чем ***количество_байт*** байт строки 1 и строки 2.

8. Дополните код программы **lab_03_02.c**. Создайте строковые переменные **s1** со значением “Good morning, ” и **s2** со значением “Good evening, ”. Осуществите сравнение строк с помощью функции **strcmp**, записав результат в переменную **compared**. Выведите значение переменной **compared**. Объясните, почему был получен именно такой результат.

9. Дополните код программы **lab_03_02.c**. Создайте переменную **name**, введя в качестве ее значения Ваше имя. Осуществите конкатенацию строки **s1** со всей строкой **name**, записав результат в **s1**. Выведите значение переменной **s1**.

10. Дополните код программы **lab_03_02.c**. Осуществите конкатенацию строки **s2** с двумя байтами строки **name**, записав результат в **s2**. Выведите значение переменной **s2**.

11. Дополните код программы **lab_03_02.c**. Осуществите поиск символа **'r'** в строке **str2**. Выведите его индекс в строке на терминал.

12. Дополните код программы **lab_03_02.c**. Создайте строку **str4** с содержимым *"To be, or not to be - that is the question; Whether 'tis nobler in the*

mind to suffer; The slings and arrows of outrageous fortune; Or to take arms against a sea of troubles; And by opposing end them.". Объявите массив для хранения строк **`stringsArray`**, являющимся двумерным символьным массивом. Выделите из строки **`str4`** составляющие части, заключенные между точками или точками с запятой, заполнив ими массив **`stringsArray`**, тем самым разделив отрывок стихотворения на строки.

Лабораторная работа № 4: Ввод данных с клавиатуры

1. Создайте новую программу **lab_04_01.c** по следующему шаблону:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
    // указатели на переменные
    int value = 101;
    int *pValue = &value;
    printf("value = %d \n", value);
    printf("*pValue = %d \n", *pValue);
    printf("*(&value) = %d \n", *(&value));
    printf("value address = %p \n", &value);
    printf("pValue = %p \n", pValue);
    printf("pValue address = %p \n", &pValue);
    // указатели на массивы
    int a[6] = {9,8,7,6,5,4};
    printf("a[0] = %d \n", a[0]);
    printf("a[0] = %d \n", *a);
    printf("a[1] = %d \n", *(a+1));
    // выделение памяти
    int *arr;
    arr = malloc(1000 * sizeof(int));
    if (arr != NULL) {
        int i;
        for (i = 0; i < 100; i++) {
            arr[i] = i;
        }
    }
    free(arr);
    arr = NULL;
    // перераспределение памяти
    char *str;
    str = (char *) malloc(5 * sizeof(char));
    strcpy(str, "Hello");
    printf("str = %s; address of 'str' = %p\n", str,
str);
    str = (char *) realloc(str, 15 * sizeof(char));
    if (str != NULL) {
        strcat(str, ", World!");
        printf("str = %s; address of 'str' = %p\n", str,
str);
    }
```

```
    free(str);
    str = NULL;
    return 0;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией.

Указатели

Указатели представляют собой ссылку на часть памяти, где хранится значение, и объявляются аналогично переменным с добавлением ***** перед именем указателя. Существует два вида указателей:

- с определенным типом (**int**, **double** и т.д.), указывающие на значение данного типа,
- нетипизированные (**void**), которые могут ссылаться на значение любого из простых типов.

Использование обозначения **&** перед именем переменной позволяет получить ее адрес. Для вывода адреса переменной в шестнадцатеричном формате используется спецификатор **%p**.

Все массивы в языке Си по умолчанию могут быть адресованы с использованием указателей, например, строковые значения могут быть объявлены как в виде массива символов, так и с использованием указателя: **char *a**. Объявление массивов с использованием указателей позволяет определять динамические массивы. Указатель на массив ***array** ссылается на первый элемент массива. Для получения доступа ко второму элементу необходимо к указателю на первый элемент прибавить единицу ***(array + 1)**.

Для выделения и освобождения памяти под указатель служат стандартные функции библиотеки **<stdlib.h>**:

- Функция **malloc()** принимает на вход значение количества байт для резервирования и возвращает нетипизированный указатель на переменную. Если указанное количество памяти выделить не удалось, функция вернет значение **NULL**.
- Функция **realloc()** принимает на вход указатель, для которого ранее была выделена память, и новое значение объема памяти для данного указателя. Если указанное количество памяти выделить не удалось, функция вернет значение **NULL**.
- Функция **free()** принимает на вход имя указателя, для которого была использована операция выделения памяти, и освобождает выделенную память.

После завершения работы с указателем следует обнулить его с целью предотвращения неумышленного изменение данных: **arr = NULL**.

2. Дополните код программы **lab_04_01.c**. Сравните выводы значений на строках 7-12. Почему выводится именно такое значение в каждом из случаев? Создайте переменную **year** со значением 2016. Осуществите вывод всеми возможными способами: 1) ее адреса, 2) ее значения.

3. Дополните код программы **lab_04_01.c**. Создайте целочисленный массив **b**, состоящий из 10 элементов. Используя указатели, поочередно обратитесь ко всем нечетным элементам массива в цикле и выведите их значения.

4. Дополните код программы **lab_04_01.c**. Создайте указатель **ptr** на динамический массив вещественного типа двойной точности. Выделите на него память для 1000 элементов. Используя два отдельных цикла, сначала заполните 130 элементов массива числами по формуле: **ptr[i] = i/100**, после чего, предварительно освободив выделенную память без обнуления указателя, во втором цикле выведите значения массива в терминал. Ознакомьтесь с выводом. Обнулите указатель **ptr** и попытайтесь вывести значения массива. Ознакомьтесь с полученными результатами.

5. Дополните код программы **lab_04_01.c**. Создайте указатель **info** на строковую переменную. Выделите на него память для 4 символов. Заполните указатель **info** значением **"View"**. Перераспределите память с помощью функции **realloc**, выделив для указателя info 100 байт. Осуществите конкатенацию значения **info** со строкой **" Source Code"**. Выведите значение **info** в терминал. Ознакомьтесь с выводом. Обнулите указатель **info** и попытайтесь вывести значения массива. Ознакомьтесь с полученными результатами.

6. Создайте новую программу **lab_04_02.c** по следующему шаблону:

```
#include <stdio.h>
int main() {
    char name[20], surname[50], group[6];
    int day, month, year;
    printf("Введите ваше имя и фамилию: ");
    scanf("%s %s", name, surname);
    printf("Введите дату вашего рождения (DD.MM.YYYY):
");
    scanf("%d.%d.%d", &day, &month, &year);
    printf("Введите номер группы: ");
```

```
    scanf("%s", group);
    printf("\n--АНКЕТА--\n");
    printf("%s %s\n", name, surname);
    printf("%d.%d.%d \n", day, month, year);
    printf("%s \n", group);
    return 0;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией.

Ввод данных с помощью `scanf()`

Ввод данных с клавиатуры в языке Си осуществляется с помощью функции **`scanf()`** и спецификаторов аналогичных используемым при выводе информации. В частности, при вводе с клавиатуры строковых данных формат операции будет следующим: **`scanf("%s", <имя_переменной>)`**, считанное значение будет записано в указанную переменную.

При использовании спецификаторов с шириной числа следует учитывать, что поток считываемых символов является единым для всей программы. Будет считано указанное спецификатором количество символов, после чего следующий **`scanf()`** начнет считывать информацию с того момента, где остановился на предыдущем этапе.

Следует учитывать, что по окончании считывания символьной информации, длина конечной строки будет на один символ больше за счет автоматического добавления в конец строки терминального символа ‘\0’.

При вводе числовых значений с клавиатуры запись осуществляется по адресу переменной. Необходимость использования адреса переменной (**`&`**) обуславливается тем, что **`scanf()`** по умолчанию работает с указателями.

При работе с потоком ввода необходимо учитывать следующее:

- Перевод строки при вводе (нажатие "Enter") является отдельным символом **`\n`**, если он указан в формате считывания, или терминальным в ином случае.
- Так же, как и при стандартном указании значения для переменной, при вводе целого числа в переменную типа **`int`**, превышающего границу значений данного типа, оно будет автоматически преобразовываться в **`long int`** (но не более). Таким образом, при вводе в переменную типа **`int`** значений, превышающих **`long int`**, будет происходить переполнение и в переменную запишется дополнительный код введенного числа относительно границы типа.

- Ввод чисел в экспоненциальном формате осуществляется так же, как и при объявлении значений такого вида, например: **`1.5e-5`** или **`-18e5`**.

7. Дополните код программы **`lab_04_02.c`**. Создайте одномерный целочисленный массив **`a`**, состоящий из 10 элементов. Заполните массив значениями в цикле, вводя каждое значение с клавиатуры. Осуществите вывод заполненного массива в терминал.

8. Дополните код программы **`lab_04_02.c`**. Создайте строковую переменную **`hello`**. Осуществите посимвольный ввод фразы "Hello, World!" в переменную **`hello`**. Осуществите вывод значения переменной в строковом виде.

9. Дополните код программы **`lab_04_02.c`**. Создайте целочисленную переменную **`num`**. Добавьте операцию ввода значения **`num`** с клавиатуры. Запустите код и введите такое значение, чтобы возникло переполнение для переменной **`num`**. Проверку осуществляйте путем вывод записанного в переменную значения в терминал.

10. Дополните код программы **`lab_04_02.c`**. Создайте вещественную переменную двойной точности **`expnum`**. Добавьте операцию ввода значения **`expnum`** с клавиатуры и вывода значения после присваивания с точностью 10 знаков после запятой. Запустите код и введите число в экспоненциальном формате для записи в переменную **`expnum`**. Осуществите проверку записанного числа, визуально сравнив его с выведенным значением.

Лабораторная работа № 5: Функции

1. Создайте новую программу **lab_05_01.c** по следующему шаблону:

```
#include <stdio.h>
int year = 2016; //глобальная переменная
// функция без возврата значений
void printBirthday(int day, int month, int year) {
    printf("Дата рождения: %d.%d.%d \n", day, month,
year);
}
// функция, возвращающая целое число, передача арг. по
значению
int sum(int a, int b) {
    return a + b;
}
// передача арг. по значению и по ссылке
void sumArray(int a[], int aLen, int b[], int bLen, int
*c) {
    if (aLen == bLen) {
        int i = 0;
        for (i = 0; i <aLen; i++) {
            c[i] = a[i] + b[i];
        }
    }
}
// главная функция
int main() {
    int dayB = 1,
        monthB = 1,
        yearB = 1970;
    printBirthday(dayB,monthB,yearB);
    int val1 = 5,
        val2 = 8,
        res;
    res = sum(val1, val2);
    printf("val1 + val2 = %d \n", res);
    int a[5] = {1,2,3,4,5};
    int b[5] = {6,7,8,9,10};
    int c[5];
    sumArray(a, 5, b, 5, c);
    int i = 0;
    for (i = 0; i < 5; i++) {
        printf("c[%d] = %d \n",i,c[i]);
    }
```

```
    return 0;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией.

Глобальные и локальные переменные. Функции

Функции являются отдельными структурными блоками программы, принимающими на вход некоторые аргументы, и возвращающими определенные значения.
В общем виде, формат объявления функций можно представить следующим образом:

```
<тип_возвращаемого_значения> <имя_функции>
(<аргументы>) {
          <блок операций>
    [ return <значение>;]
}
```

Главной функцией языка Си является функция **main** целочисленного типа, возвращающая значение 0. **<тип_возвращаемого_значения>** для функций, не возвращающих значений, устанавливается как **void**. В случае, если функция должна возвращать некоторое значение, при объявлении функции указывается его тип, а в теле функции используется ключевое слово **return** после которого указывается нужное значение. Объявление аргументов функций происходит в обязательном порядке с указанием их типов, например **int sum(int a, int b){return a+b;}**.

Глобальная переменная – это переменная, областью видимости которой являются все функции, содержащиеся в программе. Локальная переменная – переменная, областью видимости которой является конкретная функция, в которой переменная была объявлена или в которую была передана в качестве аргумента. Глобальные переменные объявляются вне метода **main()**.

Аргументы могут быть переданы в функцию двумя способами:

- По значению: **int sum(int a, int b){ return a+b; }**
- По ссылке: **void sum(int a, int *b){ *b+=a; }**

При передаче в качестве аргумента функции массива по значению, указывается тип переменной, ее имя и квадратные скобки с указанием длины массива, если она заранее определена, или пустые квадратные скобки в ином случае, и длина передаваемого массива, например:
void sum(int a[],int aLen){ //… }.

Рассмотрим также передачу в функцию сложных аргументов – параметров командной строки, введенных при запуске программы. Считывание параметров происходит исключительно внутри функции **main**. Для этого необходимо указать для **main** входные аргументы:

```
void main (int argc, char *argv[]) {}
```

В приведенном примере **argv** является массивом аргументов переменной длины, которые вводятся с клавиатуры в терминал при запуске программы, а **argc** представляет собой целое число, равное количеству аргументов. Запуск программы осуществляется путем ввода имени скомпилированного файла программы и перечислением всех дополнительных аргументов через пробел.

2. Дополните код программы **lab_05_01.c**. Замените значения переменных **day**, **month** и **year**, объявленные в функции **main**, на день, месяц и год вашего рождения соответственно. Создайте функцию **printAnotherBirthday,** аналогичную **printBirthday,** с входными аргументами **day** и **month**. Добавьте вызов функции **printAnotherBirthday** в функцию **main**. Запустите код. Ознакомьтесь с выводом. Почему было выведено именно такое значение года?

3. Дополните код программы **lab_05_01.c**. В функции **printAnotherBirthday** добавьте операцию увеличения значения переменной **year** на единицу. Осуществите вывод значения переменной **year** до и после вызова функции. Ознакомьтесь с выводом. Почему было выведено именно такое значение года?

4. Дополните код программы **lab_05_01.c**. Создайте функцию, вычисляющую факториал целого числа, введенного с клавиатуры. После расчетов осуществите вывод числа в терминал из функции **main** в формате: «Факториал числа *число* = *результат*», где вместо '*число*', подставьте значение, введенное с клавиатуры, а вместо '*результат*' – вычисленное значение факториала.

5. Дополните код программы **lab_05_01.c**. Создайте функцию, вычисляющую 11-е число последовательности Фибоначчи с использованием рекурсии. Входные аргументы функции: порядковый номер числа. Осуществите вывод полученного значения в терминал.

6. Модифицируйте программу **lab_05_01.c**. Объявите в функции **main** входные аргументы для считывания трех параметров командной строки.

Добавьте в функцию **main** вывод полученных на входе значений. Запустите программу с указанным количеством параметров. Ознакомьтесь с результатом.

7. Создайте новую программу **lab_05_02.c** по следующему шаблону:

```
#include <stdio.h>
#include <math.h>
#define PI 3.14159265358979323846
int main() {
    double alphaDeg = 45,
           alpha = PI*alphaDeg/180; // угол в радианах
    printf("alphaDeg = %.0lf, alpha = %lf\n", alphaDeg,
alpha);
    printf("sin(alpha) = %lf\n", sin(alpha));
    printf("cos(alpha) = %lf\n", cos(alpha));
    printf("tan(alpha) = %lf\n", tan(alpha));
    printf("atan(tan(alpha)) = %lf\n", atan(tan(alpha)));
    printf("floor(alpha) = %lf\n", floor(alpha)); // окр.
вниз
    printf("ceil(alpha) = %lf\n", ceil(alpha)); // окр.
вверх
    double a = 5, b = 3;
    printf("exp(a) = %lf\n", exp(a)); // вычисление
экспоненты
    printf("log(a) = %lf\n", log(a)); // натуральный
логарифм
    printf("log10(a) = %lf\n", log10(a)); // лог. по осн.
10
    printf("pow(a,b) = %lf\n", pow(a,b)); // возв. a в
степень b
    printf("sqrt(a) = %lf\n", sqrt(a)); // корень числа a
    printf("fabs(-alpha) = %lf\n", fabs(-alpha)); //
модуль вещ.
    return 0;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Для компиляции файла используйте следующую конструкцию:

```
gcc lab_05_02.c –o " lab_05_02" -lm
```

Ознакомьтесь с выведенной информацией.

Глобальные константы и библиотека математических функций <math.h>

Для объявления глобальных констант используется ключевое слово **#define** вне всех функций программы. Рекомендуется объявлять значения констант сразу после подключения необходимых библиотек. Формат объявления глобальных констант:

#define <имя_константы> <значение_константы>

Библиотека **<math.h>** определяет математические операции. Список доступных в библиотеке операций приведен далее:

Функция	Входные аргументы	Описание
sin(a)	**double a** – угол (рад.)	Вычисляет синус **a**
sinh(a)	**double a**	Вычисляет гиперболический синус **a**
asin(a)	**double a**	Вычисляет арксинус **a**
cos(a)	**double a**	Вычисляет косинус **a**
cosh(a)	**double a**	Вычисляет гиперболический косинус **a**
acos(a)	**double a**	Вычисляет арккосинус **a**
tan(a)	**double a**	Вычисляет тангенс **a**
tanh(a)	**double a**	Вычисляет гиперболический тангенс **a**
atan(a)	**double a**	Вычисляет арктангенс **a**
atan2(a,b)	**double a, b**	Вычисляет арктангенс величины **a/b**
exp(a)	**double a**	Вычисляет экспоненту в степени **a**
frexp(a,*eP)	**double a, int *eP**	Разделяет число a на мантиссу, возвращаемую функцией, и экспоненту ***eP**: **a = mantissa*2*exp**
ldexp(a,e)	**double a, int e**	Вычисляет значение, равное **a*2*exp**
log(a)	**double a**	Вычисляет натуральный логарифм числа **a**
log10(a)	**double a**	Вычисляет логарифм по основанию 10 числа **a**
modf(a,*eP)	**double a, int *eP**	Разбивает число a на целую часть – ***eP** и дробную часть, возвращаемую функцией
pow(a,b)	**double a,b**	Возводит число **a** в степень **b**

sqrt(a)	**double a**	Вычисляет корень числа **a**
ceil(a)	**double a**	Округляет число **a** до ближайшего целого, большего либо равного **a**
floor(a)	**double a**	Округляет число **a** до ближайшего целого, меньшего либо равного **a**
fabs(a)	**double a**	Возвращает модуль вещ. числа **a** в виде вещ. числа
fmod(a,b)	**double a,b**	Вычисляет остаток от деления вещественного числа **a** на вещ. число **b**

Для некоторых операционных систем (в частности Linux) необходимо указывать параметр **-lm** при компиляции файла программы, использующего библиотеку **<math.h>**, для обеспечения привязки библиотеки к Вашему файлу. Пример строки для компиляции программы с использованием параметра **-lm**:

```
gcc file.c –o "file" -lm
```

8. Дополните код программы **lab_05_02.c**. Создайте функцию **triangleArea**, вычисляющую площадь треугольника по длинам трех сторон, используя формулу Герона. Значения длин сторон вводятся с клавиатуры и передаются в функцию в качестве входных аргументов. Результат выполнения функции запишите в переменную **result1**, выведите ее значение в терминал.

9. Дополните код программы **lab_05_02.c**. Создайте функцию **findSide**, вычисляющую сторону треугольника по длинам двух сторон и значению угла, используя теорему косинусов. Значения длин сторон и угла (в градусах) вводятся с клавиатуры и передаются в функцию в качестве входных аргументов. Результат выполнения функции запишите в переменную **result2**, выведите ее значение в терминал.

10. Дополните код программы **lab_05_02.c**. Вычислите натуральный логарифм экспоненты в 7 степени с помощью функций библиотеки **<math.h>**. Результат выполнения функции запишите в переменную **result3**, выведите ее значение в терминал.

11. Создайте программу **lab_05_03.c** по следующему шаблону:

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

```
typedef struct {
    long int population;
    char * capital;
    char * name;
} Country;

struct Rectangle {
    int width;
    int height;
};

typedef struct Rectangle rect;

void countryInfo(Country c);
void countryInfoPtr(Country * ptr);

int main() {

    Country rus;
    rus.population = 146544710;
    rus.name = "Russia";
    rus.capital = "Moscow";
    printf("Name: %s\n", rus.name);
    printf("Population (Russia): %ld\n", rus.population);
    rect r1;
    r1.width = 10;
    r1.height = 15;
    rect r2 = {20, 30};
    printf("Area of r1: %d\n", r1.width*r1.height);
    printf("Area of r2: %d\n", r2.width*r2.height);
    struct Rectangle r3;
    r3.width = 12;
    r3.height = 13;
    printf("Area of r3: %d\n", r3.width*r3.height);
    printf("Information for struct:\n");
    countryInfo(rus);
    printf("-----------------------\n");
    printf("Information for struct pointer:\n");
    Country * rus1 = &rus;
    countryInfoPtr(rus1);
    return 0;
}

void countryInfo(Country c) {
    printf("Country %s:\n", c.name);
```

```
    printf("Population - %ld\n", c.population);
    printf("Capital - %s\n", c.capital);
}

void countryInfoPtr(Country * ptr) {
    printf("Country %s:\n", ptr->name);
    printf("Population - %ld\n", ptr->population);
    printf("Capital - %s\n", ptr->capital);
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией.

Тип данных `struct`

Тип данных **`struct`** (структуры) является отдельным форматом данных, содержащим в себе значения различных типов (полей). Все поля в обязательном порядке объявляются в теле структуры через точку с запятой с указанием их типов. Сами структуры объявляются следующим образом:

```
struct <имя_структуры> {
    <тип поля_1> <имя поля_1>;
    <тип поля_2> <имя поля_2>;
    …
}
```

Объявление объекта указанной структуры происходит с указанием типа структуры:

```
struct <имя_структуры> <имя_переменной>;
```

Обращение к полям для указания или получения их значений происходит с использованием точки “**.**”, если структура задана объектом, и с использованием “**->**”, если используется указатель на структуру.

Являясь практически отдельным типом данных, структуры позволяют объявлять отдельные типы данных, построенные на их основе с использованием **`typedef`**:

```
typedef struct [<имя_структуры>] {
    <тип поля_1> <имя поля_1>;
    <тип поля_2> <имя поля_2>;
    …
} <название_типа_данных>;
```

При объявлении самостоятельного типа данных, становится возможным его использование аналогично простым типам данных, например:

```
Country rus;
rus.population = 146544710;
```

Также можно объявлять тип на основе ранее объявленной структуры:

```
typedef struct Rectangle rect;
```

12. Дополните код программы **lab_05_03.c**. Объявите структуру **iterator** с полями **i** и **j** типа **int**, тип данных **matrix** на основе структуры с полями **rows**, **columns**, ***body** типов **int**, **int** и **double** соответственно. Создайте тип данных **iter** на основе структуры **iterator**.

13. Дополните код программы **lab_05_03.c**. В главной функции создайте матрицу **m** (объект типа **matrix**) с количеством строк 3 (поле **rows**) и количеством столбцов 2 (поле **columns**) и задайте значения матрицы, заполнив поле **body** в соответствии с установленными значениями количества строк и столбцов.

14. Дополните код программы **lab_05_03.c**. Создайте функцию **printMatrix**, входным аргументом которой является объект типа **matrix**. В функции реализуйте вывод значений матрицы в терминал с использованием итератора цикла типа **iter**. Осуществите проверку написанной функции вызвав ее для созданной матрицы **m**.

15. Дополните код программы **lab_05_03.c**. Создайте функцию **printMatrixPtr**, входным аргументом которой является указатель на объект типа **matrix**. В функции реализуйте вывод значений матрицы в терминал с использованием итератора цикла типа **iter**. Осуществите проверку написанной функции вызвав ее для указателя на созданную матрицу **m**.

Лабораторная работа № 6: Работа с файлами

1. Создайте файл **file_in.txt** с содержимым "Hello, World!".

2. Создайте новую программу **lab_06_01.c** по следующему шаблону:

```
#include <stdio.h>
int main() {
    FILE *fileIn;
    FILE *fileOut;
    int i,c;
    // чтение файла
    char read[20];
    fileIn = fopen("file_in.txt", "r");
    if (!fileIn) printf("Ошибка чтения файла");
    else{
        i = 0;
       c = fgetc(fileIn);
        while (!feof(fileIn)) {
            read[i] = c;
          c = fgetc(fileIn);
            i++;
        }
    }
    printf("%s \n",read);
    fclose(fileIn);
    // запись в файл
    fileOut = fopen("file_out.txt", "w");
    if (!fileOut) printf("Ошибка чтения файла");
    else{
        fputs("Числа, кратные трем:\n",fileOut);
        for (i = 0; i < 15; i++) {
          if (i%3 == 0) {
                fprintf(fileOut,"%d\n",i);
  }
        }
    }
    fclose(fileOut);
    // переименование файла
    char *oldName = "file_in.txt",
        *newName = "fileIn.txt";
    if(rename(oldName, newName) == 0) {
        printf("%s был переименован %s.\n", oldName,
newName);
    }
```

```
    else {
        printf("Ошибка переименования. %s \n", oldName);
    }
    // копирование файла (file2 -> file1)
    char *file1 = "file_in.txt",
         *file2 = "fileIn.txt";
    FILE *f1 = fopen(file1, "a");
    FILE *f2 = fopen(file2, "r");
    if (!f1) printf("Ошибка чтения файла");
    else{
        c = fgetc(f2);
        while (!feof(f2)) {
            fputc(c,f1);
            c = fgetc(f2);
        }
        printf("Данные %s были скопированы в
%s\n",file2,file1);
    }
    fclose(f1);
    fclose(f2);
    // удаление файла
    char *filename = "fileIn.txt";
    if(remove(filename) == 0) {
        printf("Файл '%s' был удален.\n", filename);
    }
    else {
        printf("Ошибка удаления %s\n", filename);
    }
    return 0;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией.

Работа с файлами

Для работы с файлами в языке Си предусмотрена структура **FILE**. Функции для чтения, записи, редактирования файлов и другие операции, выполняемые над файлами, описаны в библиотеке **<stdio.h>**:

Функция	Входные аргументы	Описание
fopen	**char *filename, char *mode**	Подготавливает файл к работе, открывая его в соответствии с указанным во втором параметре режимом. Аргумент ***mode**

		определяет операции, выполняемые над файлом. Возможные значения ***mode**: • "**r**" – только чтение файла • "**w**" – только запись в файл (перезапись) • "**a**" – запись в файл (дополнение) • "**r+**" – чтение и запись • "**w+**" – чтение и запись • "**a+**" – чтение и запись дополнением
fclose	**FILE *f**	Закрывает доступ к файлу после окончания работы с ним.
feof	**FILE *f**	Осуществляет проверку на достижение конца файла. Если конец файла не достигнут возвращается 0, иначе – ненулевое значение.
fscanf	**FILE *f,** **char *format**	Аналогично функции **scanf**, в заданном формате ***format** считывает символы из потока, которым для **fscanf** является переданный в качестве входного аргумента файл.
fprintf	**FILE *f,** **char *format**	Аналогично функции **printf**, в заданном формате ***format** осуществляет вывод информации в поток, которым для **fprintf** является переданная в качестве входного аргумента структура типа **FILE**.
fgets	**char *str,** **int n,** **FILE *f**	Считывает из файла ***f** не более **n-1** символов и записывает их в ***str**.
fputs	**char *str,** **FILE *f**	Записывает содержимое ***str** в файл ***f**.
fgetc	**FILE *f**	Считывает из файла ***f** символ со следующей позиции в потоке.
fputc	**int c,** **FILE *f**	Записывает символ с кодом **c** в файл ***f** на следующую позиции в потоке.
rename	**char *oldName,** **char *newName**	Переименовывает файл с именем **oldName** в **newName**
remove	**char *name**	Удаляет файл с именем **name**

При попытке использовать **fopen** для несуществующего файла при открытии его на запись, будет создан файл с указанным именем.

3. Создайте новую программу **lab_06_02.c** для выполнения дальнейших заданий по следующему шаблону:

```
#include <stdio.h>
int main() {
    return 0;
}
```

4. Создайте следующие файлы:

Имя файла	Содержимое
lab_06_f01.txt	4,3 1.1,2.4,6.7 8.3,9.0,11.5 1.9,2,3 7.6,4,5.8
lab_06_f02.txt	Good morning, _! Let's drink some tea!

5. Дополните код программы **lab_06_02.c**. Создайте функцию **createArray**, осуществляющую формирование двумерного массива вещественного типа по данным файла **lab_06_f01.txt**, при условии, что в первой строке файла указаны количество строк и столбцов через запятую. Каждая отдельная строка в файле, начиная со второй, соответствует строке двумерного массива, значения в строках разделены запятыми. Функция **createArray** принимает на вход имя файла для обработки. После окончания формирования массива, осуществите вывод его значений в терминал.

6. Дополните код программы **lab_06_02.c**. Создайте функцию **createCSV**, осуществляющую запись созданного на предыдущем шаге двумерного массива в файл **lab_06_f01.csv** в том же формате, что использовался для файла **lab_06_f01.txt**. Функция **createCSV** принимает на вход двумерный массив, который необходимо записать в файл. Удостоверьтесь в правильности работы функции, проверив содержимое созданного файла после завершения обработки.

7. Дополните код программы **lab_06_02.c**. Создайте функцию **greetMe**, осуществляющую замену символа '**_**' в файле **lab_06_f02.txt** на Ваше имя, введенное с клавиатуры и переданное в функцию в качестве входного аргумента. Создайте переменную **greeting** с итоговым значением файла. Выведите значение переменной в терминал.

8. Дополните код программы **lab_06_02.c**. Осуществите копирование файла **lab_06_f02.txt** в файл с именем **lab_06_f03.txt**. Удостоверьтесь

в правильности работы кода, проверив содержимое созданного файла после завершения обработки.

9. Дополните код программы **lab_06_02.c**. Создайте функцию **deleteFile**, осуществляющую удаление файла с именем, переданным в функцию в качестве входного параметра. В функцию **main** добавьте вывод в терминал вопроса "Вы действительно хотите удалить файл **<имя_файла>** ?", где **<имя_файла>** записано в переменной **filename** и известно заранее. Если пользователь вводит с клавиатуры единицу, вызывается функция **deleteFile** для **filename="lab_06_f02.txt"**, иначе ничего не происходит. Проверьте правильность работы кода, введя разные значения с клавиатуры.

Лабораторная работа № 7: Матрицы и действия над ними

1. Требуется разработать библиотеку **xMatrix.h**, содержащую следующие функции:

```
#ifndef XMATRIX_H_INCLUDED
#define XMATRIX_H_INCLUDED
/**
* Чтение матрицы из файла
*
* @param char * filename - имя файла
*
* Формат входного файла:
* m,n
* a11,a12,...,a1n
* a21,a22,...,a2n
* ...
* am1,am2,...,amn
*
* Первая строка файла содержит кол-во строк m и кол-во
столбцов
* n, указанные через запятую.
* Вторая и последующие строки содержат значения
* элементов матрицы.
*
* @return double * matrix - одномерный массив размером
m*n+2,
* содержащий количество строк (элемент matrix[-2]),
* столбцов (элемент matrix[-1]) и все элементы матрицы,
* записанные построчно.
*
*/
double * readMatrix(char *filename);
/**
* Создание матрицы из строки
*
* @param char * str - матрица размером m*n,
представленная в
* виде строки формата:
* [a11,a12,...,a1n;a21,a22,...,a2n;am1,am2,...,amn]
* В качестве разделителей между элементами одной строки
* используется запятая, между строками - точка с запятой
*
* @return double * matrix - одномерный массив размером
m*n+2,
```

```
* содержащий количество строк (элемент matrix[-2]),
* столбцов (элемент matrix[-1]) и все элементы матрицы,
* записанные построчно.
*
*/
double * createMatrix(char *str);
/**
* Создание матрицы, состоящей из нулей
*
* @param unsigned int m - количество строк
* @param unsigned int n - количество столбцов
*
* @return double * matrix - одномерный массив размером
m*n+2,
* содержащий количество строк (элемент matrix[-2]),
* столбцов (элемент matrix[-1]) и все элементы матрицы,
* записанные построчно.
*
*/
double * zeroMatrix(unsigned int m, unsigned int n);
/**
* Создание матрицы, состоящей из единиц
*
* @param unsigned int m - количество строк
* @param unsigned int n - количество столбцов
*
* @return double * matrix - одномерный массив размером
m*n+2,
* содержащий количество строк (элемент matrix[-2]),
* столбцов (элемент matrix[-1]) и все элементы матрицы,
* записанные построчно.
*
*/
double * oneMatrix(unsigned int m, unsigned int n);
/**
* Создание единичной матрицы
*
* @param unsigned int k - порядок матрицы
*
* @return double * matrix - одномерный массив размером
k*k+2,
* содержащий количество строк (элемент matrix[-2]),
* столбцов (элемент matrix[-1]) и все элементы матрицы,
* записанные построчно.
*
```

```
*/
double * eyeMatrix(unsigned int k);
/**
* Получение элемента матрицы по его позиции
*
* @param unsigned int i - номер строки
* @param unsigned int j - номер столбца
* @param unsigned int n - количество столбцов
*
* @return double ind - индекс элемента одномерного
массива,
* в котором записана матрица
*
*/
double elemIndex(unsigned int i, unsigned int j, unsigned
int n);
/**
* Отображение матрицы, вывод значений в терминал
* @param double * matrix - одномерный массив, в котором
* записана матрица
*
* Матрица выводится на экран в формате:
* a11,a12,...,a1n
* a21,a22,...,a2n
* ...
* am1,am2,...,amn
*
*/
void printMatrix(double * matrix);
/**
* Сохранение матрицы в файл
*
* @param double * matrix - одномерный массив, в котором
* записана матрица
* @param char * filename - имя файла
*
* Матрица записывается в файл в формате:
* m,n
* a11,a12,...,a1n
* a21,a22,...,a2n
* ...
* am1,am2,...,amn
*
* @return int flag - результат операции (1-запись прошла
успешно
```

```
* или 0-ошибка записи)
*
*/
int saveMatrix(double * matrix, char * filename);
/**
* Получение числа строк
*
* @param double * matrix - одномерный массив, в котором
* записана матрица
*
* @return int m - количество строк
*
*/
int getRowsNum(double * matrix);
/**
* Получение числа столбцов
*
* @param double * matrix - одномерный массив, в котором
* записана матрица
*
* @return int n - количества столбцов
*
*/
int getColumnsNum(double * matrix);
/**
* Транспонирование матрицы
*
* @param double * matrix - одномерный массив, в котором
* записана исходная матрица
*
* @return double * m - одномерный массив, в котором
* записана транспонированная матрица
*
*/
double * transposeMatrix(double * matrix);
/**
* Объединение матриц
*
* @param double * matrix1 - одномерный массив, в котором
* записана первая матрица
* @param double * matrix2 - одномерный массив, в котором
* записана вторая матрица
* @param char * align - положение второй матрицы
относительно
* первой ("top","left","right","bottom")
```

```
*
* Функция должна осуществлять проверку размеров матриц на
* соответствие для выполнения операции
*
* @return double * matrix - одномерный массив, в котором
* записана результирующая матрица
*
*/
double * combineMatrices(double * matrix1, double *
matrix2, char * align);
/**
* Умножение матрицы на число
*
* @param double * matrix - одномерный массив, в котором
* записана матрица
* @param double num - число
*
* @return double * m - одномерный массив, в котором
* записана результирующая матрица
*
*/
double * multiplyByNumber(double * matrix, double num);
/**
* Умножение матриц
*
* @param double * matrix1 - одномерный массив, в котором
* записана первая матрица
* @param double * matrix2 - одномерный массив, в котором
* записана вторая матрица
*
* Функция должна осуществлять проверку размеров матриц на
* соответствие для выполнения операции
*
* @return double * matrix - одномерный массив, в котором
* записана результирующая матрица
*
*/
double * multiplyByMatrix(double * matrix1, double *
matrix2);
/**
* Сложение матриц
*
* @param double * matrix1 - одномерный массив, в котором
* записана первая матрица
* @param double * matrix2 - одномерный массив, в котором
```

```
* записана вторая матрица
*
* Функция должна осуществлять проверку размеров матриц на
* соответствие для выполнения операции
*
* @return double * matrix - одномерный массив, в котором
* записана результирующая матрица
*
*/
double * addMatrix(double * matrix1, double * matrix2);
/**
* Возведение матрицы в степень
*
* @param double * matrix - одномерный массив, в котором
* записана матрица
* @param unsigned int degree - степень
*
* @return double * m - одномерный массив, в котором
* записана результирующая матрица
*
*/
double * powMatrix(double * matrix, unsigned int degree);
#endif
```

Создайте файл **xMatrix.c**, в котором определите все перечисленные функции.

Библиотека функций

Файл **xMatrix.h** является заголовочным файлом, содержащим объявления всех доступных в библиотеке функций. В заголовочном файле рекомендуется проводить проверку на существование библиотеки с аналогичным именем с помощью конструкции:

```
#ifndef XMATRIX_H_INCLUDED
#define XMATRIX_H_INCLUDED
. . .
#endif
```

Файл **xMatrix.c** содержит определения функций и не является исполняемым файлом, то есть не содержит функции **main**. В комментарии перед объявлением каждой функции находится ее краткое описание, в котором **@param** – аргумент функции, **@return** – возвращаемое значение. Создаваемая библиотека подключается к основным файлам проекта с

помощью конструкции:

```
#include "xMatrix.h"
```

2. Создайте новый файл **lab_07_01.c** и подключите библиотеку **xMatrix.h**. Для компиляции используйте следующую команду:

```
gcc lab_07_01.c xMatrix.c –o "lab_07_01"
```

3. Осуществите полное тестирование написанных Вами функций по следующему алгоритму:

- Проверьте создание несимметричных матриц **A1**, **A2**, **A3** и **A4** размерностью **m1×n1**, **m2×n2**, **m3×n3** и **m4×n4** из файла и из строки, где равными параметрами являются исключительно **n1=m2**, **n2=n3**, **m1=m4**. Выведите матрицы на экран.
- Проверьте создание и вывод на экран нулевой матрицы **A5** размерностью **m5×n5**.
- Проверьте создание и вывод на экран матрицы **A6**, состоящей из единиц, размерностью **m6×n6**.
- Проверьте создание и вывод на экран единичной матрицы **A7** порядка **n7**.
- Проверьте получение количества строк для матрицы **A6**. Выведите значение на экран.
- Проверьте получение количества столбцов для матрицы **A6**. Выведите значение на экран.
- Проверьте объединение матриц для каждого случая, при корректных и некорректных размерностях:
  - **A4** добавляется к **A1** и **A3** добавляется к **A1** слева
  - **A4** добавляется к **A1** и **A3** добавляется к **A1** справа
  - **A2** добавляется к **A3** и **A2** добавляется к **A1** сверху
  - **A2** добавляется к **A3** и **A2** добавляется к **A1** снизу
- Проверьте умножение матрицы **A7** на число. Запишите результирующую матрицу в переменную **A8**. Выведите результирующую матрицу на экран.
- Проверьте сохранение матрицы **A8** в файл с именем **"mat_A8.csv"**. Считайте матрицу **A9** из файла **"mat_A8.csv"**. Выведите матрицу **A9** на экран.
- Проверьте сложение матриц одинаковых и разных размеров. Выводите результирующую матрицу или сообщение об ошибке на экран.
- Проверьте умножение сцепленных и несцепленных матриц. Выведите результирующую матрицу или сообщение об ошибке на экран.
- Проверьте возведение квадратных и прямоугольных матриц в степень. Выведите результирующую матрицу или сообщение об ошибке на экран.

Лабораторная работа № 8: Система линейных алгебраических уравнений

1. Дополните библиотеку **xMatrix.h** следующими функциями:

```
/**
* Вычисление определителя матрицы
*
* @param double * matrix - одномерный массив, в котором
* записана матрица
*
* Функция должна осуществлять проверку на размерность
* матрицы с выводом сообщения об ошибке в случае
* некорректного аргумента
*
* @return double n - значение определителя матрицы
*
*/
double determinant(double * matrix);
/**
* Определение ранга матрицы
*
* @param double * matrix - одномерный массив, в котором
* записана матрица
*
* @return unsigned int n - значение ранга матрицы
*
*/
unsigned int rank(double * matrix);
/**
* Проверка равенства двух матриц
*
* @param double * matrix1 - одномерный массив, в котором
* записана первая матрица
* @param double * matrix2 - одномерный массив, в котором
* записана вторая матрица
*
* @return unsigned int n - результат сравнения матриц:
* 0 - матрицы не равны, 1 - в случае равенства матриц
*
*/
unsigned int equalMatrix(double * matrix1, double *
matrix2);
/**
* Вычисление обратной матрицы
*
```

```
* @param double * matrix - одномерный массив, в котором
* записана матрица
*
* Функция должна осуществлять проверку на на неравенство
* определителя матрицы нулю и размерность матрицы с
выводом
* сообщения об ошибке в случае некорректного аргумента
*
* @return double * m - одномерный массив, в котором
* записана результирующая матрица
*
*/
double * inverseMatrix(double * matrix);
/**
* Вычисление точного решения системы уравнений с
заданными
* параметрами
*
* @param double * matrix1 - матрица коэффициентов системы
* уравнений
* @param double * matrix2 - матрица правых частей
* уравнений
*
* В случае если система является несовместной или
* неопределенной, функция должна выводить соответствующее
* сообщение
*
* @return double * matrix - одномерный массив, в котором
* записана результирующая матрица
*
*/
double * exactSystemSolution(double * matrix1, double *
matrix2);
```

Дополните файл **xMatrix.c** определениями перечисленных функций.

2. Создайте новый файл **lab_08_01.c** и подключите библиотеку **xMatrix.h**.
3. Осуществите полное тестирование написанных Вами функций по следующему алгоритму:
   - Создайте следующие матрицы:
     - Квадратную ненулевую матрицу **A1** порядка **n1**
     - Квадратную нулевую матрицу **A2** порядка **n2**
     - Неквадратную ненулевую матрицу **A3** размерностью **m3×n3**
     - Неквадратную нулевую матрицу **A4** размерностью **m4×n4**

- Проверьте вычисление определителя матрицы для матриц **A1**, **A2**, **A3**.
- Проверьте определение ранга матрицы для матриц **A1**, **A2**, **A3**, **A4**.
- Проверьте сравнение матриц для матриц **A1** и **A2**, **A1** и **A3**, **A3** и **A4**, **A1** и **A1**.
- Проверьте вычисление обратной матрицы для матриц **A1**, **A2**, **A3**, **A4**.
- Проверьте вычисление точного решения для следующих систем уравнений:
  - $\begin{vmatrix} 1 & -1 \\ 2 & -2 \end{vmatrix} x = \begin{vmatrix} 1 \\ 0 \end{vmatrix}$
  - $\begin{vmatrix} 0 & 1 \\ -1 & 1 \end{vmatrix} x = \begin{vmatrix} -1 \\ 2 \end{vmatrix}$
  - $\begin{vmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \end{vmatrix} x = \begin{vmatrix} 1 \\ 1 \\ -2 \end{vmatrix}$
  - $\begin{vmatrix} 1 & -1 \\ 2 & -2 \end{vmatrix} x = \begin{vmatrix} 2 \\ 4 \end{vmatrix}$

Лабораторная работа № 9: Основы программирования на языке C++

1. Запустите онлайн-среду разработки "Tutorials Point" с рабочим пространством для языка программирования C++.

2. Создайте новый файл **lab_09_01.cpp** и запишите в него текст программы:

```
#include <iostream> // библиотека ввода/вывода
#include <string> // библиотека для строкового типа
данных
#include <cstdlib> // stdlib для работы с памятью
#include <cmath> // библиотека математических функций
#include <fstream> // библиотека для работы с файлами
#include <iomanip> // библиотека для форматированного
вывода
using namespace std;

#define PI 3.14159 // глобальный идентификатор

void getCosSinTable();

double perimeter(double array[], int len);

int main() {
    // Ввод/вывод данных
    char name[30];
    int age, day, month, year;
    cout << "Добро пожаловать! Как Вас зовут?\n";
    cin >> name;
    cout << "Здравствуйте, " << name << "! Сколько Вам
лет?\n";
    cin >> age;
    cout << age << " - замечательный возраст, " << name
<<
            "!\nПозвольте узнать, когда Ваш День
Рождения?\n";
    cin >> day >> month >> year;
    cout <<"Надеюсь, что Вы хорошо отметите свой День
Рождения "
         << day << " числа " << month << " месяца\n";

    // Простые типы переменных и операции над ними
    int a = 10;
```

```
    const int b = 3; // константа
    bool t = true, f = false;
    double c = 3.2, d = 4.7;
    char e = 'n', r = 110;
    cout << "bool type" << endl;

    cout << " t AND f = " << (t && f) <<
            ";\n t OR f = " << (t || f) << endl; //
endl='\n'
    cout << "int type" << endl;
    cout << " a+b = " << (a+b) <<
            ";\n a/b = " << (a/b) <<
            ";\n (double) a/b = " << ((double) a/b) <<
            ";\n a%b = " << (a%b) << endl;
    cout << " a++ = " << (a++) <<
            ";\n ++a = " << (++a) <<
            ";\n a-- = " << (a--) <<
            ";\n --a = " << (--a) << endl;
    cout << "double type" << endl;
    cout << " c+d = " << (c+d) << ";\n c/d = " << (c/d)
<<
            endl;
    cout << " e = " << e << ";\n r = " << r << ";\n" <<
endl;

    // Строковый тип данных
    string str;
    string header = "Big news!!";
    string body = "Scientists developed new rules for "
                  "happiness!";
    cout << header + " " + body << endl; // сложение
строк
    cout << body.append(" Really?")
         << endl; // сложение строки
     body.push_back('!'); // добавление символа к строке
     cout << body << endl;
     str.assign(body,0,10); // выделение фрагмента строки
     cout << str;
     cout << str.size() << endl; // длина строки
     cout << str.length() << endl << endl; // длина
строки
     // Ввод строк с клавиатуры
     string likeWeather, dislikeWeather;
     cout << "Какое у Вас любимое время года?\n";
     cin >> likeWeather;
```

```
     cout << "А нелюбимое? После окончания ввода"
         " введите символ ;\n";
     getline(cin, dislikeWeather, ';');
     cout << dislikeWeather << endl;
     cout << "Удалим символ '\\n': " <<
             dislikeWeather.substr(1,
                 dislikeWeather.length()-1) <<
             endl; // выделение подстроки из строки
    cout << "Любимое время года: " << likeWeather <<
endl;
    cout << "Нелюбимое время года: " << dislikeWeather <<
endl;

    // Условия
    double temperature = 25;
    if (temperature > 15) {
        cout << "На улице хорошая погода\n";
    }
    else cout << "На улице прохладно\n";
    // Сокращенная форма условия - тернарный оператор
    (temperature > 15) ? cout << "На улице хорошая
погода\n" :
                         cout << "На улице прохладно\n";

    // Циклы
    cout << "Цикл for: ";
    for (int i = 0; i < 10; i++) {
        cout << i << " ";
    }
    int i = 0;
    cout << "\nЦикл while: ";
    while (i < 10) {
        cout << i << " ";
        i++;
    }
    i = 0;
    cout << "\nЦикл do: ";
    do {
        cout << i << " ";
        i++;
    } while (i < 10);
    cout << endl << endl;

    // Массивы
    cout << "Массив: ";
```

```
    int array[10];
    for (int i = 0; i < 10; i++) {
        array[i] = i%2;
        cout << array[i] << " ";
    }
    cout << endl;
    cout << "Длина массива: " <<
sizeof(array)/sizeof(int) <<
            endl << endl;

    //Указатели и функции
    int num = 5;
    int *pNum = &num;
    int **ppNum = &pNum;

    cout << "num = " << num << endl;
    cout << "*pNum = " <<  *pNum << endl;
    cout << "**ppNum = " <<  **ppNum << endl;
    cout << "&num = " <<  &num << endl;
    cout << "pNum = " <<  pNum << endl;
    cout << "&pNum = " <<  &pNum << endl;
    cout << "ppNum = " <<  ppNum << endl;

    double * arr;
    int arrLen = 4;
    arr = (double *) malloc(arrLen * sizeof(double));
     if (arr != NULL) {
        for (int i = 0; i < arrLen; i++) {
         arr[i] = (double) i/3;
       }
     }
     for (int i = 0; i < arrLen; i++) {
        cout << "arr[" << i << "] = " <<
              arr[i] << ", ";
    }
     free(arr);
    arr = NULL;
    cout << endl;

    double triangle[4];
    for (int i = 0; i < 4; i++) {
    triangle[i] = (double) i/3;
    }
    cout << "Perimeter (no pointer) = " <<
            perimeter( triangle,
```

```
                sizeof(triangle)/sizeof(double) ) <<
endl;

    getCosSinTable();

    // Работа с файлами

    string writeStr = "We want you to know both C and
C++";
    string readStr;

    //Открытие/создание файла, file - имя файла
    ofstream file1("example.txt");
    if (file1) {
        // Запись в файл
        file1 << writeStr; // запись строки
        file1.put('!'); // запись символа
        file1.close(); // закрыть файл
    }

    // Открытие файла на добавление
    ofstream file2("example.txt", ios::app);
    if (file2) {
        file2 << "\nHope, you enjoyed our course!";
        file2.close();
    }


    // Открытие файла на чтение
    ifstream file3("example.txt");
    char ch;
    if (file3) {
        file3.get(ch); // чтение символа
        while(!file3.eof()) {
            readStr += ch;
            file3.get(ch); // чтение символа
        }
        cout << readStr << endl;
        file3.close();
    }

    return 0;
}

void getCosSinTable() {
```

```
    cout << "angle | sin   | cos" << fixed <<
setprecision(2) << endl;
    cout << "0     | " << sin(0)    << "   | " << cos(0)
<< endl;
    cout << "PI/6  | " << sin(PI/6) << "   | " <<
cos(PI/6) << endl;
    cout << "PI/4  | " << sin(PI/4) << "   | " <<
cos(PI/4) << endl;
    cout << "PI/3  | " << sin(PI/3) << "   | " <<
cos(PI/3) << endl;
    cout << "PI/2  | " << sin(PI/2) << "   | " <<
cos(PI/2) << endl;
}

double perimeter(double array[], int len) {
    double result;
    for (int i=0; i < len; i++) {
        result += array[i];
    }
    return result;
}
```

Удостоверьтесь в работоспособности программы, запустив ее через терминал. Ознакомьтесь с выведенной информацией. Компиляцию программы следует осуществлять с использованием следующей команды:

```
g++ lab_09_01.cpp -o lab_09_01
```

Для компиляции программ, написанных на языке программирования C++ используется компилятор **g++**.

Ввод и вывод данных

Программы на языке C++ пишутся с учетом некоторого пространства имен. В приведенном во втором задании примере используется пространство имен **std**. Для упрощения записи команд, пространство было объявлено в начале файла в формате:

```
using namespace std;
```

Для вывода информации в терминал в языке программирования C++ используется следующий формат записи:

```
cout << (<имя_переменной_или_литерал>);
```

cout – это объект выходного потока пространства имен **namespace**. Стандартная библиотека ввода-вывода в C++ – **iostream**. Без предварительного объявления пространства имен, формат для команд ввода/вывода был бы изменен с целью указания используемого пространства имен вручную:

```
std::cout << (<имя_переменной_или_литерал>);
```

Для обозначения перехода на новую строку помимо указания **'\n'**, можно использовать **endl** в формате:

```
cout << (<имя_переменной_или_литерал>) << endl;
```

Желательно не использовать **endl**, если речь идет о выводе на физическое запоминающее устройство.
При выводе значения переменных в терминал используется следующий формат:

```
cout << (<имя_переменной_или_литерал>) <<
<имя_переменной> << endl;
```

Для ввода данных с клавиатуры используется следующий формат записи:

```
cin >> <имя_переменной>;
```

Также для чтения строкового типа данных с целью безопасности возможно использование следующей функции для считывания данных до указанного в параметрах символа:

```
getline(cin, dislikeWeather, ';');
```

Его можно использовать всякий раз, когда речь не идет о выводе на физическое запоминающее устройство.

Для осуществления форматирования ввода и вывода в C++ требуется подключить библиотеку **iomanip**. Для форматирования используются различные функции:

Операция	Функция / Параметр	Пример вызова
Установка точности ввода/вывода данных	**setprecision(int num)**	**cout << setprecision(2);** *Все числа, выводимые далее внутри смыслового блока программы, будут выведены с точностью до 2 знака*
Выравнивание значений при различных результатах. Например, `1.5->1.50`, если до или после него есть числа с двумя значащими цифрами после точки.	**fixed**	**cout << fixed;**
Установка символа, заполняющего пустые порядки	**setfill(int c)**	**cout << setfill('0');**
Вывод чисел с плавающей точкой в экспоненциальном формате	**scientific**	**cout << scientific;**
Вывод индикаторы системы счисления	**showbase**	**cout << showbase;**

3. Создайте файл **lab_09_02.cpp**. Напишите программу, которая анализирует введенную с клавиатуры строку, имеющую формат

[a11,a12,...,a1n;a21,a22,...,a2n;am1,am2,...,amn]

и выводит на экран результат в виде:

```
m,n
a11,a12,...,a1n
a21,a22,...,a2n
  ...
am1,am2,...,amn
```

Здесь **a11,a12,…,a1n,a21,a22,…,a2n,am1,am2,…,amn** – вещественные числа, **m** и **n** – натуральные числа.

Простые типы переменных и операции над ними

В языке программирования C++ используются следующие типы данных:

Название	Размер в байтах	Значение
char	1	от -128 до 127
bool	1	**true**, **false**
int	2	от -2 147 483 648 до 2 147 483 647
unsigned int	2	от 0 до 4 294 967 295
short int	2	от -32 768 до 32 767
unsigned short int	2	от 0 до 65535
long int	4	от -2147483648 до 2147483647
unsigned long int	4	от 0 до 4 294 967 295
float	4	от $3.4e^{-38}$ до $3.4e^{+38}$
double	8	от $1.7e^{-308}$ до $1.7e^{+308}$

По сравнению с языком программирования Си в языке C++ был добавлен булевский тип данных со значениями **true**, **false**. Для работы с данными доступны стандартные математические операции: сложение, вычитание, умножение, деление, вычисление остатка от деления. Для булевского типа используются логические операции (**&&**, **||**, **!**).

4. Дополните код программы **lab_09_02.cpp**. Для каждой логической функции двух аргументов напишите собственную функцию, вычисляющую ее значение, протестируйте работу каждой функции на всех возможных наборах значений аргументов с выводом протокола тестирования в терминал:

```
x1 x2 fn(x1,x2)
0  0  fn(0,0)
0  1  fn(0,1)
1  0  fn(1,0)
1  1  fn(1,1)
```

Строковый тип данных

При использовании библиотеки для работы со строками (**<string>**) становится доступным строковый тип данных **string**, для которого доступны различные операции:

Операция	Функция	Пример вызова
Конкатенация	**-**	**str3 = str1 + str2;**
Добавление символа в конец строки	**push_back(char c)**	**str1.push_back('s');**
Присвоение фрагмента из строки `s` [начиная с позиции `p1` длиной `p2`]	**assign(string s [, int p1, int len])**	**str3.assign(str1,0,3);**
Длина строки	**size()** **length()**	**str.size();** **str.length();**
Выделение фрагмента	**substr(int p1, int len)**	**str.substr(0,3);**
Сравнение строк	**compare(string s)**	**str2.compare(str1);**
Копирование строк	**copy(string s)**	**str2.copy(str1, 5, 0);**
Поиск позиции вхождения подстроки, начиная с позиции **pos**	**find(string s, int pos)**	**str. find(s, p)** *Вместо* ***string s*** *может быть использован символ* ***char c***
Поиск первого вхождения символа, начиная с позиции **pos**	**find_first_of(string s, int pos)**	**str. find_first_of(s, p)** *Вместо* ***string s*** *может быть использован символ* ***char c***

5. Дополните код программы **lab_09_02.cpp**. Создайте строковую переменную со следующим содержимым:

```
#include <iostream>
double sum (double a, double b) {
    return a + b;
}
int main () {
    cout << sum (3, 2) << endl;
return 0;
}
```

Анализируя строку, получите и выведите в терминал отдельно объявления функций и тела функций.

Тернарный оператор

В языке программирования C++ доступны два варианта записи условий типа **if (<условие>) {…} else if(<условие>) {…} else {…}.** Первый вариант – это стандартная полная запись условия. Второй вариант позволяет записать условия типа **if(<условие>){…} else{…}** в сокращенном виде, используя тернарный оператор:

(<условие>) ? <блок операций> : <блок операций>;

При этом первый **<блок операций>** выполняется тогда, когда выполняется условие, второй – в ином случае. После условия обязательно должен стоять знак вопроса, блоки операций разделяются двоеточием.

В циклах, аналогично языку программирования Си, в C++ могут быть использованы ключевые слова, позволяющие опустить выражения, указанные после ключевого слова, и перейти к следующей итерации (**continue**) или полностью остановить выполнение цикла (**break**).

6. Дополните код программы **lab_09_02.cpp**. Напишите программу, которая вычисляет и выводит в терминал количество натуральных чисел от 1 до 1000, которые делятся без остатка на введенное с клавиатуры натуральное число, используя тернарный оператор.

Работа с файлами

Для осуществления работы с файлами существует библиотека **fstream**. В спецификации библиотеки указаны функции открытия и закрытия файла, а также реализующие основные операции над ними.

Операция	Функция	Пример вызова
Открытие файла на запись (замена содержимого)	`ofstream <name>(string filename)`	`ofstream file("f.txt");`
Открытие файла на запись (дополнение и другие параметры)	`ofstream <name>( string filename, <parameter>)`	`ofstream file("f.txt", ios::app);`
Запись строки в файл	`<<`	`file << "Hello!";`
Добавление символа в конец файла	`put(char c)`	`file.put('a');`
Открытие файла на чтение	`ifstream <name>(string filename)`	`ifstream file ("f.txt");`
Открытие файла на чтение с параметрами	`ifstream <name>( string filename, <parameter>)`	`ifstream file("f.txt", ios::in);`
Чтение символа из потока (файла) в переменную `c`. Также возможно использование с записью количества символов `num` в строку `s`.	`get(char c)` `get(string s,[int num])`	`file.get(c);`
Чтение строки из файла [с количеством символов `num`]	`getline(string s,[int num])`	`file.getline(s);`
Проверка достижения конца файла	`eof()`	`file.eof();`
Закрытие файла	`close()`	`file.close();`

При открытии файла на чтение или запись возможно добавление параметров в вызов функции. Эти параметры, представленные в виде предварительно определенных констант языка, отвечают за режим открытия файла. Всего существует 6 типов параметров, которые при желании могут быть объединены добавлением символа ' | ' между именами параметров. Список параметров открытия файла:

- **`ios::in`** – чтение

- **ios::out** – запись
- **ios::app** – запись добавлением
- **ios::ate** – перевод каретки в конец файла
- **ios::binary** – файл интерпретируется как двоичный
- **ios::trunc** – существующее содержимое файла удаляется

7. Дополните код программы **lab_09_02.cpp**. В файлах **matrix1.txt** и **matrix2.txt** записаны сцепленные прямоугольные матрицы. Необходимо считать матрицы из файлов, вычислить их произведение и записать результат в файл **matrix3.txt**. Формат входного файла:

```
m,n
a11,a12,...,a1n
a21,a22,...,a2n
...
am1,am2,...,amn
```

Лабораторная работа № 10: Объектно-ориентированное программирование на языке C++

1. Создайте новый файл **lab_10_01.cpp** и запишите в него текст программы:

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <fstream>
using namespace std;

class Matrix {
    protected:
        // Количество строк
        unsigned int rowsNum;
        // Количество столбцов
        unsigned int columnsNum;
        // Элементы матрицы (одномерный массив)
        double * matrix;

    public:
        // Конструктор класса
        Matrix() {}
        // Деструктор класса
        ~Matrix() {}
        // Создание матрицы из строки
        void createMatrix(string str) {}
        // Создание матрицы из файла
        void createMatrix(FILE *f) {}
        // Задание значения элемента
        double setElement(unsigned int i, unsigned int j)
{}
        // Получение значения элемента
        double getElement(unsigned int i, unsigned int j)
{}
        // Получение количества строк матрицы
        unsigned int getRowsNum() {}
        // Получение количества столбцов матрицы
        unsigned int getColumnsNum() {}
        // Расчет ранга матрицы
        double getRank() {}
        // Транспонирование матрицы
        void transpose() {}
        // Сохранение матрицы в файл
```

```
        void saveToFile(char *filename) {}
        // Вывод матрицы в терминал
        void print() {}
        // Представление матрицы в виде строки
        string toString() {}
};
```

Класс и объект класса

В объектно-ориентированном программировании основными понятиями являются «класс» и «объект» класса. Класс представляет собой категорию, к которой можно отнести множество объектов, обладающих характерными признаками и выполняющих одинаковые действия. Объявление класса происходит с использованием ключевого слова **class** и указанием имени класса (имена классов начинаются с большой буквы). Класс может содержать свойства и методы (функции, описывающие действия его объектов), которые могут иметь различные права доступа, устанавливаемые модификаторами:

- **public** – доступ к свойствам и методам может быть осуществлен напрямую без ограничений;
- **protected** – доступ к свойствам и методам открыт для самого класса и его производных, т.е. унаследованных от него;
- **private** – доступ к свойствам и методам осуществляется исключительно внутри самого класса.

По умолчанию все содержимое класса является доступным для чтения и записи только для него самого.

Рассмотрим пример объявления класса:

```
class Animal{
    public:
        string group; // общедоступное свойство
        string getName() {} // метод
        string setName() {}
        string setGroup() {}


    protected:
        string name; // защищенное свойство
};
```

Для каждого класса по умолчанию существует пустой конструктор объектов класса, не принимающий на вход никаких параметров. В соответствии с требованиями, можно также объявить дополнительные конструкторы, в которые уже могут быть переданы параметры, или же переопределить

существующий пустой конструктор. Конструкторы для класса всегда определяются типом **public**. Например:

```
class Animal{

    // Общедоступные свойства и методы
    public:
        string group;
        Animal () {
            cout << "No parameters";
        }
        Animal (string str) {
            cout << "Parameter: " << str;
        }
        string getName() {} // метод
        string setName() {}
        string setGroup() {}
    // Защищенные свойства и методы
    protected:
        string name;
};
```

Для создания объектов класса и доступа к свойствам и методам используются следующие конструкции:

```
int main() {
    // Создание объекта из пустого конструктора
    Animal cat;

    // Создание объекта из конструктора с параметром
    Animal tiger("param");

    // Установка значения общедоступного свойства
    // с помощью метода
    cat.setGroup("domestic");

    // Установка значения общедоступного свойства
напрямую
    tiger.group = "wild";

    // Установка значения защищенного свойства с
помощью метода
    cat.setName("Barsik");

    return 0;
}
```

С целью освобождения памяти после работы с объектами класса, для класса необходимо помимо конструктора определять деструктор в формате:

```
class Animal{
    public:
        Animal () {} // конструктор
        ~Animal () {} // деструктор
};
```

2. Дополните код программы **lab_10_01.cpp**. Создайте пустой класс **RectangleMatrix**, реализующий прямоугольную матрицу и операции над ней.

Наследование и переопределение методов

Для классов, которые могут расширять ранее созданные классы можно применять наследование. Это позволяет классам, унаследованным от некоторого класса, использовать методы родительского класса, а также переопределять их. Переопределение методов позволяет адаптировать методы родительского класса в соответствии с требованиями дочернего. Также в дочернем классе можно определять методы, являющиеся пустыми в родительском или же добавлять собственные методы. Рассмотрим родительский класс "Геометрическая фигура", для которой определен метод расчета ее площади. При создании класса "Квадрат", переопределяем метод расчета площади, адаптируя его под конкретную фигуру. При наследовании имя родительского класса пишется справа от дочернего через двоеточие с указанием типа методов и свойств, которые будут унаследованы (**public**, **private**, **protected**):

```
class Geometric {
    public:
        double side1;
        double side2;
        double calculateArea() {
            return side1*this.side2;
        };
};

class Square: public Geometric {
    public:
        double calculateArea() {
            return side1*side1;
        };
};
```

В приведенном примере ключевое слово **this** показывает на текущий объект класса, для которого был вызван метод. Для получения информации о свойствах объекта внутри методов можно как использовать **this**, так и обходиться без него.

3. Дополните код программы **lab_10_01.cpp**. Унаследуйте общедоступные свойства и методы родительского класса **Matrix** в дочерний класс **RectangleMatrix**. Определите унаследованные методы в классе **RectangleMatrix.**

4. Дополните код программы **lab_10_01.cpp**. Создайте классы **RowMatrix**, **ColumnMatrix** и **SquareMatrix**, унаследованные от **RectangleMatrix** для определения классов матрицы-строки, матрицы-столбца и квадратной матрицы. Для класса **SquareMatrix** добавьте и определите следующие методы:

```
double determinant() {} // расчет определителя матрицы
void inverse() {} // вычисление обратной матрицы
```

Переопределение математических операций

При создании классов может потребоваться переопределить различные математические операции в зависимости от переданных параметров. Внутри созданного класса возможно определить математические операции, указав нужные аргументы и необходимые для выполнения действия в теле метода. Для переопределения оператора необходимо определить метод, имя которого будет содержать ключевое слово **operator**, а также тип операции (математический знак), которую нужно переопределить. Например, операция сложения двух объектов рассмотренного ранее класса **Geometric** может быть изменена следующим образом:

```
class Geometric {
    public:
        Geometric operator+(Geometric const& r) {
            . . .
        };
};
```

В языке программирования C++ возможно объявление двух методов с одинаковым именем, но разными аргументами. Компилятор самостоятельно осуществит проверку на тип входных аргументов, вызвав нужный метод.

5. Дополните код программы **lab_10_01.cpp**. Для класса **RectangleMatrix** переопределите операции сложения матриц (**+**), умножения матрицы на число (*****), умножения матрицы на матрицу (*****). Для класса **SquareMatrix** добавьте определение операции возведения матрицы в степень (**^**). Для классов **RowMatrix** и **ColumnMatrix** определите операцию скалярного произведения (*****) для объектов одного класса.

6. Дополните код программы **lab_10_01.cpp**. Для классов **RowMatrix** и **ColumnMatrix** определите следующие методы:

```
double euclideNorm() {} // расчет евклидовой нормы
double cubeNorm() {} // расчет кубической нормы
double octahedralNorm() {} // расчет октаэдрической нормы
```

7. Осуществите полное тестирование программы **lab_10_01.cpp** по следующему алгоритму:

- Создайте матрицы одинаковых размерностей **m1×n1 A1**, **A2** как объекты класса **RectangleMatrix** и **A3**, **A4** размерностей **n1×n1** как объекты класса **SquareMatrix** из строки и из файла соответственно.
- Задайте новое значение элементу (1,1) матрицы **A1.**
- Получите и выведите в терминал значение элемента (1,1) матрицы **A2.**
- Получите и выведите в терминал количество строк матрицы **A1.**
- Получите и выведите в терминал количество столбцов матрицы **A1.**
- Вычислите ранг матриц **A1** и **A3**. Выведите полученные значения в терминал.
- Сохраните в файл результат транспонирования матрицы **A1.**
- Выведите матрицу **A1** в терминал.
- Выведите строковое представление матрицы **A2** в терминал.
- Рассчитайте определитель матрицы **A2** и выведите полученное значение в терминал.
- Рассчитайте обратную матрицу для матрицы **A3** и выведите результирующую матрицу в терминал.
- Осуществите проверку переопределенных математических операций над матрицами:
    - Сложите матрицы **A1** и **A3**, **A3** и **A4**. Выведите результат в терминал.
    - Умножьте матрицу **A1** на **A2**, **A1** на **A3**. Выведите результат в терминал.
    - Умножьте матрицу **A3** на число. Выведите результирующее значение в терминал.
    - Возведите матрицу **A4** в квадрат. Выведите результат в терминал.

- Создайте матрицы **A5**, **A6** как объекты классов **RowMatrix** и **ColumnMatrix**.
- Вычислите скалярное произведение матриц **A6** и **A5**. Выведите результат в терминал.
- Вычислите евклидову норму матрицы **A5**. Выведите результирующее значение в терминал.
- Вычислите кубическую норму матрицы **A6**. Выведите результирующее значение в терминал.
- Вычислите октаэдрическую норму матрицы **A5**. Выведите результирующее значение в терминал.

Миссия университета – генерация передовых знаний, внедрение инновационных разработок и подготовка элитных кадров, способных действовать в условиях быстро меняющегося мира и обеспечивать опережающее развитие науки, технологий и других областей для содействия решению актуальных задач.

КАФЕДРА КОМПЬЮТЕРНЫХ ОБРАЗОВАТЕЛЬНЫХ ТЕХНОЛОГИЙ

Кафедра «Компьютерные образовательные технологии» (КОТ) была создана в 2001 году и осуществляла подготовку специалистов направления «Информационные системы» на факультете информационных технологий и программирования по ряду дисциплин (информатика, информационные технологии, дискретная математика, моделирование систем, веб-программирование и др.). В 2003 году кафедра КОТ перешла в статус выпускающей кафедры. В этот год был впервые открыт набор студентов на специальность «Информационные технологии в образовании». В 2011 году кафедра КОТ перешла в состав факультета компьютерные технологии и управление (КТиУ), а в 2015 – в состав факультета программной инженерии и компьютерной техники (ФПИиКТ) мегафакультета КТиУ.

Андрей Владимирович Лямин, Елена Николаевна Череповская

Языки программирования C/C++. Компьютерный практикум

Компьютерный практикум

В авторской редакции
Редакционно-издательский отдел Университета ИТМО
Зав. РИО Н.Ф. Гусарова
Подписано к печати
Заказ №
Тираж
Отпечатано на ризографе

Редакционно-издательский отдел
Университета ИТМО
197101, Санкт-Петербург, Кронверкский пр., 49