

Оглавление

Лабораторная работа №1. Линейные структуры.....	1
Лабораторная работа №2. Условные операторы.....	5
Лабораторная работа №3. Циклы	7
Лабораторная работа №4. Массивы.....	11
Лабораторная работа №5. Функции.	14
Лабораторная работа №6. Строковый тип данных в языке Си.....	16
Лабораторная работа №7. Структуры.	21
Лабораторная работа №8. Файлы.	25
Лабораторная работа №9. Графика. Программирование движущихся объектов.	35
Лабораторная работа №10. Модульное программирование в Си.....	38
Лабораторная работа №11. Списки.	39

Лабораторная работа №1. Линейные структуры.

Теоретическая часть

Структура Си программы:

```
#include <stdio.h> // доступ к библиотеке stdio.h – стандартная библиотека ввода-вывода
#include <stdlib.h> // доступ к библиотеке stdlib.h – дополнительная библиотека
```

```
int main () {
    Оператор 1;
    Оператор 2;
    ....
    return 0;
}
```

Типы данных языка Си:

Таблица 1. Целые типы

Название типа	Обозначение	Размер в байтах	Область значений
Символьный тип	char	1	-128 до 127
Целый тип со знаком	signed, int	зависит от реализации	зависит от реализации
Короткий целый тип со знаком	short, signed short	2	-32 768 до 32 767
Длинный целый тип со знаком	long, signed long	4	-2.147.483.648 до 2.147.483.647
Символьный тип без знака	unsigned char	1	0 до 255
Целый тип без знака	unsigned	зависит от реализации	
Короткий целый тип без знака	unsigned short	2	0 до 65535
Длинный целый тип без знака	Unsigned long	4	0 до 4.294.967.295

Таблица 2. Вещественные типы

Название типа	Обозначение	Размер в байтах
Вещественный тип	float	4
Двойной вещественный	double	8

Объявления переменных. Объявление простой переменной определяет имя переменной и ее тип; оно может также определять класс памяти переменной.

Синтаксис:

<спецификатор типа><пробел><идентификатор>[, <идентификатор>...];

Имя переменной — это идентификатор, заданный в объявлении.

Можно определить несколько различных переменных в одном объявлении, задавая список идентификаторов, разделенных запятой. Каждый идентификатор списка именует переменную.

Примеры;

```
int x; /* Пример 1 */
unsigned long reply, flag; /* Пример 2 */
double order; /* Пример 3 */
```

В первом примере объявляется простая переменная x. Эта переменная может принимать любое значение из множества значений, определяемых для типа int.

Во втором примере объявлены две переменные: reply и flag. Обе переменные имеют тип unsigned long.

В третьем примере объявлена переменная order, которая имеет тип double. Этой переменной могут быть присвоены величины с плавающей запятой.

В языке Си важен регистр букв, т.е. прописные и строчные буквы воспринимаются как разные (int с и int С объявляет 2 разные переменные с и С и обращение к ним в программе будет разным).

Например:

```
int x, X; // объявление двух целых переменных x и X
```

```
char y, symbol; // объявление двух символьных переменных y и symbol
```

```
float chislo; // объявление вещественной переменной chislo
```

Оператор присваивания =

Например, для объявленных переменных из предыдущего примера:

```
X=123;
```

```
x=3;
```

```
y='d'; symbol='ф';
```

```
chislo=23.56;
```

Переменной можно присвоить значение уже при ее объявлении, например, можно объединить 2 предыдущих примера:

```
int x=5, X=12*5; // объявление двух целых переменных x и X с присвоением им значений 5 и 60
```

```
//соответственно
```

```
char y='d', symbol='ф'; // объявление двух символьных переменных y и symbol
```

```
float chislo=23.56; // объявление вещественной переменной chislo
```

Если нужно присвоить нескольким переменным одинаковое значение, то можно использовать следующую конструкцию:
a=b=c=56;

Всем трем переменным присвоится значение 56.

Если при присвоении переменной вещественного типа используется операция деления над переменными целого типа, то необходимо их преобразование к вещественному значению:

неправильно

```
int a=2, b=3;
```

```
float c;
```

```
c=a/b; // c присвоится 0 потому что две целочисленные переменные делятся по законам целочисленной арифметики,
```

```
// и в результате получается ноль, который преобразуется затем в число с плавающей точкой, оставаясь по-прежнему нулем.
```

Правильно

```
int a=2, b=3;
```

```
float c;
```

```
c=a/(float)b; // c присвоится значение 0.66666...
```

или

```
int a=2, b=3;
```

```
float c;
```

```
c=(float) a/b; // c присвоится значение 0.66666...
```

Либо можно было объявить одну из переменных a или b вещественным типом

```
int a=2;
```

```
float b=3;
```

```
float c;
```

```
c=a/b;
```

Операции над переменными :

-, +, / (если оба операнда целого типа, то действует как целочисленное деление), % (остаток от деления, операндами должны быть целые числа), *

Операции присваивания в Си могут вычислять и присваивать значения в одной операции. Используя составные операции присваивания вместо двух отдельных операций, можно сократить код программы и улучшить ее эффективность.

Знак	Операция	Примеры
++	Унарный инкремент (увеличение значения операнда на 1)	int i=3; i++; или ++i; в результате i будет равно 4
--	Унарный декремент (уменьшение значения операнда на 1)	int i=3; i--; или --i; в результате i будет равно 2
=	Умножение с присваиванием	double x; int y; x=y; Операция аналогична простому присваиванию x = x * y

/=	Деление с присваиванием	double x; int y; x/=y; Операция аналогична простому присваиванию $x = x/y$
%=	Остаток от деления с присваиванием	double x; int y; $x \% = y$; Операция аналогична простому присваиванию $x = x \% y$
+=	Сложение с присваиванием	double x; int y; $x += y$; Операция аналогична простому присваиванию $x = x + y$
-=	Вычитание с присваиванием	double x; int y; $x -= y$; Операция аналогична простому присваиванию $x = x - y$

Операции инкремента (++) или декремента (--) могут появляться перед или после своего операнда. Когда операция появляется перед своим операндом, то величина операнда инкрементируется или декрементируется и становится результатом вычисления выражения. Когда операция стоит после своего операнда, то увеличение или уменьшение значения операнда происходит после вычисления всего выражения и использования результата.

Например:

```
b=4;
a=++b; // в результате и переменная a и переменная b получают значение 5
```

```
b=4;
a=b++; // в результате переменная a будет равна 4, переменная b получит значение 5
```

Вывод на экран и ввод с клавиатуры

Для вывода информации на экран в языке Си используется оператор форматного вывода printf("<строка формата>", <выводимые значения, разделенные запятой >);

В строке формата помещается любой текст, поясняющий выводимые значения, и обозначения выводимых переменных и выражений.

Например:

```
int x=2 , y=3, z;
z=x+y;
printf("Значение z=%d \n", z);
```

выведет на экран строчку

Значение z=5

%d означает, что после знака равно будет выведено целое число в десятичной системе. \n означает перевод строки.

Поэтому printf называется оператором форматного вывода – этот оператор определяет формат выводимых значений. Все форматы помещаются после знака % и \ между символами “ ”. После % указываются типы выводимых значений. Сами выводимые значения помещаются после кавычек и запятой.

%d – печать десятичного целого.
 %6d – печать десятичного целого в поле из шести позиций.
 %f – печать вещественного числа.
 %6f – печать вещественного числа в поле из шести позиций.
 %.2f – печать вещественного числа с двумя цифрами после десятичной точки.
 %6.2f – печать вещественного числа с двумя цифрами после десятичной точки в поле из шести позиций.
 %c – печать символьного значения (1 символ)
 %s – печать строкового значения (много символов)
 %% – печать символа %

(полное описание форматов вывода <http://www.cplusplus.com/reference/cstdio/printf/>

или https://ru.wikipedia.org/wiki/Система_типов_Cи)

Например:

```
int x=3;
float y=3.5;
char z='Q';
printf("Значения %d %.2f \n %c \n выведены \n ", x,y,z);
```

на экран будет выведено

```
Значения 3 3.50
Q
выведены
```

\n можно использовать несколько раз в одном printf, таким образом разбивая вывод на несколько строк.

Для ввода данных с клавиатуры в диалоговом режиме используется оператор форматного ввода данных с клавиатуры scanf("<строка формата>", <указатели переменных>);

Например:

```
int i;
float x;
char c;
printf("Введите целое, дробь и символ через пробел");
scanf("%d %f %c", &i, &x, &c); //& – это указатель на переменную, которая идет за ним.
printf("Вот они %d %.2f %c", i, x, c);
```

на экране появится надпись

Введите целое, дробь и символ через пробел

После этого через пробел надо будет ввести данные в нужном порядке (сначала целое, потом вещественное, потом символ). Все, что введете, продублируется на экране благодаря printf.

Если в `scanf("%d %f %c", &i, &x, &c);` в строке формата вместо пробелов между обозначениями типов вводимых данных поставить запятые, то и при вводе данных нужно будет ставить запятые.

```
scanf("%d,%f,%c", &i, &x, &c);
```

все символы, которые будут в строке формата и не являются обозначениями типов вводимых данных, должны будут вводиться пользователем вместе со значениями переменных.

Например:

```
scanf("вот %d,%f,%c", &i, &x, &c);
```

пользователь должен будет сначала ввести буквы **вот**, поставить пробел, потом ввести целое число, поставить запятую, ввести вещественное число, поставить запятую, ввести символ (именно в таком порядке, т.к. этот формат задан строкой формата, поэтому это и называется форматным вводом). Если не соблюсти заданный формат, будет присвоено не определенное заранее значение.

Пример решения задачи:

```
// Вычисление площади прямоугольника
#include <stdio.h> // подключение стандартной библиотеки ввода-вывода
#include <conio.h> // подключение дополнительной библиотеки

int main()
{
    float a,b; // длина и ширина прямоугольника
    float s;    // площадь прямоугольника
    printf("\nВычисление площади прямоугольника\n");
    printf("Введите исходные данные:\n");
    printf("Длина (см.) -> ");
    scanf("%f", &a);
    printf("Ширина (см.) -> ");
    scanf("%f", &b);
    s = a * b;
    printf("Площадь параллелограмма: %10.2f кв.см.\n", s);
    printf("\n\nДля завершения нажмите <Enter>");
    getch(); //процедура для ожидания нажатия клавиши
    return 0;
}
```

Пример работы программы

```
Вычисление площади прямоугольника
Введите исходные данные:
Длина (см.) -> 5
Ширина (см.) -> 4
Площадь параллелограмма:      20.00 кв.см.
```

Для завершения нажмите <Enter>

Задания:

1. Дан радиус окружности, подсчитать длину окружности.
2. Дан радиус окружности, подсчитать площадь круга.
3. Дан прямоугольный треугольник с катетами a и b. Найти гипотенузу c.
4. Дан произвольный треугольник со сторонами a, b и c. Найти площадь треугольника.
5. Найти расстояние между двумя точками с данными координатами.
6. Найти среднее арифметическое трёх заданных чисел.
7. По ребру найти площадь грани, площадь боковой поверхности и объём куба.
8. Даны два действительных числа a и b. Получить их сумму, разность и произведение.
9. Дана длина ребра куба. Найти объём куба и площадь его боковой поверхности.
10. Даны два действительных положительных числа. Найти среднеарифметическое и среднегеометрическое этих чисел (или их модулей).
11. Даны катеты прямоугольного треугольника. Найти его гипотенузу и площадь.
12. Дана сторона равностороннего треугольника. Найти площадь этого треугольника.
13. Даны гипотенуза и катет прямоугольного треугольника. Найти катет и радиус вписанной окружности.
14. Вычислить расстояние между двумя точками с координатами x_1, y_1 и x_2, y_2 .
15. Треугольник задан координатами своих вершин. Найти периметр и площадь треугольника.
16. Написать программу вычисления объема параллелепипеда. Пользователь вводит длины сторон параллелепипеда.
17. Написать программу вычисления объема куба. Пользователь вводит длину стороны куба.

18. Написать программу вычисления стоимости покупки, состоящей из нескольких тетрадей и карандашей. Пользователь вводит цену тетради, кол-во тетрадей, цену карандаша, кол-во карандашей.
19. Написать программу вычисления стоимости покупки, состоящей из нескольких тетрадей и такого же количества обложек к ним. Пользователь вводит цену тетради, цену обложки и кол-во штук.
20. Написать программу вычисления стоимости некоторого количества (по весу) яблок. Пользователь вводит цену 1 кг. яблок и вес яблок в кг.

Лабораторная работа №2. Условные операторы

Теоретическая часть

Оператор if

Простой вариант

```
If (условие) оператор1;
else оператор2; // можно без else
```

Расширенный вариант

If (несколько условий)

```
{
    оператор1;
    оператор2;
    ...
}
else
{
    оператор;
    оператор;
    ...
}
```

Операции отношения

Бинарные операции отношений сравнивают первый операнд со вторым и вырабатывают значение 1(true) и 0 (false). Типом результата является int.

Знак	Отношение
<	Первый операнд меньше, чем второй операнд
>	Первый операнд больше, чем второй операнд
<=	Первый операнд меньше или равен второму операнду
>=	Первый операнд больше или равен второму операнду
==	Первый операнд равен второму операнду
!=	Первый операнд не равен второму операнду

Логические операции

Знак	Операция	Примеры
!	Логическое НЕ	unsigned short y, x; if(!(x<y)) printf("x>=y"); Если x больше или равно y, то результат выражения равен 1 (true). Если x меньше y, то результат равен 0 (false).
&&	Логическое И	int x,y; if (x<y && y<z) printf ("x is less than z\n"); Если x меньше чем y и y меньше чем z, вызывается функция printf для печати сообщения. Если x больше чем y, то второй операнд (y<z) не вычисляется и печати не происходит.
	Логическое ИЛИ	int x,y; if (x == y x == z) printf ("x is equal to either y or z\n"); Сообщение печатается, если x равен y или z. Если x равен y то второй операнд не вычисляется.

0 && 0 = 0 0 || 0 = 0

0 && 1 = 0 0 || 1 = 1

1 && 0 = 0 1 || 0 = 1

1 && 1 = 1 1 || 1 = 1

Операция логического не «!» вырабатывает значение 0, если операнд есть true и значение 1, если операнд есть false. Результат имеет тип int. Операнд должен быть целого, адресного типа или типа с плавающей запятой.

Операнды логических операций И (&&) и ИЛИ (||) могут быть целого, адресного типа или типа с плавающей запятой. Типы первого и второго операндов могут быть различными. Операнды логических выражений вычисляются слева направо. Если значения первого операнда достаточно, чтобы определить результат операции, то второй операнд не вычисляется.

Результатом логической операции является 0 или 1. Тип результата есть int.

Условная операция

В Си есть одна тернарная операция — это условная операция "?:". Она имеет следующее синтаксическое представление:

<операнд 1>?<операнд 2>:<операнд 3>

Выражение <операнд 1> вычисляется с точки зрения его равенства нулю. Оно может быть целого, вещественного или адресного типа. Если <операнд 1> имеет ненулевое значение, то вычисляется <операнд 2> и результатом условной операции является значение выражения <операнд 2>. Если <операнд 1> равен нулю, то вычисляется <операнд 3> и результатом является значение выражения <операнд 3>.

Пример:

j = (i<0) ? (-i) : (i);

В примере j присваивается абсолютное значение i (модуль числа i). Если i меньше нуля, то j присваивается -i. Если i больше или равно нулю, то j присваивается i.

switch

Синтаксис:

```
switch (<выражение>) {  
case <константное выражение>:  
    <оператор>; break;  
[case <константное выражение>:  
    <оператор>; break; ]  
...  
[default:  
    <оператор>]  
}
```

Оператор **switch** передает управление одному из операторов <оператор> своего тела. Оператор, получающий управление, — это тот оператор, для которого **case**-константное выражение равно значению **switch**-выражения в круглых скобках.

Выполнение тела оператора начинается с выбранного оператора и продолжается до конца тела или до тех пор, пока очередной оператор не передает управление за пределы тела.

Оператор **default** выполнится, если **case**-константное выражение не равно значению **switch**-выражения <выражение>. Если **default**-оператор опущен, а соответствующий case не найден, то выполняемый оператор в теле **switch** отсутствует.

Switch-выражение — это целая величина размера **int** или короче.

Каждое case-константное выражение преобразуется к типу **switch**-выражения. Значение каждого **case**-константного выражения должно быть уникальным внутри тела оператора.

Case и **default** метки в теле оператора **switch** существенны только при начальной проверке, когда определяется стартовая точка для выполнения тела оператора. Все операторы, появляющиеся между стартовым оператором и концом тела, выполняются независимо от их меток.

В заголовке составного оператора, формирующего тело оператора **switch**, могут появиться объявления, но инициализаторы, включенные в объявления, не будут выполнены. Назначение оператора **switch** состоит в том, чтобы передать управление непосредственно на выполняемый оператор внутри тела, обойдя строки, которые содержат инициализацию.

В теле **switch** после каждого оператора следует оператор **break**. Оператор **break** осуществляет принудительный выход из **switch**. Последний оператор **break** не является обязательным, поскольку без него управление было бы передано из тела на конец составного оператора.

Множественные метки. Оператор тела **switch** может быть помечен множественными метками, как показано в нижеследующем примере:

```
char c;  
printf("Введите букву -> ");  
scanf("%c", &c);  
switch (c)  
{  
    case 'a':  
    case 'i':  
    case 'e':  
    case 'u':  
    case 'o':  
    case 'y': printf("Это гласная \n"); break;  
    default: printf("Это не гласная \n");  
}
```

Пример выполнения задания:

Написать программу проверки знания даты основания Санкт-Петербурга. В случае неправильного ответа пользователя программа должна выводить правильный ответ.

```

#include <stdio.h>
#include <stdlib.h>
int main ( )
{
    int year; // ответ испытуемого
    printf("\nВ каком году был основан Санкт-Петербург? \n");
    printf("Введите число и нажмите <Enter>  -> " );
    scanf("%d", &year);
    if (year == 1703) printf("Правильно.");
    else
    {
        printf("Вы ошиблись, " );
        printf ("Санкт-Петербург был основан в 1703 году \n");
    }
    printf("\nДля завершения нажмите <Enter>");
    return 0;
}

```

Пример выполнения программы на экране

```

в каком году был основан Санкт-Петербург?
Введите число и нажмите <Enter>
-> 1705
Вы ошиблись, Санкт-Петербург был основан в 1703 году.

```

Задания

1. Найти max значение из трёх величин, определяемых арифметическими выражениями $a = \operatorname{tg}(x)$, $b = \operatorname{ctg}(x)$, $c = \ln(x)$ при заданных значениях x .
2. Даны действительные числа a, b, c . Проверить выполняется ли неравенство $c < b < a$.
3. Даны действительные числа a, b, c . Утроить эти числа, если $a > b > c$ и заменить их абсолютными значениями, если это не так.
4. Даны два действительных числа. Заменить первое число нулём, если оно меньше или равно второму, и оставить числа без изменения иначе.
5. Даны действительные числа x, y . Меньшее из этих двух чисел заменить их полусуммой, а большее – их удвоенным произведением.
6. Даны действительные числа a, b, c, d . Если $a < b < c < d$, то каждое число заменить наибольшим из них; если $a > b > c > d$, то числа оставить без изменения; иначе все числа заменяются их квадратами.
7. Даны действительные числа a, b, c . Выяснить, имеет ли уравнение $ax^2+bx+c=0$ действительные корни. Если действительные корни имеются, то найти их. В противном случае ответом должно служить сообщение, что действительных корней нет.
8. Даны два действительных числа. Вывести первое число, если оно больше второго, и оба числа, если это не так.
9. Даны три действительных числа. Выбрать из них те, которые принадлежат интервалу (1,3).
10. Даны три действительных числа. Выбрать из них те, которые принадлежат интервалу (4,6).
11. Даны три действительных числа. Возвести в квадрат те из них, значения которых не отрицательны.
12. Если сумма трёх попарно различных действительных чисел x, y, z меньше единицы, то наименьшее из этих трёх чисел заменить полусуммой двух других; иначе заменить меньшее из x и y полусуммой двух оставшихся значений.
13. Даны два числа. Если первое число больше второго по абсолютной величине, то необходимо уменьшить первое в 5 раз, иначе оставить числа без изменения.
14. Даны действительные положительные числа x, y, z . Выяснить, существует ли треугольник с длинами сторон x, y, z .
15. Составить программу определения большей площади из двух фигур: круга или квадрата. Известно, что сторона квадрата равна a , радиус круга равен r . Вывести и напечатать значение площади большей фигуры.
16. Определить, является ли целое число чётным.
17. Определить, верно ли, что при делении неотрицательного целого числа a на положительное целое число b , получается остаток, равный одному из двух заданных чисел r или s .
18. Написать программу нахождения суммы большего и меньшего из трех чисел.
19. Написать программу, которая вычисляет частное двух чисел. Программа должна проверять правильность введенных пользователем данных и, если они неверные (делитель равен нулю), выдавать сообщение об ошибке.
20. Напишите программу, которая запрашивает у пользователя номер дня недели, затем выводит название дня недели или сообщение об ошибке, если введены неверные данные.

Лабораторная работа №3. Циклы

Теоретическая часть

Оператор do

Синтаксис:

do

<оператор>

while (<выражение>);

Тело оператора **do** выполняется один или несколько раз (цикл с постусловием), пока выражение <выражение> истинно (равно единице). Вначале выполняется оператор <оператор> тела, затем вычисляется выражение <выражение>. Если выражение ложно, то оператор **do** завершается и управление передается следующему оператору в программе. Если выражение истинно (не равно нулю), то тело оператора выполняется снова и снова проверяется выражение. Выполнение тела оператора продолжается до тех пор, пока выражение не станет ложным. Оператор **do** может также завершить выполнение при выполнении операторов **break**, **goto** или **return** внутри тела оператора **do**.

Пример программы, вычисляющей факториал введенного числа:

```
#include <stdio.h>
#include <stdlib.h>
main ()
{
    int i, fact=1;
    printf("Input i ");
    scanf("%d", &i);
    do {
        fact=fact*i;
        i--;
    }
    while (i>0);
    printf("fact=%d \n", fact);
}
```

Если пользователь введет число 3, то результат выполнения программы будет 6 (1*2*3).

Оператор while

Синтаксис:

while (<выражение>) <оператор>;

Тело оператора **while** выполняется нуль или более раз до тех пор, пока выражение <выражение> станет ложным (равным нулю). Вначале вычисляется выражение <выражение>. Если <выражение> ложно, то тело оператора **while** не выполняется и управление передается на следующий оператор программы. Если <выражение> является истиной (не нуль), то выполняется тело оператора. Перед каждым следующим выполнением тела оператора <выражение> перевычисляется. Оператор **while** может также завершиться при выполнении операторов **break**, **goto**, **return** внутри тела **while**.

Пример программы, вычисляющей факториал введенного числа:

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int i, fact=1; // объявление переменных, fact начальное значение 1
    printf("Введите i ");
    scanf("%d", &i);
    while (i>1) { // пока i больше 1 выполнять
        fact=fact*i; //умножение
        i--; } // декрементация i
    printf("Факториал =%d \n", fact);
    return 0;
}
```

Если пользователь введет число 3, то результат выполнения программы будет 6 (1*2*3).

Оператор for

Синтаксис:

for ([<инициализация>]; [<условие>]; [<выражение цикла>]) <оператор>

Тело оператора **for** (цикл с предусловием) выполняется нуль и более раз до тех пор, пока условное выражение <условие> не станет ложным. Выражения инициализации и цикла могут быть использованы для инициализации и модификации величин во время выполнения оператора **for**.

Первым шагом при выполнении оператора **for** является вычисление выражения инициализации, если оно имеется, затем — вычисление условного выражения с тремя возможными результатами.

1. Если условное выражение истинно (не равно нулю), то выполняется тело оператора. Затем вычисляется выражение цикла (если оно есть). Процесс повторяется снова с вычисления условного выражения.

2. Если условное выражение опущено, то его значение принимается за истину, и процесс выполнения продолжается, как показано выше. В этом случае оператор **for** может завершиться только при выполнении в теле оператора операторов **break**, **goto**, **return**.

3. Если условное выражение ложно, то выполнение оператора **for** заканчивается и управление передается следующему оператору в программе.

Оператор **for** может завершиться при выполнении операторов **break**, **return**, **goto** в теле оператора.

Пример программы, вычисляющей факториал введенного числа:

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
    int i,n,fact=1;
    printf("Input n ");
    scanf("%d", &n);
    for (i=1; i<=n; i++) fact=fact*i;
    printf("fact=%d \n",fact);
    return 0;
}
```

Оператор break

Синтаксис:

break;

Оператор **break** прерывает выполнение операторов **do**, **for**, **switch** или **while**, в которых он появляется. Управление передается оператору, следующему за прерванным. Появление оператора **break** вне операторов **do**, **for**, **switch**, **while** приводит к ошибке.

```
for(i=0; i < 20; i++)
{
    printf("i=%d\n", i);
    if(i == 7)
    {
        printf("Достигли 7-й итерации и выходим\n");
        break;          /* выход из цикла */
    }
}
printf("конец, i=%d\n", i); /* печатает 7 */
```

Оператор continue

Синтаксис

continue;

Оператор **continue** передает управление на следующую итерацию в операторах цикла **do**, **for**, **while**, в которых он может появиться. Оставшиеся операторы в теле вышеперечисленных циклов при этом не выполняются. Внутри **do** или **while** циклов следующая итерация начинается с перевычисления выражения **do** или **while** операторов. Для оператора **for** следующая итерация начинается с выражения цикла оператора **for**.

Пример:

```
int main()
{
    int a,b;
    for (a=1, b=0; a<100; b+=a, a++)
    {
        if (b%2) continue;
        ... /* обработка четных сумм */
    }
    return 0;
}
```

Список задач

Простые циклы

1. Найти сумму квадратов чисел от m до n.
2. Найти сумму квадратов нечётных чисел в интервале, заданном значениями переменных m и n.
3. Найти сумму квадратов четных чисел в интервале, заданном значениями переменных m и n.
4. Найти сумму целых положительных чисел, кратных 4 и меньших 100.
5. Вывести на экран числовой ряд действительных чисел от n до m с шагом 0,2.
6. Написать программу, которая выводит таблицу квадратов первых n целых положительных чисел. Ниже приведен рекомендуемый вид экрана во время работы программы (для первых 10 чисел).

```
Таблица квадратов.
Число    Квадрат
-----
1         1
2         4
3         9
4        16
```

5	25
6	36
7	49
8	64
9	81
10	100

7. Написать программу, которая выводит таблицу степени двойки. Ниже приведен рекомендуемый вид экрана во время работы программы (для первых 5 чисел).

Таблица степеней двойки.	
Степень	Значение

0	1
1	2
2	4
3	8
4	16
5	32

8. Составить программу, печатающую таблицу косинусов с нужным пользователем интервалом в заданном промежутке с точностью до 4-го знака. В Си в библиотеке `math.h` в функции `cos` угол нужно задавать в радианах. Чтобы задать нужный угол в радианах, нужно градусную меру умножить на $\pi/180$. Например, чтобы вычислить $\cos(45)$, нужно написать $\cos(45*3.14/180)$.
9. Составить программу, печатающую квадраты всех натуральных чисел от 0 до заданного натурального n .
10. Даны целые числа K и N ($N > 0$). Вывести N раз число K .
11. Даны два целых числа A и B ($A < B$). Вывести в порядке возрастания все целые числа, расположенные между A и B (включая сами числа A и B), а также количество N этих чисел.
12. Даны два целых числа A и B ($A < B$). Вывести в порядке убывания все целые числа, расположенные между A и B (не включая числа A и B), а также количество N этих чисел.
13. Даны два целых числа A и B ($A < B$). Найти сумму всех целых чисел от A до B включительно.
14. Даны два целых числа A и B ($A < B$). Найти произведение всех целых чисел от A до B включительно.
15. Даны два целых числа A и B ($A < B$). Найти среднее арифметическое всех целых чисел от A до B включительно.
16. Дано число n . В интервале от 1 до n сложить все четные и перемножить все нечетные числа.
17. Дано число n . В интервале от 1 до n сложить все числа, которые делятся на 3 без остатка.
18. Дано число n . В интервале от 1 до n сложить все числа, которые делятся на 5 без остатка.
19. Дано число n . В интервале от 1 до n сложить все числа, которые делятся на 3 и не делятся на 2.
20. Дается a_1 , d , n . Напечатать все члены полученной арифметической прогрессии с заданными параметрами и их сумму.

Сложные циклы

- Написать программу, выясняющую кол-во четных цифр во введенном пользователем числе.
- Написать программу, выясняющую кол-во нечетных цифр во введенном пользователем числе.
- Разложить заданное число на простые множители.
- Число, равное сумме всех своих делителей, включая единицу, называется совершенным. Найти и напечатать все совершенные числа в интервале от 2 до x .
- Среди всех n -значных чисел указать те, сумма цифр которых равна данному числу k (пользователь вводит n и k).
- Среди всех n -значных чисел указать те, произведение цифр которых равно данному числу k (пользователь вводит n и k).
- Найти наибольшую и наименьшую цифры в записи данного натурального числа n .
- Дано натуральное число N . Найти и вывести все числа в интервале от 1 до $N-1$, у которых сумма всех цифр совпадает с суммой цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. $N = 44$. Числа: 17, 26, 35.
- Дано натуральное число N . Найти и вывести все числа в интервале от 1 до $N-1$, у которых произведение всех цифр совпадает с произведением цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. $N = 32$. Числа: 6, 16, 23.
- Дано натуральное число N . Найти и вывести все числа в интервале от 1 до $N-1$, у которых произведение всех цифр совпадает с суммой цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. $N = 44$. Числа: 18, 24.
- Дано натуральное число N . Найти и вывести все числа в интервале от 1 до $N-1$, у которых сумма всех цифр совпадает с произведением цифр данного числа. Если таких чисел нет, то вывести слово «нет». Пример. $N = 32$. Числа: 24, 15, 6.
- Составить программу, которая выводит на экран таблицу умножения от 1 до 9 в десятичной системе счисления.
- С клавиатуры вводится двузначное число, среди всех четырехзначных чисел вывести на экран те, которые начинаются или заканчиваются этим числом.
- Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых сумма первых двух цифр равна сумме последних двух.
- Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых произведение первых двух цифр равно сумме последних двух.

16. Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых произведение первых двух цифр равно произведению последних двух.
17. Среди четырехзначных чисел из интервала, заданного пользователем, найти все, у которых сумма первых двух цифр равна произведению последних двух.
18. Пользователь вводит числа x , a , b . Из промежутка от a до b найти все числа, сумма цифр которых дает x .
19. Пользователь вводит числа x , a , b . Из промежутка от a до b найти все числа, разность цифр которых по модулю дает x .
20. Пользователь вводит x , a , b . Из промежутка от a до b найти все числа, произведение цифр которых по модулю дает x .

Лабораторная работа №4. Массивы.

Теоретическая часть

Объявление массива определяет тип массива и тип каждого элемента. Оно может определять также число элементов в массиве.

Синтаксис:

```
<спецификатор типа> <пробел> <описатель>[<константное выражение>;  
<спецификатор типа> <пробел> <описатель>[];
```

Здесь квадратные скобки — это специальные символы, участвующие в объявлении. Объявление массива может представляться в двух синтаксических формах, указанных выше. <Описатель> задает имя переменной. Квадратные скобки, следующие за описателем, указывают на то, что объявляется массив. Константное выражение, заключенное в квадратные скобки, определяет число элементов в массиве. Каждый элемент имеет тип, задаваемый спецификатором типа.

Во второй синтаксической форме опущено константное выражение в квадратных скобках. Эта форма может быть использована только тогда, когда массив инициализируется, объявлен как формальный параметр или объявлен как ссылка на массив, явно определенный где-то в программе.

Массив массивов (многомерный массив) определяется путем задания списка константных выражений в квадратных скобках:

```
<спецификатор типа> <описатель>[<константное выражение>] [<константное выражение>]...
```

Каждое <константное выражение> в квадратных скобках определяет число элементов в данном измерении массива, так что объявление двумерного массива содержит два константных выражения, трехмерного — три и т.д.

Двумерный массив можно представить в виде матрицы, в которой первое константное выражение означает кол-во строк, а второе — кол-во столбцов.

Двумерные массивы в языке Си хранятся построчно, т.е. сначала все элементы первой строки, потом второй и т.д.

Примеры:

```
int scores[10]; /* Пример 1 */  
float matrix[10][15]; /* Пример 2 */
```

В первом примере объявляется переменная типа массив с именем `scores` из 10 элементов типа `int`. Во втором примере объявляется двумерный массив с именем `matrix`. Массив состоит из 150 элементов типа `float`, т.е. его можно представить в виде матрицы из 10 строк и 15 столбцов. Нумерация элементов массива в языке С начинается с 0!!!

Операции с массивами.

Заполнение массива. Можно сразу при объявлении массива:

Пример 1.

```
int ar[5] = { 12, 23, 34, 45, 56 };
```

Либо можно заполнить массив в цикле. Для одномерного массива используется обычный цикл, для двумерного — вложенный цикл. Пример заполнения одномерного массива:

Пример 2.

```
int a[5];  
int i;  
for(i=0; i < 5; i++)  
    a[i] = i; // заполнение массива индексами его элементов, a[0]=0, a[1]=1, a[2]=2, a[3]=3, a[4]=4
```

В данном случае индекс цикла служит также и индексом в массиве, и значением элемента массива.

Пример заполнения двумерного массива:

Пример 3.

```
int f[2][3]={1,2,3}, {4,5,6}; // заполнение при объявлении
```

Пример 4.

```
int a[3][2];  
int i,j;  
for(i=0; i < 3; i++)  
    for (j=0; j < 2; j++)
```

```
a[i][j] = i*10+j; // заполнение массива индексами его элементов, a[0][0]=0, a[0][1]=1,  
//a[1][0]=10, a[1][1]=11 и т.д.
```

Вывод значений массива на экран

Чтобы вывести на экран значения элементов массива, также используется цикл. Пример для одномерного массива:

Пример 5.

```
for(i=0; i < 5; i++)  
    printf("a[%d]=%d ",i, a[i]); // вывод на экран
```

для массива, заполненного в **примере 2**, на экране появится

a[0]=0 a[1]=1 a[2]=2 a[3]=3 a[4]=4

Пример вывода на экран значений двумерного массива

Пример 6.

```
for(i=0; i < 3; i++){  
    for (j=0; j < 2; j++) printf("a[%d][%d]=%d ",i,j, a[i][j]);  
    printf("\n"); // чтобы каждая строка массива выводилась с новой строки экрана  
}
```

Для массива, заполненного в **примере 4**, на экране появится

a[0][0]=0 a[0][1]=1
a[1][0]=10 a[1][1]=11
a[2][0]=20 a[2][1]=21

Использование генератора случайных чисел для заполнения массивов.

Заполнение больших массивов удобно осуществлять с помощью генератора случайных чисел. Для его использования необходимо подключить библиотеку `stdlib.h` и `time.h` (последнее необязательно для `turbo c`)

Общий случай:

```
srand (time (NULL)); // запуск генератора  
переменная = нач значение +rand() % конечное значение
```

или для turbo c

```
randomize();  
переменная = rand() % конечное значение;
```

Случайные числа будут генерироваться в диапазоне от 0 до конечного значения.

Например:

для одномерного массива

```
srand (time (NULL));  
int a[100];  
int i;  
for(i=0; i < 100; i++)  
    a[i] =rand()%10; // в интервале от 0 до 10
```

для двумерного массива

```
srand (time (NULL));  
int a[3][2];  
int i,j;  
for(i=0; i < 3; i++)  
    for (j=0; j < 2; j++)  
        a[i][j] =rand()%100; // в интервале от 0 до 100
```

Указатели в языке Си

Указатель – это переменная, которая содержит адрес другой переменной, в которой хранятся какие-либо данные.

Например, объявление

```
int *pa;
```

задает указатель на переменную целого типа. Как и обычные переменные, указатель после объявления содержит «мусор». Чтобы его «настроить» на определенную переменную, нужно получить ее адрес. Для этого в языке Си есть специальный оператор `&`.

Пример:

```
#include <stdio.h>  
main() {  
    int i;  
    int *px; /* px - указатель на int */  
    i=2;
```

```

px=&i; /*в px - адрес i */
*px=3; /* i теперь равно 3! */
printf("%d\n",i);
}

```

В программе из примера сначала объявляются две переменные: i (типа int) и px (указатель на int).

Затем переменной i присваивается значение 2. Далее идет строчка

```
px=&i;
```

которая засылает в указатель px адрес переменной i.

Следующая строка

```
*px=3;
```

записывает число 3 по адресу, хранящемуся в px. Но в px хранится адрес переменной i, значит i теперь равна 3!

Действительно, скомпилировав и запустив программу, мы увидим, что она выводит на экран число 3.

Звездочка*, стоящая перед указателем px называется оператором раскрытия ссылки. Получается, что если px — указатель на int, то *px — переменная типа int. Теперь понятно, почему так странно объявляется указатель. Строка int *px; как бы говорит нам, что *px — это что-то типа int.

Массивы и указатели.

Компилятор рассматривает имя массива как адрес его нулевого элемента. Встретив какой-то элемент массива, например arr[i], компилятор преобразует его по правилам работы с указателями: *(arr + i). То есть, arr — это как бы указатель, i — целочисленная переменная. Сумма arr+i указывает на i-й элемент массива, а оператор раскрытия ссылки * дает значение самого элемента.

Динамические массивы

Если объявить массив как указатель, то можно не указывать его размер при объявлении. Но нужно будет выделить память под будущий размер массива с помощью функции malloc (а потом эту память освободить с помощью функции free).

```

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

int main(){
    int *p; // объявляем p как указатель на целое
    int i,n;
    srand(time(NULL));
    printf("Введите размерность массива -> ");
    scanf("%d",&n); // вводим размерность массива
    p=(int *)malloc(n*sizeof(int)); // выделяем под наш массив типа int* размер памяти,
    // равный n*(размер памяти под 1 элемент типа int)
    //т.е. p=(int *)malloc(n*sizeof(int));

    for(i=0;i<n;i++)
    {
        p[i]=rand()%5; // генерируем i-ый элемент массива p
        printf("%d ",p[i]); // выводим на экран
    }
    free(p); // освободить память, выделенную под p
    return 0;
}

```

Теперь, какое значение n с клавиатуры мы введем, такой размер памяти под наш массив и выделится, и столько элементов будет выведено на экран.

Список задач

I. Одномерные массивы.

1. Дан массив. Удалить из него нули и после каждого числа, оканчивающего на 5, вставить 1.
2. Случайным образом генерируется массив чисел. Пользователь вводит числа a и b. Заменить элемент массива на сумму его соседей, если элемент массива четный и номер его лежит в промежутке от a до b.
3. В одномерном массиве удалить промежутки элементов от максимального до минимального.
4. Дан одномерный массив. Переставить элементы массива задом-наперед.
5. Сформировать одномерный массив случайным образом. Определить количество четных элементов массива, стоящих на четных местах.
6. задается массив. Определить порядковые номера элементов массива, значения которых содержат последнюю цифру первого элемента массива 2 раза (т.е. в массиве должны быть не только однозначные числа).
7. Сформировать одномерный массив из целых чисел. Вывести на экран индексы тех элементов, которые кратны трем и пяти.
8. задается массив. Написать программу, которая вычисляет, сколько раз введенная с клавиатуры цифра встречается в массиве.
9. задается массив. Узнать, какие элементы встречаются в массиве больше одного раза.
10. Даны целые числа a_1, a_2, \dots, a_n . Вывести на печать только те числа, для которых $a_i \geq i$.
11. Дан целочисленный массив с количеством элементов n. Напечатать те его элементы, индексы которых являются степенями двойки.
12. задана последовательность из N чисел. Определить, сколько среди них чисел меньших K, равных K и больших K.

13. Задан массив действительных чисел. Определить, сколько раз меняется знак в данной последовательности чисел, напечатать номера позиций, в которых происходит смена знака.
14. Задана последовательность N чисел. Вычислить сумму чисел, порядковые номера которых являются простыми числами.
15. Дан массив чисел. Указать те его элементы, которые принадлежат отрезку $[c, d]$.
16. Массив состоит из нулей и единиц. Поставить в начало массива нули, а затем единицы.
17. Дан массив целые положительных чисел. Найти среди них те, которые являются квадратами некоторого числа x .
18. В массиве целых чисел найти наиболее часто встречающееся число. Если таких чисел несколько, то определить наименьшее из них.
19. Дан целочисленный массив с количеством элементов n . Сжать массив, выбросив из него каждый второй элемент.
20. Дан массив, состоящий из n натуральных чисел. Образовать новый массив, элементами которого будут элементы исходного, оканчивающиеся на цифру k .
21. Даны действительное число x и массив $A[n]$. В массиве найти два члена, среднее арифметическое которых ближе всего к x .
22. Даны два массива A и B . Найти, сколько элементов массива A совпадает с элементами массива B .

II. Двумерные массивы.

1. Написать программу, генерирующую магические квадраты заданного пользователем размера.
2. Дан двумерный числовой массив. Значения элементов главной диагонали возвести в квадрат.
3. Дан двумерный массив. Поменять местами значения элементов столбца и строки на месте стыка минимального значения массива (или первого из минимальных). Например, если индекс минимального элемента (3;1), т.е. он находится на пересечении 3 строки и 1 столбца, то 3 строку сделать 1 столбцом, а 1 столбец сделать 3 строкой.
4. Дан двумерный массив. Сформировать одномерный массив только из четных элементов двумерного массива.
5. Дан двумерный массив. Найти сумму элементов массива, начиная с элемента, индексы которого вводит пользователь, и заканчивая элементом, индексы которого вводит пользователь.
6. Дан двумерный массив. Сформировать одномерный массив только из элементов двумерного массива с четной суммой индексов.
7. Дан двумерный массив. Сделать из него 2 одномерных: в одном – четные элементы двумерного массива, в другом – нечетные.
8. Вычислить сумму и число положительных элементов матрицы $A[N, N]$, находящихся над главной диагональю.
9. В квадратной матрице определить максимальный и минимальные элементы. Если таких элементов несколько, то максимальный определяется по наибольшей сумме своих индексов, минимальный – по наименьшей.
10. Для заданной квадратной матрицы сформировать одномерный массив из ее диагональных элементов.
11. Заданы матрица порядка n и число k . Вычесть из элементов k -й строки диагональный элемент, расположенный в этой строке.
12. Заданы матрица порядка n и число k . Вычесть из элементов k -го столбца диагональный элемент, расположенный в этом столбце.
13. Дана прямоугольная матрица. Найти строку с наибольшей и наименьшей суммой элементов. Вывести на печать найденные строки и суммы их элементов.
14. Дана прямоугольная матрица. Найти столбец с наибольшей и наименьшей суммой элементов. Вывести на печать найденные столбцы и суммы их элементов.
15. Дан двумерный массив. Выяснить, в каких строках сумма элементов меньше введенного пользователем значения.
16. Дан двумерный массив. Выяснить, в каких столбцах произведение элементов меньше введенного пользователем значения.
17. Дан двумерный массив. Выяснить, есть ли столбец и строка с одинаковой суммой элементов. Если есть, напечатать их номера.
18. Дан двумерный массив. Выяснить, есть ли столбец и строка с одинаковым произведением элементов. Если есть, напечатать их номера.
19. Дан двумерный массив. Выяснить, есть ли строки с одинаковой суммой элементов. Если есть, вывести их номера.
20. Дан двумерный массив. Выяснить, есть ли столбцы с одинаковой суммой элементов. Если есть, вывести их номера.
21. Дан двумерный массив. Определить максимальный среди положительных элементов, минимальный среди отрицательных элементов и поменять их местами.
22. Дан двумерный массив. Заменить первый нуль в каждом столбце на количество нулей в этом столбце.
23. Дан двумерный массив. Заменить первый нуль в каждой строке на количество нулей в этой строке.

Лабораторная работа №5. Функции.

Теоретическая часть

В отличие от других языков программирования высокого уровня в языке программирования C нет деления на процедуры, подпрограммы и функции, здесь вся программа строится только из функций.

С использованием функций в языке программирования C связаны три понятия – определение функции (описание действий, выполняемых функцией), объявление функции (задание формы обращения к функции) и вызов функции.

Определение функции задает тип возвращаемого значения, имя функции, типы и число формальных параметров, а также объявления переменных и операторы, называемые телом функции, и определяющие действие функции.

Пример:

```
int rus (char r)
{ if ((r>='А' && r<='П') || (r>='п' && r<='ё')) return 1; else return 0; }
```

В данном примере определена функция с именем *rus*, имеющая один параметр с именем *r* и типом *char*. Функция возвращает целое значение, равное 1, если параметр функции является буквой русского алфавита, или 0 в противном случае.

В языке программирования С нет требования, чтобы определение функции обязательно предшествовало ее вызову. Определения используемых функций могут следовать за определением функции *main*, перед ним, или находится в другом файле. Однако для того, чтобы компилятор мог осуществить проверку соответствия типов передаваемых фактических параметров типам формальных параметров до вызова функции нужно поместить объявление (прототип) функции.

Объявление функции имеет такой же вид, что и определение функции, с той лишь разницей, что тело функции отсутствует, и имена формальных параметров тоже могут быть опущены. Для функции, определенной в последнем примере, прототип может иметь вид

```
int rus (char r); или rus (char);
```

Функция возвращает значение, если ее выполнение заканчивается оператором *return*, содержащим некоторое выражение. Указанное выражение вычисляется, преобразуется, если необходимо, к типу возвращаемого значения и возвращается в точку вызова функции в качестве результата. Если оператор *return* не содержит выражения или выполнение функции завершается после выполнения последнего ее оператора (без выполнения оператора *return*), то возвращаемое значение не определено.

Для функций, не использующих возвращаемое значение, должен быть использован тип *void*, указывающий на отсутствие возвращаемого значения. Если функция определена как функция, возвращающая некоторое значение, а в операторе *return* при выходе из нее отсутствует выражение, то поведение вызывающей функции после передачи ей управления может быть непредсказуемым.

Если функция не использует параметров, то наличие круглых скобок обязательно, а вместо списка параметров рекомендуется указать слово *void*.

Порядок и типы формальных параметров должны быть одинаковыми в определении функции и во всех ее объявлениях. Типы фактических параметров при вызове функции должны быть совместимы с типами соответствующих формальных параметров. Тип формального параметра может быть любым основным типом, структурой, объединением, перечислением, указателем или массивом. Если тип формального параметра не указан, то этому параметру присваивается тип *int*.

Тело функции – это составной оператор, содержащий операторы, определяющие действие функции.

Пример: Написать программу сложения, вычитания и умножения 2-х чисел с использованием функций.

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

int a,b; //глобальные переменные

int et1(int a, int b) //функция, возвращает значение типа int - сумму чисел
{
    int c=a+b;
    return c;
}

void et2(int a, int b) // функция без возвращаемого значения, выводит на экран разницу двух чисел
{ printf("\n%d-%d=%d",a,b,a-b); }

void et3 (void) // функция без возвращаемого значения и без параметров, выводит на экран
                //произведение двух чисел
{ printf("\n%d*d=%d",a,b,a*b); }

int main()
{
    scanf("%d %d",&a,&b);
    printf("\n%d+%d = %d",a,b, et1(a,b));
    et2(a,b);
    et3();
    return 0;
}
```

Передача функции массива как параметра

Чтобы передать функции массив, как один из параметров функции (например, чтобы создать функцию для вывода массива на экран), нужно объявить параметр-массив как указатель, а при вызове функции передать ей указатель на его первый элемент, или на элемент, с которого нужно начать обработку массива.

Например, напишем программу с описанием функции вывода массива на экран:

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

void vivod(int *mas, int n) //объявляем и описываем функцию вывода массива на экран,
                           // при этом объявляем параметр int *mas - указатель на целое,
                           //через него и будем передавать массив
{
    printf("\n");
```

```

        for (int i=0;i<n;i++)
            printf(" %d",mas[i]);
    }

int main()
{
    int *p; // объявляем p как указатель на целое
    int i,n;
    srand(time(NULL));
    printf("Введите размерность массива -> ");
    scanf("%d",&n);
    p=(int *)malloc(n*sizeof(int)); //
    for(i=0;i<n;i++)
        p[i]=rand()%5; // генерируем i-ый элемент массива p

    vivod(p,n); // вызываем описанную ранее функцию и передаем ей наш массив
                // (который ранее мы объявили как указатель на целое)
    vivod(p,5); // то же самое, но выводим только 5 первых элементов
    vivod(&p[5],4); // передаем функции указатель на пятый элемент, и, начиная с него, будет
                  // выведено 4 элемента
    free(p);
    return 0;
}

```

Список задач

1. Написать функцию, которая вычисляет объем цилиндра. Параметрами функции должны быть радиус и высота цилиндра.
2. Напишите программу поиска максимального из четырех чисел с использованием функции поиска большего из двух.
3. Дано число n. Вычислите сумму: $1! + 2! + 3! + \dots + n!$, создав функцию вычисления факториала числа.
4. Создайте программу вычисления суммы трехзначных чисел, в десятичной записи которых нет четных цифр, оформив функцию, определяющую наличие четных цифр в числе.
5. Создайте программу вычисления суммы трехзначных чисел, в десятичной записи которых нет нечетных цифр, оформив функцию, определяющую наличие нечетных цифр в числе.
6. Написать функцию, меняющую местами значения элементов массива. Параметрами функции должны быть массив и номера элементов, которые нужно поменять.
7. Написать функцию, умножающую все элементы массива на число. Параметрами функции является массив и число, на которое нужно умножить его элементы.
8. Написать функцию, умножающую первые n элементов массива на некоторое число. Параметрами функции должны быть массив и кол-во первых элементов, которые надо изменить.
9. Написать функцию, которая выводит на экран строку, состоящую из звездочек. Длина строки (количество звездочек) является параметром функции.
10. Написать функцию, которая выводит строку, состоящую из одинаковых символов. Длина строки и символ являются параметрами функции.
11. Написать функцию Procent, которая возвращает нужный процент от полученного в качестве аргумента числа.
12. Написать функцию glasn, которая возвращает 1, если символ, полученный функцией в качестве аргумента, является гласной буквой русского алфавита, и 0 — в противном случае.
13. Написать функцию sogl, которая возвращает 1, если символ, полученный функцией в качестве аргумента, является согласной буквой русского алфавита, и 0 — в противном случае.
14. Написать функцию, которая складывает поэлементно 2 массива. Параметрами функции должны быть оба массива и массив, в который записывается результат.
15. Написать функцию, которая умножает поэлементно 2 массива. Параметрами функции должны быть оба массива и массив, в который записывается результат.
16. Написать функцию, которая переставляет местами первые n элементов массива с последними n элементами. Параметрами функции должны быть массив и число n.
17. Написать функцию, которая переставляет местами первые n элементов массива A с последними n элементами массива B. Параметрами функции должны быть оба массива и число n.
18. Написать функцию, вычисляющую сумму арифметической прогрессии. В качестве параметров ей передается первый элемент последовательности, разность и кол-во (a, d, n).
19. Написать функцию, решающую квадратное уравнение $ax^2+bx+c=0$. В качестве параметров ей должны передаваться a, b, c.
20. Написать функцию, которая вычисляет объем и площадь поверхности параллелепипеда.

Лабораторная работа №6. Строковый тип данных в языке Си.

Теоретическая часть

Строка в языке С — это массив символов.

Объявление строковой переменной

char str1[10]; // Строка — массив из 10 символов. Начальное значение символов не определено.

char str2[10]="Hello";


```
/* Используется инициализация (не присваивание!). В первые 5 символов записывается "Hello", в 6 – нуль-терминатор, значение трех последних не определено.*/
char str3[10]={'H', 'e', 'l', 'l', 'o', '\0'}; //Эквивалентно предыдущему.
char str4[10]="Very long line";
//Ошибка. Массив из 10 элементов нельзя инициировать более длинной последовательностью.
char str5[]="Very long line";
/*Компилятор автоматически определяет длину массива (в нашем случае 15) и инициализирует его последовательностью символов.*/
```

Присваивание строк

Первый и самый очевидный способ присваивания строк – присваивание отдельных символов. Например:

```
str1[0]='H';
str1[1]='e';
str1[2]='l';
str1[3]='l';
str1[4]='o';
str1[5]='\0';
```

Но это неудобно. Для присвоения строке нужного значения существуют несколько библиотечных функций, наиболее общепотребительной из которых является функция `char* strcpy(char* dest, const char* src)`

strcpy(<строка в которую помещается значение>, <строка из которой берется значение>) (нужно подключать модуль `string.h`)

Например:

```
char str1[10], str2[10];
strcpy(str1, "Hello");
strcpy(str2, str1);
```

При использовании этой функции следует соблюдать осторожность. Кол-во присваиваемых символов не должно превышать длину объявленной строки, иначе произойдет ошибка присвоения данных.

Ввод строк с клавиатуры

С помощью оператора `scanf` можно ввести значение строки до первого введенного пробела, после первого пробела ничего записано не будет. **Например:**

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    printf("Введите строку ");
    char s[20];
    scanf("%s", s);
    printf("Вот она %s\n", s);
    return 0;
}
```

Если пользователь введет строку **мама мыла раму**, то после слов **Вот она** будет выведено только 1 слово – **мама**.

Всю строку с пробелами можно ввести с клавиатуры с помощью оператора `gets`.

Его синтаксис:

`gets`(переменная строкового типа);

Например:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    char s1[4];
    printf("Введите строку ");
    gets(s1);
    printf("Вот она %s\n", s1);
    return 0;
}
```

Сравнение строк

Строки сравниваются по их расположению в словаре (т.е. сначала сравниваются первые символы, вернее их коды в таблице ASCII, если они равны, сравниваются следующие и т.д.). Для сравнения строк используются функции **strcmp** и **stricmp**. Первая сравнивает строки с учетом регистра, вторая – без. Однако, все это относится только к латинице.

Прототипы этих функций таковы:

`int strcmp`(<первая строка>, <вторая строка>);

int **strcmp**(<первая строка>, <вторая строка>);

Обе функции возвращают число меньше 0, если первая строка меньше второй, больше нуля – если первая строка больше второй и 0, если строки лексикографически равны.

<p>Пример1:</p> <pre>#include <stdio.h> #include <stdlib.h> #include <string.h> int main() { char s1[]="Hello"; char s2[]="Hello"; int i; i=strcmp(s1,s2); printf("%s %s\n",s1,s2); if (i<0) printf("s1 < s2"); if (i>0) printf("s1 > s2"); if (i==0)printf("s1=s2"); return 0; }</pre> <p>На экране будет выведено:</p> <p>Hello Hello s1=s2</p>	<p>Пример 2:</p> <pre>#include <stdio.h> #include <stdlib.h> #include <string.h> int main() { char s1[]="Hello"; char s2[]="Hello, world"; int i; i=strcmp(s1,s2); printf("%s %s\n",s1,s2); if (i<0) printf("s1 < s2"); if (i>0) printf("s1 > s2"); if (i==0)printf("s1=s2"); return 0; }</pre> <p>На экране будет выведено:</p> <p>Hello Hello, world s1 < s2</p>	<p>Пример 3:</p> <pre>#include <stdio.h> #include <stdlib.h> #include <string.h> int main() { char s1[]="Bcda"; char s2[]="Abcd"; int i; i=strcmp(s1,s2); printf("%s %s\n",s1,s2); if (i<0) printf("s1 < s2"); if (i>0) printf("s1 > s2"); if (i==0)printf("s1=s2"); return 0; }</pre>
--	--	---

Длина строки

Для вычисления длины строки используется функция **strlen**(<строка>);

Пример:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main() {
    char s[]="Hello";
    int i;
    i=strlen(s);
    printf("%d",i);
    getch();
    return 0;
}
```

На экран будет выведено число 5

Преобразования строк

Зачастую требуется преобразовать число в строку и наоборот. Есть несколько способов сделать это.

Во-первых, можно использовать функции **sprintf** и **sscanf**. Например, так:

```
char str[50];
int i=15;
int j;
sprintf(str, "%d", i); // Записать в str строковое представление i
sscanf(str, "%d", &j); // Записать в j число, содержащееся в строке str
sprintf(str, "i=%d and j=%d", i, j);
// содержимое str: "i=15 and j=15"
```

Эти функции очень похожи на **printf** и **scanf**, за исключением того, что они работают не с консолью, а со строковым буфером.

Во-вторых, доступно целое семейство функций **atof**, **atoi**, **atol** и **itoa**, **ltoa**. Все они очень похожи между собой. Функции из первой группы преобразуют строку в число (**float**, **int** или **long**) в зависимости от окончания. Функции из второй группы выполняют обратное преобразование.

Прототипы функций из первой группы:

```
double atof(const char* string);
int atoi(const char* string);
long atol(const char* string);
```

Вторая группа:

```
char* itoa(int value, char* string, int radix);
char* ltoa(long value, char* string, int radix);
```

Функции возвращают указатель на строку.

Пример:

```
char str1[5];
char str2[5];
char str3[5];
itoa(12, str1, 10); //str1="12"
itoa(12, str1, 16); //str1="C"
itoa(12, str1, 2); //str1="1100"
```

Конкатенация (объединение) строк

Для конкатенации следует использовать функцию **strcat**(char* dest, const char* source)
Эта функция добавляет к строке, на которую указывает *dest*, символы из строки *source*.

Кроме того, можно воспользоваться общей функцией **sprintf** так:

```
char str1[]="Hello ";
char str2[]="world";
char str3[]="!";
char str4[13];
sprintf(str3, "%s%s%s", str1, str2, str3);
```

Обработка строк

Т.к. строка – это массив символов, то в ней, как и в любом массиве, можно обратиться к любому символу, зная его индекс в строке, учитывая, что нумерация символов начинается с нуля.

Например:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
    char s[]="Hello";
    printf("%c\n",s[4]);
    return 0;
}
```

На экран будет выведен символ о (т.к. он является 4 символом в строке Hello).

Также можно работать не со всей строкой, а начиная с любой позиции в строке, **например:**

```
char s[]="hello world";
printf("%s",&s[3]); // напечатать строку s, начиная с 3-го символа
```

на экране появится

```
lo world
```

Для обработки строк можно использовать следующие дополнительные функции модуля string.h:

1. При работе со строками может потребоваться поиск в строке заданной подстроки. Для выполнения такой операции в большинстве Си-компиляторов имеется функция **strstr**:

```
strstr(string, substring);
```

Если подстрока внутри строки существует, то функция **strstr** возвращает указатель на первое появление подстроки в строке. Если подстрока не найдена, то возвращается NULL.

Например, при выполнении следующего участка кода

```
char s[]="hello my friend";
char *ptr=strstr(s,"my");
printf("%s",ptr);
```

на экране будет выведено

```
my friend
```

2. Для нахождения номера позиции, с которой начинается заданная подстрока, можно воспользоваться

```
char s[]="hello";
printf("%d",strstr(s,"lo")-s); // на экране появится 3, т.к. последовательность lo
// начинается с 3-го символа строки hello
```

3. **char *strtok(char *s1, const char *s2);**

Функция **strtok** ищет в первой строке лексемы (наборы символов), которые разделены любым из символов, входящих во вторую строку. Повторный вызов с нулевым значением первого аргумента приведет к обработке всей строки *s1* до исчерпания всех лексем.

Например, при выполнении следующего участка кода

```
char s[]="hello my friend";
```

```

char *ptr;
ptr = strtok(s, " ");
printf("ptr = %s\n", ptr);
ptr = strtok(NULL, " ");
printf("ptr = %s\n", ptr);
ptr = strtok(NULL, " ");
printf("ptr = %s\n", ptr);

```

на экран будет выведено

```

ptr=hello
ptr=my
ptr=friend

```

4. strncpy(строка2, строка1, кол-во) – копирует из строки1 в строку2 нужное кол-во символов.

Пример 1: При выполнении следующего участка кода

```

char s[]="hello world";
char s1[8],s2[4];
strncpy(s1,s,5); // из строки s в строку s1 скопировать 5 символов
printf("%s",s1);

```

на экран будет выведено

```

hello

```

Пример 2: При выполнении следующего участка кода

```

char s[]="hello world";
char s1[8];
strncpy(s1,&s[3],5); // из строки s в строку s1 скопировать 5 символов,
                    //начиная с 3-го
printf("%s",s1);

```

на экран будет выведено

```

lo wo

```

Список задач

Простая обработка строк

1. Написать программу, которая удаляет из введенной с клавиатуры строки начальные пробелы.
2. Написать программу, которая удаляет из введенной с клавиатуры строки конечные пробелы.
3. Написать программу, которая проверяет, является ли введенная с клавиатуры строка целым числом. Рекомендательный вид экрана во время выполнения программы приведен ниже (данные, введенные пользователем, выделены полужирным шрифтом).
Введите число и нажмите <Enter>
-> **23.5**
Введенная строка не является целым числом.
4. Вывести, по одному разу, цифры, встречающиеся во введенной строке.
5. Дана строка. Найти буквы, встречающиеся в тексте больше двух раз.
6. Дана строка. Определить, больше в строке гласных или согласных букв.
7. Дана строка. Определить, больше в строке букв или цифр.
8. В текстовой строке подсчитать количество символов +, -, * и цифр.
9. Дана строка символов. Найти сумму цифр, встречающихся в строке.
10. Дана строка. Определить, сколько раз в ней встречается символ, введенный пользователем.
11. Дана строка. Определить, сколько в ней слов (разделяются пробелами).
12. Дана строка. Определить, сколько слов начинается с введенной пользователем буквы.
13. Дана строка. Определить, сколько слов заканчивается введенной пользователем буквой.
14. Дана строка символов, среди которых есть двоеточие (:). Определить, сколько символов ему предшествует.
15. Дана строка символов, среди которых есть двоеточие (:). Определить, сколько символов идет после него до конца строки.
16. Дана строка и буква. Подсчитать, сколько максимально раз подряд эта буква встречается в строке.
17. Выяснить, есть ли во введенной строке симметричное слово.
18. Дана строка символов, среди которых есть двоеточие (:). Подсчитать кол-во символов между двумя двоеточиями.
19. Дана строка. Определить, больше в строке строчных букв или заглавных.
20. Дана строка. Определить, больше в строке букв или других символов.

Сложные задачи на строки

1. Напишите программу, которая выводит на экран сообщение в "телеграфном" стиле: буквы сообщения должны появляться по одной, с некоторой задержкой.
2. Дана строка, слова разделены пробелами. Вывести на экран слова в обратном порядке.

3. Дана строка (слова разделены пробелами). Вывести на экран слово, в котором найдено искомое (вводится с клавиатуры) буквосочетание.
4. Вывести буквы, имеющиеся в тексте, в алфавитном порядке, по одному экземпляру. Например: "миру мир" - и р м у.
5. В строке найти слова (разделены пробелами), начинающиеся и заканчивающиеся одинаковыми знаками.
6. Дана строка, слова в которой разделены пробелами. Вывести на экран все слова, не включающие в себя цифры.
7. Дана строка. Найти в ней повторяющиеся слова.
8. Дана строка. Удалить из нее все пробелы.
9. Дана строка. Удвоить все встречающиеся пробелы.
10. Дана строка, содержащая текст. Найти длину самого короткого слова.
11. Дана строка, содержащая текст. Найти длину самого длинного слова.
12. Дана строка, содержащая текст. Найти кол-во слов, длина которых равна введенному пользователем значению.
13. Дана строка, среди символов которой есть двоеточия. Получить все символы, расположенные между первым и вторым двоеточиями. Если второго двоеточия нет, то получить все символы после первого двоеточия.
14. Удалить группу символов, расположенных между круглыми скобками, включая сами скобки.
15. Дана строка. Подсчитать самую длинную последовательность подряд идущих одинаковых букв.

Лабораторная работа №7. Структуры.

Теоретическая часть

Структуры в си – это объединенные данные, у которых есть некоторая логическая взаимосвязь. В отличие от массивов, структуры могут содержать данные разных типов. Вот пару примеров структур в си: структура класс (имя учащегося, буква класса, средний балл); структура футбольная команда (тренер, название команды, место в турнирной таблице). Теперь давайте рассмотрим, как описываются структуры в си:

```
struct klass {  
    char name[20];  
    char klass_name;  
    float bal;    };
```

Любая структура в языке си должна начинаться с ключевого слова – struct, которое сообщает компилятору, что тут у нас будет структура. Все данные в структуре (struct) пишутся в фигурных скобках, и в конце ставится запятая с точкой (;).

Данные в структуре должны иметь уникальные имена, но в различных структурах можно использовать одинаковые названия.

Структура, которая создана выше, не занимает в памяти компьютера места, так как мы, на самом деле, просто создали свой тип данных. Объявление структуры ни чем не отличается от объявления любого типа данных в языке си. Вот пример:

```
struct klass a, b[5], *c;
```

Мы объявили переменную a типа struct klass, массив b, состоящий из 5 элементов типа struct klass и указатель на переменную struct klass.

Так же можно объявлять переменные сразу после объявления структуры:

```
struct klass {  
    char name[20];  
    char klass_name;  
    float bal;  
} a, b[5], *c;
```

Операции со структурами:

- присваивание полю структуры значение того же типа
- можно получить адрес структуры (не забываем операцию взятия адреса (&)
- можно обращаться к любому полю структуры
- для того, что бы определить размер структуры можно использовать операцию sizeof()

Инициализация структуры

Инициализация структуры в языке си происходит так же, как и при инициализации массива. Вот пример инициализации структуры:

```
struct klass a = {"Sergey", 'B', 4.5 };
```

Т.е. мы создаем переменную типа struct klass и присваиваем всем трем полям, которые у нас определены в структуре, значения. Порядок очень важен при инициализации структуры, так как компьютер сам не может отсортировать данные. Если какое-либо поле у вас будет не заполненным, то оно автоматом заполнится 0 – для целочисленных типов; NULL – для указателей; \0 (ноль-терминатор) – для строковых типов.

Обращаться в языке си к элементам структуры можно двумя способами:

1. через операцию-точку

```
struct foot_klub {  
    char name[20];
```

```

int liga;
float ochki;};

int main(){
    struct foot_klub a = {"CSKA", 1, 24.5 };
    printf ("%d",a.liga); /* вывод значения элемента liga    структуры foot_klub */
    a.liga = 2; //присваиваем элементу новое значение
    return 0;  }

```

2. через операцию-стрелку

Обозначение: '->'. Различия заключаются в том, что операцию-точку мы используем, когда дело имеем с переменной, а операцию-стрелку – когда дело имеет с указателем на переменную. Вот пример:

```

struct foot_klub *ptr = &a;
printf ("%1f", ptr->ochki);

```

Не забывайте ставить знак взятия адреса (&) перед именем структуры, так как имя структуры не является указателем. Можно обратиться еще вот так:

```
printf ("%1f", (*ptr).ochki);
```

Но при таком обращении не забывайте брать в скобки (*ptr), т.к. операция доступа имеет более высокий приоритет по сравнению с операцией разыменования.

Передача структур в функцию

Раз структура является типом данных, только созданным нами, то с ней можно делать абсолютно такие же взаимодействия, как и с встроенными типами. Давайте рассмотрим работу структур (struct) с функциями подробнее:

структуру можно передавать целиком:

```

void f(struct foot_klub str){
    printf ("%s %d %1f", str.name, str.liga, str.ochki);  }

```

можно передать указатель на структуру (в случае, если изменяются какие-либо элементы структуры, то лучше использовать именно этот способ):

```

void d (struct foot_klub *p){
    p->ochki = 12; }

struct foot_klub myclub;
d(&myclub); // передаем указатель на нашу структуру

```

можно передать элементы структуры:

```

void e (int i, float f){
    printf ("liga: %d; ochki: %1f", i, f);  }

struct foot_klub a = {"CSKA", 1, 24.5 };
e(a.liga, a.ochki);

```

При передаче самой структуры или ее элементов передаются значения. Т.е. при изменении элементов структуры в функции, настоящие элементы останутся нетронутыми. Для того, чтобы при передаче мы передавали настоящие элементы, нужно передавать в функцию структуру или с помощью указателей или с помощью ссылок.

typedef

Возможно объявлять переменные структуры без специально слова struct. **typedef** позволяет объявить синоним нашей переменной. Вот пример использования ключевого слова **typedef** в языке си:

```

typedef struct foot_klub f_club;
f_club a, *p, b[5];

```

Т.е. мы заменили struct foot_klub синонимом f_club, что в разы облегчает понимание кода. Большинство программистов на языке си используют typedef сразу при объявлении структуры, что заметно уменьшает код. Вот пример:

```

typedef struct {
    char name[20];
    int liga;
    float ochki;
} f_club;

f_club a, *p, b[5];

```

Т.е. при объявлении вместе двух ключевых слов – typedef struct, мы сразу создаем тип f_club. Теперь для объявления переменной созданного типа достаточно всего-навсего воспользоваться одним словом.

Задача: Создать массив, элементами которого являются структуры – список учеников (фамилия, возраст, рост). Вывести информацию по всем ученикам, которые не ниже введенного пользователем роста.

Пример решения #1 (с заполнением структуры сразу при ее объявлении – фиксированное кол-во записей):

```

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

```

```

struct klass{    //объявляем тип данных klass как структуру из 3-х полей
    char fam[15]; // строковое поле
    float vozr;   // вещественное поле
    int rost;     //целое поле
};

int main (){
    int n,i,rost_min;

    // объявляем массив mas_struct из 3-х элементов типа нашей структуры klass
    //и сразу его заполняем
    struct klass mas_struct[3]={{"ivanov",13,176},{"ivanchenkova",15,180},{"petrova",16,169}};

    printf("\nНаша таблица:"); //делаем заголовок таблицы
    printf("\n  Фамилия | Возраст | Рост ");
    for (i=0;i<3;i++) // выводим данные в виде таблицы
        printf("\n%14s %8.1f %7d",mas_struct[i].fam,mas_struct[i].vozr,mas_struct[i].rost);

    printf("\n");
    printf("\nВведите значение роста:");
    scanf("%d",&rost_min);
    printf("\nВот данные, удовлетворяющие поиску:");

    for (i=0;i<3;i++) // ищем подходящих по росту и выводим их
        if(mas_struct[i].rost>=rost_min)
            printf("\n%14s %8.1f %7d",mas_struct[i].fam,mas_struct[i].vozr,mas_struct[i].rost);

    return 0;
}

```

Пример решения #2 (ввод данных в структуру с клавиатуры – произвольное количество записей):

```

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

struct klass{    //см. предыдущий пример
    char fam[15];
    float vozr;
    int rost;};

int main (){
    int n,i,rost_min;
    printf("\nВведите кол-во:");
    scanf("%d",&n); // вводим кол-во наших записей
    struct klass mas_struct[n]; // создаем массив mas_struct типа структуры klass
    for (i=0;i<n;i++) // в цикле вводим значения полей для каждой записи нашей структуры
    {
        printf("Введите фамилию:");
        scanf("%s",mas_struct[i].fam);
        printf("Введите возраст:");
        scanf("%f",&mas_struct[i].vozr);
        printf("Введите рост:");
        scanf("%d",&mas_struct[i].rost);
    }

    printf("\nНаша таблица:"); // далее все как в предыдущем примере
    printf("\n  Фамилия | Возраст | Рост ");
    for (i=0;i<n;i++)
        printf("\n%14s %8.1f %7d",mas_struct[i].fam,mas_struct[i].vozr,mas_struct[i].rost);
    printf("\n");
    printf("\nВведите значение роста:");
    scanf("%d",&rost_min);
    printf("\nВот данные, удовлетворяющие поиску:");
    for (i=0;i<n;i++)
        if (mas_struct[i].rost>=rost_min)
            printf("\n%14s %8.1f %7d",mas_struct[i].fam,mas_struct[i].vozr,mas_struct[i].rost);

    return 0;
}

```

Пример решения #3 (с использованием typedef):

```

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

typedef struct klass{

```

```

    char fam[15];
    float vozr;
    int rost;
}klass;

int main ()
{
    int n,i,rost_min;
    printf("\nВведите кол-во:");
    scanf("%d",&n);
    klass mas_struct[n];
    for (i=0;i<n;i++)
    {
        printf("Введите фамилию:");
        scanf("%s",mas_struct[i].fam);
        printf("Введите возраст:");
        scanf("%f",&mas_struct[i].vozt);
        printf("Введите рост:");
        scanf("%d",&mas_struct[i].rost);
    }

    printf("\nНаша таблица:");
    printf("\n Фамилия | Возраст | Рост ");
    for (i=0;i<n;i++)
        printf("\n%14s %8.1f %7d",mas_struct[i].fam,mas_struct[i].vozt,mas_struct[i].rost);
    printf("\n");
    printf("\nВведите значение роста:");
    scanf("%d",&rost_min);
    printf("\nВот данные, удовлетворяющие поиску:");

    for (i=0;i<n;i++)
        if (mas_struct[i].rost>=rost_min)
            printf("\n%14s %8.1f %7d",mas_struct[i].fam,mas_struct[i].vozt,mas_struct[i].rost);

    return 0;
}

```

Список задач

Часть 1.

1. Создать массив, элементами которого являются структуры – список учеников (хранятся фамилия, класс, школа). Вывести информацию по ученикам, фамилии которых начинаются на букву, введенную пользователем.
2. Создать массив, элементами которого являются структуры – список спортсменов (фамилия, вид спорта, спортивное общество). Вывести информацию по тем из них, кто занимается введенным пользователем видом спорта.
3. Создать массив, элементами которого являются структуры – список учеников класса (хранятся фамилия, оценки по 3 предметам). Вывести средний балл учеников класса по предмету, введенному пользователем. Вывести учеников, имеющих средний балл выше среднего в классе.
4. Создать массив, элементами которого являются структуры – список ассортимента конфет, выпускаемых кондитерской фабрикой (название, цена за кг, срок годности). Выбрать те, стоимость которых находится в пределах, введенных пользователем. Найти конфеты с наибольшим сроком годности.
5. Создать массив, элементами которого являются структуры – список учеников музыкальной школы (фамилия, инструмент, год обучения). Вывести информацию по ученикам, играющим на инструменте, введенном пользователем.
6. Создать массив, элементами которого являются структуры – список работников фирмы (фамилия, должность, стаж, зарплата). Вывести информацию по работникам с зарплатой, не меньше введенной пользователем, и стажем, не больше введенного пользователем значения.
7. Создать массив, элементами которого являются структуры – список детей детского сада (фамилия, группа, возраст, месяц рождения). Вывести информацию по детям данного детского сада, которые родились в месяце, введенном пользователем.
8. Создать массив, элементами которого являются структуры – список учителей школы (фамилия, предмет, стаж). Вывести информацию по всем учителям, которые преподают предмет, введенный пользователем.
9. Создать массив, элементами которого являются структуры – список участников олимпиады (фамилия, класс, предмет, место). Вывести информацию по участникам, занявшим определенное место.
10. Создать массив, элементами которого являются структуры – список стран (название, площадь, столица, население). Вывести информацию по странам с населением, меньшим введенного пользователем значения.
11. Создать массив, элементами которого являются структуры – ведомость по сессии (фамилия, факультет, оценки за экзамены (минимум 3 предмета)). Вывести информацию по студентам, имеющим среднюю оценку не ниже введенной пользователем.
12. Создать массив, элементами которого являются структуры – список пациентов больницы (фамилия, номер паспорта, болезнь, фамилия лечащего врача). Вывести информацию по всем пациентам введенного пользователем лечащего врача.
13. Создать массив, элементами которого являются структуры – список клиентов автосервиса (фамилия, марка автомобиля, дата обслуживания). Вывести информацию по клиентам с определенной маркой автомобиля.
14. Создать массив, элементами которого являются структуры – список сеансов кинотеатра (название фильма, время начала, длительность, жанр). Вывести информацию по всем фильмам определенного жанра.

15. Создать массив, элементами которого являются структуры – список лекарств аптеки (название, цена, форм-фактор (таблетки, мазь, порошок и т.д.), фирма-производитель). Вывести информацию по всем лекарствам конкретной фирмы.

Часть 2.

1. Создать массив, элементами которого являются структуры – список фирм (название, сфера деятельности, фамилия директора). Создать функцию, сортирующую данную структуру (например, по фамилии), и функцию для поиска нужного значения в структуре. Использовать эти функции по назначению в программе (для вывода отсортированной информации и поиска нужной информации).
2. Создать массив, элементами которого являются структуры – список факультетов НГГУ (название, фамилия декана, число студентов). Создать функцию, сортирующую данную структуру (например, по числу студентов), и функцию для изменения нужного значения в структуре. Использовать эти функции по назначению в программе (для вывода отсортированной информации и изменения нужной информации).
3. Создать массив, элементами которого являются структуры – список участников конференции (фамилия, откуда, номер телефона). Создать функцию, сортирующую данную структуру (например, по фамилии), и функцию для поиска нужного значения в структуре. Использовать эти функции по назначению в программе (для вывода отсортированной информации и поиска нужной информации).
4. Создать массив, элементами которого являются структуры – список стартовой пятерки баскетбольной команды (фамилия, рост, амплуа (нападающий, защитник, центровый), номер). Создать функцию, сортирующую данную структуру (например, по росту), и функцию для изменения игрока под определенным номером. Использовать эти функции по назначению в программе (для вывода отсортированной информации и изменения нужной информации).
5. Создать массив, элементами которого являются структуры – список учета книг в библиотеке (автор, название, кто взял, когда). Создать функцию, сортирующую данную структуру (например, по автору), и функцию для поиска нужного значения в структуре. Использовать эти функции по назначению в программе (для вывода отсортированной информации и поиска нужной информации).
6. Создать массив, элементами которого являются структуры – список телефонов (номер, фамилия абонента, адрес). Создать функцию, сортирующую данную структуру (например, по фамилии), и функцию для изменения информации определенного номера. Использовать эти функции по назначению в программе (для вывода отсортированной информации и изменения нужной информации).
7. Создать массив, элементами которого являются структуры – список постояльцев отеля (фамилия, дата заезда, дата выезда, номер апартаментов). Создать функцию, сортирующую данную структуру (например, по дате заезда), и функцию для поиска нужного значения в структуре. Использовать эти функции по назначению в программе (для вывода отсортированной информации и поиска нужной информации).
8. Создать массив, элементами которого являются структуры – список клиентов фитнес-клуба (фамилия, частота посещений в неделю, личный тренер). Создать функцию, сортирующую данную структуру (например, по частоте посещений в неделю), и функцию для изменения информации клиента. Использовать эти функции по назначению в программе (для вывода отсортированной информации и изменения нужной информации).
9. Создать массив, элементами которого являются структуры – список посетителей курсов (фамилия, название курса, длительность курса). Создать функцию, сортирующую данную структуру (например, по длительности), и функцию для поиска нужного значения в структуре. Использовать эти функции по назначению в программе (для вывода отсортированной информации и поиска нужной информации).
10. Создать массив, элементами которого являются структуры – список школ (номер школы, фамилия директора, кол-во учащихся, кол-во учителей). Создать функцию, сортирующую данную структуру (например, по кол-ву учащихся), и функцию для изменения информации конкретной школы. Использовать эти функции по назначению в программе (для вывода отсортированной информации и изменения нужной информации).
11. Создать массив, элементами которого являются структуры – список отелей (название, кол-во звезд, кол-во мест, фамилия метрдотеля). Создать функцию, сортирующую данную структуру (например, по длительности), и функцию для поиска нужного значения в структуре. Использовать эти функции по назначению в программе (для вывода отсортированной информации и поиска нужной информации).
12. Создать массив, элементами которого являются структуры – список ресторанов (название, фамилия шеф-повара, основная кухня, кол-во столиков). Создать функцию, сортирующую данную структуру (например, по кол-ву столиков), и функцию для изменения информации конкретного ресторана. Использовать эти функции по назначению в программе (для вывода отсортированной информации и изменения нужной информации).
13. Создать массив, элементами которого являются структуры – список журналов и газет (название, тираж, фамилия главреда, год основания). Создать функцию, сортирующую данную структуру (например, по тиражу), и функцию для поиска нужного значения в структуре. Использовать эти функции по назначению в программе (для вывода отсортированной информации и поиска нужной информации).
14. Создать массив, элементами которого являются структуры – заказы магазина (фамилия клиента, товар, стоимость, адрес). Создать функцию, сортирующую данную структуру (например, по стоимости), и функцию для изменения информации конкретного заказа. Использовать эти функции по назначению в программе (для вывода отсортированной информации и изменения нужной информации).

Лабораторная работа №8. Файлы.

Теоретическая часть

Для работы с файлами используется тип данных FILE, т.е. нужно объявить переменную-указатель типа FILE.

Например:

FILE *f;

Открытие файла: функция `fopen`

Прототип функции открытия файла выглядит следующим образом:

```
FILE *fopen(const char *path, const char *mode);
```

Здесь `path` – путь к файлу (например, имя файла или абсолютный путь к файлу), `mode` - режим открытия файла. Строка `mode` может содержать несколько букв. Буква "r" (от слова `read`) означает, что файл открывается для чтения (файл должен существовать). Буква "w" (от слова `write`) означает запись в файл, при этом старое содержимое файла теряется, а в случае отсутствия файла он создается. Буква "a" (от слова `append`) означает запись в конец существующего файла или создание нового файла, если файл не существует.

Кроме того, при открытии можно разрешить выполнять как операции чтения, так и записи; для этого используется символ + (плюс). Порядок букв в строке `mode` следующий: сначала идет одна из букв "r", "w", "a", затем в произвольном порядке могут идти символы "b", "t", "+". Буквы "b" и "t" можно использовать, даже если в операционной системе нет различий между бинарными и текстовыми файлами, в этом случае они просто игнорируются.

Значения символов в строке `mode` сведены в следующую таблицу:

r	Открыть существующий файл на чтение
w	Открыть файл на запись. Старое содержимое файла теряется, в случае отсутствия файла он создаётся.
a	Открыть файл на запись. Если файл существует, то запись производится в его конец.
t	Открыть текстовый файл.
b	Открыть бинарный файл.
+	Разрешить и чтение, и запись.

Несколько примеров открытия файлов:

```
FILE *f, *g, *h;
```

```
...
```

```
// 1. Открыть текстовый файл "abcd.txt" для чтения
```

```
f = fopen("abcd.txt", "rt");
```

```
// 2. Открыть бинарный файл "c:\Windows\Temp\tmp.dat"
```

```
// для чтения и записи
```

```
g = fopen("c:/Windows/Temp/tmp.dat", "wb+");
```

```
// 3. Открыть текстовый файл "c:\Windows\Temp\abcd.log"
```

```
// для дописывания в конец файла
```

```
h = fopen("c:\\Windows\\Temp\\abcd.log", "at");
```

В случае удачи функция `fopen` открытия файла возвращает ненулевой указатель на структуру типа `FILE`, описывающую параметры открытого файла. Этот указатель надо затем использовать во всех файловых операциях. В случае неудачи (например, при попытке открыть на чтение несуществующий файл) возвращается нулевой указатель. При этом глобальная системная переменная `errno`, описанная в стандартном заголовочном файле `errno.h`, содержит численный код ошибки. В случае неудачи при открытии файла этот код можно распечатать, чтобы получить дополнительную информацию:

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

```
#include <errno.h>
```

```
...
```

```
FILE *f = fopen("filnam.txt", "rt");
```

```
if (f == NULL) {
```

```
    printf("Ошибка открытия файла с кодом %d\n", errno);
```

```
    ...
```

```
}
```

Диагностика ошибок: функция `perror`

Использовать переменную `errno` для печати кода ошибки не очень удобно, поскольку необходимо иметь под рукой таблицу возможных кодов ошибок и их значений. В стандартной библиотеке Си существует более удобная функция `perror`, которая печатает системное сообщение о последней ошибке вместо ее кода. Печать производится на английском языке, но есть возможность добавить к системному сообщению любой текст, который указывается в качестве единственного аргумента функции `perror`. Например, предыдущий фрагмент переписывается следующим образом:

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

```
...
```

```
FILE *f = fopen("filnam.txt", "rt");
```

```
if (f == 0) {
    perror("Не могу открыть файл на чтение");
    ...
}
```

Функция `perror` печатает сначала пользовательское сообщение об ошибке, затем после двоеточия системное сообщение. Например, при выполнении приведенного фрагмента в случае ошибки из-за отсутствия файла будет напечатано
Не могу открыть файл на чтение: No such file or directory

Функции бинарного чтения и записи `fread` и `fwrite`

После того как файл открыт, можно читать информацию из файла или записывать информацию в файл. Рассмотрим сначала функции бинарного чтения и записи `fread` и `fwrite`. Они называются бинарными потому, что не выполняют никакого преобразования информации при вводе или выводе (с одним небольшим исключением при работе с текстовыми файлами, которое будет рассмотрено ниже): информация хранится в файле как последовательность байтов ровно в том виде, в котором она хранится в памяти компьютера.

Функция чтения `fread` имеет следующий прототип:

```
size_t fread(
    char *buffer, // Массив для чтения данных
    size_t elemSize, // Размер одного элемента
    size_t numElems, // Число элементов для чтения
    FILE *f // Указатель на структуру FILE
);
```

Здесь `size_t` определен как беззнаковый целый тип в системных заголовочных файлах. Функция пытается прочесть `numElems` элементов из файла, который задается указателем `f` на структуру `FILE`, размер каждого элемента равен `elemSize`. Функция возвращает реальное число прочитанных элементов, которое может быть меньше, чем `numElems`, в случае конца файла или ошибки чтения. Указатель `f` должен быть возвращен функцией `fopen` в результате успешного открытия файла.

Пример использования функции `fread`:

```
FILE *f;
double buff[100];
size_t res;

f = fopen("tmp.dat", "rb"); // Открываем файл
if (f == 0) { // При ошибке открытия файла
    // Напечатать сообщение об ошибке
    perror("Не могу открыть файл для чтения");
    exit(1); // завершить работу с кодом 1
}

// Пытаемся прочесть 100 вещественных чисел из файла
res = fread(buff, sizeof(double), 100, f);
// res равно реальному количеству прочитанных чисел
```

В этом примере файл `"tmp.dat"` открывается на чтение как бинарный, из него читается 100 вещественных чисел размером 8 байт каждое. Функция `fread` возвращает реальное количество прочитанных чисел, которое меньше или равно, чем 100.

Функция бинарной записи в файл `fwrite` аналогична функции чтения `fread`. Она имеет следующий прототип:

```
size_t fwrite(
    char *buffer, // Массив записываемых данных
    size_t elemSize, // Размер одного элемента
    size_t numElems, // Число записываемых элементов
    FILE *f // Указатель на структуру FILE
);
```

Функция возвращает число реально записанных элементов, которое может быть меньше, чем `numElems`, если при записи произошла ошибка – например, не хватило свободного пространства на диске. Пример использования функции `fwrite`:

```
FILE *f;
double buff[100];
size_t num;
...
f = fopen("tmp.res", "wb"); // Открываем файл "tmp.res"
if (f == 0) { // При ошибке открытия файла
    // Напечатать сообщение об ошибке
}
```

```

    perror("Не могу открыть файл для записи");
    exit(1); // завершить работу программы с кодом 1
}

```

```

// Записываем 100 вещественных чисел в файл
res = fwrite(buff, sizeof(double), 100, f);
// В случае успеха res == 100
Заккрытие файла: функция fclose

```

По окончании работы с файлом его надо обязательно закрыть. Система обычно запрещает полный доступ к файлу до тех пор, пока он не закрыт. (Например, в нормальном режиме система запрещает одновременную запись в файл для двух разных программ.) Кроме того, информация реально записывается полностью в файл лишь в момент его закрытия. До этого она может содержаться в оперативной памяти (в так называемой файловой кеш-памяти), что при выполнении многочисленных операций записи и чтения значительно ускоряет работу программы.

Для закрытия файла используется функция `fclose` с прототипом

```
int fclose(FILE *f);
```

В случае успеха функция `fclose` возвращает ноль, при ошибке отрицательное значение (точнее, константу конец файла EOF, определенную в системных заголовочных файлах как минус единица). При ошибке можно воспользоваться функцией `perror`, чтобы напечатать причину ошибки. Отметим, что ошибка при закрытии файла – явление очень редкое (чего не скажешь в отношении открытия файла), так что анализировать значение, возвращаемое функцией `fclose`, в общем-то, не обязательно. Пример использования функции `fclose`:

```
FILE *f;
```

```

f = fopen("tmp.res", "wb"); // Открываем файл "tmp.res"
if (f == 0) { // При ошибке открытия файла
    // Напечатать сообщение об ошибке
    perror("Не могу открыть файл для записи");
    exit(1); // завершить работу программы с кодом 1
}
...

```

```

// Закрываем файл
if (fclose(f) < 0) {
    // Напечатать сообщение об ошибке
    perror("Ошибка при закрытии файла");
}

```

Пример: подсчет числа символов и строк в текстовом файле

В качестве содержательного примера использования рассмотренных выше функций файлового ввода приведем программу, которая подсчитывает число символов и строк в текстовом файле. Программа сначала вводит имя файла с клавиатуры. Для этого используется функция `scanf` ввода по формату из входного потока, для ввода строки применяется формат `%s`. Затем файл открывается на чтение как бинарный (это означает, что при чтении не будет происходить никакого преобразования разделителей строк). Используя в цикле функцию чтения `fread`, мы считываем содержимое файла порциями по 512 байтов, каждый раз увеличивая суммарное число прочитанных символов. После чтения очередной порции сканируется массив прочитанных символов и подсчитывается число символов `"\n"`. В конце закрывается файл и печатается результат.

// Файл "wc.cpp" Подсчет числа символов и строк в текстовом файле

```

#include <stdio.h> // Описания функций ввода-вывода
#include <stdlib.h> // Описание функции exit

int main()
{
    char fileName[256]; // Путь к файлу
    FILE *f;           // Структура, описывающая файл
    char buff[512];     // Массив для ввода символов
    size_t num;         // Число прочитанных символов
    int numChars = 0;   // Суммарное число символов := 0
    int numLines = 0;   // Суммарное число строк := 0
    int i;              // Переменная цикла

    printf("Введите имя файла: ");
    scanf("%s", fileName);

    f = fopen(fileName, "rb"); // Открываем файл на чтение
    if (f == 0) { // При ошибке открытия файла
        // Напечатать сообщение об ошибке
        perror("Не могу открыть файл для чтения");
    }
}

```

```

    exit(1); // закончить работу программы с кодом 1 ошибочного завершения
}
while ((num = fread(buff, 1, 512, f)) > 0) { // Читаем
    // блок из 512 символов. num -- число реально
    // прочитанных символов. Цикл продолжается, пока
    // num > 0

    numChars += num; // Увеличиваем число символов

    // Подсчитываем число символов перевода строки
    for (i = 0; i < num; ++i) {
        if (buff[i] == '\n') {
            ++numLines; // Увеличиваем число строк
        }
    }
}
fclose(f);
// Печатаем результат
printf("Число символов в файле = %d\n", numChars);
printf("Число строк в файле = %d\n", numLines);
return 0; // Возвращаем код успешного завершения
}

```

Пример выполнения программы: она применяется к собственному тексту, записанному в файле "wc.cpp".

Введите имя файла: wc.cpp

Число символов в файле = 1635 // у вас может быть др. результат

Число строк в файле = 50 // у вас может быть др. результат

Форматный ввод-вывод: функции fscanf и fprintf

В отличие от функции бинарного ввода fread, которая вводит байты из файла без всякого преобразования непосредственно в память компьютера, функция форматного ввода fscanf предназначена для ввода информации с преобразованием ее из текстового представления в бинарное. Пусть информация записана в текстовом файле в привычном для человека виде (т.е. так, что ее можно прочитать или ввести в файл, используя текстовый редактор). Функция fscanf читает информацию из текстового файла и преобразует ее во внутреннее представление данных в памяти компьютера. Информация о количестве читаемых элементов, их типах и особенностях представления задается с помощью формата. В случае функции ввода формат – это строка, содержащая описания одного или нескольких вводимых элементов. Форматы, используемые функцией fscanf, аналогичны применяемым функцией scanf. Каждый элемент формата начинается с символа процента "%". Наиболее часто используемые при вводе форматы приведены в таблице:

%d	целое десятичное число типа int (d - от decimal)
%f	вещ. число типа float (f - от float)
%lf	вещ. число типа double (lf - от long float)
%c	один символ типа char
%s	ввод строки. Из входного потока выделяется слово, ограниченное пробелами или символами перевода строки '\n'. Слово помещается в массив символов. Конец слова отмечается нулевым байтом.

Прототип функции fscanf выглядит следующим образом:

```
int fscanf(FILE *f, const char *format, ...);
```

Многоточие здесь означает, что функция имеет переменное число аргументов, большее двух, и что количество и типы аргументов, начиная с третьего, произвольны. На самом деле, фактические аргументы, начиная с третьего, должны быть указателями на вводимые переменные. Несколько примеров использования функции fscanf:

```
int n, m; double a; char c; char str[256];
```

```
FILE *f;
```

```
...
```

```
fscanf(f, "%d", &n); // Ввод целого числа
```

```
fscanf(f, "%lf", &a); // Ввод вещественного числа
```

```
fscanf(f, "%c", &c); // Ввод одного символа
```

```
fscanf(f, "%s", str); // Ввод строки (выделяется очередное
// слово из входного потока)
```

```
fscanf(f, "%d%d", &n, &m); // Ввод двух целых чисел
```

Функция fscanf возвращает число успешно введенных элементов. Таким образом, возвращаемое значение всегда меньше или равно количеству процентов внутри форматной строки (которое равно числу фактических аргументов минус 2).

Функция fprintf используется для форматного вывода в файл. Данные при выводе преобразуются в их текстовое представление в соответствии с форматной строкой. Ее отличие от форматной строки, используемой в функции ввода fscanf,

заключается в том, что она может содержать не только форматы для преобразования данных, но и обычные символы, которые записываются без преобразования в файл. Форматы, как и в случае функции `fscanf`, начинаются с символа процента "%". Они аналогичны форматам, используемым функцией `fscanf`. Небольшое отличие заключается в том, что форматы функции `fprintf` позволяют также управлять представлением данных, например, указывать количество позиций, отводимых под запись числа, или количество цифр после десятичной точки при выводе вещественного числа.

Прототип функции `fprintf` выглядит следующим образом:

```
int fprintf(FILE *f, const char *format, ...);
```

Многоточие, как и в случае функции `fscanf`, означает, что функция имеет переменное число аргументов. Количество и типы аргументов, начиная с третьего, должны соответствовать форматной строке. В отличие от функции `fscanf`, фактические аргументы, начиная с третьего, представляют собой выводимые значения, а не указатели на переменные.

Пример работы с текстовыми файлами: рассмотрим небольшую программу, выводящую данные в файл "tmp.dat":

```
#include <stdio.h> // Описания функций ввода вывода
#include <math.h>  // Описания математических функций
#include <string.h> // Описания функций работы со строками

int main() {
    int n = 4, m = 6; double x = 2.;
    char str[256] = "Print test";
    FILE *f = fopen("tmp.dat", "wt"); // Открыть файл
    if (f == 0) {                     // для записи
        perror("Не могу открыть файл для записи");
        return 1; // Завершить программу с кодом ошибки
    }
    fprintf(f, "n=%d, m=%d\n", m, n);
    fprintf(f, "x=%.4lf, sqrt(x)=%.4lf\n", x, sqrt(x));
    fprintf(f, "Строка \"%s\" содержит %d символов.\n", str, strlen(str));
};
fclose(f); // Закрыть файл
return 0;  // Успешное завершение программы
}
```

В результате выполнения этой программы в файл "tmp.dat" будет записан следующий текст:

n=6, m=4

x=2.0000, sqrt(x)=1.4142

Строка "Print test" содержит 10 символов.

В последнем примере форматная строка содержит внутри себя двойные апострофы. Это специальные символы, выполняющие роль ограничителей строки, поэтому внутри строки их надо экранировать (т.е. защищать от интерпретации как специальных символов) с помощью обратной косой черты \, которая, напомним, в системе Unix и в языке Си выполняет роль защитного символа. Отметим также, что мы воспользовались стандартной функцией `sqrt`, вычисляющей квадратный корень числа, и стандартной функцией `strlen`, вычисляющей длину строки.

Другие полезные функции ввода-вывода в файл

Стандартная библиотека ввода-вывода Си содержит ряд других полезных функций ввода-вывода. Отметим некоторые из них.

Посимвольный ввод-вывод

```
int fgetc(FILE *f); // ввести символ из потока f
int fputc(int c, FILE *f); // вывести символ в поток f
```

Построчковый ввод-вывод

```
char *fgets(char *line, int size, FILE *f); // ввести строку из потока f
char *fputs(char *line, FILE *f); // вывести строку в поток f
```

Позиционирование в файле

```
int fseek(FILE *f, long offset, int whence); // установить текущую позицию в файле f
long ftell(FILE *f); // получить текущую позицию в файле f
int feof(FILE *f); // проверить, достигнут ли конец файла f
```

Функция `fgetc` возвращает код введенного символа или константу EOF (определенную как минус единицу) в случае конца файла или ошибки чтения. Функция `fputc` записывает один символ в файл. При ошибке `fputc` возвращает константу EOF (т.е. отрицательное значение), в случае удачи - код выведенного символа с (неотрицательное значение).

Пример использования функции `fgetc`: перепишем рассмотренную ранее программу `ws`, подсчитывающую число символов и строк в текстовом файле:

```
// Файл "wc1.cpp"
// Подсчет числа символов и строк в текстовом файле
// с использованием функции чтения символа fgetc
#include <stdio.h> // Описания функций ввода-вывода
```

```
#include <stdlib.h>

int main() {
    char fileName[256]; // Путь к файлу
    FILE *f;            // Структура, описывающая файл
    int c;              // Код введенного символа
    int numChars = 0;    // Суммарное число символов := 0
    int numLines = 0;    // Суммарное число строк := 0
    printf("Введите имя файла: ");
    scanf("%s", fileName);
    f = fopen(fileName, "rb"); // Открываем файл
    if (f == 0) { // При ошибке открытия файла
        // Напечатать сообщение об ошибке
        perror("Не могу открыть файл для чтения");
        return 1; // закончить работу программы с кодом 1
    }
    while ((c = fgetc(f)) != EOF) { // Читаем символ
        // Цикл продолжается, пока c != -1 (конец файла)

        ++numChars; // Увеличиваем число символов

        // Подсчитываем число символов перевода строки
        if (c == '\n') {
            ++numLines; // Увеличиваем число строк
        }
    }
    fclose(f);
    // Печатаем результат
    printf("Число символов в файле = %d\n", numChars);
    printf("Число строк в файле = %d\n", numLines);
    return 0; // Возвращаем код успешного завершения
}
```

Пример выполнения программы `ws1` в применении к собственному тексту, записанному в файле `ws1.cpp`:

Введите имя файла: `ws1.cpp`

Число символов в файле = 1334

Число строк в файле = 44

Функция `fgets` с прототипом `char *fgets(char *line, int size, FILE *f);`

выделяет из файла или входного потока `f` очередную строку и записывает ее в массив символов `line`. Второй аргумент `size` указывает размер массива для записи строки. Максимальная длина строки на единицу меньше, чем `size`, поскольку всегда в конец считанной строки добавляется нулевой байт. Функция сканирует входной поток до тех пор, пока не встретит символ перевода строки `"\n"` или пока число введенных символов не станет равным `size - 1`. Символ перевода строки `"\n"` также записывается в массив непосредственно перед терминирующим нулевым байтом. Функция возвращает указатель `line` в случае успеха или нулевой указатель при ошибке или конце файла.

При выполнении файловых операций исполняющая система поддерживает указатель текущей позиции в файле. При чтении или записи `n` байтов указатель текущей позиции увеличивается на `n`; таким образом, чтение или запись происходят последовательно. Библиотека ввода-вывода Си предоставляет, однако, возможность нарушать эту последовательность путем позиционирования в произвольную точку файла. Для этого используется стандартная функция `fseek` с прототипом

```
int fseek(FILE *f, long offset, int whence);
```

Первый аргумент `f` функции определяет файл, для которого производится операция позиционирования. Второй аргумент `offset` задает смещение в байтах, оно может быть как положительным, так и отрицательным (его лучше высчитывать с учетом размера типа данных). Третий аргумент `whence` указывает, откуда отсчитывать смещение. Он может принимать одно из трех значений, заданных как целые константы в стандартном заголовочном файле `"stdio.h"`:

<code>SEEK_CUR</code>	смещение отсчитывается от текущей позиции
<code>SEEK_SET</code>	смещение отсчитывается от начала файла
<code>SEEK_END</code>	смещение отсчитывается от конца файла

Например:

```
fseek(f, 0, SEEK_SET); // устанавливает текущую позицию в начало файла.
```

```
fseek(f, -4*sizeof(int), SEEK_END); //устанавливает текущую позицию в четвертую запись перед концом
//файла (в файле – целые числа).
```

```
fseek(f, 12*sizeof(float), SEEK_CUR); //устанавливает текущую позицию в двенадцатую запись
//относительно текущего положения (в файле – вещественные числа).
```

Отметим, что смещение может быть положительным даже при использовании константы `SEEK_END` (т.е. при позиционировании относительно конца файла): в этом случае при следующей записи размер файла соответственно увеличивается.

Функция возвращает нулевое значение в случае успеха и отрицательное значение EOF (равное -1) при неудаче – например, если указанное смещение некорректно при заданной операции или если файл или поток не позволяет выполнять прямое позиционирование.

Узнать текущую позицию относительно начала файла можно с помощью функции ftell с прототипом

```
long ftell(FILE *f);
```

Функция ftell возвращает текущую позицию (неотрицательное значение) в случае успеха или отрицательное значение -1 при неудаче (например, если файл не разрешает прямое позиционирование).

Наконец, узнать, находится ли текущая позиция в конце файла, можно с помощью функции feof с прототипом

```
int feof(FILE *f);
```

Она возвращает ненулевое значение (т.е. истину), если конец файла достигнут, и нулевое значение (т.е. ложь) в противном случае. Например, в следующем фрагменте в цикле проверяется, достигнут ли конец файла, и, если нет, считывается очередной байт:

```
FILE *f;
...
while (!feof(f)) { // цикл пока не конец файла
    int c = fgetc(f); // прочесть очередной байт
    ... // ...
} // конец цикла
```

Пример работы с бинарным файлом — запись вещественных чисел в файл и чтение их оттуда:

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

int main()
{
    srand(time(NULL));
    int i;
    FILE *f = fopen("tmp.dat", "wb"); // Открыть файл для записи
    if (f == NULL) {
        perror("Не могу открыть файл для записи");
        return 1; // Завершить программу с кодом ошибки
    }
    float a[10], al[10];
    for (i=0; i<10; i++) a[i]=(float)(rand()%1000)/100; // генерация вещественных чисел с точностью
                                                         //до сотого в интервале от 0 до 10
    for (i=0; i<10; i++) printf(" %.2f ",a[i]); // вывод на экран сгенерированных чисел
    printf("\n");
    int res = fwrite(a, sizeof(float), 10, f); // Записываем 10 вещественных чисел в файл
    fclose(f); // Закрыть файл
    f = fopen("tmp.dat", "rb"); // Открыть файл для чтения
    if (f == 0) {
        perror("Не могу открыть файл для чтения");
        return 1; // Завершить программу с кодом ошибки
    }
    fseek(f, -2 * sizeof ( float ), SEEK_END); // установить позицию на 2 с конца запись
    res = fread(al, sizeof ( float ), 2, f); // считать из файла 2 записи типа float
    for (i=0; i<2; i++) printf(" %.2f ",al[i]); // выводим на экран, что считали из файла
    fclose(f); // Закрыть файл
    getch();
    return 0;
}
```

В ходе выполнения на экране появится следующее

1.17 9.99 7.90 8.95 3.21 4.16 0.41 6.41 1.31 3.55

1.31 3.55

В первой строке – все числа, которые были сгенерированы и записаны в файл tmp.dat, во второй – считанные последние 2 числа из файла tmp.dat.

Пример работы с бинарным файлом — запись массива из структур в файл и чтение из него:

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

typedef struct { // создаем структуру
    float q1;
    int q2;
} mystruc;
```



```

int main()
{
    srand(time(NULL));
    int i;
    FILE *f = fopen("tmp.dat", "wb"); // Открыть файл для записи
    if (f == NULL) {
        perror("Не могу открыть файл для записи");
        return 1; // Завершить программу с кодом ошибки
    }
    mystruc buff[10], buff1[10];
    for (i=0; i<10; i++)
    {
        buff[i].q1=(float)(rand()%100)/10; // генерация вещественных чисел с точностью до 1 знака
                                           // после запятой в интервале от 0 до 10
        buff[i].q2=i; // заполняем номером элемента
    }
    for (i=0; i<5; i++) printf(" %.1f %d \n",buff[i].q1, buff[i].q2);
    printf("\n");
    int res = fwrite(buff, sizeof(mystruc), 5, f); // Записываем 5 элементов типа созданной
                                                    //структуры в файл
    fclose(f); // Закрыть файл
    f = fopen("tmp.dat", "rb"); // Открыть файл для чтения
    if (f == 0) {
        perror("Не могу открыть файл для записи");
        return 1; // Завершить программу с кодом ошибки
    }
    fseek(f, 2 * sizeof ( mystruc ), SEEK_SET); // установить позицию в файле на 2-ю запись
    res = fread(buff1, sizeof ( mystruc ), 3, f); // считать из файла 3 записи в массив buff1
    for (i=0; i<3; i++) printf(" %.1f %d \n",buff1[i].q1, buff1[i].q2); // Выводим считанное на экран
    fclose(f); // Закрыть файл
    return 0;
}

```

В ходе выполнения программы на экране появится примерно следующее:

```

3.6 0
1.1 1
2.7 2
1.9 3
0.5 4

2.7 2
1.9 3
0.5 4

```

Список задач

Часть 1. Текстовые файлы.

1. Дан файл, содержащий текст. Выяснить, входит ли данное слово в указанный текст, и если да, то сколько раз.
2. Дан файл, содержащий произвольный текст. Выяснить, чего в нем больше: букв или цифр.
3. Дан файл, содержащий текст. Провести частотный анализ текста, т. е. указать (в процентах), сколько раз встречается та или иная буква.
4. Даны файл, содержащий текст, и некоторые буквы. Найти слово, содержащее наибольшее количество указанных букв.
5. Дан файл, содержащий текст. Вывести в другой файл в алфавитном порядке все слова, встречающиеся в этом тексте.
6. Дан текстовый файл f. В файле не менее двух компонент. Определить являются ли два первых символа файла цифрами. Если да, то установить, является ли число, образованное этими цифрами, чётным, и вывести его в другой файл.
7. Дан символьный файл f. Записать в файл g компоненты файла f в обратном порядке.
8. Дан файл, содержащий текст, записанный строчными буквами. Получить в другом файле тот же текст, записанный заглавными буквами.
9. Дан файл, содержащий текст. Выбрать из него те символы, которые встречаются в нем только один раз, в том порядке, в котором они встречаются в тексте, и записать их в другой файл.
10. Даны файл, содержащий текст, и некоторая буква. Подсчитать, сколько слов начинается с указанной буквы, и вывести их в другой файл.
11. Дан файл, содержащий текст, включающий русские и английские слова. Подсчитать, каких букв в тексте больше – русских или латинских.
12. Дан текстовый файл. Удалить из него все лишние пробелы, оставив между словами не более одного пробела. Результат поместить в новый файл.

13. Дан файл, содержащий текст. Подсчитать количество слов, начинающихся и заканчивающихся на одну и ту же букву, и вывести их в другой файл.
14. Дан файл, содержащий текст. Переписать в другой файл этот текст через один символ (т.е. если в файл записано «привет», то в другой нужно записать «пие»).
15. Дан файл, содержащий текст с русскими и английскими словами. Вывести в отдельные файлы русские и английские слова.
16. Дан файл, содержащий текст. Переписать в другой файл только гласные буквы данного файла.
17. Дан файл, содержащий текст. Переписать в другой файл только согласные буквы данного файла.
18. Дан файл, содержащий текст. Переписать в другой файл только слова, начинающиеся с согласной, а заканчивающиеся гласной.
19. Дан файл, содержащий текст. Переписать в другой файл только слова, начинающиеся с гласной, а заканчивающиеся согласной.
20. Дан файл, содержащий текст. Переписать в другой файл только слова, начинающиеся и заканчивающиеся гласной.
21. Дан файл, содержащий текст. Переписать в другой файл только слова, начинающиеся и заканчивающиеся согласной.

Часть 2. Бинарные файлы.

1. Заполнить файл n целыми случайными числами. Получить в файле g те компоненты файла f , которые являются четными.
2. Записать в файл N действительных чисел. Вычислить произведение четных компонентов файла и сумму нечетных, и вывести полученные значения в другой файл.
3. Заполнить файл n целыми случайными числами. Получить в другом файле все компоненты, которые делятся на a и не делятся на b .
4. Записать в файл N целых случайных чисел. Подсчитать количество пар противоположных чисел среди компонентов этого файла (4 и -4 – противоположные числа) и вывести их в другой файл.
5. Заполнить файл f n целыми случайными числами. Из файла f получить файл g , исключив повторные вхождения чисел.
6. Создать структуру из двух вещественных полей. Записать в файл n компонент типа этой структуры. В другой файл записать сумму полей каждой из структур, записанных в предыдущий файл.
7. Создать структуру из двух полей: вещественное и целое. Записать в файл n компонент типа этой структуры. Переписать в другой файл последние K компонент из записанных в предыдущий файл.
8. Создать структуру из двух строковых полей. Записать в файл n компонент типа этой структуры. В другой файл записать строки, полученные соединением полей каждой из структур, записанных в предыдущий файл.
9. Создать структуру из двух полей: целое и строковое. Записать в файл n компонент типа этой структуры. Переписать в другой файл компоненты с k -го по m -ый из записанных в предыдущий файл.
10. Создать структуру из двух полей: строковое и символьное. Записать в файл n компонент типа этой структуры. Переписать в другой файл компоненты из записанных в предыдущий файл через один.
11. Записать в файл N произвольных натуральных чисел. Переписать в другой файл те элементы, которые кратны K .
12. Заполнить файл случайными действительными числами. Найти сумму минимального и максимального элементов этого файла и вывести их в другой файл.
13. Записать в файл N натуральных случайных чисел $a_1, a_2, a_3, \dots, a_n$. Сформировать новый файл, элементами которого являются числа $a_1, a_1+a_2, a_1+a_2+a_3, \dots, a_1+a_2+a_3+\dots+a_n$.
14. Записать в файл N натуральных случайных чисел. Получить в другом файле все компоненты файла, кроме тех, которые кратны K .
15. Заполнить файл f целыми случайными числами. Найти количество удвоенных нечетных чисел среди компонентов файла и вывести их в другой файл.
16. Создать структуру из двух полей: целое и вещественное. Записать в файл n компонент типа этой структуры. Переписать в отдельные файлы целое и вещественное поля из записанных в предыдущий файл компонент.
17. Создать структуру из двух целых полей. Записать в файл n компонент типа этой структуры. В другой файл записать произведение полей каждой из структур, записанных в предыдущий файл.
18. Создать структуру из двух символьных полей. Записать в файл n компонент типа этой структуры. В другой файл записать строки, полученные соединением полей каждой из структур, записанных в предыдущий файл.
19. Создать структуру из двух вещественных полей. Записать в файл n компонент типа этой структуры. В другой файл записать разность полей каждой из структур, записанных в предыдущий файл.
20. Создать структуру из двух полей: вещественное и символьное. Записать в файл n компонент типа этой структуры. Переписать в другой файл первые K компонент из записанных в предыдущий файл.

21. Заполнить файл f натуральными случайными числами. Найти количество квадратов нечетных чисел среди компонентов и вывести их в другой файл.
22. Записать в файл N действительных случайных чисел. Найти наибольшее по модулю из значений компонентов с нечетными номерами и вывести его в другой файл.
23. Заполнить файл f целыми случайными числами. Из файла f получить файл g, исключив повторные вхождения чисел. Порядок следования чисел сохранить.

Лабораторная работа №9. Графика. Программирование движущихся объектов.

Для работы с графикой в CodeBlocks нужно убедиться, что в соответствующих папках есть следующие файлы:

папка с CodeBlocks \MinGW\include\ graphics.h

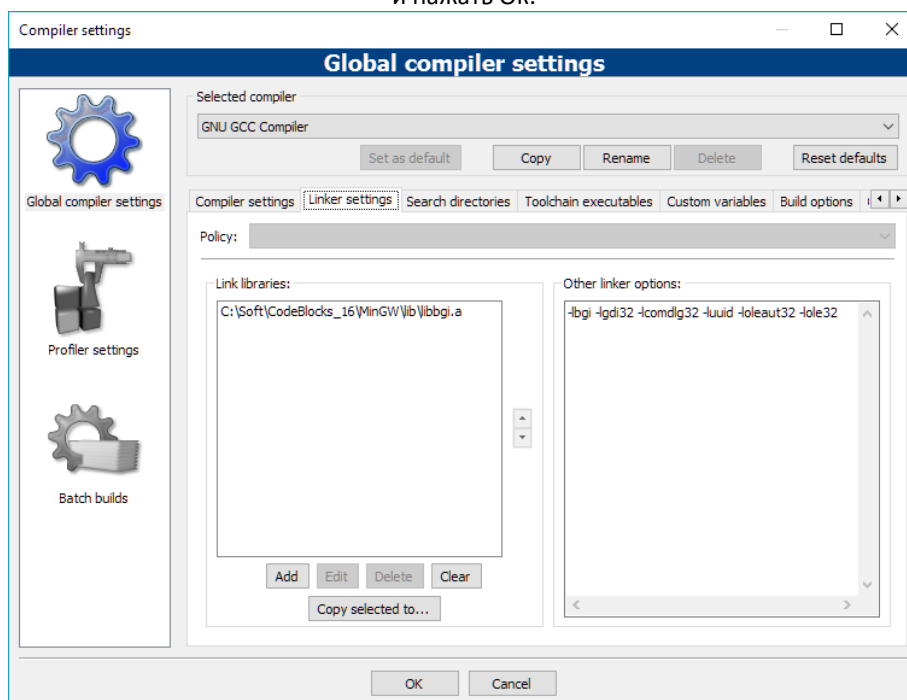
папка с CodeBlocks \MinGW \lib\ libbgi.a

Если их там нет, то нужно взять их из сетевой папки и переписать в нужные каталоги.

Также нужно прописать специальные настройки для компилятора (Сервис->Параметры компилятора или Settings->Compiler ...), как показано на рисунке, т.е. вкладка Linker settings, в левой части (Link libraries) нажать Add и выбрать файл папка с CodeBlocks \ MinGW \ lib \ libbgi.a), в правой части вписываем следующую строку:

-lbgi -lgdi32 -lcomdlg32 -luuid -oleaut32 -ole32

и нажать Ок.



Далее, в программе, нужно подключить модуль graphics.h:

```
#include <graphics.h>
```

После этого в программе можно использовать графические функции.

Первой в программе должна идти функция initwindow(int X, int Y);

где X – ширина графического окна, Y – высота графического окна. При этом точка с координатами (0;0) находится в верхнем левом углу.

После вызова этой функции можно использовать любые графические функции:

void putpixel (int x, int y, int color); – поставить точку с координатами (x;y) цветом color;

void setcolor (int color); – установить цвет рисования (можно использовать predefined константы BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, LIGHTGRAY, DARKGRAY, LIGHTBLUE, LIGHTGREEN, LIGHTCYAN, LIGHTRED, LIGHTMAGENTA, YELLOW, WHITE);

void setlinestyle (int linestyle, unsigned upattern, int thickness); – устанавливает стиль рисования линий: linestyle отвечает за стиль линии (0 – сплошная, 1 – пунктирная; 2 – штрихпунктирная; 3 – штриховая);, вместо upattern лучше ставить 0, thickness отвечает за толщину линий (0, 1, 2 и т.д.);

void line (int x1, int y1, int x2, int y2); – нарисовать линию из точки (x1;y1) в точку (x2;y2);

void moveto(int x, int y); – сделать текущей точку с координатами (x;y);

void lineto (int x, int y); – нарисовать линию от текущей точки до точки с координатами (x;y);

void linerel (int dx, int dy); – нарисовать линию от текущей точки до точки с приращением координат dx и dy;

void setfillstyle (int pattern, int color); – установка параметров заливки (pattern может принимать следующие значения: 0 – заполнение цветом фона, 1 – однородное заполнение цветом заполнения, 2 – горизонтальными линиями, 3 – \\\\\\\\\, 4 – \\\\\\\\\ толстыми линиями, 5 – \\\\\\\\\\\\\\\\\ толстыми линиями, 6 – \\\\\\\\\\\\\\\\\, 7 – клеткой, 8 – крестиком, 9 – частой клеткой, 10 – редкими точками, 11 – частыми точками), color – цвет заливки;

void circle (int x, int y, int r); – рисует окружность с координатами в точке (x;y) радиусом r;

```

void ellipse (int x, int y, int BegA, int EndA, int RX, int RY); – эллипсная дуга с центром в точке
X,Y с начальным и конечным углами BegA и EndA, горизонтальным радиусом RX и вертикальным радиусом RY:
void floodfill (int x, int y, int border); – закрасить область, в которой лежит точка с координатами (x;y),
текущим цветом рисования и текущим стилем заполнения, до границы цвета border;
void setbkcolor (int color); – установить цвет фона;
void outtext (char *textstring); – вывести, начиная с текущей позиции, строку textstring;
void outtextxy (int x, int y, char *textstring); – вывести, начиная с позиции (x;y), строку textstring;
void rectangle (int left, int top, int right, int bottom); – нарисовать прямоугольник с
координатами диагонали
(left; top) – (right; bottom);
void sector (int x, int y, int stangle, int endangle, int xradius, int yradius); –
обведенный линией и закрашенный эллипсный сектор;
void arc( int x, int y, int BegA, int EndA, int r ); – дуга окружности с центром в точке (x; y),
радиусом r, начальным углом BegA и конечным углом EndA. Углы измеряются в градусах против часовой стрелки от
направления оси X;
void bar( int left, int top, int right, int bottom ); – заполненная прямоугольная область с
заданными координатами углов
void bar3d( int left, int top, int right, int bottom, int depth, int topflag ); –
параллелепипед с заданными координатами углов, глубиной (depth) и параметром рисования вершины topflag (0
или 1);
void cleardevice( ); – очистка графического окна;
void drawpoly(int n_points, int* points); – рисование многоугольника, n_points – кол-во точек, points –
массив с координатами точек-вершин;
void fillellipse( int x, int y, int xradius, int yradius ); – обведенный линией и закрашенный
эллипс;
void fillpoly(int n_points, int* points); – рисование закрашенного многоугольника, n_points – кол-во точек,
points – массив с координатами точек-вершин;

```

Заканчивая работу с графикой нужно вызовом функции closegraph();

Пример программы, рисующей многоугольник

```

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

int main(){
    initwindow(800,600);
    int maxx,maxy;
    int poly[10]; // массив, в котором будут храниться координаты точек
    maxx = getmaxx(); // узнаем максимальную координату по x
    maxy = getmaxy(); // узнаем максимальную координату по y

    poly[0] = 20;          /* координата первой точки по x */
    poly[1] = maxy/2;      /* координата первой точки по y */

    poly[2] = maxx-20;    /* 2 точка*/
    poly[3] = 20;

    poly[4] = maxx-50;    /* 3 точка*/
    poly[5] = maxy-20;

    poly[6] = maxx/2;     /* 4 точка*/
    poly[7] = maxy/2;

    poly[8] = poly[0];     /* возврат в исходную точку */
    poly[9] = poly[1];

    drawpoly(5,poly); // рисуем многоугольник из 5 точек
    getch();
    closegraph();
    return 0;
}

```

Для программирования движущихся объектов можно использовать следующий алгоритм:

1. нарисовать объект;
2. подождать некоторое время;
3. нарисовать этот же объект цветом фона;
4. нарисовать объект со смещением координат;
5. повторить нужное кол-во раз с пункта 2.

Например, программа, изображающая движущийся круг:

```

#include <graphics.h>

```

```

#include <stdio.h>
#include <ctype.h>

int main()
{
    int i,j;
    initwindow(800,600); // открыть окно для графики
                           // размером 800 на 600 пикселей
    setcolor(WHITE); // установить белый цвет
    floodfill(1,1,GREEN); //закрасить все белым цветом, пока не встретится зеленый,
                           // а он не встретится
    for (i=0;i<500;i++) // будем в цикле рисовать и стирать, т.е. рисовать цветом фона
    {
        // параметр цикла будет служить приращением к координатам центра окружности
        setcolor(RED); // установить красный цвет
        circle(100+i,100,50); // нарисовать окружность радиусом 50
                               // с центром в точке (100+i,100), т.е. на первой итерации цикла
                               // рисуется окружность с центром в (100,100), на второй итерации
                               // - с центром в (101,100) и т.д.
        delay(10); // подождать 10 миллисекунд
        setcolor(WHITE); // установить белый цвет
        circle(100+i,100,50); // нарисовать ту же самую окружность цветом фона
    }
    setcolor(RED); // установить белый цвет
    circle(100+i,100,50); // нарисовать последнюю окружность
    closegraph(); // закрыть окно с графикой
    return 0;
}

```

Пример движения окружности по траектории (**!!! есть в папке на сервере, tuda-suda.cpp**):

```

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

void ris_circle(int dx, int dy, int maxX,int maxY, int xcoord, int ycoord, int r)
// функция движения окружности: dx, dy - на сколько смещать окружность за одну итерацию цикла,
// от этого будет зависеть скорость движения окружности,
// maxX, maxY - на сколько по X и Y перемещать в целом,
// xcoord, ycoord - координаты центра окружности,
// с этого значения начинается текущее перемещение окружности,
// r - радиус окружности
{
    int i,j; // i,j будут отвечать за приращение текущих координат
    for (i=0,j=0; abs(i)<maxX || abs(j)<maxY;i+=dx,j+=dy)
    // abs(i)<maxX || abs(j)<maxY - условие остановки движения,
    // т.е. если i или j достигнут заданного максимума, причем по модулю,
    // т.к. и i, и j может уменьшаться, если будут заданы отрицательные значения dx или dy
    {
        setcolor(RED); // установить красный цвет
        circle(xcoord+i,ycoord+j,r); // нарисовать окружность с текущими координатами
        if (kbhit()) break; // если будет нажата любая клавиша, то произойдет выход из цикла
        // kbhit() возвращает ненулевое значение, если была нажата любая клавиша
        delay(1); // подождать 1 миллисекунду
        setcolor(WHITE); // установить белый цвет
        circle(xcoord+i,ycoord+j,r); // нарисовать ту же самую окружность цветом фона
    }
}

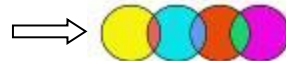
int main()
{ int r=50; // радиус равен 50
  initwindow(800,600); // открыть окно для графики
                        // размером 800 на 600 пикселей
  setcolor(WHITE); // установить белый цвет
  floodfill(1,1,GREEN); // закрасить все белым цветом, пока не встретится зеленый,
                        //а он не встретится

  while (!kbhit()) { // цикл будет продолжаться, пока не будет нажата какая-либо клавиша
    ris_circle(5,0,500,0,100,100,r); // вызов созданной процедуры для движения вправо
    ris_circle(0,4,0,200,100+500,100,r); // для движения вниз
    ris_circle(-5,0,500,0,100+500,100+200,r); // для движения влево
    ris_circle(0,-4,0,200,100,100+200,r); // для движения вверх
  }

  closegraph(); // закрыть окно с графикой
  return 0;
}

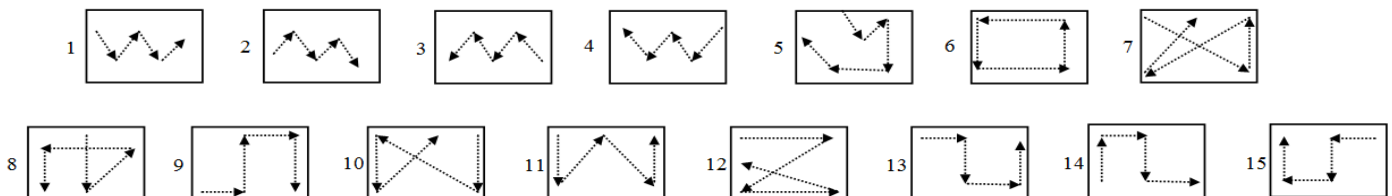
```

1. Нарисуйте три окружности с заключёнными в них треугольником, квадратом и звездой. Все фигуры должны быть разного цвета, разного типа заливки и типа линий.
2. Нарисуйте радугу.
3. Нарисуйте олимпийский флаг (кольца – разного типа линий, залитые разными цветами и типом заливки).
4. Нарисуйте контур прямоугольника, в котором нарисуйте линиями разного цвета и стиля своё имя.
5. Нарисуйте контур прямоугольника, в котором нарисуйте линиями разного цвета и стиля свою фамилию.
6. Нарисуйте прямоугольную сетку на экране линиями разного цвета и стиля.
7. Нарисуйте разноцветную мишень.
8. Нарисуйте косую сетку на экране линиями разного цвета и стиля.
9. Нарисуйте 5 треугольников с разным типом линий и типом и цветом заливки.
10. Нарисуйте 3 круговые диаграммы, состоящие из 10 заполненных секторов, используя различные орнаменты и цвета заполнения.
11. Нарисуйте шахматное поле.
12. Нарисовать на экране 5 заполненных звёзд разного цвета и стиля заполнения.
13. Нарисовать на экране 5 заполненных ромбов разного цвета и стиля заполнения.
14. Нарисовать на экране 5 заполненных параллелограммов разного цвета и стиля заполнения.
15. Нарисуйте 10 окружностей разного цвета, пересекающихся в 2-х точках. Цвет областей пересечения должен отличаться от цветов окружностей, которые эту область образуют.



Часть 2. Графика в движении.

Запрограммировать движение объекта (по вариантам: **1** – круг, **2** – эллипс ($r1=20$, $r2=50$), **3** – квадрат, **4** – прямоугольник (длина 20, ширина 40), **5** – равносторонний треугольник, **6** – прямоугольный треугольник, **7** – фамилия, **8** – имя, **9** – номер группы, **10** – аббревиатура университета, **11** – аббревиатура факультета, **12** – номер кабинета, **13** – номер этажа, **14** – номер



своего варианта, **15** – аббревиатура предмета). Траектория движения по вариантам:

Лабораторная работа №10. Модульное программирование в Си.

Каждый модуль программы должен описываться двумя файлами с одним именем, но разными расширениями. Первый файл должен содержать текст функций модуля, то есть реализацию модуля и обычно имеет расширение *.c (или *.cpp). Второй файл содержит описание интерфейса модуля, то есть прототипы функций модуля и описания глобальных переменных модуля, и обычно имеет расширение *.h и называется заголовочным файлом.

В некоторых случаях может оказаться целесообразным наличие большого количества маленьких модулей, например, с разного рода сервисными библиотечными функциями. В этом случае допускается использовать один заголовочный файл для описания интерфейсов нескольких модулей.

Во всех случаях надо следить за тем, чтобы прототип любой функции имелся лишь в одном экземпляре и находился в заголовочном файле. Файл с исходным текстом функций модуля (реализацией модуля) должен содержать директиву препроцессора #include, подключающую заголовочный файл, описывающий интерфейс данного модуля. Это необходимо для того, чтобы компилятор был в состоянии проверить соответствие интерфейса модуля его реализации.

По сути, модуль отличается от обычной программы тем, что в нем нет функции main(), вся остальная структура такая же, а в заголовочном файле помещаются прототипы функций, без реализаций, и глобальные переменные и пользовательские типы данных (если они есть) .

При компиляции проекта, состоящего из нескольких модулей, компилятору нужно явно указать, какие файлы с какими связываются.

Например, реализуем функции из темы «Функции» для сложения и вычитания 2-х чисел в виде отдельного модуля.

Для этого нам понадобится создать 2 файла: mymod.cpp и mymod.h (**!!!должны быть в одном каталоге**)

Содержимое файла mymod.cpp:

```
#include <conio.h>
#include <stdio.h>
#include "mymod.h" // подключаем описание интерфейса для нашего модуля

int et1(int a,int b) // реализуем функцию et1
{
    int c=a+b;
    return c; }

void et2(int a, int b) // реализуем функцию et2
{
    printf("%d - %d =%d\n",a,b,a-b); }
```

Для добавления основного файла модуля (cpp-файла) к текущему проекту в Code::Blocks нажимаем пункт меню File -> New->File. В появившемся окне выбираем C/C++ source. В следующем окне выбираем C++, потом указываем имя создаваемого модуля и путь к нему (каталог текущего проекта), ставим галочки возле Debug и Release.

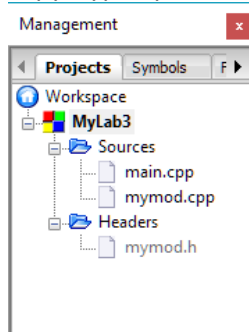
Содержимое файла mymod.h:

```
int et1(int a,int b); // объявляем функцию et1
```

```
void et2(int a, int b); // объявляем функцию et2
```

Для добавления заголовочного файла модуля (h-файла) к текущему проекту в Code::Blocks нажимаем пункт меню File -> New->File. В появившемся окне выбираем C/C++ header. В следующем окне выбираем C++, потом указываем имя создаваемого заголовочного файла и путь к нему (каталог текущего проекта), ставим галочки возле Debug и Release.

Структура проекта теперь должна быть следующей:



Теперь мы можем использовать созданный модуль в любой программе, если подключим интерфейсный файл mymod.h, например:

```
#include <conio.h>
#include <stdlib.h>
#include <stdio.h>
#include "mymod.h" // подключаем наш модуль через его интерфейсный файл

int main()
{
    int x,y,z;
    printf("Введите два числа ->");
    scanf("%d %d",&x,&y);
    z=et1(x,y); // используем функцию et1 из созданного модуля
    printf("\n x+y=%d \n",z);
    et2(x,y); // используем функцию et2 из созданного модуля
    system("pause");
    return 0;
}
```

Список задач

Переделать все задачи, в которых требовалось написать свою функцию, с использованием модуля, в который поместить все свои функции.

Лабораторная работа №11. Списки.

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

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

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

1. Получить память для него;

2. Поместить туда информацию;
3. Добавить элемент в конец списка (или начало).

Элемент списка состоит из разнотипных частей (хранящая информация и указатель), и его естественно представить структурой.

Пример описания подобной структуры:

```
typedef struct nd
{ int val;
  struct nd * next; } ND; // структура ND , у которой 2 поля
```

Примеры

Создание списка.

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

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

```
typedef struct nd
{ int val;
  struct nd * next; } ND;
```

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

```
ND *beg=NULL, *end=NULL, *p;
```

Создадим первый элемент:

```
p=(ND *) malloc(sizeof(ND)); // Выделение памяти для первого звена списка
p->val=3;
beg=p;
p->next=end; // следующий элемент приравниваем к end, т.е. к NULL
```

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

```
p->next=(ND *)malloc(sizeof(ND)); // Выделение памяти для второго звена списка
p=p->next; // сделать текущим следующий, под который уже выделена память
p->val=5;
p->next=end;
```

```
p->next=(ND *)malloc(sizeof(ND)); // Выделение памяти для третьего звена списка
p=p->next; // сделать текущим следующий, под который уже выделена память
p->val=1;
p->next=end;
```

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

Присоединение нового элемента к голове списка:

```
p=(ND *)malloc(sizeof(ND)); // Выделение памяти для нового звена списка
p->val=33;
p->next=beg; // следующим за новым звеном делаем начальное звено
beg=p; // начальным звеном (головой) делаем новое звено
```

Просмотр списка.

Просмотр элементов списка осуществляется последовательно, начиная с его начала. Указатель *p* последовательно ссылается на первый, второй и т. д. элементы списка до тех пор, пока весь список не будет пройден. При этом с каждым элементом списка выполняется некоторая операция – например, печать элемента.

```
printf("\nСодержимое списка:");
p=beg; // делаем текущим головное звено
while (p!=NULL) // пока не дойдем до значения NULL
{ printf("\nval=%d",p->val); // печатаем значение поля val текущего звена списка
  p=p->next; // делаем текущим следующий элемент
}
```

Удаление элемента из списка.

При удалении элемента из списка необходимо различать три случая:

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

Удаление элемента из начала списка.

```
p = beg; //запомним адрес первого элемента списка
beg = beg->next; // теперь beg указывает на второй элемент списка
free(p); // освободим память, занятую переменной p
```

Удаление элемента из середины списка.

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

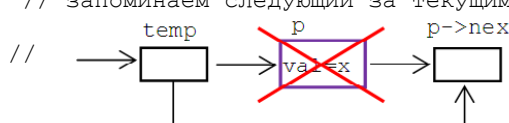
```
int x=5; // значение, которое будем искать
```



```

p = beg;
ND *temp;
while (p!=end && p->val !=x) // пока не конец списка и пока поле val не равно x
{ temp=p;                  // запоминаем текущий
  p = p->next;              // запоминаем следующий за текущим
}
temp->next= p->next;        //
free(p);

```



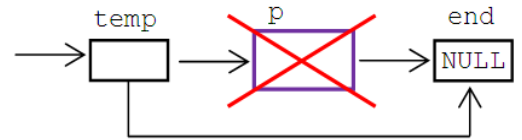
Удаление из конца списка.

Оно производится, когда указатель temp показывает на предпоследний элемент списка, а p – на последний.

```

p=beg; temp=beg;          // оба указателя перемещаем на начало списка
while (p->next!=end)      // пока следующим за текущим не будет end, т.е. NULL
{ temp=p;                // temp указывает на p
  p = p->next;            // p указывает на следующий за p
}
temp->next= end;          // следующий за temp - end, т.е. NULL
free(p);                  // освобождаем память, которую занимал p

```



Уничтожение списка

После окончания работы со списком в конце программы необходимо освободить память, которую он занимал.

```

// Уничтожение списка
p=beg;          //переходим в начало списка
while (beg!=NULL) // пока не конец списка
{ beg=p->next;   // начало списка смещаем на следующий элемент
  free(p);      // освобождаем занимаемую память
  p=beg;        // текущим элементом делаем новое начало
}

```

Список задач

Часть 1.

1. Сформировать список строк и а) сохранить его в текстовом файле; б) сохранить его в обратном порядке в текстовом файле.
2. Сформировать список строк из текстового файла.
3. Написать функцию, которая вычисляет среднее арифметическое элементов непустого списка.
4. Написать процедуру присоединения списка L2 к списку L1.
5. Написать функцию, которая создает список L2, являющийся копией списка L1, начинающегося с данного узла (задает пользователь).
6. Написать функцию, которая подсчитывает количество вхождений элемента, который вводит пользователь, в списке.
7. Написать функцию, которая удаляет из списка все вхождения элемента, который вводит пользователь.
8. Сформировать список целых чисел и удалить из него все четные.
9. Сформировать список вещественных чисел и вычислить сумму.
10. Написать функцию, которая проверяет, упорядочены ли элементы списка.
11. Написать функцию, подсчитывающую количество слов в списке, которые начинаются с той же буквы, что и следующее слово.
12. Написать функцию, которая использует исходный список L и создает два новых списка L1 и L2. L1 содержит нечетные узлы, а L2 – четные.
13. Написать функцию, которая использует исходный список L и создает два новых списка L1 и L2. L1 содержит нечетные числа, а L2 – четные.

Часть 2.

1. Составить программу, которая вставляет в список L новый элемент F за каждым вхождением элемента E.
2. Составить программу, которая вставляет в список L новый элемент F перед первым вхождением элемента E, если E входит в L.
3. Составить программу, которая удаляет из списка L все элементы E, если таковые имеются.
4. Составить программу, которая удаляет из списка L за каждым вхождением элемента E один элемент, если таковой имеется и он отличен от E.
5. Составить программу, которая удаляет из списка L все отрицательные элементы.
6. Составить программу, которая проверяет, есть ли в списке L хотя бы два одинаковых элемента.
7. Составить программу, которая переносит в конец непустого списка L его первый элемент.
8. Составить программу, которая вставляет в список L за первым вхождением элемента E все элементы списка L, если E входит в L.
9. Составить программу, которая переворачивает список L, т.е. изменяет ссылки в этом списке так, чтобы его элементы оказались расположенными в обратном порядке.
10. Составить программу, которая в списке L из каждой группы подряд идущих одинаковых элементов оставляет только один.
11. Составить программу, которая формирует список L, включив в него по одному разу элементы, которые входят одновременно в оба списка L1 и L2.

12. Составить программу, которая формирует список L , включив в него по одному разу элементы, которые входят в список $L1$, но не входят в список $L2$.
13. Составить программу, которая формирует список L , включив в него по одному разу элементы, которые входят в один из списков $L1$ и $L2$, но в то же время не входят в другой.