Оглавление

Базовые понятия языка программирования С 4
Первая программа на языке программирования С 8
Лабораторная работа №1. Программирование арифметических операций 13
Лабораторная работа №2 Препроцессорные средства 16
Лабораторная работа №3 Массивы и указатели 31
Лабораторная работа №4 Матрицы и действия над ними 46
Лабораторная работа №5 Сортировка массивов 55
Лабораторная работа №6 Структуры и объединения 66
Лабораторная работа №7. Функции. Рекурсивные функции. 82
Лабораторная работа №8. Работа со строками 100
Лабораторная работа №9. Типовые динамические структуры данных. Стек. Двоичное дерево поиска. 107
Лабораторная работа №10 Ввод/вывод. Функция main() Параметры функции main(). 110
Лабораторная работа №11 (дополнительно) Консольное приложение 118
Ответы на вопросы 119

Базовые понятия языка программирования С

В алфавит языка С входят:

- прописные и строчные буквы латинского алфавита (A..Z, a..z);
- Цифры (0..9);
- специальные знаки “ , { } | [ ] ( ) + - / % \ ; ` . : ? < = > _ ! & * # ~ ^
- управляющие символы (“обобщенные пробельные символы”), используемые для отделения лексем друг от друга (например, пробел, табуляция, переход на новую строку);

Идентификатор – последовательность букв, цифр и символов подчеркивания, начинающиеся с буквы или символа подчеркивания.

Ключевые (служебные) слова – идентификаторы, зарезервированные в языке, которые нельзя использовать в качестве свободно выбираемых имен. В таблице 1 представлены служебные слова, определенные в языке С.

Таблица 1

auto	**break**	**case**	**char**
const	**continue**	**default**	**do**
double	**else**	**enum**	**extern**
float	**for**	**goto**	**if**
int	**long**	**register**	**return**
short	**signed**	**sizeof**	**static**
struct	**switch**	**typedef**	**union**
unsigned	**void**	**volatile**	**while**

Литералы (константы) – это фиксированные значения, которые не могут быть изменены программой. Синтаксис языка С определяет пять типов констант:

- *Символьные.* Например: ‘A’, ‘a’, ‘8’, ‘+’ и т.д. Некоторые символы в С имеют специальное назначение, поэтому их нельзя ввести с клавиатуры. По этой причине в язык С введены специальные символьные последовательности, называемые управляющими последовательностями. В таблице 2 приводится их список.

Таблица 2

‘\n’	перевод строки
‘\t’	горизонтальная табуляция
‘\r’	возврат каретки
‘\\’	обратная косая черта
‘\’’	одиночная кавычка (апостроф)
‘\”’	двойная кавычка
‘\0’	нулевой символ
‘\a’	звуковой сигнал (звонок)
‘\b’	возврат на одну позицию
‘\f’	подача страницы (для перехода к началу следующей страницы)
‘\v’	вертикальная табуляция
‘\?’	вопросительный знак
‘\N’	восьмеричная константа (где n – восьмеричная константа)
‘\xN’	шестнадцатеричная константа (где n –шестнадцатеричная константа)

- *Целые (десятичные, восьмеричные, шестнадцатеричные). Например: 255, 0377, 0xFF*
- *Вещественные (с плавающей точкой).* Например: 3. 3e0 3.14159 .314159E1 0.0
- *Нулевой указатель (NULL).*
- *Константы перечисляемого типа.*
  enum тип_перечисления {список_именованных_констант};
  Пример:
  enum DAY {SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY};
- *Строковые.* Пример: "Это пример \n строковой константы"

Способы объявления именованных констант

- enum тип_перечисления {список_именованных_констант};
- const тип_константы имя_константы = значение_константы;
  Пример: cons double alpha = 7.2134;
- #define имя_константы значение_константы;
  Пример: #define MAX 10

Тип данных. В общем случае, под *типом данных* понимается множество значений, объединенных определенной совокупностью допустимых операций. В таблице 3 представлены основные типы данных.

Таблица 3

Тип данных	Размер, бит	Диапазон значений
Целые		
unsigned char	8	0..255
char	8	-128..127
enum	16	-32768..32767
unsigned int	16	0..65536
short int (short)	16	-32768..32767
unsigned short	16	0..65536
int	16	-32768..32767
unsigned long	32	0..4294967295
long	32	-2147483648.. 2147483647
Вещественные		
float	32	3.4e-38..3.4e+38
double	64	1.7e-308..1.7e+308
long double	80	3.4e-4932..1.1e+4932

Каждый из целочисленных типов может быть определен либо как знаковый **signed**, либо как беззнаковый **unsigned** (по умолчанию **signed**).

Переменная – это именованный фрагмент памяти для хранения данных.

Форма определения переменных:

```
тип_данных список_имен_переменных;
```

где `список_имен_переменных` – это выбранные программистом идентификаторы, которые в списке разделяются запятыми.

Переменным можно присваивать начальные значения, явно указывая их в определениях. Такой прием называется инициализацией.

```
тип_данных имя_переменной = значение;
```

Оператор – это символ, который указывает компилятору на выполнение конкретных математических действий или логических манипуляций.

В таблице 4 приведены все операторы, определенные стандартом языка. Операторы в таблице разбиты на группы в соответствии с их рангами. Если в выражении несколько операторов одного ранга, то они выполняются в соответствии с правилами ассоциативности либо слева направо (→), либо справа налево (←).

Таблица 4

Ранг	**Операторы**	**Ассоциативность**
1	( ) [ ] -> .	→
Унарные (одноместные)		
2	! ~ + - ++ -- & * (тип) sizeof	←
Мультипликативные бинарные		
3	% * /	→
Аддаптивные бинарные		
4	+ -	→
Поразрядного сдвига		
5	<< >>	→
Отношения (сравнения)		
6	< <= => >	→
7	== !=	→
Поразрядные		
8	& (поразрядная конъюнкция «И»)	→
9	^ (поразрядное исключающее «ИЛИ»)	→
10	| (поразрядная дизъюнкция «ИЛИ»)	→
логические		
11	&& (конъюнкция «И»)	→
12	|| (дизъюнкция «ИЛИ»)	→
Условный оператор		
13	?:	←
Операторы присваивания		
14	= *= /= %= += -= &= ^= |= <<= >>=	←
Запятая		
15	,	→

Комментарий – это текст пояснительного содержания, встраиваемый в программу.

В языке С поддерживаются два типа комментариев: однострочный и многострочный.

```
//это однострочный комментарий

/*
    Это
    многострочный
    комментарий
*/
```

Первая программа на языке программирования С

Создание консольного приложения в Microsoft Visual Studio

Так как создание консольного приложения для различных версий Microsoft Visual Studio (MS VS), в основном, ни чем не отличается, поэтому не будем ссылаться на конкретную версию среды разработки.

1. Запустите приложение MS VS. На экране появится главное окно программы (рис.1).

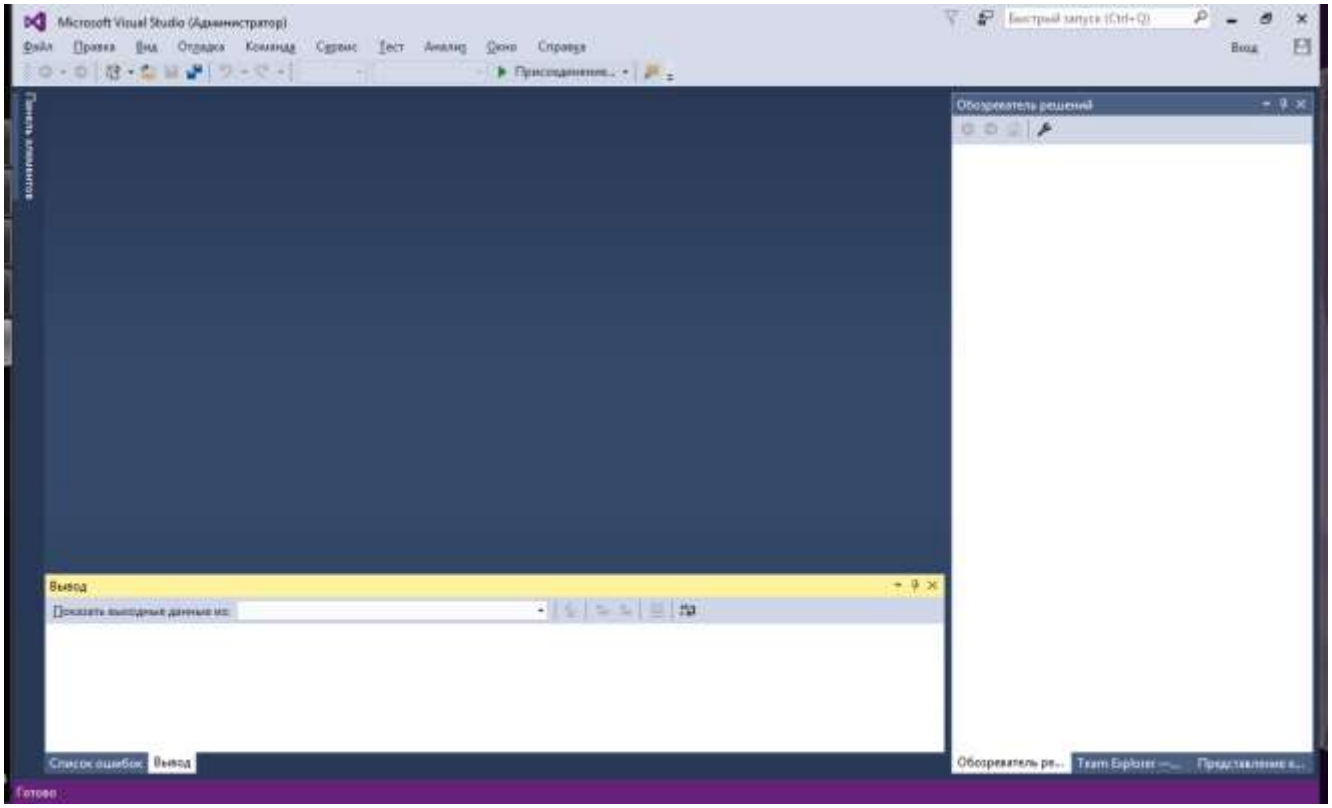


Рис.1. Главное окно программы Visual Studio

2. Для создания нового приложения в главном меню выберите

   Файл -> Создать -> Проект…

   - На экране появится диалоговое окно выбора типа проекта (рис.2). Выберите тип проекта Visual C++ -> Win32 ->Консольное приложение Win32 (Win32 Console Application).
   - Далее, следует заполнить поля Имя (Name), Расположение (Location), Имя решения (Solution Name).
   - Нажмите *ОК*.
3. Далее, в окне *Мастер приложений* нажмите кнопку *Далее* и укажите параметры приложения (рис.3)
   - *Тип приложения – Консольное приложение*.
   - *Дополнительные параметры – Пустой проект*.
   - Нажмите *Готово*.

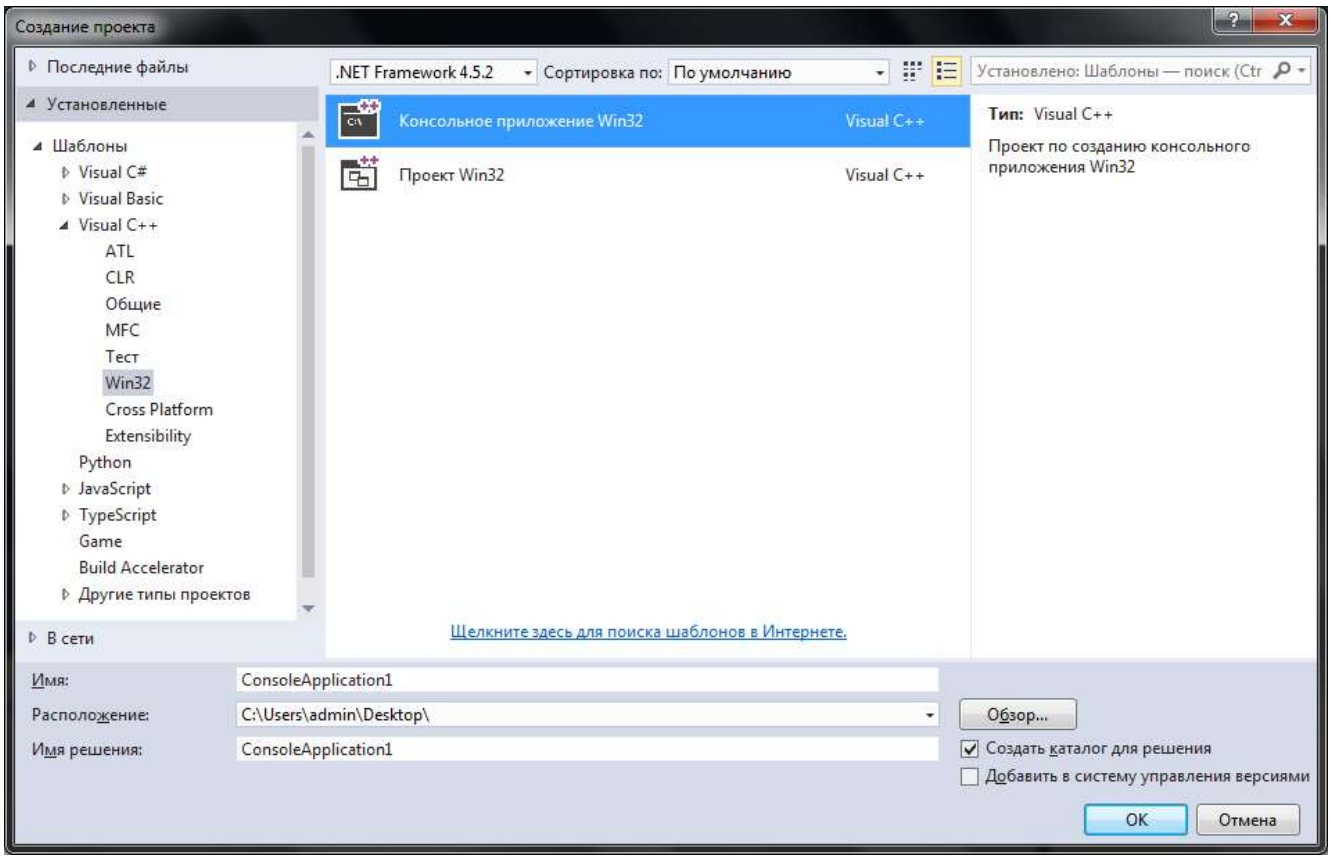


Рис.2. Диалоговое окно выбора типа проекта

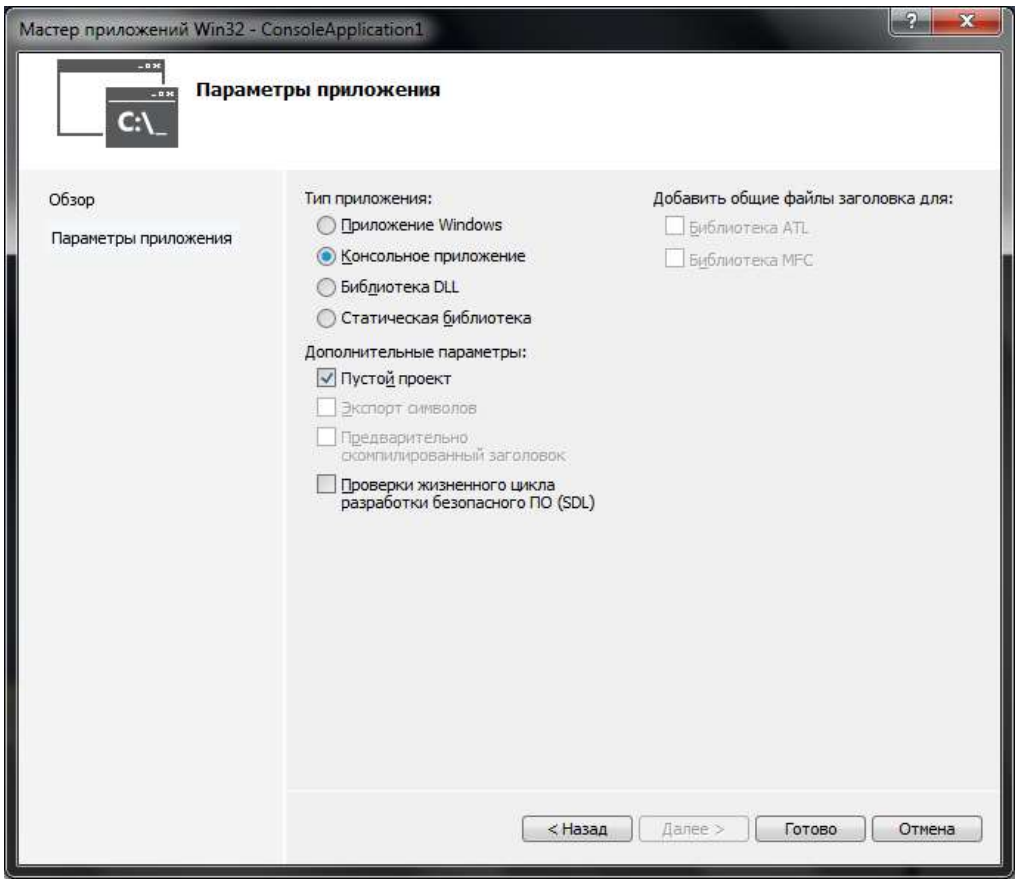


Рис.3. Окно настройки консольного проекта

4. В вкладке *Обозреватель решений* создайте файл исходного кода

   Обозреватель решений ->…->Файлы исходного кода -> Щелчок правой кнопкой мыши -> Добавить -> Создать элемент…

   - В появившемся окне (рис.4) выберите тип файла *Файл C++ (.cpp).*
   - В поле Имя укажите имя файла с расширением «*.c*» (*имя_файла.c*). По умолчанию компилятор Visual C++ рассматривает все файлы с расширением «*.c*» как исходные файлы на языке C, а файлы с расширением «*.cpp*» как исходные файлы на языке C++.
   - Нажмите *Добавить*.

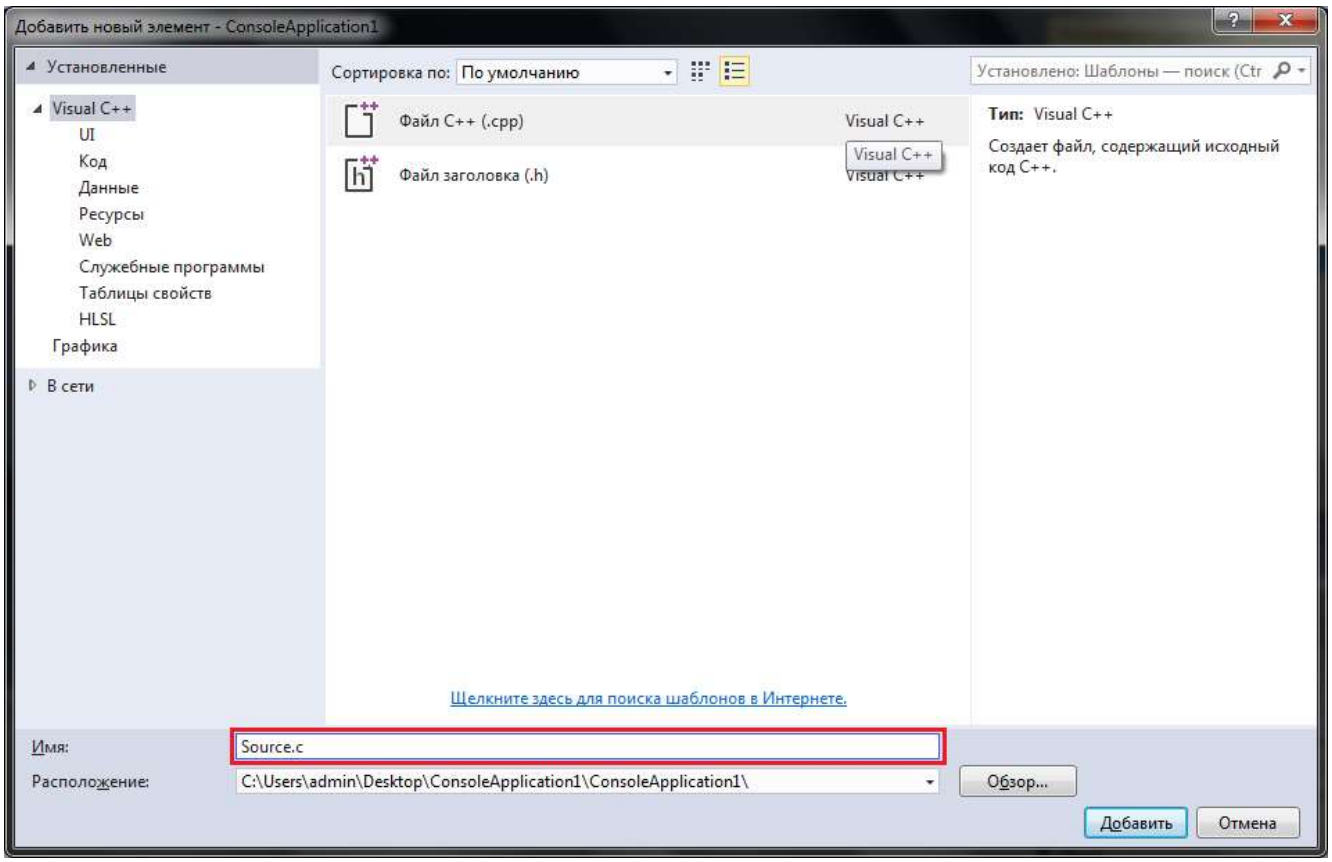


Рис.4. Окно создания нового элемента

5. В появившемся окне введите следующий текст программы

```
1  //преобразование галлонов в литры
2  #include <stdio.h>
3  #define GAL_IN_LIT 3.78541178
4
5  void main(){
6       double gallons;     //объем жидкости в галлонах
7       double liters;      //объем жидкости в литрах
8
9       scanf("%lf", &gallons);
10      liters = gallons * GAL_IN_LIT;
11      printf("%lf gallons is %lf liters\n", gallons, liters);
12 }
```

Для запуска программы нажмите клавиши Ctrl + F5.

Рассмотрим более подробно представленный код программы для преобразования галлонов в литры.

1	Однострочный комментарий
2	Директива препроцессора `#include` обязывает компилятор включить либо заголовочный файл стандартной библиотеки языка С, либо другой исходный файл имя которого указано в директиве.
3	Директива препроцессора `#define` используется для определения идентификато-

	ра и символьной последовательности, которая будет подставлена вместо идентификатора везде, где он встречается в исходном коде программы. Соответственно идентификатор `GAL_IN_LIT` в строке 10 будет заменен препроцессором на значение указанное в директиве.
5	Функция `main()` является главной функцией программы, без которой программа не может быть выполнена. Эта функция всегда должна присутствовать в программе с фиксированным именем **main**.
6	Определение переменной gallons для хранения информации о галлонах
7	Определение переменной liters для хранения информации о литрах
9	Ввод галлонов. Функция scanf() – это функция общего назначения ввода данных с консольного устройства. Функция scanf() возвращает количество введенных полей, а при возникновении ошибки – значение EOF (определено в заголовке stdio.h)
10	Вычисление количества литров в галлонах, результат присваивается переменной liters.
11	Вывод полученной информации. Функция printf() записывает данные в стандартное устройство вывода (консоль).

Важно! Каждый оператор, каждое определение и каждое описание в программе на языке С завершается точкой с запятой ';'. Любое допустимое выражение, за которым следует ';' воспринимается как оператор. Это справедливо и для пустого выражения, т.е. отдельный символ «точка с запятой» считается пустым оператором.

Лабораторная работа №1.
Программирование арифметических операций

Задание

1. Ознакомиться с функциями и константами в заголовочном файле `math.h`
2. Ознакомиться с функциями ввода-вывода `scanf()` и `printf()`
3. Создать консольный проект
4. Написать программу в соответствии с заданным вариантом
5. При разработке программы использовать функции библиотеки `math.h`
6. Для ввода-вывода использовать функции `scanf()` и `printf()`
7. Ввод: неизвестные переменные
   Вывод: результат работы программы

№	Задание
1	Вычислить площадь круга радиуса R.
2	$y = \frac{x}{e^x}$
3	Вычислить площадь полной поверхности прямоугольного параллелепипеда.
4	$y = e^{\sqrt{x+1}}$
5	Вычислить радиус сферы, описанной вокруг куба с ребром длины a.
6	$y = \frac{e^x}{1+x^2}$
7	Вычислить площадь равнобедренной трапеции с радиусом с радиусом вписанной окружности r , и углом при основании α.
8	$y = \frac{1}{\sqrt{1-x^4-x^2}}$
9	Вычислить площадь сектора круга радиуса R, ограниченного центральным углом α.
10	$y = e^{\sqrt{\ln x}}$
11	Вычислить площадь боковой поверхности цилиндра с радиусом основания R и высотой H.
12	$y = \frac{(x+1)^3\sqrt[4]{x-2}}{\sqrt[5]{(x-3)^2}}$
13	Вычислить расстояние между двумя точками на плоскости.
14	$y = x^3 + 3x^2$
15	Вычислить площадь полной поверхности цилиндра с радиусом основания R и высотой H.
16	$y = \frac{2x}{x^2-3x+2}$
17	Вычислить площадь шара радиуса R.
18	$y = \frac{x^2+1}{3(x^2-1)} + (x^2-1)(1-x)$
19	Вычислить площадь икосаэдра с ребром длины a.
20	$y = \frac{1-x^3}{\sqrt{\pi}}$
21	Вычислить объем шара радиуса R.

22	$y = \sqrt{x + \sqrt{x + \sqrt{x}}}$
23	Вычислить объем икосаэдра с ребром длины a.
24	$y = \sqrt{1 + \sqrt{\pi x}}$
25	Вычислить радиус сферы, вписанной в икосаэдр с ребром длины a.
26	$y = \frac{1 + \sqrt{x}}{1 + \sqrt{2x}}$
27	Вычислить радиус сферы, описанной вокруг икосаэдра с ребром длины a.
28	$y = \sqrt{\frac{x + 1}{x - 1}}$
29	Вычислить площадь поверхности тетраэдра с ребром длины a.
30	$y = e^{\pi x} \ln(x + 1)$
31	Вычислить объем тетраэдра с ребром длины a.
32	$y = \ln(1 - 2x)$
33	Вычислить радиус сферы, вписанной в тетраэдр с ребром длины a.
34	$y = e^{\sqrt{\ln x}}$
35	Вычислить радиус сферы, описанной вокруг тетраэдра с ребром длины a.

Контрольные вопросы

1. Какое из этих имен можно использовать, как имя переменной?
   a) 1st
   b) IF
   c) do
   d) value#1
   e) Нет правильного ответа.

2. Как вывести на экран значение переменной var типа float с двумя цифрами после десятичной точки?
   a) printf("%f", var);
   b) printf("%2f", var);
   c) printf("%f", &var);
   d) printf("%.2f", var);
   e) Нет правильного ответа.

3. Какое из этих имен можно использовать как имя переменной?
   a) DO
   b) 1st
   c) value#1
   d) my var
   e) Нет правильного ответа.

4. Как ввести с клавиатуры значение переменной вещественного типа удвоенной точности?
   a) scanf("%d", &var);
   b) scanf(var);
   c) scanf("%lf", var);
   d) scanf("%lf", &var);

5. Выбрать типы данных, имеющиеся в языке C.
   a) char, int, float, complex, unsigned int, void, enum.
   b) char, short, int, float, double, long.
   c) char, short, int, long char, float, double.
   d) unsigned char, char, short, int, long float, long int, logical, string.
   e) Нет правильного ответа.

6. Что будет выведено на экран как результат работы следующего кода?

```
#include <stdio.h>
void main(){
     int a = 0;
     ++a++;
     printf(“%d\n”, a);
}
```

   a) 1
   b) 2
   c) Возникнет ошибка на этапе компиляции.
   d) 3
   e) Нет правильного ответа.

7. Выберите все символьные константы?
   a) ‘\0’
   b) “\t”
   c) “a”
   d) “5”
   e) ‘\a’
   f) Нет правильного ответа.

8. Что выведет следующая программа?

```
double a;
double b = 2;
a = b * 5 / 2 + 5 / 2 * b;
printf("%f", a);
```

   a) 5.000000
   b) 8.000000
   c) 9.000000
   d) 10.00000

9. Как правильно записать в выражении число π?
   a) M_PI
   b) 3.14
   c) 3.14159265358979323846
   d) 3.14159
   e) Нет правильного ответа

10. Какая функция используется для получения арккосинуса числа?
    a) arcos();
    b) cos();
    c) acos();
    d) abs();

Лабораторная работа №2
Препроцессорные средства

Препроцессор языка С – это часть компилятора, которая подвергает программу различным текстовым преобразованиям до реальной трансляции исходного кода в объектный.

Команды препроцессора называются ***директивами препроцессора***.

Препроцессор языка С включает следующие директивы (таблица 2.1).

Таблица 2.1.

Директива	Описание
`#define`	Определение макроса или препроцессорного идентификатора
`#include`	Включение заголовка или другого исходного файла
`#undef`	Отмена определения макроса или препроцессорного идентификатора
`#if`	Проверка условия выражения
`#ifdef`	Проверка определенности идентификатора
`#ifndef`	Проверка неопределенности идентификатора
`#else`	Начало альтернативной ветви для #if
`#endif`	Окончания условной директивы #if
`#elif`	Составная директива #else / #if
`#line`	Смена номера следующей ниже строки
`#error`	Формирование текста сообщения об ошибке трансляции
`#pragma`	Действия, предусмотренные реализацией
`#`	Пустая директива

Кроме препроцессорных директив имеются три препроцессорные операции (таблица 2.2).

Таблица 2.2.

Операция	Описание
defined	Проверка истинности операнда
##	Конкатенация препроцессорных лексем
#	Преобразование операнда в строку символа

Обобщенный формат директивы препроцессора

```
#имя_директивы лексемы_препроцессора
```

Теперь рассмотрим более подробно каждую из директив.

Директивы `#define, #undef`

Директива `#define` используется для определения идентификатора и символьной последовательности, которая будет подставлена вместо идентификатора, везде, где он встречается в исходном коде программы. Этот идентификатор называется ***макро-именем***, а процесс замены – ***макроподстановкой***.

Общий формат использования этой директивы имеет следующий вид.

```
#define макроимя последовательность символов
```

После включения этой директивы каждое вхождение текстового фрагмента `макроимя` будет заменяться заданным элементов `последовательность_символов`.

Пример:

```
#define WIDTH 640
void main(){
     printf("%d", WIDTH);     //640
}
```

Некоторые примеры макроопределений

```
#define ONE 1
#define TWO ONE+1
#define THREE ONE+TWO
```

Директива #define имеет еще одно назначение: макроимя может использоваться с аргументами.

Пример:

```
#define MAX(a,b) (a < b) ? b : a
```

Для удаления макроопределения используется директива **`#undef`**.

```
#undef макроимя
```

Существуют встроенные макроимена, доступные препроцессору во время обработки (таблица 2.3). В конкретных реализациях набор предопределенных констант шире. Для получения более полных сведений о предопределенных препроцессорных именах следует обращаться к документации по конкретному компилятору.

Таблица 2.3.

Макроимя	Описание
__LINE__	Десятичная константа – номер текущей обрабатываемой строки файла с программой на языке С.
__FILE__	Строка символов – имя компилируемого файла. Имя изменяется всякий раз, когда препроцессор включает директиву #include с указанием имени другого файла. Когда включения файла по команде #include завершаются, восстанавливается предыдущее значение макроимени __FILE__.
__DATE__	Строка символов в формате “месяц число год”, определяющая дату начала обработки исходного файла.
__TIME__	Строка символов вида “часы:минуты:секунды”, определяющая время начала обработки препроцессором исходного файла

_STDC__	Константа, равная 1, если компилятор работает в соответствии с ANSI-стандартом. В противном случае, значение макроимени не определено.

Директива `#include`

Директива препроцессора `#include` обязывает компилятор включить либо стандартный заголовок, либо другой исходный файл, имя которого указано в директиве `#include`.

Формат использования директивы

```
#include <имя_файла>
#include "имя_файла"
#include имя_макроса
```

Отличительной особенностью директивы является указание имени файла в угловых скобках, либо в двойных кавычках.

- *Если имя файла заключено в угловые скобки, то поиск файла будет осуществляться в одном или нескольких специальных каталогах, определенных конкретной реализацией.*
- *Если имя файла заключено в кавычки, поиск файла выполняется в текущем (рабочем) каталоге. Если файл не найден, то поиск повторяется с использованием первого способа (как если бы имя файла было заключено в угловые скобки).*

Директивы `#if, #ifdef, #ifndef, #else, #endif, #elif`

Директивы этой группы называются директивами условной компиляции. Эти директивы позволяют избирательно компилировать части исходного кода.

Если выражение, стоящее после директивы **`#if`** оказывается истинным, то будет скомпилирован код, расположенный между нею и директивой **`#endif`**.

```
#if условное_выражение
     последовательность инструкций
#endif
```

Директива **`#else`** определяет альтернативу на случай невыполнения директивы `#if`.

Формат использования.

```
#if условное_выражение
     последовательность инструкций 1
#else
     последовательность инструкций 2
#endif
```

Пример:

```
#define MAX 8
…
void main(){
     #if MAX == 0
          printf("MAX равен 8");
     #else
          printf("MAX не равен 8");
     #endif
}
```

Для организации многозвенной схемы ветвлений введена директива **#elif**.

```
#if выражение_1
     последовательность инструкций 1
#elif выражение_2
     последовательность инструкций 2
#elif выражение_3
     последовательность инструкций 3
//…
#elif выражение_N
     последовательность инструкций N
#endif
```

С помощью директивы **#ifdef** проверяется, определен ли с помощью директивы `#define` к текущему моменту идентификатор, помещенный после `#ifdef`.

```
#ifdef макроимя
     последовательность инструкций
#endif
```

В директиве **#ifndef** проверяется обратное условие (неопределенность идентификатора).

```
#ifndef макроимя
     послдеовательность инструкций
#endif
```

Директива #line

Директива `#line` используется для нумерации строк.

```
#line константа
```

Директива указывает компилятору, что следующая ниже строка текста имеет номер, определяемый целой десятичной константой. Директива может одновременно изменять не только номер строки, но и имя файла

```
#line константа "имя_файла"
```

После препроцессорной обработки каждая строка имеет следующий вид:

имя_файла номер_строки текст_на языке_С

Директива #error

Директива дает указание компилятору остановить компиляцию. Используется в основном для отладки.

```
#error сообщение
```

При встречи директивы #error отображается заданное сообщение и другая информация (зависит от конкретной реализации рабочей среды), после чего компиляция прекращается.

Пример:

```
#define N 5
#if(N != 5)
     #error N должно быть меньше 5!
#endif
```

Пустая директива

Существует директива, использование которой не вызывает никаких действий. Она имеет вид

```
#
```

Директива #pragma

#pragma имя

Директива определяет действия, зависящие от конкретной реализации компилятора.

Она позволяет выдавать компилятору различные инструкции. Здесь элемент имя представляет имя желаемой #pragma-инструкции. Если указанное имя не распознается компилятором, директива #pragma попросту игнорируется без сообщения об ошибке.

Для получения подробной информации о возможных вариантах использования директивы #pragma стоит обратиться с системной документации по используемому компилятору.

Операция defined

При условной обработке текста (с использованием `#if #endif`) для упрощения записи сложного условия выбора можно использовать унарную операцию defined.

```
defined орперанд,
```

где операнд – либо идентификатор, либо заключенный в скобки идентификатор, либо обращение к макросу.

```
#if defined N
```

эквивалентно

```
#ifdef N
```

Еще пример использования операции defined

```
#if defined N && !defined M
```

```
    последовательность инструкций
#endif
```

Операция #

Оператор преобразует следующий за ним аргумент в строку, заключенную в кавычки. Для понимания рассмотрим пример.

```
#define print(A) printf(#A"=%f", A)
double a = 3.14159;
print(sin(a/2));    //sin(a/2)=1.0
```

Операция ##

Операция, допускаемая только между лексемами строки замещения, позволяет выполнять конкатенацию лексем, включаемых в строку замещения.

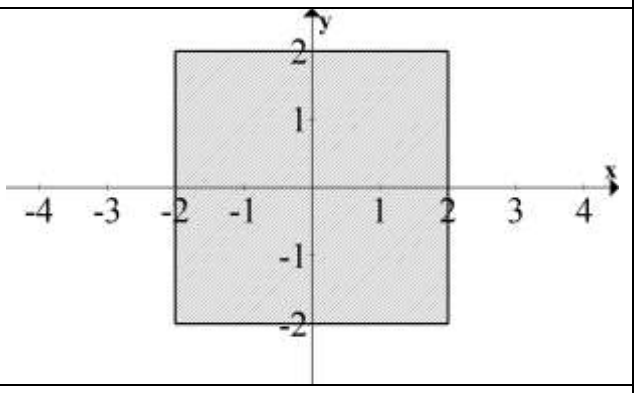
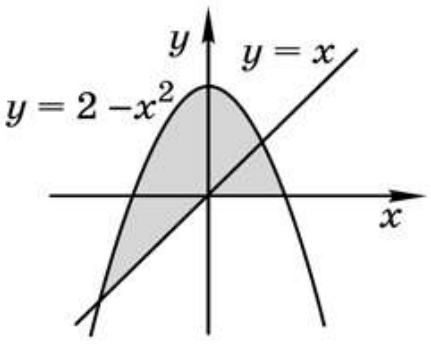
Рассмотрим наглядно пример использования операции ##.

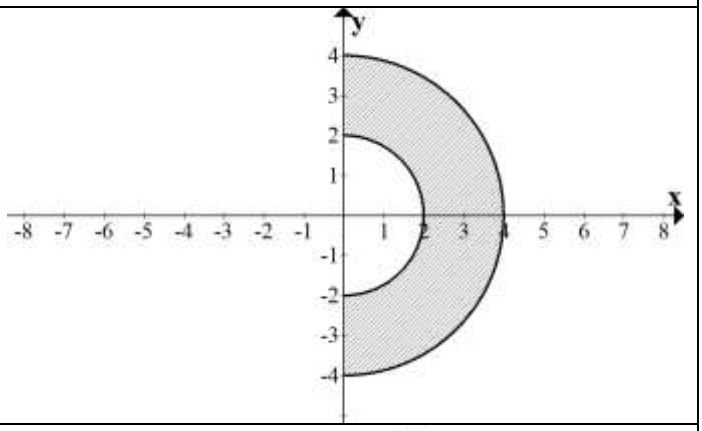
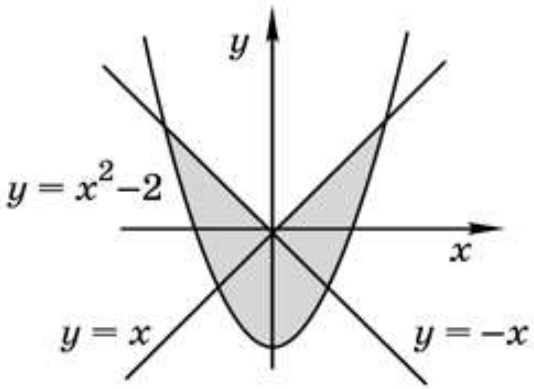
```
#define one(a, b, c, d) a(bcd)
#define two(a, b, c, d) a(b c d)
#define three(a, b, c, d) a(b##c##d)

one(sin, x, +, y)   //sin(bcd)
two(sin, x, +, y)   //sin(x + y)
three(sin, x, +, y) //sin(x+y).
```

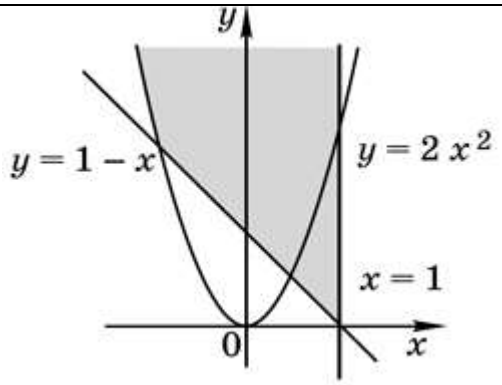
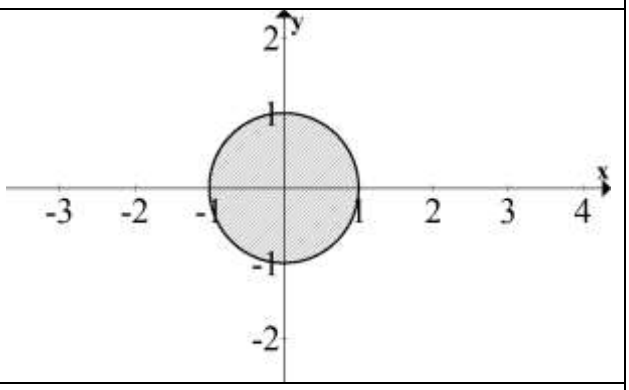
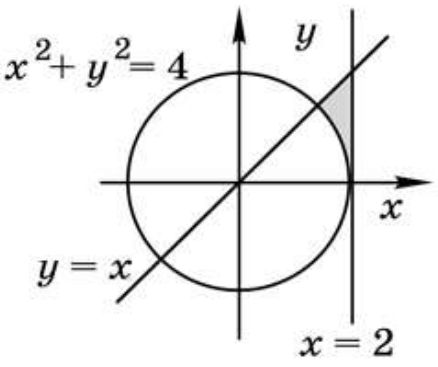
В случае one последовательность “bcd” воспринимается как отдельный идентификатор.

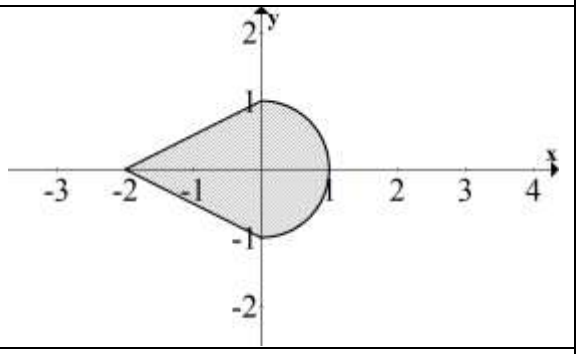
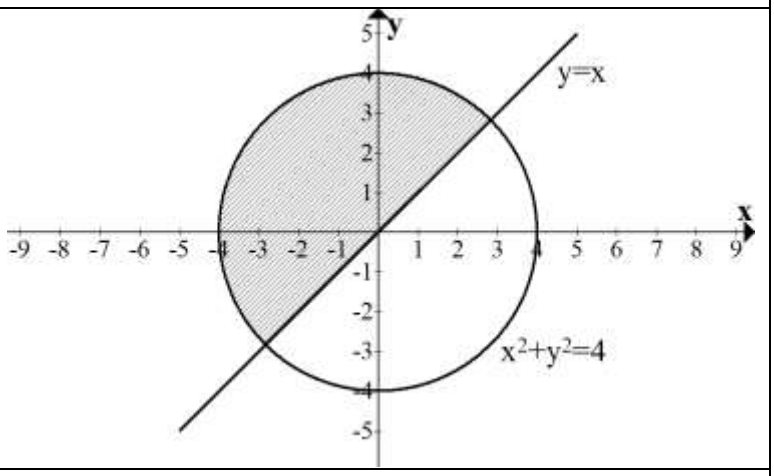
Задание

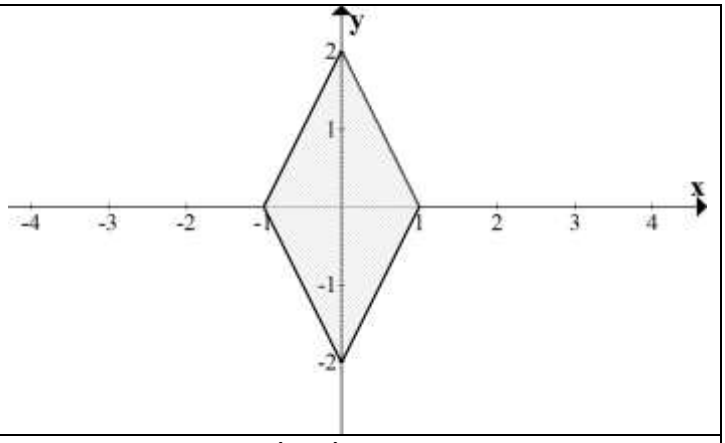
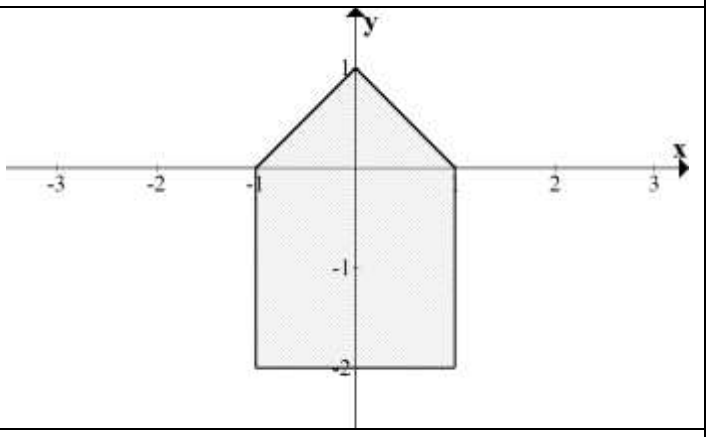
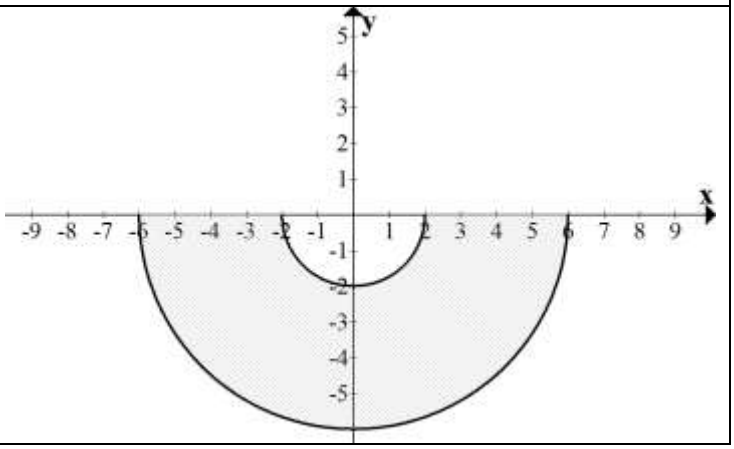
Используя директивы препроцессора, напишите программу в соответствии с заданным вариантом.

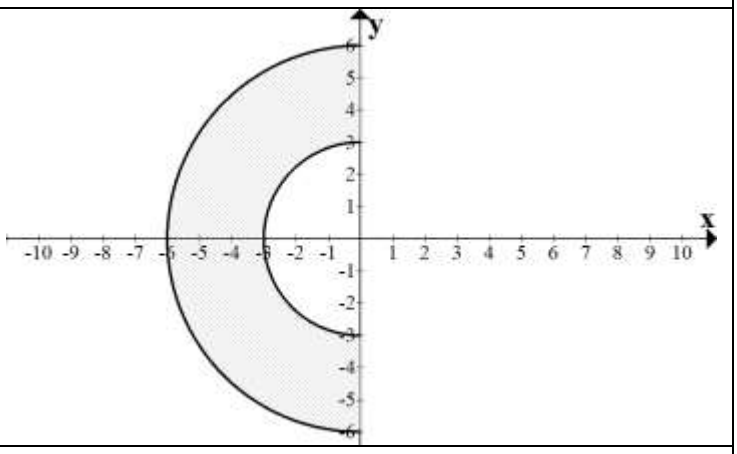
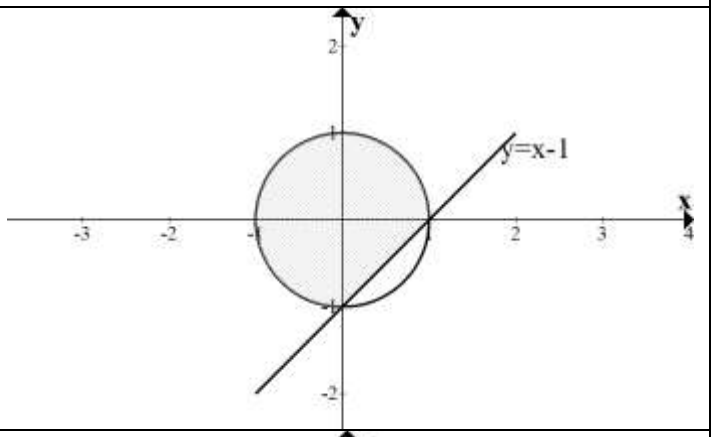
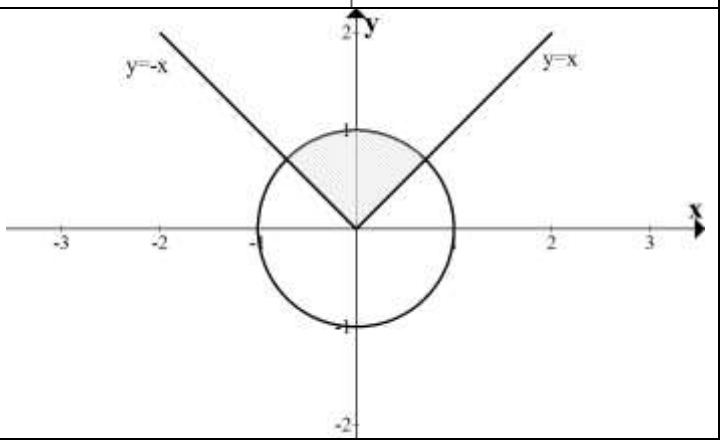
№	Задание
1	$$y=\begin{cases}\dfrac{1}{\lvert x+\sin(x)\rvert}, \text{если } x<0\\ 0, \text{если } x=0\\ x+\sqrt{3\sin(x)}, \text{если } x>0\end{cases}$$ Идентификатор *X* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификатора.
2	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
3	$$y=\begin{cases}2x+20, \text{если } 0\le x\le 40\\ 100, \text{если } 40\le x\le 60\\ -\dfrac{2}{3}x+140, \text{если } 60<x<150\end{cases}$$ Идентификатор *X* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификатора.
4	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 

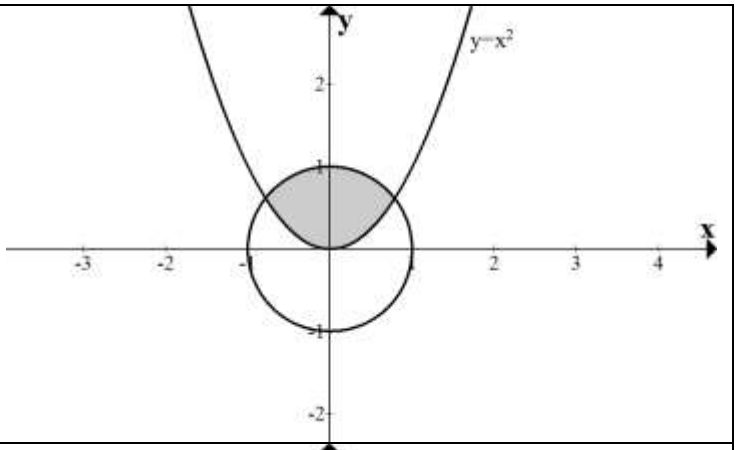
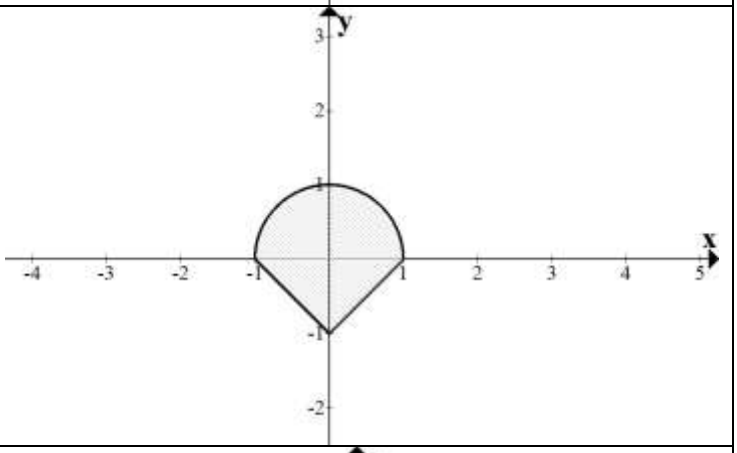
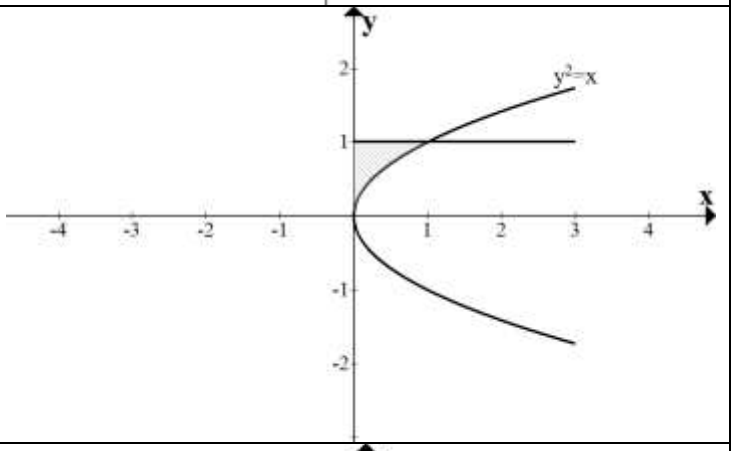
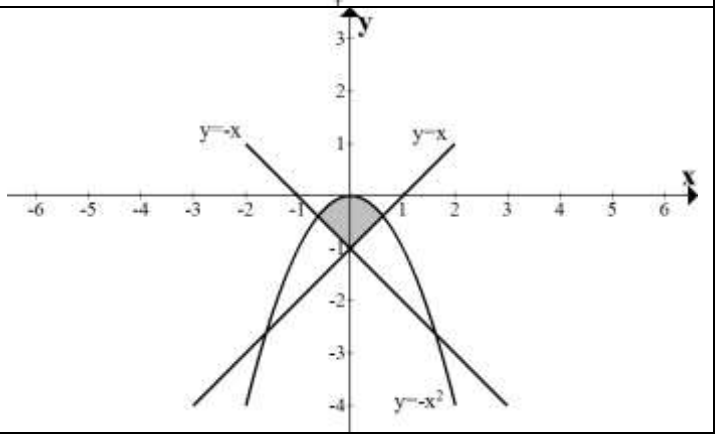
5	Вычислите значение функции в точках $-\frac{\pi}{2}, -\frac{\pi}{3}, 0, \frac{\pi}{2}$ $f(x) = \begin{cases} \sin(x), & \text{если } x > 0 \\ \cos(x) - 1, & \text{если } x \le 0 \end{cases}$ В качестве результата вывести значения функции в заданных точках, через запятую.
6	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
7	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
8	С помощью макроопределения определите функцию нахождения минимального из двух чисел и сравните два числа *A* и *B*, определенные с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов *A* и *B*. $A = \frac{5}{3-\sqrt{2}} + \frac{5}{3+\sqrt{2}}$ $B = \frac{3-\sqrt{5}}{3+\sqrt{5}} + \frac{3+\sqrt{5}}{3-\sqrt{5}}$
9	Даны координаты точки плоскости (x, y). Напишите программу, определяющую область, которой принадлежит точка: или номер четверти, или имя оси, или начало координат. Идентификаторы *X, Y* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов.

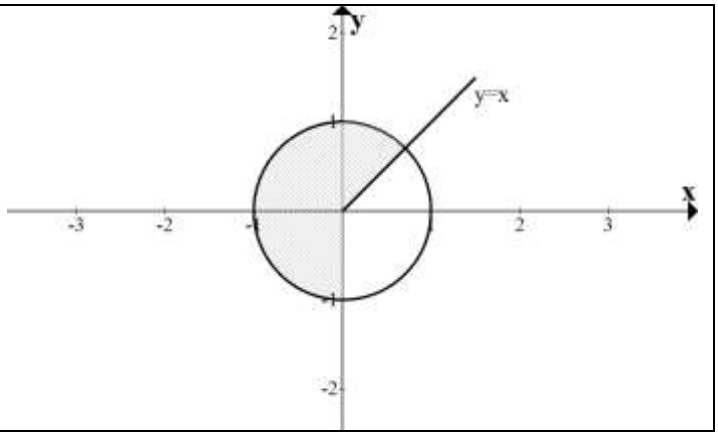
10	Напишите программу для решения квадратного уравнения вида $ax^2+bx+c=0$. Идентификаторы *A,B,C* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. В качестве результата необходимо вывести корни в порядке возрастания или сообщение об отсутствии корней.
11	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
12	Среди трех чисел A, B, C найдите среднее. Если числа равны, вывести сообщение. Идентификаторы *A, B, C* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов.
13	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
14	$y=\begin{cases}4x+10, \text{ если } x \geq 0\\ 8+x, \text{ если } x<0\end{cases}$ Идентификатор *X* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификатора.
15	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 

16	$$y=\begin{cases}-2x, \text{ если } x<0\\ x, \text{ если } 0\le x\le 1\\ 1, \text{ если } 1<x<3\\ -3x+10, \text{ если } x\ge 0\end{cases}$$ Идентификатор *X* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификатора.
17	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
18	$$y=\begin{cases}\sqrt{4-(x-2)^2}, \text{ если } 0\le x\le 4\\ -\sqrt{4-(x+2)^2}, \text{ если } 0>x\ge -4\\ 0, \text{ если } \lvert x\rvert>4\end{cases}$$ Идентификатор *X* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификатора.
19	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
20	$$F(a,b,c)=\begin{cases}a*b, \text{ если } c=1\\ a+b, \text{ если } c=2\\ a-b, \text{ если } c=3\end{cases}$$ Идентификаторы *A, B, C* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов.

21	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
22	Найти расстояние от точки плоскости с координатами (x, y) до границы окружности радиуса R с центром в начале координат. Идентификаторы *X, Y, R* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов.
23	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
24	$$y = \begin{cases} 1, \text{если } x = 0 \\ x^2, \text{если } x = 1 \\ \frac{1}{2}(4x^2 - 3), \text{если } x = 2 \\ \frac{1}{2}(7x^2 - 5x^2), \text{если } x = 3 \end{cases}$$ Идентификатор *X* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификатора.
25	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 

26	Напишите программу для проверки числа X на четность. Идентификатор *X* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификатора.
27	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
28	Напишите программу, определяющую, пройдет ли график функции $y = 2x^2 - 5x + 3$ через заданную точку с координатами (x, y). Идентификаторы *X, Y* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов.
29	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 
30	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов. 

31	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов.	
32	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов.	
33	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов.	
34	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов.	

35	Напишите программу определяющую, лежит ли точка с указанными координатами (x, y) внутри закрашенной области. Идентификаторы *X, Y,* определите с помощью директивы препроцессора. В ходе препроцессорной обработки необходимо обеспечить проверку существования идентификаторов.	

Контрольные вопросы

1. Что делает препроцессор?
   a) Производит пооператорную (покомандную) обработку и выполнение исходной программы.
   b) Подвергает программу различным текстовым преобразованиям до реальной трансляции исходного кода.
   c) Преобразует всю программу в модуль на машинном языке.
   d) Распределяет память для выходной программы.
   e) Нет правильного ответа.

2. Может ли имя макроса совпадать с именем какой-либо функции того же файла?
   a) Да.
   b) Нет.
   c) Может, но только в том случае, если в обоих именах отличается регистр букв.

3. Для чего предназначена директива #pragma?
   a) Для определения идентификатора и символьной последовательности
   b) Для включения в текст программы какого-либо файла
   c) Для определения действия, зависящего от конкретной реализации компилятора
   d) Для проверки существования идентификатора
   e) Нет правильного ответа.

4. Что будет выведено в результате выполнения следующей программы?

```
#define MAX(X, Y) X>Y ? printf(#X"=%d", X) : printf(#Y"=%d", Y)
int a = 6, b = 5;
MAX(a, b)
```

   a) 6 = 6
   b) 5 = 5
   c) a = 5
   d) b = 6
   e) a = 6
   f) b = 5
   g) Ошибка компиляции.
   h) Нет правильного ответа.

5. Какое из перечисленных макроопределений является правильным?
   a) #define MAX(x, y) x > y ? y : x
   b) #define MIN(x, y) x > y ? x : y
   c) #line MAX(x, y) x > y ? x : y
   d) #include MIN(x,y) x < y ? x : y
   e) #define MAX(x, y) x >y ? x : y
   f) Нет правильного ответа.

6. Выберите правильный вариант определения директивы препроцессора #include?
   a) #include <”имя_файла”>
   b) #include <’ имя_файла’>
   c) #include < имя_файла >
   d) #include ‘имя_файла’
   e) #include “имя_файла”
   f) Нет правильного ответа.

7. К директивам условной компиляции относят:
   a) #include
   b) #pragma
   c) #ifndef
   d) #error
   e) #elif
   f) #ifdef
   g) Нет правильного ответа.

8. Использование директивы #if обязательно предполагает наличие:
   a) #endif
   b) #else
   c) #elif
   d) defined
   e) Нет правильного ответа.

Лабораторная работа №3
Массивы и указатели

Перед тем, как приступить к рассмотрению массивов и указателей, необходимо ознакомиться с инструкциями управления, без которых невозможно обойтись при написании приложений. К управляющим инструкциям относятся: инструкции выбора, итерационные инструкции, инструкции перехода.

Инструкции выбора

Условный оператор `if`

Инструкция if позволяет сделать выбор между несколькими выполняемыми ветвями программы.

Обобщенный формат оператора if:

```
if(условие){
     последовательность инструкций 1;
}
else{
     последовательность инструкций 2;
}
```

Конструкция `if-else-if`

```
if(условие_1){
     последовательность инструкций 1;
} else
     if(условие_2){
          последовательность инструкций 2;
     } else
     ….
          if(условие_n){
               последовательность инструкций n;
          }
```

Инструкция `switch`

Инструкция switch является инструкцией многонаправленного ветвления, которая позволяет выбирать одну из множества альтернатив.

Обобщенный формат:

```
switch(выражение){
     case константа_1:
          последовательность инструкций 1;
          break;
     …
     case константа_n:
          последовательность инструкций n;
          break;
     default: последовательность инструкций;
}
```

Элемент выражение должен при вычислении давать целочисленное или символьное значение. Инструкция break завершает выполнение кода, определенного инструкцией switch. Если ни одна из заданных case-констант не совпадает с результатом вычисления switch, то выполняется последовательность инструкций в ветви default. Ветвь default является необязательной и может быть опущена.

Итерационные инструкции

Цикл for

Цикл for является самым универсальным циклом в языке С. Это так называемый счетчик.

Обобщенный формат цикла for:

```
for(инициализация; условие; итерация){
     последовательность инструкций;
}
```

Пример: вывод чисел от 0 до 10

```
for(i = 0; i <= 10; i++)
     printf("%d", i);
```

Элементы инициализация, условие, итерация могут отсутствовать.

Цикл while

```
while(условие){
     последовательность инструкций;
}
```

Цикл do-while

```
do{
     последовательность инструкций;
} while(условие);
```

В отличие от цикла while, в цикле do-while условие выполнения проверяется в конце каждой итерации. Это означает, что цикл do-while выполнится, по крайней мере, хотя бы один раз.

Инструкции перехода

Инструкция continue

Инструкция continue позволяет немедленно перейти к следующей итерации цикла. Используется совместно с операторами for, while, do-while.

Инструкция break

Инструкция break позволяет немедленной выйти из цикла. Используется совместно с операторами for, while, do-while.

Массивы

Массив – это структурированный тип данных, состоящий из фиксированного числа упорядоченных элементов, имеющих один и тот же тип.

Массивы различают на одномерные и n-мерные. Доступ к отдельному элементу массива осуществляется с помощью индекса, который описывает позицию элемента внутри массива. В языке С первый элемент массива имеет нулевой индекс.

Строка – это символьный массив, который завершается нулевым символом.

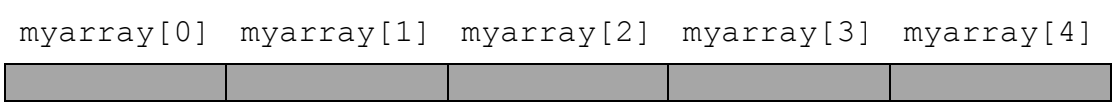
В языке С не выполняется проверка “нарушения границ” массива. Вся ответственность за соблюдение границ массива лежит только на программистах. В случае выхода за границы массива в программе, последствия – непредсказуемы.

Обобщенная форма определения одномерного массива:

```
тип имя_массива[размер];
```

Пример:

```
int myarray[5];
```



При определении массивов возможна их инициализация.

```
тип имя_массива[размер] = {список_значений};
тип имя_массива[] = {список_значений};
```

В последнем случае размер массива вычисляется автоматически на этапе компиляции.

Пример:

```
int myarray2[] = {0, 9, 1, 2, 8, 3};
```

myarray2[0]	myarray2[1]	myarray2[2]	myarray2[3]	myarray2[4]	myarray2[5]
0	**9**	**1**	**2**	**8**	**3**

```
char str[] = “Эта строка является символьным массивом”;
```

str[0]	str[1]	str[2]	…	str[38]	str[39]
Э	**т**	**а**	**…**	**м**	**\0**

Помимо одномерных массивов можно использовать многомерные массивы.

Обобщенная форма определения многомерных массивов:

```
тип имя_массива[размер_1][размер_2]…[размер_n];
тип имя_массива[размер_1]…[размер_n] = {{список_значений_1},…,{список_значений_n}};
```

Простейший пример многомерного массива – двумерный массив.

Пример:

```
int myarray3[4][5] = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {11, 12,
13, 14, 15}, {16, 17, 18, 19, 20}};
```

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20

Для того чтобы узнать размер массива, необходимо воспользоваться следующей инструкцией:

```
размер_массива = sizeof(имя_массива) / sizeof(первый_элемент_массива);
```

```
length1 = sizeof(myarray2) / sizeof(myarray2[0]);
length2 = sizeof(myarray3[0]) / sizeof(myarray[0][0]);
```

Оператор **`sizeof`** предназначена для получения размера операнда в байтах.

Указатели

Указатель – переменная, значением которой является адрес ячейки памяти.

Формат определения указателя:

```
тип *имя_переменной;
```

Тип указателя определяет тип данных, на которые он будет ссылаться. Если тип указателя отличается от типа переменной, на которую он ссылается, то на этапе компиляции возникнет ошибка.

Операция непрямого доступа – это процесс использования указателей для доступа к некоторому объекту.

С указателями используются два оператора: “`&`” и “`*`”. Оператор “`&`” – унарный, возвращает адрес памяти, по которому расположен его операнд. Оператор “`*`” – унарный, возвращает значение, расположенное по адресу, заданному его операндом.

Пример:

```
int a = 5;
int *ptr = &a; //переменная-указатель ptr содержит адрес переменной a
printf(“%d”, *ptr); // 5
```

С помощью указателей можно присваивать значения области памяти, на которые они указывают.

Пример

```
int a, *ptr;
ptr = &a;
*ptr = 10;
//a = 10
```

Значением указателя может быть нулевой адрес, т.е. значение указателя, заведомо, не равно никакому адресу. Для этого случая определена специальная константа `NULL` в заголовочном файле `stdio.h`.

Арифметические операции над указателями несколько отличаются от арифметических операций над переменными. Унарные операции “`&`” и “`*`” имеют более высокий приоритет.

Например, при инкрементировании или декрементировании значение указателя будет увеличиваться или уменьшаться на величину, равную размеру базового типа. Т.е., если указатель `int *ptr` содержит адрес 1000, то после выполнения операции инкрементирования `ptr++`, значение указателя будет равным 1004, а не 1001 (т.к. размер типа `int` в 32-разрядной среде равен 4 байта).

Помимо арифметических операций, над указателями можно применять операторы отношения.

Указатели и массивы

Имя массива без индекса образует указатель на начало этого массива. Рассмотрим пример.

```
//…
int arr[] = {9, 0, 3, 4};
printf("%d", *arr); //9
printf("%d", *(++arr));  //0
printf("%d", *(++arr));  //3
printf("%d", *(++arr));  //4
printf("%d", *(++arr));  //Эта строка приведет к ошибке, т.к. происходит нарушение границ массива
//…
```

В рассмотренном примере каждый раз происходит увеличение указателя на 1, тем самым осуществляется переход к следующему элементу массива.

Не всегда удобно использовать массивы с фиксированным числом элементов. Такой подход применяется для статического распределения памяти, когда память выделяется процессу в момент его порождения. Для использования массивов с переменным количеством элементов, применяются указатели и средства динамического распределения памяти.

Функции, представленные в таблице 3.1, динамически выделяют память и возвращают адрес начала выделенного участка памяти.

Таблица 3.1. 1

Функция	Описание
`void* malloc(unsigned n)`	Возвращает указатель на начало области (блока) динамической памяти длинной в n байт. При неудачном завершении возвращается значение NULL.
`void* calloc(unsigned n, unsigned m)`	Возвращает указатель на начало области (блока) обнуленной динамической памяти, выделенной для размещения n элементов по m байт каждый. При неудачном завершении возвращает значение NULL.
`void* realloc(void *ptr, unsigned nsize)`	Изменяет размер блока ранее выделенной динамической памяти до размера nsize байт. ptr – адрес начала изменяемого блока. Если ptr равер NULL(память не выделялась), то функция выполняется как malloc. Функция может перемещать блок памяти, в этом случае функция возвращает указатель на новый адрес блока в памяти.

`void *free(void *ptr)`	Освобождает ранее выделенный участок (блок) памяти, адрес первого байта которого равен значению ptr.

«Мы в ответе за тех, кого приручили»
по повести А. Сент-Экзюпери «Маленький принц»

Необходимо запомнить ГЛАВНОЕ ПРАВИЛО: ВСЕГДА освобождайте выделенную память!!! Память, выделенная динамически, по завершению программы автоматически НЕ освобождается!!!

Из таблицы видно, что функции `calloc` и `malloc` возвращают тип `void*`. Для того, чтобы задать тип размещаемого объекта, необходимо воспользоваться операцией приведения типа.

Для того, чтобы сделать операцию независимой от архитектуры ЭВМ и ОС, при выделении памяти используют операцию `sizeof(тип)`.

Обобщенная форма динамического выделения памяти для одномерного массива:

```
тип *имя_указателя = (тип*)malloc(размер * sizeof(тип));
тип *имя_указателя = (тип*)calloc(размер, sizeof(тип));
тип *имя_указателя1 = (тип*)realloc(имя_указателя2, размер * sizeof(тип))
realloc(имя_указателя, размер * sizeof(тип));
```

Пример динамического выделения памяти:

```
int *arr_int;
float *arr_float;
//выделяется память для 5 элементов типа int.
arr_int = (int *)calloc(5, sizeof(int));
//выделяется память для 5 элементов типа int.
arr_float = (float*)malloc(10 * sizeof(float));
//изменяется размер блока, на который ссылается указатель arr_int
realloc(arr_int, 10 * sizeof(20));
//…
//освобождение ранее выделенной памяти
free(arr_int);
free(arr_float);
//…
```

После динамического выделения памяти, к отдельным элементам можно обращаться двумя способами:

- Как к элементам массива:
  ```
  printf("%d", arr_int[5]);     //0
  arr_float[4] = 3.14;
  ```
- С помощью указателей:
  ```
  *(имя_указателя + номер_элемента);
  printf("%d", *(arr_int + 5)); //0
  *(arr_float + 4) = 3.14;
  ```

Вместо многомерных массивов можно использовать массивы указателей.

Обобщенная форма динамического выделения памяти под массив указателей:

```
тип *имя_указателя[размер];
for(i = 0; i < размер; i++){
     имя_указателя[i] = (тип*)calloc(размер, sizeof(тип));
}
//…
for(i = 0; i < размер; i++)
     free(имя_указателя[i]);
```

ИЛИ

```
тип **имя_указателя;
имя_указателя = (тип**)calloc(размер_1, sizeof(тип*));
for(i = 0; i < размер_1; i++){
имя_указателя[i] = (тип*)calloc(размер_2, sizeof(тип));
}
//…
for(i = 0; i < размер_1; i++)
     free(имя_указателя[i]);
free(имя_указателя);
```

Также, как и с одномерными массивами к элементам можно обращаться несколькими способами:

- Как к элементам массива:
  `имя_указателя[i][j];`
- С помощью указателей:
  `*(*(имя_указателя + i) + j);`

Задание

№	Задание
1	Написать программу, в которой с помощью средств динамического распределения памяти создается двумерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Создать два одномерных массива, первоначальная длина которых равна 0. В первый поместить элементы с четным значением, во второй – с нечетным значением. Размер одномерных массивов при добавлении увеличивается на единицу. Вывести на экран исходный двумерный массив и два одномерных. Размер массива: 10 x 10 Диапазон случайных значений: -100..100
2	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными числами с помощью функций randomize() и random(). Вывести на экран исходный массив и элементы с неотрицательным значением. Размер массива: 50 Диапазон случайных значений: -150…100
3	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными числами с помощью функции randomize() и random(). Заменить все элементы с отрицательные значения на положительные. Вывести на экран массив до и после замены. Размер массива: 50 Диапазон случайных значений: -100...50
4	Написать программу, в которой с помощью средств динамического распределения памяти создается двумерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Разделить каждый элемент массива на среднее арифметическое значение элементов строки, в которой они расположены. Вывести на экран исходный и полученный массивы, среднее арифметическое каждой строки. Размер массива: 20 x 15 Диапазон случайных значений: 0…200
5	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными числами с помощью функции randomize() и random(). Подсчитать количество пар соседних элементов с одинаковыми значениями. Вывести на экран исходный массив, количество пар, индексы первых элементов пар. Размер массива: 200 Диапазон случайных значений: 0…20

6	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными числами с помощью функции randomize() и random(). Ввести с клавиатуры N чисел из диапазона и добавить их в конец исходного массива. При этом размер исходного массива должен изменяться динамически. Необходимо обеспечить проверку корректности ввода чисел. Размер массива: 20 Диапазон значений: -60..60
7	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Заменить все элементы с отрицательным значением средним арифметическим значением всех положительных чисел. Вывести на экран массив до и после замены, среднее арифметическое всех положительных чисел. Размер массива: 50 Диапазон случайных значений: -50…30
8	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными числами с помощью функции randomize() и random(). Ввести с клавиатуры N чисел из диапазона и добавить их в начало исходного массива. При этом размер исходного массива должен изменяться динамически. Необходимо обеспечить проверку корректности ввода чисел. Размер массива: 30 Диапазон значений: -90..120
9	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными числами с помощью функции randomize() и random(). Вывести на экран массив, минимальное и максимальное значения массива. Размер массива: 20 Диапазон случайных значений: -200…200
10	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными числами с помощью функции randomize() и random(). Найти непрерывный участок из 10 элементов, сумма которых максимальна. Вывести на экран массив, индексы начала и конца участка, сумму элементов. Размер массива: 150 Диапазон случайных значений: 50…250
11	Написать программу, в которой с помощью средств динамического распределения памяти создается двумерный массив и заполняется случайными числами с помощью функции randomize() и random(). Вычислить сумму элементов каждого столбца. Вывести на экран массив, суммы столбцов. Размер массива: 20 x 20 Диапазон случайных значений: -100…100
12	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Найти количество пар соседних элементов, которые имеют одинаковые абсолютные значения, но противоположные знаки. Вывести на экран массив, количество пар, индексы первых элементов пар. Размер массива: 200 Диапазон случайных значений: -50…50

13	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Во всех последовательностях положительных чисел изменить порядок элементов на противоположный. Вывести на экран массив до и после замены. Размер массива: 50 Диапазон случайных значений: -50..50
14	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный символьный массив и случайным образом заполняется символами кириллицы латиницы, а также знаками препинания. Ввести строку S. Проверить, может ли строка S быть составлена из символов массива. Регистр букв можно не учитывать. Рекомендуется использовать таблицу ASCII-кодов.
15	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Подсчитать количество всех, непрерывных последовательностей положительных чисел, длина которых больше 7. Вывести на экран массив, количество последовательностей и индексы начальных элементов последовательностей. Размер массива: 200 Диапазон случайных значений: -50..50
16	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Каждый элемент заменить на среднее арифметическое его и его соседей слева и справа. Вывести на экран массив до и после замены. Размер массива: 100 Диапазон случайных значений: -100..100
17	Написать программу, в которой с помощью средств динамического распределения памяти создается двумерный массив. Заполнить массив таким образом, чтобы значение каждого элемента равно сумме номера строки и столбца, на пересечении которых он находится. Вычислить сумму элементов каждой строки. Вывести на экран массив, суммы элементов строк. Размер массива: 15 x 15
18	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Для каждого положительного элемента определить, есть ли в массиве отрицательный элемент с противоположным значением; если да, заменить эти элементы на 0. Вывести на экран массив до и после замены. Размер массива: 200 Диапазон случайных значений: -50..50

19	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Поменять местами 1-й положительный элемент с последним положительным элементом, 2-й - предпоследним и т.д. Вывести на экран массив до и после замены. Размер массива: 50 Диапазон случайных значений: -80..60
20	Написать программу, в которой с помощью средств динамического распределения памяти создается матрица и заполняется случайными значениями с помощью функции randomize() и random(). Вычислить произведение суммы элементов главное диагонали на сумму элементов i-ой строки. Вывести на экран исходную матрицу и полученное произведение. Размер матрицы: 15 x 15 Диапазон случайных значений: 40…100
21	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Во всех последовательностях отрицательных чисел поменять местами элементы с максимальным и минимальным значениями этих последовательностей. Вывести на экран массив до и после замены. Размер массива: 50 Диапазон случайных значений: -100..100
22	Написать программу, в которой с помощью средств динамического распределения памяти создается матрица и заполняется случайными значениями с помощью функции randomize() и random(). Вычислить определитель матрицы. Вывести на экран исходную матрицу и вычисленный определитель. Размер матрицы: 3 x 3 Диапазон значений: -15…15
23	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Заменить в массиве все вхождения минимальных значений на максимальное значение. Вывести на экран массив до и после замены, минимальное и максимальное значения. Размер массива: 50 Диапазон случайных значений: -40..40
24	Написать программу, в которой с помощью средств динамического распределения памяти создается матрица и заполняется случайными значениями с помощью функции randomize() и random(). Найти сумму минимальных элементов главной и побочной диагоналей. Вывести на экран матрицу, минимальные элементы и их сумму. Размер матрицы 50 x 50 Диапазон случайных значений: -120..80

25	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Заменить в массиве все вхождения минимальных значений на последний элемент массива. Вывести на экран массив до и после замены, минимальное значение, индексы замененных элементов. Размер массива: 100 Диапазон случайных значений: 0..100
26	Написать программу, в которой с помощью средств динамического распределения памяти создается матрица и заполняется случайными значениями с помощью функции randomize() и random(). Найти сумму максимальных элементов главной и побочной диагоналей. Вывести на экран матрицу, максимальные элементы и их сумму. Размер матрицы 50 x 50 Диапазон случайных значений: -120..80
27	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Заменить в массиве все вхождения максимальных значений на последний элемент массива. Вывести на экран массив до и после замены, максимальное значение, индексы замененных элементов. Размер массива: 100 Диапазон случайных значений: -30..70
28	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Вычислить количество различных чисел в массиве. Вывести на экран массив и количество различных чисел. Размер массива: 400 Диапазон случайных значений: -1500…1500
29	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Ввести целое число A из диапазона значений. Найти длину первой непрерывной последовательности из значений меньше A в массиве. Вывести на экран массив, индекс первого и последнего элементов последовательности. Размер массива: 100 Диапазон случайных значений: -30..70

30	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Ввести целые числа A и B из диапазона значений. Найти длину первой непрерывной последовательности из значений в диапазоне от A до B в массиве. Вывести на экран массив, длину последовательности, индекс первого и последнего элементов последовательности. Размер массива: 100 Диапазон случайных значений: -90..270
31	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Ввести целое число A из диапазона значений. Определить максимальную длину непрерывной последовательности из значений, делящихся без остатка на A, в массиве. Вывести на экран массив, индекс первого и последнего элементов последовательности. Размер массива: 100 Диапазон случайных значений: 0..70
32	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Определить максимальную длину непрерывной последовательности из нечетных значений в массиве. Вывести на экран массив, индекс первого и последнего элементов последовательности. Размер массива: 100 Диапазон случайных значений: -100..1000
33	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Определить максимальную длину непрерывной последовательности из отрицательных значений в массиве. Вывести на экран массив, индекс первого и последнего элементов последовательности. Размер массива: 100 Диапазон случайных значений: -30..70
34	Написать программу, в которой с помощью средств динамического распределения памяти создается одномерный массив и заполняется случайными значениями с помощью функции randomize() и random(). Определить максимальную длину непрерывной последовательности из положительных значений в массиве. Вывести на экран массив, индекс первого и последнего элементов последовательности. Размер массива: 100 Диапазон случайных значений: -30..70
35	Написать программу, в которой с помощью средств динамического распределения памяти создается матрица и заполняется случайными значениями с помощью функции randomize() и random(). Наименьший элемент каждой строки, начиная со второй заменить наибольшим элементом предшествующей строки. Вывести на экран матрицу до и после замены. Размер матрицы: 20 x 20 Диапазон случайных значений: 10..99

Контрольные вопросы

1. Выберите правильный вариант прототипа функции realloc(…)?
   a) void realloc(void *s);
   b) void realloc(unsigned s);
   c) void realloc(unsigned n, unsigned m);
   d) void *realloc(unsigned n, unsigned m);
   e) void *realloc(void *n, unsigned m);
   f) void *realloc(unsigned s);
   g) Нет правильного ответа.

2. Как инициализировать массив символов myarray, состоящий из 10 символов?
   a)
   ```
   for(i = 0; i <= 10; i++)
        scanf("%c", myarray);
   ```
   b)
   ```
   for(i = 0; i <= 10; i++)
        scanf("%c", myarray[i]);
   ```
   c)
   ```
   for(i = 0; i < 10; i++)
        scanf("%c", myarray[i]);
   ```
   d)
   ```
   for(i = 0; i < 10; i++)
        scanf("%c", myarray[0]);
   ```
   e) Нет правильного ответа.

3. Как правильно получить значение седьмого элемента массива myarray?
   a) myarray[7];
   b) myarray[6];
   c) *(myarray + 7);
   d) myarray + 6;
   e) Нет правильного ответа

4. Выберите правильный вариант прототипа функции malloc(…)?
   a) void *malloc(unsigned s);
   b) void malloc(unsigned s);
   c) void malloc(unsigned n, unsigned m);
   d) void *malloc(unsigned n, unsigned m);
   e) void malloc(void *s);
   f) void *malloc(void *n, unsigned m);
   g) Нет правильного ответа.

5. Что хранится в указателе?
   a) Размер некоторого объекта.
   b) Размер адресного пространства, выделенного программе.
   c) Значение некоторого объекта
   d) Адрес некоторого объекта
   e) Нет правильного ответа.

6. Выберите правильный вариант прототипа функции calloc(…)?
   a) void *calloc(unsigned s);
   b) void calloc(unsigned n, unsigned m);
   c) void calloc(unsigned s);
   d) void calloc(void *s);
   e) void *calloc(unsigned n, unsigned m);
   f) void *calloc(void *n, unsigned m);
   g) Нет правильного ответа.

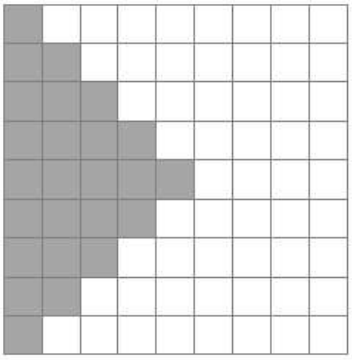
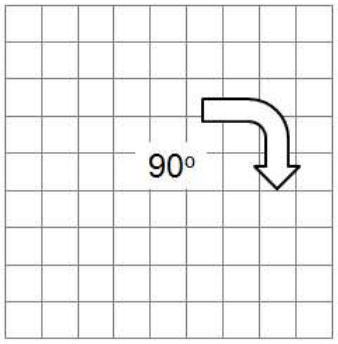
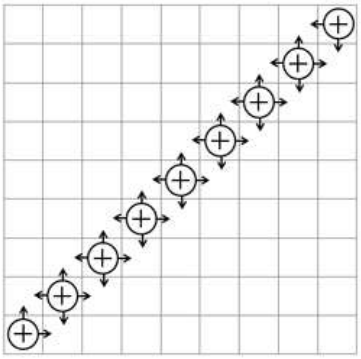
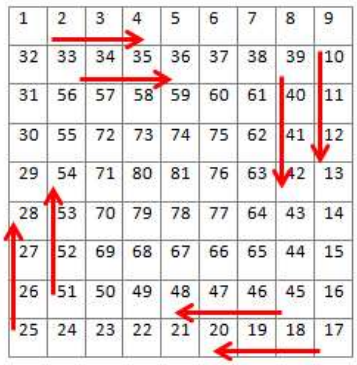
7. Как объявить двумерный массив чисел myarray типа int, состоящий из 5 строк и 7 столбцов?
   a) float myarray[5][7];
   b) myarray[5][];
   c) int myarray[][7];
   d) int myarray[5][7];
   e) Нет правильного ответа

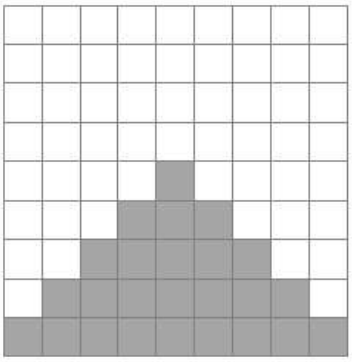
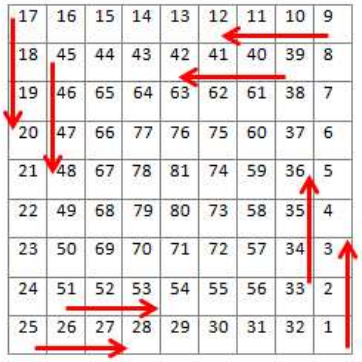
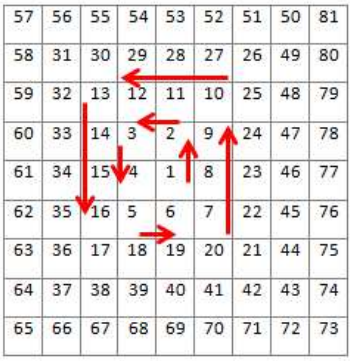
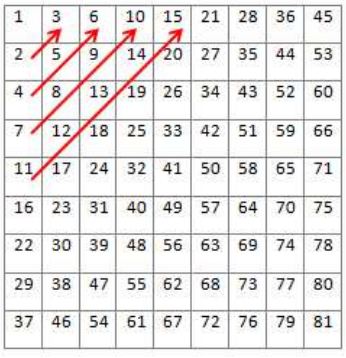
8. Как записать в указатель ptr адрес переменной n типа int?
   a) ptr = *n;
   b) ptr = n;
   c) ptr = &n;
   d) int ptr = n;
   e) Нет правильного ответа.

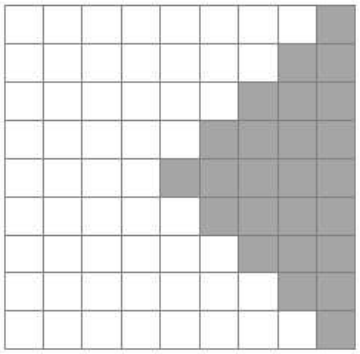
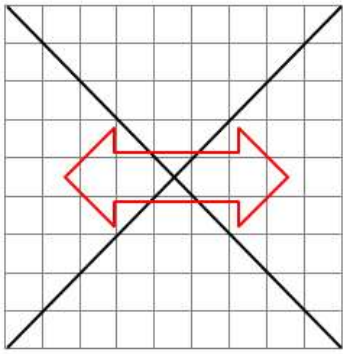
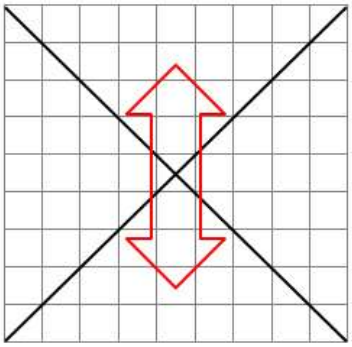
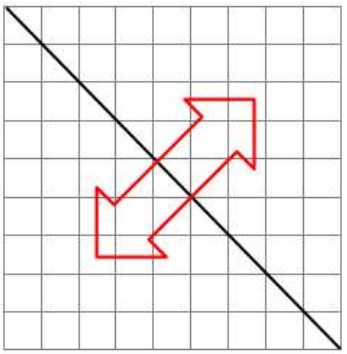
9. Какая функция логически является лишней?
   a) calloc(…);
   b) malloc(…);
   c) realloc(…);
   d) free(…);
   e) Нет правильного ответа
   f) Нет правильного ответа.

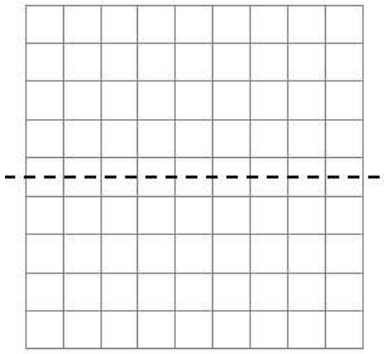
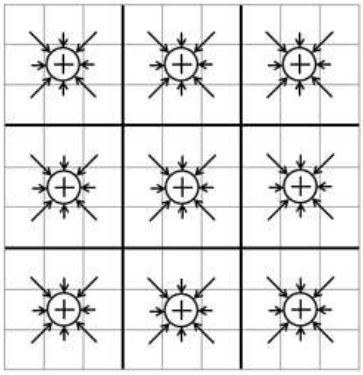
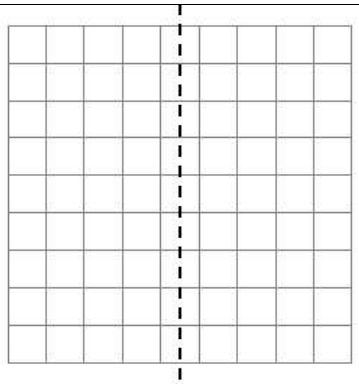
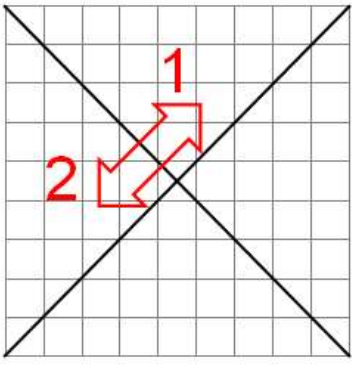
10. Выберите правильный вариант прототипа функции free(…)?
    a) void *free(unsigned s);
    b) void free(unsigned s);
    c) void *free(unsigned n, unsigned m);
    d) void free(unsigned n, unsigned m);
    e) void *free(void *n, unsigned m);
    f) void free(void *s);
    g) Нет правильного ответа.

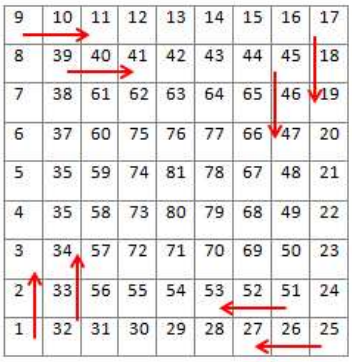
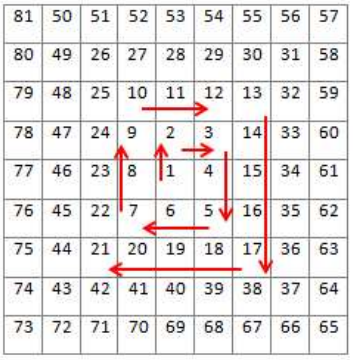
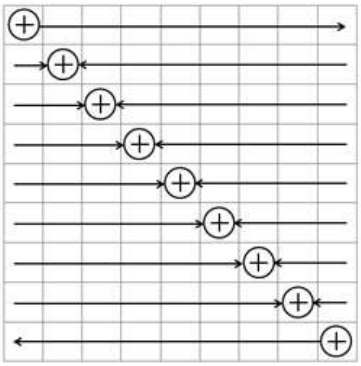
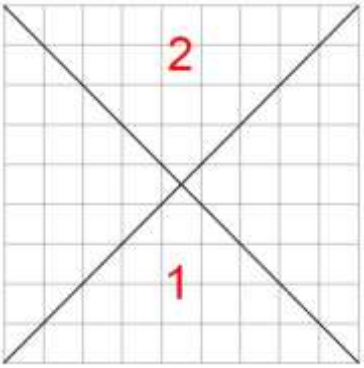
Лабораторная работа №4
Матрицы и действия над ними

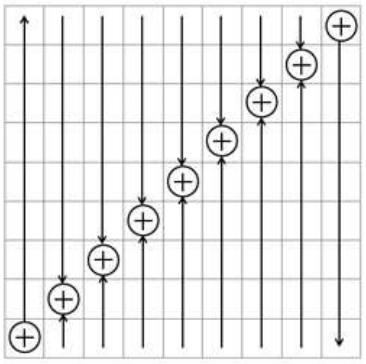
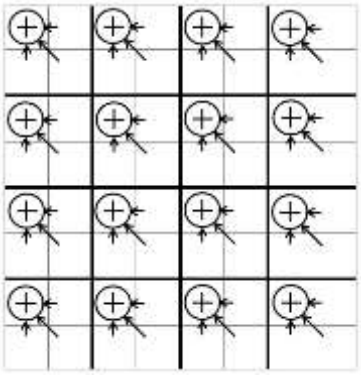
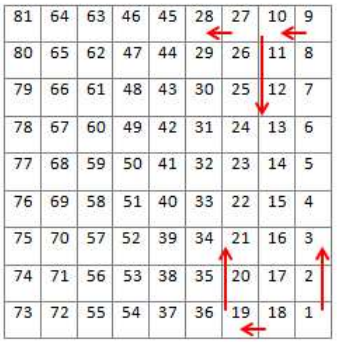
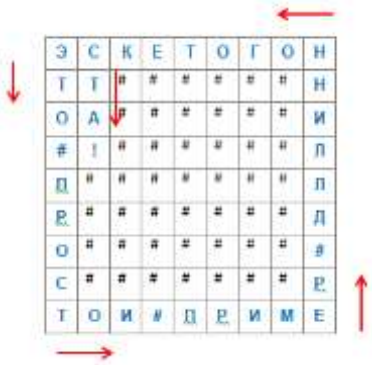
№	Задание	Иллюстрация
1	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Найти максимальный элемент в закрашенной области. Вывести на экран матрицу, максимальный элемент. Размер матрицы: 9 x 9 Диапазон случайных значений: -99…99	
2	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Развернуть матрицу на 90° по часовой стрелке. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: -40…40	
3	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Элементы побочной диагонали содержат значения суммы элементов, которые лежат на той же строке и том же столбце. Вывести матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: 0…50	
4	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел. Заполнить матрицу линейной последовательностью чисел (1, 2, 3, …) по спирали (от левого верхнего угла – вправо – вниз – влево - вверх). Вывести на экран матрицу. Размер матрицы: 9 x 9	

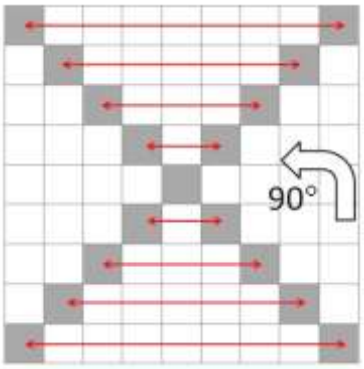
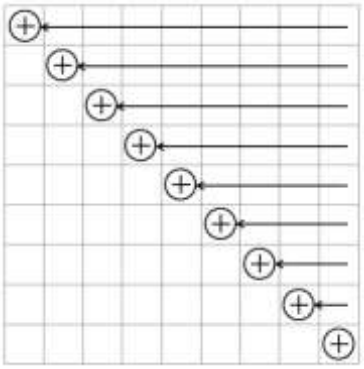
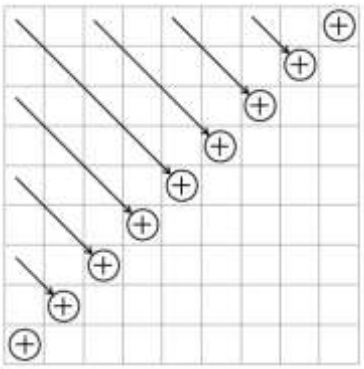
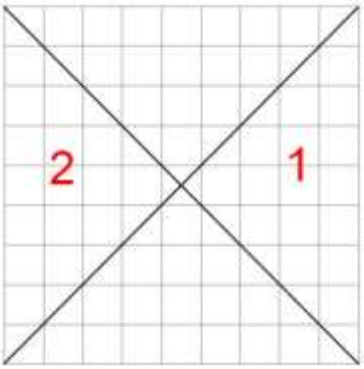
5	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Найти максимальный элемент в закрашенной области. Вывести на экран матрицу, максимальный элемент. Размер матрицы: 9 x 9 Диапазон случайных значений: 0…70	
6	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел. Заполнить матрицу линейной последовательностью чисел (1, 2, 3, …) по спирали (от правого нижнего угла – вверх – влево – вниз - вправо). Вывести на экран матрицу. Размер матрицы: 9 x 9	
7	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел. Заполнить матрицу линейной последовательностью чисел (1, 2, 3, …) по спирали (от центра – вверх – влево – вниз - вправо). Вывести на экран матрицу. Размер матрицы: 9 x 9	
8	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел. Заполнить матрицу линейной последовательностью чисел (1, 2, 3, …) от левого верхнего угла по диагонали вправо - вверх. Вывести на экран матрицу. Размер матрицы: 9 x 9	

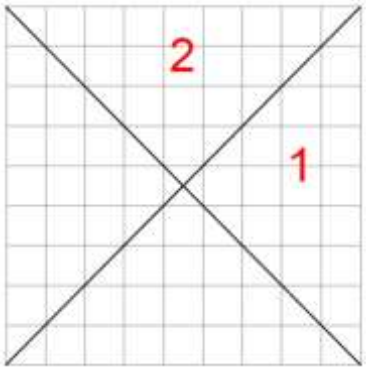
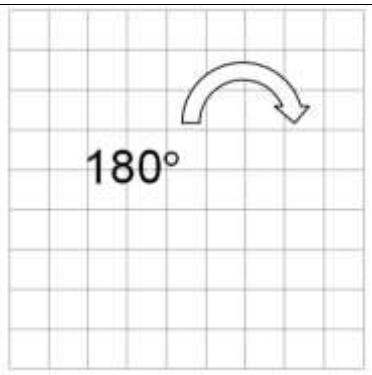
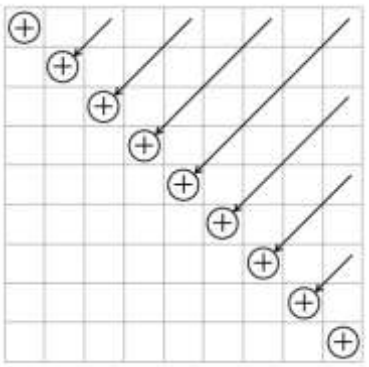
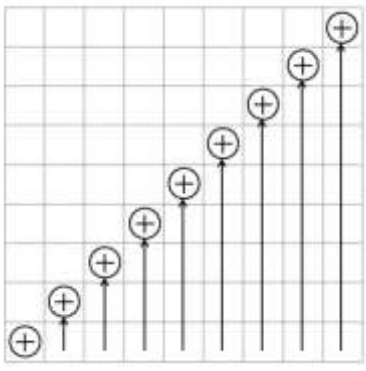
9	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Найти максимальный элемент в закрашенной области. Вывести на экран матрицу, максимальный элемент. Размер матрицы: 9 x 9 Диапазон случайных значений: -50…0	
10	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Отобразить симметрично, относительно вертикальной оси, секторы матрицы, которые лежат слева и справа от главной и побочной диагоналей. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: 0…30	
11	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Отобразить симметрично, относительно горизонтальной оси, секторы матрицы, которые лежат сверху и снизу от главной и побочной диагоналей. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: 30…60	
12	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Отобразить симметрично секторы матрицы, которые лежат сверху и снизу от главной диагонали. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: 60…90	

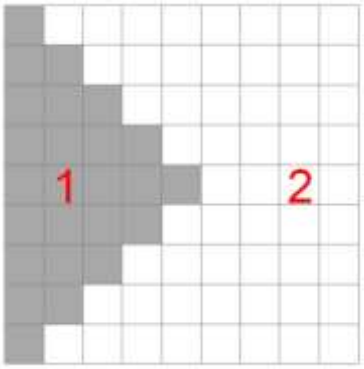
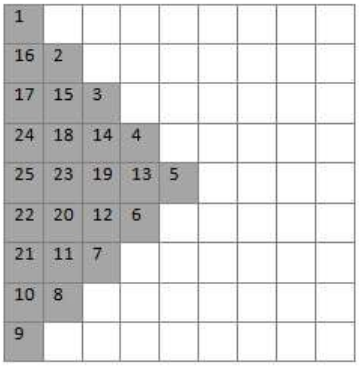
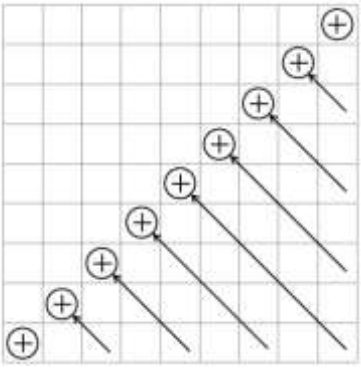
13	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Отобразить верхнюю и нижнюю половины зеркально относительно горизонтальной оси. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: -30…0	
14	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Разбить матрицу на квадраты размером 3x3. В центре каждого квадрата поместить сумму остальных элементов квадрата. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: 0…30	
15	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Отобразить правую и левую половины зеркально относительно вертикальной оси. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: 0…99	
16	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Отобразите симметрично, относительно главной диагонали, первый и второй сектора. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: 60…90	

17	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел. Заполнить матрицу линейной последовательностью чисел (1, 2, 3, …) по спирали (от левого нижнего угла – вверх – вправо – вниз - влево). Вывести на экран матрицу. Размер матрицы: 9 x 9	
18	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел. Заполнить матрицу линейной последовательностью чисел (1, 2, 3, …) по спирали (от центра – вверх – вправо – вниз - влево). Вывести на экран матрицу. Размер матрицы: 9 x 9	
19	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. В каждый элемент главной диагонали поместите сумму элементов строки, на которой он находится. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: -30…30	
20	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица вещественных чисел и заполняется случайными числами. Сравнить элементы двух секторов и вывести на экран информацию о том, в каком из секторов больше нулей, положительных и отрицательных чисел. А также вывести исходную матрицу. Размер матрицы: 9 x 9 Диапазон случайных значений: -70…50	

21	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. В каждый элемент побочной диагонали поместите сумму элементов столбца, на котором он находится. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: -60…60	
22	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Разбить матрицу на квадраты размером 2x2 и для каждого квадрата вычислить сумму его элементов. Вывести на экран матрицу до и после преобразования. Размер матрицы: 8 x 8 Диапазон случайных значений: 50…99	
23	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел. Заполнить матрицу линейной последовательностью чисел (1, 2, 3, …) в виде змейки (от правого нижнего угла – вверх). Вывести на экран матрицу. Размер матрицы: 9 x 9	
24	Написать программу, в которой с помощью средств динамического распределения памяти создается символьная квадратная матрица. Ввести строку S. Записать строку S в виде спирали в матрицу (с левого верхнего угла – вниз – вправо – вверх – влево.). Пробелы, а также пустые элементы матрицы заполнить символом #. Размер матрицы: 9 x 9	

25	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Отобразить симметрично элементы главной и побочной диагонали. Повернуть матрицу против часовой стрелки на 90°. Вывести на экран исходную матрицу, матрицу после замены элементов диагоналей, матрицу после поворота. Размер матрицы: 9 x 9 Диапазон случайных чисел: -40…40	
26	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Для каждого элемента побочной диагонали рассчитать сумму элементов, как показано на иллюстрации. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных чисел: -20…20	
27	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Для каждого элемента побочной диагонали вычислить сумму так, как показано на иллюстрации. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных чисел: -35…35	
28	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Найти в секторе 1 элемент с максимальным значением и заменить значения всех элементов сектора 2 на найденное максимальное число. Вывести матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: -99…99	

29	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Найти в секторе 1 элемент с минимальным значением и заменить значения всех элементов сектора 2 на найденное максимальное число. Вывести матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: -88…88	
30	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Развернуть матрицу на 180° по часовой стрелке. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: 0…99	
31	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Для каждого элемента главной диагонали вычислить сумму так, как показано на иллюстрации. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных чисел: 0…50	
32	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Для каждого элемента побочной диагонали рассчитать сумму элементов, как показано на иллюстрации. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных чисел: 0…40	

33	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Найти в секторе 1 элемент с максимальным значением и заменить значения всех элементов сектора 2 на найденное максимальное число. Вывести матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных значений: -99…99	
34	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел. Заполнить матрицу линейной последовательностью чисел (1, 2, 3, …) так, как показано на иллюстрации. Вывести на экран матрицу, пустые элементы матрицы заполнить символом #. Размер матрицы: 9 x 9 Диапазон случайных чисел: 0…99	
35	Написать программу, в которой с помощью средств динамического распределения памяти создается квадратная матрица целых чисел и заполняется случайными числами. Для каждого элемента побочной диагонали вычислить сумму так, как показано на иллюстрации. Вывести на экран матрицу до и после преобразования. Размер матрицы: 9 x 9 Диапазон случайных чисел: 40…60	

Лабораторная работа №5
Сортировка массивов

Сортировка – процесс обработки данных, выполняющий распределение элементов некоторого множества по группам в соответствие с некоторым определенным критерием.

Основными характеристиками алгоритмов сортировки являются:

1. ***Вычислительная сложность (асимптотическая сложность)*** – параметр, характеризующий быстродействие алгоритма, как зависимость объема работы от объема входных данных.
2. ***Память***. Существуют алгоритмы, для выполнения которых требуется дополнительная память.
3. ***Устойчивость***. Сортировка называется устойчивой, если при выполнении сортировки порядок элементов, имеющих одинаковые ключи, не меняется.
4. ***Естественность поведения*** – улучшение работы алгоритма, если при обработке учитывается, что данные отсортированы или частично отсортированы.

Помимо основных свойств, алгоритмы делятся на два подкласса:

- **Внутренняя сортировка**. Сортируемые данные хранятся в оперативной памяти с произвольным доступом к ячейкам
- **Внешняя сортировка**. Сортируемые данные хранятся на внешнем носителе. Доступ к носителю осуществляется последовательным образом. В каждый момент времени можно считать или записать только элемент, следующий за текущим. В этом случае обычно используются специальные методы сортировки, использующие дополнительную память.

Сортировка с помощью прямого обмена (метод пузырька)

Сортировка методом пузырька – один из самых простых методом сортировки. Название метода выбрано не случайно и основывается на утверждении: в процессе работы алгоритма более «легкие» элементы множества постепенно «всплывают» вверх.

Обобщенный алгоритм сортировки методом пузырька:

1. Начать просмотр элементов массива *a[n]* от начала к концу. *n* – размер массива.
2. На *i*-ом шаге выполнить вложенный просмотр элементов массива *a[n]*, от конца до *i*-го элемента.
3. На *j*-ом шаге вложенного перебора сравнить соседние элементы *a[j] и a[j - 1]* между собой по некоторому условию *P*.
4. Если условие *P* – не выполнено, то перейти в п.5. Иначе поменять местами элементы *a[j]* и *a[j - 1]*, перейти в п.5.
5. Если *j == i*, то перейти в п.6. Иначе *j = j - 1*, перейти в п.3.
6. Если на любой *j-ой* итерации выполнено условие *P*, то *i = i + 1*, перейти в п.2. Иначе перейти в п.7.
7. Закончить выполнение алгоритма.

На рисунке 5.1 представлен процесс работы алгоритма. i и j – итерационные переменные внешнего и вложенного циклов, соответственно.

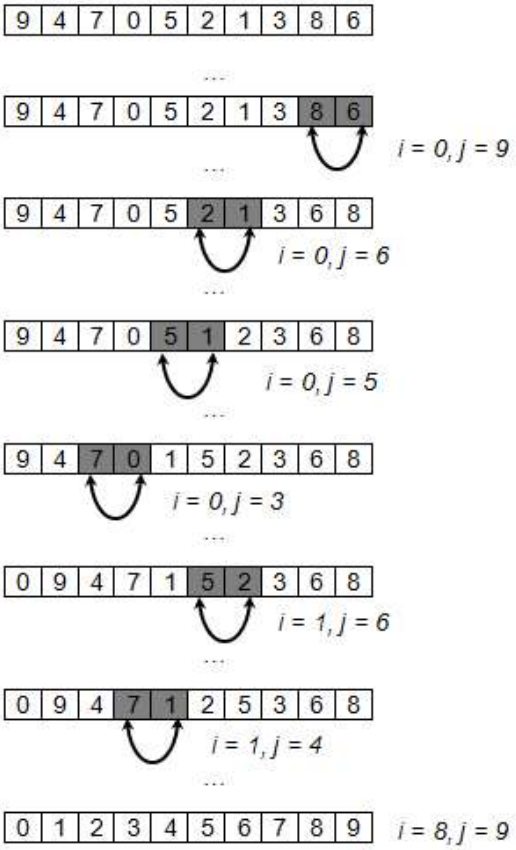


Рис 5.1. Метод пузырька

Сложности алгоритма сортировки методом пузырька составляет $O(N^2)$. В худшем случае, количество операций сравнения: $\frac{n^2 - n}{2}$, в лучшем: n - 1. Для работы алгоритма не требуется дополнительной памяти. Алгоритм устойчив. Поведение алгоритма естественно, так как, если исходная последовательность упорядочена или частично упорядочена, то число сравнений уменьшится.

Метод выбора (простого выбора)

Идея метода заключается в том, что необходимо последовательно перебирать элементы массива *a[n], n* – размер массива. На каждой *i*-ой итерации найти элемент с минимальным значением из интервала *a[i+1]…a[n]* и поменять местами с *i*-ым элементом.

Обобщенный алгоритм сортировки методом выбора

1. Начать просмотр элементов массива *a[n]* от начала к концу, *n* – размер массива.
2. На i-ой итерации найти элемент с минимальным значением из интервала *a[i+1]…a[n]*.
3. Если минимальный элемент найден, поменять местами с *i*-ым элементом.
4. *i = i + 1*. Если *i == n-1*, то перейти в п.5, иначе перейти в п.2.
5. Закончить выполнение алгоритма.

На рисунке 5.2 представлен процесс работы алгоритма методом выбора.

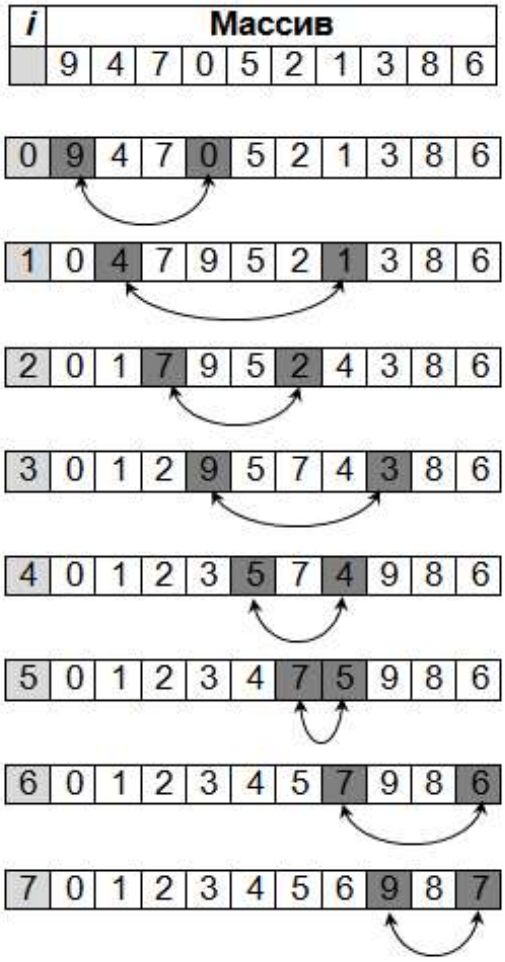


Рис. 5.2. Метод выбора

Сложность алгоритма сортировки методом выбора составляет $O(N^2)$. Количество операций сравнения: $\frac{n^2 - n}{2}$. Для работы алгоритма не требуется дополнительной памяти. Алгоритм неустойчив, так уже после 1-ой итерации порядок ключей будет изменен. Алгоритм ведет себя неестественно, так как если входная последовательность упорядочена, то число сравнений не изменится.

Сортировка простыми вставками

Алгоритм данного метода основан на предположении, что исходная последовательность *a[n]* уже разбита на две части:

- упорядоченную: *a[0]…a[i];*
- неупорядоченную: *a[i + 1]…a[n].*

В сортировках методом пузырька или методом выбора можно заметить, что на *i*-ой итерации элементы *a[0]…a[i]* упорядочены по некоторому заданному правилу и больше не подлежат перестановке. Для метода вставок все иначе, по ходу алгоритма в последовательность *a[0]…a[i]* будут вставляться новые элементы.

Поиск подходящего места для очередного элемента *a[i]* осуществляется путем последовательных сравнений с элементами *a[0]…a[i - 1]*, стоящих перед ним.

Таким образом, в процессе работы алгоритма происходит перемещение элемента *p* неупорядоченной последовательности к началу массива, до тех пор, пока не будет найден элемент меньше *p*, или не будет достигнуто начало последовательности.

Обобщенный алгоритм сортировки методом простых вставок:

1. Условно разбить последовательность на две части
   - *a[0] – упорядоченная;*
   - *a[1]…a[n] – неупорядоченная.*
2. Начать просмотр неупорядоченной последовательности.
3. На *i*-ой итерации сравнить элемент *a[i]* с элементами *a[j]…a[0] (j = i - 1)*, стоящими перед ним.
4. Если *a[j]* < *a[i]*, перейти в п.6. Иначе, поменять местами элементы *a[i]* и *a[j]*.
5. *j = i - 1*. Если *j >= 0*, то перейти в п.6, иначе перейти в п.4.
6. *i = i + 1*. Если конец массива достигнут, то перейти в п.7.
7. Закончить выполнение алгоритма.

На рисунке 5.3 представлен процесс перемещения элемента неупорядоченной последовательности к началу массива.

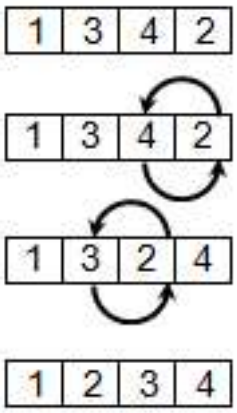


Рис.5.3. Сортировка вставками

Сложность алгоритма сортировки вставками оценивается, как $O(n^2)$. Количество операций сравнения в лучшем случае: n - 1, в худшем: $\frac{n^2 - n}{2}$. Метод сортировки вставками является устойчивым и обладает естественным поведением.

Быстрая сортировка (Quick Sort)

Алгоритм быстрой сортировки был разработан и предложен Ч. Хоаром в 1960г. Алгоритм быстрой сортировки так же называют сортировкой с разделением. Алгоритм организован по принципу «разделяй-и-властвуй». В стандартной библиотеке языка С содержится функция qsort(), которая реализует алгоритм быстрой сортировки.

```
void qsort(void *buf,
          size_t num,
          size_t size,
          int (*compare) (const void *, const void *)
);
```

`buf` – параметр-указатель на сортируемый массив;

`num` – количество элементов;

`size` – размер одного элемента в байтах;

Для передачи функции сравнения используется параметр compare. Функция имеет следующий прототип:

```
int func_compare (const void *arg1, const void *arg2);
```

Функция возвращает следующие значения:

Сравнение	Возвращаемое значение
`arg1 < arg2`	Меньше нуля
`arg1 = arg2`	Нуль
`arg1 > arg2`	Больше нуля

Основная идея алгоритма:

1. В исходном массиве выбирается опорный элемент *a[i]*.
2. Элементы массива сравниваются с опорным элементом. Элементы, меньшие опорного элемента – перемещаются влево, а элементы, больше или равные опорному элементу, перемещаются вправо.
3. Для обоих полученных массивов (меньших и больших), если их длина больше единицы, рекурсивно повторяется процедура разделения.

Обобщенный алгоритм быстрой сортировки:

1. В массиве *a[n]*, где *n* – размер массива, выбрать опорный элемент $p = n / 2$.
2. Начать просмотр массива с начала ($i = 0$) и конца ($j = n-1$).
3. Если $a[i] < p$, то $i = i + 1$, повторить п.3. Иначе перейти в п.4.
4. Если $a[j] > p$, то $j = j – 1$, повторить п.4. Иначе перейти в п.5.
5. Если $i \leq j$, то поменять местами элементы *a[i]* и *a[j]*.
6. $i = i + 1$, $j = j - 1$.
7. Если $i \leq j$, перейти в п.3, иначе перейти в п.8.
8. Если $j > 0$, повторить все с п.1 для *a[0]..a[j]*, иначе перейти в п.9.
9. Если $n-1 > i$, повторить все с п.1 для *a[i]..a[n-1]*, иначе перейти в п.10.
10. Закончить выполнение алгоритма.

На рисунке 5.4. представлен процесс работы метода быстрой сортировки.

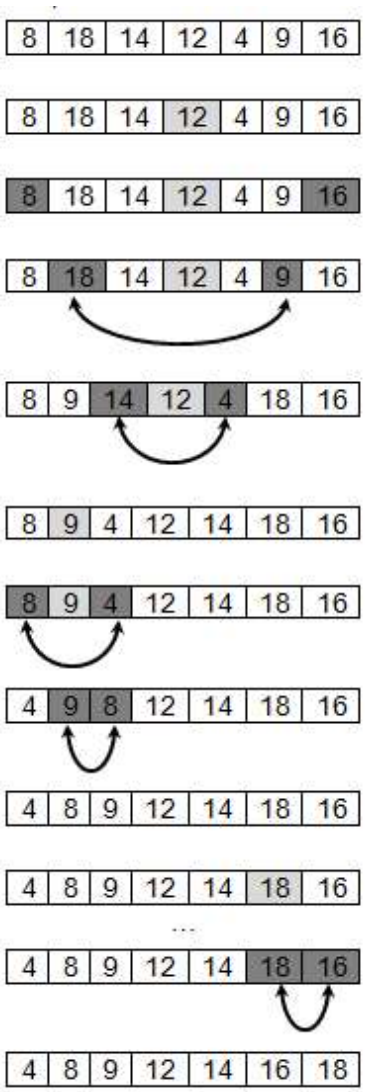


Рис.5.4. Быстрая сортировка

Сложность алгоритма быстрой сортировки оценивается, как *O(n log n)*. Метод неустойчив, но имеет естественное поведение. Используется дополнительная память, т.к. глубина рекурсии составляет *O(log n)*.

Сортировка слияниями

В основе метода сортировки слияниями лежит принцип: «разделяй и властвуй». Метод был предложен Джоном фон Нейманом. В процессе выполнения алгоритма массив разбивается на равные части, каждая из которых сортируется отдельно по такому же принципу, после чего обе части комбинируются воедино.

Обобщенный алгоритм сортировки слияниями:

1. Исходный массив делится пополам, после чего каждая из половин рекурсивно делится пополам до тех пор, пока размер очередной половины не станет равным единице.
2. Два единичных массива сливаются в общий, при этом из каждого выбирается меньший элемент (сортировка по возрастанию) и записывается в свободную левую ячейку результирующего массива. Затем из двух результирующих массивов

собирается третий результирующий массив и т.д. Если один массив закончился, элементы другого записываются в результирующий массив.

3. В конце операции слияния, элементы результирующего массива записываются в исходный массив.

Сложность алгоритма сортировки слияниями составляет *O(n log n).* Алгоритм является устойчивым.

Задание

*Выберите вариант по формуле (n + (n * 5)) mod 10 + 1, где n – номер варианта в предыдущей лабораторной работе.*

№	Задание
1	Сравните быстродействие двух методов сортировки: **метод пузырька и метод простого выбора**. Для этого необходимо написать программу, в которой реализуются алгоритмы указанных методов. Использовать массив случайных чисел, создаваемый с помощью средств динамического распределения памяти. Проверьте правильность работы алгоритмов, применив их для массива размера - 20. Для измерения времени выполнения используйте функцию clock() библиотеки time.h. Рекомендуется использовать массивы больших размеров, например 10000.
2	Сравните быстродействие двух методов сортировки: **метод пузырька и метод простых вставок**. Для этого необходимо написать программу, в которой реализуются алгоритмы указанных методов. Использовать массив случайных чисел, создаваемый с помощью средств динамического распределения памяти. Проверьте правильность работы алгоритмов, применив их для массива размера - 20. Для измерения времени выполнения используйте функцию clock() библиотеки time.h. Рекомендуется использовать массивы больших размеров, например 10000.
3	Сравните быстродействие двух методов сортировки: **метод пузырька и быстрая сортировка**. Для этого необходимо написать программу, в которой реализуются алгоритмы указанных методов. Использовать массив случайных чисел, создаваемый с помощью средств динамического распределения памяти. Проверьте правильность работы алгоритмов, применив их для массива размера - 20. Для измерения времени выполнения используйте функцию clock() библиотеки time.h. Рекомендуется использовать массивы больших размеров, например 10000.
4	Сравните быстродействие двух методов сортировки: **метод пузырька и сортировка слияниями**. Для этого необходимо написать программу, в которой реализуются алгоритмы указанных методов. Использовать массив случайных чисел, создаваемый с помощью средств дина-

	мического распределения памяти. Проверьте правильность работы алгоритмов, применив их для массива размера - 20. Для измерения времени выполнения используйте функцию clock() библиотеки time.h. Рекомендуется использовать массивы больших размеров, например 10000.
5	Сравните быстродействие двух методов сортировки: **метод простого выбора и метод простых вставок**. Для этого необходимо написать программу, в которой реализуются алгоритмы указанных методов. Использовать массив случайных чисел, создаваемый с помощью средств динамического распределения памяти. Проверьте правильность работы алгоритмов, применив их для массива размера - 20. Для измерения времени выполнения используйте функцию clock() библиотеки time.h. Рекомендуется использовать массивы больших размеров, например 10000.
6	Сравните быстродействие двух методов сортировки: **метод простого выбора и быстрая сортировка**. Для этого необходимо написать программу, в которой реализуются алгоритмы указанных методов. Использовать массив случайных чисел, создаваемый с помощью средств динамического распределения памяти. Проверьте правильность работы алгоритмов, применив их для массива размера - 20. Для измерения времени выполнения используйте функцию clock() библиотеки time.h. Рекомендуется использовать массивы больших размеров, например 10000.
7	Сравните быстродействие двух методов сортировки: **метод простого выбора и сортировка слияниями**. Для этого необходимо написать программу, в которой реализуются алгоритмы указанных методов. Использовать массив случайных чисел, создаваемый с помощью средств динамического распределения памяти. Проверьте правильность работы алгоритмов, применив их для массива размера - 20. Для измерения времени выполнения используйте функцию clock() библиотеки time.h. Рекомендуется использовать массивы больших размеров, например 10000.
8	Сравните быстродействие двух методов сортировки: **метод простых вставок и быстрая сортировка**. Для этого необходимо написать программу, в которой реализуются алгоритмы указанных методов. Использовать массив случайных чисел, создаваемый с помощью средств динамического распределения памяти. Проверьте правильность работы алгоритмов, применив их для массива размера - 20. Для измерения времени выполнения используйте функцию clock() библиотеки time.h. Рекомендуется использовать массивы больших размеров, например 10000.

9	Сравните быстродействие двух методов сортировки: **метод простых вставок и сортировка слияниями**. Для этого необходимо написать программу, в которой реализуются алгоритмы указанных методов. Использовать массив случайных чисел, создаваемый с помощью средств динамического распределения памяти. Проверьте правильность работы алгоритмов, применив их для массива размера - 20. Для измерения времени выполнения используйте функцию clock() библиотеки time.h. Рекомендуется использовать массивы больших размеров, например 10000.
10	Сравните быстродействие двух методов сортировки: **быстрая сортировка и сортировка слияниями**. Для этого необходимо написать программу, в которой реализуются алгоритмы указанных методов. Использовать массив случайных чисел, создаваемый с помощью средств динамического распределения памяти. Проверьте правильность работы алгоритмов, применив их для массива размера - 20. Для измерения времени выполнения используйте функцию clock() библиотеки time.h. Рекомендуется использовать массивы больших размеров, например 10000.

Контрольные вопросы:

1. Правда ли, что алгоритм пузырьковой сортировки основан на делении массива данных пополам?
   a) Да
   b) Нет

2. Дан массив целых чисел arr[10] = {8, 5, 1, 0, 9, 4, 2, 3, 7, 6}. Массив сортируется по возрастанию методом простого выбора, начиная с начала. Как будет выглядеть массив после первой итерации цикла?
   a) 8, 5, 1, 0, 9, 4, 2, 3, 6, 7;
   b) 0, 1, 5, 8, 9, 4, 2, 3, 7, 6;
   c) 0, 5, 1, 8, 9, 4, 2, 3, 7, 6;
   d) 0, 5, 1, 8, 9, 4, 2, 3, 6, 7;
   e) нет правильного ответа.

3. Сортировка n-мерного массива методом пузырька обладает особенностью, что каждый элемент массива, расположенный не на своем месте:
   a) достигает своего места за один проход;
   b) достигает своего места за 2 прохода;
   c) достигает своего места за 3 прохода;
   d) достигает своего места за n/2 проходов;
   e) достигает своего места за n проходов;
   f) достигает своего места за n^2 проходов.
   g) нет правильного ответа.

4. Идея сортировки массива a[n] методом простых вставок заключается в том, что:
   a) элементы перебираются последовательно, и на каждой *i-ой* итерации ищется элемент с минимальным значением из интервала *a[i+1]…a[n]*;

b) поиск подходящего места для очередного элемента *a[i]* осуществляется путем последовательных сравнений с элементами *a[0]…a[i - 1]*, стоящих перед ним;
c) в процессе работы алгоритма более «легкие» элементы множества постепенно «всплывают» вверх;
d) Происходит деление массива пополам, затем полученные половины снова делятся пополам и т. д;
e) массив разбивается на равные части, каждая из которых сортируется отдельно по такому же принципу;
f) нет правильного ответа.

5. Идея сортировки массива a[n] методом пузырька заключаются в том, что:
   a) элементы перебираются последовательно, и на каждой *i-ой* итерации ищется элемент с минимальным значением из интервала *a[i+1]…a[n]*;
   b) в процессе работы алгоритма более «легкие» элементы множества постепенно «всплывают» вверх;
   c) поиск подходящего места для очередного элемента *a[i]* осуществляется путем последовательных сравнений с элементами *a[0]…a[i - 1]*, стоящих перед ним;
   d) Происходит деление массива пополам, затем полученные половины снова делятся пополам и т. д;
   e) массив разбивается на равные части, каждая из которых сортируется отдельно по такому же принципу;
   f) нет правильного ответа.

6. Идея сортировки массива a[n] слияниями заключается в том, что:
   a) элементы перебираются последовательно, и на каждой *i-ой* итерации ищется элемент с минимальным значением из интервала *a[i+1]…a[n]*;
   b) в процессе работы алгоритма более «легкие» элементы множества постепенно «всплывают» вверх;
   c) поиск подходящего места для очередного элемента *a[i]* осуществляется путем последовательных сравнений с элементами *a[0]…a[i - 1]*, стоящих перед ним;
   d) Происходит деление массива пополам, затем полученные половины снова делятся пополам и т. д;
   e) массив разбивается на равные части, каждая из которых сортируется отдельно по такому же принципу;
   f) нет правильного ответа.

7. Сложность алгоритма сортировки массива методом пузырька пропорциональна
   a) N;
   b) $\sqrt{N}$;
   c) Log N;
   d) N2;
   e) N3;
   f) нет правильного ответа.

8. Если сортировку методом простого выбора применить для последовательности «2413», то будут получены следующие результаты:
   a) первый проход: 1 4 2 3; второй проход: 1 2 4 3; третий проход: 4 2 3 1;
   b) первый проход: 1 4 2 3; второй проход: 1 2 4 3; третий проход: 1 2 3 4;

c) первый проход: 1 4 2 3; второй проход: 1 3 4 2; третий проход: 1 2 3 4;
d) первый проход: 3 4 2 1; второй проход: 1 2 4 3; третий проход: 1 2 3 4;
e) нет правильного ответа.

Лабораторная работа №6
Структуры и объединения

Структуры

Структура – это совокупный тип данных, состоящий из множества именованных элементов данных разного типа, объединенных в единой целое.

Член данных (элемент, поле) – переменная базового или пользовательского типа, входящая в состав структурного типа.

Имя структуры - это спецификатор типа структуры.

Обобщенный формат определения структурного типа:

```
struct имя_структурного_типа{
     тип имя_переменной_1;
     тип имя_переменной_2;
     тип имя_переменной_3;
     …
     тип имя_переменной_n;
};
```

Существует несколько способов объявления структуры:

1. Использовать конструкцию

   ```
   struct имя_структуры структурные_переменные;
   ```

2. Структура может быть объявлена при определении структурного типа

   ```
   struct имя_структурного_типа{
        тип имя_переменной_1;
        тип имя_переменной_2;
        тип имя_переменной_3;
        …
        тип имя_переменной_n;
   } структурные_переменные;
   ```

3. Можно не указывать имя структуры, а только определить внутреннее строение структуры:

   ```
   struct{
        тип имя_переменной_1;
        тип имя_переменной_2;
        тип имя_переменной_3;
        …
        тип имя_переменной_n;
   } структурные_переменные;
   ```

4. Воспользоваться спецификатором **typedef**, с помощью которого можно ввести синоним для любого базового или пользовательского типа. При таком подходе можно не указывать имя структуры после ключевого слова **struct**, в противном случае, в дальнейшем можно использовать и имя структуры, и синоним для объявления структурных переменных.

   ```
   typedef struct имя_структуры{
        тип имя_переменной_1;
        тип имя_переменной_2;
        тип имя_переменной_3;
        …
   ```

```
        тип имя_переменной_n;
    } синоним_имя_структуры;
```

Разрешается создавать массивы структур, например

```
struct имя_структуры имя_массив[размер];
```

Для доступа к полям структуры используется оператор «точка». Обобщенный формат доступа к полям структуры:

```
структурная_переменная.имя_переменной;
```

Инициализировать структуру можно двумя способами:

- Инициализировать каждое поле по отдельности
```
структурная_переменная.имя_переменной_1 = значение;
структурная_переменная.имя_переменной_2 = значение;
…
структурная_переменная.имя_переменной_n = значение;
```
- Указать список начальных значений
```
struct имя_структуры структурная переменная = {
     значение переменной_1,
     …
     значение переменной_n
}
```

Размер памяти в байтах, выделяемый для структуры можно измерить с помощью операции **sizeof**(имя_структуры).

Содержимое одной структуры можно присвоить другой, если обе структуры имеют одинаковый тип:

```
struct имя_структуры переменная_1, переменная_2;
…
//инициализация переменной_1
…
переменная_2 = переменная_1;
```

Указатели на структуры можно использовать таким же способом, как и указатели на переменные любого другого типа, например:

```
struct CarModel{
     int id_car;
     char model;
} cm;


struct CarModel *cmp;
//в указатель cmp будет помещен адрес структурной переменной cm.
cmp = &cm;
```

Для того, чтобы получить доступ к членам структуры с помощью указателя, используется оператор стрелка “->”.

Пример:

```
cmp -> id_car;
```

Объединения

Объединение – это переменная, которая может содержать объекты различных типов и размеров. Объединение является неким подобием структуры, но с нулевым смещением от ее начала. Это означает, что при таком размещении разные объекты занимают один и тот же участок памяти. Для объявления объединения предназначен спецификатор union.

Обобщенный формат объявления объединения:

```
union имя_объединения{
      тип имя_переменной_1;
      тип имя_переменной_2;
      …
      тип имя_переменной_n;
} список_переменных;
```

Пример:

```
union MyUnion{
      char var1[2];
      int var2;
};
```

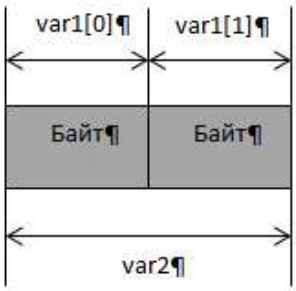


Рис.6.1. Схема размещения объединения в памяти

Объединяющий тип можно вводить с помощью ключевого слова typedef, подобно структурному типу.

Размер объединения определяется размером самого большого элемента, принадлежащего ему.

Доступ к элементам объединения осуществляется таким же образом, как и к структурам.

Битовые поля

Битовое поле – это элемент структуры или объединения, обеспечивающие удобный доступ к отдельным битам данных. С помощью битовых полей можно формировать объекты с длиной внутреннего представления, не кратной байту. Это позволяет экономить память, плотно упаковывая информацию.

Обобщенный формат объявления битовых полей:

```
struct{
     тип_1 имя_поля_1: ширина поля_1;
     тип_2 имя_поля_2: ширина_поля_2;
     …
     тип_n имя_поля_n: ширина_поля_n;
} имя структуры;
```

, где `тип_n` – тип **`int`** (со спецификатором **`unsigned`** или **`signed`**), `ширина_поля_n` – целое неотрицательное число, значение которого обычно не должно превышать длины слова конкретной ЭВМ.

В зависимости от компилятора, поля могут размещаться в памяти как слева направо, так и справа налево.

Пример:

```
struct{
     int x:5;
     int y:3;
} ms;
ms.x = 4;
ms.y = 2;
```

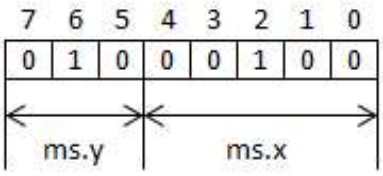


Рис 6.2. Размещение битовых полей в байте

Задание

<table>
<tr><th>№</th><th>Задание</th></tr>
<tr><td>1</td><td>1. Написать программу, в которой будут вводиться 15 строк таблицы, приведенной ниже.
2. Сортировать таблицу по значениям в первом столбце.
3. Вывести на экран таблицу (включая границы).

<table>
<tr><td colspan="4">Ведомость</td></tr>
<tr><td>Номер страховки</td><td>Фамилия</td><td>Количество отработанных часов</td><td>Почасовая оплата</td></tr>
</table>
</td></tr>
<tr><td>2</td><td>1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу по значениям во втором столбце.
3. Вывести на экран таблицу (включая границы).

<table>
<tr><td colspan="3">Глубочайшие впадины суши</td></tr>
<tr><td>Название</td><td>Глубина от уровня моря, м</td><td>Местоположение</td></tr>
</table>
</td></tr>
<tr><td>3</td><td>1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу по значениям во втором столбце.
3. Вывести на экран таблицу (включая границы).

<table>
<tr><td colspan="4">Крупнейшие реки</td></tr>
<tr><td>Название</td><td>Длина, км</td><td>Площадь бассейна,</td><td>Средний расход воды, куб. м/с</td></tr>
</table>
</td></tr>
<tr><td>4</td><td>1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу по значениям в третьем столбце.
3. Вывести на экран таблицу (включая границы).

<table>
<tr><td colspan="3">Денежные единицы стран мира</td></tr>
<tr><td>Страна</td><td>Денежная единица</td><td>Количество на 1 доллар США</td></tr>
</table>
</td></tr>
<tr><td>5</td><td>1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу по значениям в четвертом столбце.
3. Вывести на экран таблицу (включая границы).

<table>
<tr><td colspan="4">Рыбы</td></tr>
<tr><td>Название</td><td>Максимальная длина</td><td>Максимальный вес</td><td>Вес</td></tr>
</table>
</td></tr>
</table>

6

1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу по значениям во втором столбце.
3. Вывести на экран таблицу (включая границы).

Крупнейшие озера			
Название	Площадь водной поверхности (кв. км)	Высота уровня, м	Местоположение

7

1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу сначала по значениям в первом столбце, а затем по значениям в третьем столбце.
3. Вывести на экран таблицу (включая границы).

Картинная галерея			
Автор	Название картины	Год написания	Сумма страховки, руб

8

1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу по значениям в четвертом столбце.
3. Вывести на экран таблицу (включая границы).

Японские острова			
Остров	Площадь (кв. км)	Количество малых островов	Протяженность береговой линии (км).

9

1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу по значениям в четвертом столбце.
3. Вывести на экран таблицу (включая границы).

Офисные пакеты			
Наименование	Производитель	Количество компонентов	Цена, руб

10

1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу сначала по значениям в первом столбце, а затем по значениям в третьем столбце.
3. Вывести на экран таблицу (включая границы).

Книги					
Автор	Название	Год издания	Издательство	Количество страниц	ISBN

11

1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу сначала по значениям во втором столбце, а затем по значениям в третьем столбце.
3. Вывести на экран таблицу (включая границы).

Отдел кадров					
Должность	Фамилия	Инициалы	Год рождения	Стаж работы, мес.	Оклад

12

1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу сначала по значениям в первом столбце, а затем по значениям в седьмом столбце.
3. Вывести на экран таблицу (включая границы).

Автомобили						
Марка	Модель	Тип кузова	Объем двигателя, л	Трансмиссия	Цвет	Цена

13

1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу по значениям во втором столбце.
3. Вывести на экран таблицу (включая границы).

Глубочайшие впадины суши		
Название	Глубина от уровня моря, м	Местоположение

14

1. Написать программу, в которой будут вводиться 15 строк таблицы.
2. Сортировать таблицу сначала по значениям в первом столбце, а затем по значениям в третьем столбце.
3. Вывести на экран таблицу (включая границы).

USB-накопители					
Производитель	Модель	Объем памяти, ГБ	Интерфейс	Размеры	Цвет

15

1. Написать программу, в которой будут вводиться значения таблицы для 10 наименований компьютерных мышей.
2. Сортировать таблицу сначала по значениям в первой строке.
3. Вывести на экран таблицы (включая границы).

Производитель, модель компьютерной мыши	
Тип	
Количество клавиш	
Разрешение оптического сенсора	
Длина провода	
Интерфейс подключения	
Цвет	
Размеры	
Вес	

16

1. Написать программу, в которой будут вводиться значения таблицы для 10 наименований процессоров.
2. Сортировать таблицу сначала по значениям в первой строке, а затем по значениям во второй строке.
3. Вывести на экран таблицы (включая границы).

Производитель, модель процессора	
Socket	
Ядро	
Количество ядер	
Тактовая частота	
Напряжение на ядре	
Объем кэша	
Максимальная рабочая температура	
Вес	

17

1. Написать программу, в которой будут вводиться значения таблицы для 10 наименований мониторов.
2. Сортировать таблицу сначала по значениям в первой строке, а затем по значениям в третьей строке.
3. Вывести на экран таблицы (включая границы).

Производитель, модель монитора	
Тип	
Диагональ	
Разрешение	
Тип ЖК-матрицы	
Подсветка	
Размеры	
Вес	

<table>
<tr><td>18</td><td>
1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы
2. Сортировать таблицу сначала по значениям в первом столбце, а затем по значениям в четвертом столбце.
3. Вывести на экран таблицы (включая границы).

<table>
<tr><th colspan="7">Горы</th></tr>
<tr><td>Название</td><td>Страна</td><td>Период образования</td><td>Площадь, тыс.км^2</td><td>Длина</td><td>Ширина</td><td>Высшая точка, м</td></tr>
</table>
</td></tr>
<tr><td>19</td><td>
1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу сначала по значениям в первом столбце, а затем по значениям в пятом столбце.
3. Вывести на экран таблицы (включая границы).

<table>
<tr><th colspan="5">Федеральные трассы</th></tr>
<tr><td>Номер</td><td>Страна</td><td>Начало</td><td>Конец</td><td>Протяженность</td></tr>
</table>
</td></tr>
<tr><td>20</td><td>
1. Написать программу, в которой будут вводиться значения таблицы для 10 строк таблицы.
2. Сортировать таблицу сначала по значениям в первом столбце, а затем по значениям в шестом столбце.
3. Вывести на экран таблицы (включая границы).

<table>
<tr><th colspan="6">Города России</th></tr>
<tr><td>Название города</td><td>Субъект Федерации</td><td>Страна</td><td>Дата основания</td><td>Площадь, км^2</td><td>Население</td></tr>
</table>
</td></tr>
</table>

21

1. Написать программу, в которой будут вводиться значения таблицы для 10 наименований видеокарт.
2. Сортировать таблицу сначала по значениям в первой строке, а затем по значениям в пятой строке.
3. Вывести на экран таблицы (включая границы).

Производитель, модель видеокарты	
Тип	
Графический процессор	
Частота графического процессора	
Объем видеопамяти	
Частота видеопамяти	
Разрядность шины видеопамяти	
Разъемы	
Охлаждение	
Количество поддерживаемых мониторов	

22

1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям в третьем столбце.
3. Вывести на экран таблицы (включая границы).

Ведомость общественного транспорта			
Вид транспорта	№ маршрута	Протяженность маршрута, км	Время в дороге

23

1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям в первом столбце.
3. Вывести на экран таблицы (включая границы).

Прайс-лист			
Наименование товара	Тип товара	Цена за 1шт	Минимальное количество в партии

24

1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям в третьем столбце, а затем по значениям во втором столбце.
3. Вывести на экран таблицы (включая границы).

Скорость звука в жидкостях		
Вещество	Температура (°C)	Скорость (м/с)

25

1. Написать программу, в которой будут вводиться значения таблицы для 10 наименований велосипедов.
2. Сортировать таблицу по значениям в первой строке.
3. Вывести на экран таблицы (включая границы).

Производитель, модель велосипеда	
Тип	
Материал рамы	
Размер рамы	
Количество скоростей	
Тип переднего тормоза	
Тип заднего тормоза	
Вес	
Максимальный вес пользователя	
Страна производитель	

26

1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям во втором столбце.
3. Вывести на экран таблицы (включая границы).

Лучшие спортивные достижения по легкой атлетике			
Название вида	ФИО рекордсмена	Дата установления рекорда	Результат

27

1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям в первом столбце, а затем по значениям в четвертом столбце.
3. Вывести на экран таблицы (включая границы).

Информация о стационарных больных					
ФИО	Возраст	Адрес	Дата поступления	Диагноз	ФИО лечащего врача

28

1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям в первом столбце.
3. Вывести на экран таблицы (включая границы).

Химические элементы					
Атомный номер	Название	Символическое обозначение	Масса атома	Заряд атомного ядра	Основные химические свойства

29

1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям в первом столбце.
3. Вывести на экран таблицы (включая границы).

Предприятия				
Название	ФИО директора	Количество сотрудников	Адрес	Уставной капитал

30

1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям в первом столбце.
3. Вывести на экран таблицы (включая границы).

Абоненты телефонной связи			
ФИО абонента	Номер телефона	Тип соединения	Адрес владельца
Тип соединения: индивидуальный, спаренный, коллективного пользования			

31

1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям во втором столбце, а затем по значениям в первом столбце
3. Вывести на экран таблицы (включая границы).

Информация о жильцах общежития			
ФИО	Номер комнаты	Факультет	Площадь

32

1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям в первом столбце.
3. Вывести на экран таблицы (включая границы).

Информация о вузах			
Название вуза	Число студентов	Количество факультетов	Количество кафедр

<table>
<tr><td>33</td><td>
1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям в первом столбце.
3. Вывести на экран таблицы (включая границы).
<table>
<tr><td colspan="4">Посетители библиотеки</td></tr>
<tr><td>Номер билета</td><td>ФИО</td><td>Количество книг на руках</td><td>Дата перерегистрации</td></tr>
</table>
</td></tr>
<tr><td>34</td><td>
1. Написать программу, в которой будут вводиться значения таблицы для 15 строк таблицы.
2. Сортировать таблицу по значениям во втором столбце.
3. Вывести на экран таблицы (включая границы).
<table>
<tr><td colspan="4">Сплавы с высоким сопротивлением</td></tr>
<tr><td>Сплав</td><td>Сопротивление</td><td>Темп. коэфф. сопр</td><td>Максимальная температура</td></tr>
</table>
</td></tr>
<tr><td>35</td><td>
1. Написать программу, в которой будут вводиться значения таблицы.
2. Сортировать таблицу по значениям во втором столбце.
3. Вывести на экран таблицы (включая границы).
<table>
<tr><td colspan="4">Элементарные частицы</td></tr>
<tr><td>Частица</td><td>Группа</td><td>Заряд</td><td>Масса покоя</td></tr>
</table>
</td></tr>
</table>

Контрольные вопросы

1. Выберите верное определение структуры в языке С.
   a) Структура - это совокупный тип данных, состоящий из множества именованных элементов данных разного типа, объединенных в единой целое.
   b) Структура - сложный тип данных, позволяющий размещать в одном и том же месте оперативной памяти объекты, возможно, различного типа.
   c) Структура – это сложный тип данных, состоящий из фиксированного числа упорядоченных элементов, имеющих один и тот же тип.
   d) Нет правильного ответа.

2. Укажите правильное объявление структуры типа `Vehicle`.
   e) `Vehicle v;`
   a) `typedef Vehicle v;`
   b) `union Vehicle v;`
   c) `struct Vehicle v;`
   d) Нет правильного ответа.

3. Как правильно объявить структуру?
   a) Объявление
```
struct тип_структуры{
     тип_1 имя_элемента_1.имя_структурной_переменной;
     тип_2 имя_элемента_2 имя_структурной_переменной;
     …
     тип_n имя_элемента_n имя_структурной_переменной;
};
```
   b) Объявление
```
union тип_структуры{
     тип_1 имя_элемента_1;
     тип_2 имя_элемента_2;
     …
     тип_n имя_элемента_n;
};
```
   c) Объявление
```
struct тип_структуры{
     тип_1 имя_элемента_1;
     тип_2 имя_элемента_2;
     …
     тип_n имя_элемента_n;
};
```
   d) Объявление
```
struct тип_структуры{
     тип_1: имя_элемента_1;
     тип_2: имя_элемента_2;
     …
     тип_n: имя элемента_n;
};
```
   e) Нет правильного ответа.

4. Что храниться в переменной `vhl`?
```
struct Vehicle
{
     …
};
struct Vehicle *vhl;
```
   a) Содержимое структуры `Vehicle`.
   b) Адрес структуры `Vehicle`.
   c) Структурная переменная `vhl` не инициализирована — содержимое не определено.
   d) Будет храниться размер участка оперативной памяти в байтах, необходимого для размещения структуры `Vehicle`.
   e) Нет верного ответа

5. Укажите правильный доступ к переменной passengers структуры v.
   a) `v-> passengers;`
   b) `v[passengers]`

c) `v.passengers;`
d) `v:passengers;`
e) Нет правильного ответа.

6. Как записать число 10 в поле mpg с помощью указателя v?
a) `v.mpg = 10;`
b) `v->mpg = 10;`
c) `*v.mpg = 10;`
d) `v.mpg = *10;`
e) Нет правильного ответа.

7. Необходимо объявить структуру типа Vehicle, состоящую из следующих полей:
1)структурная переменная типа int с именем passengers;

2)структурная переменная типа int с именем fuelcap;

3)структурная переменная типа int с именем mpg;

4)указатель на структурную переменную типа Vehicle с именем ptr.

Выберите правильный вариант объявления:

a) Объявление
```
Vehicle{
   int passengers;
   int fuelcap;
   int mpg;
   struct Vehicle *ptr;
};
```

b) Объявление
```
struct Vehicle
{
   int passengers;
   int  fuelcap;
   int mpg;
   struct Vehicle *ptr;
};
```

c) Объявление
```
Vehicle
{
   int passengers;
   int  fuelcap;
   int mpg;
   struct Vehicle ptr;
};
```

d) Объявление
```
Vehicle
{
   int passengers;
   int  fuelcap;
   int mpg;
   Vehicle ptr;
};
```

e) Объявление

```
struct Vehicle
{
  int passengers;
  int  fuelcap;
  int mpg;
  struct Vehicle ptr;
};
```

f) Объявление

```
Vehicle
{
  int passengers;
  int  fuelcap;
  int mpg;
  Vehicle *ptr;
};
```

g) Нет правильного ответа.

8. Укажите верное определение объединения в языке С.
   a) Объединение - это совокупный тип данных, состоящий из множества именованных элементов данных разного типа, объединенных в единой целое.
   b) Объединение - сложный тип данных, позволяющий размещать в одном и том же месте оперативной памяти объекты различного типа и размера.
   c) Объединение – это сложный тип данных, состоящий из фиксированного числа упорядоченных элементов, имеющих один и тот же тип.
   d) Объединение – это сложный тип данных, обеспечивающий удобный доступ к отдельным битам данных.
   e) Нет правильного ответа.

.

Лабораторная работа №7.
Функции. Рекурсивные функции.

Понятие «функции» в языке программирования С определено неоднозначно. Во-первых, ***функция*** – это производный тип. Во-вторых, ***функция*** – это минимальный исполняемый модуль программы. И в третьих, ***функция*** – это подпрограмма, содержащая одну или несколько инструкций и выполняющая одну или несколько задач. Также необходимо отметить, что в языке Си нельзя определять вложенные функции, т.е. нельзя определить одну функцию внутри другой.

Функции условно делят на два типа. К первому типу относятся функции, написанные программистом. К другому типу относят функции стандартной библиотеки языка С (коллекция встроенных функций), которые можно использовать в своих программах. Хороший стиль программирования предполагает, что каждая функция выполняет только одну задачу.

Обобщенный формат определения функции:

```
тип имя_фунции (спецификация параметров)
{
     тело функции
}
```

`тип` – обозначение типа возвращаемого функцией значения, либо `void` (для функций, не возвращающих значения).

`имя_функции` – произвольно выбираемое программистом имя (идентификатор), не совпадающее со служебными словами и с именами других переменных (и функций) программы. В языке С для функции, с которой начинается выполнение программы, зарезервировано имя `main()`.

`спецификация_параметров` – это список формальных параметров, каждый элемент которого имеет вид:

```
тип имя_параметра.
```

`тело_функции` – это часть определения функции, ограниченная фигурными скобками и размещенная вслед за заголовком функции.

Прототип функции

В языке С все функции должны быть объявлены до их использования. Это условие достигается с помощью использования прототипа функций. Прототипы содержат три вида информации:

- *Тип возвращаемого значения;*
- *Тип ее параметров;*
- *Количество параметров.*

Прототипы позволяют компилятору выполнить следующие три важные операции:

- *Сообщают компилятору, код какого типа необходимо генерировать при вызове функции. Различия в типах параметров и значении, возвращаемом функцией, обеспечивают различную обработку компилятором.*

- *Позволяют обнаружить недопустимые преобразования типов аргументов, используемых при вызове функции, в тип, указанный в объявлении ее параметров, и сообщить о них.*
- *Позволяют компилятору выявить различия между количеством аргументов, используемых при вызове функции, и количеством параметров, заданных в определении функции.*

Не смотря на то, что язык С допускает прототипы, их использование не обязательно.

Рассмотрим пример, демонстрирующий использование прототипа функции.

```
//…
void func1(void);
//…
void main(){
     func1();  //здесь С-компилятор уже знает о существовании функции func1()
     func2();  //ошибка, С-компилятор не знает ничего о функции func2()
}

void func1(void){
     //тело функции
}

void func2(void){
     //тело функции
}
```

В данном примере объявление прототипа функции `func1()` позволяет располагать вызов функции в коде еще до ее описания. Тогда как, при вызове функции `func2()` компилятор сгенерирует ошибку, т.к. отсутствует прототип функции, вызов функции осуществляется еще до ее описания. Обратите внимание, что в качестве параметра функции указывается пустой тип void. Как будет показано далее, такой прием используется для улучшения читабельности кода.

Области видимости функций

Правила действий областей видимости функций определяют, какая часть кода имеет доступ к той или иной переменной. Другими словами, правила действия областей видимости функций определяют время «жизни» переменной.

Существует три вида переменных: ***локальные, глобальные и формальные параметры***.

Локальные переменные:

- *Это переменные, объявленные внутри функции.*

- *Локальные переменные известны только внутри тела функции, которой они принадлежат. Локальная переменная создается при «входе» в функцию и уничтожается при выходе из нее.*
- *Не сохраняют своего значения при выходе из функции.*

Глобальные переменные:

- *Известны на протяжении всего времени выполнения программы.*
- *Сохраняют свои значения во время выполнения всего кода программы.*
- *Если глобальная и локальная переменные имеют одинаковые имена, то преимущество отдается локальным переменным. это означает, что присутствие локальной переменной скрывает глобальную переменную с таким же именем.*

Формальные параметры

- *Это переменные, которые принимают значения аргументов, передаваемых функции при ее вызове.*
- *В теле функции воспринимаются, как локальные переменные*
- *Тип объявляемых параметров должен совпадать с типом аргументов, передаваемых функции.*

Возврат значений из функций

Для возврата значений из функции используется инструкция `return`, которая выполняет две функции:

- *Возвращает управление к инициатору вызова функции (немедленное завершение работы функции).*
- *Ее можно использовать для передачи значения, возвращаемого функцией.*

Обобщенная форма определения инструкции `return`:

```
return возвращаемое_значение;
```

Каждая функция, кроме типа `void`, может возвращать значение указанного типа, при этом осуществляется немедленное завершение работы функции. Это означает, что функцию можно использовать в качестве операнда в выражении.

Функции, которые не возвращают никаких значений, объявляются как `void`.

Если тип функции указан `void`, то функция должна возвращать значение этого типа, т.е. указывается `return` без каких-либо значений.

Помимо обычных значений, функции могут возвращать указатели. Для этого необходимо объявить возвращаемый тип функции, как указатель. Соответственно, возвращаемое значение должно являться указателем.

Пример:

```
#include <stdio.h>
int return_example(){
     int n = 5;
     return n;
     //здесь выполнение функции прекращается
```

```
    n = 6;
    return n;
}

void main(){
    printf("%d\n", return_example());  //5
    printf("%d\n", return_example() * 2);   //10

    return;
}
```

Передача параметров при вызове функции

При вызове функции ей можно передавать параметры. В языке С существует два способа передачи параметров: по значению и через указатель. Рассмотрим два этих способа.

Передача по значению

При передаче параметров по значению в функцию передается только копии аргументов.

Аргумент – это значение, передаваемое функции при вызове.

Формальный параметр – это определяемая функцией переменная, которая принимает передаваемый функции аргумент.

Функции, которые принимают аргументы, называются ***параметризованными функциями***.

Пример:

```
void func1(int x1, int x2){
    printf("%d", x1+x2);
}
//…
void main(){
//…
int n = 5;
    func1(4, n);   //9
//…
}
```

Здесь переменные `x1` и `x2` являются формальными параметрами, а 4 и `n` – аргументы.

Передача по адресу

При передаче параметра по адресу функции передается не копия значения аргумента, а адрес области памяти, в которой хранится аргумент. Для этого достаточно объявить параметр типа указатель.

При выполнении некоторой операции в функции, которая использует указатель, эта операция фактически выполняется над переменной, адресуемой этим указателем. Таким образом, такая функция может изменить значение объекта, адресуемого ее параметром.

Пример:

```
void func2(int *ptr){
        *ptr = 5;
}
//…
void main(){
        int n = 4;
        func2(n);
        printf("%d", n);        //5
        //…
}
```

Передача массивов

Существует два способа передачи массива в функцию:

- *в качестве параметра объявляется массив, тип и размер которого совпадают с массивом, используемым при вызове функции.*
- *в качестве параметра объявляется указатель на i-й элемент массива.*

В первом случае, С-компилятор автоматически преобразует его в указатель. Во втором случае передается указатель на *i*-й элемент массива. Это означает, что все операции внутри функции выполняются над массивом, переданным в качестве аргумента.

Рассмотрим пример, демонстрирующий применение этих способов. В функции `func1()` в качестве параметра объявляется массив, тип и размер которого известны. А в функции `func2()` в качестве параметра объявляется указатель на массив.

```
//…
void func1(int myarray[5]){
     int i;
for(i = 0; i < 5; i++)
     myarray[i] = i;
}


void func2(int *myarray){
     int i;
     for(i = 0; i < 5; i++)
```

```
        myarray[i] = i * 2;
}

void main(){
     int arr[5], i;
     func1(arr);
     for(i = 0; i < 5; i++)
          printf("%d ", arr[i]);   //0, 1, 2, 3, 4

     func2(arr);
     for(i = 0; i < 5; i++)
          printf("%d ", arr[i]);   //0, 2, 4, 6, 8
}
```

Так как в языке С строка является массивом символов, то в качестве параметра необходимо объявить указатель на начало строки типа (char *).

Функции с переменным количеством параметров

В языке С можно определять функции, количество параметров у которых при компиляции не фиксировано. Более того, могут быть неизвестны типы параметров.

Обобщенный формат определения функций с переменным количеством параметров выглядит следующим образом:

```
тип имя_функции(спецификация_явных_параметров, …){
//…
}
```

Функции с переменным количеством параметров обязаны иметь хотя бы один обязательный (явный) параметр, после которого ставится запятая и указывается многоточие. Многоточие указывает компилятору, что дальнейший контроль соответствия количества и типов параметров при обработке вызова функции проводить не нужно.

Проблема в том, что у переменного списка параметров отсутствуют имена. Каждая функция с переменным количеством параметров должна иметь механизм определения их количества и типов. Существует два подхода:

- *Добавление в конец списка реально использованных (необязательных) фактических параметров специального параметра-индикатора с уникальным значением, которое будет сигнализировать об окончании списка. При таком подходе в теле функции параметры последовательно перебираются, и их значения сравниваются с заранее известным концевым признаком.*
- *Передача в функцию сведений о реальном количестве фактических параметров с помощью одного из явно задаваемых параметров.*

В обоих подходах переход от одного фактического параметра к другому осуществляется с помощью указателей, т.е. с использованием адресной арифметики.

В следующем примере функция sum выполняет суммирование фактических параметров. Эта программа демонстрирует подход, когда количество фактических параметров известно заранее и указывается явно.

```
//…
int sum(int n, ...){
     //n - число суммируемых параметров
     int result = 0;
     int *ptr = &n;

     for(; n > 0; n--){
          result += *(++ptr);
     }

     return result;
}

void main(){
     printf("%d\n", sum(5, 1, 2, 3, 4, 5));
}
```

В следующем примере демонстрируется подход, когда в конец списка фактических параметров добавляется параметр-индикатор, сигнализирующий об окончании списка.

```
int sum(int start, ...){
     //start - начало списка переменного количества параметров
     int result = 0;
     int *ptr = &start;

     if(*ptr == 0)
          return 0;
     for(; *ptr; ptr++){
          result += *ptr;
     }

     return result;
}

void main(){
     printf("%d\n", sum(1, 2, 3, 4, 5, 0));
}
```

Возникает вопрос получения функцией переменного количества параметров разного типа. В этом случае, можно передавать признаки типа параметров с помощью дополнительных обязательных параметров.

Макросредства для переменного числа параметров

Рассмотренный выше способ объявления переменного количества параметров ориентирован на конкретный тип машин и привязан к реализации компилятора. Для обеспечения мобильности программ с функциями, имеющими изменяющиеся списки параметров, предлагается включать специальный набор макроопределений, доступных при включении заголовочного файла `stdarg.h`.

Макрокоманды:

```
void va_start(va_list param, последний явный параметр);
type va_arg(va_list param, type);
void va_end(va_list param);
```

Первые фактические параметры, используемые при обращении к макрокомандам, должны быть типа `va_list`, который определен в файле stdarg.h.

Теперь рассмотрим, как это работает.

Объявим переменную типа `va_list`.

```
va_list temp;
```

Переменная `va_list` обладает свойствами указателя. С помощью макроса `va_start` переменная `temp` связывается с первым необязательным параметром, т.е. с началом списка неизвестной длины.

Использование макроса `va_arg(va_list param, type)` позволяет получить значение очередного фактического параметра типа `type` и заменить значение указателя на адрес следующего фактического параметра в списке.

Макрокоманда `va_end()` предназначена для организации корректного возврата из функции с переменным списком параметров. Макрокоманда `va_end()` должна быть вызвана после того, как функция обработает весь список фактических параметров.

Для наглядности, использование макрокоманд продемонстрировано на представленной схеме.

Реальные фактические параметры

обязательные			**необязательные**			
int	int	float	int	char	int[]	long double
a	b	c	d	e	f	g

```
va_list temp;  //определили указатель
va_start(temp, c);  //настройка указателя
```

обязательные			необязательные			
int	int	float	int	char	int[]	long double
a	b	c	d	e	f	g

▲

temp

```
int myvar = va_arg(temp, int);   /*считали в myvar значение d типа
                                 int, одновременно сместили указатель
                                 на длину переменной типа int */
```

обязательные			необязательные			
int	int	float	int	char	int[]	long double
a	b	c	d	e	f	g

▲

temp

```
char mych = va_arg(temp, char);     /*считали в mych значение e типа
                                    char, одновременно сместили ука-
                                    затель на длину переменной типа
                                    char */
```

обязательные			необязательные			
int	int	float	int	char	int[]	long double
a	b	c	d	e	f	g

▲

temp

```
int *ptr = va_arg(temp, int*);   /*считали в ptr адрес f типа int*,
                                 одновременно сместили указатель на
                                 длину переменной типа int*  */
```

обязательные			необязательные			
int	int	float	int	char	int[]	long double
a	b	c	d	e	f	g

temp

И так далее.

Следующий пример демонстрирует применение макрокоманд. В этой программе функция с переменным количеством параметров принимает в качестве первого параметра количество элементов, далее перечисляются фактические значения типа double, на основе которых создается массив.

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

double * create_array(int size, ...){
     va_list ptr;
     double *arr;
     int i;

     arr = (double*)calloc(size, sizeof(double));
     if(!arr)
          return NULL;

     va_start(ptr, size);
     for(i = 0; i < size; i++)
          arr[i] = va_arg(ptr, double);
     va_end(ptr);

     return arr;
}

void main(){
     double *myarr;
     int n = 10;
     int i;
```

```
     myarr = create_array(n, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0,
9.0, 10.0);
     for(i = 0; i < n; i++){
          printf("%lf\n", myarr[i]);
     }

     free(myarr);
}
```

Рекурсивные функции

Функция называется ***рекурсивной,*** если она может прямо или косвенно вызывать саму себя.

Функция называется ***косвенно рекурсивной***, если она содержит обращение к другой функции, содержащей прямой или косвенный вызов определяемой (первой) функции.

Если в теле функции явно используется вызов этой же функции, то имеет место ***прямая рекурсия***.

Достоинством рекурсии заключается в том, что некоторые типы алгоритмов рекурсивно реализуются проще, чем итеративные эквиваленты.

Классическим примером рекурсивной функции является функция вычисления факториала неотрицательного целого числа:

```
long fact(int n){
     if(n < 0)
          return 0;
     if(n == 0)
          return 1;
     return n * fact(n-1);
}
```

Когда функция вызывает саму себя, в системном стеке выделяется память для новых локальных переменных и параметров, и код функции сначала выполняется с этими новыми переменными. Рекурсивный вызов не создает новой копии функции. Новыми являются только аргументы. При возвращении каждого рекурсивного вызова из стека извлекаются старые локальные переменные и параметры, и выполнение функции возобновляется с «внутренней» точки ее вызова. Поскольку локальные переменные и параметры сохраняются в системном стеке, может наступить момент, когда память в стеке будет переполнена. В результате этого, могут быть потеряны важные данные.

Задание

№	Задание
1	Сочетание из n по *k* – это набор *k* элементов, выбранных из данного множества, содержащего n различных элементов. Число сочетаний из *n* по *k* равно $C_n^k = \frac{n!}{k!(n-k)!}$. Написать программу, вычисляющую число сочетаний по формуле. Использовать рекурсивный вызов функций. Ввод: целые числа *n* и *k*.
2	Необходимо написать программу, которая удаляет все элементы массива с четным индексом. Программа должна состоять из нескольких функций: • Функция для создания динамического массива, содержащая переменное количество параметров. В качестве параметров перечисляются элементы массивов. Функция возвращает указатель на созданный массив. • Функция удаления элементов массива. • Функция вывода массива до и после удаления. Использовать рекурсивный вызов функций. Ввод: размер массива.
3	Написать программу, вычисляющую выражение для *n* = 10. $\sqrt{1+\sqrt{2+\sqrt{3+\sqrt{4+....\sqrt{n}}}}}$ Запрещается использовать циклы. Использовать рекурсивный вызов функций. Ввод: целое число *n*.
4	Написать программу, которая определяет, является ли натуральное число n точной степенью двойки. Запрещается использовать стандартные функции возведения в степень. Используйте рекурсивный вызов функций. Ввод: целое число *n*.
5	Написать программу для поиска *n*-го члена арифметической прогрессии. Используйте рекурсивный вызов функций. Ввод: целое число *n*.
6	Написать программу, которая вычисляет остаток от деления чисел *a* на *b*, т.е. $f(x) = a \bmod b \ \ (a \geq 0, b \geq 0)$. Запрещается использовать операции умножения и деления. Использовать рекурсивный вызов функций. Ввод: целые числа *a* и *b*.
	Написать программу, которая производит замену минимального простого числа последовательности на максимальное число последовательности. Программа должна состоять из нескольких функций: • Функция для создания массива с помощью средств динамического распределения памяти и заполнения массива случайными числами. Функция возвращает указатель на созданный массив. • Функции поиска минимального простого и максимального элементов после-

	довательности. • Замена минимального простого элемента последовательности на максимальный элемент последовательности. • Вывод массива на экран до и после замены, а также пары индексов элементов, которые подверглись замене. Запрещается использовать циклы. Использовать рекурсивный вызов функций. Ввод: размер массива.
7	Написать программу для поиска *n*-го члена геометрической прогрессии. Используйте рекурсивный вызов функций. Ввод: целое число *n*.
8	Написать программу, вычисляющую x^n для любого вещественного числа *x* и любого целого *n*. Запрещается использовать стандартные функции возведения числа в степень. Использовать рекурсивный вызов функций. Ввод: вещественное число *x* и целое число *n*.
9	Необходимо написать программу, которая удаляет все элементы массива с нечетным значением. Программа должна состоять из нескольких функций: • Функция для создания динамического массива, содержащая переменное количество параметров. В качестве параметров перечисляются элементы массивов. Функция возвращает указатель на созданный массив. • Функция удаления элементов массива. • Функция вывода массива до и после удаления. Использовать рекурсивный вызов функций. Ввод: размер массива.
10	Написать программу, в которой выводятся индексы элементов с максимальным значением. Программа должна состоять из нескольких функций: • Функция для создания динамического массива и заполнения массива случайными числами. Функция возвращает указатель на созданный массив. • Функция поиска максимального элемента. • Функция вывода индексов элементов с максимальным значением. Запрещается использовать циклы. Использовать рекурсивный вызов функций. Ввод: размер массива.
11	Написать программу, которая вычисляет среднее арифметическое значение элементов массива. Программа должна состоять из нескольких функций: • Функция для создания динамического массива и заполнения массива случайными числами. Функция возвращает указатель на созданный массив. • Функция вычисления среднего арифметического значения. Запрещается использовать циклы. Использовать рекурсивный вызов функций. Ввод: размер массива.

12	Написать программу, которая выполняет разворот числа *n*. Например, разворот числа 102 дает результат 201. Запрещается использовать массивы, строки, и т.д. Использовать только рекурсивный вызов функций и арифметические операции. Ввод: целое число *n.*
13	Написать программу, для нахождения значения выражения по формуле для n=2..10: $$\frac{n!}{\sqrt{1+\sqrt{2+\sqrt{3+\sqrt{4+...\sqrt{n}}}}}}$$ Запрещается использовать циклы. Использовать рекурсивный вызов функций.
14	Написать программу для нахождения парных простых чисел (чисел-близнецов) из множества *n, n+1, n+2, …, 2n*. Использовать рекурсивный вызов функций. Ввод: целое число *n*.
15	Написать программу для перевода двоичной записи в десятичную запись. Использовать рекурсивный вызов функций. Ввод: целое число в двоичной форме.
16	Написать программу для перевода десятичной записи в двоичную запись. Использовать рекурсивный вызов функций. Ввод: целое число в десятичной форме.
17	Написать программу для перевода двоичной записи в восьмеричную запись. Использовать рекурсивный вызов функций. Ввод: целое число в двоичной форме.
18	Написать программу для перевода двоичной записи в шестнадцатеричную запись. Использовать рекурсивный вызов функций. Ввод: целое число в двоичной форме.
19	Написать программу для вычисления суммы 1! + 2! + 3! + … +n!. Запрещается использовать циклы. Использовать рекурсивный вызов функций. Ввод: целое число *n*.
20	Написать программу для определения n-го члена ряда Фибоначчи. Использовать рекурсивный вызов функций. Ввод: целое число *n*.
21	Написать программу для вычисления суммы членов ряда по формуле $$\sum_{k=1}^{n}\frac{x^k}{k!}$$ Запрещается использовать стандартные функции возведения в степень и циклы. Использовать рекурсивный вызов функций. Ввод: целые числа *x* и *n*.

22	Написать программу для вычисления суммы членов ряда по формуле $$\prod_{k=1}^{n}\left(\frac{(1-x)^{k+1}+1}{((k-1)!+1)^2}\right)$$ Запрещается использовать стандартные функции возведения в степень и циклы. Использовать рекурсивный вызов функций. Ввод: целые числа *x* и *n*.
23	Написать программу, которая производит замену каждого элемента массива на среднее арифметическое его и его соседей справа и слева. Программа должна состоять из нескольких функций: • Функция для создания массива с помощью средств динамического распределения памяти и заполнения массива случайными числами. Функция возвращает указатель на созданный массив. • Замена элементов массива на среднее арифметическое. • Вывод массива на экран до и после замены. Запрещается использовать циклы. Использовать рекурсивный вызов функций. Ввод: размер массива.
24	Написать программу для вычисления выражения по формуле для *n* = 10…5 $$\sqrt{n!-\sqrt{(n-1)!-\sqrt{(n-2)!-\ldots-\sqrt{2!-1}}}}$$ Запрещается использовать циклы. Использовать рекурсивный вызов функций.
25	Написать программу для перевода восьмеричной записи в шестнадцатеричную запись. Использовать рекурсивный вызов функций. Ввод: целое число в восьмеричной форме.
26	Написать программу для перевода шестнадцатеричной записи в восьмеричную запись. Использовать рекурсивный вызов функций. Ввод: целое число в шестнадцатеричной форме.
27	Написать программу для вычисления суммы членов ряда по формуле $$\sum_{k=1}^{n}\frac{(-1)^k(k+1)}{k!}$$ Запрещается использовать стандартные функции возведения в степень и циклы. Использовать рекурсивный вызов функций. Ввод: целое число *n*.
28	Написать программу для вычисления суммы членов ряда по формуле $$\sum_{k=1}^{n}\left(1-\frac{1}{k!}\right)^4$$ Запрещается использовать функции возведения в степень. Использовать рекурсивный вызов функций. Ввод: целое число *n*.

29	Написать программу для вычисления суммы членов ряда по формуле $$\sum_{k=1}^{n}\sum_{m=1}^{n}\frac{n}{2k+m}$$ Использовать рекурсивный вызов функций. Ввод: целое число *n*.
30	Написать программу, которая производит замену всех отрицательных значений элементов массива на положительные значения. Программа должна состоять из нескольких функций: • Функция для создания массива с помощью средств динамического распределения памяти и заполнения массива случайными числами. Функция возвращает указатель на созданный массив. • Замена отрицательных значений элементов массива на положительные значения. • Вывод массива на экран до и после замены. Запрещается использовать циклы. Использовать рекурсивный вызов функций. Ввод: размер массива.
31	Написать программу, которая производит замену всех элементов массива с максимальным значением на последний элемент массива. Программа должна состоять из нескольких функций: • Функция для создания массива с помощью средств динамического распределения памяти и заполнения массива случайными числами. Функция возвращает указатель на созданный массив. • Замена элементов с максимальным значением на последний элемент. • Вывод массива на экран до и после замены. Запрещается использовать циклы. Использовать рекурсивный вызов функций. Ввод: размер массива.
32	Написать программу для нахождения наибольшего числа *k*, для которого выполняется неравенство $5k < n$, где *n* – целое число ($1 < n < 10^5$). Запрещается использовать стандартные функции возведения в степень. Использовать рекурсивный вызов функций. Ввод: целое число *n*.
33	Написать программу для вычисления суммы членов ряда по формуле $$\sum_{k=1}^{n}\frac{1}{(2k)^3}$$ Запрещается использовать стандартные функции возведения в степень и циклы. Использовать рекурсивный вызов функций. Ввод: целое число *n*.

34	Написать программу для вычисления суммы членов ряда по формуле $$\sum_{k=1}^{n} \frac{1}{k^k}$$ Запрещается использовать стандартные функции возведения в степень и циклы. Использовать рекурсивный вызов функций. Ввод: целое число *n*.
35	Написать программу для вычисления суммы членов ряда по формуле $$\sum_{k=1}^{n} \frac{n(n+1)}{2}$$ Запрещается использовать стандартные функции возведения в степень и циклы. Использовать рекурсивный вызов функций. Ввод: целое число *n*.

Контрольные вопросы

1. Что такое рекурсивная функция?
   a) Функция, которая повторно вызывает функцию main()
   b) Функция, которая имеет тип void.
   c) Функция, которая вызывает сама себя прямо или косвенно.
   d) Функция, которая принимает переменное количество параметров.
   e) Нет правильного ответа.

2. Что называется формальными параметрами функции?
   a) Переменные, используемые для передачи значений в вызывающую функцию.
   b) Переменные, через которые происходит присваиванием значений соответствующим фактическим параметрам при вызове функций.
   c) Переменные, используемые для передачи значений в вызывающую функцию.
   d) Переменные, используемые внутри тела функции и получающие значение при вызове функции путем копирования в них значений фактических параметров.
   e) Нет правильного ответа.

3. Что называется прототипом функции?
   a) Прототип функции предназначен для передачи управления и фактических аргументов заданной функции.
   b) Прототип функции определяет ее имя, тип возвращаемого значения, класс памяти и может также задавать тип некоторых или всех аргументов функции.
   c) Прототип функции определяет имя, формальные параметры и тело функции.
   d) Прототип функции – это совокупность объявлений и операторов, предназначенная для выполнения некоторой определенной задачи.
   e) Нет правильного ответа.

4. Как можно описать формальный параметр функции, соответствующий фактическому параметру – массиву из 10 целых чисел?

```
a) int a[10]
b) int a
c) int *a
```

d) `int *a[]`
e) `int *a[10]`
f) Нет правильного ответа

5. Выбрать прототип для функции

```
void print(char str[]){
     printf("%s", str);
}
```

a) `int print(char []);`
b) `void print(char []);`
c) `void print(void);`
d) `int print(void);`
e) Нет правильного ответа.

6. Какой прототип имеет следующая функция?

```
void menu(void){
     printf("menu choices");
}
```

a) `int menu(char []);`
b) `void menu(char []);`
c) `void menu(void);`
d) `int menu(void);`
e) Нет правильного ответа.

7. Для чего предназначена операция return?
   a) Для возврата значения функцией.
   b) Для немедленного завершения итерации и выхода из цикла.
   c) Для немедленного завершения функции.
   d) Для перехода к следующей итерации цикла.
   e) Нет правильного ответа.

8. Выберите правильное определение функции, у которой в качестве параметра указан двумерный массив.
   a) `void func(int &param);`
   b) `void func(double *param);`
   c) `void func(double **param);`
   d) `void func(double param[][]);`
   e) `void func(double param[][20]);`
   f) Нет правильного ответа.

Лабораторная работа №8.
Работа со строками

В языке С нет средств для работы со строками, как с единым целым. ***Строка*** в языке программирования С представляется в виде одномерного символьного массива, который завершается символом конца строки `'\0'` (нулевым символом). Поэтому при определении строки необходимо учитывать нулевой символ и задавать размер массива на единицу больше.

Строковый литерал – это список символов, заключенный в двойные кавычки. Например, `"Hello, world!"`.

Строка, представленная в виде `""` называется ***нулевой*** и состоит только из нулевого символа.

Для работы со строками язык С поддерживает множество функций, которые расположены в стандартной библиотеке `string.h`. Рассмотрим некоторые из них.

Копирование

```
int strcpy(char *str1, const char *str2);
```

Функция копирует, начиная с адреса `str1` в память, на которую указывает `str2`, до тех пор, пока не будет скопирован нулевой байт. В качестве результата возвращает число скопированных символов.

Конкатенация (соединение)

```
char *strcat(char *str1, const char *str2);
```

Функция присоединяет строку, на которую указывает `str2` к концу строки, на которую указывает `str1`. Обе строки должны завершаться нулевым символом. В качестве результата возвращается указатель на первый символ, расположенный по адресу `str1`.

Сравнение строк

```
int strcmp(const char *str1, const char *str2);
```

Функция сравнивает посимвольно строку `str2` со строкой `str1`. Сравнение осуществляется лексикографически, т.е. в соответствии с алфавитным порядком. Если строка `str1` больше строки `str2`, то в качестве результата функция возвращает положительное число. Если `str1 < str2`, то в качестве результата возвращается отрицательное число. И если строки равны, то возвращается значение 0.

Длина строки

```
unsigned strlen(char *str);
```

Возвращает длину строки, на которую указывает `str`.

Вхождение подстроки в строку

```
char *strstr(const char *str1, const char *str2);
```

Функция возвращает указатель на первое вхождение строки `str2` в строку `str1`. Если вхождения не найдены, возвращается NULL.

Вхождение символа в строку

```
char *strchr(const char *str, int ch);
```

Функция возвращает указатель на первое вхождение символа `ch` в строке `str`.

Задание

№	Задание
1	Написать программу для подсчета слов в строке. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С. Под термином «слово» понимается любая последовательность символов, ограниченная любым количеством пробелов и/или символом начала/конца строки.
2	Написать программу, которая выделяет из строки подстроку заданной длины, начиная с заданной позиции. Строка, длина и начало позиции передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
3	Написать программу, которая дописывает одну строку в конец другой. Строки передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
4	Написать программу, которая символы строки выводятся в два столбца. В левом столбце – символы до середины, в правом столбце – символы после середины. Если длина строки является нечетным числом, то средний символ строки попадает в первый столбец. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
5	Написать программу, в которой осуществляется поиск первого вхождения подстроки в строку. Строки передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
6	Написать программу, определяющую, является ли введенная строка десятичным числом. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
7	Написать программу, которая сравнивает две строки, независимо от регистра символов. Строки передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
8	Написать программу, определяющую, является ли введенная строка шестнадцатеричным числом. Формат шестнадцатеричного числа 0xXXXXh, где XXXX – шестнадцатеричная запись числа. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
9	Написать программу, которая перекрывает символы одной строки заданным количеством символов другой строки, начиная с заданной позиции. Строки, количество символов и номер позиции передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
10	Написать программу, которая по цифровой записи числа выводит его запись в словесной форме на русском языке. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
11	Написать программу, которая удаляет из строки слово с заданным номером. Строка и слово передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С. Под термином «слово» понимается любая последовательность символов, ограничен-

	ная любым количеством пробелов.
12	Написать программу, которая выводит из строки отдельно гласные и отдельно согласные символы по мере их появления. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
13	Написать программу, которая удаляет из начала и из конца строки заданный символ. Строка и символ передаются в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
14	Написать программу, определяющую, является ли введенная строка палиндромом, т.е. читается одинаково как слева направо, так и справа налево. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
15	Написать программу, в которой из строки удаляется заданное количество символов с заданной позиции. Строка, количество символов и номер позиции передаются в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
16	Написать программу, которая из введенной строки выводит сначала слова, заканчивающиеся на гласную букву, а затем вывести слова, заканчивающиеся на согласную букву. Строка, количество символов и номер позиции передаются в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С. Под термином «слово» понимается любая последовательность символов, ограниченная любым количеством пробелов и/или символом начала/конца строки.
17	Написать программу, которая находит первое вхождение символа в строку и разбивает строку на две части - до и после символа. Строка и символ передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
18	Написать программу, которая сортирует символы в каждом слове строки по убыванию, пробелы не учитываются. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С. Под термином «слово» понимается любая последовательность символов, ограниченная любым количеством пробелов и/или символом начала/конца строки.
19	Написать программу, в которой осуществляется поиск последнего вхождения подстроки в строку. Строки передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
20	Написать программу, которая сортирует символы внутри строки по возрастанию, пробелы не учитываются. Строка передается функции в качестве аргумента. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
21	Написать программу, которая в строке str1 заменяем все вхождения строки str2 строкой str3. Строки передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.

22	Написать программу, которая выделяет из строки заданное количество слов, начиная со слова с заданным номером. Строка, количество слов, номер первого слова передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С. Под термином «слово» понимается любая последовательность символов, ограниченная любым количеством пробелов и/или символом начала/конца строки.
23	Написать программу, которая выводит из строки длину слова с заданным номером. Строка и номер слова передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С. Под термином «слово» понимается любая последовательность символов, ограниченная любым количеством пробелов и/или символом начала/конца строки.
24	Написать программу для определения самого длинного слова в строке. Строка передается функции в качестве аргумента. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С. Под термином «слово» понимается любая последовательность символов, ограниченная любым количеством пробелов и/или символом начала/конца строки.
25	Написать программу, которая вставляет одну строку в другую строку, начиная с заданной позиции. Строки и номер позиции передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
26	Написать программу, которая в каждом слове строки удаляет символы, совпадающие с последним символом слова. Строка передается функции в качестве аргумента. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
27	Написать программу, которая заменяет в строке все множественные вхождения символа одним. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
28	Написать программу, которая удаляет лишние пробелы в строке. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
29	Написать программу, которая переводит текстовую запись числа на русском (или английском) языке в цифровую запись. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
30	Написать программу, которая заменяет в строке одну заданную комбинацию символов другой заданной комбинацией. Строка и комбинации символов передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
31	Написать программу, которая шифрует строку методом Цезаря. Строка и ключ передаются в функцию в качестве параметров.
32	Написать программу, которая переписывает все символы строки в обратном порядке. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
33	Написать программу, которая меняет местами два слова в строке. Строка и номера слов передаются в функцию в качестве параметров. При реализации функции

	запрещается пользоваться функциями стандартной библиотеки языка С. Под термином «слово» понимается любая последовательность символов, ограниченная любым количеством пробелов и/или символом начала/конца строки.
34	Написать программу, которая копирует одну строку в другую строку заданное количество раз, начиная с заданной позиции. Строки и количество повторений передаются в функцию в качестве параметров. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.
35	Написать программу, которая удаляет из строки все знаки пунктуации. Строка передается в функцию в качестве параметра. При реализации функции запрещается пользоваться функциями стандартной библиотеки языка С.

Контрольные вопросы

1. Какой оператор позволяет проверить равенство двух строк?
   a) `if(!strcmp(str1, str2));`
   b) `if(strcmp(str1, str2) < 0);`
   c) `if(strcmp(str1, str2) > 0);`
   d) `if(strcmp(str1, str2) != 0);`
   e) нет правильного ответа

2. Какая длина у строки «C programming language»?
   a) 22
   b) 20
   c) 23
   d) 21
   e) Нет правильного ответа.

3. Как определить строковую переменную?
   a) `string str;`
   b) `char str;`
   c) `char *str;`
   d) `char str[20];`
   e) `string *str;`
   f) `string str[20];`
   g) Нет правильного ответа.

4. Каким символом завершается строка в языке С?
   a) ‘\n’
   b) ‘\r’
   c) ‘\a’
   d) ‘\0’
   e) ‘.’
   f) Нет правильного ответа.

5. Какая из следующих функций предназначена для сравнения строк?
   a) `compare()`
   b) `strcmp()`
   c) `strcompare()`
   d) `strcat()`
   e) нет правильного ответа.

6. Какая из следующих функций добавляет одну строку в конец другой?
   a) `stradd();`
   b) `add()`
   c) `strcat()`
   d) `strstr();`
   e) `append()`
   f) `нет правильного ответа.`

7. Что называется строкой в языке С?
   a) Массив элементов типа char, заканчивающийся нулевым символом.
   b) Массив элементов типа char, заканчивающийся символом ‘\n’.
   c) Отдельный тип данных, обозначаемый ключевым словом string.
   d) Массив элементов типа char.
   e) Нет правильного ответа.

8. Как объявить указатель ptrstr типа char и инициализировать его адресом строки «Hello»?
   a) `char ptrstr = “hello”;`
   b) `*ptrstr = “hello”;`
   c) `char *ptrstr = “hello”;`
   d) `char ptrstr = “hello”;`
   e) `нет правильного ответа.`

Лабораторная работа №9.
Типовые динамические структуры данных.
Стек. Двоичное дерево поиска.

Стек.

Стек – это одномерный, динамически изменяемый набор данных. Добавление нового элемента осуществляется только в вершину стека. Элемент, находящийся на вершине стека, называется «головой». Извлечение (удаление) элемента осуществляется только из вершины стека.

Для обозначения стека используется аббревиатура FILO (First In Last Out) – «Первым пришел, последним ушел». Принцип организации стека можно сравнить по аналогии с магазином огнестрельного оружия, когда первый вставленный патрон последним поступает в ствол. Для реализации стека используется массив или односвязный список.

Рис. 9.1. Реализация стека с помощью односвязного списка

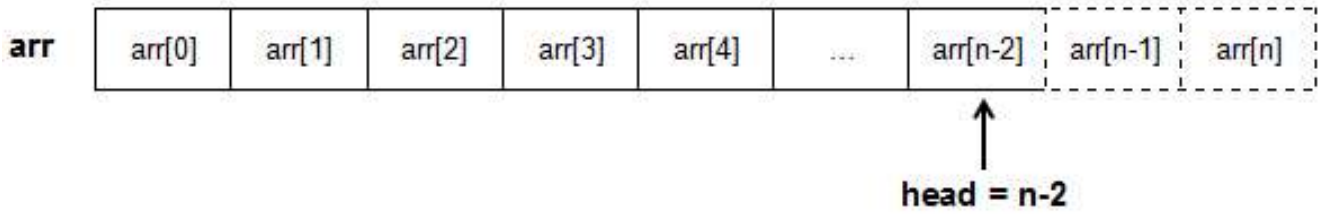


Рис. 9.1. Реализация стека с помощью массива

Существуют три операции со стеком:

1. добавление нового (головного) элемента;
2. удаление элемента из головы;
3. чтение головного элемента.

Двоичное дерево поиска

Дерево (корневое дерево) – это конечное множество N узлов, у которого:

- *Имеется один специально обозначенный узел, называемый корнем дерева.*

- *Остальные узлы содержатся в m ≥ 0 попарно непересекающихся подмножествах $N_1, N_2, \ldots, N_m$, каждое из которых является деревом.*

Висячее дерево – дерево, в котором не выделен корень.

Степень узла – количество поддеревьев узла. Узел нулевой степени называется листом.

Уровень корня равен 1. Уровни поддеревьев узла имеют уровень на 1 больше, чем уровень узла.

Высота дерева – это наибольший уровень узла в дереве.

На рисунке 9.3. представлено корневое дерево, высота которого равна 4.

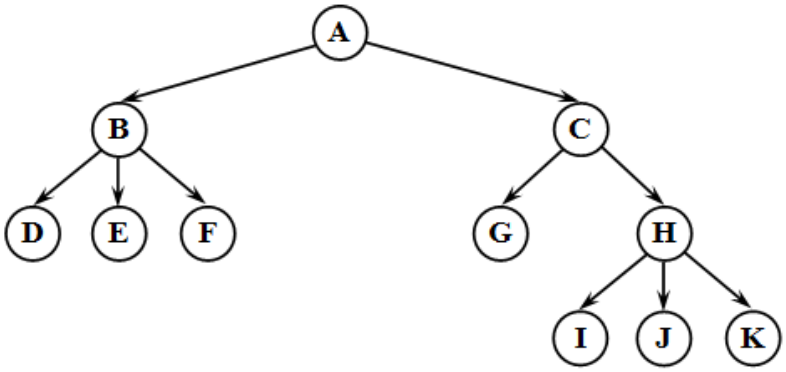


Рис. 9.3. Корневое дерево

Бинарное дерево – конечное множество узлов, которое или пусто, или состоит из корня и двух непересекающихся бинарных деревьев, называемых левым и правым поддеревьями корня.

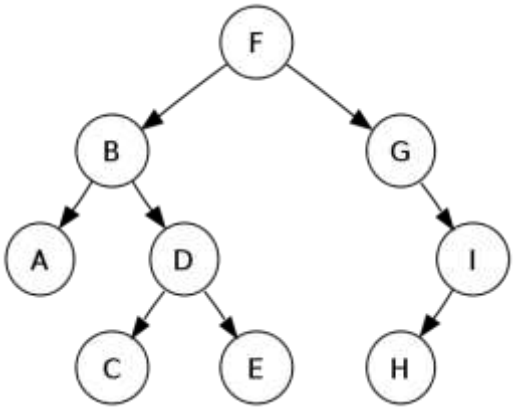


Рис. 9.4. Бинарное дерево

Обход дерева подразумевает такой порядок работы с его узлами, для которого каждый из них посещается точно один раз.

Для обхода бинарных деревьев могут быть использованы три способа, определяемых рекурсивно:

1. Прямой обход
   a) Обработка корня
   b) Обход левого поддерева
   c) Обработка правого поддерева

2. Обратный обход
   a) Обход левого поддерева
   b) Обработка корня
   c) Обход правого поддерева

3. Концевой обход
   a) Обход левого поддерева
   b) Обход правого поддерева
   c) Обработка корня

Задания

Задание 1. Стек

На основании лабораторной работы №6 необходимо разработать программу для представления данных (строк) в таблице виде стека. Для этого необходимо реализовать следующие функции:

- *Инициализация стека (головного элемента)*
- *Ввод данных и добавление нового элемента (строки) в стек.*
- *Чтение головного элемента стека.*
- *Извлечение головного элемента стека*
- *Вывод таблицы на экран (включая границы).*
- *Уничтожение стека (удаление всех данных из стека).*

Подсказка! Каждый элемент стека должен иметь указатель на элемент такого же типа данных. Т.е., внутри структуры должен быть указатель на следующий элемент такого же структурного типа.

Задание 2. Двоичное дерево поиска

Написать программу, реализующую следующие операции с двоичным деревом:

- *Построить двоичное дерево и заполнить его случайными значениями.*
- *Получить высоту дерева.*
- *Получить количество узлов бинарного дерева.*
- *Копирование бинарного дерева.*
- *Зеркальное отображение бинарного дерева.*
- *Сравнение двух деревьев.*
- *Поиск заданного элемента в дереве и вывод маршрута до него.*

Реализовать каждую операцию в виде отдельной функции.

Лабораторная работа №10
Ввод/вывод. Функция main() Параметры функции main()

Ввод/вывод

В языке С для организации ввода/вывода используются функции, реализованные в библиотеке языка С. В библиотеке языка С поддерживается три уровня ввода/вывода:

- *потоковый ввод/вывод;*
- *ввод/вывод нижнего уровня;*
- *ввод/вывод для консоли и портов.*

Потоковый ввод/вывод

На уровне потокового ввода/вывода обмен данными производится побайтно

При вводе с диска (при чтении файла) данные помещаются в буфер операционной системы, а затем побайтно или определенными порциями передаются программе. При выводе данных в файл они накапливаются в буфере, а при заполнении буфера записываются в виде единого блока на диск за одно обращение. Функции библиотеки ввода/вывода языка С, поддерживающие обмен данными с файлами на уровне потока, позволяют обрабатывать данные различных размеров и форматов, обеспечивая при этом буферизованный ввод и вывод.

Поток – это файл вместе с предоставляемыми средствами буферизации.

Действия, осуществляемые при работе с потоком:

- *открытие и закрытие потока (связывание указателя на поток с конкретным файлом);*
- *ввод и вывод: символов, строк, форматированных данных, порций данных произвольной длины;*
- *анализ ошибок потокового ввода/вывода и условий достижения конца потока (конца файла);*
- *управление буферизацией потока и размером буфера;*
- *получение и установка указателей текущей позиции в потоке.*

Для работы с потоком его необходимо объявить как структурный тип **`FILE`**. В структуре **`FILE`** содержится указатель текущей позиции в потоке, указатель на буфер и другая информация. При открытии потока в программу возвращается указатель на поток.

Для открытия файла используется функция

```
FILE * fopen(имя_файла, режим_открытия);
```

Параметры `имя_файла` и `режим_открытия` являются указателями на массив символов. Существует шесть режимов для открытия фа йлов:

1. `"w"` – новый текстовый файл открывается для записи. Если файл уже существовал, то предыдущее содержимое стирается, файл создается заново.
2. `"r"` – существующий текстовый файл открывается только для чтения.
3. `"a"` – текстовый файл открывается (или создается, если нет файла) для добавления в него новой порции информации (добавление в конец файла). В отличие

от режима “w” режим “a” позволяет открывать уже существующий файл, не уничтожая его предыдущей версии, и писать в продолжение файла.

4. `"w+"` – новый текстовый файл открывается для записи и последующих многократных исправлений. Если файл уже существует, то предыдущее содержимое стирается. Последующие после открытия файла запись и чтение из него допустимы в любом месте файла, в том числе запись разрешена и в конец файла, т.е. файл может увеличиваться.
5. `"r+"` – существующий текстовый файл открывается как для чтения, так и для записи в любом месте файла; однако в этом режиме невозможна запись в конец файла, т.е. недопустимо увеличение размеров файла.
6. `"a+"` – текстовый файл открывается или создается (если файла нет) и становится доступным для изменений, т.е. для записи и для чтения в любом месте; при этом в отличие от режима “r+” в режиме “a+” можно вести запись в конец файла, т.е. увеличивать его размеры.

После окончания работы с файлом его необходимо закрыть явно, используя функцию

```
int fclose(указатель_на_поток);
```

Для вывода сообщения об ошибке используется функция

```
void perror(const char *str);
```

Стандартные файлы и функции для работы с ними

Когда программа начинает выполняться, автоматически открываются пять потоков, из которых основными являются:

- *стандартный поток ввода;*
- *стандартный поток вывода;*
- *стандартный поток вывода сообщений об ошибках.*

Для ввода/вывода данных с помощью стандартных потоков в библиотеке языка С определены следующие функции:

Ввод/вывод отдельного символа

`int getchar(void)`
Читает очередной символ из стандартного потока.
`int putchar(int ch)`
Записывает символ `ch` в стандартный поток вывода.

Ввод/вывод строки

`char *gets(char * str)`
Считывает строку `str` из стандартного входного потока.
`char puts(char * str)`
Записывает строку `str` в стандартный поток вывода

Ввод/вывод в режиме форматирования данных

int printf(форматная_строка, список_аргументов)
Функция форматированного вывода в стандартный поток
int scanf(форматная_строка, список_аргументов)
Функция форматированного ввода из стандартного потока.

Работа с файлами на диске

Также как и для стандартных потоков ввода/вывода существует набор функций для работы с файлами на диске:

Ввод (чтение) одного символа из файла

int fgetc(FILE * ptr)
Считывает символ из файла, связанного с потоком `str`.
int getc(FILE * ptr)
Считывает символ из файла, связанного с потоком `str`. В отличие от `fgetc`, функция может быть определена как макрос, поэтому, передаваемый аргумент не должен быть выражением.

Запись одного символа в файл

int fputc(int ch, FILE * ptr)
Выводит символ `ch` в стандартный поток, связанный с потоком `ptr`.
int putc(int c, FILE * ptr)
Выводит символ `ch` в стандартный поток, связанный с потоком `ptr`. В отличие от `fputc`, функция может быть определена как макрос, поэтому, передаваемый аргумент не должен быть выражением.

Форматный обмен с файлами

int fprintf(FILE * ptr, форматная_строка, список_переменных)
Функция форматного вывода в файл, связанного с потоком `ptr`.
int fscanf(ptr, форматная_строка, список_переменных)
Функция форматного ввода из файла, связанного с потоком `ptr`.

Строковый обмен с файлами

char * fgets(char *str, int n, FILE * ptr)
Читает не более `n-1` символов из файла, связанного с потоком `ptr`, в массив, адресуемый указателем `str`. За последним символом, помещенным в массив, записывается `'\0'`.
int fputs(const char *str, FILE * ptr)
Записывает символьную строку, на которую указывает `str`, в файл, связанный с потоком `ptr`.

Позиционирование в потоке
int fseek(FILE * ptr, long offset, int point) Перемещает указатель текущей позиции в файле, связанном с потоком `ptr`, на `offset` байт относительно точки отсчета, определенной значением `point` (0 – от начала файла, 1 – от текущей позиции, 2 – от конца файла).
long ftell(FILE * ptr) Возвращает значение указателя текущей позиции в файле, связанном с потоком `ptr`.

Ввод/вывод нижнего уровня

Ввод/вывод, ориентированный на поток, обычно применяется для выполнения достаточно стандартных операций ввода/вывода. Применение рассмотренных выше функций обмена с потоками гарантирует успешность переноса программы в различные операционные системы.

Функции ввода/вывода более низкого уровня позволяют пользоваться средствами ввода/вывода операционной системы непосредственно. При этом не выполняется буферизация и форматирование данных.

При низкоуровневом открытии файла с ним связывается указатель файла, а дескриптор файла.

Дескриптор – это целое значение, характеризующее размещение информации об открытии файла во внутренних таблицах операционной системы.

В библиотеки языка С имеются следующие основные функции ввода/вывода нижнего уровня.

int open(const char * path, int access, unsigned mode) Открывает файл, на имя которого указывает path в режиме, заданном `access`. Возвращает дескриптор файла. Mode определяет тип файла и права доступа.
int close(int handle) Закрывает файл, связанный с дескриптором `handle`.
int creat(const char *path, int mode) Создает новый файл в режиме, заданном в `mode`.
int read(int handle, void *buf, unsigned length) Читает length байтов из файла, связанного с дескриптором `handle` в буфер, на который указывает `buf`.
int write(int handle, void * buf, unsigned nbyte) Записывает `nbyte` байтов из буфера, на который указывает `buf`, в файл, связанный с дескриптором `handle`.
int eof(int handle) Проверяет, достигнут ли конец файла, связанного с дескриптором `handle`.
long lseek(int handle, long offset, int point) Перемещает указатель текущей позиции в файле, связанном с дескриптором `handle` на `offset` байтов относительно точки отсчета, определенной значением point (0 – от

начала файла, 1 – от текущей позиции, 2 – от конца файла).
long tell(int handle) Возвращает значение указателя текущей позиции в файле, связанном с дескриптором handle.

Функция main(). Параметры функции main()

Основная функция каждой программы может иметь такой заголовок

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

, где argv – массив указателей на строки; argc – параметр типа int, значение которого определяет размер массива argv.

Параметры функции main() предназначены для обеспечения связи выполняемой программы с операционной системой, точнее, с командной строкой, из которой запускается программа, тем самым передавать исполняемой программе любую информацию.

Если нет необходимости передавать информацию исполняемой программе, то параметры обычно опускаются.

При запуске программы из операционной системы командная строка явно доступна, и именно в ней записывается имя выполняемой программы. Вслед за именем можно разместить нужное количество «слов», разделяя их друг от друга пробелами. Каждое «слово» из командной строки становится строкой значением, на которую указывает очередной элемент параметра argv[i], где 0 < i < argc.

Рассмотрим пример. При запуске программы myprogram.exe из командной строки ей можно передать параметры.

C:\>myprogram.exe param1 param2 param3

В момент запуска содержимое параметров функции main() будет следующим:

```
argc = 4
argv[0] = "myprogram.exe"
argv[1] = "param1"
argv[2] = "param2"
argv[3] = "param3".
```

Задания

Написать программу, которая читает текстовый файл и выполняет его преобразование в зависимости от варианта.

1. Имя файла, который подлежит обработке, должно быть параметром программы.
2. Файл должен иметь не менее 15-20 строк текста.
3. Самостоятельно подготовьте исходный текстовый файл.
4. Программу следует разрабатывать, исходя из предположения, что весь текст из файла не может быть размещен в оперативной памяти.
5. Допускается создавать при выполнении программы временные файлы, которые не должны сохраняться после окончания программы.

№	Задание
1	В каждой строке удалить лишние пробелы между словами и, сохраняя первоначальную длину строки, разместить текст по центру строки.
2	В каждой строке заменить последовательность слов «один», «два», «три» на «1-2-3»
3	В каждой строке вставить после знаков препинания пробелы, если их там нет.
4	В каждой строке поменять местами соседние два слова.
5	Разбить каждую строку на две строки приблизительно одного размера, не разрывая слова.
6	В каждой строке все слова, длина которых превышает среднюю длину слов в строке, сократить до средней длины.
7	Разместить текст в несколько столбцов, так чтобы n-ое слово i-ой строки размещалось под n-ым словом i+1-ой строки.
8	Скопировать второе слово каждой строки в начало следующей строки.
9	В каждой строке удалить пары слов, в которых одно слово является зеркальным отображением второго.
10	В каждой строке удалить лишние пробелы между словами и разместить их по левому краю, сохраняя первоначальную длину строки.
11	Удалить из текста все слова, состоящие из одной буквы.
12	Удалить из текста все слова с четными номерами.
13	В каждом слове все буквы разместить в алфавитном порядке.
14	В каждой строке удалить те слова, в которых первые три буквы совпадают с начальными буквами последнего слова строки.
15	В каждом слове строки изменить порядок букв на противоположный.
16	Во всем тексте вставить переход на новую строку там, где есть два или больше пробела подряд.
17	В каждой строке поменять местами первое слово с последним, втрое – с предпоследним.
18	В каждой строке самое длинное слово заменит на «длинное слово».
19	Удалить из текста все слова, в которых содержатся буквосочетания «ов».

20	В каждой строке удалить лишние пробелы между словами и разместить их по правому краю, сохраняя первоначальную длину строки.
21	В каждой строке заменить последовательность слов «1», «2», «3» на «один-два-три».
22	Удалить из текста слова, в которых какие-либо буквы повторяются.
23	В каждой строке все слова разместить в алфавитном порядке.
24	Переформировать весь текст так, чтобы каждое предложение занимало одну строку. (Признак конца предложения - точка.)
25	В каждой строке все слова, которые совпадают с первым словом строки, заменить на второе слово.
26	Во всем тексте все появления слова (выберите слово сами) взять в кавычки.
27	В каждой строке для всех слов, которые встречаются в строке больше одного раза, оставить только первое вхождение.
28	Во всем тексте после каждых трех слов вставить слово «программист».
29	В каждой строке заменить ту комбинацию из трех символов, которая чаще всех встречается в строке, на символ '$'.
30	Выровнять все строки до размера самой длинной строки в файле, вставляя между словами пробелы.
31	Удалить из строки все знаки пунктуации.
32	Переписать все символы каждой строки в обратном порядке.
33	Шифровать и дешифровать текст, используя метод Цезаря.
34	Заменить все вхождения цифровых записей чисел в текстовую запись на русском.
35	Заменить в каждой строке множественные вхождения символа одним символом.

Контрольные вопросы

1. Какое назначение имеет функция `getchar` в языке С?
   a) Функция записывает один символ с определенным кодом в выходной поток. Обращение к `getchar` приводит к появлению на экране одного символа.
   b) Функция читает один символ из входного потока, код которого выдается в качестве результата.
   c) Функция читает строку символов из входного потока и помещает ее по определенному адресу.
   d) Функция записывает строку символов в выходной поток. Обращение к `getchar` приводит к появлению на экране строки символов.
   e) Нет правильного ответа.

2. Дан указатель на файл с именем `input_file`. Как открыть файл `myfile.txt` для чтения?
   a) `input_file = "myfile.txt" opened as "rt";`
   b) `open input_file as "myfile.txt" for "rt";`
   c) `fopen(input_file, "myfile.txt", "rt");`
   d) `input_file = fopen("myfile.txt", "rt");`
   e) `Нет правильного ответа.`

3. В чем заключаются особенности работы функций ввода/вывода нижнего уровня?
   a) Отсутствует буферизация данных. Отсутствует форматирование данных.
   b) Используются средства ввода/вывода операционной системы ЭВМ.
   c) Используется потоковый ввод/вывод информации.
   d) Обмен данными производится побайтно
   e) Нет правильного ответа.

4. Как закрыть файл, связанный с потоком `input_file`?
   a) `close input_file;`
   b) `fclose(input_file);`
   c) `fcloseall();`
   d) `input_file(fclose);`
   e) `Нет правильного ответа.`

5. Как с помощью функции `scanf` ввести строку символов в массив words?
   a) `scanf("%s\n", words);`
   b) `scanf(" %c", words);`
   c) `scanf("%c", words);`
   d) `scanf("%s", words);`
   e) `Нет правильного ответа.`

6. С чего начинается выполнение любой программы на языке С?
   a) `С функции start()`
   b) `С функции initialize()`
   c) `С функции main()`
   d) `С функции begin()`
   e) `Нет правильного ответа.`

7. Какая функция предназначена для получения одного символа из входного потока?
   a) `fread();`
   b) `kbhit();`
   c) `getch();`
   d) `fgetc()`
   e) `Нет правильного ответа.`

8. Чему равно значение параметра `argc` функции `main` при следующем запуске программы? `C:\> myprogram.exe`
   a) 2
   b) 0
   c) 1
   d) -1
   e) Нет правильного ответа.

Лабораторная работа №11 (дополнительно)
Консольное приложение

Данная лабораторная работа является необязательной для выполнения и оценивается дополнительными баллами.

Необходимо разработать консольное приложение. Взаимодействие с пользователем осуществляется посредством клавиатуры и пользовательского интерфейса, реализованного с использованием ASCII-символов. Например, пользовательский интерфейс для консольного приложения «Калькулятор» может быть реализован, как показано на рисунке 11.1.

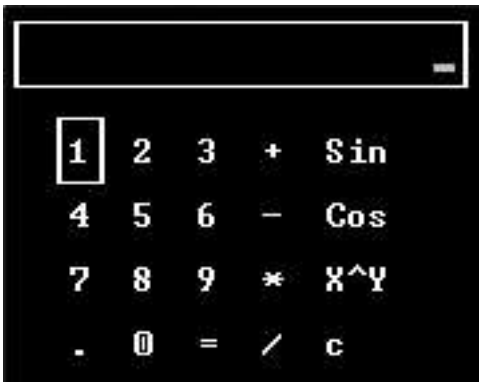


Рис. 11.1. Пользовательский интерфейс консольного приложения «Калькулятор»

№	Задание
1	Простой текстовый редактор
2	Калькулятор
3	Игра «Змейка»
4	Игра «Лабиринт»
5	Игра «Арканоид»
6	Игра «Крестики-Нолики»
7	Игра «Пятнашки»
8	Игра «Шашки»
9	Игра «Морской бой»
10	Нахождение минимального пути в лабиринте.

Также вы можете предложить свою тему для разработки!!!

Ответы на вопросы

№	Ответы	
1	1 – b 2 – d 3 – a 4 – d 5 – b	6 – c 7 – a, e 8 – c 9 – a 10 – c
2	1 – b 2 – c 3 – c 4 – e	5 – e 6 – c, e 7 – c, e, f 8 – a
3	1 - e 2 - c 3 - b 4 - a 5 - d	6 - e 7 - d 8 - c 9 - d 10 - f
5	1 – b 2 – c 3 – e 4 – b	5 – b 6 – e 7 – d 8 – c
6	1 – a 2 – c 3 – c 4 – d	5 – c 6 – b 7 – b 8 – b
7	1 – c 2 – d 3 – b 4 – a, c	5 – b 6 – c 7 – a, c 8 – c, e
8	1 – a 2 - c 3 – d 4 – d	5 – b 6 – c 7 – a 8 – c
10	1 – b 2 – d 3 – a, b 4 – b	5 – d 6 – c 7 – d 8 – c