

# Задания на самостоятельное выполнение

## 1 Задание №0

Разработать программу, обеспечивающую вывод на экран следующих сообщений:

```
Hello, world!  
Здравствуй, мир!
```

## 2 Задание №1

Разработать программу, обеспечивающую перевод температуры по Фаренгейту в температуру по Цельсию.

Формула перевода температуры по Фаренгейту в температуру по Цельсию:

$$C^o = \frac{5}{9} * (F^o - 32) \quad (1)$$

Программа должна обеспечивать:

- выдачу следующего сообщения для ввода значения температуры в градусах Фаренгейта:

Enter Farenheit:

(для выдачи сообщений использовать функцию **printf(...)**);

- ввод значения температуры в градусах Фаренгейта в виде вещественного числа (для ввода значения использовать функцию **scanf(...)**);
- выдачу следующего сообщения с вычисленным значением температуры в градусах Цельсия:

Celsii: X

где X — вычисленное значение температуры в градусах Цельсия.

Каждое выдаваемое сообщение должно начинаться с новой строки.

### 3 Задание №2

Используя результаты разработанной в рамках задания 1 программы, реализовать программу, обеспечивающую выдачу таблицы соответствия температур в градусах Цельсия температурам в градусах Фаренгейта в диапазоне значений от  $0F^o$  до  $150F^o$  с шагом  $10F^o$ .

Таблица должна состоять из двух столбцов. Каждый столбец должен иметь ширину 12 символов. В первом столбце выводится значение  $F^o$ , во втором — соответствующее значение  $C^o$ . Столбцы должны быть выровнены по вертикали. Значения выводить с точностью до двух знаков после десятичной точки.

Пример таблицы:

|       |       |
|-------|-------|
| 0.00  | 17.77 |
| 10.00 | 12.22 |
| ..... |       |

### 4 Задание №3

Разработать программу, обеспечивающую выдачу таблицы соответствия температур в градусах Цельсия температурам в градусах Фаренгейта в диапазоне значений от  $0F^o$  до  $150F^o$  с шагом  $10F^o$ .

Таблица должна состоять из двух столбцов. В первом столбце выводится значение  $F^o$ , во втором — соответствующее значение  $C^o$ . Столбцы должны быть выровнены по вертикали. Значения выводить с точностью до двух знаков после десятичной точки.

Пример таблицы (знак `␣` означает знак пробела):

```
+-----+-----+
|␣␣␣␣␣F␣␣␣␣␣|␣␣␣␣␣C␣␣␣␣␣␣|
+-----+-----+
|␣␣␣␣␣␣0.00|␣␣␣␣␣␣17.77|
|␣␣␣␣␣␣10.00|␣␣␣␣␣␣12.22|
|␣␣␣␣␣␣.....|␣␣␣␣␣␣.....|
+-----+-----+
```

Реализовать три варианта программы:

1. с использованием цикла **while**;
2. с использованием цикла **do-while**;
3. с использованием цикла **for**.

Варианты должны быть реализованы в одном и том же исходном модуле.

Рекомендация: для вывода результата использовать функцию стандартной библиотеки `printf()`.

## 5 Задание №4

Разработать программу, обеспечивающую получение последовательности  $x_1, x_2, \dots, x_{20}$ , образованной по следующему закону:

$$x_1 = 0; x_2 = \frac{5}{8}; x_i = \frac{x_{i-1}}{2} + \frac{3}{4}x_{i-2}, i = 3, 4, \dots, 20$$

Результат работы программы должен быть представлен в форме таблицы следующего вида (знак `␣` означает знак пробела):

```
x01␣=␣␣␣␣␣␣0.000
x02␣=␣␣␣␣␣␣0.625
.....
x20␣=␣.....
```

Столбцы таблицы должны быть выровнены по вертикали. Значения выводить с точностью до трёх знаков после десятичной точки.

Реализовать три варианта программы:

1. с использованием цикла **while**;
2. с использованием цикла **do-while**;
3. с использованием цикла **for**.

Варианты должны быть реализованы в одном и том же исходном модуле.

Рекомендация: для вывода результата использовать функцию стандартной библиотеки `printf()`.

## 6 Задание №5

Разработать программу, обеспечивающую ввод с экрана натурального значения  $n$ , последовательности целых чисел  $a_1, a_2, \dots, a_n$  и, далее, вычисление суммы положительных и количества отрицательных членов последовательности  $a_1, a_2, \dots, a_n$ .

Сценарий работы программы должен выглядеть следующим образом (знак `␣` означает знак пробела):

```
SOURCE␣DATA:
Enter␣natural␣value:
n␣=␣...

Enter␣sequence:
a01␣=␣...
...

RESULT:
Sum␣of␣positive␣elements␣␣␣:␣...
Number␣of␣negative␣elements:␣...
```

Реализовать три варианта программы:

1. с использованием цикла **while**;
2. с использованием цикла **do-while**;
3. с использованием цикла **for**.

Варианты должны быть реализованы в одном и том же исходном модуле.  
Рекомендации:

- для ввода данных использовать функцию стандартной библиотеки `scanf()`;
- для вывода данных использовать функцию стандартной библиотеки `printf()`.

## 7 Задание №6

Разработать программу, обеспечивающую побайтовую печать значения целой переменной.

Дано:

```
int a = 0x01020304;
```

Получить:

- целое значение первого байта памяти, занимаемой переменной `a`;
- целое значение второго байта памяти, занимаемой переменной `a`;
- целое значение третьего байта памяти, занимаемой переменной `a`;
- целое значение четвёртого байта памяти, занимаемой переменной `a`.

Формат вывода:

Byte\_1: □...

Byte\_2: □...

Byte\_3: □...

Byte\_4: □...

Значения вывести в шестнадцатеричном формате в виде `NN`, где `N` — шестнадцатеричная цифра. То есть выводимый результат должен занимать ровно две позиции. Если значение результата представляется одной позицией, то должен быть напечатан ведущий 0. Например, значение 9 должно быть напечатано в виде 09.

Для вывода значений в шестнадцатеричном виде используется спецификация формата `%x`.

Получить десятичное значение шестнадцатеричного числа `0x01020304`.

Дополнить программу следующими действиями::

1. `int a = ...;`  
где вместо многоточия должно быть подставлено полученное десятичное число.
2. Получить:
  - целое значение первого байта памяти, занимаемой переменной `a`;
  - целое значение второго байта памяти, занимаемой переменной `a`;
  - целое значение третьего байта памяти, занимаемой переменной `a`;
  - целое значение четвёртого байта памяти, занимаемой переменной `a`.

Выполнить программу для данного случая.  
Ответить на вопрос: изменился ли результат?

## 8 Задание №7

Разработать и реализовать функцию, обеспечивающую сортировку элементов целочисленного массива. Спецификация функции:

```
int mysort(int array[], int n);
```

Здесь:

- `array` — имя массива для сортировки;
- `n` — количество элементов массива.

В случае успешного выполнения функция `mysort()` должна возвращать значение 0, иначе — значение 1. Например, если значение параметра `n` меньше или равно 0, то функция должна вернуть в качестве результата значение 1.

Разработать и реализовать функцию `main`, обеспечивающую объявление и заполнение целочисленного массива `srcarray` значениями, а также обеспечивающую вывод отсортированного массива. Тестирование проводить на массиве с количеством элементов не меньше 10 и не больше 99.

Вывод результата должен быть представлен в следующем виде (пример для массива из 12 элементов):

```
SORTED_ARRAY:  
srcarray[01]_=_...  
srcarray[02]_=_...  
...  
srcarray[12]_=_...
```

Выбрать метод сортировки по собственному усмотрению.  
Выполнить следующую требования:

- функция `main` должна быть реализована в файле с именем `test07.c`;
- функция `mysort` должна быть реализована в файле с именем `mysort.c`;
- описание функции `mysort` должно быть реализовано в файле с именем `mysort.h`;
- в файле `test07.c` в соответствующем месте должна быть строка `#include "mysort.h"`.

Рекомендуемая литература для выбора алгоритма сортировки:

- Никлас Вирт “Алгоритмы и структуры данных”;
- Дональд Кнут “Искусство программирования на ЭВМ”, том 3 “Сортировка и поиск”.

## 9 Задание №8

1. Разработать и реализовать функции, обеспечивающие:
  - определение длины строки символов;
  - лексикографическое сравнение двух строк символов;
  - обмен значениями двух строк символов.
2. Разработать и реализовать функцию `main()`, обеспечивающую тестирование работы функций обработки строк символов, указанных выше.

### 9.1 Определение длины строки символов

Спецификация функции:

```
int stringlength(char *string);
```

Функция возвращает в качестве результата длину в байтах строки `string`.

### 9.2 Лексикографическое сравнение строк символов

Спецификация функции:

```
int stringcompare(char *string01, char *string02);
```

Функция возвращает в качестве результата одно из следующих значений:

- 0 — если строки `string01` и `string02` лексикографически равны;
- -1 — если строка `string01` лексикографически меньше строки `string02`;
- 1 — если строка `string01` лексикографически больше строки `string02`.

Понятие лексикографического порядка означает, что строка `string01` меньше строки `string02`, если первые  $m$  символов строк совпадают, а символ  $m + 1$  строки `string01` меньше символа  $m + 1$  строки `string02`.

### 9.3 Обмен значениями двух строк символов

Спецификация функции:

```
int stringswap(char *string01, char *string02);
```

Функция обеспечивает обмен содержимого строк `string01` и `string02`.

Функция возвращает в качестве результата значение 0.

### Требования к реализации функции `main()`

Функция `main()` должна обеспечить возможность ввода исходных данных и выдачу результатов в соответствии со следующим сценарием:

Enter\_string\_for\_stringlength:

stringlength\_result:

\*\*\*

Enter\_string01\_for\_stringcompare:

Enter\_string02\_for\_stringcompare:

stringcompare\_result:

\*\*\*

Enter\_string01\_for\_stringswap:

Enter\_string02\_for\_stringswap:

string01\_after\_stringswap:

string02\_after\_stringswap:

### 9.4 Требования к выполнению задания

1. Вводимое количество символов для отдельной строки не будет превышать значения 80 (без учёта завершающего нуль-символа).
2. Функции обработки строк символов должны быть реализованы в файле с именем `mystring.c`.
3. Описания реализованных функций обработки строк символов должны быть размещены в файле `mystring.h`.
4. Реализация функции `main()` должна быть размещена в файле `test08.c`.

5. При реализации функций обработки строк символов (`strlen()`, `strcmp()`, `strcpy()`) **ЗАПРЕЩАЕТСЯ ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ СТАНДАРТНОЙ БИБЛИОТЕКИ ФУНКЦИЙ СИСТЕМЫ ПРОГРАММИРОВАНИЯ C.**

## 9.5 Используемые функции ввода/вывода

Для ввода строк символов с экрана можно использовать одну из следующих двух функций стандартной библиотеки системы программирования C:

- `gets()` — считывание строки символов с экрана до перехода на новую строку;
- `scanf()` — форматный ввод; для ввода строки символов использовать спецификацию формата `%s`.

Для вывода строк символов на экран можно использовать одну из следующих двух функций стандартной библиотеки системы программирования C:

- `puts()` — вывод строки символов на экран;
- `printf()` — форматный вывод; для вывода строки символов использовать спецификацию формата `%s`.

## 10 Задание №9

Разработать и реализовать функцию, обеспечивающую сортировку в лексикографическом порядке строк символов.

Спецификация функции:

```
int mysortstr01(char array[][81], int n);
```

Здесь:

- `array` — имя массива строк символов для сортировки;
- `n` — количество элементов массива.

Разработать и реализовать функцию `main`, обеспечивающую объявление и заполнение массива строк символов значениями, а также обеспечивающую вывод отсортированного массива.

Тестирование проводить на массиве с количеством строк не меньше 10.

Вывод результата должен быть представлен в следующем виде (пример):

```
SORTED_STRINGS:
string[01]=...
string[02]=...
...
```



Здесь:

- `string[01]` — первый элемент массива (элемент с индексом 0);
- `string[02]` — второй элемент массива (элемент с индексом 1);
- и т.д.

Выбрать метод сортировки по собственному усмотрению. Допускается использовать метод сортировки, реализованный в рамках задания №7.

Выполнить следующие требования:

- длина отдельной строки символов не должна превышать значения 80 (без учёта завершающего нуль-символа);
- при реализации функций **ЗАПРЕЩАЕТСЯ ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ СТАНДАРТНОЙ БИБЛИОТЕКИ СИСТЕМЫ ПРОГРАММИРОВАНИЯ C**. Использование функций стандартной библиотеки разрешается только при реализации функции `main()`;
- при реализации функции `mysortstr01` использовать функции работы со строками символов, реализованные в рамках задания №8 и, соответственно, при сборке программы тестирования, необходимо использовать уже существующий файл `mystring.c` .
- функция `main` должна быть реализована в файле с именем `test09.c`;
- функция `mysortstr01` должна быть реализована в файле с именем `mysortstr.c`;
- описание функции `mysortstr01` должно быть реализовано в файле с именем `mysortstr.h`;
- в качестве результата выполнения работы должны быть предоставлены следующие файлы:

- `mysortstr.c` ;
- `mysortstr.h` ;
- `mystring.c` ;
- `mystring.h` ;
- `test09.c` .

Рекомендуемая литература для выбора алгоритма сортировки:

- Никлас Вирт “Алгоритмы и структуры данных”;
- Дональд Кнут “Искусство программирования на ЭВМ”, том 3 “Сортировка и поиск”.

## 11 Задание №10

Разработать и реализовать функцию, обеспечивающую сортировку в лексикографическом порядке строк символов.

Спецификация функции:

```
int mysortstr02(char* array[], int n);
```

Здесь:

- **array** — имя массива строк символов для сортировки (МАССИВ УКАЗАТЕЛЕЙ НА СТРОКИ СИМВОЛОВ);
- **n** — количество элементов массива.

Разработать и реализовать функцию **main**, обеспечивающую объявление и заполнение массива строк символов значениями, а также обеспечивающую вывод отсортированного массива.

Тестирование проводить на массиве с количеством строк не меньше 10 и не больше 25.

Вывод результата должен быть представлен в следующем виде:

```
SORTED_STRINGS:  
string[01]=...  
string[02]=...  
...
```

Выбрать метод сортировки по собственному усмотрению. Допускается использовать метод сортировки, реализованный в рамках задания №7.

Выполнить следующие требования:

- при реализации функций **ЗАПРЕЩАЕТСЯ ИСПОЛЬЗОВАНИЕ ФУНКЦИЙ СТАНДАРТНОЙ БИБЛИОТЕКИ СИСТЕМЫ ПРОГРАММИРОВАНИЯ C**. Использование функций стандартной библиотеки разрешается только при реализации функции **main()**;
- при реализации функции **mysortstr02** использовать функции работы со строками символов, реализованные в рамках задания №8 и, соответственно, при сборке программы тестирования, необходимо использовать уже существующий файл **mystring.c** ;
- функция **main** должна быть реализована в файле с именем **test10.c**;
- реализация функции **mysortstr02** должна быть добавлена в уже существующий файл с именем **mysortstr.c**;
- описание функции **mysortstr02** должно быть добавлено в уже существующий файл с именем **mysortstr.h**;

- в качестве результата выполнения работы должны быть предоставлены следующие файлы:

```

- mysortstr.c ;
- mysortstr.h ;
- mystring.c ;
- mystring.h;
- test10.c .

```

Рекомендуемая литература для выбора алгоритма сортировки:

- Никлас Вирт “Алгоритмы и структуры данных”;
- Дональд Кнут “Искусство программирования на ЭВМ”, том 3 “Сортировка и поиск”.

## 12 Задание №11

Разработать и реализовать программу, обеспечивающую

- ввод последовательности строк символов;
- сортировку введённой последовательности символов в лексикографическом порядке.

Признаком завершения ввода последовательности строк символов является ввод пустой строки (первый символ введённой строки является нуль-символом) и, следовательно, количество вводимых строк символов заранее не может быть определено. С целью выполнения данного требования использовать функцию `realloc(...)` стандартной библиотеки функций системы программирования C.

Также отсутствует ограничение на количество символов вводимой строки. Признаком завершения ввода строки символов является ввод символа, соответствующего символу перехода на новую строку (символ `'\n'`). С целью выполнения данного требования, разработать и реализовать функцию, обеспечивающую ввод с экрана строки символов произвольной длины. Спецификация данной функции:

```
char* mygetstring();
```

Функция возвращает в качестве результата адрес области памяти, в которой хранится введённая строка символов. Область памяти под хранение вводимой строки символов должна выделяться динамически. Если память для хранения вводимой строки символов не может быть выделена, то функция возвращает в качестве результата значение `NULL`.

Для выполнения сортировки введённой последовательности строк символов использовать функцию сортировки `mysortstr02(...)`, реализованную в рамках задания №10.

После вывода отсортированной последовательности введённых строк символов на экран, освободить динамическую память, выделенную в процессе работы программы.

При вводе последовательности строк символов не выводить на экран никаких сообщений. Ввод последовательности строк символов должен выполняться сразу после запуска программы на выполнение.

Результат выполнения программы должен быть выведен на экран и содержать исходную и отсортированную последовательности строк символов:

SOURCE\_STRINGS:

...

SORTED\_STRINGS:

...

Выполнить следующие требования:

- при реализации программы использовать, помимо прочих, следующие функции стандартной библиотеки системы программирования:

- `getchar()` — функция ввода символа с экрана; описание функции содержится в файле `stdio.h` ;
- `realloc(...)` — функция динамического перераспределения памяти; описание функции содержится в файле `stdlib.h` .

- в качестве результата выполнения работы должны быть предоставлены следующие файлы:

- `mystring.c` — файл должен содержать реализации всех разработанных ранее функций, а именно:

```
* strlen(...);  
* strcmp(...);  
* strswap(...);  
* mysortstr01(...);  
* mysortstr02(...);
```

Также, в данном файле должна быть представлена реализация функции `mygetstring(...)` .

- `mystring.h` — файл должен содержать описание всех функций, реализация которых содержится в файле `mystring.c` .
- `test11.c` — файл с реализацией функции `main`, обеспечивающей работу программы и вывод результатов.

Рекомендуемая литература для выбора алгоритма сортировки:

- Никлас Вирт “Алгоритмы и структуры данных”;

- Дональд Кнут “Искусство программирования на ЭВМ”, том 3 “Сортировка и поиск”.

Файлы с документацией по функциям стандартной библиотеки системы программирования gcc (на английском языке):

- `libc.html`;
- `libc.pdf`.

Ссылка на источник документации по функциям стандартной библиотеки системы программирования gcc:

<https://www.gnu.org/software/libc/manual/>

## 13 Задание №12

Разработать и реализовать программу, обеспечивающую

- ввод последовательности строк символов;
- сортировку введённой последовательности символов в лексикографическом порядке;
- задание при запуске программы следующих параметров вывода результата сортировки:
  - a — вывод результата сортировки по возрастанию;
  - d — вывод результата сортировки по убыванию;
  - u — вывод только уникальных записей.

При разработке программы использовать результаты, полученные при выполнении задания №11.

Для задания при запуске программы параметров вывода результата использовать аргументы функции `main()`:

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

Если, при запуске программы, не заданы параметры, то по умолчанию принимается, что задан параметр `-a`.

Если, при запуске программы, задан один единственный параметр `-u`, то это равносильно заданию пары параметров `-a` и `-u`.

Допустимые параметры могут располагаться в любом порядке. Помимо этого, допускается задание параметров в следующем виде:

- au или -ua — соответствует заданию пары параметров `-a` и `-u`;
- du или -ud — соответствует заданию пары параметров `-d` и `-u`.

Если, при запуске программы, заданы ошибочные и/или противоречивые параметры (например, одновременно заданы параметры `-a` и `-d`), то на экран выводятся следующие предупреждения:

Usage:

```
mysort [options]
  options:
    -a --- sort ascending
    -d --- sort descending;
    -u --- unique sorting.
```

и программа завершает выполнение.

В остальном следовать требованиям задания №11.

В качестве результата должны быть предоставлены следующие файлы:

- `mystring.c`;
- `mystring.h`;
- `test12.c` — файл с реализацией функции `main`.

## 14 Задание №13

Разработать и реализовать программный модуль, обеспечивающий работу со стеком элементов. Тип элементов стека задаётся при создании стека. В отдельном стеке элементы имеют один и тот же тип данных.

При выполнении данного задания необходимо воспользоваться функциями динамического распределения памяти, такими как `malloc()`, `calloc()`, `free()`.

### 14.1 Описание структуры данных

```
struct stack_element{
    void *element;
    struct stack_element *prev;
};

struct _stack{
    int element_size;
    struct stack_element *top;
    void *buffer;
};

typedef struct _stack *stack;
```

Структура `struct stack_element` описывает структуру данных отдельного элемента стека. Описание полей данной структуры:

- `void *element` — указатель на область памяти, в которой хранится значение элемента стека; данная область памяти должна выделяться динамически при выполнении операции помещения элемента в стек;
- `struct stack_element *prev` — указатель на предыдущий элемент стека; если предыдущий элемент отсутствует, то данный указатель имеет значение `NULL`.

Структура `struct _stack` описывает стек в целом. Описание полей данной структуры:

- `int element_size` — размер в байтах области памяти, выделяемой под значение элемента стека;
- `struct stack_element *top` — указатель на вершину стека; если в стеке нет элементов, то значение этого указателя равно `NULL`;
- `void *buffer` — указатель на область памяти, в которую помещается значение выталкиваемого элемента; необходимость наличия такой области памяти связано с тем, что операция выталкивания элемента из стека должна не только возвращать значение этого элемента, но и освобождать память, динамически выделенную под выталкиваемый элемент; следовательно, размер динамически выделяемой памяти, на которую будет указывать поле `buffer` должен быть равен значению поля `element_size`; указанная область памяти должна выделяться динамически один раз при выполнении операции создания стека.

Операция `typedef struct _stack *stack` обеспечивает возможность использовать имя `stack`, как имя типа для “стековых” переменных.

## 14.2 Описание операций работы со стеком

Для работы со стеком должны быть реализованы следующие операции:

- создать стек;
- поместить элемент в стек;
- вытолкнуть элемент из стека;
- удалить стек;

### 14.2.1 Создать стек

```
stack create_stack(int _element_size);
```

Параметры и возвращаемые значения:

- `_element_size` — размер в байтах значений элементов стека;
- функция возвращает значение адреса динамически выделенной памяти, если не было ошибок при создании стека. Иначе, функция возвращает значение `NULL`.

#### 14.2.2 Поместить элемент в стек

```
int push(stack _stack, void *_element);
```

Параметры и возвращаемые значения:

- `_stack` — указатель на стек;
- `void *_element` — указатель на область памяти, в которой хранится помещаемое в стек значение;
- функция возвращает значение `-1` в случае ошибки и `0` в случае нормального завершения.

#### 14.2.3 Вытолкнуть элемента из стека

```
void * pop(stack _stack);
```

Параметры и возвращаемые значения:

- `_stack` — “стековая” переменная;
- функция возвращает значение поля `buffer`. Если стек пуст, то функция возвращает значение `NULL`.

Таким образом, данная функция должна обеспечить сохранение выталкиваемого из стека значения по адресу, содержащемуся в поле `buffer`, и освободить память, выделенную ранее под выталкиваемый элемент.

#### 14.2.4 Удалить стек

```
void delete_stack(stack _stack);
```

Параметры и возвращаемые значения:

- `_stack` — удаляемая “стековая” переменная;
- функция освобождает динамически выделенную память.



#### 14.2.5 Вывести содержимое стека

```
void print_stack(stack _stack,  
                void (*print_head)(),  
                void (*print_body)(void *_element),  
                void (*print_foot)());
```

Параметры и возвращаемые значения:

- `_stack` — “стековая” переменная, содержимое которой должно быть выведено;
- `print_head` — указатель на функцию, обеспечивающую вывод “шапки”;
- `print_body` — указатель на функцию, обеспечивающую вывод значения элемента стека;
- `print_foot` — указатель на функцию, обеспечивающую вывод “подвала”.

### 14.3 Требования к выполнению работы

- Описания структуры и функций должны быть помещены в файл `stack.h`.
- Реализация функций должна быть представлена в файле `stack.c`.
- Тестовая программа должна быть представлена в файле `test13.c`.
- Реализация функций, передаваемых в качестве параметров в функцию `print_stack()`, должна быть приведена в файле `test.c`.
- Тестовая программа должна обеспечить проверку всех возможных ситуаций работы функций.