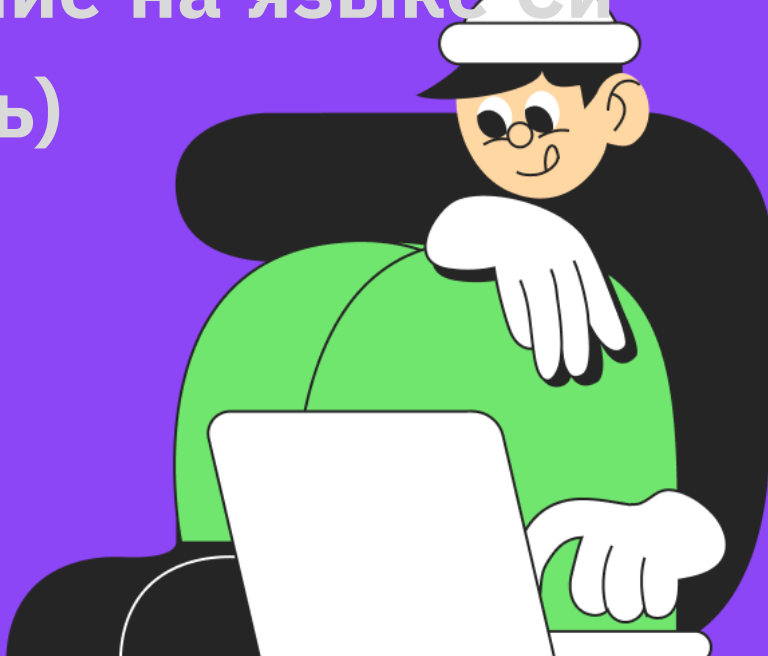
GeekBrains

Урок 8
Строки
Многомерные массивы

Программирование на языке Си (базовый уровень)



Оглавление

Введение

На предыдущей лекции вы узнали:

- Что такое указатель
- Как получить адрес переменной и значение по адресу

- Как передать аргумент по указателю в функцию
- Что такое адресная арифметика
- Что такое массив и как с ним работать

На этой лекции вы найдете ответы на такие вопросы как / узнаете:

- Что такое строки?
- Преобразование строки в массив байт
- Функции библиотек stdlib.h и string.h
- Массивы указателей, многомерные и VLA
- Сортировка qsort и чтение деклараторов
- Вычисление корней квадратного уравнения (массивы и строки)

Термины, используемые в лекции

Программирование – это управление компьютером для решения различных задач связанных с обработкой информации

Строки

Строка в языке Си — это массив из символов типа char, в конце символ с кодом 0(\0).

Все программы и библиотечные функции работающие со строками основаны на том, что в конце строки обязательно есть символ 0.

Особенности строк:

- Все строки состоят из символов с кодом отличным от 0
- Длина строки не содержится в самой строке
- Нет никаких ограничений на длину строки

Примеры объявления строк

```
char s[10];// Строка из 9 значимых символов
char  s[N];// Строка из N-1 значимых символов, в строке всегда есть символ \0
```

Строки также как и массивы можно объявлять с инициализацией

```
char st[] = "hello"; // st[0]='h' st[1]='e' st[2]='l' st[3]='l' st[4]='o' st[5]='\0'
printf("%lu", sizeof(st)); // 6
```

В этой строке 5 значащих символа и 6-ой символ с кодом 0. Размер занимаемой памяти - 6 байт.

```
char st[10] = "hello"; // st[0]='h' st[1]='e' st[2]='l' st[3]='l' st[4]='o' st[5]='\0'
printf("%lu", sizeof(st)); // 10
```

В этой строке 5 значащих символа и 6-ой символ с кодом 0. Размер занимаемой памяти - 10 байт.

Ввод и вывод строк

Для ввода и вывод строк можно также использовать функции printf и scanf, указав соответствующие спецификаторы в параметре форматной строки.

```
char s[10];
scanf("%s",s); // считать строку до первого пробельного символа или \n
printf("%s",s); // напечатать строку
```

Последовательно считывает в строку s символы вводимой строки и добавляет в конец строки символ '\0'. Ввод выполняется до первого символа пробел, символа перевода на следующую строку и т.п. Необходимо позаботиться о размере массива, куда считывается строка, сама функция никак не контролирует это. Если строка больше, чем выделенное под нее место, то произойдет аварийное завершение программы. Можно ограничить размер считываемой строки.

```
char s[10];
scanf("%9s",s); // считать строку не более 9 символов
```

🔥**Внимание!** При выводе строки учитывается символ ‘\0’

Рассмотрим пример посимвольного ввода строки с помощью функции getchar().

```
    char s[10], c;
    int i=0;
    while( (c=getchar())!='\n' )
        s[i++]=c;
    s[i]='\0';
```

Обратите внимание на скобки внутри while. Считываем строку до первого символа “перенос строки” и заносим все в массив. Необходимо поставить признак конца строки после считывания. Напечатать строку можно также с помощью функции посимвольного вывода putchar().

```
   while( s[i] ) // s[i] != 0
        putchar(s[i++]);
```

Множество сканирования scanf

Функция scanf позволяет использовать некоторое подобие регулярного выражения для задания шаблона считываемой строки. В форматной строке можно задать допустимые символы - **множество сканирования**. Можно определить символы, которые будут считываться и присваиваться элементам соответствующего символьного массива. Например:

`char s[100];`
`scanf("%[a-z]",s);// считать стр буквы`
`printf("%s\n",s);`
	`helloWORLD`
	
	`hello`

`scanf("%[0-9]",s);// считать цифры`
 	
`printf("%s\n",s);`
	`123WORLD`
	
	`123`

`scanf("%[^\n]",s);// все кроме \n`
 	
`printf("%s\n",s);`
	`Hello world`
	
	`Hello world`

```
char s1[100], s2[100];
scanf("%s%s",s1,s2);
printf("s1 = %s s2 = %s\n",s1,s2);
```

```
Hello world

s1 = Hello s2 = world
```

```
char s1[100], s2[100];
scanf("%[0-9]=%[a-z]", s1, s2);
printf("s1 = %s s2 = %s\n", s1, s2);
```

```
123 = hello

s1 = 123 s2 = hello
```

В данном примере ожидается строка начинающаяся с цифр, потом идет символ ‘=’, а затем маленькие английские буквы.

Функция scanf возвращает количество успешно распознанных спецификаторов в качестве результата или -1 если на вход передан конец строки или произошла ошибка.

```
char s1[100], s2[100];
int r;
r=scanf("%[0-9]=%[a-z]",s1,s2);
printf("r = %d\n",r);
```

```


123=hello
r = 2
```

```
r=scanf("%[0-9]=%[a-z]",s1,s2);
printf("r = %d\n",r);
```

```
123=123
r = 1
```

```
r=scanf("%[0-9]=%[a-z]",s1,s2);
printf("r = %d\n",r);
```

```
hello=123
r = 0
```

```
r=scanf("%[0-9]=%[a-z]",s1,s2);
printf("r = %d\n",r);
```

```
\EOF
r = -1
```

Задачи на строки

Для работы со строками существует стандартная библиотека string.h. Рассмотрим пример функции strlen (const char *cs) - возвращает длину строки.

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

int main(void) {
   char st[10] = "hello";
      printf("Sizeof  =  %lu\n",
sizeof(st));
      printf("Strlen  =  %lu\n",
strlen(st));
   return 0;
}
```

```
Sizeof = 10
Strlen = 6
```

🔥 **Внимание!** Для сравнения строк нельзя использовать операции ==, != и т.п., поскольку при этом происходит сравнение указателей на начало соответствующих строк, а не самих строк.

```c
    char st1[10] = "hello";
    char st2[10] = "hello";
    if(st1 == st2)
        printf("Yes");
    else
        printf("No");
```

```
No
```

1. Реализовать строковую функцию strlen(const char *cs)

```c
#include <stdio.h>
int strlen(const char *src)
{
    int len=0;
    while (*src++) len++;
    return len;
}
int main(int argc, char **argv)
{
char* str={"Hello!"};
    printf("%d\n",strlen(str));
    return 0;
```

```
}
```

2. Реализовать строковую функцию **strcpy(char *dst, const char *src)** Копирования строки src (включая '\0') в строку **dst**. Функция возвращает указатель на первый символ строки **dst**.

```c
#include <stdio.h>
char *strcpy (char *dst, char *src)
{
char *ptr = dst;
    while(*dst++=*src++);
    return ptr;
}
int main(int argc, char **argv)
{
char str1[]={"Hello!"};//char* str1 = {"Hello!"};
char str2[]={"World!"};//char* str2={"World!"}
    printf("%s\n",strcpy(str2,str1));
    printf("%s\n",str2);
    return 0;
}
```

3. Реализовать строковую функцию strcmp(const char *cs, const char *ct) Функция сравнивает в лексикографическом порядке строку cs со строкой ct. Если строка cs меньше строки ct, возвращается значение < 0, если строка cs больше строки ct, возвращается значение > 0, в случае равенства строк возвращается значение 0.

```c
#include <stdio.h>
int strcmp(const char *a, const char *b){
    while ( *a && *b && *a == *b )
        ++a, ++b;
    return *a - *b;
}
int main(void){
    char *a = "abcde";
    char *b = "xyz";
    char *c = "abcd";
    char *d = "xyz";
    int res;
```

```
    printf("A = %s\nB = %s\nC = %s\nD = %s\n\n", a, b, c, d);
    printf("A is %s B\n", ( ( res = strcmp(a, b) ) == 0 ) ?
"equal to" : ( res < 0 ) ? "less" : "greater than");
    printf("A is %s C\n", ( ( res = strcmp(a, c) ) == 0 ) ?
"equal to" : ( res < 0 ) ? "less" : "greater than");
    printf("A is %s D\n", ( ( res = strcmp(a, d) ) == 0 ) ?
"equal to" : ( res < 0 ) ? "less" : "greater than");
    printf("B is %s C\n", ( ( res = strcmp(b, c) ) == 0 ) ?
"equal to" : ( res < 0 ) ? "less" : "greater than");
    printf("B is %s D\n", ( ( res = strcmp(b, d) ) == 0 ) ?
"equal to" : ( res < 0 ) ? "less" : "greater than");
    printf("C is %s D\n", ( ( res = strcmp(c, d) ) == 0 ) ?
"equal to" : ( res < 0 ) ? "less" : "greater than");
    return 0;
}
```

4. Посчитать количество слов в тексте, слова разделены одним или несколькими пробелами.

```
int main(void) {
    char s[100];
    int count=0;
    while(scanf("%s",s)==1)
        count++;
          printf("In  this  text  %d
words\n",count);
    return 0;
}
```

```
hello    world.     I love
peace

In this text 5 words
```

Попробуйте решить данную задачу с использованием функции getchar().

5. Реализовать функцию, которая возвращает сумму цифр в переданной ей строке.

   `str_sum_digits(const char *cs)`

Преобразование строки в массив байт

Реализовать функцию, которая преобразует переданную строку в массив байт, возвращает количество байт

`int StrToHex(const char *str,char* Hex)`

```c
#include <stdio.h>
#include <stdint.h>
int strlen(const char *src)
{
    int len=0;
    while (*src++) len++;
    return len;
}
int CharToHex(char c)
{
  int result=-1;
  if(c >= '0' && c <= '9')
  result=c-'0';
  else if(c>='A' && c<='F')
  result=c-'A'+10;
  else if(c>='a' && c<='f')
  result=c-'a'+10;
  return result;
}
```

```c
int StrToHexMas(char*
Str,uint8_t* mas);
int main(int argc, char **argv)
{
    uint8_t arr[10];
    int len = StrToHexMas("AAa a
1 15",arr);
   printf("%s\n","AAa a 1 15");
   printf("%d\n",len);
    for(int i=0;i<len;i++)
        printf("%02x,",arr[i]);
    return 0;
}
```

```c
//данные идут последовательно, не более двух символов
int StrToHexMas(char* Str,uint8_t* mas)
{
   int Result = 0; //полученное число
   int data   = 0; //временная переменная
   int i      = 0; //счетчик символов по строке
   int index  = 0; //счетчик данных в массиве
   int StrLenght = strlen(Str);
   printf("%d\n",StrLenght);
   while(i<StrLenght)//выполняем цикл, пока есть символы в строке
   {
      Result=0;                    //обнуляем число
      data = CharToHex(Str[i++]); //анализируем очередной символ
      if(data>=0)                  //если это значащий символ
      {
         Result = data;
         if(i<StrLenght)    //проверка на выход за границы массива
         {
            data = CharToHex(Str[i++]);//анализируем очередной
символ
            if(data>=0)              //если это данные
            {
               Result *= 16;
               Result += data;
```

```
            }
        }
        mas[index++]=Result;    //кладем число в массив
    }
  }
  return index;
}
```

Библиотека string.h

В заголовочном файле описаны функции для работы с памятью, работы со строками, такие как копирование, объединение, нахождение длины строки, нахождение символа и подстроки
Функции для работы с памятью

Имя	Примечания
memcpy	копирует n байт из области памяти src в dest, которые не должны пересекаться
memmove	копирует n байт из области памяти src в dest, которые в отличие от memcpy могут перекрываться
memchr	возвращает указатель на первое вхождение значения c среди первых n байтов s или NULL, если не найдено
memcmp	сравнивает первые n символов в областях памяти
memset	заполняет область памяти одним байтом z

Функции для работы со строками

Имя	Примечания
***strcpy**	**копирует строку из одного места в другое**
*strncpy	копирует до n байт строки из одного места в другое
strlen	**возвращает длину строки**
*strpbrk	находит первое вхождение любого символа, перечисленного в accept
*strstr	находит первое вхождение строки needle в haystack

strcat	дописывает строку src в конец dest
strncat	дописывает не более n начальных символов строки src (или всю src, если её длина меньше) в конец dest
*strchr	возвращает адрес символа c в строке s, начиная с головы, или NULL, если строка s не содержит символ c
*strrchr	возвращает адрес символа c в строке s, начиная с хвоста, или NULL, если строка s не содержит символ c
strcmp	**лексикографическое сравнение строк (возвращает "0", если строки одинаковые, положительное, если первая строка больше, и отрицательное, если меньше)**
strncmp	лексикографическое сравнение первых n байтов строк

Пример функций **strcat**, **strcpy**, **strncpy** библиотеки **string.h**

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

int main(void)
{
    char destination[30] = "Hello ";
    char source[30] = "GB!!!";
    strcat(destination, source);
    printf("%s\n", destination);    //Hello GB!!!
    strcpy(source,destination);
    printf("%s\n", source);    //Hello GB!!!
    int n = strlen(source); // количество копируемых символов
    strncpy(destination, source, n-1);
//функция strncpy НЕ заканчивает скопированную строку нулевым символом,
//что может привести к переполнению буфера.
//Поэтому после копирования следует вручную устанавливать нулевой символ
```

Пример функций **strstr** библиотеки **string.h**

```
    destination[n-1] = '\0';
    printf("%s %d\n", destination,n);     //Hello GB!! 11
    char substring[14] = "GB!";
    char *substring_ptr = strstr(destination, substring);
    // если подстрока найдена
    if(substring_ptr)
    {
        // вычисляем позицию подстроки в строке
        long position = substring_ptr - destination;
            printf("Substring index: %ld\n", position); // Substring
index: 6
    }
    else    // если подстрока не найдена
    {
        printf("Substring not found\n");
    }
    return 0;
}
```

Заголовочный файл stdlib.h

Имя	Описание	Соответствие стандартам	
Преобразование типов		C89	C99
atof	строка в число двойной точности (double; НЕ float)	Да	Да
atoi	строка в целое число (integer)	Да	Да
atol	строка в длинное целое число (long integer)	Да	Да
atoll	строка в длинное целое число (long long integer)	Нет	Да
strtod	строка в число двойной точности (double)	Да	Да
strtof	строка в число одиночной точности (float)	Нет	Да
strtol	строка в длинное целое число (long integer)	Да	Да
strtold	строка в длинное число двойной точности (long double)	Нет	Да

strtoll	строка в длинное целое число (long long integer)	Нет	Да
strtoul	строка в беззнаковое длинное целое число (unsigned long integer)	Да	Да
strtoull	строка в беззнаковое длинное целое число (unsigned long long integer)	Нет	Да

Функции atoi, atof

```c
#include <stdio.h>   //Для printf()
#include <stdlib.h >  //Для atoi()
int main (void)
{
   char *Str = "652.2345brrt 7";
//Строка для преобразования
   char *ptr=Str;
   //Преобразование строки в число
типа int
   int Num = atoi (Str);
   //Вывод результата
преобразования
   printf ("%d\n",Num);
   float Num1 = atof (Str);
   //Вывод результата
преобразования
   printf ("%f\n",Num1);
   ptr+=13;
   int Num3 = atoi (ptr);
   //Вывод результата
преобразования
   printf ("%d\n",Num3);
   return 0;
}
```

Заголовочный файл stdlib.h функции

Имя	Генерация псевдослучайных последовательностей	C89	C99
rand	генерирует псевдослучайное значение	Да	Да
srand	устанавливает начальное значение генератора псевдослучайных чисел	Да	Да
	Выделение и освобождение памяти		
malloc calloc realloc	выделяет память из кучи	Да	Да
free	освобождает память обратно в кучу	Да	Да
	Сортировка и поиск		

qsort	сортировка массива	Да	Да
	Многобайтовые операции/ широкие символы	Нет	Да
	Контроль процесса выполнения программы	Да	Да
	Математика		Да

Массивы указателей

В таком массиве каждый элемент это ссылка на объект одного типа. Обычно массивы указателей ссылаются на строки. Каждый элемент содержит адрес нулевого символа строки. Очевидное преимущество такого способа хранения - это разная длина хранящихся строк.

```
    const uint8_t *ps[] = {"one", "two", "three", NULL}; // NULL признак конца
    for(uint32_t i=0; ps[i] ;i++)
        printf("%s\n", ps[i]);
```

Многомерные массивы

Можно определять многомерные массивы, при этом массив размерности n рассматривается как одномерный массив, каждый элемент которого представляет собой массив размерности n - 1

Объявляются многомерные массив в общем случае так:

```
тип имя [индекс_1] [индекс_2] ... [индекс_n];
```

Пример:

```
int matr[3][5]; // 3 строки и 5 столбцов
```

По сути, двумерный массив является фактически одномерным массивом, каждый элемент которого в свою очередь также является одномерным массивом. Двумерным он называется потому, что управляется при помощи двух индексов (индекс строки, первый при объявлении, и индекс столбца).

В данном примере массив matr - это массив состоящий из 3 объектов, каждый из которых в свою очередь состоит из 5 элементов.

matr[0][0]	matr[0][1]	matr[0][2]	matr[0][3]	matr[0][4]
matr[1][0]	matr[1][1]	matr[1][2]	matr[1][3]	matr[1][4]
matr[2][0]	matr[2][1]	matr[2][2]	matr[2][3]	matr[2][4]

Можно объявлять массивы размерностью больше 2-ух

```
int box[10][20][30]; // 10 матриц по 20 строк и 30 столбцов
```

Существует два варианта инициализации многомерного массива:

```
int m[4][3] = {{11, 15, 30},
               {10, 20, 31},
               { 5,  8,  0},
               {17, 25, 47}};
```

что полностью эквивалентно варианту:

```
int m[4][3] = {11, 15, 30,
               10, 20, 31,
               5,  8,  0,
               17, 25, 47};
```

поскольку инициализируются все элементы многомерного массива. Если же инициализируются не все элементы, то внутренние фигурные скобки обязательны:

```
int m[4][3] = {{11, 15, 30},
       {10, -2},
       {5},
       {}};
```

при такой записи первый внутренний массив проинициализируется полностью, во втором остается в неопределенном состоянии последний элемент, в третьем подмассиве - два последних элемента останутся без инициализации, хотя память под них будет выделена, как и под четвертый подмассив. В такой записи:

```
int m[4][3] = {11, 15, 30,
       10, -2,
       5};
```

компилятор также не найдёт ошибок, хотя результат будет совсем иным, чем в предыдущей записи, а именно: пятерка проинициализирует третий элемент второго подмассива, а третий и четвертый подмассивы останутся без инициализации

Как правило для обработки массивов используются вложенные циклы. Причем для обработки двумерных массивов - два вложенных цикла, для трехмерных - три цикла и т.д.

```
    int matr[3][5];
    int i,j;
    for(i=0;i<3;i++)
    for(j=0;j<5;j++)
        matr[i][j]=i+j;
```

```
    int matr[3][2] = {
        {1,2},   //строка 0
        {10,-5}, //строка 1
        {0,-17}  //строка 2
    };
```

Можно объявить указатель на двумерный массив, однако при этом теряется информация о размере строки.

```
    int matr[3][2];
    int *pm;
    pm = matr;
    pm[1][1] = 123;  //ТАК
НЕЛЬЗЯ
```

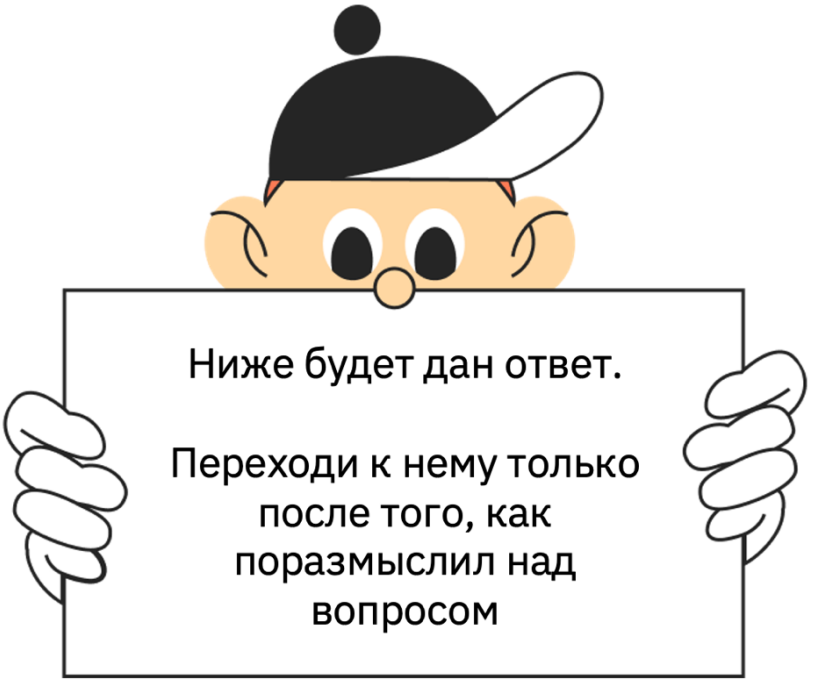
```
    int matr[3][2];
    int *pm;
    pm = matr;
    *(pm + 3) = 123; // matr[1][1]
    pm[3] = 123; // или так
    printf("%d\n",matr[1][1]);
```

А можно объявить указатель на строку матрицы

```
    int matr[3][2];
    int (*pm)[2]; //указатель на строку из 2-ух int
    pm = matr;
    pm[1][1] = 123; //теперь все ок
    printf("%d\n",matr[1][1]); // 123
```

Что будет напечатано?

`int m[3][3] =`
`{`
`{1, 2, 3},`
`{4, 5, 6},`
`{7, 8, 9}`
`};`
`int *p1;`
`int (*p2)[3];`
`p1 = m[1];`
`p2 = m + 1;`
`p1++;`
`p2++;`
`printf ("%d %d\n", *p1, **p2);`
	?



`int m[3][3] =`
`{`
`{1, 2, 3},`
`{4, 5, 6},`
`{7, 8, 9}`
`};`
`int *p1;`
`int (*p2)[3];`
`p1 = m[1];`
`p2 = m + 1;`
`p1++;`
`p2++;`
`printf ("%d %d\n", *p1, **p2);`
	5 7

	m[i]	m [i] [j]		
m	m[0]	1	2	3
	m[1]	4	5	6
	m[2]	7	8	9

VLA массивы

В C99 можно описывать локальные массивы неконстантного размера (Variable-length array) и выделять для них динамическую память. Без использования указателей VLA массивы могут быть использованы как локальные переменные либо параметры функций.

```
uint32_t n;
scanf("%u",&n); //вводим количество элементов
int32_t ar[n]; //создаем VLA массив
```

🔥 **Внимание!** Если массив используется в качестве параметра функции и размерности массива передаются в эту функцию, то указатель должен следовать после описания размеров.

```
// Делайте так
void func(int m, int n, int arr[m][n])
```

```
// ТАК НЕЛЬЗЯ - ОШИБКА
void func(int arr[m][n], int m, int n)
```

Задачи

1. Написать функцию, которая выводит на печать матрицу

```
void print_matrix_1 (int m, int n,
```

```
void print_matrix_2 (int m, int n,
```

```
int *a)
{
    int i, j;
    for (i = 0; i < m; i++){
        for (j = 0; j < n; j++){
            printf ("%d ", a[i * n +
j]);
        }
        printf ("\n");
    }
}
```

```
int a[m][n])
{
    int i, j;
    for (i = 0; i < m; i++){
        for (j = 0; j < n; j++){
            printf ("%d ",
a[i][j]);
        }
        printf ("\n");
    }
}
```

2. Написать функцию, которая вычисляет след матрицы.

Сортировка qsort

Функция **qsort(base, num , size , (int(*) (const void *, const void *)) comparator)** сортирует массив, на который указывает параметр base, используя quicksort — алгоритм сортировки широкого назначения, разработанный Т. Хоаром. Параметр num задает число элементов массива, параметр size задает размер в байтах каждого элемента.

Функция, на которую указывает параметр compare, сравнивает элементы массива с ключом. Формат функции compare следующий:

int func_name(const void *arg1, const void *arg2)

Она должна возвращать следующие значения:

- ➔ Если arg1 меньше, чем arg2, то возвращается отрицательное целое.
- ➔ Если arg1 равно arg2, то возвращается 0.
- ➔ Если arg1 больше, чем arg2, то возвращается положительное целое.

Массив сортируется по возрастанию таким образом, что наименьший адрес соответствует наименьшему элементу.

```
/* сравнение двух целых V1 */
int comparator (const int *a, const int *b) {
    return *a - *b;
}
```

```
int main(void)
{
….
    qsort(a, SIZE, sizeof (int), (int(*) (const void *, const
void *)) comparator);
….
}
```

```
/* сравнение двух целых V2 */
int comparator (const void *a, const void *b) {
    return *(int *)a - *(int *)b;
}
int main(void)
{
….
    qsort(a, SIZE, sizeof (int), comparator);
….
}
```

Добавим typedef в версию V1

```
typedef  int(*comparator_type) (const void *, const void *);

/* сравнение двух целых V1 */
int comparator (const int *a, const int *b) {
    return *a - *b;
}
int main(void)
{
….
    qsort(a, SIZE, sizeof (int), (comparator_type) comparator);
….
}
```

Читаем декларатоpы

Сайт для декодирования деклараторов https://cdecl.org/

Мы ранее рассматривали различные конструкции, модифицирующие тип, такие как указатели, массивы, функции.

Декларатор — это синтаксическая конструкция состоящая из комбинации модификаторов типа, таких как указатели, массивы, функции.

<базовый тип**> <**декларатор**> [= <**инициализатор**>];**

Декларатор обычно содержит имя определяемого объекта, но в общем случае может не содержать имя определяемого объекта.

Анонимные деклараторы допускаются в операции приведения типа и при описании формальных параметров в прототипах функций.

char (*(*x[3])())[10]; — на самом деле это значит объявить **x** как массив 3 указателей на функцию, возвращающую указатель на массив из 10 char.

Такая (на первый взгляд «странная») форма определения производных типов на самом деле введена по аналогии с выражениями. Декларатор можно рассматривать как некоторое выражение над типом.

В таком выражении есть три операции:

[] постфиксная — массив из заданного количества элементов

() постфиксная — функция с заданными параметрами

* префиксная — указатель

() группировка членов в выражении

Постфиксные операции имеют самый высокий приоритет и читаются слева направо от определяемого имени. Префиксная операция имеет более низкий приоритет и читается справа налево. Скобки могут использоваться для изменения порядка чтения.

Таким образом, декларатор читается, начиная от имени определяемого объекта следуя правилам приоритетов операций. Имя определяемого объекта — это первое имя после базового типа.

```
char *( *(*var)() )[10];
 ^  ^ ^^^  ^   ^
 7  6 4 2 1  3   5
```

В этом примере шаги пронумерованы по порядку и интерпретируются следующим образом:

1. Идентификатор var объявлен как
2. указатель на
3. функцию, возвращающую
4. указатель на
5. массив из 10 элементов, которые являются
6. указателями на
7. значения типа char.

```
int a[3][4]; - массив из 3 элементов типа массива из 4 элементов int

char **b; - указатель на указатель на char

char *c[]; - массив из неопределённого количества элементов типа указатель на тип char

int *d[10]; - массив из 10 элементов типа указатель на тип int

int (*a)[10]; - указатель на массив из 10 элементов типа int

int (*a)[5]; - указатель на массив из 5 элементов типа int

int *f(); - функция, возвращающая указатель на int

int (*f)(); - f это указатель на функцию, возвращающую значение int

int *(*f)(); - указатель на функцию, возвращающую указатель на int
```

Вычисление корней квадратного уравнения (массивы и строки)

Перепишем программу в соответствии с парадигмой: разделять ввод, вывод и вычисления.

```
#define COEF_N 3
```

```
#define ROOT_N 2
enum {NO_ROOTS=-1,COMPLEX_ROOTS=0,C=2,B=1,A=0};
//Сделаем вычисление корней квадратного уравнения отдельной функцией.
void InputStr(int size, float coef[]) {}
void Print(int size, float roots[]) {}
int  CalcRoots(float*coef,float* roots) {}

void SquareEquationNew(void)
{
float coef[COEF_N];
float roots[ROOT_N];
    InputStr(COEF_N,coef);
    int rootsNumber = CalcRoots(coef,roots);
    Print(rootsNumber,roots);
}
```

Функция ввода

```
void Input(int size, float coef[])
{
    printf("Вычисление корней квадратного уравнения \\
\"a*x*x+b*x+c=0\"\n");
    coef[A] = InputFloat("Введите a:\n");//1
    coef[B] = InputFloat("Введите b:\n");//18
    coef[C] = InputFloat("Введите c:\n");//32
}
```

Вариант со строками

```
const char* str[]={"Введите a:\n","Введите b:\n","Введите
c:\n",NULL};
void InputStr(int size, float coef[])
{
    printf("Вычисление корней квадратного уравнения \\
\"a*x*x+b*x+c=0\"\n");
    for(int i=0;i<COEF_N;i++)
        coef[i] = InputFloat(str[i]);//1
}
```

Функция вывода

```
void Print(int size, float roots[])
{
    switch(size)
    {
        case 2:
            printf("Корни квадратного уравнения \n");
            printf("X1 = %f \n",roots[0]);
            printf("X2 = %f \n",roots[1]);
        break;
        case 1: printf("Корень линейного уравнения %f\n",roots[0]);
break;
```

```
        case COMPLEX_ROOTS:
            printf("Корни квадратного уравнения комплексные \n");
        break;
        case NO_ROOTS: printf("Корней НЕТ!\n"); break;
        default:       printf("Ошибка количества корней%d!\n",size);
    }
}
```

Вычисление корней квадратного уравнения через массивы

```
int CalcRoots(float*coef,float*
roots)
{
int rootsNumber=NO_ROOTS;
    float b = coef[B]/2;//заменим
на b
    if(coef[A]!=0)
    {
        float d =
b*b-coef[A]*coef[C];
        if(d<0)

rootsNumber=COMPLEX_ROOTS;
        else
        {
            rootsNumber=2;

CalcRealRoots(d,b,coef[A],
roots);
        }
    }
    else
    {
        if(coef[B]!=0)
        {
            roots[0] = -coef[C] /
coef[B];
            rootsNumber=1;
        }
        else
            rootsNumber=NO_ROOTS;
    }
    return rootsNumber;
}
void CalcRealRoots(float sqrD,float
B,float a,float roots[])
{
    float d = sqrtf(sqrD);
    roots[0] = (-B + d)/a; //2
    roots[1] = (-B - d)/a; //16
}
```

Точка входа вычисления корней квадратного уравнения

```
float InputFloat(const char*
message)
{
float number;
    printf("%s",message);
    scanf("%f",&number);
    return number;
}
int main(int argc, char **argv)
{
char Choice;
    setlocale(LC_ALL, "Rus");
    while(1)
    {
        printf("1. Вычисление
```

```
switch(Choice)
        {
            case '1':

SquareEquationNew();
            break;
            case '0':
            case 'q':
            case 'Q':
                return 0;
            break;
            default:
                printf("Непонятный
выбор %x\n",Choice);
            break;
```

```
корней квадратного уравнения\n");
        printf("0. Выход\n");
        printf("Для выход нажмите Q\n");
        Choice = getch();
```

```
        }
    }
    return 0;
}
```

Дополнительные материалы

1. Домашнее задание задачи B1 - B21
2. Оформление программного кода http://www.stolyarov.info/books/pdf/codestyle2.pdf
3. Стандарт разработки программного обеспечения MISRA на языке C http://easyelectronics.ru/files/Book/misra_c_rus.pdf
4. string.h — Википедия
5. C Library - <string.h> - GeeksforGeeks
6. C | Работа со строками

Используемые источники

1. cprogramming.com - учебники по C и C++
2. free-programming-books - ресурс содержащий множество книг и статей по программированию, в том числе по C и C++ (там же можно найти ссылку на распространяемую бесплатно автором html-версию книги Eckel B. «Thinking in C++»)
3. tutorialspoint.com - ещё один ресурс с множеством руководств по изучению различных языков и технологий, в том числе содержит учебники по C
4. Юричев Д. «Заметки о языке программирования Си/Си++» - «для тех, кто хочет освежить свои знания по Си/Си++»
5. Онлайн версия «Си для встраиваемых систем»