

23.09.2024

# Функции. Символьные строки и строковые функции.

*Филиппов Михаил Витальевич*

[m.filippov@g.nsu.ru](mailto:m.filippov@g.nsu.ru)

89232283872

Императивное программирование, 2024-2025

**N** \* Новосибирский  
государственный  
университет  
**\*НАСТОЯЩАЯ НАУКА**

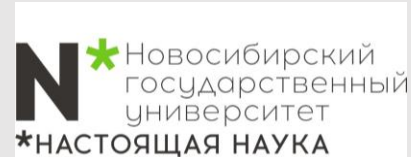


# Давайте познакомимся



## Филиппов Михаил Витальевич

- Окончил магистратуру ФФ НГУ
- Окончил аспирантуру ИТ СО РАН
- Являюсь м.н.с. ИТ СО РАН
- 7+ лет опыт в программировании C/C++



# Адженда

**Функции**

**45 минут**

**Символьные  
строки и  
строковые  
функции**

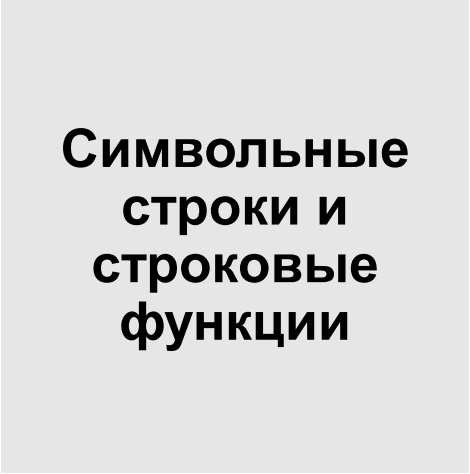
**45 минут**

# Адженда



**Функции**

**45 минут**



**Символьные  
строки и  
строковые  
функции**

**45 минут**

# Что такое функция?

**Функция** — это самодостаточная единица кода программы, спроектированная для выполнения отдельной задачи. Структура функции и способы ее возможного использования определяются синтаксическими правилами.

## ***Почему вы должны использовать функции?***

- Избавляют от необходимости в многократном написании одного и того же кода;
- Функцию можно применять внутри программы там, где она необходима, или же использовать ее в разных программах;
- Функции имеют смысл, т.к. это делает программу более модульной, таким образом улучшая ее читабельность и упрощая внесение изменений либо исправлений

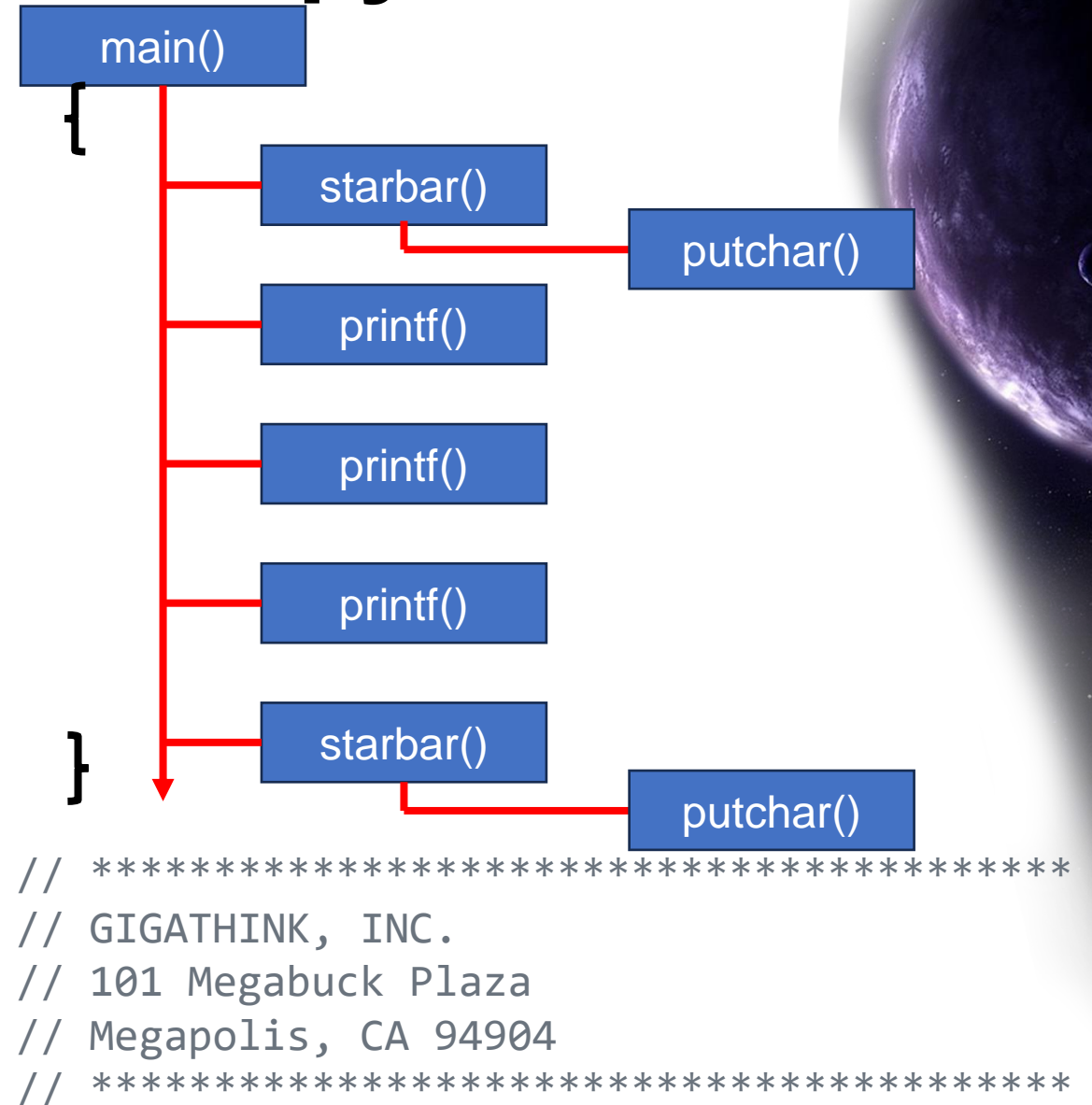
```
int main()
{
    float list[SIZE];
    readlist(list, SIZE); // читает список чисел
    sort(list, SIZE);     // сортирует эти числа
    average(list, SIZE);  // находит среднее значение этих чисел
    bargraph(list, SIZE); // выводит гистограмму
    return 0;
}
```





# Использование простой функции

```
#include <stdio.h>
#define NAME "GIGATHINK, INC."
#define ADDRESS "101 Megabuck Plaza"
#define PLACE "Megapolis, CA 94904"
#define WIDTH 40
void starbar(void); /* прототип функции */
int main(void)
{
    starbar();
    printf("%s\n", NAME);
    printf("%s\n", ADDRESS);
    printf("%s\n", PLACE);
    starbar(); /* использование функции */
    return 0;
}
void starbar(void) /* определение функции */
{
    int count;
    for (count = 1; count <= WIDTH; count++)
        putchar('*');
    putchar('\n');
}
```



# Аргументы функции

```
void show_n_char(char ch, int nurn) /* определение функции show_n_char () */
{
    int count;
    for (count = 1; count <= nurn; count++)
        putchar(ch);
}
int main(void)
{
    int spaces;
    show_n_char('*', WIDTH); /* использование констант в качестве аргументов */
    putchar('\n');
    show_n_char(SPACE, 12); /* использование констант в качестве аргументов */
    printf("%s\n", NAME);
    spaces = (WIDTH - strlen(ADDRESS)) / 2; /* сколько пробелов нужно вывести */
    show_n_char(SPACE, spaces); /* использование переменной в
качестве аргумента */
    printf("%s\n", ADDRESS);
    show_n_char(SPACE, (WIDTH - strlen(PLACE)) / 2);
    printf("%s\n", PLACE);
    show_n_char('*', WIDTH);
    putchar('\n');
    return 0;
}
```

```
// *****
//                GIGATHINK, INC.
//                101 Megabuck Plaza
//                Megapolis, CA 94904
// *****
```

# Аргументы функции

```
#include <stdio.h>
int imin(int, int);
int main(void)
{
    system("chcp 65001");
    int evil1, evil2;
    printf("Введите два целых числа (или q для завершения):\n");
    while (scanf("%d %d", &evil1, &evil2) == 2)
    {
        printf("Меньшим из двух чисел %d и %d является %d.\n",
               evil1, evil2, imin(evil1, evil2));
        printf("Введите два целых числа (или q для завершения):\n");
    }
    printf("Программа завершена.\n");
    return 0;
}

int imin(int n, int m)
{
    if (n < m)
        return n;
    else
        return m;
}
```

```
// Введите два целых числа (или q для завершения):
// 509 333
// Меньшим из двух чисел 509 и 333 является 333.
// Введите два целых числа (или q для завершения):
// -9393 6
// Меньшим из двух чисел -9393 и 6 является -9393.
// Введите два целых числа (или q для завершения):
// q
// Программа завершена.
```



# Типы функций

- Функции должны объявляться с указанием типов.
- Функция с возвращаемым значением должна быть объявлена с тем же типом, что и у возвращаемого значения.
- Функция без возвращаемого значения должна быть объявлена с типом `void`.

До C99, если функции не назначен тип, то такая функция относится к типу `int`.

## **Примеры объявлений:**

```
double klink(int a, int b)
```

```
double sqrt(double);
```

`void print_name();` эквивалентно `void print_name(void);`, однако во втором случае выглядит более наглядно для компилятора

## **Определение и прототип:**

```
int imax(int a, int b) { return a > b ? a : b; }
```

```
int main ()
```

```
{
```

```
    int x, z;
```

```
    z = imax(x , 50);
```

```
}
```



# Прототипы функций

Не правильно!

```
#include <stdio.h>
int imax(); /* объявление в старом стиле */
int main(void)
{
    system("chcp 65001");
    printf("Наибольшим значением из %d и %d
является %d.\n", 3, 5, imax(3));
    printf("Наибольшим значением из %d и %d
является %d.\n", 3, 5, imax(3.0, 5.0));
    return 0;
}
int imax(n, m)
int n, m;
{
    return (n > m ? n : m);
}
// Наибольшим значением из 3 и 5 является 3.
// Наибольшим значением из 3 и 5 является 0.
```

Правильно!

```
#include <stdio.h>
int imax(int, int); /* прототип */
int main(void)
{
    system("chcp 65001");
    // printf("Наибольшим значением из %d и %d
является %d.\n", 3, 5, imax(3));
    printf("Наибольшим значением из %d и %d является
%d.\n", 3, 5, imax(3.0, 5.0));
    return 0;
}
int imax(int n, int m)
{
    return (n > m ? n : m);
}
// main.c:7:68: error: too few arguments to function
'imax'
//      7 |      printf(" и %d .\n", 3, 5, imax(3));
// Наибольшим значением из 3 и 5 является 5.
```

# Рекурсия

**Рекурсия** – процесс вызова функции как саму себя.  
 Рекурсивные решения более элегантны, но менее эффективны, чем решения с циклами.

```
#include <stdio.h>
void up_and_down(int);
int main(void)
{
    up_and_down(1);
    return 0;
}
void up_and_down(int n)
{
    printf("Уровень %d: ячейка n %p\n", n, &n); // 1
    if (n < 4)
        up_and_down(n + 1);
    printf("УРОВЕНЬ %d: ячейка n %p\n", n, &n); // 2
}
```

| Переменные                 | n | n | n | n |
|----------------------------|---|---|---|---|
| После вызова уровня 1      | 1 |   |   |   |
| После вызова уровня 2      | 1 | 2 |   |   |
| После вызова уровня 3      | 1 | 2 | 3 |   |
| После вызова уровня 4      | 1 | 2 | 3 | 4 |
| После возврата из уровня 4 | 1 | 2 | 3 |   |
| После возврата из уровня 3 | 1 | 2 |   |   |
| После возврата из уровня 2 | 1 |   |   |   |
| После возврата из уровня 1 |   |   |   |   |

Уровень 1: ячейка n 00000057291ffc40  
 Уровень 2: ячейка n 00000057291ffc10  
 Уровень 3: ячейка n 00000057291ffbe0  
 Уровень 4: ячейка n 00000057291ffbb0  
 УРОВЕНЬ 4: ячейка n 00000057291ffbb0  
 УРОВЕНЬ 3: ячейка n 00000057291ffbe0  
 УРОВЕНЬ 2: ячейка n 00000057291ffc10  
 УРОВЕНЬ 1: ячейка n 00000057291ffc40

# Рекурсия факториал

```
#include <stdio.h>
long fact(int n) // функция, основанная на цикле
{
    long ans;
    for (ans = 1; n > 1; n--)
        ans *= n;
    return ans;
}
long rfact(int n) // рекурсивная версия
{
    return n > 0 ? n * rfact(n - 1) : 1;
}
int main(void)
{
    int num;
    while (scanf("%d", &num) == 1)
    {
        printf("цикл: факториал %d = %ld\n", num, fact(num));
        printf("рекурсия: факториал %d = %ld\n", num, rfact(num));
    }
    printf("Программа завершена.\n");
    return 0;
}
```

```
// 10
// цикл: факториал 10 = 3628800
// рекурсия: факториал 10 = 3628800
// 5
// цикл: факториал 5 = 120
// рекурсия: факториал 5 = 120
// q
// Программа завершена.
```

# Рекурсия и изменение порядка

```
#include <stdio.h>
void to_binary(unsigned long n) /* рекурсивная функция */
{
    int r;
    r = n % 2;
    if (n >= 2) {to_binary(n / 2);}
    putchar(r == 0 ? '0' : '1');
    return;
}
int main(void)
{
    unsigned long number;
    printf("Введите целое число (q для завершения): \n");
    while (scanf("%lu", &number) == 1)
    {
        printf("Двоичный эквивалент: ");
        to_binary(number);
        putchar('\n');
        printf("Введите целое число (q для завершения):\n");
    }
    printf("Программа завершена.\n");
    return 0;
}
```

```
// Введите целое число (q для завершения):
// 9
// Двоичный эквивалент: 1001
// Введите целое число (q для завершения):
// 255
// Двоичный эквивалент: 11111111
// Введите целое число (q для завершения):
// 1024
// Двоичный эквивалент: 1000000000
// Введите целое число (q для завершения):
// q
// Программа завершена.
```



# Рекурсия преимущества и недостатки

Преимущества:

- ✓ предлагает простейшее решение ряда задач программирования

Недостатки:

- ✗ некоторые рекурсивные алгоритмы могут быстро исчерпать ресурсы памяти компьютера
- ✗ рекурсию трудно документировать и сопровождать

Пример Фибоначчи:

```
unsigned long Fibonacci(unsigned n) {  
    if (n > 2)  
        return Fibonacci(n-1) + Fibonacci(n-2);  
    else  
        return 1;  
}
```

**Не является ли функция main () особенной?**

Да, ее особенность в том, что, когда программа, состоящая из нескольких функций, собирается вместе, выполнение начинается с первого оператора в main.

# Выяснение адресов: операция &

Унарная операция & предоставляет адрес, по которому хранится переменная. Если pooh является именем переменной, то &pooh — адрес этой переменной.

```
#include <stdio.h>
void mikado(int bah) /* определение функции */
{
    int pooh = 10; /* локальная для mikado () */
    printf("Внутри mikado() pooh = %d и &pooh = %p\n", pooh, &pooh);
    printf("Внутри mikado() bah = %d и &bah = %p\n", bah, &bah);
}
int main(void)
{
    int pooh = 2, bah = 5; /* локальные для main() */
    printf("Внутри main() pooh = %d и &pooh = %p\n", pooh, &pooh);
    printf("Внутри main() bah = %d и &bah = %p\n", bah, &bah);
    mikado(pooh);
    return 0;
}
// Внутри main() pooh = 2 и &pooh = 000000e128bfff80c
// Внутри main() bah = 5 и &bah = 000000e128bfff808
// Внутри mikado() pooh = 10 и &pooh = 000000e128bfff7cc
// Внутри mikado() bah = 2 и &bah = 000000e128bfff7e0
```

# Изменение переменных в вызывающей функции

```
#include <stdio.h>
void interchange(int u, int v)
{
    int temp;
    printf("Первоначально u = %d и v = %d.\n", u, v);
    temp = u;
    u = v;
    v = temp;
    printf("Теперь u = %d и v = %d.\n", u, v);
}
int main(void)
{
    int x = 5, y = 10;
    printf("Первоначально x = %d и y = %d.\n", x, y);
    interchange(x, y);
    printf("Теперь x = %d и y = %d.\n", x, y);
    return 0;
}
```

```
// Первоначально x = 5 и y = 10.
// Первоначально u = 5 и v = 10.
// Теперь u = 10 и v = 5.
// Теперь x = 5 и y = 10.
```

# Указатели в функции

```
#include <stdio.h>
void interchange(int *u, int *v)
{
    int temp;
    temp = *u; // temp получает значение, на которое
указывает u
    *u = *v;
    *v = temp;
}
int main(void)
{
    int x = 5, y = 10;
    printf("Первоначально x = %d и y = %d.\n", x, y);
    interchange(&x, &y); // передача адресов в
функцию
    printf("Теперь x = %d и y = %d.\n", x, y);
    return 0;
}
// Первоначально x = 5 и y = 10.
// Теперь x = 10 и y = 5.
```



# Объявление параметров массива

Поскольку **имя массива** — это адрес его первого элемента, фактический аргумент в виде имени массива требует, чтобы соответствующий формальный аргумент был указателем. В этом и только в этом контексте С интерпретирует `int ar[]` как `int *ar`, т.е. `ar` является типом указателя на `int`. Поскольку в прототипах разрешено опускать имя, **все четыре** приведенных ниже прототипа **эквивалентны**:

```
int sum(int *ar, int n);  
int sum (int *, int);  
int sum(int ar[], int n);  
int sum(int [], int);
```

В определениях функций имена опускать нельзя, поэтому следующие две формы определения эквивалентны:

```
int sum(int *ar, int n) {  
    // здесь находится код  
}  
int sum(int ar [ ], int n) ; {  
    // здесь находится код  
}
```





# Функции, массивы и указатели

```
#include <stdio.h>
#define SIZE 10
int sum(int ar[], int n) // насколько велик массив?
{
    int i, total = 0;
    for (i = 0; i < n; i++)
        total += ar[i];
    printf("Размер ar составляет %zd байтов.\n", sizeof ar);
    return total;
}
int main(void)
{
    int marbles[SIZE] = {20, 10, 5, 39, 4, 16, 19, 26, 31, 20};
    long answer;
    answer = sum(marbles, SIZE);
    printf("Общая сумма элементов массива marbles равна %ld.\n", answer);
    printf("Объем памяти, отведенной под массив marbles, составляет %zd байтов.\n",
           sizeof marbles);
    return 0;
}
// Размер ar составляет 8 байтов.
// Общая сумма элементов массива marbles равна 190.
// Объем памяти, отведенной под массив marbles, составляет 40 байтов.
```

# Использование параметров типа указателей

```
#include <stdio.h>
#define SIZE 10
int sump(int *start, int *end)
{ /* использование арифметики указателей */
    int total = 0;
    while (start < end)
    {
        total += *start; // добавить значение к total
        start++;          // переместить указатель на следующий элемент
    }
    return total;
}
int main(void)
{
    int marbles[SIZE] = {20, 10, 5, 39, 4, 16, 19, 26, 31, 20};
    long answer;
    answer = sump(marbles, marbles + SIZE);
    printf("Общая сумма элементов массива marbles равна %ld.\n", answer);
    return 0;
}
// Общая сумма элементов массива marbles равна 190.
```



# Защита содержимого массива

Обычно числовые данные передаются по значению, если только программа не нуждается в изменении этого значения — в таком случае передается указатель. Массивы не предоставляют подобного выбора; вы обязаны передавать указатель. Все дело в эффективности.

## Пример 1:

```
void add_to(double ar[], int n, double val)
{
    for (int i = 0; i < n; i + + )
        ar [i] += val;
}
```

## Пример 2:

```
int sum(int ar[], int n) { // ошибочный код
    int total = 0;
    for (int i = 0 ; i < n; i + + )
        total += ar[i]++; // ошибочное инкрементирование
    return total;
}
```



# const – решение проблем

```
#include <stdio.h>
#define SIZE 5
void show_array(const double ar[], int n)
{ /* выводит содержимое массива */
    for (int i = 0; i < n; i++)
        printf("%8.3f ", ar[i]);
    putchar('\n');
}
void mult_array(double ar[], int n, double mult)
{ /* умножает каждый элемент массива на один и тот же множитель */
    for (int i = 0; i < n; i++)
        ar[i] *= mult;
}
int main(void)
{
    double dip[SIZE] = {20.0, 17.66, 8.2, 15.3, 22.22};
    printf("Исходный массив dip:\n");
    show_array(dip, SIZE);
    mult_array(dip, SIZE, 2.5);
    printf("Массив dip после вызова функции mult_array():\n");
    show_array(dip, SIZE);
    return 0;
}
```

Если функция не задумывалась как изменяющая содержимое массива, применяйте ключевое слово `const` при объявлении формального параметра в прототипе и в определении функции.

```
// Исходный массив dip:
// 20.000 17.660 8.200 15.300 22.220
// Массив dip после вызова функции mult_array():
// 50.000 44.150 20.500 38.250 55.550
```

# Функции и многомерные массивы (1/3)

```
#include <stdio.h>
#define ROWS 3
#define COLS 4
void sum_rows(int ar[][COLS], int rows);
void sum_cols(int[][COLS], int);
int sum2d(int (*ar)[COLS], int rows);
int main(void)
{
    system("chcp 65001");
    int junk[ROWS][COLS] = {
        {2, 4, 6, 8},
        {3, 5, 7, 9},
        {12, 10, 8, 6}};
    sum_rows(junk, ROWS);
    sum_cols(junk, ROWS);
    printf("Сумма всех элементов = %d\n", sum2d(junk,
ROWS));
    return 0;
}
```



# Функции и многомерные массивы (2/3)

```
void sum_rows(int ar[][COLS], int rows)
{
    int tot;
    for (int r = 0; r < rows; r++)
    {
        tot = 0;
        for (int c = 0; c < COLS; c++)
            tot += ar[r][c];
        printf("строка %d: сумма = %d\n", r, tot);
    }
}

void sum_cols(int ar[][COLS], int rows)
{
    int tot;
    for (int c = 0; c < COLS; c++)
    {
        tot = 0;
        for (int r = 0; r < rows; r++)
            tot += ar[r][c];
        printf("столбец %d: сумма = %d\n", c, tot);
    }
}
```

# Функции и многомерные массивы (3/3)

```
int sum2d(int ar[][COLS], int rows)
{
    int tot = 0;
    for (int r = 0; r < rows; r++)
        for (int c = 0; c < COLS; c++)
            tot += ar[r][c];
    return tot;
}
// строка 0: сумма = 20
// строка 1: сумма = 24
// строка 2: сумма = 36
// столбец 0: сумма = 17
// столбец 1: сумма = 19
// столбец 2: сумма = 21
// столбец 3: сумма = 23
// Сумма всех элементов = 80
```

# Массивы переменной длины

Массивы переменной длины позволяют использовать переменные для размерности массива. Например:

```
int quarters = 4;  
int regions = 5;  
double sales[regions][quarters]; // массив переменной длины
```

С массивами переменной длины связаны некоторые ограничения.

- Они должны иметь автоматический класс хранения, а это означает их объявление либо в функции без применения модификаторов класса хранения `static` или `extern`, либо в виде параметров функции.
- Инициализация в объявлении невозможна.
- В стандарте C11 массивы переменной длины являются необязательной возможностью языка в отличие от C99, где они были обязательными.



# Массивы переменной длины 1/2

```
#define ROWS 3
#define COLS 4
int sum2d(int rows, int cols, int ar[rows][cols]);
int main(void)
{
    int rs = 3, cs = 10;
    int junk[ROWS][COLS] = {{2, 4, 6, 8},
                             {3, 5, 7, 9},
                             {12, 10, 8, 6}};
    int morejunk[ROWS - 1][COLS + 2] = {{20, 30, 40, 50, 60, 70},
                                         {5, 6, 7, 8, 9, 10}};

    int varr[rs][cs]; // массив переменной длины
    for (int i = 0; i < rs; i++)
        for (int j = 0; j < cs; j++)
            varr[i][j] = i * j + j;
    printf("Традиционный массив 3x5\n");
    printf("Сумма всех элементов = %d\n", sum2d(ROWS, COLS, junk));
    printf("Традиционный массив 2x6\n");
    printf("Сумма всех элементов = %d\n", sum2d(ROWS - 1, COLS + 2, morejunk));
    printf("Массив переменной длины 3x10\n");
    printf("Сумма всех элементов = %d\n", sum2d(rs, cs, varr));
    return 0;
}
```



# Массивы переменной длины 2/2

```
// функция с параметром типа массива переменной длины
int sum2d(int rows, int cols, int ar[rows][cols])
{
    int tot = 0;
    for (int r = 0; r < rows; r++)
        for (int c = 0; c < cols; c++)
            tot += ar[r][c];
    return tot;
}
// Традиционный массив 3x5
// Сумма всех элементов = 80
// Традиционный массив 2x6
// Сумма всех элементов = 315
// Массив переменной длины 3x10
// Сумма всех элементов = 270
```





# Составные литералы

Предположим, что вы хотите передать в функцию значение с помощью параметра `int`; вы можете передать переменную `int`, но также константу `int`, такую как `5`. До выхода стандарта C99 положение дел с функцией, принимающей аргумент типа массива, было другим; можно было передавать массив, но отсутствовал эквивалент для константы типа массива. Стандарт C99 изменил эту ситуацию, введя составные литералы.

**Литералы** — это константы, которые не являются символическими.

**Например:**

`5` — литерал типа `int`,

`81.3` — литерал типа `double`,

`'У'` — литерал типа `char`,

`"elephant"` — строковый литерал

Для массивов составной литерал выглядит подобно списку инициализации массива, который предварен именем типа, заключенным в круглые скобки.

**Пример составного литерала:**

```
(int [2]){10, 20} // составной литерал
```

```
int * ptl; ptl = (int [2]) {10, 20};
```

```
total = sum ((int []) {4,4,4,5,5,5}, 6);
```

```
int (*pt2)[4]; // объявление указателя на массив из массивов с 4 значениями int
```

```
pt2 = (int [2][4]) {{1,2,3,-9), {4,5,6,-8}};
```



# Составные литералы 1/2

```
#include <stdio.h>
#define COLS 4
int sum2d(const int ar[][COLS], int rows);
int sum(const int ar[], int n);
int main(void)
{
    int total1, total2, total3;
    int *pt1;
    int(*pt2)[COLS];
    pt1 = (int[2]){10, 20};
    pt2 = (int[2][COLS]){{1, 2, 3, -9}, {4, 5, 6, -8}};
    total1 = sum(pt1, 2);
    total2 = sum2d(pt2, 2);
    total3 = sum((int[]){4, 4, 4, 5, 5, 5}, 6);
    printf("total1 = %d\n", total1);
    printf("total2 = %d\n", total2);
    printf("total3 = %d\n", total3);
    return 0;
}
```



# Составные литералы 2/2

```
int sum(const int ar[], int n)
{
    int total = 0;
    for (int i = 0; i < n; i++)
        total += ar[i];
    return total;
}
int sum2d(const int ar[][COLS], int rows)
{
    int tot = 0;
    for (int r = 0; r < rows; r++)
        for (int c = 0; c < COLS; c++)
            tot += ar[r][c];
    return tot;
}
// total1 = 30
// total2 = 4
// total3 = 27
```



# Адженда

**Функции**

**45 минут**

**Символьные  
строки и  
строковые  
функции**

**45 минут**

# Демонстрационная программа

```
// strings1.c
#include <stdio.h>
#define MSG "Я - символьная строковая константа."
#define MAXLENGTH 81
int main(void)
{
    system("chcp 65001");
    char words[MAXLENGTH] = "Я являюсь строкой, хранящейся в массиве.";
    const char *pt1 = "Что-то указывает на меня.";
    puts("Вот несколько строк:");
    puts(MSG);
    puts(words);
    puts(pt1);
    return 0;
}
// Вот несколько строк:
// Я - символьная строковая константа.
// Я являюсь строкой, хранящейся в массиве.
// Что-то указывает на меня.
```

# Символьные строковые литералы

**Строковый литерал**, который также называют строковой константой, представляет собой произвольную последовательность символов, помещенную в двойные кавычки.

## Например

```
char greeting[50] = "Здравствуйте, " " как вы себя" " чувствуете" " сегодня?";
```

эквивалентно:

```
char greeting[50] = "Здравствуйте, как вы себя чувствуете сегодня?";
```

**Символьные строковые** константы размещаются в статическом классе хранения, т.е. если вы используете строковую константу в функции, то эта строка сохраняется только однажды и существует на протяжении времени выполнения программы. Вся фраза, заключенная в кавычки, действует в качестве указателя на место хранения строки.

```
#include <stdio.h>
int main(void)
{
    printf("%s, %p, %c\n", "You", " are ", *"a space scientist");
    return 0;
}
// You, 00007ff76eb6b000, a
```



# Массивы символьных строк и инициализация

Следующее объявление инициализирует массив `m1` символами заданной строки:

```
const char m1[40] = "Постарайтесь уложиться в одну строку.";
```

Ключевое слово `const` отражает намерение не изменять эту строку.

Показанная форма является сокращением для стандартной формы инициализации массива:

```
const char m1[40] = {'П', 'о', 'с', 'т', 'а', 'р', 'а', 'й', 'т', 'е', 'с', 'ь',  
    ' ', 'у', 'л', 'о', 'ж', 'и', 'т', 'ь', 'с', 'я',  
    ' ', 'в', ' ', 'о', 'д', 'н', 'у', ' ', 'с', 'т',  
    'р', 'о', 'к', 'у', ' ', '\0'};
```

Без завершающего нулевого символа вы получите символьный массив, а не строку.

*Компилятор самостоятельно определит этот размер:*

```
const char m2 [] = "Если вам не о чем думать, вообразите что-нибудь.";
```

Лишние элементы инициализируются символом `\0`



```
const char pets[12] = "Кот баюн." ;
```

# Массивы символьных строк и инициализация

```
int n = 8;
char cookies[1];           // допустимо
char cakes[2 + 5];         // допустимо, поскольку размер константный
выражением
char pies[2 * sizeof(long double) + 1]; // допустимо
char cruris[n];            // не допускалось до выхода стандарта C99;
                           // в C99 это массив переменной длины

char car[10] = "Луна";
car == &car[0]
*car == 'л'
*(car + 1) == car[1] == 'у'

// Например, в примере имеется следующее объявление:
const char *pt1 = "Что-то указывает на меня.";
// Это объявление очень близко к такому объявлению:
const char ar1[] = "Что-то указывает на меня.";
```

# Массивы или указатели

Инициализация массива приводит к копированию строки из статической памяти в массив, тогда как инициализация указателя просто копирует адрес строки.

```
#define MSG "Я особенный."
#include <stdio.h>
int main()
{
    char ar[] = MSG;
    const char *pt = MSG;
    printf("адрес \"Я особенный.\": %p \n", "Я особенный.");
    printf("        адрес ar: %p\n", ar);
    printf("        адрес pt: %p\n", pt);
    printf("        адрес MSG: %p\n", MSG);
    printf("адрес \"Я особенный.\": %p \n", "Я особенный.");
    return 0;
}
// адрес "Я особенный.": 00007ff70b90b000
//        адрес ar: 00000067d6fff820
//        адрес pt: 00007ff70b90b000
//        адрес MSG: 00007ff70b90b000
// адрес "Я особенный.": 00007ff70b90b000
```



# Различия между массивами и указателями

```
char heart[] = "Я люблю Тилли!";
const char *head = "Я люблю Милли!";
for (i = 0; i < 7; i++)
    putchar(heart[i]); // Я люблю
putchar('\n');
for (i = 0; i < 7; i++)
    putchar(head[i]); // Я люблю
putchar('\n');
for (i = 0; i < 7; i++)
    putchar(*(heart + i)); // Я люблю
putchar('\n');
for (i = 0; i < 7; i++)
    putchar(*(head + i)); // Я люблю
putchar('\n');

while (*(head) != '\0') /* остановиться в конце строки */
    putchar(*(head++)); /*вывести символ, переместить указатель */
head = heart; /* head теперь указывает на массив heart */
heart = head; /* недопустимая конструкция */
heart[8] = 'М';
```



# Массивы символьных строк

```
#include <stdio.h>
#define SLEN 40
#define LIM 5
int main(void)
{
    const char *mytalents[LIM] = {
        "Мгновенное складывание чисел",
        "Точное умножение", "Накапливание данных",
        "Исполнение инструкций с точностью до буквы",
        "Знание языка программирования C"};
    char yourtalents[LIM][SLEN] = {
        "Хождение по прямой",
        "Здоровый сон", "Просмотр телепередач",
        "Рассылка писем", "Чтение электронной почты"};
    int i;
    puts("Сравним наши таланты.");
    printf("%-52s %-25s\n", "Мои таланты", "Ваши таланты");
    for (i = 0; i < LIM; i++)
        printf("%-52s %-25s\n", mytalents[i], yourtalents[i]);
    printf("\nразмер mytalents: %zd, размер yourtalents: %zd\n",
        sizeof(mytalents), sizeof(yourtalents));
    return 0;
}
```

```
// Сравним наши таланты.
// Мои таланты
// Мгновенное складывание чисел
// Точное умножение
// Накапливание данных
// Исполнение инструкций с точностью до буквы
// Знание языка программирования C

Ваши таланты
Хождение по прямой
Здоровый сон
Просмотр телепередач
Рассылка писем
Чтение электронной почты

// размер mytalents: 40, размер yourtalents: 200
```

# Массивы символьных строк

|   |   |   |   |   |   |    |    |    |
|---|---|---|---|---|---|----|----|----|
| Я | б | л | о | к | о | \0 | \0 | \0 |
|---|---|---|---|---|---|----|----|----|

|   |   |   |   |   |    |    |    |    |
|---|---|---|---|---|----|----|----|----|
| Г | р | у | ш | а | \0 | \0 | \0 | \0 |
|---|---|---|---|---|----|----|----|----|

|   |   |   |   |   |   |   |   |    |
|---|---|---|---|---|---|---|---|----|
| А | п | е | л | ь | с | и | н | \0 |
|---|---|---|---|---|---|---|---|----|

```
const char fruit1[3][9] =  
    {"Яблоко",  
     "Груша",  
     "Апельсин"};
```

|   |   |   |   |   |   |    |
|---|---|---|---|---|---|----|
| Я | б | л | о | к | о | \0 |
|---|---|---|---|---|---|----|

|   |   |   |   |   |    |
|---|---|---|---|---|----|
| Г | р | у | ш | а | \0 |
|---|---|---|---|---|----|

|   |   |   |   |   |   |   |   |    |
|---|---|---|---|---|---|---|---|----|
| А | п | е | л | ь | с | и | н | \0 |
|---|---|---|---|---|---|---|---|----|

```
const char *fruit2[3] =  
    {"Яблоко",  
     "Груша",  
     "Апельсин"};
```

Отличия в объявлениях



# Указатели и строки

```
#include <stdio.h>
int main(void)
{
    const char *mesg = "Не позволяйте себя запутать!";
    const char *copy;
    copy = mesg;
    printf("%s\n", copy);
    printf("mesg = %s; &mesg = %p; value = %p\n",
           mesg, &mesg, mesg);
    printf("copy = %s; &copy = %p; value = %p\n",
           copy, &copy, copy);
    return 0;
}
// Не позволяйте себя запутать!
// mesg = Не позволяйте себя запутать!; &mesg =
0000004ba15ffd98; value = 00007ff7fa49b000
// copy = Не позволяйте себя запутать!; &copy =
0000004ba15ffd90; value = 00007ff7fa49b000
```



# Ввод строк - gets()

Функция gets() не делает проверку, уместится ли вводимая строка в массив. **C99** - gets() применять не рекомендуется; **C11** исключил функцию gets() из стандарта

```
#include <stdio.h>
#define STLEN 81
int main(void)
{
    char words[STLEN];
    puts("Введите строку.");
    gets(words);
    printf("Ваша строка, выведенная дважды:\n");
    printf("%s\n", words);
    puts(words);
    puts("Готово.");
    return 0;
}
// Введите строку.
// Hello students!
// Ваша строка, выведенная дважды:
// Hello students!
// Hello students!
// Готово.
```



# Ввод строк - fgets()

Функция fgets() предотвращает возможную проблему переполнения, принимая второй аргумент, который ограничивает количество считываемых символов.

- Если второй аргумент имеет значение n, то функция fgets() прочитает n-1 символов или будет читать до появления символа новой строки в зависимости от того, что произойдет раньше.
- Если функция fgets() сталкивается с символом новой строки, она сохраняет его в строке, в отличие от функции gets(), которая отбрасывает его.
- Функция fgets() принимает третий аргумент, указывающий файл, из которого должно производиться чтение. Для чтения с клавиатуры в качестве этого аргумента используется stdin (от standard input- стандартный ввод); этот идентификатор определен в stdio.h.



# Ввод строк - fgets()

```
#include <stdio.h>
#define STLEN 15
int main(void)
{
    char words[STLEN];
    puts("Введите строку.");
    fgets(words, STLEN, stdin);
    printf("Ваша строка, выведенная дважды (с помощью puts (), а затем fputs ()):\\n");
    puts(words);
    fputs(words, stdout);
    puts("Введите еще одну строку.");
    fgets(words, STLEN, stdin);
    printf("Ваша строка, выведенная дважды (с помощью puts (), а затем fputs()):\\n");
    puts(words);
    fputs(words, stdout);
    puts("Готово.");
    return 0;
}
```

```
// Введите строку.
// hello
// Ваша строка, выведенная дважды (с помощью puts (), а затем fputs ()):
// hello

// hello
// Введите еще одну строку.
// Good morning, students!
// Ваша строка, выведенная дважды (с помощью puts (), а затем fputs()):
// Good morning,
// Good morning, Готово.
```



# Ввод строк - fgets()

```
#include <stdio.h>
#define STLEN 10
int main(void)
{
    char words[STLEN];
    puts("Введите строки (или пустую строку для выхода из
программы):");
    while (fgets(words, STLEN, stdin) != NULL && words[0] != '\n')
        fputs(words, stdout);
    puts("Готово.");
    return 0;
}
// Введите строки (или пустую строку для выхода из программы):
// London is the capital of Great Britain
// London is the capital of Great Britain
// Moscow is the capital of Russia
// Moscow is the capital of Russia

// Готово.
```



# Ввод строк - `gets_s()`

Необязательная функция `gets_s()` в C11, подобно `fgets()`, применяет аргумент для ограничения количества читаемых символов. С учетом определений из примера следующий код будет считывать строку ввода в массив `words`, обеспечивая появление символа новой строки в числе первых 9 символов ввода: `gets_s(words, STLEN);`

Ниже описаны три основных отличия этой функции от `fgets()`.

- Функция `gets_s()` просто выполняет чтение из стандартного ввода, поэтому она не нуждается в третьем аргументе.
- Если функция `gets_s()` считывает символ новой строки, то отбрасывает его, а не сохраняет.
- Если `gets_s()` прочитает максимальное количество символов, и среди них символ новой строки отсутствует, она предпринимает несколько действий.
  1. устанавливает первый символ целевого массива в нулевой символ.
  2. читает и отбрасывает последующие введенные данные, пока не встретится символ новой строки или признак конца файла.
  3. возвращает нулевой указатель.
  4. вызывает зависящую от реализации функцию “обработчика” (или же выбранную вами функцию), которая может привести к выходу из программы или прекращению ее работы.



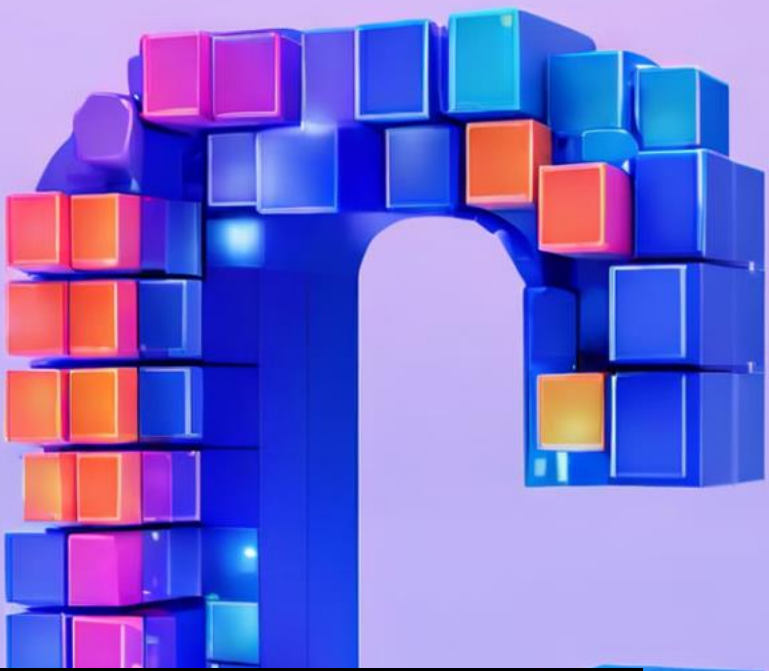


# Ввод строк – scanf()

| Оператор ввода      | Исходная очередь ввода | Содержимое строки имени | Остальная часть очереди |
|---------------------|------------------------|-------------------------|-------------------------|
| scanf("%s", name);  | Иванов Иван            | Иванов                  | Иван                    |
| scanf("%5s", name); | Иванов Иван            | Ивано                   | в Иван                  |
| scanf("%5s", name); | Иван Иванов            | Иван                    | Иванов                  |

```
#include <stdio.h>
int main(void)
{
    char name1[11], name2[11];
    int count;
    printf("Введите два имени.\n");
    count = scanf("%5s %10s", name1, name2);
    printf("Прочитано %d имени: %s и %s.\n",
           count, name1, name2);
    return 0;
}
```

```
// Введите два имени.
// Alex Mikhail
// Прочитано 2 имени: Alex и Mikhail.
// Введите два имени.
// Mikhail Alexandra
// Прочитано 2 имени: Mikha и il.
// Введите два имени.
// Misha Alexandrovna
// Прочитано 2 имени: Misha и Alexandrov.
```



# Вывод строк - puts()

```
#include <stdio.h>
#define DEF "Я - строка, определенная директивой #define."
int main(void)
{
    char str1[80] = "Массив был инициализирован моим значением.";
    char dont[] = {'Y', 'E', 'S', '!'};
    const char *str2 = "Указатель был инициализирован моим значением.";
    puts("Я - аргумент функции puts ().");
    puts(DEF);
    puts(str1);
    puts(str2);
    puts(&str1[5]);
    puts(str2 + 4);
    puts(dont); /* dont не является строкой */
    return 0;
}
// Я - аргумент функции puts ().
// Я - строка, определенная директивой #define.
// Массив был инициализирован моим значением.
// Указатель был инициализирован моим значением.
// в был инициализирован моим значением.
// атель был инициализирован моим значением.
// YES!Массив был инициализирован моим значением.
```



# Ввод строк - `fgets()`, `printf()`

Функция **`fputs()`** представляет собой версию `puts()`, ориентированную на файлы. Важные отличия между ними описаны ниже.

- Функция `fputs()` принимает второй аргумент, указывающий файл, в который должна производиться запись. Для вывода на дисплей можно применять аргумент `stdout` (от `standard output` — стандартный вывод), который определен в `stdio.h`.
- В отличие от `puts()`, функция `fputs()` не делает автоматическое дополнение вывода символом новой строки.

Функция **`printf()`** менее удобна в употреблении, чем `puts()`, но она более универсальна. Функция `printf()` не выводит автоматически каждую строку в новой строке на экране. Вместо этого вы должны самостоятельно указывать, где должны начинаться новые строки.



# Строковые функции - strlen()

```
#include <stdio.h>
#include <string.h> /* содержит прототипы строковых функций */
void fit(char *, unsigned int);
int main(void)
{
    char mesg[] = "Все должно быть максимально простым,"
                  " но не более. ";

    puts(mesg);
    fit(mesg, 35);
    puts(mesg);
    puts("Рассмотрим еще несколько строк.");
    puts(mesg + 36);
    return 0;
}

void fit(char *string, unsigned int size)
{
    if (strlen(string) > size)
        string[size] = '\\0';
}

// Все должно быть максимально простым, но не более.
// Все должно быть максимально простым
// Рассмотрим еще несколько строк.
// но не более.
```



# Строковые функции - strcat()

**Функция `strcat()`** (от string concatenation — конкатенация строк) в качестве аргументов принимает две строки. Копия второй строки присоединяется в конец первой строки, и такая объединенная версия становится новой первой строкой. Вторая строка не изменяется.

```
#include <stdio.h>
#include <string.h> /* объявление strcat() */
#define SIZE 80
int main(void)
{
    char flower[SIZE];
    char addon[] = " пахнет как старые валенки.";
    puts("Какой у вас любимый цветок?");
    gets(flower);
    strcat(flower, addon);
    puts(flower);
    puts(addon);
    return 0;
}
// Какой у вас любимый цветок?
// Rose
// Rose пахнет как старые валенки.
//  пахнет как старые валенки.
```



# Строковые функции - `strncat()`

Функция **`strcat()`** не проверяет, уместится ли вторая строка в первый массив. Функция **`strncat()`** принимает второй аргумент, указывающий максимальное количество добавляемых символов.

```
#include <stdio.h>
#include <string.h>
#define BUGSIZE 13
int main(void)
{
    char addon[] = " пахнет как старые валенки.";
    char bug[BUGSIZE];
    int available;
    puts("Какое у вас любимое насекомое?");
    gets(bug);
    available = BUGSIZE - strlen(bug) - 1;
    strncat(bug, addon, available);
    puts(bug);
    return 0;
}
// Какое у вас любимое насекомое?
// mosquito
// mosquito пах
```





# Строковые функции - strcmp()

Функция **strcmp()** (от string comparison — сравнение строк) делает для строк то же, что операции отношений делают для чисел. В частности, она возвращает 0, если оба строковых аргумента одинаковы, и ненулевое значение в противном случае.

```
#include <stdio.h>
#include <string.h> // объявление strcmp()
#define ANSWER "Grant"
#define SIZE 40
int main(void)
{
    char try[SIZE];
    puts("Кто похоронен в могиле Гранта?");
    scanf("%s", try);
    while (strcmp(try, ANSWER) != 0)
    {
        puts("Неправильно! Попробуйте еще раз.");
        scanf("%s", try);
    }
    puts("Теперь правильно!");
    return 0;
}
```

```
// Кто похоронен в могиле Гранта?
// Grant
// Теперь правильно!
```



# Возвращаемое значение функции strcmp()

```
#include <stdio.h>
#include <string.h>
int main(void)
{
    printf("strcmp(\"A\", \"A\") возвращает ");
    printf("%d\n", strcmp("A", "A"));
    printf("strcmp(\"A\", \"B\") возвращает ");
    printf("%d\n", strcmp("A", "B"));
    printf("strcmp(\"B\", \"A\") возвращает ");
    printf("%d\n", strcmp("B", "A"));
    printf("strcmp(\"C\", \"A\") возвращает ");
    printf("%d\n", strcmp("C", "A"));
    printf("strcmp(\"Z\", \"a\") возвращает ");
    printf("%d\n", strcmp("Z", "a"));
    printf("strcmp (\"apples\", \"apple\") возвращает ");
    printf("%d\n", strcmp("apples", "apple"));
    return 0;
}
```

```
// strcmp("A", "A") возвращает 0
// strcmp("A", "B") возвращает -1
// strcmp("B", "A") возвращает 1
// strcmp("C", "A") возвращает 1
// strcmp("Z", "a") возвращает -1
// strcmp("apples", "apple")
// возвращает 1
```

# Строковые функции - `strncmp()`

Функция `strcmp()` сравнивает строки до тех пор, пока не найдет пару соответствующих символов, которые отличаются друг от друга. Функция `strncmp()` сравнивает строки до тех пор, пока не обнаружит в них различия либо пока не сравнит количество символов в обеих строках, указанное в третьем аргументе.

```
#include <stdio.h>
#include <string.h>
#define LISTSIZE 6
int main()
{
```

```
// Найдено: астрономия
// Найдено: астрофизика
// Найдено: астролябия
// Количество слов в списке, начинающихся с астро: 3
```

```
    const char *list[LISTSIZE] = {"астрономия", "астатиэм",
                                   "астрофизика", "остракизм", "астеризм", "астролябия"};
    int count = 0;
    for (int i = 0; i < LISTSIZE; i++)
        if (strncmp(list[i], "astro", 5) == 0)
        {
            printf("Найдено: %s\n", list[i]);
            count++;
        }
    printf("Количество слов в списке, начинающихся с астро: %d\n", count);
    return 0;
}
```

# Строковые функции - strcpy()

Функция **strcpy()** принимает в качестве аргументов два указателя на строки. Второй указатель, который ссылается на исходную строку, может быть объявленным указателем, именем массива или строковой константой. Первый указатель, ссылающийся на копию, должен указывать на объект данных, такой как массив с размером, достаточным для хранения этой строки.

```
char *s_gets(char *st, int n) //собственная функция для вывода
{
    char *ret_val;
    int i = 0;
    ret_val = fgets(st, n, stdin);
    if (ret_val)
    {
        while (st[i] != '\n' && st[i] != '\0')
            i++;
        if (st[i] == '\n')
            st[i] = '\0';
        else // требуется наличие words[i] == '\0'
            while (getchar() != '\n')
                continue;
    }
    return ret_val;
}
```

# Строковые функции - strcpy()

```
#include <stdio.h>
#include <string.h> // объявление strcpy()
#define SIZE 40
#define LIM 5
int main(void)
{
    char qwords[LIM][SIZE], temp[SIZE];
    printf("Введите %d слов, которые начинаются с буквы k:\n", LIM);
    while (int i < LIM && s_gets(temp, SIZE))
    {
        if (temp[0] != 'k')
            printf("%s не начинается с буквы k!\n", temp);
        else
        {
            strcpy(qwords[i], temp);
            i++;
        }
    }
    puts("Список принятых слов:");
    for (i = 0; i < LIM; i++)
        puts(qwords[i]);
    return 0;
}
```

```
// Введите 5 слов, которые
// начинаются с буквы k:
// kitchen
// cat
// cat не начинается с буквы k!
// keep
// kind
// know
// key
// Список принятых слов:
// kitchen
// keep
// kind
// know
// key
```

# Строковые функции - strcpy()

Функция strcpy () обладает еще двумя свойствами, которые вы можете считать удобными. Во-первых, ее типом является char\*. Она возвращает значение своего первого аргумента — адрес символа. Во-вторых, первый аргумент не обязательно должен указывать на начало массива; это позволяет копировать только часть массива.

```
#include <stdio.h>
#include <string.h> // объявление strcpy ()
#define WORDS "наихудшим"
#define SIZE 60
int main(void)
{
    const char *orig = WORDS;
    char copy[SIZE] = "Будьте лучшим, чем могли бы быть.";
    char *ps;
    puts(orig);
    puts(copy);
    ps = strcpy(copy + 7, orig);
    puts(copy);
    puts(ps);
    return 0;
}
```

```
// наихудшим
// Будьте лучшим, чем могли бы быть.
// Будьте наихудшим
// наихудшим
```



# Строковые функции - sprintf()

Функция sprintf() объявлена в заголовочном файле stdio.h, а не в string.h. Она работает подобно printf (), но осуществляет запись в строку, а не на дисплей.

```
#include <stdio.h>
#define MAX 20
char *s_gets(char *st, int n);
int main(void)
{
    char first[MAX];
    char last[MAX];
    char formal[2 * MAX + 10];
    double prize;
    puts("Введите свое имя:");
    s_gets(first, MAX);
    puts("Введите свою фамилию:");
    s_gets(last, MAX);
    puts("Введите сумму денежного приза:");
    scanf("%lf", &prize);
    sprintf(formal, "%s, %-19s: $%.2f\n", last, first, prize);
    puts(formal);
    return 0;
}
```

```
// Введите свое имя:
// Mikhail
// Введите свою фамилию:
// Philippov
// Введите сумму денежного приза:
// 25000
// Philippov, Mikhail           : $25000.00
```

# Другие строковые функции

**char \* strcpy(char \* restrict s1, const char \* restrict s2);**

Эта функция копирует строку (включая нулевой символ), указанную s2, в ячейку, на которую указывает s1. Возвращаемым значением является s1.

**char \* strncpy (char \* restrict s1, const char \* restrict s2, size\_t n);**

Эта функция копирует в ячейку, указанную s1, не более n символов из строки, на которую указывает s2. Возвращаемым значением является s1. Символы, следующие за нулевым символом, не копируются, и если исходная строка короче n символов, оставшаяся часть целевой строки заполняется нулевыми символами. Если исходная строка содержит n или больше символов, нулевой символ не копируется. Возвращаемым значением является s1.

**char \* strcat (char \* restrict s1, const char \* restrict s2);**

Строка, указанная s2, копируется в конец строки, на которую указывает s1. Первый символ строки s2 копируется поверх нулевого символа строки s1. Возвращаемым значением является s1.

**char \* strncat (char \* restrict s1, const char \* restrict s2, size\_t n);**

К строке s1 добавляется не более n символов строки s2, причем первый символ строки s2 копируется поверх нулевого символа строки s1. Нулевой символ и любые другие символы, которые за ним следуют в строке s2, не копируются, а к результату добавляется нулевой символ. Возвращаемым значением является s1.

# Другие строковые функции

**int strcmp (const char \* s1, const char \* s2);**

Эта функция возвращает положительное значение, если в последовательности сопоставления машины строка s1 следует за строкой s2, значение 0, если строки идентичны, и отрицательное значение, если в последовательности сопоставления первая строка предшествует второй.

**int strncmp (const char \* s1, const char \* s2, size\_t n);**

Эта функция работает подобно strcmp(), за исключением того, что процедура сравнения останавливается после просмотра n символов либо при появлении первого нулевого символа, в зависимости от того, что произойдет раньше.

**char \*strchr (const char \* s, int c);**

Эта функция возвращает указатель на первую ячейку строки s, в которой содержится символ c. (Завершающий нулевой символ является частью строки, так что его тоже можно искать.) Если символ не найден, функция возвращает нулевой указатель.

**char \*strpbrk (const char \* s1, const char \* s2);**

Эта функция возвращает указатель на первую ячейку строки s1, в которой содержится любой символ, найденный в строке s2. Эта функция возвращает нулевой указатель, если ни одного символа не найдено.

# Другие строковые функции

**`char *strrchr (const char * s, int c);`**

Эта функция возвращает указатель на последнее вхождение символа `c` в строке `s`. (Завершающий нулевой символ является частью строки, так что его тоже можно искать.) Если символ не найден, функция возвращает нулевой указатель.

**`char *strstr (const char * s1, const char * s2);`**

Эта функция возвращает указатель на первое вхождение строки `s2` внутри строки `s1`. Если строка не найдена, функция возвращает нулевой указатель.

**`size_t strlen (const char * s);`**

Эта функция возвращает количество символов, не включая нулевой, находящихся в строке `s`.

# Демо ПК – сортировка строк

```
#include <stdio.h>
#define SIZE 81 /* лимит на длину строки, включая \0 */
#define LIM 20 /* максимальное количество читаемых строк */
#define HALT /* нулевая строка для прекращения ввода */
void stsrst(char *strings[], int num); /* функция сортировки строк */
char *s_gets(char *st, int n);
int main(void)
{
    char input[LIM][SIZE]; /* массив для сохранения входных данных */
    char *ptstr[LIM]; /* массив переменных типа указателя */
    int ct = 0; /* счетчик ввода */
    int k; /* счетчик вывода */
    printf("Введите до %d строк, и они будут отсортированы.\n", LIM);
    printf("Чтобы остановить ввод, нажмите клавишу Enter в начале строки.\n");
    while (ct < LIM && s_gets(input[ct], SIZE) != NULL && input[ct][0] != '\0')
    {
        ptstr[ct] = input[ct]; /* установка указателей на строки */
        ct++;
    }
    stsrst(ptstr, ct); /* сортировщик строк */
    puts("\nОтсортированный список:\n");
    for (k = 0; k < ct; k++)
        puts(ptstr[k]); /* отсортированные указатели */
    return 0;
}
/* функция сортировки указателей на строки */
void stsrst(char *strings[], int num)
{
    char *temp;
    int top, seek;
    for (top = 0; top < num - 1; top++)
        for (seek = top + 1; seek < num; seek++)
            if (strcmp(strings[top], strings[seek]) > 0)
            {
                temp = strings[top];
                strings[top] = strings[seek];
                strings[seek] = temp;
            }
}
char *s_gets(char *st, int n)
{
    char *ret_val;
    int i = 0;
    ret_val = fgets(st, n, stdin);
    if (ret_val)
    {
        while (st[i] != '\n' && st[i] != '\0')
            i++;
        if (st[i] == '\n')
            st[i] = '\0';
        else // требуется наличие words[i] == '\0'
            while (getchar() != '\n')
                continue;
    }
    return ret_val;
}
```

# Резюме

- Мы познакомились с функциями
- Узнали зачем нужны `const` и `*` в функциях
- Разобрали строковые функции
- Разобрали функции ввода вывода
- Посмотрели на первую сортировку