

**Курс «Алгоритмы и алгоритмические языки»
1 семестр 2016/2017**

Лекция 7

Пример программы. Количество дней между двумя датами

```
int main (void)
{
    while (1) {
        int m1, d1, y1, m2, d2, y2;
        int t1, t2;
        int days1, days2, total;

        if (scanf ("%d%d%d%d%d%d", &d1, &m1, &y1, &d2, &m2, &y2) != 6)
            break;
        t1 = check_date (d1, m1, y1);
        if (t1 == 1 || (t2 = check_date (d2, m2, y2)) == 1)
            break;
        else if (t1 == 2 || t2 == 2)
            continue;

        days1 = days_from_jan1 (d1, m1, y1);
        days2 = days_from_jan1 (d2, m2, y2);
        total = days_between_years (y1, y2) + (days2 - days1);
        printf ("Days between dates: %d, weeks between days: %d\n",
                total, total / 7);
    }
    return 0;
}
```

Пример программы. Количество дней между двумя датами

```
#include <stdio.h>

static int check_date (int d, int m, int y)
{
    if (!d || !m || !y)
        return 1;
    if (d < 0 || m < 0 || y < 0)
    {
        printf ("%d %d %d: wrong date\n", d, m, y);
        return 2;
    }
    return 0;
}

while (1) {
<...>
    t1 = check_date (d1, m1, y1);
    if (t1 == 1 || (t2 = check_date (d2, m2, y2)) == 1)
        break;
    else if (t1 == 2 || t2 == 2)
        continue;
<...>
}
```

Пример программы. Количество дней между двумя датами

```
static int leap_year (int y)
{
    return (y % 400 == 0) || (y % 4 == 0 && y % 100 != 0);
}
```

```
static int days_in_year (int y)
{
    return leap_year (y) ? 366 : 365;
}
```

```
static int days_between_years (int y1, int y2)
{
    int i;
    int days = 0;

    for (i = y1; i < y2; i++)
        days += days_in_year (i);
    return days;
}
```

Пример программы. Количество дней между двумя датами

```
static int days_from_jan1 (int d, int m, int y)
{
    int days = 0;

    switch (m) {
        case 12: days += 30;
        case 11: days += 31;
        case 10: days += 30;
        case 9: days += 31;
        case 8: days += 31;
        case 7: days += 30;
        case 6: days += 31;
        case 5: days += 30;
        case 4: days += 31;
        case 3: days += leap_year (y) ? 29 : 28;
        case 2: days += 31;
        case 1: break;
    }
    return days + d;
}
```

Инициализация массивов

тип имя_массива[размер1]...[размерN] = {список_значений};

Пример (справа для наглядности использованы дополнительные фигурные скобки – группировка подагрегатов)

```
int sqrs[10][2] = {  
    1, 1,  
    2, 4,  
    3, 9,  
    4, 16,  
    5, 25,  
    6, 36,  
    7, 49,  
    8, 64,  
    9, 81,  
    10, 100  
}
```

```
int sqrs[10][2] = {  
    {1, 1},  
    {2, 4},  
    {3, 9},  
    {4, 16},  
    {5, 25},  
    {6, 36},  
    {7, 49},  
    {8, 64},  
    {9, 81},  
    {10, 100}  
}
```

Инициализация массивов

тип имя_массива[размер1]...[размерN] = {список_значений};

- ◆ Можно не указывать размер массива – он будет вычислен по количеству элементов инициализатора

```
int sqrs[] = {1, 4, 9, 16, 25}; /* 5 элементов */
```

- ◆ C99: инициализация лишь некоторых элементов (остальные инициализируются нулями)

```
int days[12] = {31, 28, [4] = 31, 30, 31, [1] = 29};
```

- ◆ При инициализации одного элемента дважды используется последняя
- ◆ После задания номера элемента дальнейшие инициализаторы присваиваются следующим по порядку элементам
- ◆ Можно использовать модификаторы `const`, `static` и т.п.
- ◆ Можно использовать любое *константное целочисленное выражение* для определения размера массива
 - ◆ `const`-переменная не является константным выражением!

Строки

- ◆ **Строка** – это одномерный массив типа `char`
Объявляя массив, предназначенный для хранения строки, необходимо предусмотреть место для символа `'\0'` (конец строки)
- ◆ **Строковая константа** (например, `"string"`).
В конец строковой константы компилятор добавляет `'\0'`.
- ◆ Стандартная библиотека функций работы со строками `<string.h>`, в частности, содержит такие функции, как:
 - ◆ `strcpy(s1, s2)` (копирование `s2` в `s1`)
 - ◆ `strcat(s1, s2)` (конкатенация `s2` и `s1`)
 - ◆ `strlen(s)` (длина строки `s`)
 - ◆ `strcmp(s1, s2)` (сравнение `s2` и `s1` в лексикографическом порядке: 0, если `s1` и `s2` совпадают, отрицательное значение, если `s1 < s2`, положительное значение, если `s1 > s2`)
 - ◆ `strchr(s, ch)` (указатель на первое вхождение символа `ch` в `s`)
 - ◆ `strstr(s1, s2)` (указатель на первое вхождение подстроки `s2` в строку `s1`)

Строки

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

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

```
int main (void)
```

```
{
```

```
    char string1[80], string2[80], smp[3] = "BMK";
```

```
    fgets (string1, 80, stdin); string1[strlen (string1)-1] = '\\0';
```

```
    fgets (string2, 80, stdin); string2[strlen (string2)-1] = '\\0';
```

```
    printf ("Строки имеют длину: первая %d, вторая %d\\n,  
           strlen (string1), strlen (string2));
```

```
    if (!strcmp (string1, string2))
```

```
        printf ("строки равны\\n");
```

```
    strncat (string1, string2, 80 - strlen (string1) - 1);
```

```
    printf ("%s\\n", string1);
```

```
    sprintf (string1, "Привет, %s", smp);
```

```
    puts (string1);
```

```
    return 0;
```

```
}
```

Строки

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

int main (void)
{
    char string1[80], string2[80], smp[3] = "BMK";
    fgets (string1, 80, stdin); string1[strlen (string1)-1] = '\0';
    fgets (string2, 80, stdin); string2[strlen (string2)-1] = '\0';

    printf ("Строки имеют длину: первая %d, вторая %d\n,
           strlen (string1), strlen (string2));

    if (!strcmp (string1, string2))
        printf ("строки равны\n");

    strncat (string1, string2, 80 - strlen (string1) - 1);
    printf ("%s\n", string1);

    sprintf (string1, "Привет, %s", smp);
    puts (string1);
    return 0;
}
```

Строки

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

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

```
int main (void)
```

```
{
```

```
    char string1[80], string2[80], smp[4] = "BMK";
```

```
    fgets (string1, 80, stdin); string1[strlen (string1)-1] = '\\0';
```

```
    fgets (string2, 80, stdin); string2[strlen (string2)-1] = '\\0';
```

```
    printf ("Строки имеют длину: первая %d, вторая %d\\n,  
           strlen (string1), strlen (string2));
```

```
    if (!strcmp (string1, string2))
```

```
        printf ("строки равны\\n");
```

```
    strncat (string1, string2, 80 - strlen (string1) - 1);
```

```
    printf ("%s\\n", string1);
```

```
    sprintf (string1, "Привет, %s", smp);
```

```
    puts (string1);
```

```
    return 0;
```

```
}
```

Операция `sizeof`

- ◆ Одноместная операция `sizeof` позволяет определить длину операнда в байтах.
 - Операнды – типы либо переменные.
 - Результат имеет тип `size_t`
- ◆ Операция `sizeof` выполняется во время компиляции, ее результат представляет собой константу.
- ◆ Операция `sizeof` помогает улучшить переносимость программ.
- ◆ Для определения объема памяти в байтах, нужного для двумерного массива:

```
number_of_bytes = d1 * d2 * sizeof (element_type)
```

где `d1` – количество элементов по первому измерению,
`d2` – количество элементов по второму измерению,
`element_type` – тип элемента массива.
- ◆ Можно поступить и проще:

```
number_of_bytes = sizeof (имя_массива)
```

Операция `sizeof`

- ◆ `sizeof` можно применять только к «полностью» определенным типам.

Для массивов это означает:

- ◆ размерности массива должны присутствовать в его объявлении
- ◆ тип элементов массива должен быть полностью определен.

- ◆ Пример. Если объявление массива имеет вид:

```
extern int arr[];
```

то операция `sizeof (arr)` ошибочна, так как у компилятора нет возможности узнать, сколько элементов содержит массив `arr`.

Операция sizeof

◇ Пример:

```
#include <stdio.h>
#include <string.h>
int main (int argc, char **argv)
{
    char buffer[10];

    /* копирование 9 символов из argv[1] в buffer;
       sizeof (char) равно 1, число элементов массива
       buffer равно его размеру в байтах */
    strncpy (buffer, argv[1],
             sizeof (buffer) - sizeof (char));
    buffer[sizeof (buffer) - 1] = '\0';
    return 0;
}
```