

**Курс «Алгоритмы и алгоритмические языки»
1 семестр 2015/2016**

Лекция 8

Символьный тип данных (char)

Символьные данные представляются в некотором коде. Популярным кодом является ASCII (American Standard Code for Information Interchange).

- ◆ Каждому символу сопоставляется его код – число типа `char`
- ◆ Требуется, чтобы в кодировке присутствовали маленькие и большие английские буквы, цифры, некоторые другие символы
- ◆ Требуется, чтобы коды цифр 0, 1, ..., 9 были последовательны
- ◆ К символьным данным применимы операции целочисленных типов (но обычно – ***операции отношения и сравнения***)
- ◆ Каждый символ, представляющий самого себя, заключается в одинарные кавычки ' и '
- ◆ Последовательность символов (строка) заключается в двойные кавычки " и "
- ◆ Специальные (управляющие) символы представляются последовательностями из двух символов. Примеры:
 - ◆ `\n` переход на начало новой строки
 - ◆ `\t` знак табуляции
 - ◆ `\b` возврат на один символ с затиранием

Символьный тип данных (char)

Таблица ASCII

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
0	\0									\t	\n						
1												ESC					
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/	
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?	
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_	
6	'	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	DEL	

Символьный тип данных (char)

- ◇ В коде *ASCII* буквы верхнего и нижнего регистра составляют непрерывные последовательности:
между **a** и **z** (соответственно, между **A** и **Z**) нет ничего, кроме букв, расположенных в алфавитном порядке.
- ◇ Это же верно и для цифр 0, 1, ..., 9
- ◇ Преобразование строки символов цифр **s** в целое число (верно для любой кодировки символов)

```
int atoi (char s[])
{
    int i, n;
    n = 0;
    for (i = 0; s[i] >= '0' && s[i] <= '9'; ++i)
        n = 10 * n + (s[i] - '0');
    return n;
}
```

Массивы

- ◇ Массивы позволяют организовывать непрерывные последовательности нескольких однотипных элементов и обращаться к ним по номеру (индексу).
- ◇ Элементы массивов располагаются в памяти последовательно и индексируются с 0:

```
int a[30]; /* элементы a[0], a[1], ... , a[29] */
```

- ◇ Все массивы – одномерные, но элементом массива может быть массив:

```
int b[3][3]; /* элементы b[0][0], b[0][1], b[0][2],  
                    b[1][0], b[1][1], b[1][2],  
                    b[2][0], b[2][1], b[2][2]  
                    */
```

- ◇ Контроль правильности индекса массива **не производится!**
- ◇ **Пример.** Программа, подсчитывающая количество вхождений в строку (текст) каждой из десяти цифр (`ndigit[10]`), пробельных символов (`nwhite`) и остальных символов (`nother`).

Массивы

```
#include <stdio.h>
int main (void)
{
    int c, i, nwhite, nother, ndigit[10];
    nwhite = nother = 0;
    for (i = 0; i < 10; ++i)
        ndigit[i] = 0;
    while (c = getchar ()) != EOF)
        if (c >= '0' && c <= '9')
            ++ndigit[c - '0'];
        else if (c == ' ' || c == '\n' || c == '\t')
            ++nwhite;
        else
            ++nother;
    printf ("digits=");
    for (i = 0; i < 10; ++i)
        printf (" %d", ndigit[i]);
    printf (" , white space=%d, other=%d\n", nwhite, nother);
    return 0;
}
```

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;
}
```

Строки

```
#include <stdio.h>
#include <string.h>
int main (void)
{
    char string1[80], string2[80], smp[4ping ] = "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;
}
```

Если `string1` – "Здравствуй, ", а `string2` – "мир!", результат:

Строки имеют длину 12 4

Здравствуй, мир!

Привет, BMK

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;
}
```