Семинар #5: Строки. Классные задачи.

Таблица ASCII

Символ	Код	С	К	С	К	С	К	С	К	С	К	С	К	С	К	С	К	С	К
\0	0	&	38	0	48	:	58	D	68	N	78	X	88	b	98	l	108	v	118
\t	9	’	39	1	49	;	59	E	69	O	79	Y	89	c	99	m	109	w	119
\n	10	(	40	2	50	<	60	F	70	P	80	Z	90	d	100	n	110	x	120
		)	41	3	51	=	61	G	71	Q	81	[	91	e	101	o	111	y	121
(пробел)	32	*	42	4	52	>	62	H	72	R	82	\	92	f	102	p	112	z	122
!	33	+	43	5	53	?	63	I	73	S	83	]	93	g	103	q	113	{	123
"	34	,	44	6	54	@	64	J	74	T	84	^	94	h	104	r	114	|	124
#	35	-	45	7	55	A	65	K	75	U	85	_	95	i	105	s	115	}	125
$	36	.	46	8	56	B	66	L	76	V	86	‘	96	j	106	t	116	~	126
%	37	/	47	9	57	C	67	M	77	W	87	a	97	k	107	u	117		

Часть 1: Символы.

Тип `char`. Спецификатор `%c` в функции `printf`

Тип `char` – это тип целочисленных чисел размером 1 байт. Часто используется для хранения кодов символов. Для считывания и печати чисел типа `char` используется спецификатор `%hhi`. Функция `printf` со спецификатором `%c` принимает на вход число и печатает соответствующий символ по таблице `ASCII`.

```
char a = 64;
printf("%hhi\n", a);
printf("%c\n", a);
```

Спецификатор `%c` в функции `scanf`

Функция `scanf` со спецификатором `%c` считывает 1 символ и записывает код `ASCII` этого символа по соответствующему адресу.

```
char a;
scanf("%c", &a);
printf("%c\n", a);
```

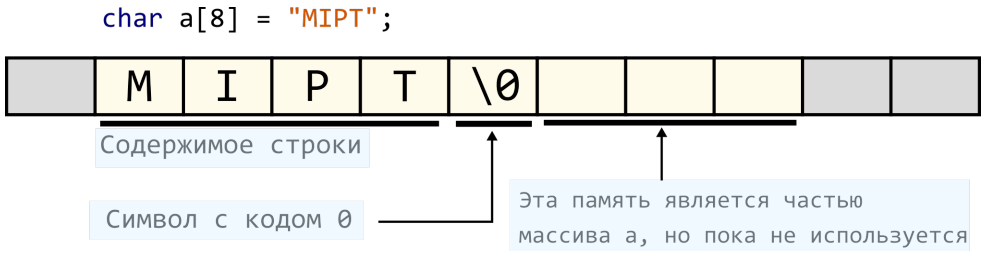
Символьные константы

Для удобства работы с символами с языке были введены символьные константы. В коде они выглядят как символы в одинарных кавычках, но являются просто числами, соответствующими коду символа.

```
int a = '@'; // Теперь a равно 64
int b = '5'; // Теперь b равно 53
printf("%i %i\n", a, b);
```

Часть 2: Строки:

Строки - это массивы чисел типа `char`, которые хранят коды символов. Самое значительное отличие строк от массивов это то, что конец строки задаётся как элемент массива символом с кодом `0`.



Объявление, инициализация и изменение строк

Создавать строки можно также как и массивы, а можно и с помощью строки в двойных кавычках.

```
char a[10] = {77, 73, 80, 84, 0};
char b[10] = {'M', 'I', 'P', 'T', '\0'};
char c[10] = "MIPT"; // Символ 0 поставится автоматически
// Использовать = со строками можно только при создании, то есть это работать не будет:
// a = "CAT";
// Изменение элементов строк работает также как и у массивов:
a[1] = 'A';
```

Печать строк. Спецификатор %s

Обычные массивы нельзя печатать одной командой `printf`, но специально для строк ввели модификатор `%s`, благодаря которому можно печатать и считывать строки одной командой.

```
char a[10] = "MIPT";
printf("%s\n", a); // Печатаем строку
```

Считывание строк

Считывать строки можно с помощью `scanf` со спецификатором `%s`. Но если вы введёте слишком большую строку, которая не поместится в массив, то произойдёт ошибка – выход за границы массива.

```
char a[100];
scanf("%s", a); // Считываем строку
```

Безопасное считывание строк

Чтобы указать `scanf` сколько символов можно считывать:

```
char a[100];
scanf("%99s", a); // Безопасно считываем строку
```

Часть 3: Строки и функции

Строки передаются в функции также как и массивы. То есть при изменении строки внутри функции она меняется и снаружи. Но только при передаче строки не обязательно передавать её размер, так как граница строки задаётся нулевым символом. Пример функции, которая заменяет один символ в строке на другой:

```
#include <stdio.h>
void change_letter(char* str, char from, char to)
{
    int i = 0;
    while (str[i])
    {
        if (str[i] == from)
            str[i] = to;
        i++;
    }
}

int main()
{
    char a[100] = "Cats and dogs";
    printf("%s\n", a);

    change_letter(a, 'a', '#');
    printf("%s\n", a);
}
```

Считывание до заданного символа

Как вы могли заметить, использование `scanf` с модификатором `%s` считывает до первого пробельного символа. Чтобы считать всю строку (то есть до символа `'\n'`), следует использовать модификатор `%[^\n]`.
Пример программы, которая считывает строку и меняет пробелы на переносы строк:

```
#include <stdio.h>
int main()
{
    char str[100];
    scanf("%[^\n]", str);
    for (int i = 0; str[i]; i++)
    {
        if (str[i] == ' ')
            str[i] = '\n';
    }
    printf("%s\n", str);
}
```

Часть 4: Стандартные функции библиотеки `string.h`:

- `size_t strlen(const char* str)` - возвращает длину строки
- `char* strcpy (char* a, const char* b))` - копирует строку `b` в строку `a`, т.е. аналог `a = b`.
- `char* strcat(char* a, const char* b)` - приклеивает копию строки `b` к строке `a`, т.е. аналог `a += b`.
- `int strcmp(const char* a, const char* b)` - лексикографическое сравнение строк (возвращает `0`, если строки одинаковые, положительное, если первая строка больше, и отрицательное, если меньше)

```
#include <stdio.h>
#include <string.h>
int main()
{
    char a[100] = "Cat";
    char b[100] = "Dog";

    // Строки это массивы, поэтому их нельзя просто присваивать
    a = b; // Это не будет работать! Нужно использовать strcpy:
    strcpy(a, b);

    // Конкатенация ( склейка ) строк. Можно воспринимать как +=
    a += b; // Это не будет работать! Нужно использовать strcat:
    strcat(a, b);

    // Строки это массивы, поэтому их нельзя просто сравнивать
    if (a == b) {...} // Это не будет работать! Нужно использовать strcmp:
    if (strcmp(a, b) == 0)) {...}
}
```

Также могут быть полезны следующие функции из библиотеки `stdio.h`:

- `sprintf` - аналог `printf`, но вместо печати на экран, 'печатает' в строку.
- `sscanf` - аналог `scanf`, но вместо считывания на экран, 'считывает' из строки.

Часть 5: Аргументы командной строки

Программы могут принимать аргументы. Простейший пример – утилита `ls`. Если запустить `ls` без аргументов:

```
ls
```

то она просто напечатает содержимое текущей директории. Если же использовать эту программу с опцией `-l`:

```
ls -l
```

то на экран выведется подробное описание файлов и папок в текущей директории. В данном примере опция `l"` является аргументом командной строки.

В случае передачи информации программе через аргументы командной строки, информация передаётся при вызове программы. Чтобы передать что-либо программе через аргументы командной строки, нужно написать это в терминале при запуске программы сразу после её запуска.

Например, если мы хотим передать программе `a.out` строку `cat`, то программу нужно вызвать так:

```
./a.out cat
```

Если же мы хотим передать программе `a.out` число `123`, то программу нужно вызвать так:

```
./a.out 123
```

Только нужно помнить, что аргументы командной строки всегда воспринимаются как строки и в данном случае число `123` передастся как строка `"123"`.

Чтение из файла

Пример программу, которая подсчитывает количество слов в файле.

```
#include <stdio.h>
#include <string.h>
int main()
{
        // Считывание слов из файла
        FILE* infile = fopen("words.txt", "r");
        char words[500][100];
        int number_of_words = 0;
        while (fscanf(infile, "%s", words[number_of_words]) != -1) {
                number_of_words++;
        }
        fclose(infile);
}
```

- **Чтение из файла:** Напишите программу, которая будет считывать слова из файла и записывать их в массив `char words[1000][100]`. Когда в файле слов для считывания не останется, функция `fscanf` будет возвращать `-1`. После этого все слова должны быть напечатаны на экран через пробел.
- **Сортировка слов:** Напишите программу, которая будет считывать слова из файла и записывать их в массив `words`. После этого все слова должны быть отсортированы по алфавиту и записаны в файл `sorted_words.txt`.