

# Строки символов



# Массивы символов и строки

Массив символов — массив (одномерный или многомерный), состоящий из элементов типа `char`.

```
char cm[5]={'a','b','c','d','e'};
```

```
или char cm[]={ 'a','b','c','d','e'};
```

```
или char cm[5];
```

потом `for()` для заполнения.

Для получения символа с клавиатуры — `getchar()`.

Строка — массив символов, у которого последний элемент всегда — символ с кодом 0 (`'\0'`). **«Полезных» символов в строке всегда на 1 меньше, чем элементов массива.**

```
char s1[6]={'H','e','l','l','o','\0'};
```

`'\0'` - «ограничитель», «нуль-терминатор» и т. п. - признак окончания строки (в отличие от обычных массивов).



# Особенности строк в Си

- Коды символов в составе строки — от 0 до 127 (код 0 — только в конце!).
- Специальные символы (`\n`, `\t`, `\b`) и пробелы тоже могут быть в строке.
- Количество символов в строке (длина строки) в общем случае неизвестно.
- Т.к. строка — вариант массива, нужно заранее выделять место в памяти.
- При получении строк с устройств ввода нужно всегда контролировать, не превышен ли максимальный размер массива.
- `s1="Hello World!"` — в «теле» функции нельзя, т. к. массивы нельзя присваивать!



# Инициализация и вывод строк

```
char s1[]={ 'H', 'e', 'l', 'l', 'o', '\0' };
```

```
char s1[MAXLEN]="Hello";
```

```
puts("Hello");  
puts(s1);
```

```
printf("Hello");  
printf("Hello\n");  
printf("Hello string: %s\n", s1);  
printf("Hello string: %30s\n", s1);
```

См. пример lect-07-01.c



# Получение строк с клавиатуры

`gets()` – определена в K&R C, C89 (обратная к `puts()`)  
В C99 объявлена «запрещенной», в C11 исключена из стандартной библиотеки.

Проблема — не контролируется длина прочитанной строки →  
возможен выход за пределы массива.  
см. пример `lect-07-02.c`.

`scanf()` – для строк спецификатор формата `"%s"`, тоже есть нюанс:  
длину строки нужно указывать явно (числом).

См. примеры `lect-07-03.c`, `lect-07-03a.c`, `lect-07-03b.c` – переменные  
смешиваются.



# Получение строк с клавиатуры

## Вариант 1. Написать собственную функцию (`new_gets()`).

Варианты окончания строки (текста)

- Символ перевода строки
- Граница массива (предельное количество элементов)
- Символ конца файла (код -1, EOF)

`new_gets()` должна проверять предельную длину, заканчивать строку «терминатором» при заполнении массива и при нажатии на <ENTER>

см. пример `lect-07-04.c`.

Чего добились: предельная длина стала параметром.

Можно тут же посчитать реальную длину — пример `lect-07-04a.c`.



# Получение строк с клавиатуры

## Вариант 2. Изучить особенности ввода/вывода в Си.

В `stdio.h` определены стандартные потоки ввода (`stdin`) и вывода (`stdout`).

`stdin` «по умолчанию» связан с клавиатурой

`stdout` «по умолчанию» связан с окном вывода («терминалом», «консолью»).

В подходе POSIX любой информационный объект, из которого можно считывать данные или в который можно записывать данные (включая потоки ввода-вывода), является файлом.

`stdin` и `stdout` — указатели на специальные файлы, соответствующие «стандартным» устройствам ввода и вывода.

( см. также <http://www.amse.ru/courses/cpp1/2010.03.03.html> )



# Получение строк с клавиатуры

`fgets(<строка>, <макс. длина>, stdin)` – получение строки с клавиатуры с контролем предельной длины – возвращает указатель на строку.

Символ перевода строки (`\n`) тоже включается в строку!

Пример `lect-07-05.c`

Разумная предельная длина строки:

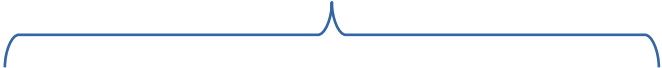
- Количество позиций в окне вывода (80)
- Максимальная длина строки в текстовых редакторах (1-4 кбайта)

Реальную длину строки придется как-то определять (`new_gets()` это делает).

Пример `lect-07-06.c`

Строка после `fgets()`

Длина строки



'H'	'e'	'l'	'l'	'o'	'\n'	'\0'
0	1	2	3	4	5	6





# Функции для работы со строками

Описаны в `string.h`

- `strlen(s)` — длина строки `s` («полезные» символы), тип `int`
- `strcmp(s1,s2)` — сравнение строк «как в словаре» (лексикографическое), тип `int`
- `strncmp(s1,s2,n)` — лексикографическое сравнение первых `n` байт, тип `int`
- `strcpy(s1,s2)` — копирование `s2` в `s1`, тип «строка» (`char*`)
- `strncpy(s1,s2,n)` — копирование `n` байт из `s2` в `s1`, тип «строка»
- `memcpy(s1,s2,n)` — копирование `n` байт из `s2` в `s1`, тип `void*`.

См. описание функций `string.h`, реализацию функций у K&R.

Примеры `lect-07-06a.c`, `lect-07-07.c`, `lect-07-08.c`.



# Функции для работы со строками

Примеры реализации функции копирования строк:

lect-07-09.c — с ограничением длины

lect-07-09a.c — с использованием арифметики указателей, цикл `while()`

lect-07-09b.c — с использованием арифметики указателей, цикл `for()`.

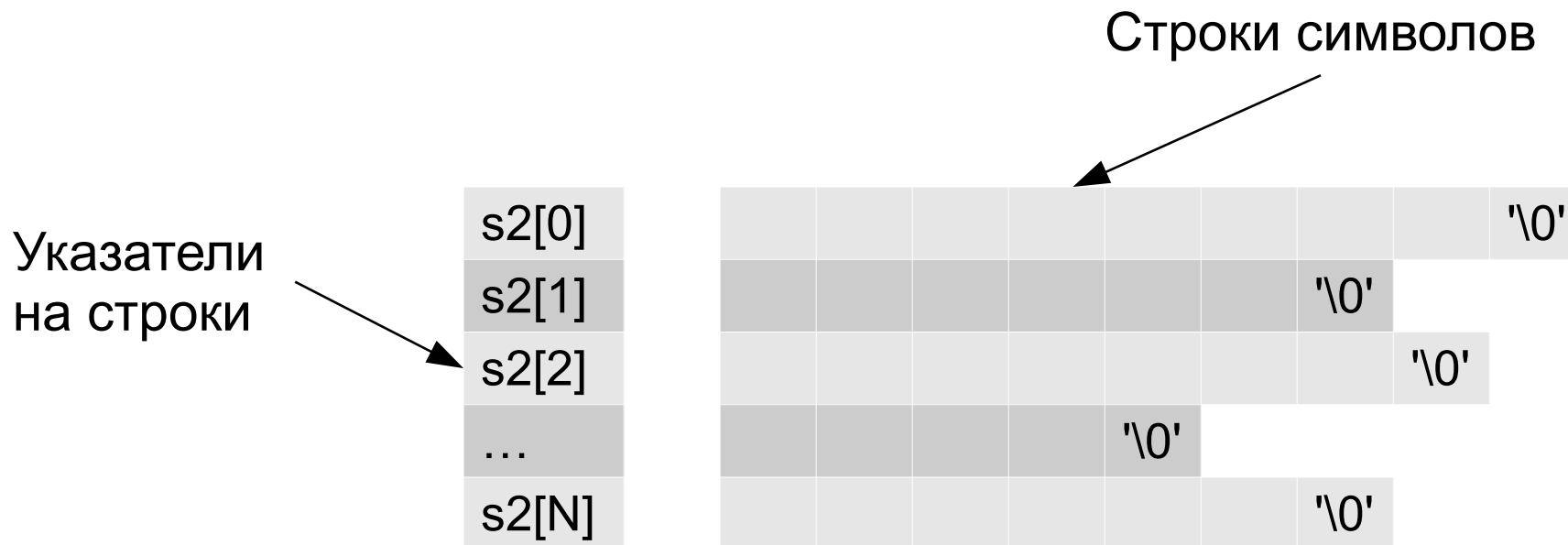


# Массив строк (текст)

Реальный текст — массив строк (абзацев), каждая строка текста заканчивается переводом строки (`\n` или `\n\r` коды 10 или 10 и 13).

Количество символов в строках текста — разное.

Каждая строка текста — одномерный массив символов с «нуль-терминатором».



# Строки в динамической памяти

## Если строка одна:

Принципы работы — как с одномерным массивом с учетом нуль-терминатора.

- Выделяем память для MAXLEN байтов, проверяем, что память выделена. Если память выделена, то
  - Получаем строку
  - Работаем со строкой
  - Освобождаем память.
- Если память не выделена, то не работаем.

`realloc()` не рекомендуется (фрагментация «кучи», лишние операции).

Пример lect-07-10.c .



# Строки в динамической памяти

## Если строк несколько (N):

Принципы работы — как с двумерным массивом, в котором одномерные массивы имеют разную длину.

- Выделяем память для N указателей на строки
- Выделяем память для MAXLEN байтов, проверяем, что память выделена. Если память выделена, то
  - Получаем строку
  - Работаем со строкой (записываем в элемент массива)
  - После завершения обработки освобождаем память для каждого элемента массива в цикле и затем для всего массива, все указатели — в NULL.
- Если память не выделена, то не работаем.

Примеры lect-07-11.c, lect-07-11a.c (требуется анализ!!!).



# Функции, полезные при обработке строк

См. `ctype.h`

- `isalpha()` – возвращает ненулевое значение (`true`), если её аргумент является буквой, в противном случае возвращается нуль (`false`)
- `isdigit()` – возвращает ненулевое значение (`true`), если её аргумент является десятичной цифрой, в противном случае возвращается нуль (`false`).
- `ispunct()` – возвращает ненулевое значение (`true`), если её аргумент является знаком препинания (любой печатаемый символ, отличный от пробела или алфавитно-цифрового символа), в противном случае возвращается нуль (`false`).

Примеры реализации: `lect-07-12.c`, `lect-07-13.c` (на самом деле всё не так 😊).

