Лекция 11. Символьные строки и строковые функции. Часть 2.

Ревун Артем Леонидович
ст. преп. кафедры вычислительных систем

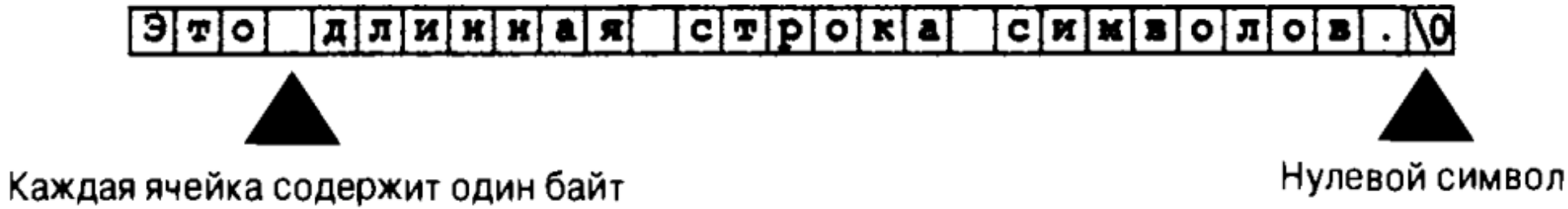
Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)

Содержание

- Разбор функций вывода строк, анализ.
- Строковые функции: strcat(), strncat(), strcmp(), strncmp(), strcpy(), strncpy(), sprintf(), strchr()
- Применение строковых и символьных функций из библиотеки C и создание собственных строковых функций
- Использование аргументов командной строки

Символьная строка (Си-style)

Символьная строка в языке Си – это последовательность из одного или большего количества символов, которая заканчивается управляющим символом '\0';
'\0' – называют нулевым символом строки



Сравнение трёх функций ввода

```
char words[10];
gets(), fgets() и gets_s() – сравним;
< Экзамен
    √       √       √
< Кафедра вычислительных систем
    x       √       √(end)
< Очень важная информация которую мы вводим.
    x       √     x
```

Вывод строк: функция puts() (1)

Функция проста в использовании, ей передаётся указатель на строку, при этом к строке будет добавлен символ '\n'.

```
int puts ( const char * str );
```

Вывод строк: функция puts() (2)

```
#define DEF "Я - строка, определенная
директивой #define.«
int main(void){
char str1[80] = "Массив был инициализирован
моим значением.";
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); …
```

Вывод строк: функция puts() (3)

```
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.
```

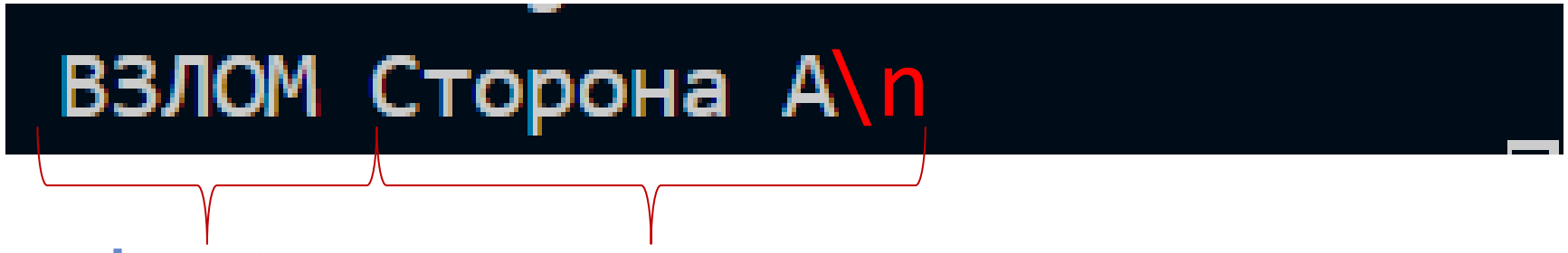
Вывод строк: функция puts() (4)

```
char side_a[] = "Сторона А";
char dont[] = {'В', 'З', 'Л',
'О','М', ' '} ;
char side_b[] = "Сторона Б";
puts(dont);
```

Какой будет результат?

Вывод строк: функция puts() (5)

```
char side_a[] = "Сторона А";
char dont[] = {'В', 'З', 'Л',
'О','М', ' '} ;
char side_b[] = "Сторона Б";
puts(dont);
```



Вывод строк: функция fputs() (1)

Данная функция в отличии от функции puts() ориентирована на файловый ввод. Однако, она не добавляет вывод символом '\n'; А также принимает второй аргумент указывающий на файл, в который должна осуществляться запись. Для вывода на экран применяется **stdout**.

```
int fputs ( const char * str, FILE * stream );
```

Вывод строк: функция fputs() (2)

```
char line[81];
while (gets(line)) puts(line);
```

```
gets не безопасно, но я попробую
gets не безопасно, но я попробую
```

```
char line[81];
while (fgets(line, 81, stdin)) fputs(line, stdout);
```

```
fgets безопаснее, и этого изменения достаточно
fgets безопаснее, и этого изменения достаточно
```

Функция gets() рассматривается только для того, что бы вы знали, как она работает, вовсе не призываем пользоваться ею.

Вывод строк: функция printf() (1)

Функция не однократно рассмотрена, она более универсальная в сравнение с предыдущими функциями, т.к. работает с разными типами данных. Она чуть медленнее, но не на столько, чтобы вы это заметили. Функция упрощает вывод, хоть и его качество полностью зависит от пользователя.

```
int printf(const char * format, ... );
```

Строковые функции: strlen() (1)

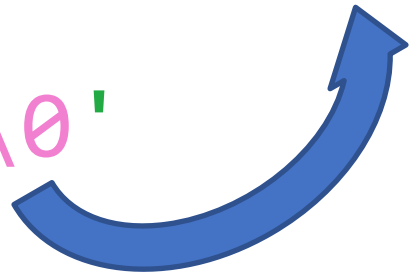
Описанные далее функции принадлежат библиотеке **string.h**

Данная функция возвращает длину строки в виде типа данных **size_t**.

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

Строковые функции: strlen() (2)

```
void fit(char *string, unsigned int size)
{
if (strlen(string) > size)
    string [size] = '\0';
}
fit("Короткая строка", 8);
```




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

Строковые функции: strcat() (1)

Название образовано от **str**ing con**cat**enation — конкатенация строк) в качестве параметров предаются указатели на две строки `destination` и `source`. Копия `source` присоединяется в конец `destination`, и такая объединенная версия становится новой `destination`. При этом `source` не изменяется.

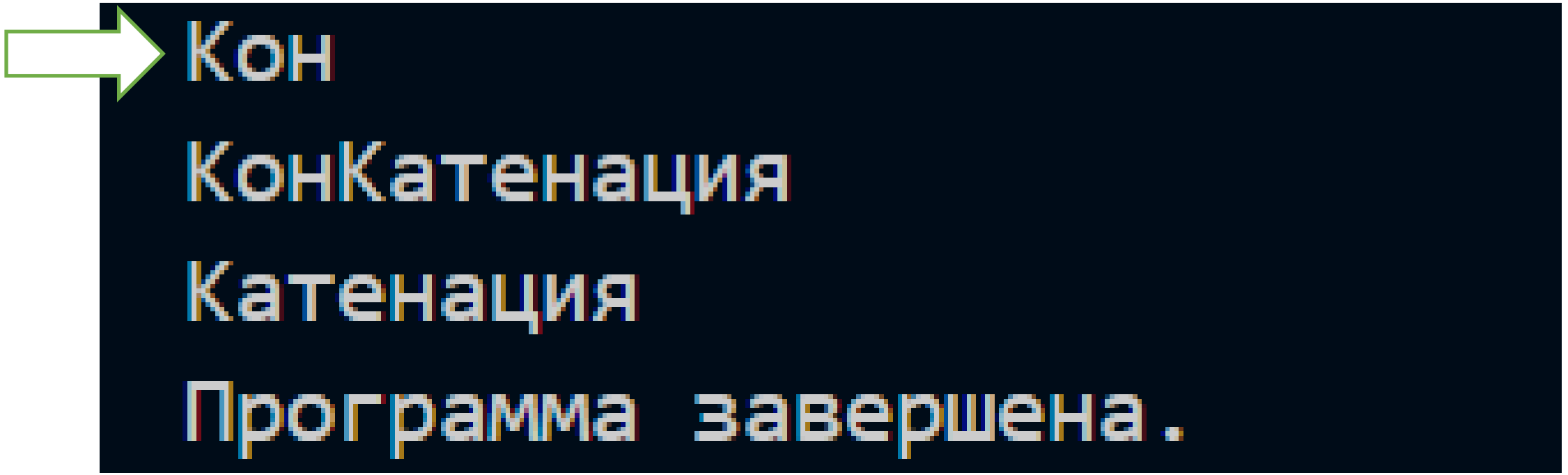
```
char * strcat (char * destination,
const char * source);
```

Строковые функции: strcat() (2)

```
char str1[40];
scanf("%5s", str1);
char str2[] = "Катенация";
strcat(str1, str2);
puts(str1);
puts(str2);
puts("Программа завершена.");
```

```
char * strcat (char * destination,
const char * source);
```

Строковые функции: strcat() (3)




```
char * strcat (char * destination,
const char * source);
```

Строковые функции: strncat() (1)

Функция **strcat()** не проверяет, умещается ли вторая строка в первый массив. Если вы не выделите достаточного пространства до первого массива, то столкнетесь с проблемой переполнения соседних ячеек памяти избыточными символами. **strncat()** альтернатива функции **strcat().** Она принимает аргумент num который должен указывать максимальное количество добавляемых символов.

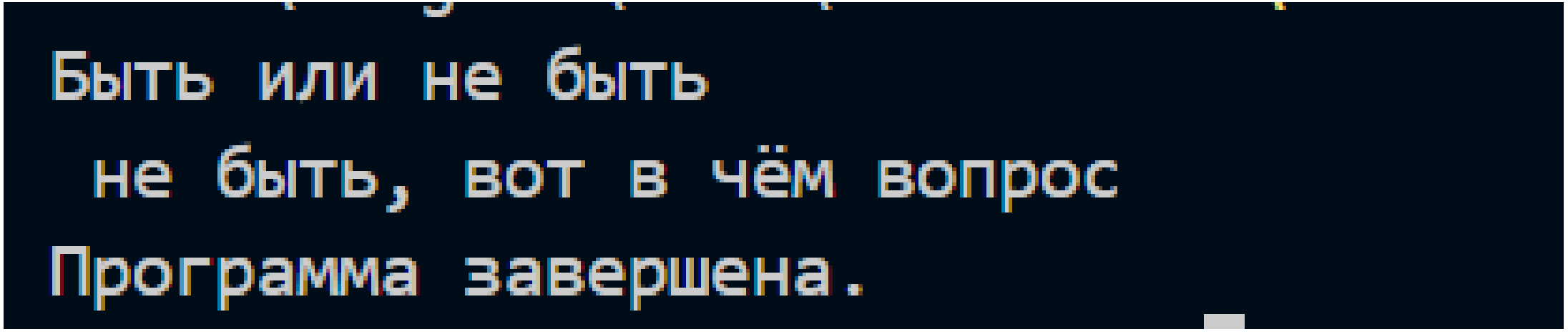
```
char * strncat (char * destination,
const char * source, size_t num );
```

Строковые функции: strncat() (2)

```
char str1[10]; char str2[30];
strcpy (str1,"Быть или");
strcpy (str2," не быть, вот в чём
вопрос");
strncat (str1, str2, 8);
puts (str1);puts (str2);
puts("Программа завершена.");
```

```
char * strncat (char * destination,
const char * source, size_t num );
```

Строковые функции: strncat() (3)




```
char * strncat (char * destination,
const char * source, size_t num );
```

Строковые функции: strncat() (4)

```
char * strncat (char * destination,
const char * source, size_t num );
```

Стоит ли использовать ?

```
char * strcat (char * destination,
const char * source);
```

Строковые функции: strcpy() (1)

Функция позволяет скопировать содержимое `source` строки в строку `destination`. Функция не осуществляет проверку, помещается `source` в `destination` или нет. Представляет собой эквивалент **операции присваивания =**.

В контексте строк Си-style, нет возможности скопировать значение одной строки в другую, т.к. это приводит к копирования адреса одной и той же строки.

```
char * strcpy (char * destination,
const char * source);
```

Строковые функции: strcpy() (2)

```
#define SIZE 40
#define LIM 5
char qwords[LIM][SIZE]; char temp[SIZE]; int i = 0;
printf("Введите %d слов, которые начинаются с буквы
c :\n", LIM);
while (i < LIM && fgets(temp, SIZE, stdin)){
if (temp[0]!='c')
printf("%s не начинается с буквы с !\n ", temp);
else{strcpy(qwords[i], temp);      i++;  }   }
puts("Список принятых слов:");
for (i = 0; i < LIM; i++)
        fputs(qwords[i], stdout);
```

Строковые функции: strcpy() (3)

```
> Введите 5 слов, которые начинаются с буквы с :
< среди
< сплошных
< снегов
< сурового
< севера
> Список принятых слов:
> среди
> сплошных
> снегов
> сурового
> севера
```

Строковые функции: strcpy() (3)

```
char * str;
strcpy(str, "Невозмутимость С");
```

Каков результат компиляции? Выполнения?

```
char * strcpy (char * destination,
const char * source);
```

Строковые функции: strcpy() (4)

```
char * str;
strcpy(str, "Невозмутимость С");
```

Каков результат компиляции? Выполнения?

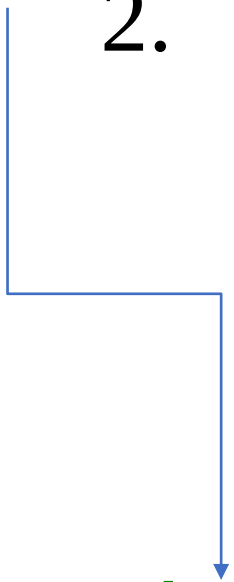
Однако не стоит забывать, что указатель `destination` должен указывать на объект данных, а `source` может быть объявленным указателем, именем массива или строковой константой.

```
char * strcpy (char * destination,
const char * source);
```

Строковые функции: strcpy() (5)

Так же следует обратить внимание на:

1. Она возвращает значение своего первого аргумента — адрес символа.
2. Первый аргумент не обязательно должен указывать на начало массива; это позволяет копировать только часть ~~массива~~ (строки).



```
char * strcpy (char * destination,
const char * source);
```

Строковые функции: strcpy() (6)

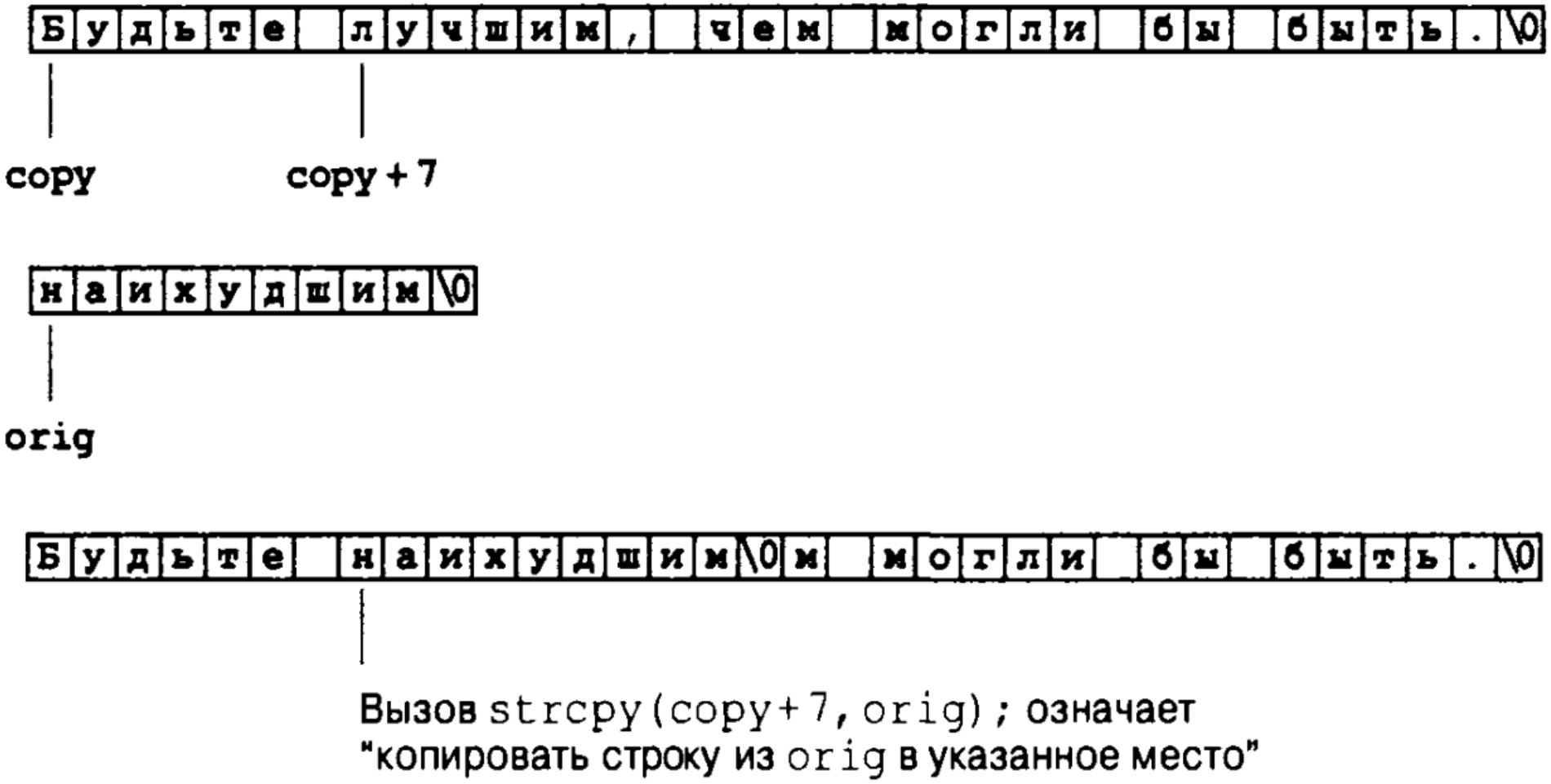
```
#define WORDS "наихудшим"
#define SIZE 40
const char * orig = WORDS;
char copy[SIZE] = "Будьте лучшим, чем
могли бы быть."; char * ps;
puts(orig); puts(copy);
ps = strcpy(copy + 7, orig);
puts(copy); puts(ps);
```

Каков будет результат программы?

Строковые функции: strcpy() (7)

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

Строковые функции: strcpy() (8)



Строковые функции: strncpy() (1)

Более безопасный способ копирования строк предусматривает применение данной функции. Эта функция принимает аргумент num, в котором указывается максимальное количество копируемых символов.

```
char * strncpy ( char * destination,
const char * source, size_t num );
```

Строковые функции: strncpy() (2)

Вызов функции копирует вплоть до `num` символов либо до появления '\0' (в зависимости от того, что произойдет раньше) из `source` в `destination`. Следовательно, если количество символов в `source` меньше, копируется вся строка, включая '\0'. Эта функция никогда не копирует более `num` символов, так что если данный лимит исчерпан до достижения конца исходной строки, то '\0' не добавляется.

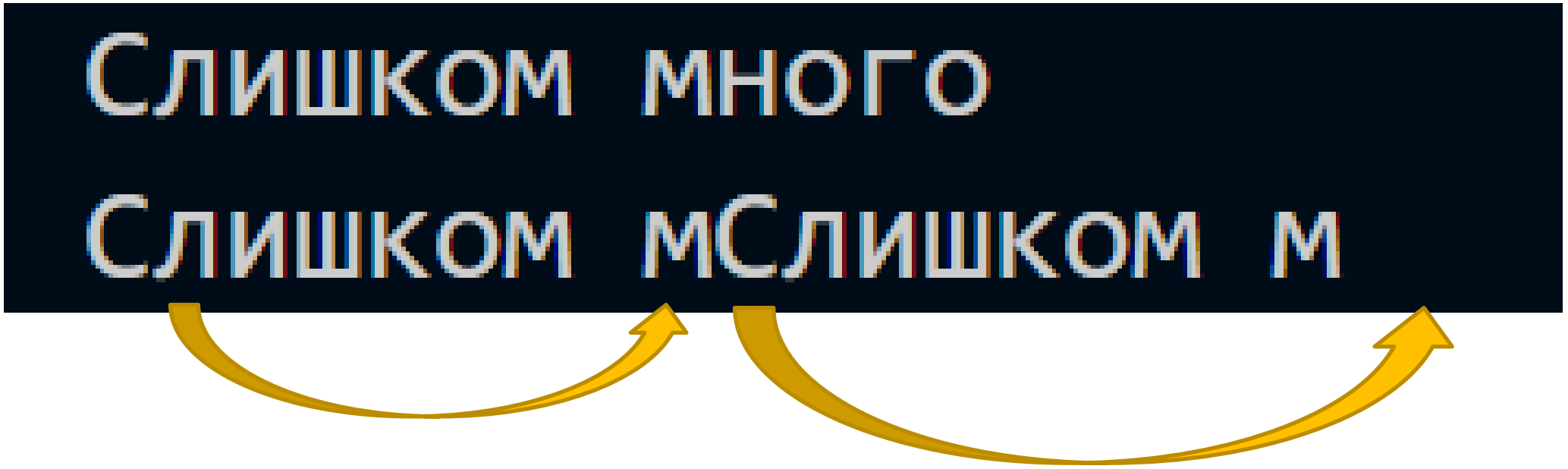
```
char * strncpy ( char * destination,
const char * source, size_t num );
```

Строковые функции: strncpy() (3)

```
#define SIZE 10
char word1[SIZE]; char word2[SIZE];
char temp[SIZE];
fgets(temp, SIZE, stdin);
strncpy(word1, temp, SIZE);
strncpy(word2, temp, SIZE - 1);
word2[SIZE - 1] = '\0';
fputs(word1, stdout);
fputs(word2, stdout);
```

Строковые функции: strncpy() (4)

```
#define SIZE 10
```




```
char * strncpy ( char * destination,
const char * source, size_t num );
```

Строковые функции: strcmp() (1)

Название функции образовано (от **str**ing **co**m**p**arison — сравнение строк). По аналогии с функциями сравнения для чисел (<, >, ==, <=, >=) функция сравнивает строки.
Если `str1 == str2`, возвращает 0;
Если `str1` > в лексикографическом порядке, возвращает <0
Если `str1` < в лексикографическом порядке, возвращает >0

```
int strcmp (const char * str1,
const char * str2);
```

Строковые функции: strcmp() (2)

Лексикографический порядок – предполагает порядок по алфавиту, в данном случае, относительно кодов ASCII.

```
strcmp("A", "A") возвращает 0                    65 - 65
strcmp("A", "B") возвращает -1                   65 - 66
strcmp("B", "A") возвращает 1                    66 - 65
strcmp("C", "A") возвращает 2                    67 - 65
strcmp("Z", "a") возвращает -7                   90 - 97
strcmp("apples", "apple") возвращает 115   115 - 0
```

```
int strcmp (const char * str1,
const char * str2);
```

Строковые функции: strncmp() (1)

Функция сравнивает строки до тех пор, пока не найдет пару соответствующих символов, которые отличаются друг от друга, и этот поиск может продолжаться вплоть до достижения конца одной из строк. Функция сравнивает строки до тех пор, пока нс обнаружит в них различия либо пока не сравнит num символов в обеих строках.

```
int strncmp (const char * str1,
const char * str2,
size_t num);
```

Строковые функции: strncmp() (2)

```
#define LISTSIZE 6
const char * list[LISTSIZE] ={
"астрономия", "астатиэм","астрофизика",
"остракизм", "астеризм", "астролябия"};
int count = 0; int i;
for (i = 0; i < LISTSIZE; i++)
if (strncmp(list[i],"астро", 5) == 0) {
printf("Найдено: %s\n", list[i]); count++;}
printf("Количество слов в списке,"
" начинающихся с астро: %d\n", count);
```

Строковые функции: strncmp() (3)

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

Строковые функции: ещё больше (1)

- `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.

- `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.

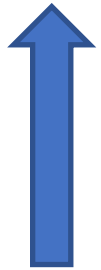
- `char *strstr(const char * s1, const char * s2);`

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

Аргументы командной строки (1)

Аргументы командной строки — предоставляют возможность передать значения в программу на языке Си до её вызова.

```
int main(int argc, char const *argv[])
```



`argc` ссчётчик аргументов

`argv` массив из `argc` значений типа char[]

Аргументы командной строки (2)

```
int main(int argc, char *argv[]){
int count;
printf("Количество аргументов, указанных в
командной строке: %d\n", argc - 1);
for (count = 1; count < argc; count++)
printf("%d: %s\n", count, argv[count]);
return 0;
}
```

```
                                .\main.exe Arguments of command prompt
Количество аргументов, указанных в командной строке: 4
1: Arguments
2: of
3: command
4: prompt
```

Аргументы командной строки (3)

```
                                .\main.exe 45 "command prompt" q
Количество аргументов, указанных в командной строке: 3
1: 45
2: command prompt
3: q
```

Аргументы командной строки (4)

При передаче аргументов через командную строку, гарантированно их восприятие в контексте `char *argv[]`
Однако возникает потребность в преобразовании их в целочисленные значения. Для этой цели может быть использована функция `atoi()` (от **a**lphanumeric **to** **i**nteger)

```
int atoi(const char * str);
```

Аргументы командной строки (5)

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[]){
int i, times;
if (argc < 2 || (times = atoi(argv[1])) < 1)
printf("Использование: %s положительное-численное",
argv[0]);
else
for (i = 0; i < times; i++) puts ("Привет, Мир!") ;
return 0;}
```

```
int atoi(const char * str);
```

Аргументы командной строки (6)

```
.\main.exe
Использование:  \main.exe положительное-число
```

```
.\main.exe 5
Привет, Мир!
Привет, Мир!
Привет, Мир!
Привет, Мир!
Привет, Мир!
```

```
int atoi(const char * str);
```

Аргументы командной строки (6)

Например, `atoi( "42regular" )` возвращает целое число 42.

Например, `atoi( "hello what?")`, аргумент не распознается как числовой и возвращается 0.

В программу включен заголовочный файл **stdlib.h**, потому что, начиная с ANSI C, он содержит объявление функции atoi().

```
int atoi(const char * str);
```

Аргументы командной строки (6)

ANSI C stdlib.h, содержит и другие аналогичные функции:

- **atof** - преобразует строку в значение типа double
- **atol** - преобразует строку в значение типа long
- **strtol** - преобразует строку в значение типа long
- **strtoul** - преобразует строку в значение типа unsigned long
- **strtod** - преобразует строку в значение типа double

strtol и **strtoul** обладают особенностью, позволяют указывать основание системы счисления. ~~bin, oct, hex, int -> Python~~

Задачи на практику

Читать:

1. Глава 11 *Символьные строки и строковые функции* (стр. 419-474).

Изучить:

1. *Вопросы для самоконтроля* (стр. 474)
2. *Пример обработки строк: сортировка строк* (стр. 462)
3. *Символьные функции* ***ctype.h*** *и строки* (стр. 465)

Выполнить:

1. https://eios.sibsutis.ru/mod/resource/view.php?id=168179

Спасибо за внимание

Ревун Артем Леонидович
ст. преп. Кафедры вычислительных систем

Курс «Программирование», осенний семестр, 2024
Сибирский государственный университет телекоммуникаций и информатики (г. Новосибирск)