



ФГОБУ ВПО "СибГУТИ"
Кафедра вычислительных систем

ЯЗЫКИ ПРОГРАММИРОВАНИЯ / ПРОГРАММИРОВАНИЕ

Представление текстовой информации

Преподаватель:

Доцент Кафедры ВС, к.т.н.

Поляков Артем Юрьевич



Представление текстовой информации

Вычислительная техника оперирует только
числовой информацией (в двоичной СС)



Текст необходимо представить
в виде набора чисел



Естественные разбиения текста:
по символам (таблица ASCII), по словам.



Представление текстовой информации (пример)

mom soap frame

+

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
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	{		}	~	DL

||

m	o	m		s	o	a	p		f	r	a	m	e
6D	6F	6D	20	73	6F	61	70	20	66	72	61	6D	65



Хранение текстовой информации (строки)

Описанный способ представления предполагает описание текста в виде **набора однотипных данных**.

Для хранения подобной информации в языках высокого уровня (например, СИ) применяются **массивы – строки**.

Для хранения одного символа в языке СИ предусмотрен тип данных **char**.

Хранение текста, приведенного на предыдущем слайде, **может быть** организовано в массиве:

char text[14];



Операции со строками

- получение символа по номеру позиции (индексу);
- определение длины строки;
- копирование строки;
- проверка на совпадение строк (с учётом или без учёта регистра символов);
- сравнение строк (в алфавитном порядке);
- поиск символа в строке;
- **конкатенация** (соединение) строк: "ab" + "cd" = "abcd";
- получение **подстроки** по индексам начала и конца:
"abcdefghijklmnopqrstuvwxyz", 3, 8 → "defghi" ;



Операции со строками

- проверка вхождения одной строки в другую (**поиск подстроки в строке**): "abcdefghij": "cdef" – да, "cfde" – нет;
- определение подстроки, состоящей только из заданного набора символов.
- разбиение строки на поля (tokens), разделенные заданным набором символов.
- замена подстроки в строке:

"abcdefghij", "cdef" на "cfde" = "ab**cfde**ghij"



Особенности хранения текстовой информации

В чем (с точки зрения хранения) заключается отличие между следующими текстовыми данными?

Actions speak louder than words

Ask no questions and you will be told no lies

Best defence is offence

Let well alone

Каким образом можно организовать хранение информации о размере строки?



Подходы к представлению строк

- **Pascal strings** – представление в виде массива символов, размер которого хранится отдельно:

14	m	o	m		s	o	a	p		f	r	a	m	e
----	---	---	---	--	---	---	---	---	--	---	---	---	---	---

- **NULL-terminated strings (C Strings, ASCIIZ)** – представление в виде массива символов, заканчивающегося специальным допустимым символом – "нуль-терминатором". В языке СИ это символ с кодом 0 (символьная константа '\0'), иногда – число **0xFF** или код символа '\$' (0x24).

m	o	m		s	o	a	p		f	r	a	m	e	\0
---	---	---	--	---	---	---	---	--	---	---	---	---	---	----



Преимущества Pascal-strings

14	m	o	m		s	o	a	p		f	r	a	m	e		...
----	---	---	---	--	---	---	---	---	--	---	---	---	---	---	--	-----

- размер строки известен => быстрые операции конкатенации и получения размера строки;
- строка может содержать любые данные, нет выделенного символа для **нуль-терминатора**;
- возможно автоматическое отслеживание выхода за границы строки при её обработке;
- быстрая операции вида «взятие N-ого символа с конца строки».



Недостатки Pascal-strings

14	m	o	m		s	o	a	p		f	r	a	m	e		...
----	---	---	---	--	---	---	---	---	--	---	---	---	---	---	--	-----

- осложнено хранение и обработка символов произвольной длины (например UTF8);
- дополнительные накладные расходы на хранение длины строки (хранение большого количества строк маленького размера);
- ограничение на максимальный размер строки (зависит от разрядности ячейки для хранения длины)



Преимущества C-strings

m	o	m		s	o	a	p		f	r	a	m	e	\0		...
---	---	---	--	---	---	---	---	--	---	---	---	---	---	----	--	-----

- меньший объем служебной информации о строке;
- представления строки без создания **отдельного типа данных**;
- отсутствуют ограничения на максимальный размер строки;
- простота получения суффикса строки;
- возможность использовать алфавит с переменным размером символа (например, UTF-8).



Недостатки C-strings

m	o	m		s	o	a	p		f	r	a	m	e	\0		...
---	---	---	--	---	---	---	---	--	---	---	---	---	---	----	--	-----

- сложная операция получения длины и конкатенации строк;
- отсутствие средств контроля за выходом за пределы строки (в случае повреждения завершающего байта возможно повреждение больших областей памяти);
- отсутствие возможности использовать символ завершающего байта в качестве элемента строки.
- <http://queue.acm.org/detail.cfm?id=2010365>, англ.;
<http://habrahabr.ru/blogs/programming/126566/#habracut>, рус.
(перевод плохой).



Нуль-терминатор

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
0.	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	TAB	LF	VT	FF	CR	SO	SI
1.	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
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

Специальный символ таблицы ASCII, который обозначает конец строки



Инициализация строк в языке СИ

В языке СИ инициализация строки может быть выполнена двумя способами:

1. Согласно правилам инициализации массивов. Символы задаются с использованием символьных констант, а нуль-терминатор необходимо указывать **явным образом**.

```
char s1[5]={ 't', 'e', 's', 't', '\0' };
```

```
char s2[] = { 'h', 'e', 'l', 'l', 'o', '\0' };
```

2. С использованием строковых констант. Содержимое строки задается в двойных кавычках, вставка нуль-терминатора происходит **автоматически**:

```
char s1[5]= "test";
```

```
char s2[] = "hello";
```



Ввод-вывод строк в языке СИ

В языке СИ ввод-вывод строк может быть выполнен двумя способами:

1. Посимвольно:

```
i=0;  
do{  
    scanf("%c",&s[i]); i++;  
}while(s[i-1] != '\n');  
s[i-1] = '\0';  


---

for(i=0; s[i] != '\0'; i++)  
    printf("%c",s[i]);
```

2. С использованием спецификатора %s:

```
scanf("%s",s); // ввод до пробела!  
fgets(s,msize,stdin); // ввод до '\n'  
printf("%s",s); // вывод до '\0'
```



Операции со строками

- получение символа по номеру позиции (индексу);
- определение длины строки;
- копирование строки;
- проверка на совпадение строк (с учётом или без учёта регистра символов);
- сравнение строк (в алфавитном порядке);
- поиск символа в строке;
- **конкатенация** (соединение) строк: "ab" + "cd" = "abcd";
- получение **подстроки** по индексам начала и конца:
"abcdefghijklmnopqrstuvwxyz", 3, 8 → "defghi" ;



Операции со строками

- определение подстроки, состоящей только из заданного набора символов.
- проверка вхождения одной строки в другую (**поиск подстроки в строке**): "abcdefghij": "cdef" – да, "cfde" – нет;
- разбиение строки на поля (tokens), разделенные заданным набором символов.
- замена подстроки в строке:

"abcdefghij", "cdef" на "cfde" = "ab**cfde**ghij"



Получение символа по индексу

Получение символа по номеру позиции (индексу) является тривиальной операцией в языке СИ.

Она реализуется через операцию индексации массивов.

Например:

```
char s[5] = "test";
```

```
s[0] == 't'
```

```
s[1] == 'e'
```

```
s[2] == 's'
```

```
s[3] == 't'
```

```
s[4] == '\0'
```



Определение длины строки

Операция определения длины строки может быть выполнена двумя способами:

1. Явно запрограммирована:

```
int i;  
for(i=0; s[i] != '\0'; i++);  
len = i;
```

2. С использованием функции strlen():

```
...  
#include <string.h>  
...  
len = strlen(s);
```



Копирование строки*

В языке СИ не предусмотрено операции присваивания строк. Существует два способа ее реализации:

1. Явное программирование:

?

2. С использованием функции `strcpy()`:

...

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

...

```
strcpy (s2, s1) ;
```



Копирование строки

В языке СИ не предусмотрено операции присваивания строк. Существует два способа ее реализации:

1. Явное программирование:

```
int i;  
for(i=0; s1[i] != '\0'; i++)  
    s2[i] = s1[i];  
s2[i] = '\0';
```

2. С использованием функции strcpy():

```
#include <string.h>  
...  
strcpy(s2, s1);
```



Проверка строк на совпадение (с учетом регистра)

В языке СИ не предусмотрено операции сравнения строк. Существует два способа ее реализации:

1. Явное программирование:

```
int i, flg = 1;
for(i=0; flg && s1[i]!='\0' && s2[i]!='\0'; i++) {
    if( s1[i]!=s2[i] ) flg = 0;
}
if( flg == 1 ) - равны!
```

2. С использованием функции strcmp():

```
#include <string.h>
if( strcmp(s1, s2) == 0 ) - равны!
```



Проверка строк на совпадение (без учета регистра)*

В языке СИ не предусмотрено операции сравнения строк. Существует два способа ее реализации:

1. Явное программирование:

?

Подсказка:

- Расстояние между любыми одинаковыми символами в верхнем и нижнем регистре одинаково и равно **'a' – 'A'**

2. С использованием функции `strcascmp()`:

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

```
if( strcascmp(s1, s2) == 0 ) – равны!
```



Справочный слайд (таблица ASCII)

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
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	{	 	}	~	



Сравнение строк в алфавитном порядке

Существует два способа реализации:

1. Явное программирование:

```
int i, flg = 0;
for(i=0; !flg && s1[i]!='\0' && s2[i]!='\0'; i++) {
    if( s1[i] < s2[i] ) flg = -1;
    else if( s1[i] > s2[i] ) flg = 1;
}
if( s1[i] < s2[i] ) flg = -1;    // если строки
else if( s1[i] > s2[i] ) flg = 1; // разной длины
if( flg == 0 ) - равны!
if( flg > 0 ) - s1 по алфавиту ниже s2
if( flg < 0 ) - s1 по алфавиту выше s2
```



Сравнение строк в алфавитном порядке (2)

Существует два способа реализации:

1. Явное программирование (предыдущий слайд ...)

2. С использованием функции `strcmp()`:

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

```
if( strcmp(s1, s2) == 0 ) – равны!
```

```
if( strcmp(s1, s2) > 0 ) – s1 по алфавиту ниже s2
```

```
if( strcmp(s1, s2) < 0 ) – s1 по алфавиту выше s2
```



Поиск символа в строке

Существует два способа реализации данной операции.

1. Явное программирование:

```
int i;  
char c = ?;  
for(i=0; (s1[i]!='\0') && (s1[i]!=c); i++);  
if( s[i] == c ) – найден по индексу i.
```

2. С использованием функции strchr():

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

```
char c = ?, *ptr;
```

```
ptr = strchr(s1, c);
```

```
ptr = &s1[10]
```



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	\0	...
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	-----



Конкатенация строк

Операция определения длины строки может быть выполнена двумя способами:

1. Явно запрограммирована:

```
int i, j;  
for(i=0; s1[i] != '\0'; i++) ;  
for(j=i; s2[j-i] != '\0'; j++) {  
    s1[j] = s2[j-i];  
}  
s1[j] = s2[j-i]; // установка нуль-терминатора
```

2. С использованием функции strcat():

```
#include <string.h>  
  
strcat(s1, s2);
```



Получение подстроки по индексам*

Данная задача может быть решена двумя способами.

1. Явное программирование:

```
int main()
{
    int i, j;
    char s1[100] = "abcdefghijklmnop";
    char s2[10];
    int s = 3, e = 8;
    for(i=s; i<=e && s1[i] != '\0'; i++)
        s2[i-s] = s1[i];

    printf("s1 = %s", s2);
}
```



Получение подстроки по индексам

Данная задача может быть решена двумя способами.

1. Явное программирование:

```
int main()
{
    int i, j;
    char s1[100] = "abcdefghijklmnop";
    char s2[10];
    int s = 3, e = 8;
    for(i=s; i<=e && s1[i] != '\0'; i++)
        s2[i-s] = s1[i];
    s2[i-s] = '\0';
    printf("s1 = %s", s2);
}
```



Получение подстроки по индексам

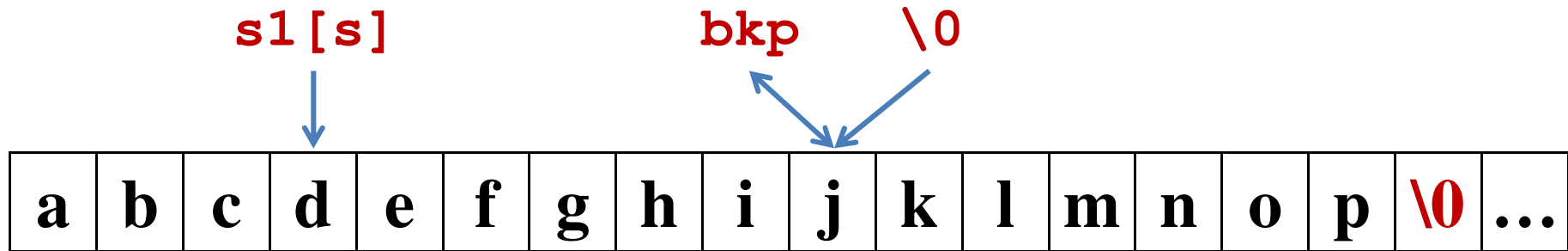
Данная задача может быть решена двумя способами.

1. Явное программирование (... см. предыдущие слайды)
2. Специальной функции, предназначенной для решения данной задачи, однако она может быть решена с использованием строковых функций:

```
// 1. временная подстановка нуля-терминатора
char bkp = s1[e+1];
s1[e+1] = '\0';
strcpy(s2, &s1[s]);
s1[e+1] = bkp;
// 2. использование strncpy
strncpy(s2, &s1[s], e-s+1);
s2[e+1] = '\0'
```



Получение подстроки по индексам



```
// 1. временная подстановка нуль-терминатора
char bkr = s1[e+1];
s1[e+1] = '\0';
strncpy(s2, &s1[s]);
s1[e+1] = bkr;
// 2. использование strncpy
strncpy(s2, &s1[s], e-s+1);
s2[e+1] = '\0'
```




Операции со строками

- определение подстроки, состоящей только из заданного набора символов.
- проверка вхождения одной строки в другую (**поиск подстроки в строке**): "abcdefghij": "cdef" – да, "cfde" – нет;
- разбиение строки на поля (tokens), разделенные заданным набором символов.
- замена подстроки в строке:

"abcdefghij", "cdef" на "cfde" = "ab**cfde**ghij"



Определение подстроки, состоящей из допустимых символов*

Операция определения длины строки может быть выполнена двумя способами:

1. Явно запрограммирована:

?

Подсказки:

1. Каждый символ строки $s1$ необходимо проверить на вхождение в $s2$.
2. Индекс первого символа, не входящего в $s2$ **является искомым решением.**
3. Задача проверки вхождения символа в строку была рассмотрена ранее.



Определение подстроки, состоящей из допустимых символов

Операция определения длины строки может быть выполнена двумя способами:

1. Явно запрограммирована (см. предыдущие слайды)
2. С использованием функции `strspn()`:

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

```
char s1[] = "hello world!", s2[] = "oelhw";
```

```
size_t t = strspn(s1, s2);
```

t = 5



h	e	l	l	o		w	o	r	l	d	!	\0	n	o	p	\0	...
---	---	---	---	---	--	---	---	---	---	---	---	----	---	---	---	----	-----



Поиск подстроки в строке

"abcdefghij": "cdef" – да, "cfde" – нет

В языке СИ предусмотрено решение данной задачи при помощи строковых функций:

```
if( strstr(s1,s2) != NULL ) – найдено
```

```
char *ptr = strstr(s1,s2);
```

```
if( ptr != NULL ) {
```

```
    ptr указывает на начало s2 в s1
```

```
}
```

ptr = &s1[2]



a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	\0	...
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	----	-----



Поиск подстроки в строке*

"abcdefghij": "cdef" – да, "cfde" – нет

Решить задачу явно (запрограммировать) .

Алгоритм:

1. Сначала найти в $s1$ вхождение $s2[0]$ – пусть это $s1[i]$.
2. Далее посимвольно сравнить остальные элементы:
 $s1[i+1] == s2[1], s1[i+2] == s2[2], \dots s1[i+k] == s2[k]$, где $(k+1) = \text{strlen}(s2)$
3. Если в процессе проверки какие-то из символов не совпали, то проверка равенства $s1[i]$ и $s2$ прекращается.
4. Продолжить поиск вхождения $s2$ в $s1$ начиная с $i+1$ элемента.



Хранение многострочного текста

В языке СИ перевод строки ранится в виде специального символа, ASCII код которого обозначается через Escape-последовательность '\n'. Данный символ не является признаком конца строки в терминах языка СИ. Это позволяет хранить многострочный текст в одной строке:

```
char str[] =  
"Hello!\nHow are you?\nCan we talk now?\n";
```

Н	е	l	l	о	!	\n	Н	о	w		a	r	e		y	o	u	?	\n
C	a	n		w	e		t	a	l	k		n	o	w	?	\n	\0		



Хранение набора строк

Одним из наиболее простых способов хранения набора строк является использование двумерного массива символов, также необходимо хранить число использованных строк:

```
char strings[6][20];  
int size;
```

D	r	o	p		i	n		t	h	e		b	u	c	k	e	t	\0	
L	e	t		w	e	l	l		a	l	o	n	e	\0					
A		h	a	r	d		n	u	t		t	o		c	r	a	c	k	\0
A		l	i	e		b	e	g	e	t	s		a		l	i	e	\0	
S	t	o	r	m		i	n		a		t	e	a	c	u	p	\0		
A	f	t	e	r		u	s		t	h	e		d	e	l	u	g	e	\0



Обработка набора строк

- инициализация набора строк;
- обращение к строке набора по индексу;
- вставка новой строки;
- удаление строки из набора;
- поиск заданной строки в наборе;
- поиск строки, содержащей заданную подстроку;
- конкатенация строк набора;
- перестановка строк в наборе;
- сортировка строк набора (в алфавитном порядке).



Инициализация набора строк

Инициализация набора строк подразумевает формирование двумерного массива символов, каждая строка которого соответствует пустой символьной строке.

```
char strings[10][256];  
int size = 0, i;  
for (i=0; i<10; i++)  
    strings[i][0] = '\\0';
```

\\0	r	o	p		i	n		t	h	e		b	u	c	k	...
\\0	e	t		w	e	l	l		a	l	o	n	e	\\0		...
...																
\\0	f	t	e	r		u	s		t	h	e		d	e	l	...



Обращение к строке набора по индексу

Для работы с отдельной строкой в наборе используются сечения двумерных массивов: **фиксируется старший индекс, который отвечает за номер строки.**

```
char strings[10][256];
```

```
...
```

```
strlen(strings[i]);
```

\0	r	o	p		i	n		t	h	e		b	u	c	k	...
L	e	t		w	e	l	l		a	l	o	n	e	\0		...
...																
\0	f	t	e	r		u	s		t	h	e		d	e	l	...



Вставка новой строки

1. Запись новой строки в первую свободную позицию.
2. Увеличение количества занятых ячеек.

```
char strings[10][256];
```

...

```
strcpy(strings[size], "Let well alone");  
size++;
```

L	e	t		w	e	l	l		a	l	o	n	e	\0		...
\0	e	t		w	e	l	l		a	l	o	n	e	\0		...
...																
\0	f	t	e	r		u	s		t	h	e		d	e	l	...



Удаление строки из набора

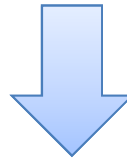
1. Перестановка с последней ненулевой строкой.
2. Запись **нуль-терминатора** в первый элемент строки двумерного массива с номером size-1.
3. Уменьшение количества свободных строк.

```
do{ strings[i][j] = strings[size-1][j];  
    j++;  
}while(strings[size-1][j] != '\0');  
strings[size-1] = '\0';  
size--;
```



Удаление строки из набора (2)

L	e	t		w	e	l	l		a	l	o	n	e	\0		...
H	e	l	l	o	!	\0	l		a	l	o	n	e	\0		...
...																
\0	f	t	e	r		u	s		t	h	e		d	e	l	...



H	e	l	l	o	!	\0	l		a	l	o	n	e	\0		...
\0	e	l	l	o	!	\0	l		a	l	o	n	e	\0		...
...																
\0	f	t	e	r		u	s		t	h	e		d	e	l	...



Поиск заданной строки в наборе

Поиск заданной строки сводится к просмотру всех строк и сравнению каждой из них с искомой:

```
char strings[10][256], s[256];  
...  
for(i=0;i<size;i++){  
    if( strcmp(strings[i], s) == 0 ){  
        // строка найдена  
        // 1. Прерывание цикла  
        // 2. Обработка строки  
        //  
    }  
}
```



Поиск в наборе строки, содержащей заданную подстроку

Поиск подстроки сводится к просмотру всех строк и поиску в каждой из них искомой подстроки:

```
char strings[10][256], s[256];  
...  
for (i=0; i<size; i++) {  
    if ( strstr(strings[i], s) != NULL ) {  
        // строка найдена  
        // 1. Прерывание цикла  
        // 2. Обработка строки  
        //  
    }  
}
```



Конкатенация строк набора

Для конкатенации строк набора требуется последовательное копирование каждой из строк в результирующую:

```
char strings[10][256], s[2560] = "";  
...  
for(i=0; i<size; i++) {  
    int l = strlen(s);  
    strcpy(&s[l], strings[i]);  
}
```




Конкатенация строк набора (2)

L	e	t		w	e	l	l	\0	a	l	o	n	e	\0		...
H	e	l	l	o	!	\0	l		a	l	o	n	e	\0		...
H	i	!	\0	!	?	\0	s		t	h	e		d	e	l	...

s \0

s L e t w e l l \0

s L e t w e l l H e l l o ! H i ! \0




Перестановка строк в наборе

Для перестановки строк набора требуется последовательный обмен соответствующих символов. **Количество обменов определяется более длинной строкой.**

```
char strs[10][256], s[2560] = "";  
  
    ...  
  
int l1 = strlen(strs[i]);  
int l2 = strlen(strs[j]);  
for(k=0; k <= l1 || k <= l2; k++) {  
    char t = strs[i][k];  
    strs[i][k] = strs[j][k];  
    strs[j][k] = t;  
}
```



Перестановка строк в наборе (2)

	L	e	t		w	e	l	l	\0	a	l	o	n	e	\0		...
	H	e	l	l	o	!	\0	l		a	l	o	n	e	\0		...
	H	i	!	\0	!	?	\0	s		t	h	e		d	e	l	...



H	i	!	\0	!	?	\0	s		a	l	o	n	e	\0		...
H	e	l	l	o	!	\0	l		a	l	o	n	e	\0		...
L	e	t		w	e	l	l	\0	t	h	e		d	e	l	...



Сортировка строк набора (в алфавитном порядке)

Для сортировки набора строк может быть использован любой алгоритм сортировки. Например, алгоритм сортировки методом пузырька.

Соотношению $x < y$ ставится в соответствие результат сравнения строк при помощи функции `strcmp`:

$$s1 < s2 \Rightarrow \text{strcmp}(s1, s2) < 0$$

т.е. строка $s1$ по алфавиту должна располагаться выше строки $s2$.



Сортировка методом пузырька

$s[0] > s[1]$

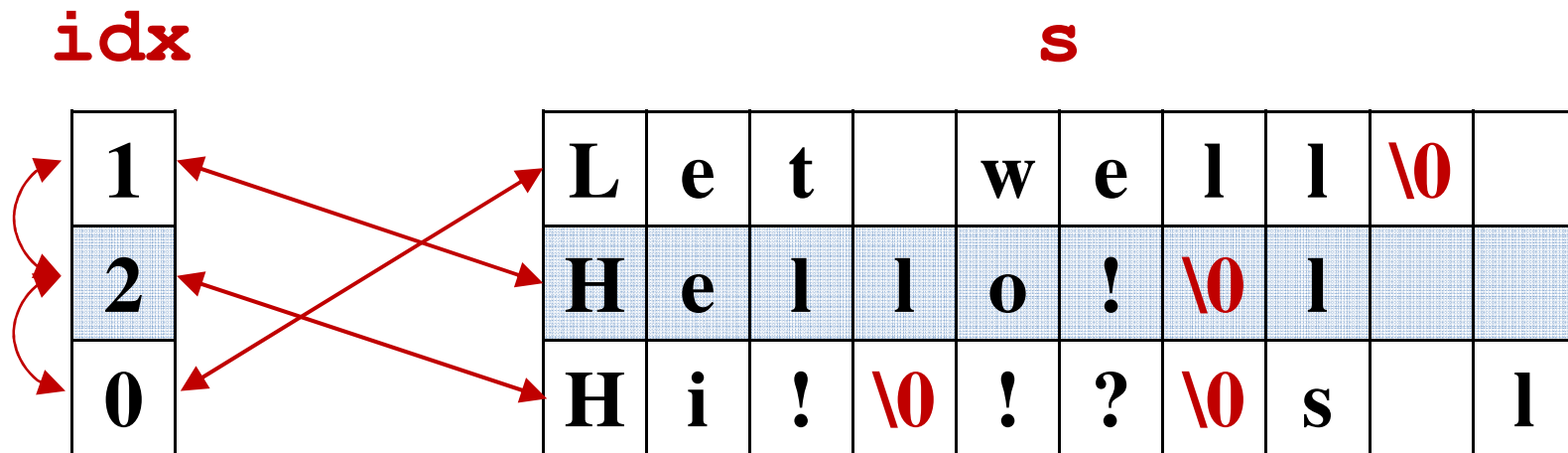
L	e	t		w	e	l	l	\0	
H	e	l	l	o	!	\0	l		
H	i	!	\0	!	?	\0	s		l

$s[1] < s[2]$



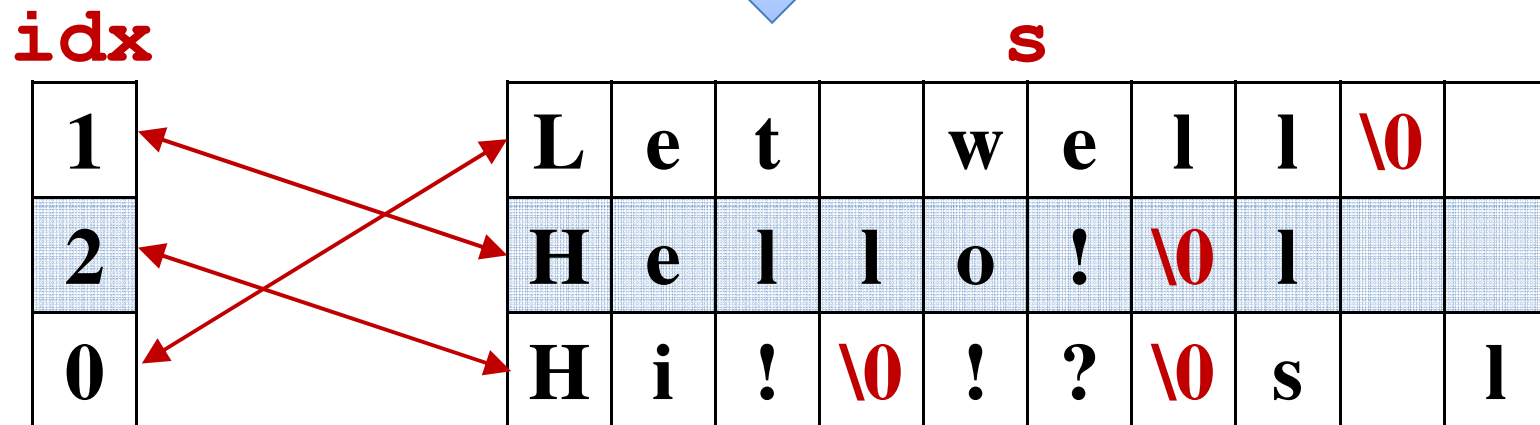
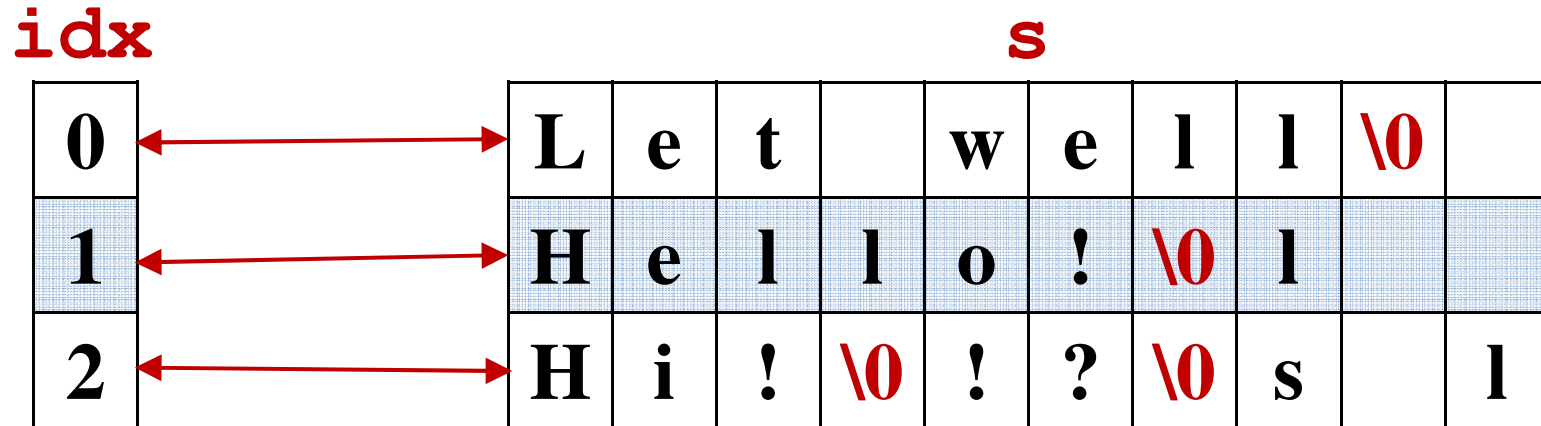
Оптимизация сортировки строк методом пузырька

- В процессе сортировки задача перестановки двух строк возникает многократно.
- Это достаточно трудоемкая операция, которая не может быть выполнена за фиксированное число действий.
- Для повышения скорости сортировки может быть использован массив целочисленных индексов.





Оптимизация сортировки строк методом пузырька (2)

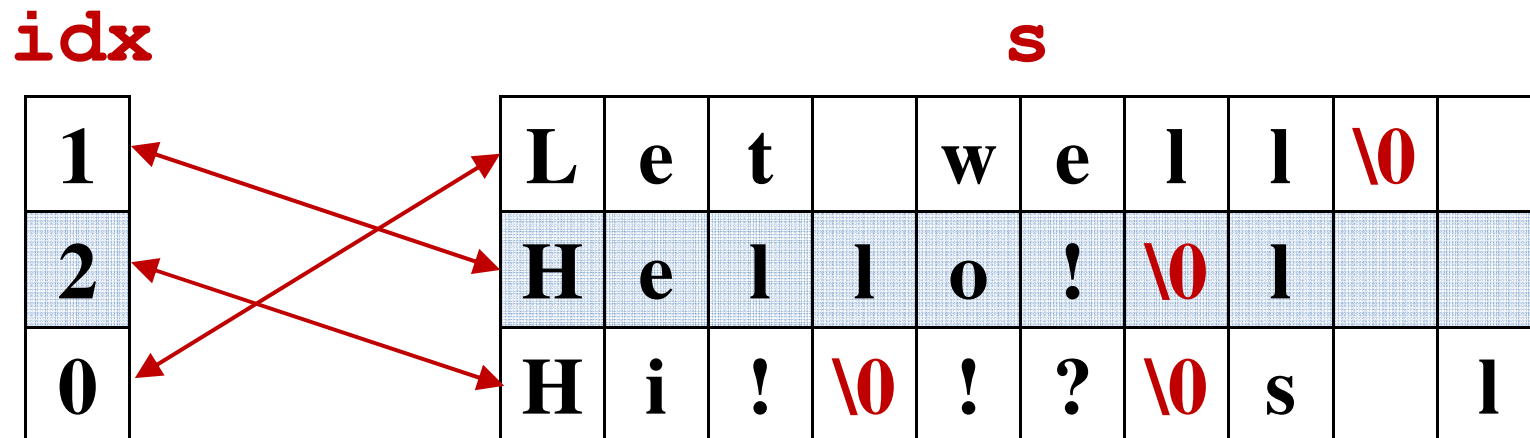




Оптимизация сортировки строк методом пузырька

- Работа с отсортированным массивом должна выполняться через полученный индексный массив. Например, для доступа к i -му элементу необходимо использовать следующее выражение:

$s[idx[i]]$





Перестановка элементов набора

$s[idx[0]] > s[idx[1]] \Rightarrow$
 $idx[0] \leftrightarrow idx[1]$

0	↔	L	e	t		w	e	l	l	\0	
1	↔	H	e	l	l	o	!	\0	l		
2	↔	H	i	!	\0	!	?	\0	s		l



1	↔	L	e	t		w	e	l	l	\0	
0	↔	H	e	l	l	o	!	\0	l		
2	↔	H	i	!	\0	!	?	\0	s		l



СПАСИБО ЗА ВНИМАНИЕ!