ФГОБУ ВПО "СибГУТИ"
Кафедра вычислительных систем

ЯЗЫКИ ПРОГРАММИРОВАНИЯ /
ПРОГРАММИРОВАНИЕ

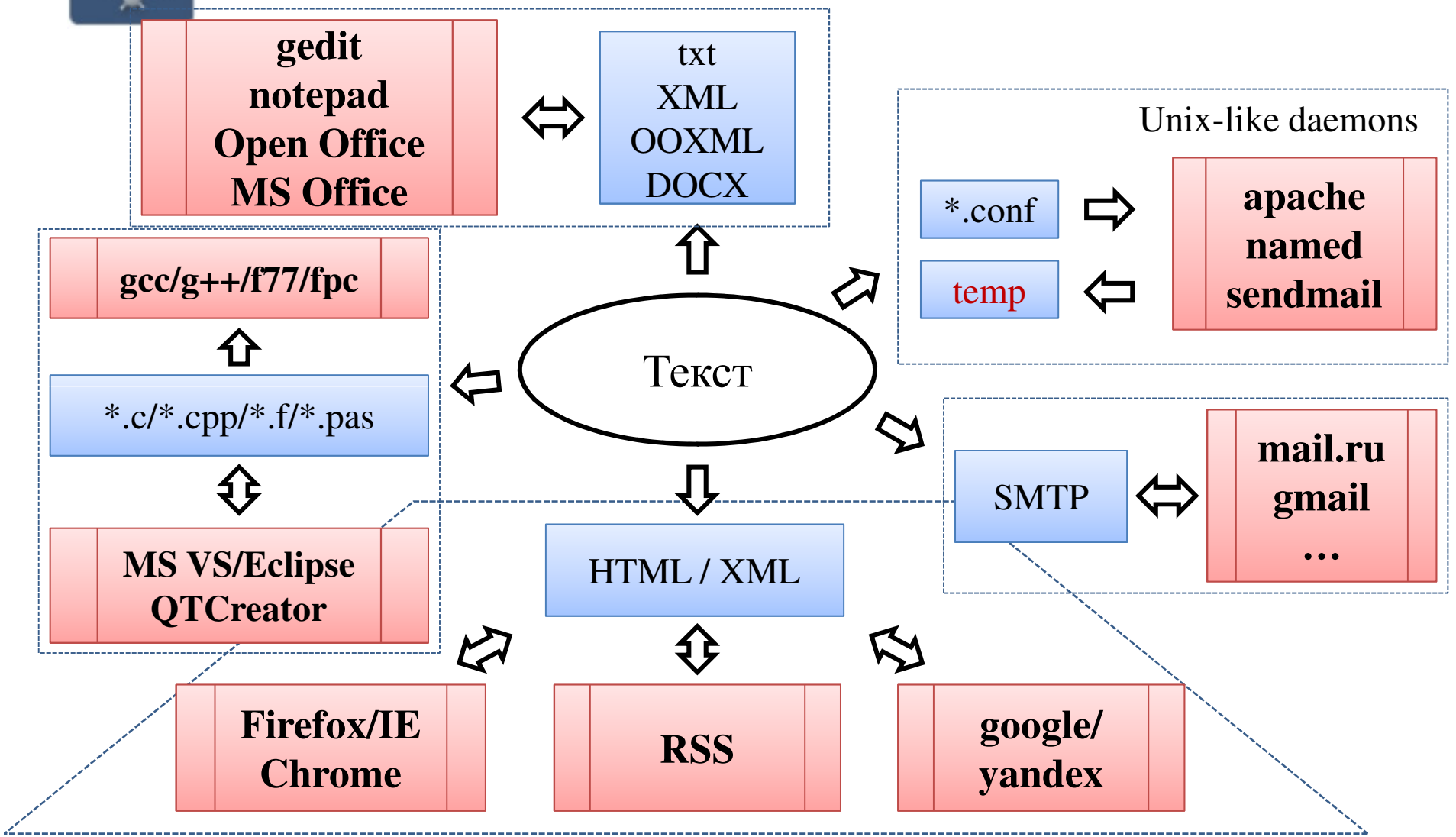
Обработка текстовой информации

Преподаватель:

Доцент Кафедры ВС, к.т.н.

Поляков Артем Юрьевич

Применение текстовой информации



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 (2)

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

Представление текстовой информации (3)

GlibC

6D	6F	6D	20	73	6F	61	70	20	66	72	61	6D	65

Файловый путь

Путь (англ. path) – набор символов, описывающий расположение файла или директории в файловой системе. Расположение задается перечислением директорий и (возможно) указанием файла.

В UNIX-подобных операционных системах разделительным знаком при записи пути является символ "/" (слэш).

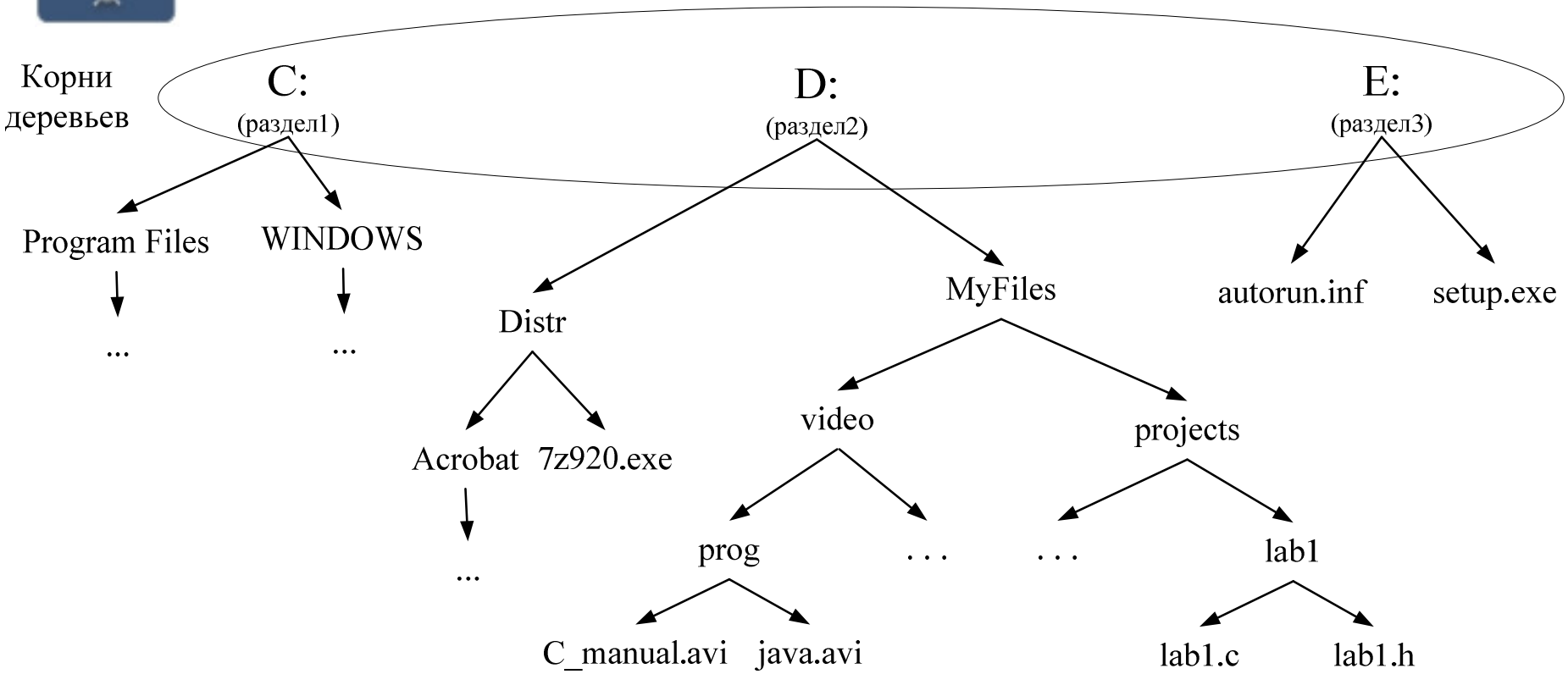
В ОС Windows – символ "\" (обратный слэш).

Примеры:

ОС Windows: C:\Windows\System32\calc.exe

ОС GNU/Linux: /usr/local/bin/gcc

Файловые пути в ОС Windows



Абсолютные пути:

D:\Distr\7z920.exe

C:\WINDOWS\

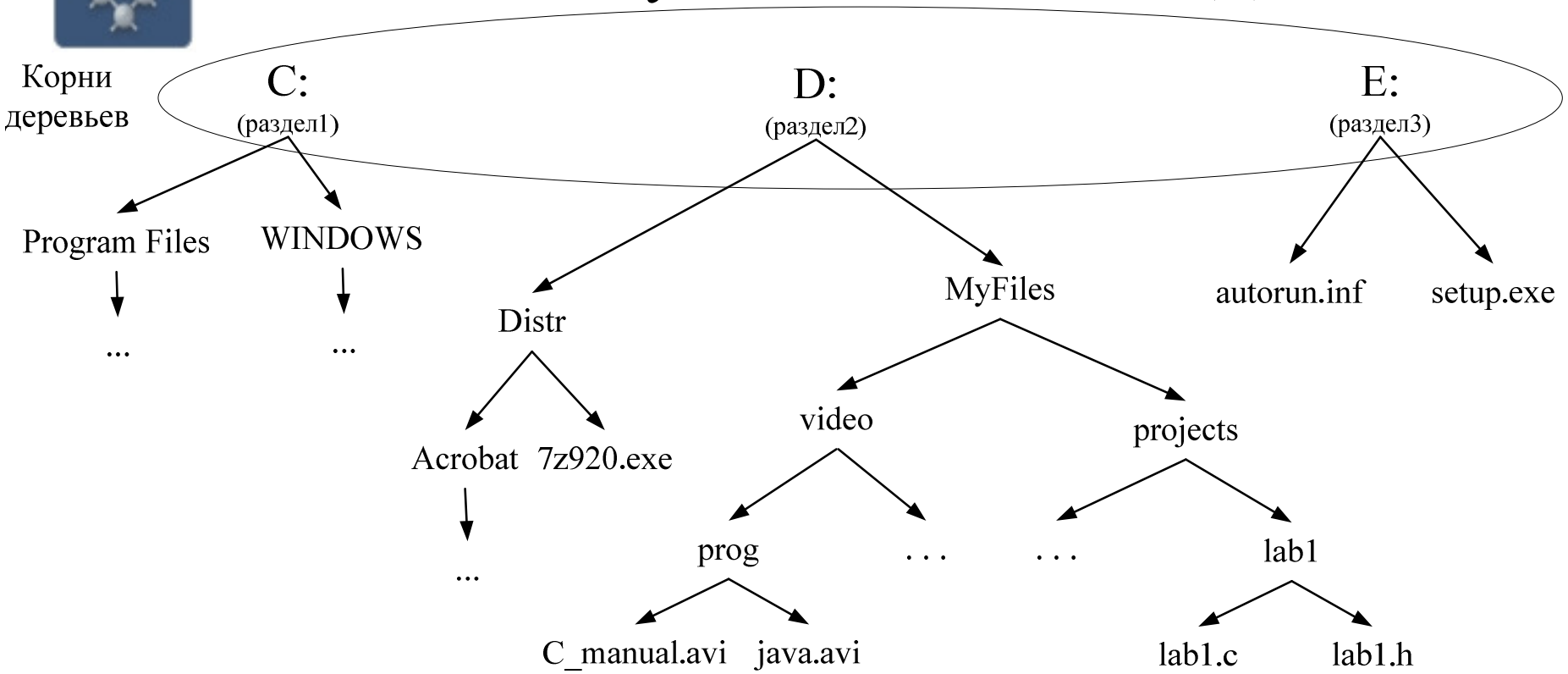
E:\autorun.inf

D:\MyFiles\video\prog\C_manual.avi

D:\MyFiles\video\prog\

D:\MyFiles\projects\lab1\lab1.c

Файловые пути в ОС Windows (2)



Относительные пути:

7z920.exe (в D:\Distr\)

WINDOWS\ (на C:\)

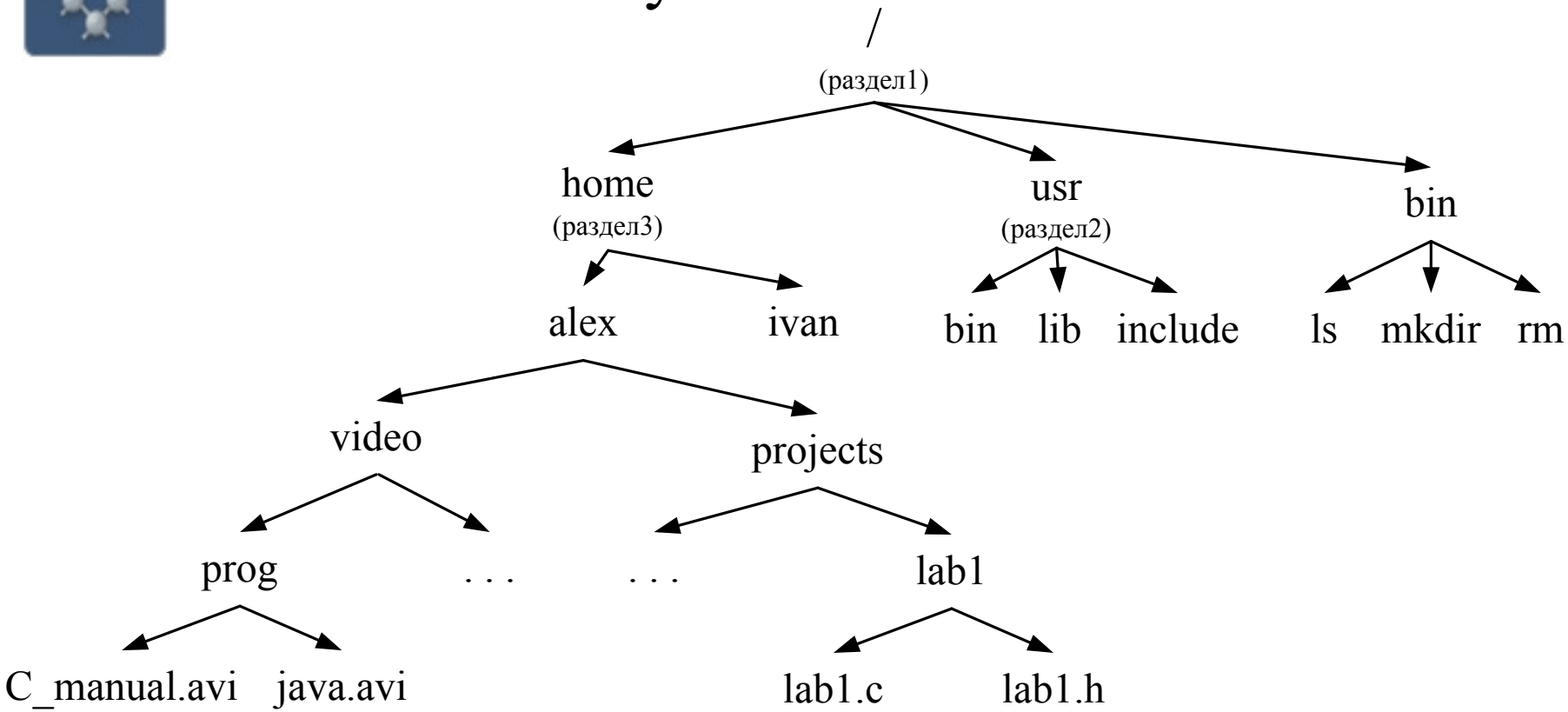
autorun.inf (на E:\)

.\prog\C_manual.avi (в D:\MyFiles\video)

video\prog\ (в D:\MyFiles\)

..\projects\lab1\lab1.c (в **D:\MyFiles\video**)

Файловые пути в ОС GNU/Linux



Абсолютные пути:

/usr/bin/

/bin/ls

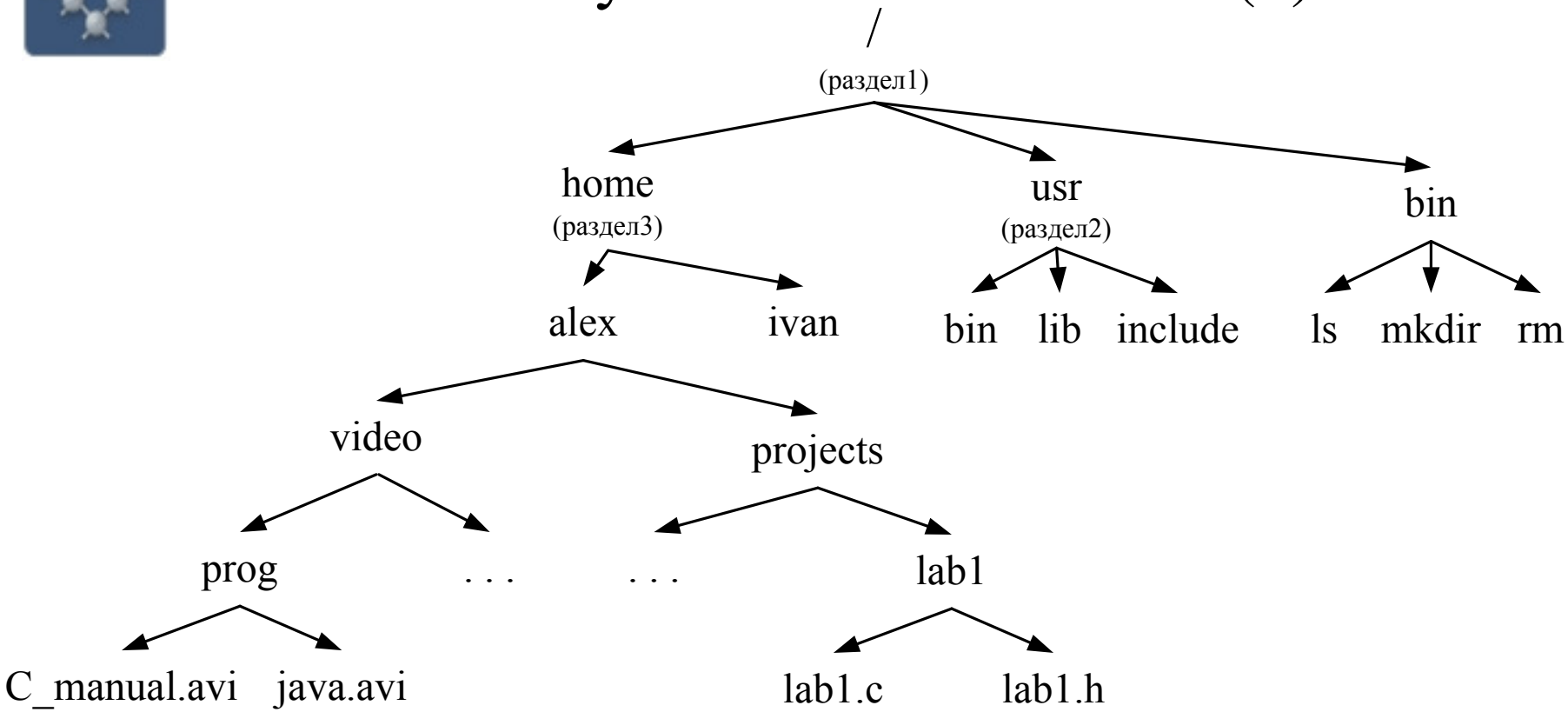
/home/alex/video

/home/alex/video/prog/C_manual.avi

/home/alex/video/prog

/home/alex/projects/lab1/lab1.c

Файловые пути в ОС GNU/Linux (2)



Относительные пути:

bin/ (в /usr/)

/ls (в /bin)

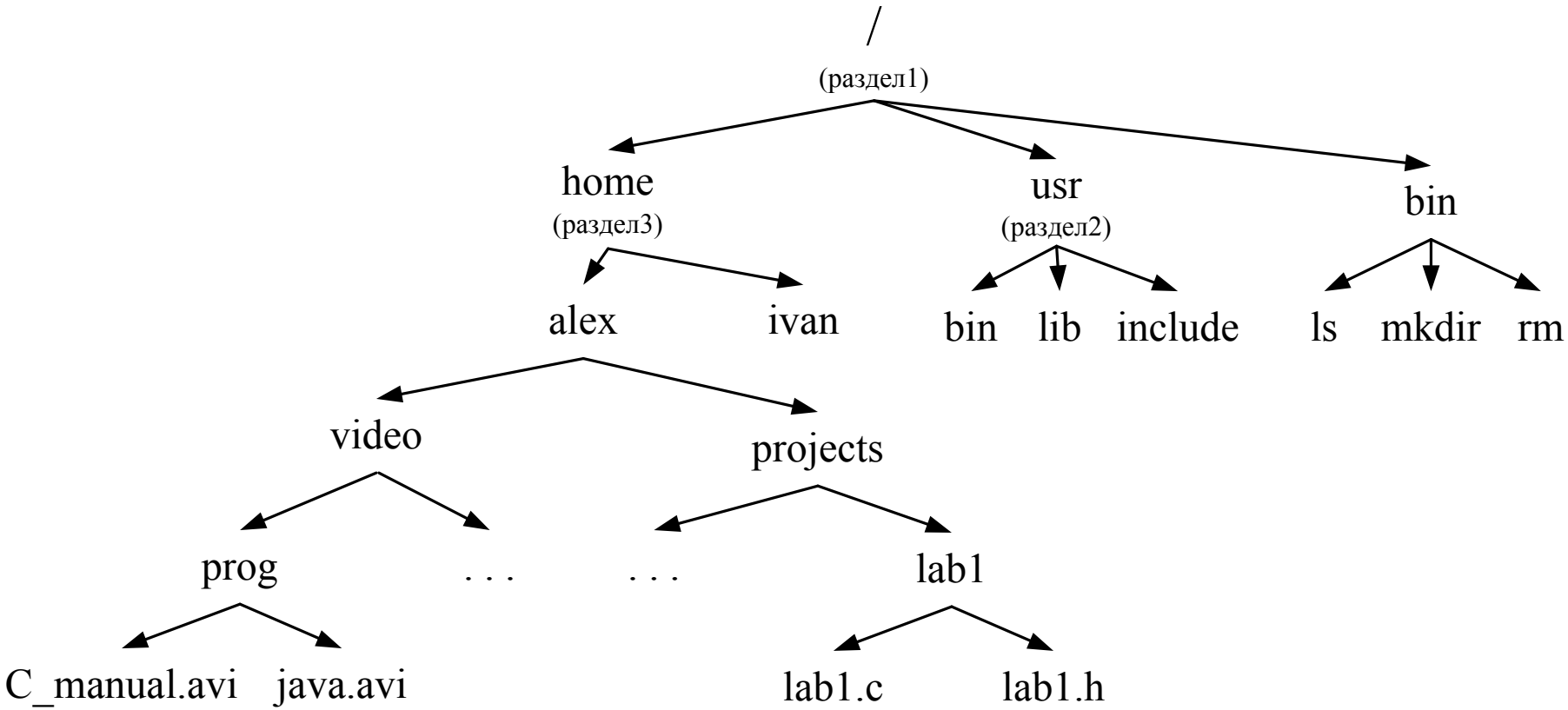
./alex/video (в /home)

./prog/C_manual.avi (в /home/alex/video/)

prog (в /home/alex/video/)

../projects/lab1/lab1.c (в /home/alex/video)

Файловые пути в ОС GNU/Linux (путь относительно дом. каталога)



Относительно ДК тек. польз. (alex)

~/video/prog/C_manual.avi

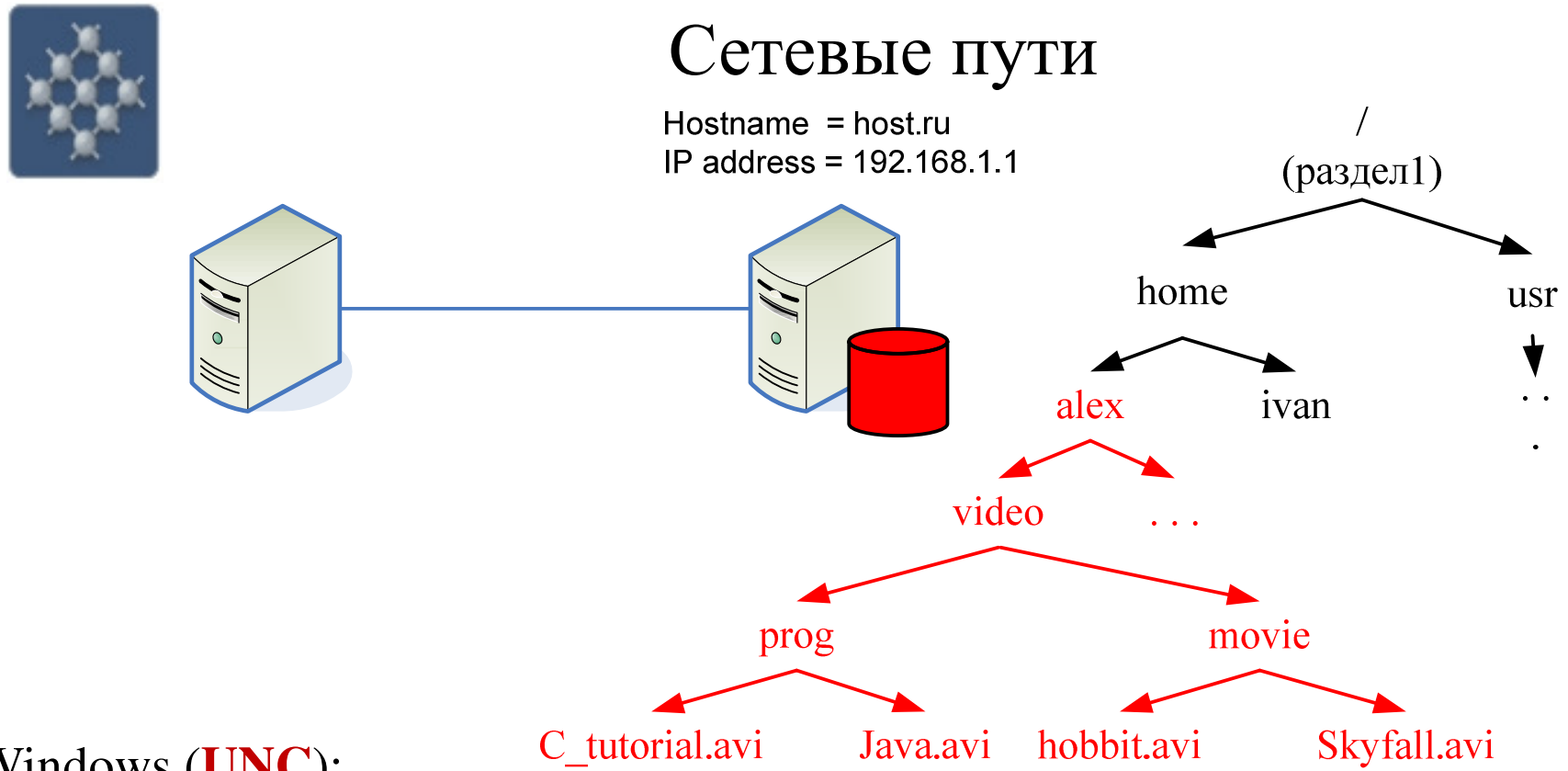
~/projects/lab1

Относительно ДК заданного польз.

~alex/video/prog/C_manual.avi

~alex/projects/lab1

Сетевые пути



Windows (**UNC**):

\\192.168.1.1\video\prog\C_tutorial.avi ; \\host.ru\video\movie\hobbit.avi

GNU/Linux (**URL**):

ftp://192.168.1.1/video/prog/Java.avi ; ftp://host.ru/video/prog/Java.avi

GNU/Linux (**SCP**):

192.168.1.1:/home/alex/video/prog; alex@host.ru:~/video/prog/Java.avi

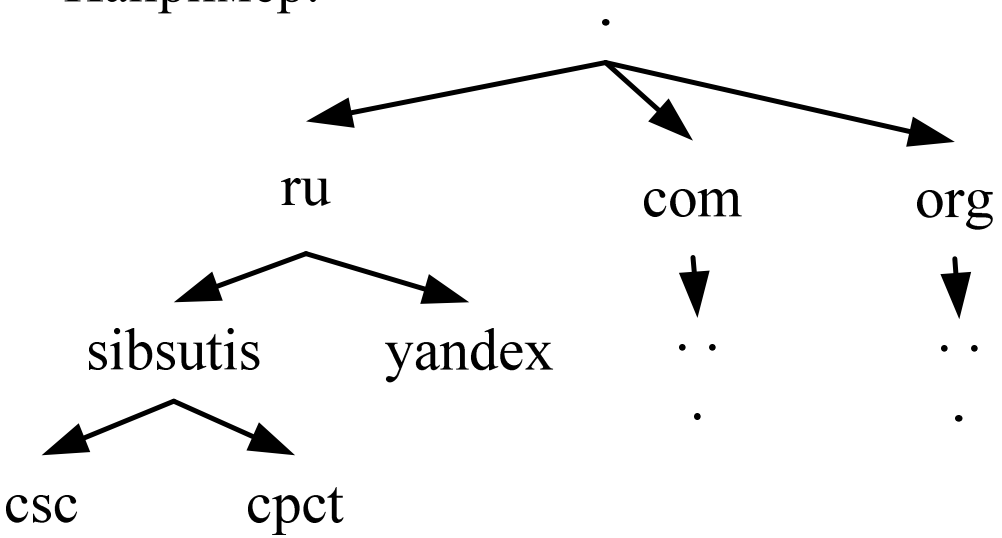
Имена узлов и IP адреса

IP адрес представляет собой набор из четырех целых чисел, разделенных точкой:

$$IP = x_1.x_2.x_3.x_4,$$

где $0 \le x_i \le 255$, например: **192.168.1.1**, **93.158.134.3**, **173.194.47.164**, **91.196.245.216**.

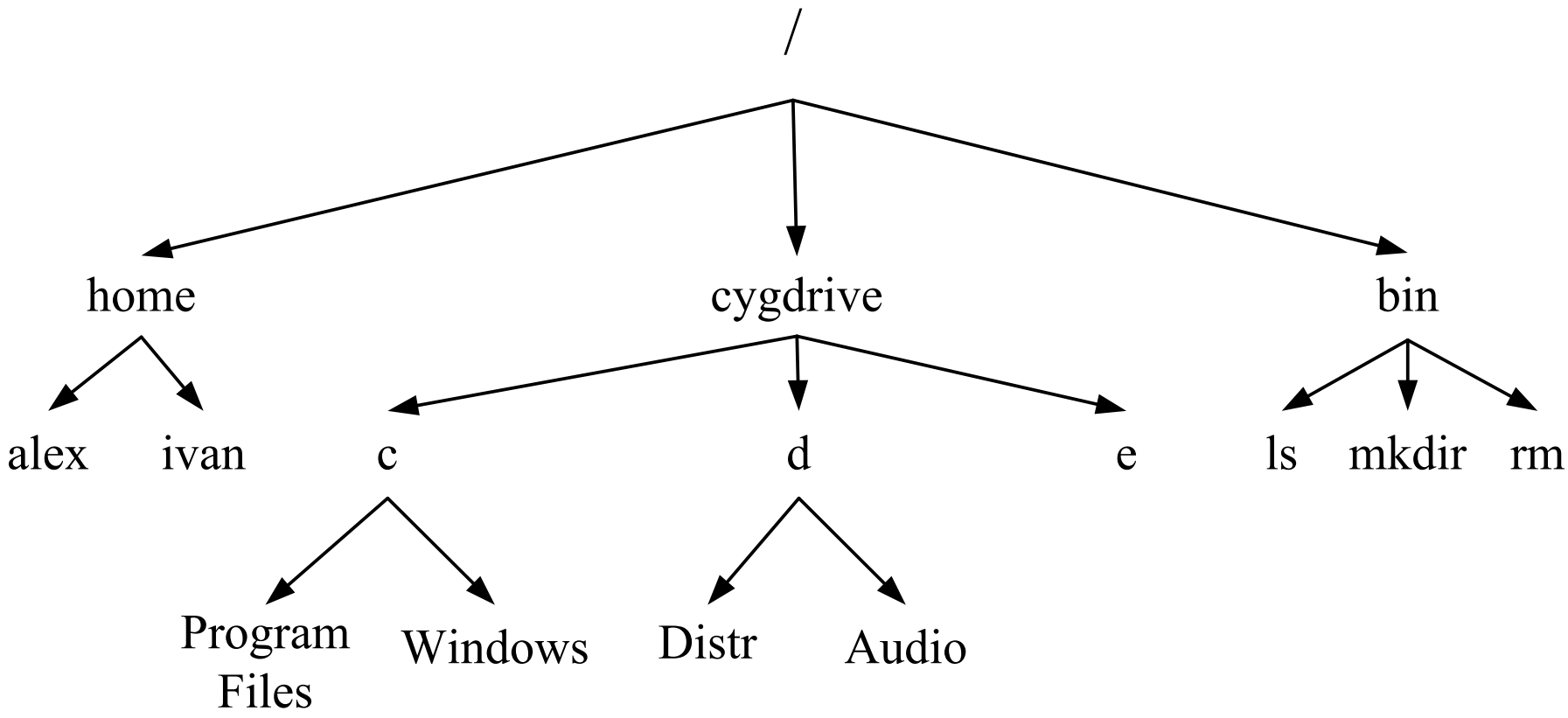
Имя узла (доменное имя. http://ru.wikipedia.org/wiki/Доменное_имя). Например:



В рамках данной лабораторной работы будем считать, что существует только три домена верхнего уровня: **ru**, **com** и **org**. ***Использование любых других доменов недопустимо и считается ошибкой.***

Cygwin

Эмулятор Linux-окружения Cygwin (http://www.cygwin.com/) предоставляет пользователю возможность работы в Linux-подобной среде из операционной системы Windows.



Ограничения

Максимальная длина пути:

260 символов

Запрещенные символы:

: * ? " < > |

Допустимые протоколы:

http, ftp, rsync, smb

Допустимые домены первого уровня:

ru, com и **org**

Допустимые IP адреса:

$\mathbf{IP} = x_1 . x_2 . x_3 . x_4$, где $0 \le x_i \le 255$

Хранение текстовой информации

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

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

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];

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

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:

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

Простейшие операции над строками

Получение символа по номеру позиции

char s1[] = "Hello world!", s2[] = "C:\\Distr\7z.exe"

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
H	**e**	**l**	**l**	**o**		**w**	**o**	**r**	**l**	**d**	**!**	**\0**					

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	**:**	****	****	**D**	**i**	**s**	**t**	**r**	****	**7**	**z**	**.**	**e**	**x**	**e**	**\0**	

Данный функционал реализуется стандартными средствами языка Си – операцией индексации `[x]`

s1[3] = 'l' **s1[0] == 'H'** **s1[5] = ' '**

s2[6] == 's' **s2[16] == '\0'** **s2[5] = 'i'**

Принадлежность символа диапазону

При обработке строк часто возникает задача определить, принадлежит ли заданный символ *ch* некоторому диапазону, например: малые латинские буквы, цифры, латинские буквы.

	.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

символ ∈ [a, z]

При обработке строк часто возникает задача определить, принадлежит ли заданный символ *ch* некоторому диапазону, например: малые латинские буквы, цифры, латинские буквы.

	.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	{	|	}	~	DEL

```
int isLowCase(char ch)
{
if( ch >= 'a' && ch <= 'z')
      return 1;
return 0;
}
```

символ ∈ [0, 9]

При обработке строк часто возникает задача определить, принадлежит ли заданный символ *ch* некоторому диапазону, например: малые латинские буквы, цифры, латинские буквы.

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
3.	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?

```
int isDigit(char ch)
{
if( ch >= '0' && ch <= '9')
      return 1;
return 0;
}
```

Приведение к нижнему регистру

При обработке строк часто возникает задача определить, принадлежит ли заданный символ *ch* некоторому диапазону, например: малые латинские буквы, цифры, латинские буквы.

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

```
char toLowCase(char ch)
{
      if( ch >= 'A' && ch <= 'Z')
            return ch + ('a' – 'A');
      return ch;
}
```

Определение длины строки (slen)

char str1[] = "Hello world!", str2[] = "C:\\Distr\7z.exe"

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
H	**e**	**l**	**l**	**o**		**w**	**o**	**r**	**l**	**d**	**!**	**\0**					

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	**:**	****	****	**D**	**i**	**s**	**t**	**r**	****	**7**	**z**	**.**	**e**	**x**	**e**	**\0**	

Длина строки – количество **полезных** символов в ней. Для определения длины строки s необходимо вычислить индекс ее первого нуль-терминатора:

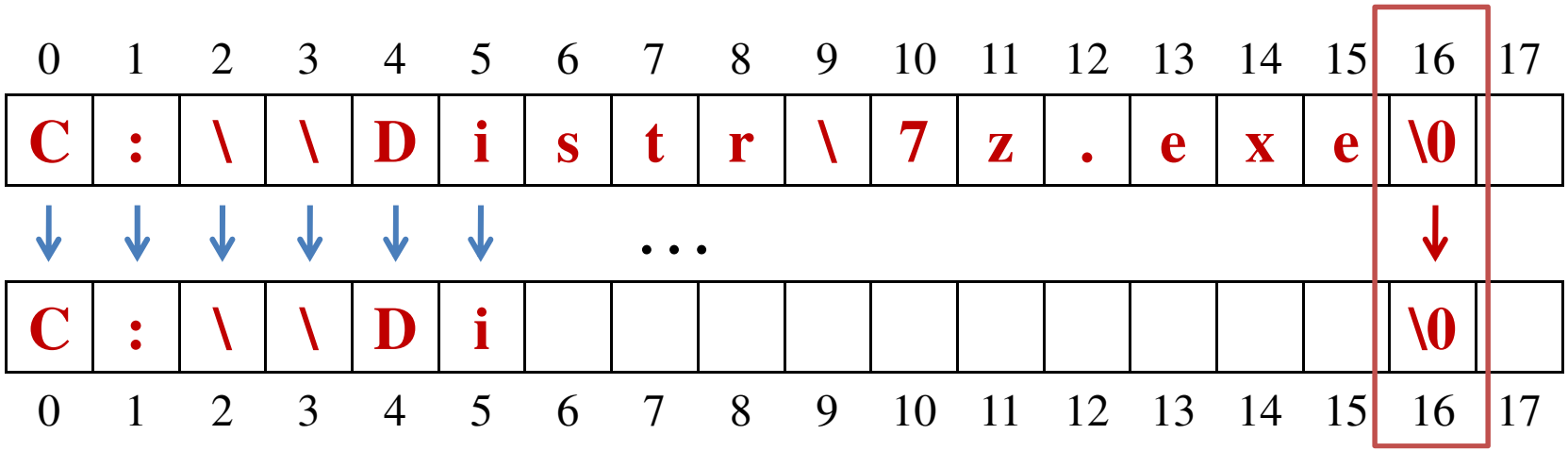
```
int i;
for(i=0;s[i] != '\0'; i++);
len = i;
```

str1[12] = '\0', количество символов в строке str1 – 12.

str1[16] = '\0', количество символов в строке str1 – 16.

Копирование строки (scopy)

char s1[] = "C:\\Distr\7z.exe", s2[128];

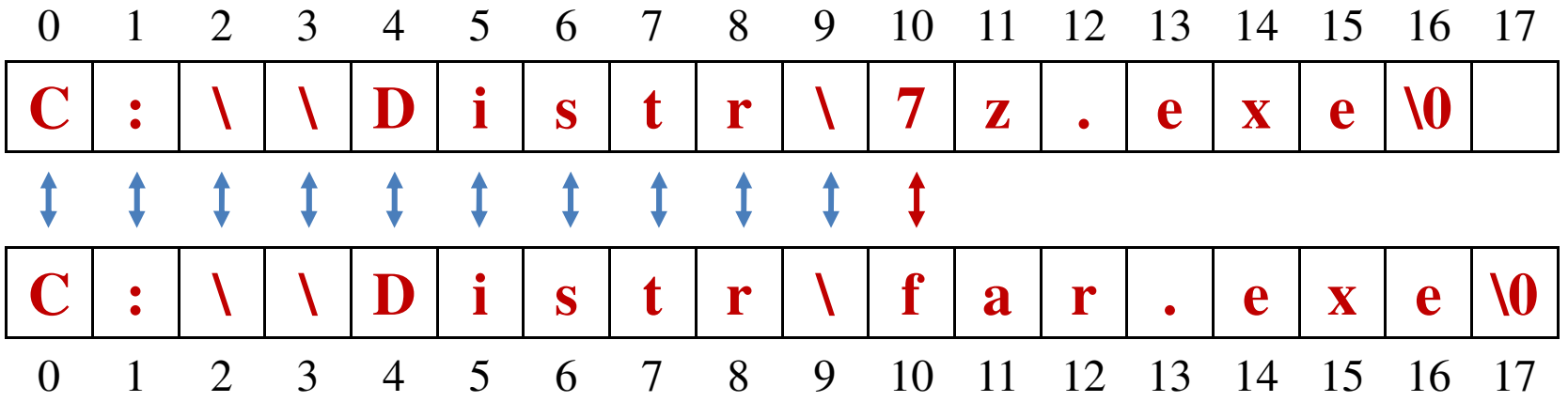


Копирование строки s1 в s2 предусматривает посимвольное присваивание элементов s1 элементам s2 начиная с индекса 0 до элемента, содержащего нуль-терминатор, включительно:

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

Проверка строк на совпадение (sequal)

char s1[] = "C:\\Distr\7z.exe", s2[] = "C:\\Distr\far.exe";



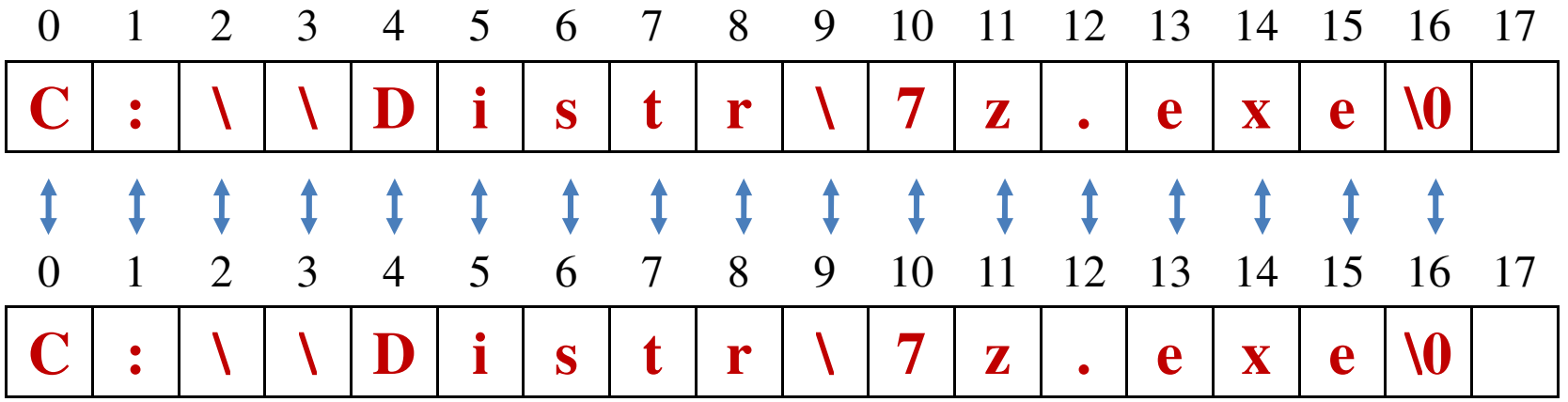
Результатом данной операции является 0, если строки совпадают и 1 в противном случае:

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

flg == 0 – результат

Проверка строк на совпадение (2)

char s1[] = "C:\\Distr\7z.exe", s2[] = "C:\\Distr\7z.exe";



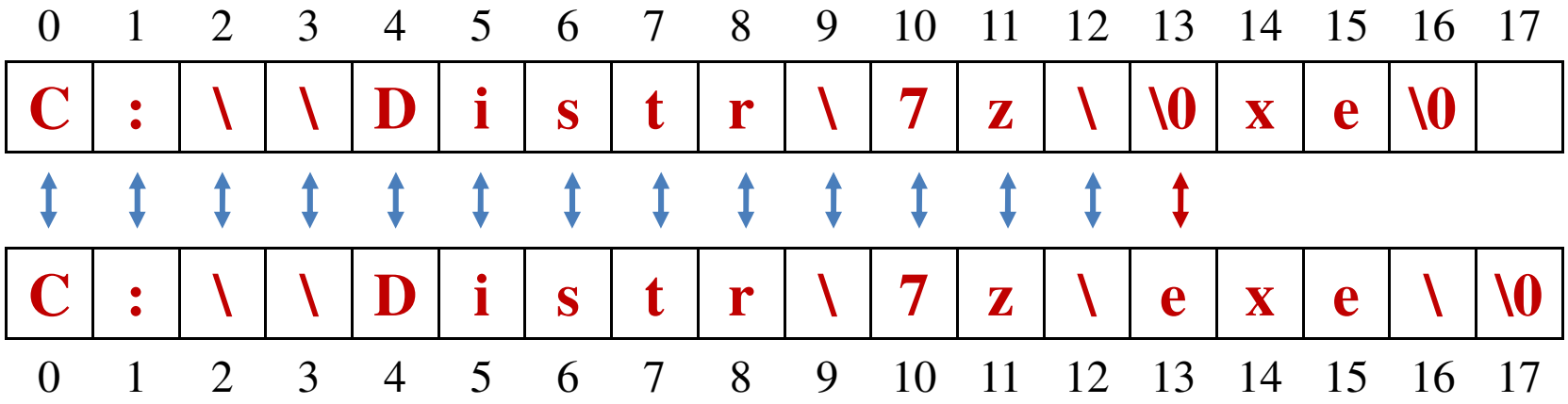
Результатом данной операции является 0, если строки совпадают и 1 в противном случае:

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

flg == 1 – результат

Проверка строк на совпадение (3)

char s1[] = "C:\\Distr\7z\", s2[] = "C:\\Distr\7z\exe\";



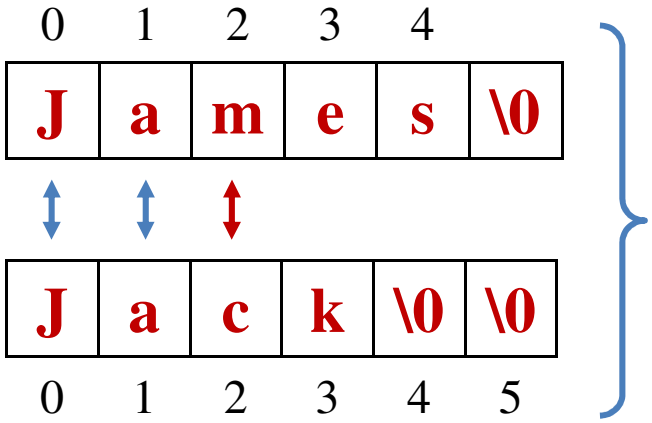
Результатом данной операции является 0, если строки совпадают и 1 в противном случае:

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

flg == 0 – результат

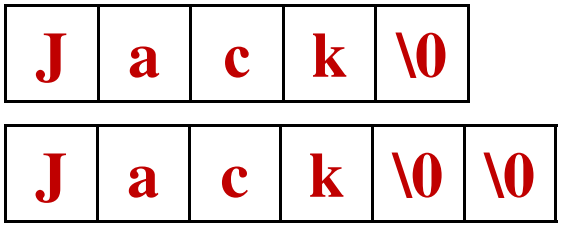
Сравнение строк по алфавиту (scmp)

```
char s1[] = "C:\\Distr\7z\", s2[] = "C:\\Distr\7z\exe\";
```

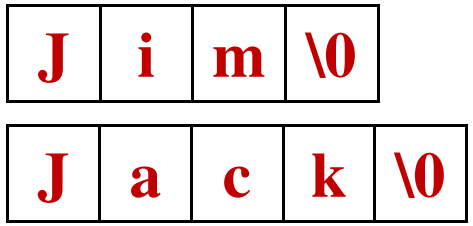


Буква 'm' в латинском алфавите расположена ниже буквы 'c'. Поэтому строки стоят в неправильном порядке

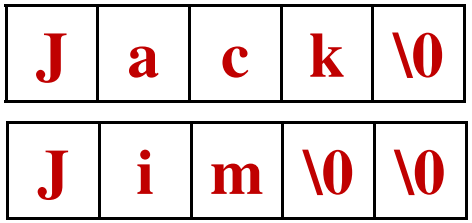
Всего возможно три варианта соотношений двух строк:



Строки равны:
0 (s1 = s2)

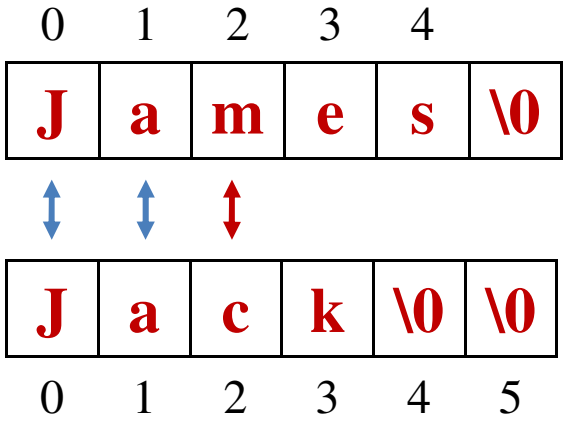


Первая по алфавиту ниже: **1 (s1 > s2)**



Первая по алфавиту выше: **-1 (s1 < s2)**

Сравнение строк по алфавиту (2)



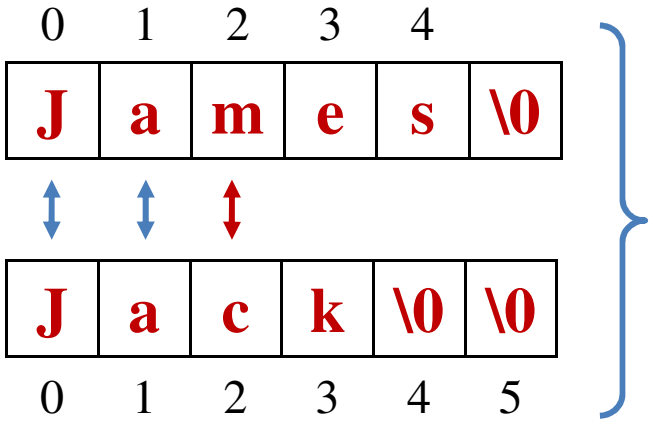
Буква 'm' в латинском алфавите расположена ниже буквы 'c'.

В таблице ASCII код 'm' > кода 'c'!

'm' = 0x6D > 0x63 = 'c'

	.0	.1	.2	.3	.4	.5	.6	.7	.8	.9	.A	.B	.C	.D	.E	.F
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 по алфавиту (3)



Буква 'm' в латинском алфавите расположена ниже буквы 'c'.

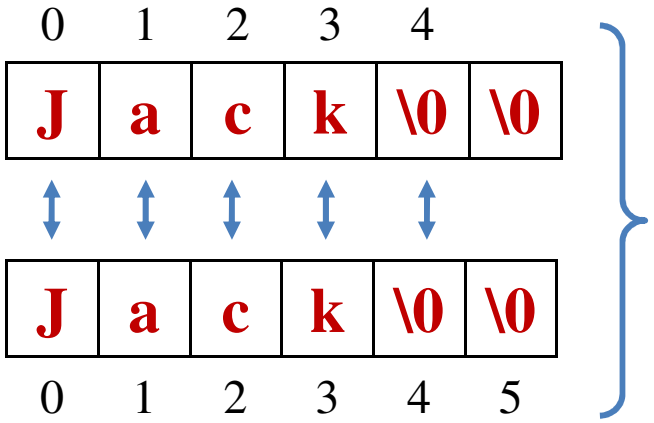
В таблице ASCII код 'm' > кода 'c'!

'm' = 0x6D > 0x63 = 'c'

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

flg == 1 – результат

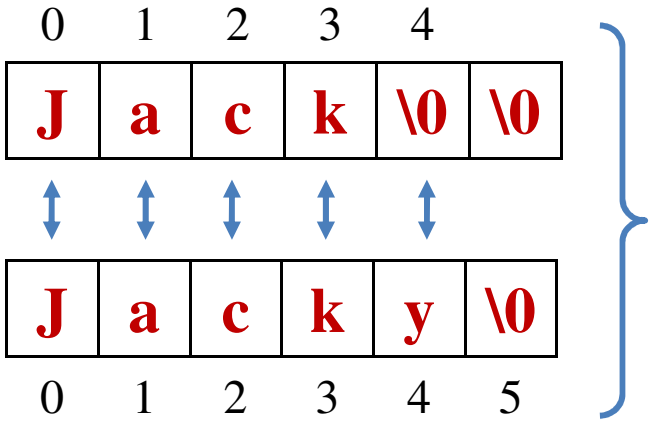
Сравнение строк по алфавиту (4)




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

flg == 0 – результат

Сравнение строк по алфавиту (4)



Буква 'm' в латинском алфавите расположена ниже буквы 'c'.

В таблице ASCII код '\0' < кода 'y'!

'y' = 0x79 > 0x00 = '\0'

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

flg == -1 – результат

Подстрока

Строка β называется *подстрокой* строки α, если существуют строки γ и δ такие, что γ β δ = α. **Пример**:

α = "lastkraftwagen", β = "kraft" – подстрока α, так как существуют строки γ = "last" и δ = "wagen", которые позволяют дополнить β до α: γ β δ = "last" "kraft" "wagen" = "lastkraftwagen".

Префиксом ω строки α, называется ее подстрока, для которой справедливо: γ ω δ = α, при этом γ = "" – пустая строка. Пример:

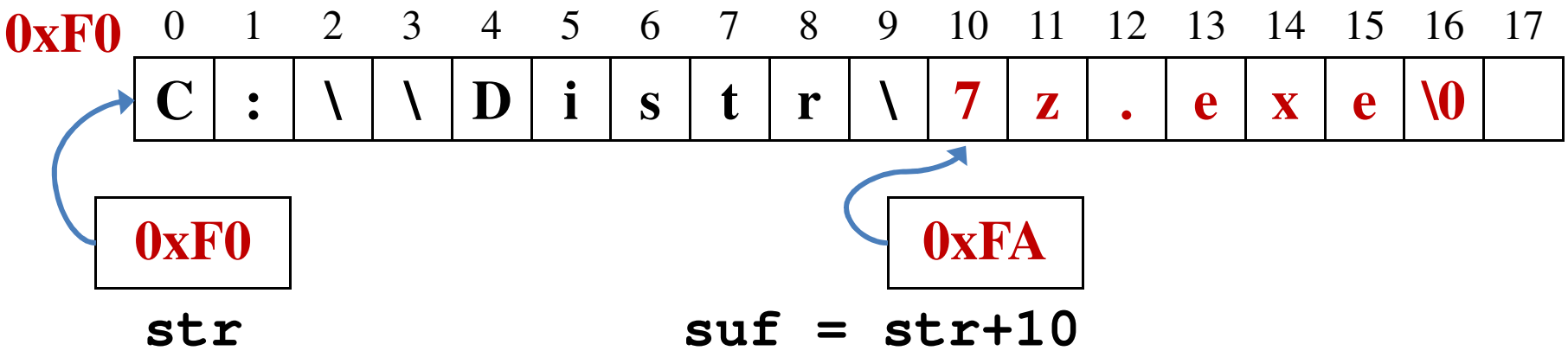
α = "lastkraftwagen", ω = "lastkraft" – префикс α, так как существуют строки γ = "" и δ = "wagen": γωδ = "" "lastkraft" "wagen".

Суффиксом φ строки α, называется ее подстрока, для которой справедливо: γφδ = α, при этом δ = "" – пустая строка. Пример:

α = "lastkraftwagen", φ = "wagen" – суффикс α, так как существуют строки γ = "lastkraft" и δ = "": γφδ = "lastkraft" "wagen".

Суффикс строки

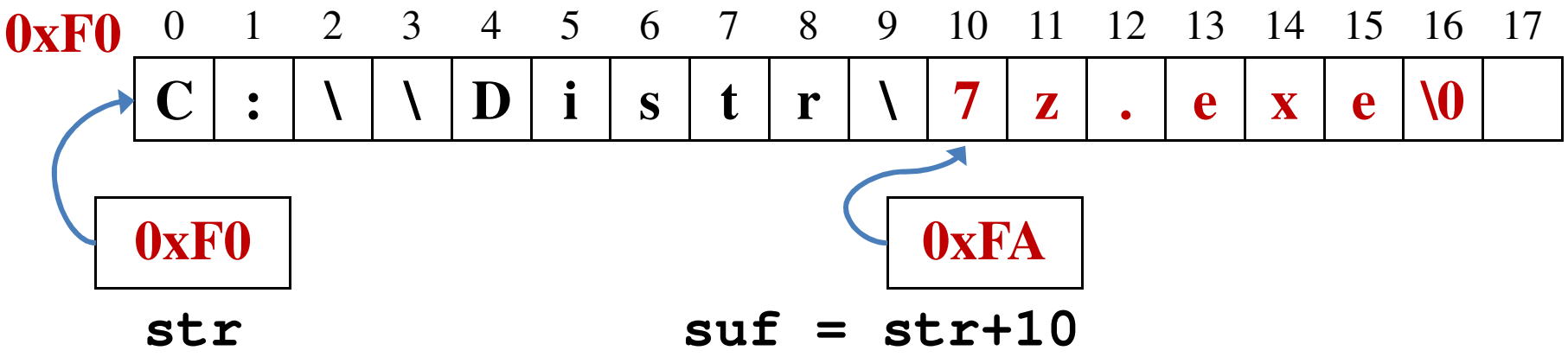
```
char str[] = "C:\\Distr\7z.exe", *suf;
```



Операция получение суффикса *suf* строки *str* предполагает наличие индекса *idx первого* элемента *suf*. В языке Си данная операция реализуется при помощи указателей. Имя строки (массива) – это указатель на ее первый элемент. Указатель на элемент str[idx] = указатель на suf.

Суффикс строки (2)

```
char str[] = "C:\\Distr\7z.exe", *suf;
```



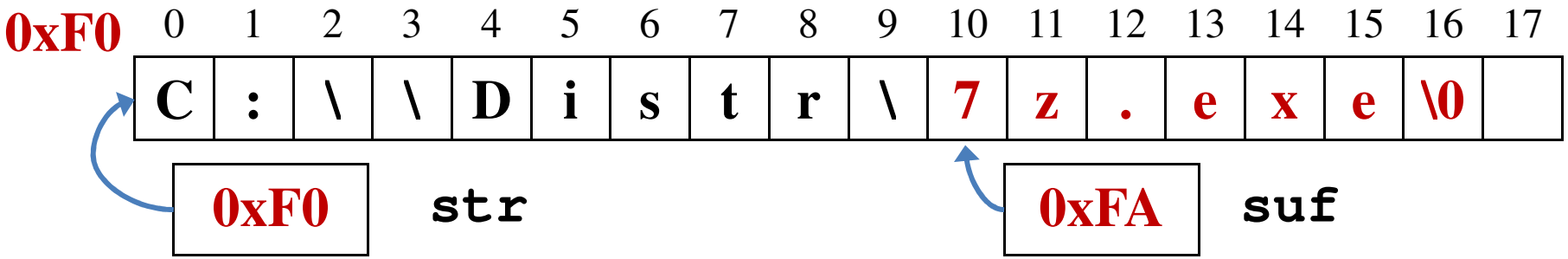
Для любой строковой функции строка suf = (str+10) равна "7z.exe", ее первый элемент лежит по адресу 0xFA:

```
slen(str) = 16, slen(suf) = 6.
```

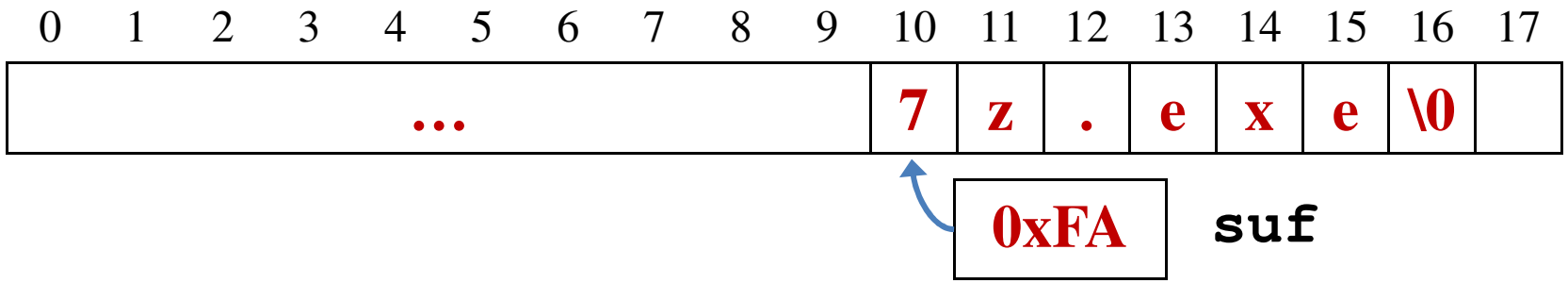
Суффикс строки (3)

main():

```
char str[]="C:\\Distr\7z.exe", *suf=(str+10);
slen(suf);
```

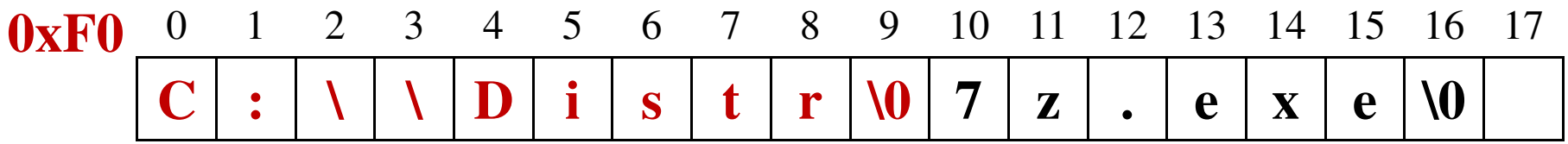


slen(char suf = 0xFA):



Префикс строки

```
char str[] = "C:\\Distr\7z.exe", *pref;
```

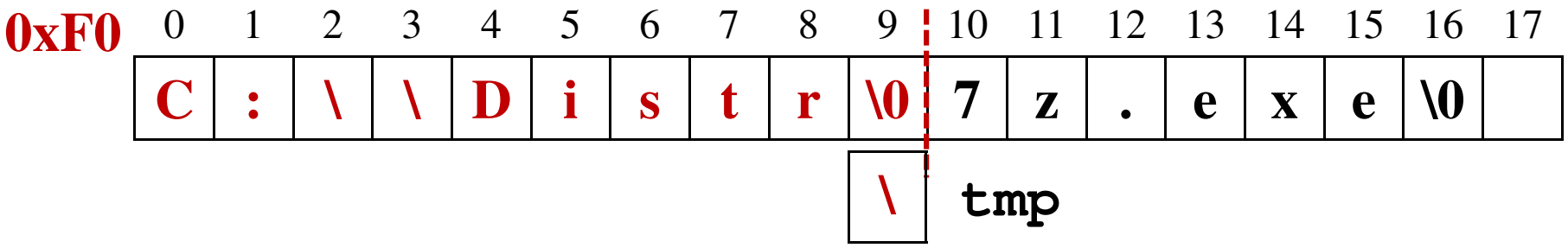


Операция получение префикса *pref* строки *str* также предполагает наличие индекса *idx последнего* элемента *pref*. В языке Си реализовать данную операцию сложнее. Требуется установка нуль-терминатора сразу за последним символом *pref*.

Если в дальнейшем строка *str* может понадобиться, то перезаписываемый символ необходимо дополнительно сохранить

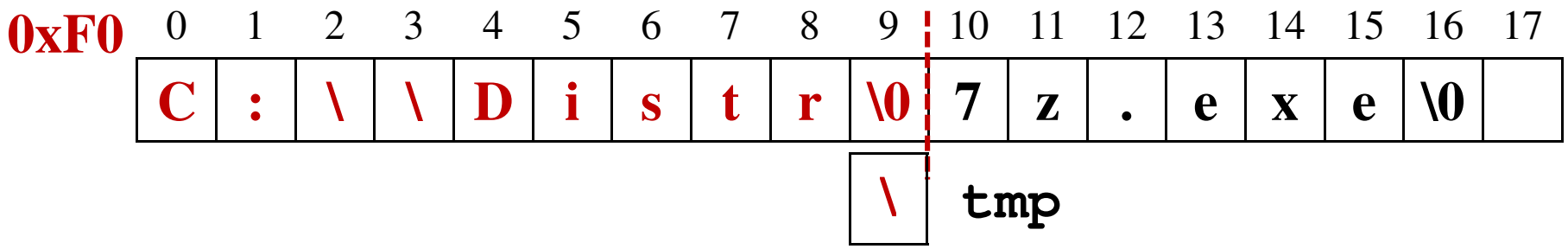
Префикс строки (2)

```
int idx = 9;
char str[] = "C:\\Distr\7z.exe", *pref;
char tmp = str[idx];
pref = str;
str[idx] = '\0';
slen(pref); // == 9
str[idx] = tmp;
```

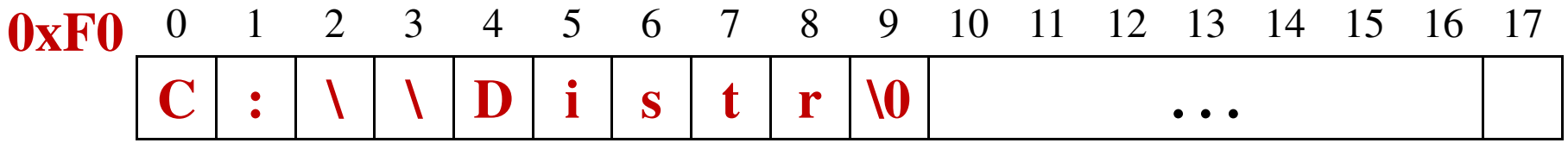


Префикс строки (3)

`main():`



`slen():`



Поиск символа в строке

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	**:**	****	****	**D**	**i**	**s**	**t**	**r**	****	**7**	**z**	**.**	**e**	**x**	**e**	**\0**	

Результатом операции поиска символа *ch* в строке *str* является индекс *idx* такой, что *str*[*idx*] = *ch*.

```
char str[ ] = "C:\\Distr\7z.exe";
char ch = '\\';
int i, idx = -1;
for(i=0;(str[i] != '\0') && (str[i] != ch); i++);
if( str[i] == ch )
    idx = i;
```

idx == 2

Поиск символа в строке (2)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	**:**	****	****	**D**	**i**	**s**	**t**	**r**	****	**7**	**z**	**.**	**e**	**x**	**e**	**\0**	

Результатом операции поиска символа *ch* в строке *str* является индекс *idx* такой, что *str*[*idx*] = *ch*.

```
char str[ ] = "C:\\Distr\7z.exe";
char ch = 'e';
int i, idx = -1;
for(i=0;(str[i] != '\0') && (str[i] != ch); i++);
if( str[i] == ch )
    idx = i;
```

idx == 13

Поиск символа в строке (3)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
C	**:**	****	****	**D**	**i**	**s**	**t**	**r**	****	**7**	**z**	**.**	**e**	**x**	**e**	**\0**	

Результатом операции поиска символа *ch* в строке *str* является индекс *idx* такой, что *str*[*idx*] = *ch*.

```
char str[ ] = "C:\\Distr\7z.exe";
char ch = 'M';
int i, idx = -1;
for(i=0;(str[i] != '\0') && (str[i] != ch); i++);
if( str[i] == ch )
    idx = i;
```

idx == -1

Типичные операции над строками

Конкатенация строк (scat)

Конкатенация (лат. concatenatio «присоединение цепями; сцепление») — операция склеивания строк. **Например**, конкатенация строк "lastkraft" и "wagen" - "lastkraftwagen".

В языке Си конкатенация *s*1 и *s*2 предполагает приписывание в конец *s*1 содержимого *s*2. Очевидно, что размер массива s1 должен быт достаточен, чтобы вместить в себя обе строки:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
s1	**C**	**:**	****	****	**D**	**i**	**s**	**t**	**r**	****	**\0**							

	0	1	2	3	4	5	6
s2	**7**	**z**	**.**	**e**	**x**	**e**	**\0**

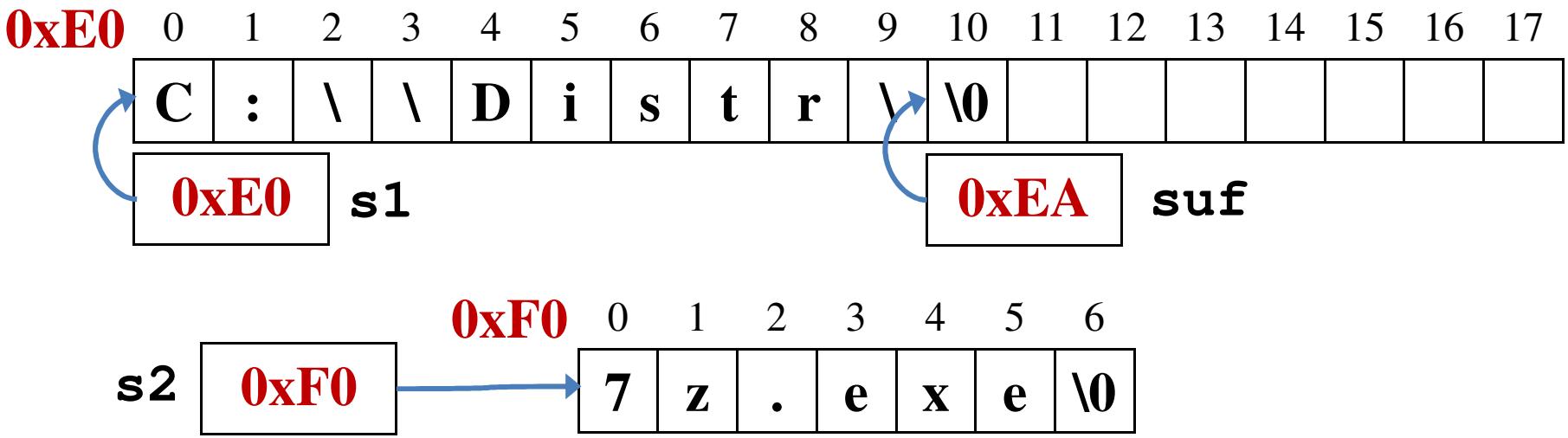
+

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
s1	**C**	**:**	****	****	**D**	**i**	**s**	**t**	**r**	****	**7**	**z**	**.**	**e**	**x**	**e**	**\0**	

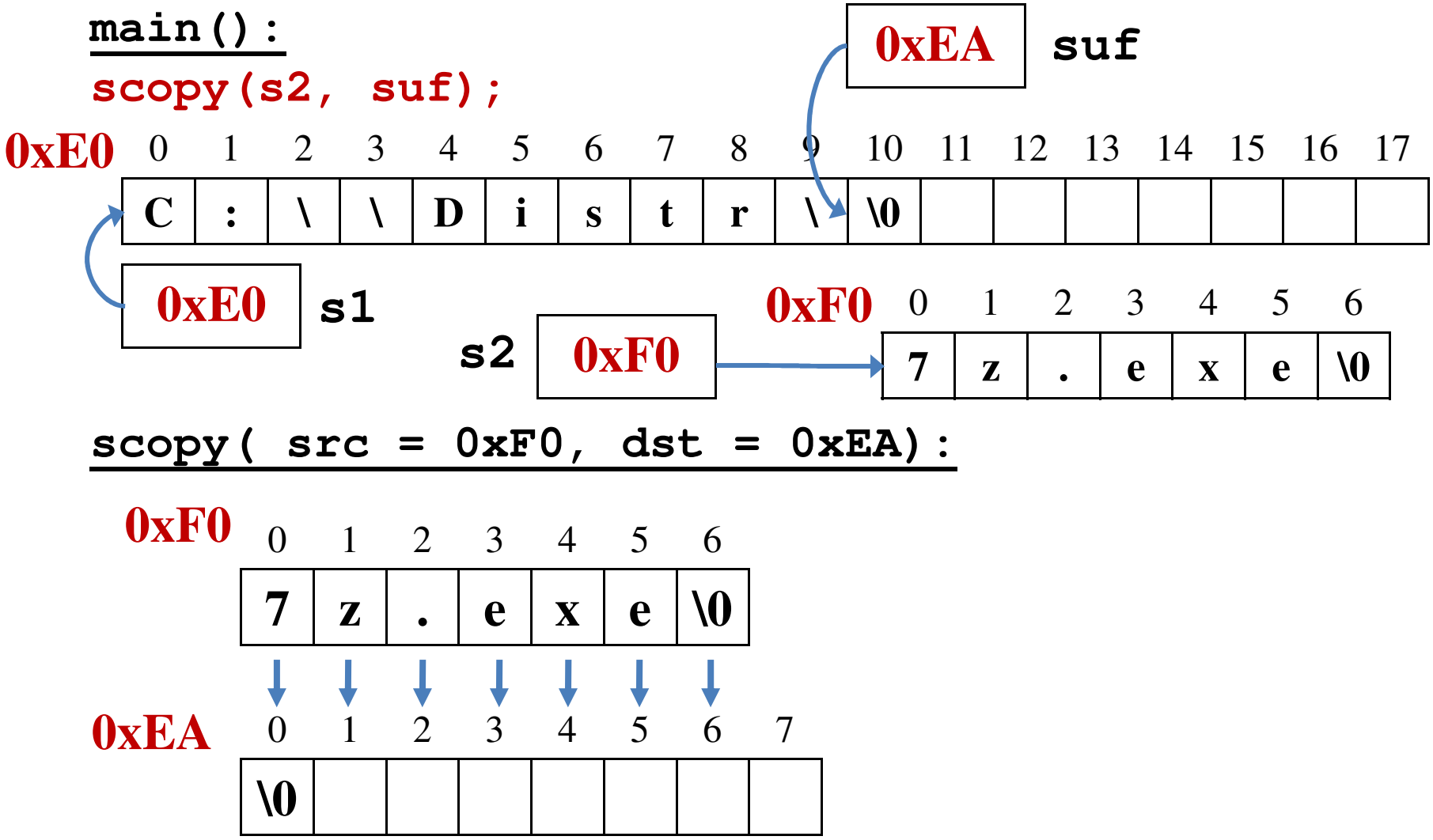
Конкатенация строк (2)

Операция конкатенации обычно реализуется с использованием операции суффикса и операции копирования.

```
main():
char s1[128] = "C:\\Distr\", s2[] = "7z.exe";
int len = slen(s1); char *suf = s1 + len;
scopy(s2, suf);
```



Конкатенация строк (3)



Получение подстроки по индексам

При обработке строк часто возникает необходимость выделить подстроку начиная с индекса begin и заканчивая индексом end. Например, для следующей строки и индексов begin = 8 и end = 10, результат должен быть "the".

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
str	t	u	r	n		t	o		t	h	e		l	e	f	t

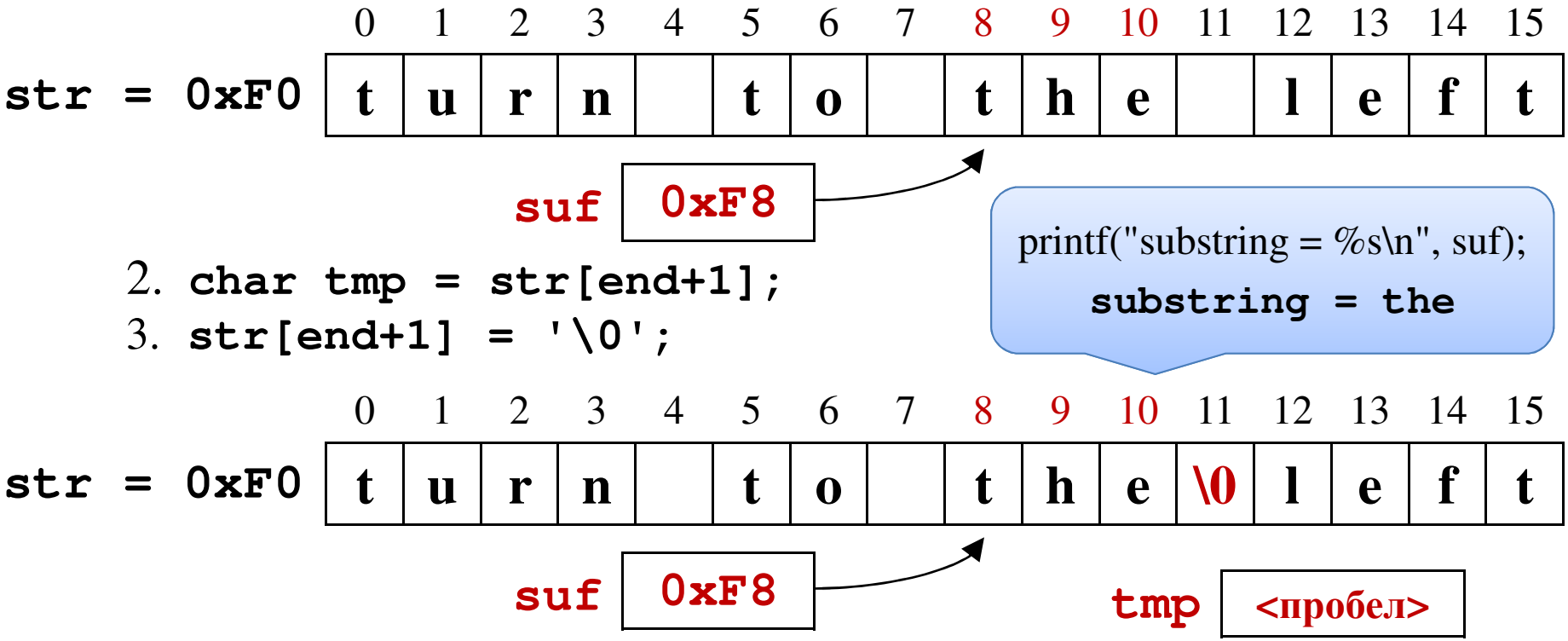
Одним из возможных решений данной задачи является копирование заданных символов в новую строку:

```
int i, j;
char substr[64];
for(i = begin, j = 0; i <= end; i++, j++){
    substr[j] = str[i];
}
substr[j] = '\0';
```

Получение подстроки по индексам (2)

Существует также решение, не требующее создания дополнительной памяти. Оно основано на получении суффикса и префикса исходной строки:

1. `char *suf = str + begin;`

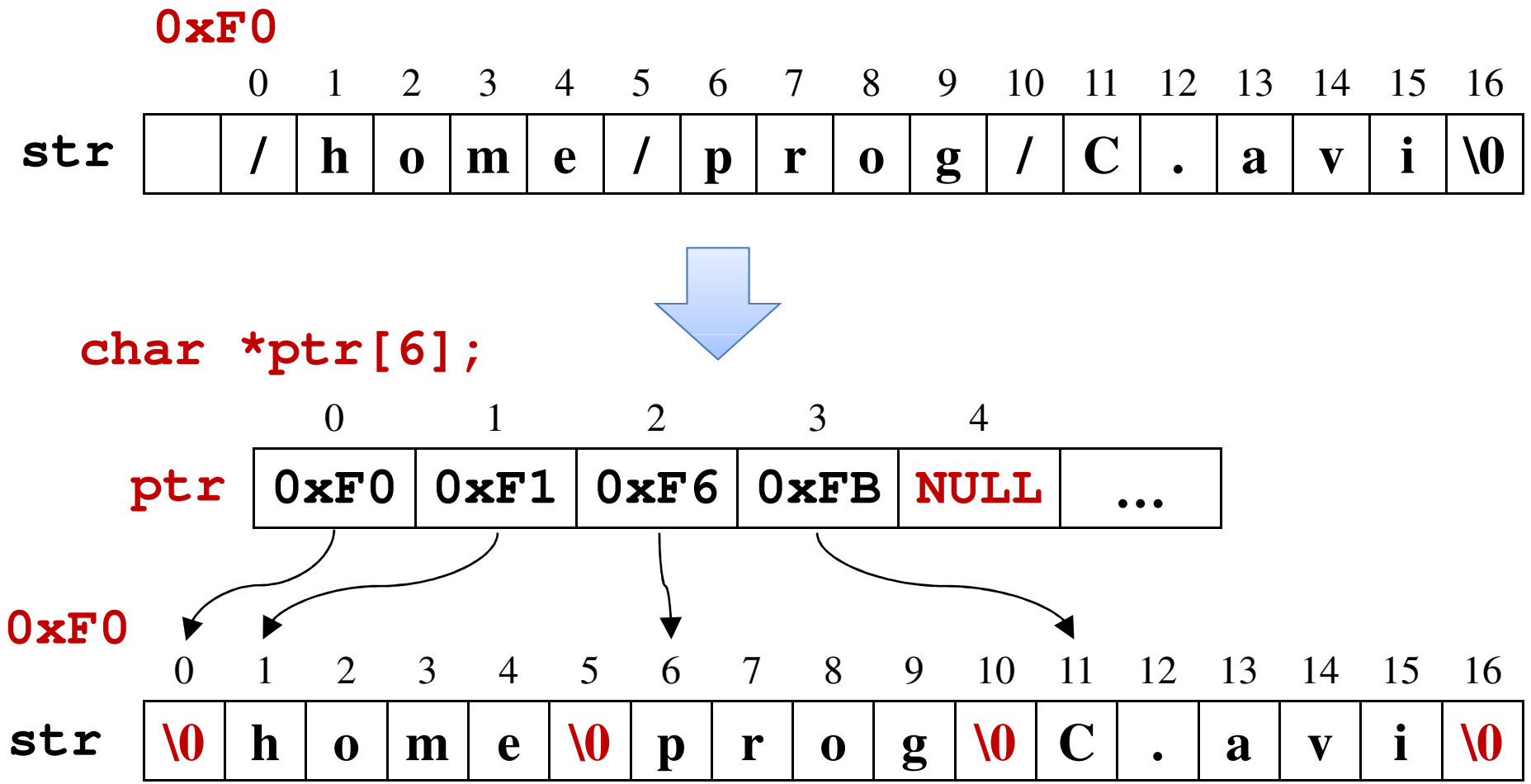


Разбиение строки на поля (**stok**)

Данная подзадача является базовой для выполнения лабораторной работы №4. Она предполагает разбиение заданной строки на **поля**, разделенные заданным **символом**.

Строка: "/home/alex/video/prog/C.avi"
Разделитель: '/'
Результат:
"home", "alex", "video", "prog", "C.avi"

Разбиение строки на поля (2)



Разбиение строки на поля (3)

Строка: "/home/alex/video/prog/C.avi"

Разделитель: '/'

Результат:

"home", "alex", "video", "prog", "C.avi"

Прототип stok:

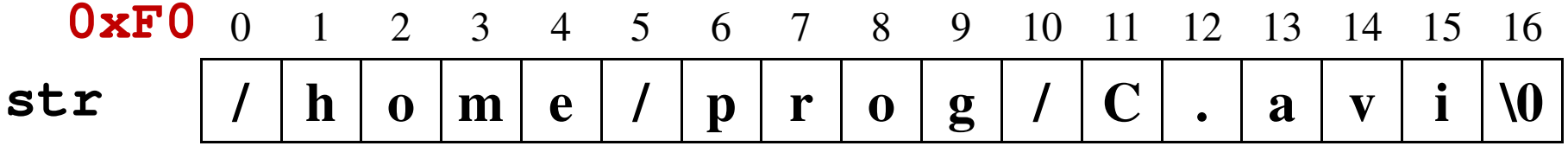
```
???? stok( ??? );
```

Что необходимо передать в функцию? (входные параметры)

Каким будет результат? (выходные значения)

Разбиение строки на поля (4)

```
int stok(char str[], char delim, char *ptr[])
{
    char *suf = str;
    ptr[0] = str; // первое поле – начало str
    int i, j = 1; // j – счетчик полей
    while( ( i = schr(suf, delim) ) >= 0 ){
        suf[i] = '\0';
        suf = suf + i + 1;
        ptr[j] = suf;
        j++;
    }
    return j;
}
```



Разбиение строки на поля (применение)

Разбиение простого файлового пути на директории и файлы:

```
strtok("/home/alex/video/prog/C.avi", '/') =>
   "home", "alex", "video", "prog", "C.avi"
```

Разбиение набора входных путей:

```
strtok("/a/b/c+a/b/d/+c/d/e", '+') =>
      "/a/b/c", "a/b/d", "c/d/e"
```

Разбиение сетевого адреса на элементы:

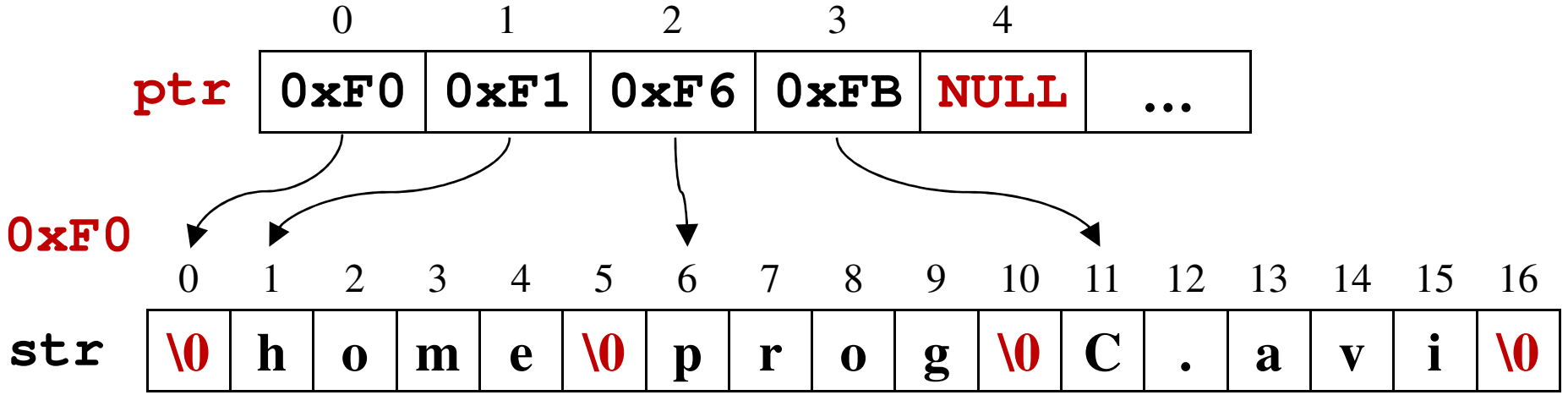
```
strtok("csc.sibsutis.ru", '.') =>
      "csc", "sibsutis", "ru"
strtok("192.168.1.1", ".") =>
      "192", "168", "1", "1"
```

Как проверить:
0 ≤ "192" ≤ 255
0 ≤ "168" ≤ 255
0 ≤ "1" ≤ 255

Восстановление строки после разбиения на поля (**suntok**)

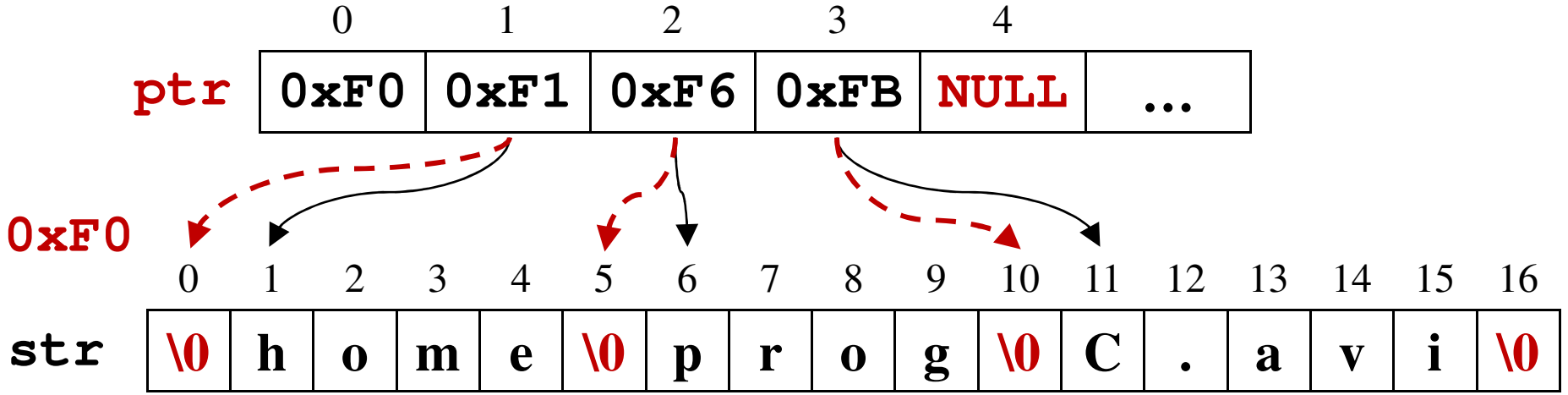
Рассмотренный способ разбиения строки на поля является деструктивным, так как изменяет разбиваемую строку.

В некоторых ситуациях, например, при проверке входных строк, желательно не нарушать целостность строки. Имеющихся после разбиения данных достаточно, чтобы провести быстрое восстановление исходной строки.



Восстановление строки после разбиения на поля (2)

```
int suntok(char str[], char delim, char *ptr[], int cnt)
{
    int i;
    for(i = 1; i < cnt; i++){
        *(ptr[i] - 1) = delim;
    }
}
```



Перевод символов строки в число

Одной из часто возникающих задач при работе со строками является преобразование строки, содержащей число в ее числовое представление.

```
1 #include <stdio.h>
2 int main()
3 {
4     char s1[] = "128";
5     int i, j = 10;
6     i = s1 + j;
7    printf("i = %d\n",i);
8 }
```

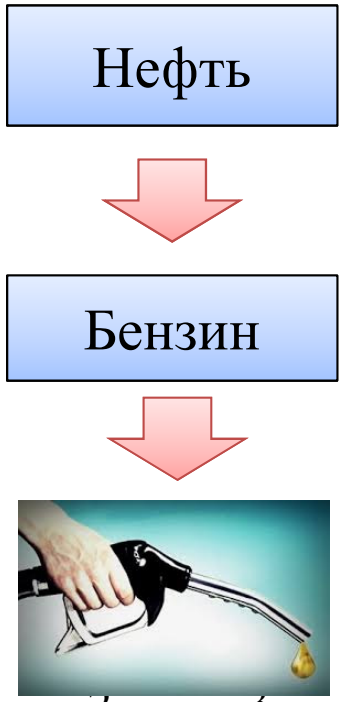
? Нефть = Бензин ?

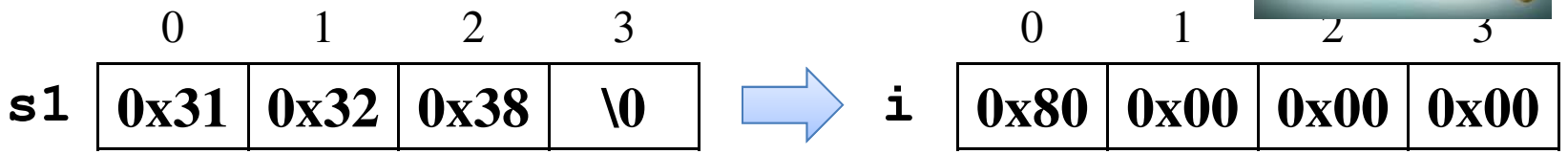
```
$ gcc -o bad_strnum bad_strnum.c
bad_strnum.c: In function `main':
bad_strnum.c:6:5: warning: assignment makes integer
   from pointer without a cast
```

Перевод символов строки в число (2)

Одной из часто возникающих задач при работе со строками является преобразование строки, содержащей число в целочисленное значение.

```
1 #include <stdio.h>
2 int main()
3 {
4     char s1[] = "128";
5     int i, j = 10;
6     i = atoi(s1);
6     i = i + j;
7     printf("i = %d\n",i);
8 }
```





символ → цифра

Число, записанное в виде строки, представляет из себя набор ASCII-кодов, которые напрямую не связаны с числовыми значениями символов:

```
char s1[] = "128";
```

	0	1	2	3
s1	0x31	0x32	0x38	\0

	.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	:	;	<	=	>	?

Однако:

1) любой ASCII-код является числом, которое допускает арифметические действия;

2) ASCII-коды символов-цифр располагаются непрерывно друг за другом.

символ → цифра (2)

Для получения числового эквивалента символа-цифры достаточно из его ASCII-кода отнять код '0'.

```
char s1[] = "128";
```

	0	1	2	3
s1	0x31	0x32	0x38	\0

	.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	:	;	<	=	>	?

```
s1[0] - '0' = 0x31 - 0x30 = 1
s1[2] - '0' = 0x32 - 0x30 = 2
s1[3] - '0' = 0x38 - 0x30 = 8
```

строка → число

Первая цифра в строке – самая старшая! Пусть строка s содержит только корректные символы, n – длина s, тогда целочисленное представление s выглядит следующим образом:

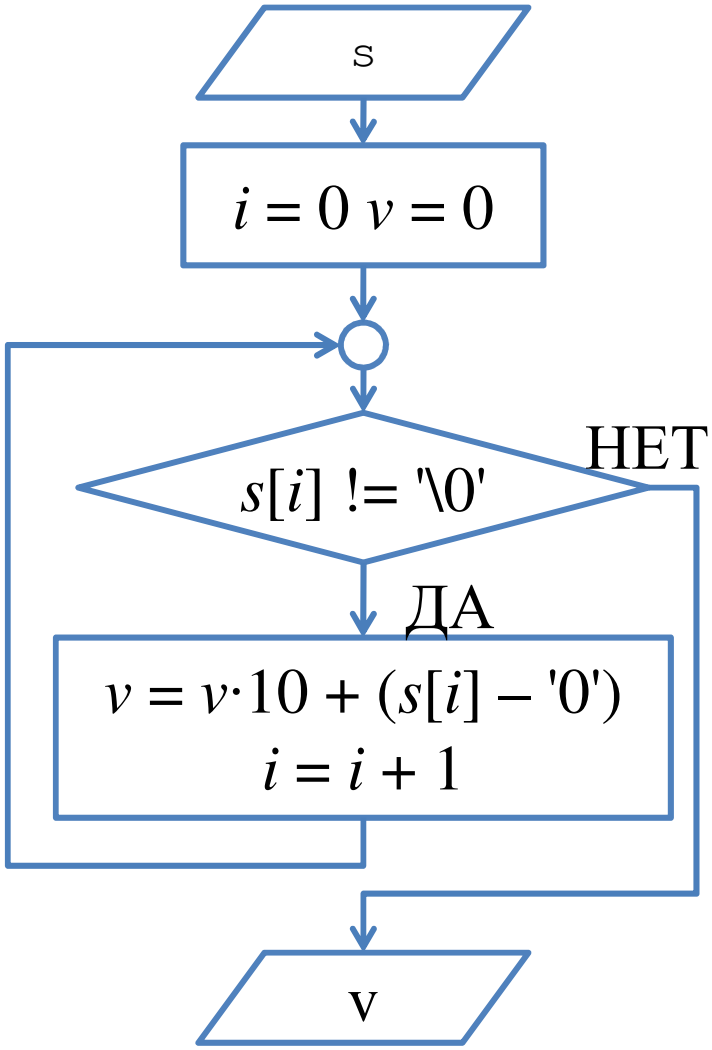
$$v = (s[0] - '0')\cdot 10^{n-1} + (s[1] - '0')\cdot 10^{n-2} + (s[2] - '0')\cdot 10^{n-3} + \ldots + (s[n-2] - '0')\cdot 10^{1} + (s[n-1] - '0')\cdot 10^{0}$$

```
char s1[] = "128";
```

	0	1	2	3
s1	0x31	0x32	0x38	\0

$$v = (0x31 - 0x30)\cdot 10^{3-1} + (0x32 - 0x30)\cdot 10^{3-2} + (0x38 - 0x30)\cdot 10^{3-3} = 1\cdot 10^{2} + 2\cdot 10^{1} + 8\cdot 10^{0} = 128$$

строка → число (2)



s[] = "1024"; $v = 0$;

i = 0: s[0] == '1'

$v = 0 \cdot 10 + (s[0] - '0') =$

$= 0 \cdot 10 + (0x31 - 0x30) =$

$= 0 \cdot 10 + 1 = 1$

i = 1: s[1] == '0'

$v = 1 \cdot 10 + (0x30 - 0x30) =$

$= 10$

i = 2: s[2] == '2'

$v = 10 \cdot 10 + (0x32 - 0x30) =$

$= 102$

i = 3: s[3] == '4'

$v = 102 \cdot 10 + (0x34 - 0x30) =$

$= 1024$

Поиск подстроки в строке

Поиск подстроки в строке (string searching/string matching) – типичная задача, возникающая при обработке текста. Данная задача предполагает наличие *образца* (например, "the") , который необходимо найти, и *текста*, в котором осуществляется поиск:

"… Please, turn to **the** left **the**n turn to **the** right …"

Большинство строковых алгоритмов (string algorithms), решающих данную задачу нацелены на поиск *первого вхождения образца* в *текст*.

К настоящему моменту известен ряд эффективных строковых алгоритмов, таких как алгоритм Бойера-Мура, Кнутта-Моррисона-Пратта, Рабина-Карпа* и т.д.

В рамках лабораторных работ будет рассмотрен "наивный" алгоритм, который не является эффективным, однако позволяет решать указанную задачу корректно.

* Данные алгоритмы рассматриваются в рамках курсового проекта

Поиск подстроки в строке (2)

Образец:

	0	1	2	3
`p`	t	h	e	\0

Текст:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
`txt`	t	u	r	n		t	o		t	h	e		l	e	f	t	

	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
	t	h	e	n		t	u	r	n		t	o		t	h	e	\0

- Алгоритм поиска *p* в *txt*?
- Результат поиска?

Поиск подстроки в строке (3)

```
int sstr(char txt[], char p[])
{
    char *suf = txt;
    int len = slen(p);
    int i, pos = -1;
    while( ( ( i = schr(suf,p[0]) ) >= 0) && (pos < 0) ){
        char tmp;
        suf = suf + i;
        tmp = suf[len];
        suf[len] = '\0';
        if( sequal(suf, p) ){ // посимвольное сравнение строк
            pos = suf - txt; // разность указателей = индекс
        }
        suf[len] = tmp;
        suf++;
    }
    return pos;
}
```

Поиск подстроки в строке (4)

p

0	1	2	3
t	h	e	\0

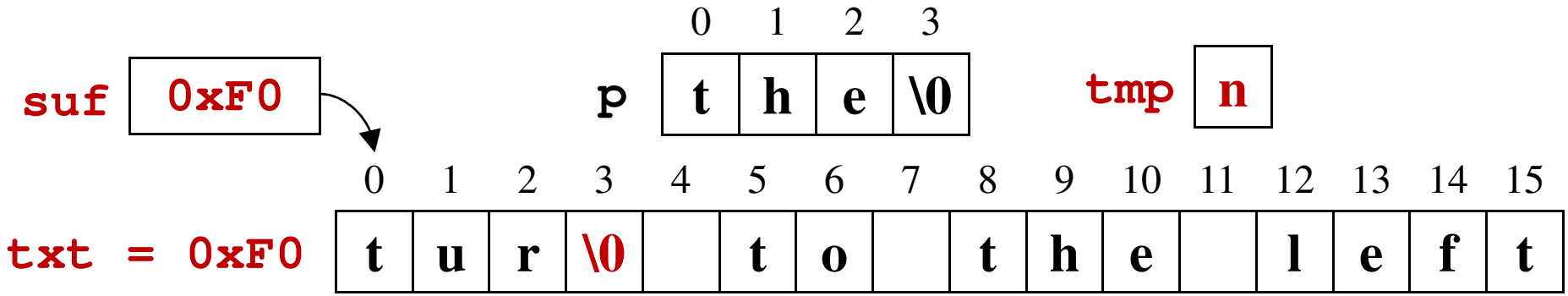
txt

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
t	u	r	n		t	o		t	h	e		l	e	f	t

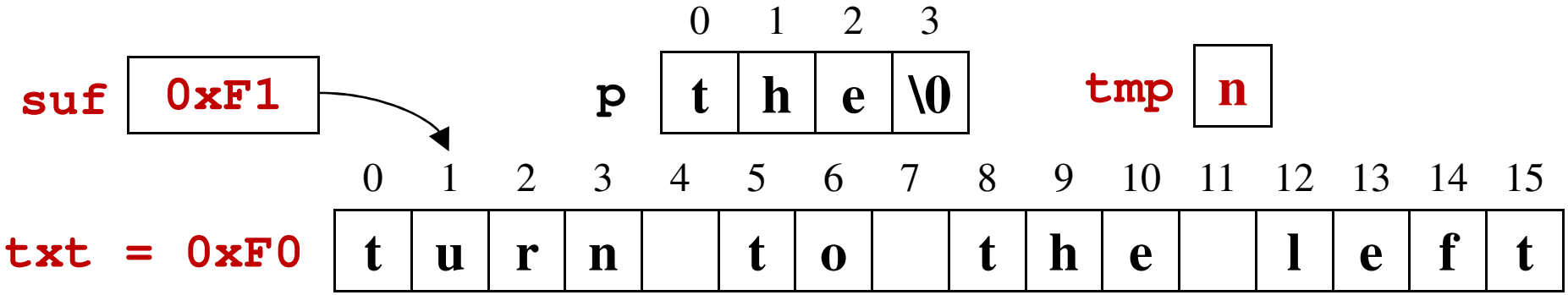
```
pos = -1
while( ( ( i = schr(suf,p[0]) ) >= 0) && (pos < 0) ){
    char tmp;
    suf = suf + i;
    tmp = suf[len];
    suf[len] = '\0';
    if( sequal(suf, p) ){ // посимвольное сравнение строк
        pos = suf - txt; // разность указателей = индекс
    }
    suf[len] = tmp;
    suf++;
}
```

Поиск подстроки в строке (5)

```
suf = txt, schr(suf,p[0]) == 0
        suf = suf + i;
tmp = suf[len]; suf[len] = '\0';
        sequal(suf, p) == 0
```

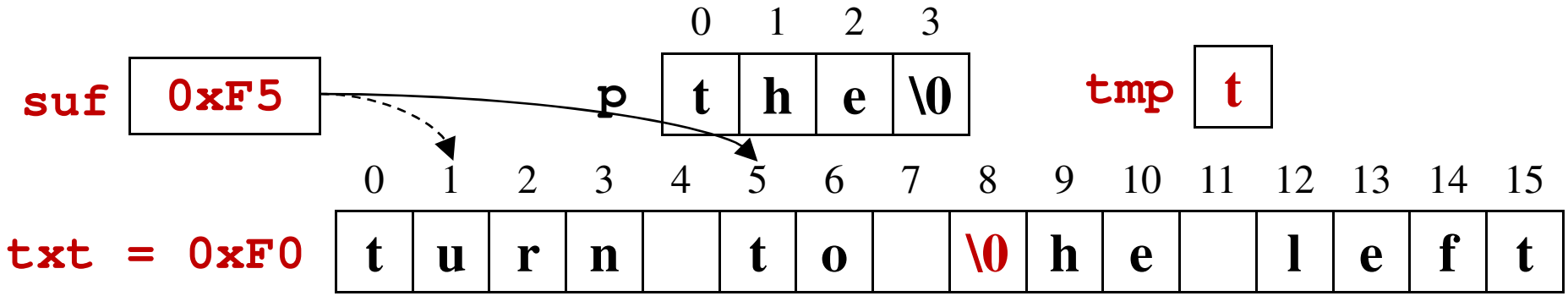



```
suf[len] = tmp;
suf++;
```

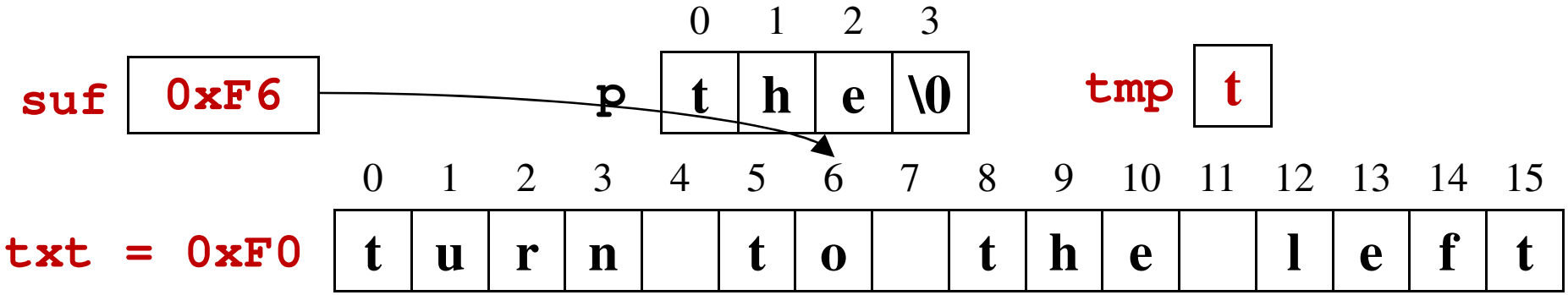


Поиск подстроки в строке (6)

```
schr(suf,p[0]) == 4
   suf = suf + i;
tmp = suf[len]; suf[len] = '\0';
   sequal(suf, p) == 0
```

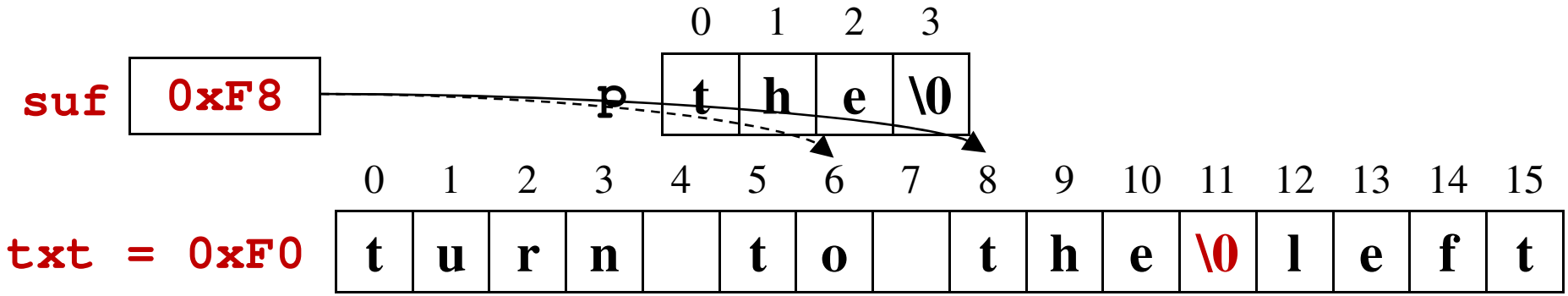



```
suf[len] = tmp;
   suf++;
```

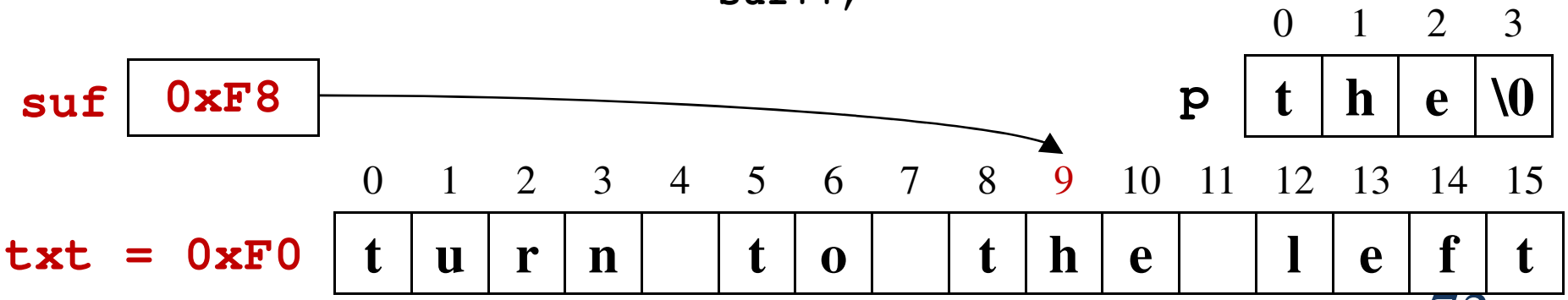


Поиск подстроки в строке (7)

```
schr(suf,p[0]) == 1
    suf = suf + i;
tmp = suf[len]; suf[len] = '\0';
sequal(suf, p) == 1
```




```
pos = suf - txt = 0xF8 - 0xF0 = 8
suf[len] = tmp;
suf++;
```



Изменение фрагмента строки

Еще одной типичной задачей при обработке текста является изменение фрагмента строки. При выполнении этой операции важным является совпадение длин исходного и нового фрагментов. Так как строки в языке СИ хранятся в символьных массивах, то не существует возможности "расширить" или "укоротить" фрагмент, за которым имеется текст.

Например, заменить в строке "turn to the left" подстроку "left" на "right" не составляет проблемы несмотря на несовпадение длин фрагментов:

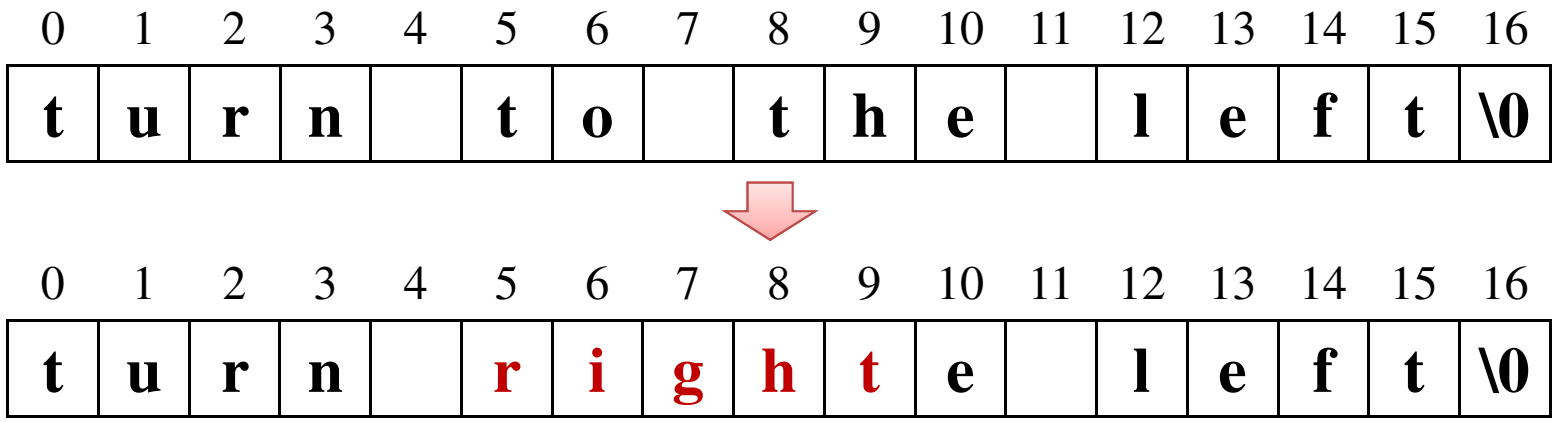
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
t	**u**	**r**	**n**		**t**	**o**		**t**	**h**	**e**		**l**	**e**	**f**	**t**	**\0**

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
t	**u**	**r**	**n**		**t**	**o**		**t**	**h**	**e**		**r**	**i**	**g**	**h**	**t**	**\0**

Изменение фрагмента строки (2)

Еще одной типичной задачей при обработке текста является изменение фрагмента строки. При выполнении этой операции важным является совпадение длин исходного и нового фрагментов. Так как строки в языке СИ хранятся в символьных массивах, то не существует возможности "расширить" или "укоротить" фрагмент, за которым имеется текст.

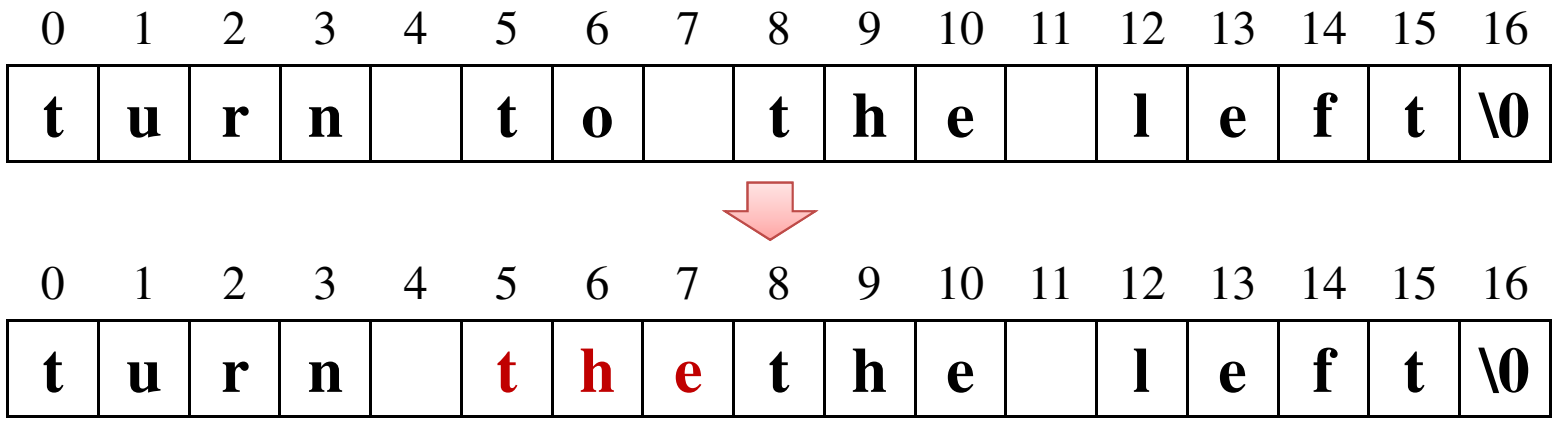
Однако, вставка слова "right" вместо "to" приведет к нарушению целостности строки:



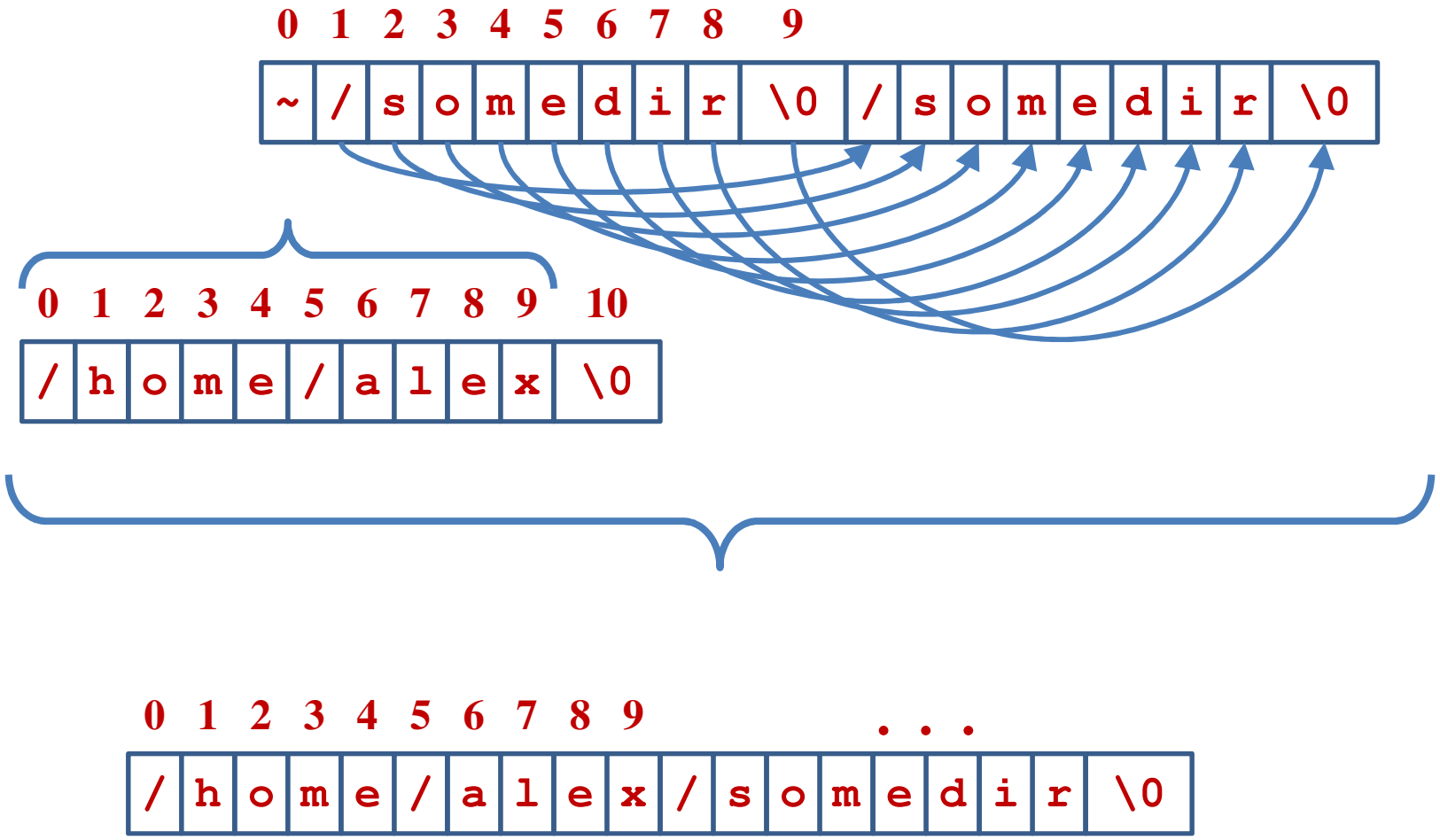
Изменение фрагмента строки (3)

Еще одной типичной задачей при обработке текста является изменение фрагмента строки. При выполнении этой операции важным является совпадение длин исходного и нового фрагментов. Так как строки в языке СИ хранятся в символьных массивах, то не существует возможности "расширить" или "укоротить" фрагмент, за которым имеется текст.

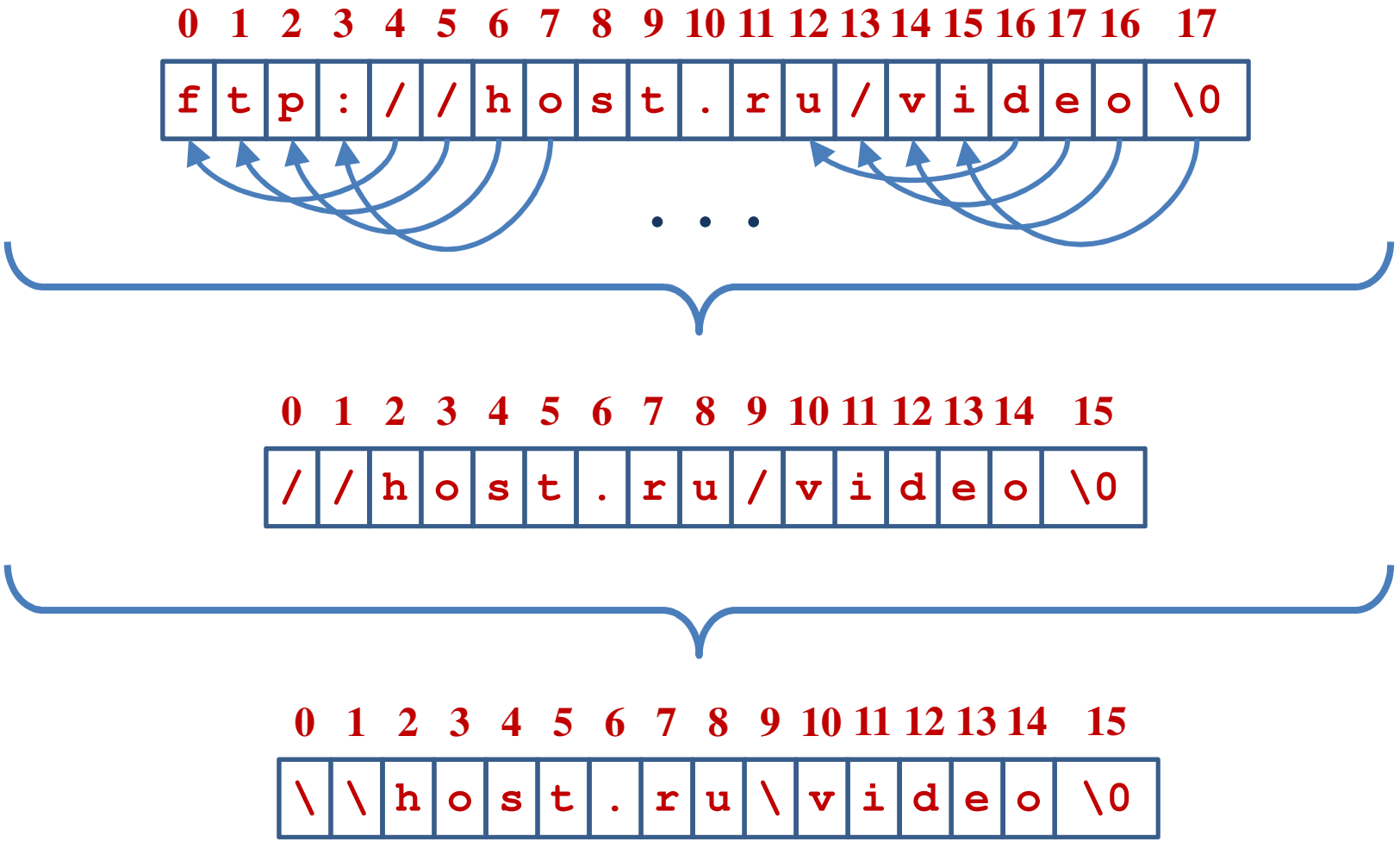
Аналогично, при замене "to the" на "the", приведет к нарушению целостности строки:



Вставка фрагмента большего размера



Вставка фрагмента меньшего размера

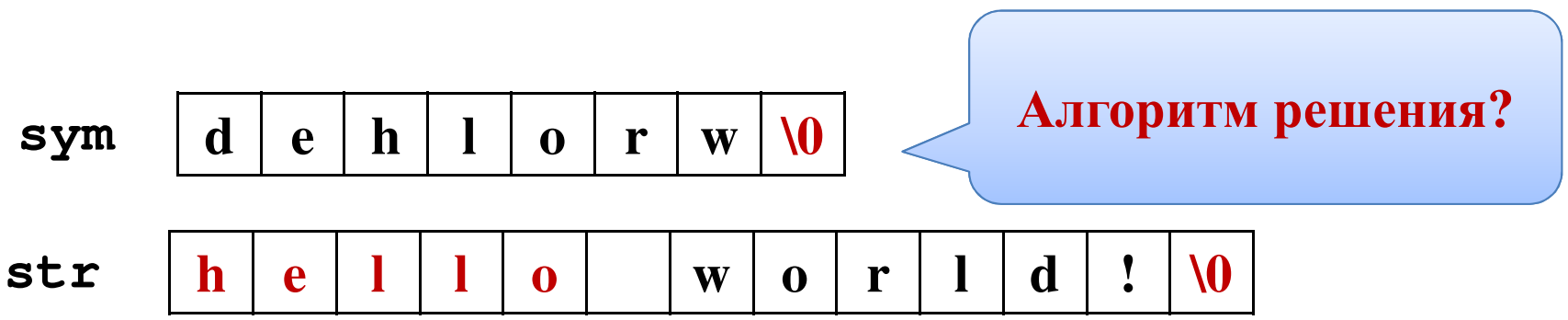


Определение подстроки, состоящей из допустимых символов (sspn)

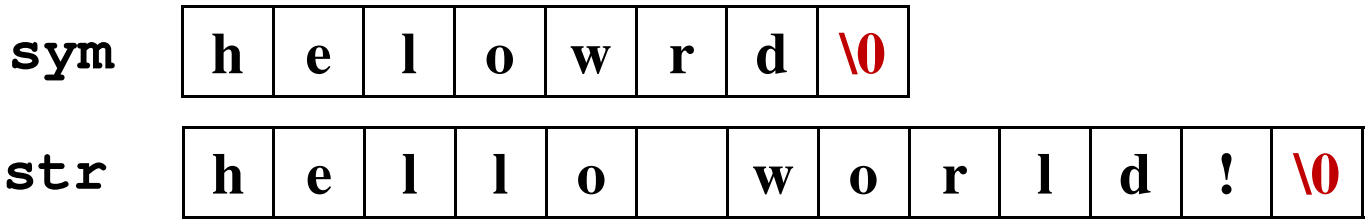
Задача определения подстроки, состоящей из допустимых символов возникает, например:

1) при проверке соответствия входных данных заданному формату.
2) при проверке имени пользователя при его создании/регистрации.

Суть задачи заключается в проверке каждого символа строки на принадлежность некоторому множеству, также заданному при помощи строки. Например, если **`str`** содержит проверяемую строку, а **`sym`** – допустимые символы, то максимальная длина подстроки – 5, так как пробел не входит в **`sym`**.



Определение подстроки, состоящей из допустимых символов (2)




```
int sspn(char str[], char sym[])
{
    int i;
    for( i = 0; str[i] != '\0' ; i++){
        if( schr(sym, str[i] ) < 0 ){
            break;
        }
    }
    return i;
}
```

Определение подстроки, состоящей из допустимых символов (3)

Выполнить пошаговую трассировку функции вручную для приведенных данных:

sym	0	…	9	a	b	…	z	\0

str	a	l	e	x	1	9	8	0	.	1	0	\0	\0

```
int sspn(char str[], char sym[])
{
    int i;
    for( i = 0; str[i] != '\0' ; i++){
        if( schr(sym, str[i] ) < 0 ){
            break;
        }
    }
    return i;
}
```

Проверка строки, не содержащей недопустимых символов (**scspn**)

Задача определения подстроки, не содержащей недопустимых символов возникает, например:

1) при проверке паролей и имен пользователей;

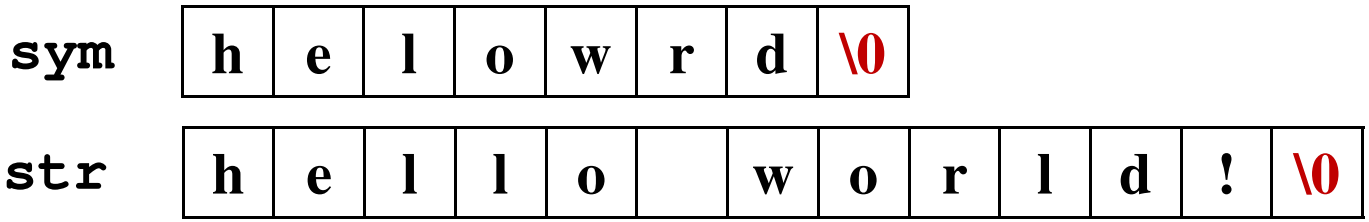
2) при разбиении строки на поля, если разделителей может быть несколько.;

Задача аналогична рассмотренной ранее процедуре sspn определения подстроки, состоящей из допустимых символов. Однако в некоторых ситуациях ее применять удобнее. Например для рассмотренного ранее примера, если нам требуется определить подстроку до разделителя это можно сделать следующим образом:

nsym		\t	\n	\0

str	h	e	l	l	o	\n	w	o	r	l	d	!	\0

Проверка строки, не содержащей недопустимых символов (2)




```
int scspn(char str[], char nsym[])
{
    int i;
    for( i = 0; str[i] != '\0' ; i++){
        if( schr(nsym, str[i] ) >= 0 ){
            break;
        }
    }
    return i;
}
```

Проверка строки, не содержащей недопустимых символов (3)

Выполнить пошаговую трассировку функции вручную для приведенных данных:

sym	.	:	@	!	&	&	\n	\0

str	a	l	e	x	1	9	8	0	.	1	0	\0	\0

```
int scspn(char str[], char sym[])
{
    int i;
    for( i = 0; str[i] != '\0' ; i++){
        if( schr(nsym, str[i] ) >= 0 ){
            break;
        }
    }
    return i;
}
```