

Увод в програмирането

Символни низове. Работа с текст

2017-2018 г.

ФМИ, специалност „Софтуерно инженерство“

В тази презентация са използвани слайдове на
доц. Атанас Семерджиев

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040	 	Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	\$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	((72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051))	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[[123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174	|	
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135]]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177		DEL

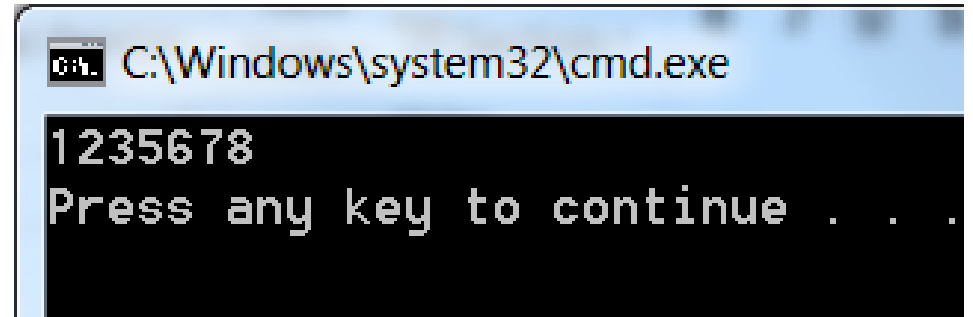
Source: www.LookupTables.com

* Източник на ASCII таблицата: <http://www.asciitable.com/>

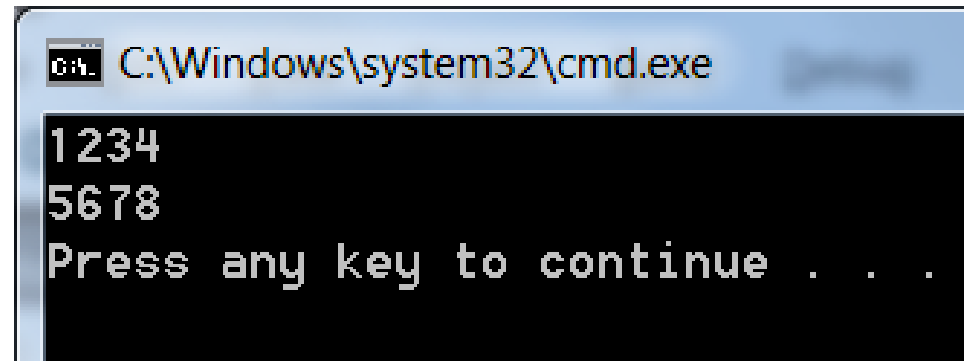
```
int main()
{
    char a = 8;
    cout << "1234" << a << "5678" << endl;
    return 0;
}
```

```
int main()
{
    cout << "1234" << '\n' << "5678" << endl;
    return 0;
}
```

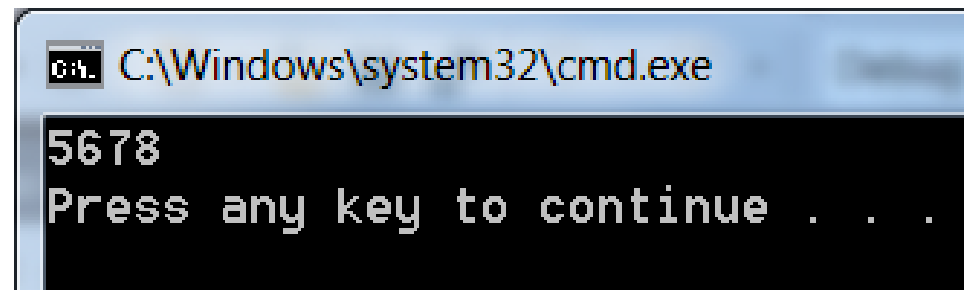
```
int main()
{
    cout << "1234" << '\r' << "5678" << endl;
    return 0;
}
```



```
C:\Windows\system32\cmd.exe
1235678
Press any key to continue . . .
```



```
C:\Windows\system32\cmd.exe
1234
5678
Press any key to continue . . .
```



```
C:\Windows\system32\cmd.exe
5678
Press any key to continue . . .
```

Escape Codes

Литерали за някои по използвани символи, които нямат графично представяне:

Тип	Описание
\n	Нов ред
\r	Carriage return
\b	Връщане на символ (backspace)
\t	Табулация
\a	Аларма/Сигнал

Преобразуване до число

```
int ToInt(char c)
{
    if (c >= '0' && c <= '9')
        return c - '0';

    return 0;
}
```

```
int ToIntBitwise(char c)
{
    if (c >= '0' && c <= '9')
        return c & 0xF;

    return 0;
}
```

Основни побитови операции

$$\begin{array}{r} \sim 1101 \\ \hline 0010 \end{array}$$

$$\begin{array}{r} | 1100 \\ 1010 \\ \hline 1110 \end{array}$$

$$\begin{array}{r} \& 1100 \\ 1010 \\ \hline 1000 \end{array}$$

$$\begin{array}{r} \wedge 1100 \\ 1010 \\ \hline 0110 \end{array}$$

Преобразуване на регистър (letter case)

Към горен регистър (uppercase)

```
char ToUpper(char c)
{
    if (c >= 'a' && c <= 'z')
        return c - ('a' - 'A');

    return c;
}
```

Към долен регистър (lowercase)

```
char ToLower(char c)
{
    if (c >= 'A' && c <= 'Z')
        return c + ('a' - 'A');

    return c;
}
```

Преобразуване чрез побитови операции

'A'	(65)	0	1	0	0	0	0	0	1
'a'	(97)	0	1	1	0	0	0	0	1

```
// 0x20 <--> 0010 0000  
char upper = 'A';  
char lower = upper | 0x20;
```

```
// 0xDF <--> 1101 1111  
char lower = 'a';  
char upper = lower & 0xDF;
```


Преобразуване на регистър (letter case)

Към горен регистър (uppercase)

```
char ToUpperBitwise(char c)
{
    if (c >= 'a' && c <= 'z')
        return c & 0xDF;

    return c;
}
```

Към долен регистър (lowercase)

```
char ToLowerBitwise(char c)
{
    if (c >= 'A' && c <= 'Z')
        return c | 0x20;

    return c;
}
```

СИМВОЛНИ НИЗОВЕ

- Последователност от символи
 - Последователност от 0 символа се нарича празен низ
- Представяне в C++: Масив от символи (char), в който след последния символ в низа е записан т.нар. терминиращ символ '\0'
 - '\0' е първият символ в ASCII таблицата, с код 0

```
int main()
{
    char word_1[] = { 'H', 'e', 'l', 'l', 'o', '\0' };
    char word_2[6] = { 'H', 'e', 'l', 'l', 'o' };
    char word_3[100] = "Hello";
    // char word_4[5] = "Hello"; Грешно!!!
    char word_5[6] = "Hello";
    char word_6[5] = { 'H', 'e', 'l', 'l', 'o' };

    cout << "word_1 is: " << word_1 << endl;
    cout << "word_2 is: " << word_2 << endl;
    cout << "word_3 is: " << word_3 << endl;
    cout << "word_5 is: " << word_5 << endl;
    cout << "word_6 is: " << word_6 << endl;

    return 0;
}
```

C:\Windows\system32\cmd.exe

word_1 is: Hello

word_2 is: Hello

word_3 is: Hello

word_5 is: Hello

word_6 is: Hello||||||||||||||||Hello

Press any key to continue . . .

Включване на специални символи в низови литерали

\'	Апостроф
\"	Кавичка
\?	Питанка
\\	Обратна наклонена черта

// Литерали – символи

char Symbol1 = 'a'; // Малка латинска буква 'a'

char Symbol2 = '\n'; // Нов ред

char Symbol3 = '\x61'; // Код на символ в hex
// Отново буквата 'a'

char Symbol4 = '\141'; // Код на символ в oct
// Отново буквата 'a'

```
char Symbol5 = '''; // НЕВАЛИДНО!!!
```

```
char Symbol6 = '\\''; // Коректно
```

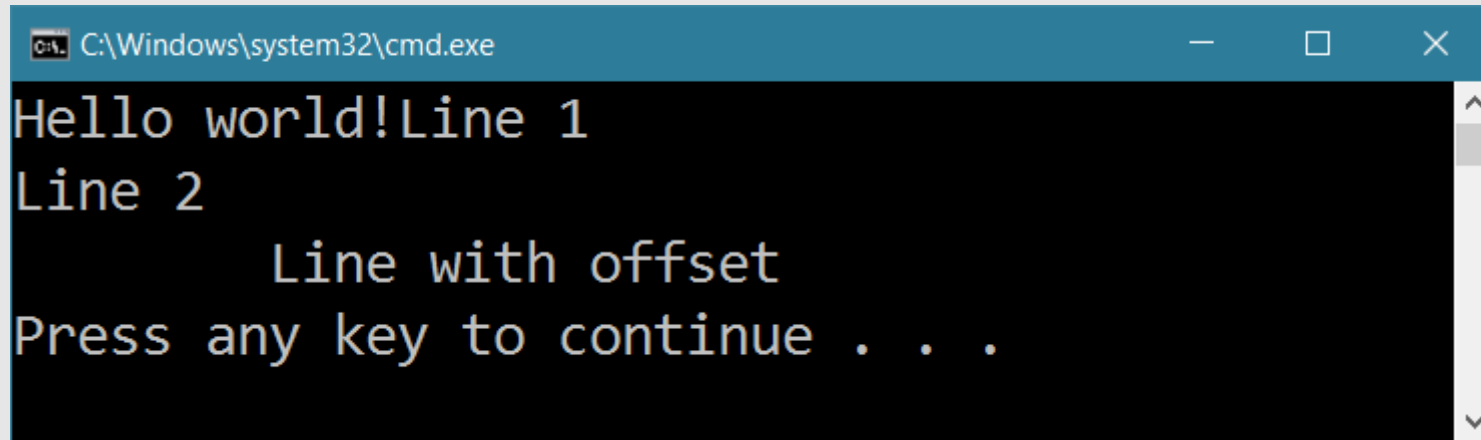
```
char Symbol7 = '\"'; // С кавичката  
// няма проблем
```

```
char Symbol8 = '\\\\'; // Обратна  
// наклонена черта
```

// Литерали – символни низове

```
cout << "Hello world!";
```

```
cout << "Line 1\nLine 2\n\tLine with offset\n";
```

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window has a black background with white text. The output of the program is displayed as follows: 'Hello world!Line 1' on the first line, 'Line 2' on the second line, and 'Line with offset' on the third line, which is indented with a tab character. Below this, the text 'Press any key to continue . . .' is shown. A vertical scrollbar is visible on the right side of the window.

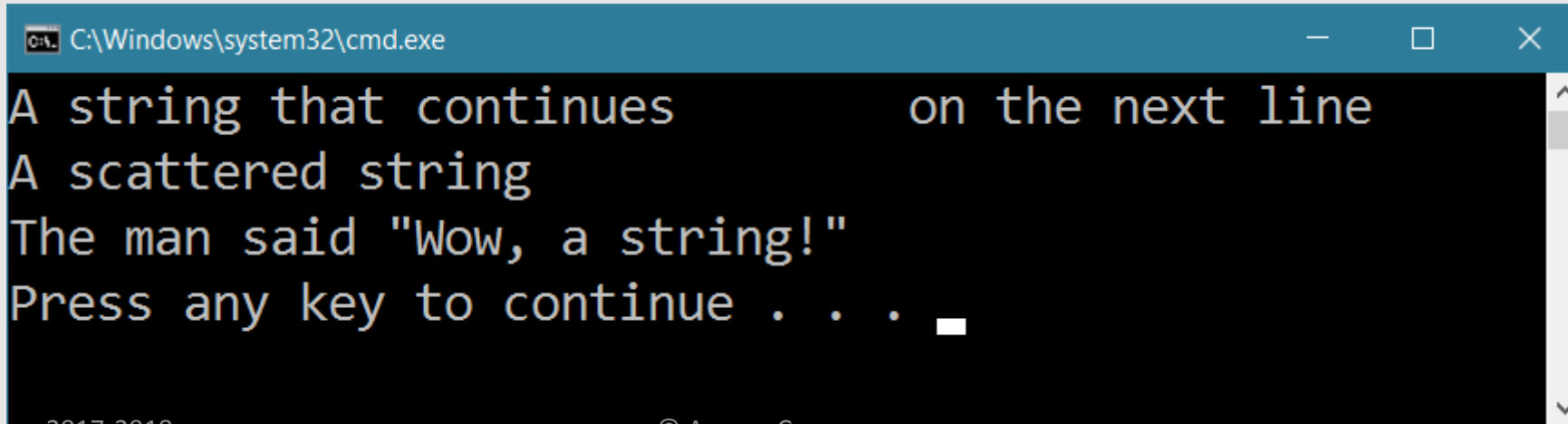
```
C:\Windows\system32\cmd.exe  
Hello world!Line 1  
Line 2  
    Line with offset  
Press any key to continue . . .
```



```
cout << "A string that continues \  
on the next line\n";
```

```
cout << "A " "scattered "  
"string\n";
```

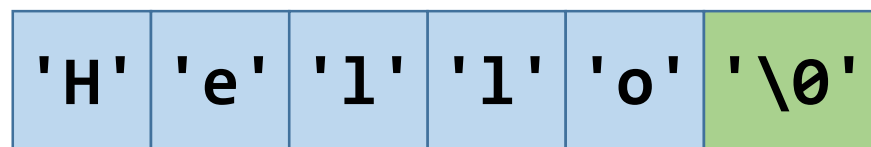
```
cout << "The man said \"Wow, a string!\"\n";
```

A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window has a black background with white text. The output of the C++ code is displayed on four lines: 'A string that continues' followed by 'on the next line' on the same line, 'A scattered string', 'The man said "Wow, a string!"', and 'Press any key to continue . . . ' followed by a white cursor block.

```
C:\Windows\system32\cmd.exe  
A string that continues      on the next line  
A scattered string  
The man said "Wow, a string!"  
Press any key to continue . . .
```

Представяне на низ в паметта

```
char word[] = "Hello";
```

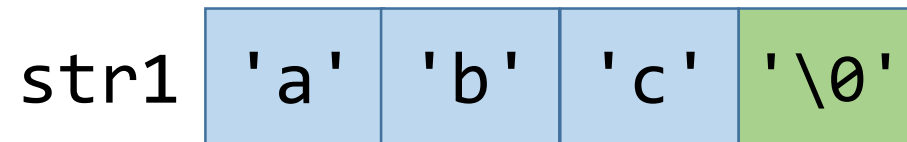


Текстът съдържа 5 символа
Представя се чрез 6 символа

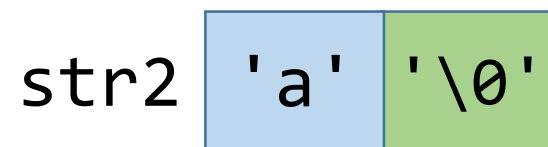
*Терминиращ
елемент*

Представяне на низ в паметта

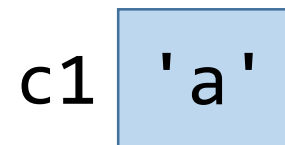
```
char str1[] = "abc";
```



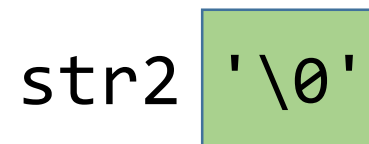
```
char str2[] = "a";
```



```
char c1 = 'a';
```



```
char str3[] = "";
```



```
char c2 = ' '; // Грешка!
```

ВАЖНО!

Между следните има разлика:

'a' – число, кодът на буквата *a*.

"a" – символен низ (масив с два елемента – код на буква и терминиращ символ).

Респективно дадените по-долу редове се изпълняват различно:

```
std::cout << 'a';
```

```
std::cout << "a";
```

Важно!

- Стринговете са масиви и затова НЕ МОЖЕ:
 - Да ги копираме с оператора за присвояване (=)
 - Да ги сравняваме с оператора за сравнение (==)
- За тези цели можем да използваме специално подготвени за целта библиотечни функции.

```
char str[] = "Abc";  
char buffer[100];
```

```
buffer = str;
```

```
if (buffer == str)  
{  
    // ...something...  
}
```

Няма смисъл!!!

Намиране на дължина

```
size_t strlen(const char* str)
{
    const char* pRead = str;

    while (*pRead != '\0')
        pRead++;

    return pRead - str;
}
```

Намиране на дължина

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

    for (pRead = str; *pRead != '\0'; pRead++)
        ;

    return str - pRead;
}
```

Копиране на низ

```
char * strcpy(char* dest, const char* src)
{
    while (*src != '\0')
    {
        *dest++ = *src++;
    }

    return dest;
}
```


Пример за коректна реализация

```
char * strcpy(char* dest, const char* src)
{
    while ((*dest++ = *src++) != '\0')
        ;

    return dest;
}
```

Копиране на низ

```
char * strcpy(char* dest, const char* src)
{
    do
    {
        *dest++ = *src;
    } while (*src++ != '\0');

    return dest;
}
```

ВАЖНО!

Когато копирате един низ s_1 в друг низ s_2 се подсигурете, че:

1. В s_2 има достатъчно място.
2. При копирането прехвърляте и терминиращата нула.

Конкатенация

- Слепваме два низа един до друг
- За целта C++ предлага функцията `strcat`
- За резултантния низ трябва да са изпълнени:
 1. В него трябва да има достатъчно място.
 2. Той трябва да е правилно терминиран.

```
// Конкатенация на dest и src
char * strcat(char* dest, const char* src)
{
    char *p = dest;

    while (*p != '\0')
        p++;

    strcpy(p, src);

    return dest;
}
```

Сливане на няколко низа

```
char partA[] = "Hello";  
char partB[] = " ";  
char partC[] = "world!";
```

```
char buffer[100];
```

```
strcpy(buffer, partA);  
strcat(buffer, partB);  
strcat(buffer, partC);
```

Сливане на няколко низа

```
char partA[] = "Hello";  
char partB[] = " ";  
char partC[] = " world!";
```

```
char buffer[100] = '\0';
```

```
strcat(buffer, partA);  
strcat(buffer, partB);  
strcat(buffer, partC);
```

// Конкатенация със слепващ елемент

```
char partA[] = "Hello";  
char partB[] = "from";  
char partC[] = "FMI!";
```

```
char buffer[100];
```

```
strcpy(buffer, partA);  
strcat(buffer, " ");  
strcat(buffer, partB);  
strcat(buffer, " ");  
strcat(buffer, partC);
```


Важно!

- Винаги когато извършвате операции с низове се подsigурявайте, че:
 - Завършват с терминираща нула
 - При копиране в целевия масив има достатъчно място, включително и за терминиращата нула!

```
char str[] = "Hello world!";  
char buffer[5];  
strcpy(buffer, str);
```

Грешка!!!

Неправилно използване на strcpy

```
void MyFunction(const char* Text)
{
    char Buffer[100];

    // Грешка: Не знаем, дали Text не съдържа
    // повече от 100 елемента!
    strcpy(Buffer, Text);

    // ...
}
```

Вариант 1: strcpy_s

```
void MyFunction(const char* Text)
{
    char Buffer[100];

    // strcpy_s връща нула при успех
    if (strcpy_s(Buffer, 100, Text))
        // Обработваме грешката

    // ...
}
```

Вариант 2: strncpy

```
const int STR_SIZE;  
void MyFunction(const char* Text)  
{  
    char Buffer[STR_SIZE];  
  
    size_t Length = strlen(Text);  
  
    strncpy(Buffer, Text, min(Length, STR_SIZE-1))  
  
    //...  
}
```

Вариант 3: външни проверки

```
void MyFunction(const char* Text)
{
    char Buffer[100];

    size_t Length = strlen(Text);

    if(Length >= 100)
        // Обработваме грешката
    else
        strcpy(Buffer, Text)

    // ...
}
```

Лексикографска наредба

AAA < BBB

ABC < ABD

ABC < ABCD

ABC ≠ abc

ABC < abc

Лексикографска наредба

Дадени са два низа:

$$\begin{aligned} A &= a_1 a_2 \dots a_n \\ B &= b_1 b \dots b_m \end{aligned}$$

Имаме, че $A < B$, т.с.т.к е изпълнено едно от следните:

$$\begin{aligned} \exists i \left(i \leq \min(n, m) \wedge a_i < b_i \wedge \forall j < i (a_j = b_j) \right) \\ n < m \wedge \forall i \leq n (a_i = b_i) \end{aligned}$$

```
int strcmp(const char *str1, const char *str2)
{
    unsigned char c1, c2;
    int diff;

    do {
        // str1 may be shorter and char is signed
        c1 = (unsigned char)*str1++;
        c2 = (unsigned char)*str2++;

        diff = c1 - c2;
    } while ((diff == 0) && (c1 != '\0'));

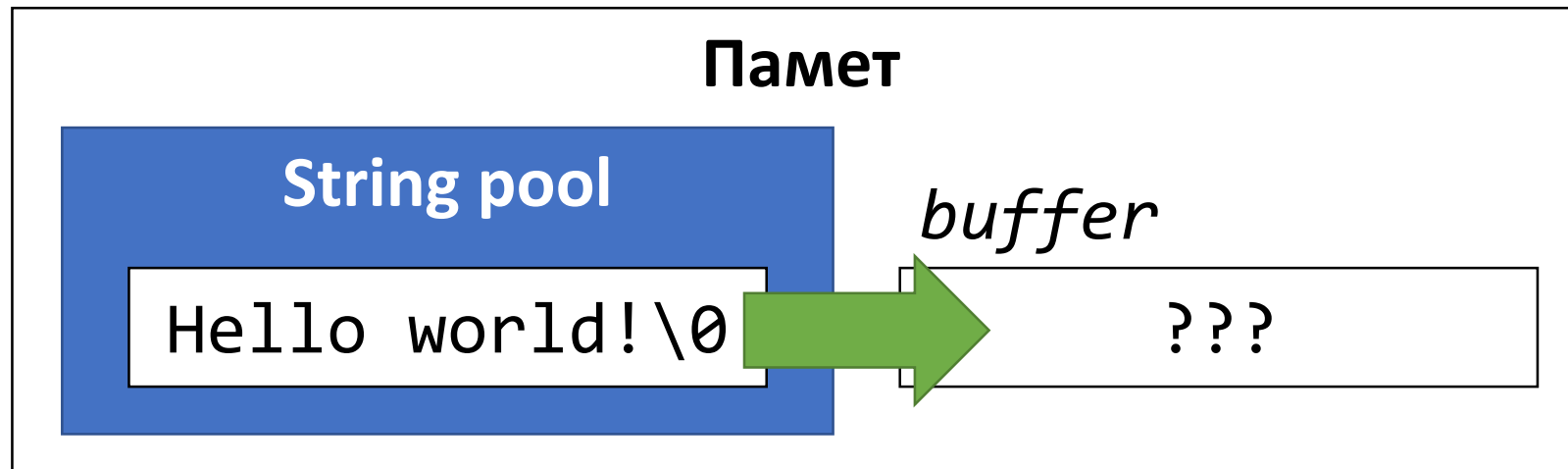
    return diff;
}

/* http://research.microsoft.com/en-us/um/redmond/projects/invisible/src/crt/strcmp.c.htm */
```



```
#include <string.h>

int main()
{
    char buffer[100];
    strcpy(buffer, "Hello world!");
}
```



Валиден код

```
#include <iostream>

void main()
{
    char str1[] = "Abc";
    char str2[] = "Abc";

    str1[0] = 'a';

    std::cout << str1 << endl;
    std::cout << str2 << endl;
}
```

Памет

str1

Abc\0

str2

Abc\0

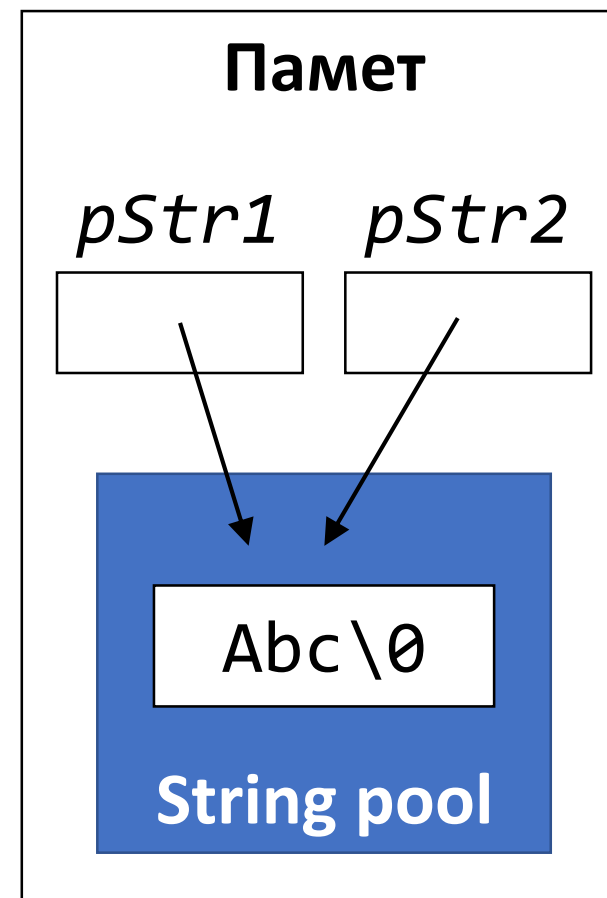
Некоректен код

```
#include <iostream>

int main()
{
    char *pStr1 = "Abc";
    char *pStr2 = "Abc";

    pStr1[0] = 'a';

    std::cout << pStr1 << std::endl;
    std::cout << pStr2 << std::endl;
}
```



(*Възможно представяне)