Лекция 17

Си-строка

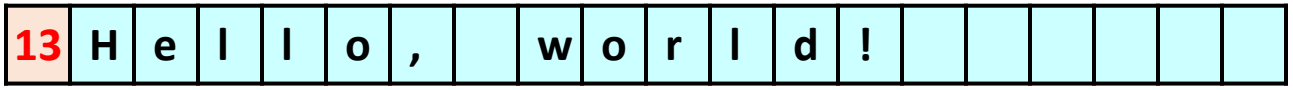
Преподаватель Палехова Ольга Александровна,
кафедра О7 БГТУ «Военмех»

Способы организации строк

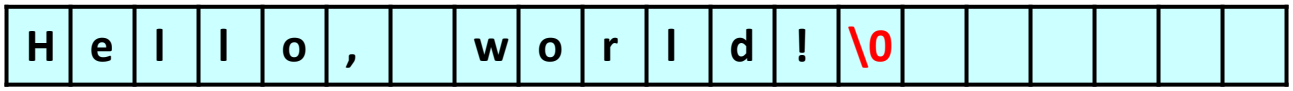
Строковый тип – тип данных, значениями которого является произвольная последовательность (строка) символов алфавита, разновидность символьного массива.

Два подхода к организации строк**:**

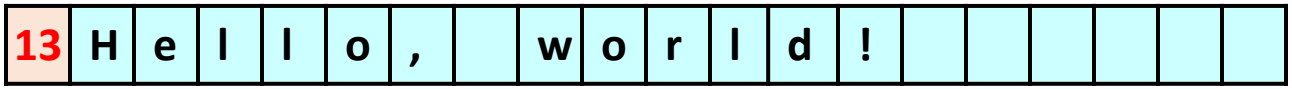
- **Паскаль-строка** (*Pascal strings*) – в служебной области хранится текущая длина строки



- **Си-строка** (*C-strings, нуль-терминированная строка*) – после последнего значащего символа строки записывается специальный символ-ограничитель *'\0"* – **нуль-терминатор** (символ с кодом 0)



Паскаль-строка



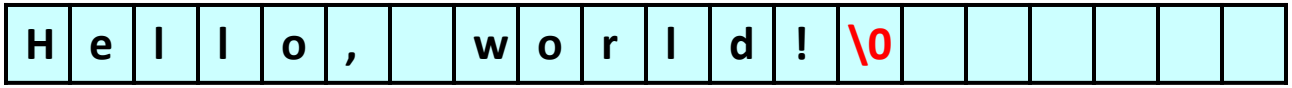
Достоинства:

- простота выполнения операций получения размера строки, добавления символов в конец строки, получения *N*-ого символа с конца строки;
- строка может содержать любые символы;
- простота слежения за выходом за границы строки.

Недостатки:

- проблемы с хранением и обработкой символов произвольной длины;
- увеличение затрат на хранение строк;
- ограничение максимального размера строки.

Си-строка



Достоинства:

- возможность представления строки без создания отдельного типа данных;
- отсутствие ограничения на максимальный размер строки;
- экономное использование памяти.

Недостатки:

- долгое выполнение операций получения длины и конкатенации строк;
- нет контроля за выходом за пределы строки;
- невозможность использовать символ завершающего байта в качестве элемента строки;
- невозможность использовать некоторые кодировки с размером символа в несколько байт.

Символьные массивы и строки

Для представления строк используются **массивы** элементов типа ***char***.

! Как рассматривать содержимое массива, решает программист

```
char s1 [100];
char s2 [5] = {65, 66, 67, 0, 71};
char s3 [5] = {'A', 'B', 'C', '\0'};
char s4 [5] = {'A', 'B', 'C'};
char s5 [5] = "ABC";
char s6 [] = {65, 66, 67, 0};
char s7 [] = {'A', 'B', 'C', '\0'};
char s8 [] = "ABC";
char s9 [] = {'A', 'B', 'C'};
char s10 [] = {65, 66, 67};
```

блок памяти

массив

строки

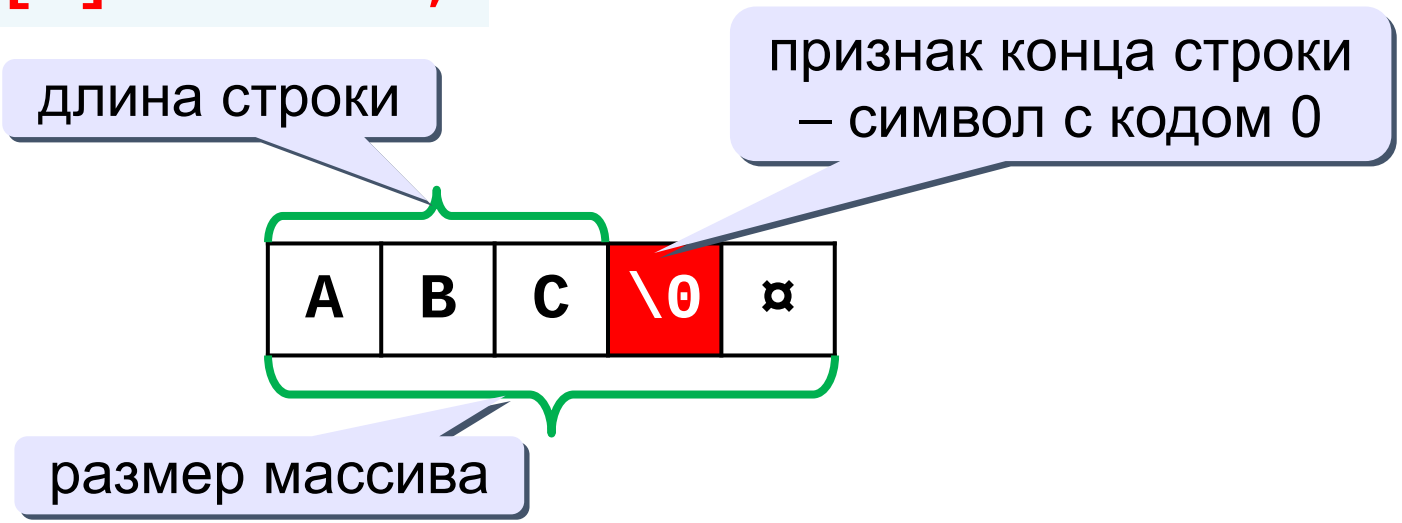
массив или строка

строки

массивы

Строка в языке Си

```
char s5 [5] = "ABC";
```



Особенности:

- **имя** строки – **указатель** на первый символ строки
- **длина** строки как минимум на 1 **меньше размера** массива
- строка **ограничена** первым **нулем**, даже если в массиве их несколько
- **длину** строки можно **вычислить**

Строковый литерал

Строковый литерал (строковая константа) – это последовательность символов, заключенная в двойные кавычки.

Особенности:

- изменять запрещено;
- тип «массив символов»;
- имеет класс памяти **static**;
- при инициализации массива ассоциируется со своим значением;
- при инициализации указателя и во всех инструкциях программы ассоциируется со своим адресом

! Пустая строка "" занимает 1 байт памяти

```
char str [5] = "ABC";
```

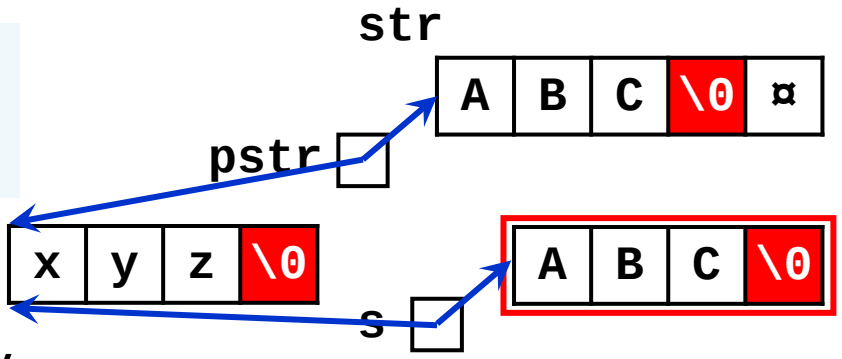
копируются символы

```
char *s = "ABC";
```

записывается адрес

Символьный массив и строковый литерал

```
char str [5] = "ABC";
char *pstr = str;
char *s = "ABC";
```



1. Попытка изменить строку

```
str = "xyz";
pstr = "xyz";
s = "xyz";
```

ошибка

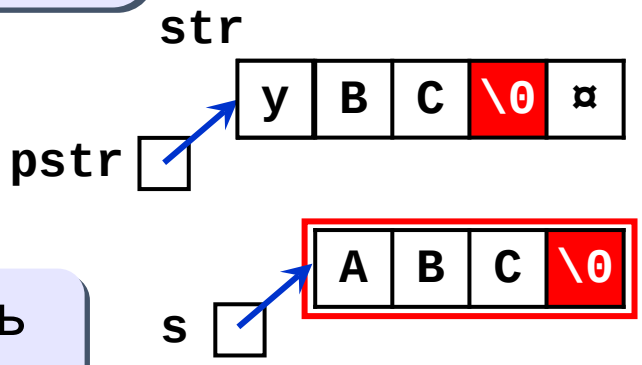
указатели получают адреса других блоков памяти

? Что будет, если str[3]='z' ?

2. Попытка изменить первый символ строки

```
*str = 'y';
*pstr = 'y';
*s = 'y';
```

попытка изменить константу



Ввод и вывод символов

Ввод одного символа:

```
int fgetc (FILE *);
int getc (FILE *);
int getchar (void);
```

откуда читаем, для ввода с клавиатуры пишем **stdin**

то же, что **getc(stdin)**

Вызов:

```
int symbol = getc (stdin);
char c = getchar();
```

Вывод одного символа:

```
int fputc (int, FILE *);
int putc (int, FILE *);
int putchar (int);
```

куда пишем, для вывода на экран указываем **stdout**

то же, что **putc(stdout)**

Вызов:

```
putc ('\n', stdout);
putchar (symbol);
putchar (c);
```

Ввод строки

Способы ввода строк:

- посимвольно в цикле

```
char str [N];
int i, c;
c = getchar();
for ( i = 0 ; i < N-1 && c != '\n' ; i++ )
{
   str[i] = c;
   c = getchar();
}
str[i] = '\0';
```

оставляем место для нуль-терминатора

до нажатия клавиши «Enter»

после последнего символа дописываем нуль-терминатор

? Что будет, если нуль не писать?

Ввод строки

- с помощью функции ***scanf()***

спецификатор	что считывается
s	строка до разделителя (**одно слово!**)
[…]	строка до первого символа, не входящего в перечисление
[^…]	строка до первого символа, входящего в перечисление

```
char s[80];
```

& не нужен

Входная строка:

```
hello, world
```

1) `scanf("%s", s);` — “hello,”
2) `scanf("%[abcdefgh\t ]", s);` — “he”
3) `scanf("%[^ ,.]", s);` — “hello”
4) `scanf("%[123]", s);` — ничего не считано, *scanf()* возвращает *0*, значение s не изменилось

! Во избежание выхода за пределы массива укажите ширину поля ввода

Ввод строки

- с помощью функции ***gets()***

```
char * gets (char *);
```

Особенности:

- строка считывается до символа перехода на новую строку;
- символ перехода на новую строку считывается и заменяется на '\0';
- можно ввести пустую строку;
- нельзя ограничить длину входной строки;
- в случае ошибки возвращает NULL.

Вызов:

```
char str[80];
gets (str);
```

Возможен выход за пределы массива

Ввод строки

- с помощью функции ***fgets()***

откуда читаем, для ввода с клавиатуры пишем **stdin**

```
char * fgets (char *, int, FILE*);
```

размер массива **size**

Особенности:

- второй параметр задает размер массива **size**, считано будет не более **size-1** символа;
- если длина входной строки меньше **size-1**, строка считывается по символ перехода на новую строку;
- символ перехода на новую строку остается в строке, следом записывается '\0';
- в случае ошибки возвращает NULL.

Вызов:

```
char str[80];
fgets (str, 80, stdin);
```

Вывод строки

Способы вывода строк:

- с помощью функции *printf()*

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

```
char s1[80] = "Hi people!", s2[5] = "Hi!\n";
printf(s1);
printf("%s\n", s1);
printf(s2);
```

```
Hi people!Hi people!
Hi!

_
```

- с помощью функции *puts()*

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

```
puts(s1);
puts(s2);
```

```
Hi people!
Hi!


_
```

принудительный переход на новую строку

‘\n’ в строке + принудительный переход на новую строку

! Строка выводится до первого нуля!

Обработка строк

Особенности:

- обработка осуществляется **в цикле**;
- если строка обрабатывается целиком, то обработка осуществляется **до нуль-терминатора**;

Задача: Заменить в строке все пробелы на подчеркивания

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

int main()
{
    char s[80];
    int i;
    fgets (s, 80, stdin);
    for ( i = 0 ; s[i] != '\0' ; i++ )
        if ( s[i] == ' ' )
            s[i] = '_';
    puts (s);
    return 0;
}
```

до нуль-терминатора

Функции библиотеки *string*

Включение заголовочного файла

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

Две группы функций:

- начинающиеся на ***str*** работают **до нуль-терминатора**;
- начинающиеся с ***mem*** работают с объектами как с **массивами байтов** (все указатели имеют тип *void**)

void ***memcpy**(s,ct, n)	копирует *n* байт из *ct* в *s* и возвращает *s*
void ***memmove**(s,ct,n)	делает то же самое, что и *memcpy*, но работает и в случае "перекрывающихся" объектов.
int **memcmp**(cs, ct, n)	сравнивает первые *n* байт *cs* и *ct*; выдает тот же результат, что и функция *strcmp*
void ***memchr**(cs, c, n)	возвращает указатель на первое вхождение байта со значением *c* в *cs* или, если среди первых *n* байт значение *c* не встретилось, NULL
void ***memset**(s, c, n)	размещает однобайтовое значение *c* в первых *n* байтах блока *s* и возвращает *s*

Функции для работы со строками

Длина строки (до нуль-терминатора):

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

Сравнение строк:

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

возвращает значение <0, если первая строка лексикографически меньше второй; 0, если строки равны; >0, если первая строка больше второй

s1	s2	strcmp(s1, s2)
"AA"	"AA"	0
"AB"	"AA"	>0
"AA"	“a"	<0
"AA"	"A"	>0

s1	s2	strcmp(s1, s2)
“111"	“9"	<0
“+0"	“-0"	<0
“11"	“A"	<0
“aB"	“Xyz"	>0

```
int strcmpi (const char *, const char *);
```

то же, что и *strcmp()*, но без учета регистра латинских букв

Копирование строк

Копирование строки:

куда

откуда

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

```
char s1[10] = "Hi!", s2[10] = "Hello!";
strcpy (s1, s2);
```

s1

H	e	l	l	o	!	\0			

s2

H	e	l	l	o	!	\0			

```
char s1[10] = "Hi!", s2[10] = "Hello!";
strcpy (s2, s1);
```

s1

H	i	!	\0						

s2

H	i	!	\0	o	!	\0			

```
char s1[10] = "Hi!", s2[10] = "Hello!";
strcpy (s1+2, s2+3);
```

s1

H	i	l	o	!	\0				

s2

H	e	l	l	o	!	\0			

Копирование строк

Копирование части строки:

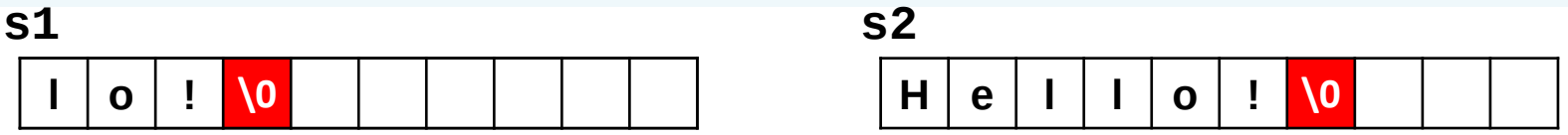
```
char * strncpy (char *, const char *, int);
```

куда

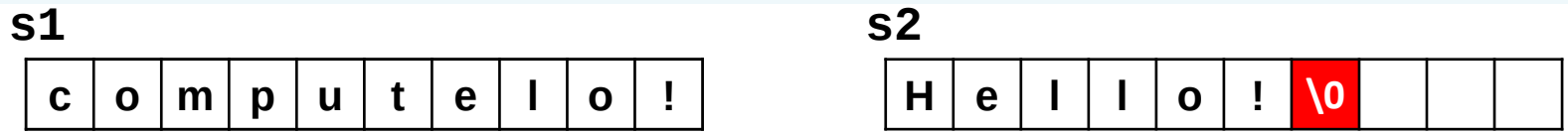
откуда

сколько символов

```
char s1[10] = "Hi!", s2[10] = "Hello!";
strncpy (s1, s2+3, 2);
```




```
char s1[10] = "computer", s2[10] = "Hello!";
strncpy (s1+7, s2+3, 3);
```



потеря нуль-терминатора, последующие обращения к строке приведут к выходу за пределы массива

Конкатенация (сцепление) строк

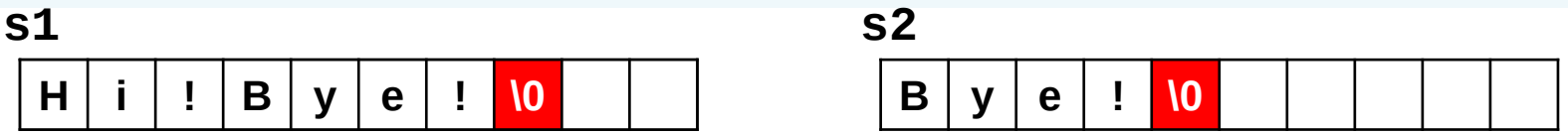
Сцепление строк:

к какой

какую

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

```
char s1[10] = "Hi!", s2[10] = "Bye!";
strcat (s1, s2);
```



Возможные проблемы:

```
strcat (s2, s2);
strcat (s2, "computer");
strcpy (s2+2, s2);
```

зацикливание

выход за пределы массива

результат непредсказуем

! При копировании и сцеплении строк важно следить за длиной строки и помнить, что обработка идет в цикле!

Поиск в строке

Все функции поиска **возвращают адрес** найденного символа или подстроки или ***NULL***, если образец не найден!

Поиск **символа**:

- первое вхождение:

```
char * strchr (const char *, char);
```

где искать

что искать

- последнее вхождение:

```
char * strrchr (const char *, char);
```

Поиск **любого символа из подмножества**:

```
char * strpbrk (const char *, const char *);
```

подмножество

Поиск **подстроки**:

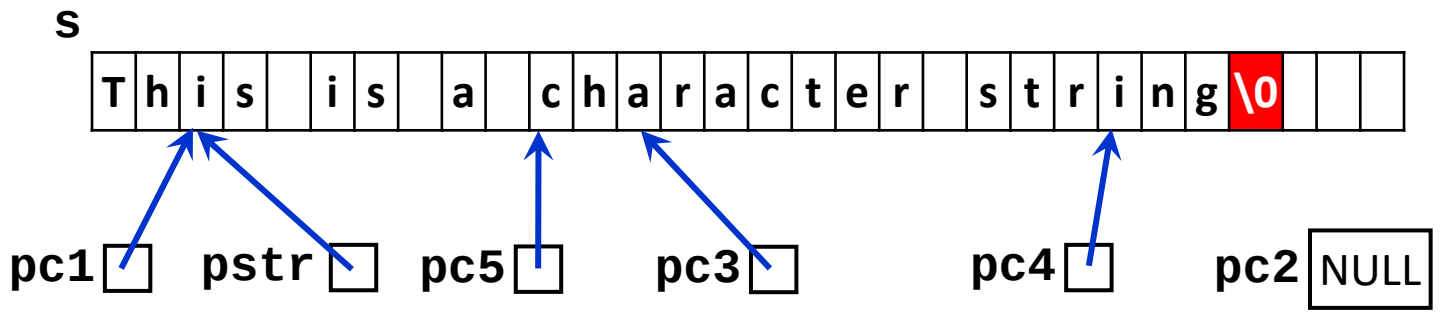
```
char * strstr (const char *, const char *);
```

где искать

что искать

Поиск в строке

```
char s[80] = "This is a character string";
char *pc1, *pc2, *pc3, *pc4, *pc5, *pstr;
```




```
pc1 = strchr (s, 'i');
pc2 = strchr (s, 'x');
pc3 = strchr (s+10, 'a');
pc4 = strrchr (s, 'i');
pc5 = strpbrk (s, "computer");
pstr = strstr (s, "is");
```

подмножество

искомая подстрока

Выделение лексем из строки

Выделение лексем:

```
char * strtok (char *, const char *);
```

где искать: при первом вызове – адрес начала строки, при последующих – NULL

множество разделителей

Особенности:

- **лексемы** могут отделяться друг от друга любыми символами из множества разделителей;
- функция **сохраняет** свое **состояние** между вызовами: при первом вызове надо передать адрес анализируемой строки, при последующих, если передается ***NULL***, функция продолжит работу с байта, следующего за тем, на котором она остановилась в предыдущем вызове;
- функция **изменяет входную строку**, заменяя разделители ‘\0’;
- **возвращает адрес** очередной **лексемы** или ***NULL***, если далее никакой лексемы не обнаружено.

? Как функция сохраняет состояние?

Обработка строк

Задача: Дана строка, слова в которой отделяются друг от друга пробелами и запятыми. Сформировать новую строку из слов заданной, в которых присутствует буква ‘a’ и при этом отсутствует буква ‘e’.

```
#include <stdio.h>
#include <string.h>
int main()
{
    char s1[80], s2[80] = "";
    char *lexeme;
    fgets (s1, 80, stdin);
    lexeme = strtok(s1, " ,");
    while ( lexeme != NULL )
    {
        if ( strchr (lexeme, 'a') != NULL &&
             strchr (lexeme, 'e') == NULL )
        {
            strcat (s2, lexeme);
            strcat (s2, " ");
        }
        lexeme = strtok(NULL, " ,");
    }
    puts(s2);
    return 0;
}
```

пустая строка

безопасный ввод

разделители – пробел или запятая

пока есть лексемы

если в слове есть ‘a’ и нет ’e’

добавляем слово в конец новой строки

ищем следующую лексему в той же строке

Функции преобразования типов

Две группы функций:

- преобразование **строки в число**
  - в вещественное

```
double strtod (const char *, char **);
double atof (const char *);
```

строка; адрес «хвоста» строки; то же

  - в целое

```
long strtol (const char *, char **, int);
int atoi (const char *);
```

строка; адрес «хвоста» строки; основание СС; в десятичное

  - универсальная

```
int sscanf (char *, const char *, ...);
```

строка; формат; адреса

- преобразования **в строку** универсальная

```
int sprintf (char *, const char *, ...);
```

итоговая строка; формат; значения