КОНСТРУИРОВАНИЕ ПРОГРАММ

Лекция № 11 — Интернационализация, широкие символы, строковые функции

Преподаватель: Поденок Леонид Петрович, 505a-5

+375 17 293 8039 (505a-5)

+375 17 320 7402 (ОИПИ НАНБ)

prep@lsi.bas-net.by

ftp://student:2ok*uK2@Rwox@lsi.bas-net.by/

Кафедра ЭВМ, 2021

2021.10.27

Оглавление

Поддержка нескольких языков — <locale.h>....3
setlocale() — задаёт текущую локаль....7
localeconv() — получить информацию о форматировании числовых данных....12
Библиотека. Базовые утилиты....15
Функции поиска и сортировки....15
Функция qsort....16
Функция bsearch....17
Требования к compar....18
Функции целочисленной арифметики....22
Функции abs, labs и llabs....22
Функции div, ldiv и lldiv....23
Библиотека. Широкие символы....25
Многобайтовые (мультибайтовые) символы....29
mblen — получить длину многобайтового символа....29
mbtowc — преобразование многобайтовой последовательности в широкий символ....33
wctomb — преобразование широкого символа в многобайтовую последовательность....37
Многобайтовые (мультибайтовые) строки....40
mbstowcs — преобразование многобайтовой строки в строку расширенных символов....41
wcstombs — преобразование строки расширенных символов в многобайтовую строку....43
Обработка строк <string.h>....47
Соглашения о строковых функциях....47
Функции копирования....48
Функция memcpy()....48
Функция memmove()....49
Функция strcpy()....50
Функция strncpy()....51
Функции конкатенации....52
Функция strcat()....52
Функция strncat()....53
Функции сравнения....54
Функция memcmp()....55
Функция strcmp()....56
Функция strcoll()....57
Функция strncmp()....58
Функция strxfrm()....59
Функции поиска....61
Функция memchr()....62
Функция strchr()....63
Функция strrchr()....64
Функция strcspn()....65
Функция strspn()....66
Функция strpbrk()....67
Функция strstr() — поиск подстроки....68
Функция strtok()....69
Разные функции....71
Функция memset()....71
Функция strerror()....72
Функция strlen()....73

Поддержка нескольких языков — <locale.h>

```
#include <locale.h>
```

Локаль — это сочетание языковых и культурных аспектов. Они включают в себя:
- язык сообщений;
- различные наборы символов;
- лексикографические соглашения;
- форматирование чисел;
- и прочее.

Программа должна определять локализацию и действовать согласно её установкам в целях достижения взаимосвязи различных культур.

Файл **<locale.h>** описывает типы данных, функции и макросы, необходимые для выполнения этой задачи.

В нём описаны функции: **setlocale(3)**, устанавливающая текущую локаль и **localeconv(3)**, которая возвращает информацию о форматировании чисел.

В локали существуют различные категории информации, которые программа может использовать — они описаны как макросы.

Используя их в качестве первого аргумента функции **setlocale(3)**, можно установить значение категории в желаемой локали.

```
LC_ALL
LC_COLLATE
LC_CTYPE
LC_MESSAGES
LC_MONETARY
LC_NUMERIC
LC_TIME
```

LC_COLLATE — эта категория определяет правила сравнения, используемые при сортировке и регулярных выражениях, включая равенство классов символов и сравнение многосимвольных элементов.

Эта категория локали изменяет поведение функций **strcoll**[1]**(3)** и **strxfrm**[2]**(3)**, которые используются для сравнения строк с учётом местного алфавита.

Например, немецкая **ßß** эсцет (sharp s) рассматривается как «ss».

```
strcoll("Scheiße", "Scheibe"); //
```

1 strcoll() — сравнивает две строки с учетом настроек текущей локали

2 strxfrm() — преобразование строки с учетом локали (strcmp() \equiv strcoll())

LC_CTYPE — эта категория определяет интерпретацию последовательности байт в символы (например, одиночный или многобайтовый символ), классификацию символов (например, буквенный или цифровой) и поведение классов символов.

В системах с **glibc** эта категория также определяет правила транслитерации символов для **iconv(1)**[3] и **iconv(3)**. Она изменяет поведение функций обработки и классификации символов, таких как **isupper(3)** и **toupper(3)**, а также многобайтных символьных функций, таких как **mblen**[4]**(3)** или **wctomb**[5]**(3)**.

LC_MONETARY — эта категория определяет форматирования, используемое для денежных значений. Она изменяет информацию, возвращаемую функцией **localeconv(3)**, которая описывает способ отображения числа, например, необходимо ли использовать в качестве десятичного разделителя точку или запятую. Эта информация используется в функции **strfmon**[6]**(3)**.

LC_MESSAGES — эта категория изменяет язык отображаемых сообщений и указывает, как должны выглядеть положительный и отрицательный ответы.

LC_NUMERIC — эта категория определяет правила форматирования, используемые для не денежных значений, например, разделительный символ тысяч и дробной части (точка в англоязычных странах, и запятая во многих других).

Она влияет на поведение функций **printf(3)**, **scanf(3)** и **strtod(3)**.

Эта информация может быть также прочитана при помощи функции **localeconv(3)**.

---

3 Преобразует текст из одной кодировки в другую

4 mblen() — определяет количество байтов в последующем многобайтовом символе

5 wctomb() — преобразует широкий символ в многобайтовую последовательность

6 strfmon() — преобразует денежное значение в строку

LC_TIME — эта категория управляет форматированием значений даты и времени. Например, большая часть Европы использует 24-часовой формат, тогда как в США используют 12-часовой.Значение этой категории влияет на поведение функции **strftime**[7]**(3)** и **strptime**[8]**(3)**.

LC_ALL — всё вышеперечисленное.

Список поддерживаемых категорий можно получить с помощью утилиты оболочки **locale**

```
$ locale
LANG=ru_RU.utf8
LC_CTYPE="ru_RU.utf8"
LC_NUMERIC="ru_RU.utf8"
LC_TIME="ru_RU.utf8"
LC_COLLATE="ru_RU.utf8"
LC_MONETARY="ru_RU.utf8"
LC_MESSAGES="ru_RU.utf8"
LC_PAPER="ru_RU.utf8"
LC_NAME="ru_RU.utf8"
LC_ADDRESS="ru_RU.utf8"
LC_TELEPHONE="ru_RU.utf8"
LC_MEASUREMENT="ru_RU.utf8"
LC_IDENTIFICATION="ru_RU.utf8"
LC_ALL=
```

7 strftime() — форматирование даты и времени

8 strptime() — конвертирует строчное представление времени в представление структуры времени tm

setlocale() — задаёт текущую локаль

```
#include <locale.h>
char *setlocale(int category,
                const char *locale);
```

Функция **setlocale()** используется для назначения или запроса текущей локали для программы.

Если **locale** не равно **NULL**, то текущая локаль программы изменяется согласно переданным аргументам.

Аргументом **category** определяется какую часть текущей локали программы нужно изменить.

LC_ALL — локаль целиком
LC_COLLATE — сортировку строк
LC_CTYPE — классы символов
LC_MESSAGES — локализированные сообщения на родном языке
LC_MONETARY — форматирование значений денежных единиц
LC_NUMERIC — форматирование не денежных числовых значений
LC_TIME — форматирование значений дат и времени

Если второй аргумент функции **setlocale(3)** равен пустой строке **""**, то локаль по умолчанию будет определяться следующим образом:

1. Если существует непустая переменная окружения **LC_ALL**, то используется её значение.
2. Если существует переменная окружения с именем одной из вышеописанных категорий и она непустая, то её значение присваивается этой категории.
3. Если существует непустая переменная окружения **LANG**, то используется её значение.

Если **locale** равно **NULL**, то только возвращается текущая локаль и ничего не меняется.

Информация о локальном форматировании чисел доступна в структуре **struct lconv**, возвращаемой функцией **localeconv(3).**

Эта структура содержит элементы, связанные с форматированием числовых значений.

Аргумент **locale** — это указатель на строку символов, содержащую требуемую настройку для **category**.

Эта строка может быть константой «**C**» или «**ru_RU**», или строкой со скрытым форматом, которую возвращает другой вызов **setlocale()**.

Если **locale** — пустая строка **""**, то любая часть локали, которую требуется изменить, будет задана исходя из переменных окружения.

Как это происходит — зависит от реализации.

В glibc, во-первых (независимо от **category**), просматривается переменная окружения **LC_ALL**, затем переменная окружения с именем как у категории (смотрите таблицу выше), и в конце учитывается переменная окружения **LANG**.

Используется первая найденная переменная окружения.

Если её значение некорректно определяет локаль, то локаль не изменяется и **setlocale()** возвращает NULL.

Локали **"C"** или **"POSIX"** являются переносимыми локалями — они существуют во всех соответствующих системах.

В **setlocale(3)** может использоваться переменная окружения **LOCPATH** и если она установлена, влияет на все непривилегированные локализованные программы. Эта переменная представляет список путей, разделённых двоеточием («**:**»), который должен использоваться при поиске данных, ассоциированных с локалью.

Скомпилированные файлы данных локали по умолчанию ищутся в подкаталогах, которые зависят от используемой в данный момент локали. Обычный путь по умолчанию для расположения архива локалей и скомпилированных файлов отдельных локалей — **/usr/lib/locale/**

```
$ ls -1 usr/lib/locale
C.utf8
...
en_US
en_US.iso885915
en_US.utf8
...
locale-archive
locale-archive.real
ru_RU
ru_RU.koi8r
ru_RU.utf8
ru_UA
ru_UA.utf8
```

Например, если для категории используется **ru_RU.utf8**, то будут просмотрены следующие подкаталоги в порядке **ru_RU.utf8**, **ru_RU**

Имя локали обычно имеет вид:

```
language[_terrirory][chartable][@modifier]
```

language (язык) — код языка согласно ISO 639;
terrirory (территория) — код страны согласно ISO 3166;
chartable (таблица символов) — набор символов или кодировка типа ISO-8859-1 или UTF-8.

Список поддерживаемых локалей можно получить по команде **locale -a**.

```
$ locale -a
C
C.utf8
...
en_US
...
POSIX
ru_RU
ru_RU.koi8r
ru_RU.utf8
ru_UA
ru_UA.utf8
```

При запуске **main()** по умолчанию выбирается переносимая локаль **"C"**.

Программу можно сделать переносимой для всех установленных в системе локалей, вызвав в начале программы

```
setlocale(LC_ALL, "");
```

При успешном выполнении **setlocale()** возвращает строку со скрытым форматом, которая соответствует набору локали.

Эта строка может находиться в статической памяти.

Строка возвращается таким образом, что последующий вызов с этой строкой и связанной с ней категорией, восстанавливают эту часть локали процесса. Если вызов не может быть выполнен, то возвращается значение **NULL**.

localeconv() — получить информацию о форматировании числовых данных

```
#include <locale.h>
struct lconv *localeconv(void);
```

Функция **localeconv()** возвращает указатель на структуру **struct lconv** для текущей локали.

Эта структура определена в файле заголовков **locale.h (locale 7)** и содержит все значения, связанные с категориями локали **LC_NUMERIC** и **LC_MONETARY**.

Программы также могут использовать функции **printf(3)** и **strfmon(3)**, поведение которых зависит от того, какая локаль сейчас используется.

Функция **localeconv()** возвращает указатель на структуру **struct lconv**.

Под эту структуру может (в glibc так и есть) быть статически выделена память, и она может быть перезаписана следующими вызовами.

Согласно POSIX, вызывающий не должен изменять содержимое структуры.

Функция **localeconv()** всегда завершается без ошибок.

Члены структуры с типом **char *** являются указателями на строки, любая из которых (кроме **decimal_point**) может указывать на **""** — это означает, что значение недоступно в текущей локали или имеет нулевую длину.

Помимо **grouping** и **mon_grouping**, строки должны начинаться и заканчиваться в начальном состоянии сдвига.

Элементы с типом **char** являются неотрицательными числами, любое из которых может быть **CHAR_MAX** — это означает, что значение недоступно в текущей локали.

```
$ cat /usr/include/locale.h
#ifndef  _LOCALE_H
#define  _LOCALE_H 1

#include <features.h>

#define __need_NULL
#include <stddef.h>
#include <bits/locale.h>

__BEGIN_DECLS

/* These are the possibilities for the first argument to setlocale.
   The code assumes that the lowest LC_* symbol has the value zero.  */
#define LC_CTYPE          __LC_CTYPE
#define LC_NUMERIC        __LC_NUMERIC
#define LC_TIME           __LC_TIME
#define LC_COLLATE        __LC_COLLATE
#define LC_MONETARY       __LC_MONETARY
#define LC_MESSAGES       __LC_MESSAGES
#define LC_ALL            __LC_ALL
#define LC_PAPER          __LC_PAPER
#define LC_NAME           __LC_NAME
#define LC_ADDRESS        __LC_ADDRESS
#define LC_TELEPHONE      __LC_TELEPHONE
#define LC_MEASUREMENT    __LC_MEASUREMENT
#define LC_IDENTIFICATION __LC_IDENTIFICATION
```

```
/* Structure giving information about numeric and monetary notation.  */
struct lconv { /* Numeric (non-monetary) information.  */
  char *decimal_point;    /* Decimal point character.  */
  char *thousands_sep;    /* Thousands separator.  */
...

  /* Monetary information.  */
  /* First three chars are a currency symbol from ISO 4217. */
  /* Fourth char is the separator.  Fifth char is '\0'.  */
  char *int_curr_symbol;
  char *currency_symbol;  /* Local currency symbol.  */
  char *mon_decimal_point;   /* Decimal point character.  */
  char *mon_thousands_sep;   /* Thousands separator.  */
  char *mon_grouping;     /* Like `grouping' element (above).  */
  char *positive_sign;    /* Sign for positive values.  */
  char *negative_sign;    /* Sign for negative values.  */
  ...
};

/* Set and/or return the current locale.  */
extern char *setlocale (int __category, const char *__locale) __THROW;
/* Return the numeric/monetary information for the current locale.  */
extern struct lconv *localeconv (void) __THROW;
...
__END_DECLS
#endif /* locale.h  */
```

Библиотека. Базовые утилиты

Функции поиска и сортировки

qsort() — сортировка элементов массива.
bsearch() — двоичный поиск в массиве, отсортированном по ключу;

```
#include <stdlib.h>

void    qsort(void *base,       // начальный элемент массива
              size_t nmemb,     // количество объектов (элементов массива)
              size_t size,      // размер объекта (элемента массива)
              int (*compar)(const void *, const void *));

void *bsearch(const void *key,  // ключ поиска
              const void *base, // начальный элемент массива
              size_t nmemb,     // количество объектов (элементов массива)
              size_t size,      // размер объекта (элемента массива)
              int (*compar)(const void *, const void *));
```

Функция qsort

```
#include <stdlib.h>

void qsort(void *base,   // начальный элемент массива
           size_t nmemb, // количество элементов массива
           size_t size,  // размер объекта (размер элемента массива)
           int (*compar)(const void *, const void *));
```

Функция **qsort** сортирует массив **nmemb** объектов, на начальный элемент которых указывает **base**. Размер каждого объекта определяется **size**.

Содержимое массива сортируется в порядке возрастания в соответствии с функцией сравнения, на которую указывает **compar**, и которая вызывается с двумя аргументами, указывающими на сравниваемые объекты.

Функция должна возвращать целое число меньше, равно или больше нуля, если первый аргумент считается соответственно меньше, равен или больше второго.

Если два элемента сравниваются как равные, их порядок в результирующем отсортированном массиве не определен.

Функция **qsort** не возвращает значения.

Функция bsearch

```
#include <stdlib.h>

void *bsearch(const void *key,  // ключ поиска
              const void *base, // начальный элемент массива
              size_t nmemb,     // количество объектов (размер массива)
              size_t size,      // размер элемента
              int (*compar)(const void *, const void *));
```

Функция **bsearch** ищет в массиве из **nmemb** объектов, на начальный элемент которых указывает **base**, элемент, соответствующий объекту, на который указывает **key**. Размер каждого элемента массива определяется **size**.

Функция сравнения, на которую указывает **compar**, вызывается с двумя аргументами, которые указывают на объект **key** и на элемент массива.

Функция должна возвращать целое число меньше, равно или больше нуля, если ключевой объект считается, соответственно, меньше, соответствует или больше, чем элемент массива.

Массив должен быть упорядочен в ключе. На практике перед вызовом **bsearch** весь массив сортируется в соответствии с функцией сравнения.

Функция **bsearch** возвращает указатель на соответствующий элемент массива или, если совпадений не найдено, нулевой указатель.

Требования к compar

compar — указатель на функцию, которая сравнивает два элемента. Эта функция многократно вызывается функцией **bsearch** для сравнения **key** с отдельными элементами в **base**, а также функцией **qsort** для сравнения элементов масива.

Функция **compar** должен следовать следующему прототипу:

```
int compar(const void* p1, const void* p2);
```

Принимает два указателя в качестве аргументов.

Для функции **bsearch** первый всегда является указателем на ключ, а второй указывает на элемент массива (оба преобразуются в **const void ***).

Для функции **qsort**, оба указывают на элементы массива (преобразуются в **const void ***).

Функция должна возвращать (устойчивым и транзитивным образом) следующие значения:

<0 Элемент, на который указывает **pkey**, идет перед элементом, на который указывает **pelem**;

0 Элемент, на который указывает **pkey**, эквивалентен элементу, на который указывает **pelem**;

>0 Элемент, на который указывает **pkey**, идет после элемента, на который указывает **pelem**.

Для типов, которые можно сравнивать с помощью обычных реляционных операторов, общая функция сравнения может выглядеть следующим образом:

```
int compare_T(const void *a, const void *b) {
    if (*(T*)a <  *(T*)b ) return -1;
    if (*(T*)a == *(T*)b ) return 0;
    if (*(T*)a >  *(T*)b ) return 1;
}
```

Пример

```
/* bsearch example */
#include <stdio.h>      /* printf */
#include <stdlib.h>     /* qsort, bsearch, NULL */

int compare_ints(const void *a, const void *b) {
  return (*(int *)a - *(int *)b);
}

int array[] = { 50, 20, 60, 40, 10, 30 };

int main () {

    int *pItem;
    int  key = 40;

    qsort(array, 6, sizeof (int), compare_ints);

    pItem = (int*)bsearch(&key, array, 6, sizeof(int), compare_ints);

    if (pItem != NULL)
        printf ("%d присутствует в массиве.\n", *pItem);
    else
        printf ("%d отсутствует в массиве.\n", key);

    return 0;
}
```

В этом примере **compare_ints** сравнивает значения, на которые указывают два параметра, как значения типа **int** и возвращает результат их вычитания, результат которого равен 0, если они равны, положителен, если значение, на которое указывает **a**, больше, чем значение, на который указывает **b**, или отрицателен, если значение, на которое указывает **b**, больше, чем **a**.

В основной функции целевой массив сортируется с помощью **qsort** перед вызовом **bsearch** для поиска значения.

Вывод:

40 присутствует в массиве.

Для C-строк в качестве аргумента сравнения для **bsearch** может напрямую использоваться **strcmp()**:

```
/* bsearch example with strings */
#include <stdio.h>      /* printf */
#include <stdlib.h>     /* qsort, bsearch, NULL */
#include <string.h>     /* strcmp */

char cstring_array[][20] = {"some","example","strings","here"};

int main () {

   char *pItem;
   char  key[20] = "example";

   // сортировка
   qsort(cstring_array, 4, 20, (int(*)(const void*,const void*))strcmp);

   // поиск
   pItem = (char*)bsearch(key, cstring_array, 4, 20,
                          (int(*)(const void*,const void*))strcmp);
   if (pItem!=NULL)
      printf ("%s is in the array.\n",pItem);
   else
      printf ("%s is not in the array.\n",key);

   return 0;
}
```

Функции целочисленной арифметики

Функции abs, labs и llabs

```
#include <stdlib.h>

int abs(int j);
long int labs(long int j);
long long int llabs(long long int j);
```

Функции **abs**, **labs** и **llabs** вычисляют абсолютное значение целого числа **j**.
Если результат не может быть представлен в типе, поведение не определено.

Возврат
Функции **abs**, **labs** и **llabs** возвращают абсолютное значение аргумента.

Функции div, ldiv и lldiv

```
#include <stdlib.h>

div_t div(int numer, int denom);
ldiv_t ldiv(long int numer, long int denom);
lldiv_t lldiv(long long int numer, long long int denom);
```

Функции **div**, **ldiv** и **lldiv** вычисляют результат целочисленного деления делимого **numer** на делитель **denom** и остаток от этого деления за одно действие.

Возврат

Функции **div**, **ldiv** и **lldiv** возвращают структуру типа **div_t**, **ldiv_t** и **lldiv_t**, соответственно, содержащую как частное, так и остаток.

Структуры содержат (в любом порядке) члены **quot** (частное) и **rem** (остаток), каждый из которых имеет тот же тип, что и аргументы **numer** и **denom**.

Если какая-либо часть результата не может быть представлена в типе, поведение не определено.

```
typedef struct { // C98, C++11
    long long int quot;         /* Quotient.  */
    long long int rem;          /* Remainder.  */
} lldiv_t;
```

Пример

```
/*
 * lldiv example
 */
#include <stdio.h>      // printf
#include <stdlib.h>     // lldiv, lldiv_t

int main () {

  lldiv_t res = lldiv(31558149LL, 3600LL);

  printf("Оборот Земли по орбите: %lld часов and %lld секунд.\n",
         res.quot,
         res.rem);
  return 0;
}
```

Вывод

```
Оборот Земли по орбите: 8766 часов and 549 секунд.
```

Библиотека. Широкие символы

<cwchar> (wchar.h) — этот заголовочный файл определяет несколько функций для работы со строками широких символов и вводит несколько типов.

Типы

mbstate_t — состояние многобайтового преобразования
size_t — беззнаковый целочисленный тип
struct tm — структура времени
wchar_t — **широкий символ**
wint_t — широкий целочисленный тип

`mbstate_t` — состояние многобайтового преобразования

Этот тип содержит информацию, необходимую для сохранения состояния при преобразованиях между последовательностями многобайтовых символов и широких символов (в любом направлении).

Кодировка многобайтовой последовательности может иметь разные состояния сдвига (shift state) вдоль анализируемой последовательности байт, которые меняют способ интерпретации следующего байта. Значения типа **`mbstate_t`** могут сохранять эти состояния между вызовами функций, так что перевод последовательности может безопасно выполняться при осуществлении нескольких вызовов.

Все допустимые многобайтовые последовательности должны начинаться (и заканчиваться) в одном и том же состоянии (называемом его начальным состоянием).

Например, UTF8 — если для кодирования требуется больше одного октета, то в октетах 2-4 два старших бита всегда устанавливаются равными 10 (10xxxxxx). Это позволяет легко отличать первый октет в потоке, потому что его старшие биты никогда не равны 10.

Количество октетов	Значащих бит	Шаблон
1	7	0xxxxxxx
2	11	110xxxxx 10xxxxxx
3	16	1110xxxx 10xxxxxx 10xxxxxx
4	21	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

Алгоритм преобразования представляет собой *конечный автомат* (FCM — Finite State Machine), который анализирует байт за байтом в строке и переходит из одного состояния в другое. «Собрав» символ, он его преобразует во внутреннее представление (широкий символ). Объект типа **`mbstate_t`** содержит состояние этого конечного автомата.

Объект типа **mbstate_t** с нулевым значением всегда описывает начальное состояние преобразования, хотя другие значения также могут представлять такое состояние (в зависимости от конкретной реализации библиотеки).

Объект типа **mbstate_t (mbs)** можно установить в исходное состояние, вызвав:

```
memset(&mbs, 0, sizeof(mbs));  // mbs is now a zero-valued object
```

Для проверки конкретного состояния два значения **mbstate_t** не должны сравниваться друг с другом — статус начального состояния объекта с типом **mbstate_t** можно проверить с помощью функции **mbsinit()**.

`wchar_t` — тип широкого символа

Тип, диапазон значений которого может представлять различные коды для всех членов самого большого расширенного набора символов, указанного среди поддерживаемых локалей.

В C++ **`wchar_t`** — это отдельный фундаментальный тип (и поэтому он не определен ни в **`<cwchar>`**, ни в каком-либо другом заголовке).

В C это **`typedef`** целочисленного типа достаточного размера.

`wint_t` — широкий целочисленный тип

Это **`typedef`** типа, способного представить любое значение типа **`wchar_t`**, которое является членом расширенного набора символов, а также дополнительное значение **`WEOF`** (не является частью этого набора).

Этот тип может быть либо псевдонимом **`wchar_t`**, либо псевдонимом целочисленного типа, в зависимости от конкретной реализации библиотеки. Но он должен оставаться неизменным при продвижении аргументов по умолчанию из значений типа **`wchar_t`**.

Многобайтовые (мультибайтовые) символы

mblen — Получить длину многобайтового символа (функция)
mbtowc — Преобразование многобайтовой последовательности в широкий символ (функция)

mblen — получить длину многобайтового символа

```
#include <stdlib.h>
int mblen(const char *pmb, size_t max);
```

Вычисляет количество байт в многобайтовом символе, на который указывает **pmb**, просматривая не более **max** байт.

mblen имеет собственное внутреннее состояние сдвига, которое при необходимости изменяется только при вызове этой функции.

Поведение этой функции зависит от категории **LC_CTYPE** выбранной локали.

pmb — указатель на первый байт многобайтового символа.

В качестве альтернативы функция может быть вызвана с нулевым указателем, и в этом случае функция сбрасывает свое внутреннее состояние сдвига на начальное значение и возвращает информацию, используют ли многобайтовые символы кодировку, зависящую от состояния.

max — максимальное количество байтов **pmb**, которое следует учитывать для многобайтового символа.

В любом случае проверяется не более чем **MB_CUR_MAX** символов.

size_t — это целочисленный тип без знака.

Возвращаемое значение

Если аргумент, переданный как **pmb**, не является нулевым указателем, возвращается размер символа, на который указывает **pmb**, в байтах, если он образует допустимый многобайтовый символ, но не завершающий нулевой символ.

Если это завершающий нулевой символ, функция возвращает ноль, а если байты **pmb** не образуют допустимый многобайтовый символ, возвращается -1.

Если аргумент, переданный в качестве **pmb**, является нулевым указателем, функция возвращает ненулевое значение, если кодировки многобайтовых символов зависят от состояния, и ноль в противном случае.

Пример

В примере используется строка с использованием локали "C".

```
#include <stdio.h>      // printf
#include <stdlib.h>     // mblen, mbtowc, wchar_t(C)

// ------------------------------------------
// печатает многобайтовую строку посимвольно.
// ------------------------------------------
void printbuffer(const char *pt, size_t max) {

  int     length;
  wchar_t dest;

  mblen(NULL, 0);         // reset mblen
  mbtowc(NULL, NULL, 0);  // reset mbtowc

  while(max > 0) {
      length = mblen(pt, max);
      if (length < 1) break;     // мусор в строке — это не символ, выходим
      mbtowc(&dest, pt, length);
      printf("[%lc]", dest);
      pt += length; max -= length;
  }
}
```

```
int main( ) {

  const char str[] = "test string";

  printbuffer(str, sizeof(str));
  return 0;
}
```

Вывод:

```
[t][e][s][t][ ][s][t][r][i][n][g]
```

Гонки данных

Функция обращается к массиву, указанному **pmb**.

Функция также обращается к объекту внутреннего состояния и изменяет его, что может вызвать гонки данных при одновременных вызовах этой функции.

Есть альтернатива, которая может использовать объект внешнего состояния. Это функция **mbrlen**.

```
#include <wchar.h>

size_t mbrlen(const char *s, size_t n, mbstate_t *ps);
```

Одновременное изменение настроек локали также может привести к гонке данных.

Если **pmb** не является ни пустым указателем, ни указателем на достаточно длинный массив, это вызовет неопределенное поведение.

mbtowc — преобразование многобайтовой последовательности в широкий символ

```
int mbtowc(wchar_t *pwc,    // широкий результат
           const char *pmb, // многобайтовый источник
           size_t max);     // максимальное кол-во байт в многобайтовом
символе
```

Многобайтовый символ, на который указывает **pmb**, преобразуется в значение типа **wchar_t** и сохраняется в месте, указанном **pwc**.

Функция возвращает длину многобайтового символа в байтах.

mbtowc имеет собственное внутреннее состояние сдвига, которое изменяется по мере необходимости только при вызове этой функции.

Вызов функции с нулевым указателем в качестве **pmb** сбрасывает состояние (и возвращает, зависят ли многобайтовые символы от состояния).

Поведение этой функции зависит от категории **LC_CTYPE** выбранной С-локали.

Параметры

pwc — указатель на объект типа **wchar_t**.

В качестве альтернативы, этот аргумент может быть нулевым указателем, и в этом случае функция не сохраняет перевод **wchar_t**, но по-прежнему возвращает длину в байтах многобайтового символа.

pmb — указатель на первый байт многобайтового символа.

В качестве альтернативы, этот аргумент может быть нулевым указателем, и в этом случае функция сбрасывает свое внутреннее состояние сдвига до начального значения и возвращает, имеют ли многобайтовые символы кодировку, зависящую от состояния.

max — максимальное количество байтов **pmb**, которое следует учитывать для многобайтового символа.

В любом случае проверяется не более чем **MB_CUR_MAX** символов.

Возвращаемое значение

Если аргумент, переданный как **pmb**, не является нулевым указателем, возвращается размер символа, на который указывает **pmb**, в байтах, если он образует допустимый многобайтовый символ, но не завершающий нулевой символ.

Если это завершающий нулевой символ, функция возвращает ноль, а если байты **pmb** не образуют допустимый многобайтовый символ, возвращается -1.

Если аргумент, переданный в качестве **pmb**, является нулевым указателем, функция возвращает ненулевое значение, если кодировки многобайтовых символов зависят от состояния, и ноль в противном случае.

Пример

В примере используется строка с использованием локали "C".

```
// mbtowc example
#include <stdio.h>      // printf */
#include <stdlib.h>     // mbtowc, wchar_t(C)

// печатает многобайтовую строку посимвольно.
void printbuffer (const char* pt, size_t max) {

  int length;
  wchar_t dest;

  mbtowc (NULL, NULL, 0);  // reset mbtowc

  while (max>0) {
    length = mbtowc(&dest, pt, max);
    if (length<1) break;
    printf("[%lc]", dest);
    pt += length;
    max -= length;
  }
}
```

```
int main( ) {

  const char str[] = "mbtowc example";

  printbuffer(str, sizeof(str));

  return 0;
}
```

Вывод

```
[m][b][t][o][w][c][ ][e][x][a][m][p][l][e]
```

Гонки данных

Функция обращается к массиву, указанному **pmb**, и изменяет объект, указанный **pwc** (если не **null**).

Функция также обращается к объекту внутреннего состояния и изменяет его, что может вызвать гонки данных при одновременных вызовах этой функции. Есть альтернатива, которая может использовать объект внешнего состояния. Это функция **mbrtowc**.

```
#include <wchar.h>

size_t mbrtowc(wchar_t *pwc, const char *s, size_t n, mbstate_t *ps);
```

Одновременное изменение настроек локали также может привести к гонке данных.

Если **pmb** не является ни пустым указателем, ни указателем на достаточно длинный массив, это вызовет неопределенное поведение.

wctomb — преобразование широкого символа в многобайтовую последовательность

```
int wctomb(char* pmb,    // многобайтовый результат
           wchar_t wc);  // широкий источник
```

В основном, функция предназначена для работы, когда **pmb** не равно **NULL** и **wc** не равно широкому символу **null** (**L'\0'**). В этом случае широкий символ **wc** преобразуется в его многобайтовый эквивалент и сохраняется в массиве, на который указывает **pmb**.

Функция возвращает длину в байтах эквивалентной многобайтовой последовательности, указанной **pmb** после вызова.

wctomb имеет собственное внутреннее состояние сдвига, которое изменяется по мере необходимости только вызовами этой функции. Вызов функции с нулевым указателем в качестве **pmb** сбрасывает состояние (и возвращает, зависят ли многобайтовые последовательности от состояния).

Поведение этой функции зависит от категории **LC_CTYPE** выбранной C-локали.

Гонки данных

Функция изменяет массив, на который указывает **pmb**.

Функция также обращается к объекту внутреннего состояния и изменяет его, что может вызвать гонки данных при одновременных вызовах этой функции. Есть функция **wcrtomb** в качестве альтернативы, которая может использовать объект внешнего состояния.

```
#include <wchar.h>

size_t wcrtomb(char *s, wchar_t wc, mbstate_t *ps);
```

Одновременное изменение настроек локали также может привести к гонке данных.

Параметры

pmb — указатель на массив, достаточно большой для хранения многобайтовой последовательности. Максимальная длина многобайтовой последовательности для символа в текущей локали составляет **MB_CUR_MAX** байт.

В качестве альтернативы функция может быть вызвана с нулевым указателем, и в этом случае функция сбрасывает свое внутреннее состояние сдвига на начальное значение и возвращает, используют ли многобайтовые последовательности кодирование, зависящее от состояния.

wc — широкий символ типа **wchar_t**.

Возвращаемое значение

Если аргумент, переданный как **pmb**, не является нулевым указателем, возвращается размер в байтах символа, записанного в **pmb**.

Если символьного соответствия нет, возвращается -1.

Если аргумент, переданный как **pmb**, является нулевым указателем, функция возвращает ненулевое значение, если кодировки многобайтовых символов зависят от состояния, и ноль в противном случае.

Пример

В примере печатаются многобайтовые символы, в которые с использованием выбранной локали преобразуется строка широких символов (в данном случае "C" по умолчанию).

```
#include <stdio.h>      // printf
#include <stdlib.h>     // wctomb, wchar_t(C)

int main() {

  const wchar_t  str[] = L"wctomb example";
  const wchar_t *pt    = str;
  char           buffer[MB_CUR_MAX];
  int            i,length;

  while (*pt) {
    length = wctomb(buffer, *pt++);
    if (length < 1) break;         // Ошибка
    for (i = 0; i < length; ++i) {
      printf ("[%c]",buffer[i]);
    }
  }
  return 0;
}
```

Вывод:

```
[w][c][t][o][m][b][ ][e][x][a][m][p][l][e]
```

Многобайтовые (мультибайтовые) строки

`mbstowcs` — Преобразование многобайтовой строки в строку расширенных символов
`wcstombs` — Преобразование строки расширенных символов в многобайтовую строку

mbstowcs — преобразование многобайтовой строки в строку расширенных символов

```
#include <stdlib.h>
size_t mbstowcs(wchar_t *dest,    // куда
                const char *src,  // откуда
                size_t max);      // максимальное кол-во широких символов
```

Преобразует многобайтовую последовательность, указанную **src**, в эквивалентную последовательность широких символов (которые сохраняются в массиве, указанном **dest**), до тех пор, пока не будут преобразовано **max** широких символов или пока не встретится нулевой символ в многобайтовой последовательности **src** (который также преобразуется и сохраняется, но не учитывается в длине, возвращаемой функцией).

Если **max** символов преобразовано успешно, результирующая строка, хранящаяся в **dest**, не будет завершена нулем.

Поведение этой функции зависит от категории **LC_CTYPE** выбранной C-локали.

Параметры

dest — указатель на массив элементов **wchar_t**, достаточно длинный, чтобы содержать результирующую последовательность (не более, чем **max** широких символов).

src — C-строка с интерпретируемыми многобайтовыми символами.

Многобайтовая последовательность должна начинаться в состоянии начального сдвига.

max — максимальное количество символов **wchar_t** для записи в **dest**.

Возвращаемое значение

Число широких символов, записанных в **dest**, не включая конечный нулевой символ.

Если обнаружен недопустимый многобайтовый символ, возвращается значение **(size_t)-1**.

Следует обратить внимание, что **size_t** - это целочисленный тип без знака, и поэтому ни одно из возможных возвращаемых значений не будет меньше нуля.

Гонки данных

Функция обращается к массиву, указанному **src**, и изменяет массив, указанный **dest**. Функция также может обращаться к объекту внутреннего состояния и изменять его, что может вызывать гонки данных при одновременных вызовах этой функции, если реализация использует статический объект (см. **mbsrtowcs** для альтернативы, которая может использовать объект внешнего состояния).

Одновременное изменение настроек локали также может привести к гонке данных.

wcstombs — преобразование строки расширенных символов в многобайтовую строку

```
#include <stdlib.h>
size_t wcstombs(char *dest,          // куда писать байты
                const wchar_t *src, // откуда рать широкие символы
                size_t max);         // максимальное кол-во байт для
преобразов.
```

Преобразует широкие символы из последовательности, указанной **src**, в многобайтовую эквивалентную последовательность (которая сохраняется в массиве, указанном **dest**), до тех пор, пока не будет преобразовано **max** байтов или пока широкие символы не преобразуются в нулевой символ.

Если **max** байтов успешно переведено, результирующая строка, хранящаяся в **dest**, не будет завершена нулем.

Результирующая многобайтовая последовательность начинается в начальном состоянии сдвига (если есть).

Поведение этой функции зависит от категории **LC_CTYPE** выбранной C-локали.

Параметры

dest — указатель на массив элементов **char**, достаточно длинный, чтобы содержать результирующую последовательность (не более, чем **max** байтов).

src — строка широких символов для преобразования.

max — максимальное количество байтов для записи в **dest**.

Возвращаемое значение

Количество байтов, записанных в **dest**, не включает конечный нулевой символ.

Если встречается широкий символ, не соответствующий допустимому многобайтовому символу, возвращается значение **(size_t)-1**.

Следует обратить внимание, что **size_t** - это целочисленный тип без знака, и поэтому ни одно из возможных возвращаемых значений не будет меньше нуля.

Пример

```
/* wcstombs example */
#include <stdio.h>      /* printf */
#include <stdlib.h>     /* wcstombs, wchar_t(C) */

int main() {
  const wchar_t str[] = L"wcstombs example";
  char buffer[32];

  printf("wchar_t string: %ls \n",str);

  int ret = wcstombs(buffer, str, sizeof(buffer));
  if (ret == 32) {
    buffer[31]='\0';
  }
  if (ret) {
    printf("multibyte string: %s \n",buffer);
  }
  return 0;
}
----------------------------------------
wchar_t string: wcstombs example
multibyte string:  wcstombs example
```

Гонки данных

Функция обращается к массиву, указанному **src**, и изменяет массив, указанный **dest**. Функция также может обращаться к объекту внутреннего состояния и изменять его, что может вызывать гонки данных при одновременных вызовах этой функции, если реализация использует статический объект (см. **wcsrtombs** для альтернативы, которая может использовать объект внешнего состояния).

Одновременное изменение настроек локали также может привести к гонке данных.

Обработка строк <string.h>

Соглашения о строковых функциях

Заголовок **<string.h>** объявляет один тип () и несколько функций, а также определяет один макрос, полезный для управления массивами символьного типа и другими объектами, рассматриваемыми как массивы символьного типа.

Тип — **size_t**, а макрос — **NULL**.

Для определения длины массивов используются различные методы, но во всех случаях аргумент **char *** или **void *** указывает на начальный (наименьший адрес) символ массива.

Если доступ к массиву осуществляется за пределами объекта, поведение не определено.

Если аргумент, объявленный как **size_t n**, определяет длину массива для функции, при вызове этой функции **n** может иметь нулевое значение.

Если иное явно не указано в описании конкретной функции, аргументы типа указателя при таком вызове должны в любом случае иметь допустимые значения.

При таком вызове функция, которая определяет местонахождение символа, не находит вхождения, функция, сравнивающая две последовательности символов, возвращает ноль, а функция, копирующая символы, копирует ноль символов.

Для всех функций в **<string.h>** каждый символ должен интерпретироваться так, как если бы он имел тип **unsigned char** (и поэтому каждое возможное представление объекта является допустимым и имеет отличное от других значение).

Функции копирования

Функция memcpy()

```
#include <string.h>
void *memcpy(void * restrict s1,         // куда
             const void * restrict s2,   // откуда
             size_t n);                  // сколько
```

Функция **memcpy** копирует **n** символов из объекта, на который указывает **s2**, в объект, на который указывает **s1**.

Если копирование происходит между перекрывающимися объектами, поведение не определено.

Возвращает

Функция **memcpy** возвращает значение **s1**.

Функция memmove()

```
#include <string.h>
void *memmove(void *s1,       // куда
              const void *s2, // откуда
              size_t n);      // сколько
```

Функция **memmove** копирует **n** символов из объекта, на который указывает **s2**, в объект, на который указывает **s1**.

Копирование происходит так, как если бы **n** символов из объекта, на который указывает **s2**, сначала копируются во временный массив из **n** символов, который не перекрывает объекты, на которые указывают **s1** и **s2**, а затем **n** символов из временного массива копируются в объект, на который указывает **s1**.

Возвращает

Функция **memmove** возвращает значение **s1**.

Функция strcpy()

```
#include <string.h>
char *strcpy(char * restrict s1,        // куда
             const char * restrict s2); // откуда
```

Функция **strcpy** копирует строку, на которую указывает **s2** (включая завершающий нулевой символ), в массив, на который указывает **s1**.

Если копирование происходит между перекрывающимися объектами, поведение не определено.

Возвращает

Функция **strcpy** возвращает значение **s1**.

Функция strncpy()

```
#include <string.h>
char *strncpy(char * restrict s1,       // куда
              const char * restrict s2, // откуда
              size_t n);                // не более, чем
```

Функция **strncpy** копирует не более **n** символов (символы, следующие за нулевым символом, не копируются) из массива, на который указывает **s2**, в массив, на который указывает[9] **s1**.

Если копирование происходит между перекрывающимися объектами, поведение не определено.

Если массив, на который указывает **s2**, является строкой, которая короче **n** символов, к копии в массиве, на который указывает **s1**, добавляются нулевые символы пока не будет записано **n** символов.

Возвращает

Функция **strncpy()** возвращает значение **s1**.

9 Таким образом, если в первых n символах массива, на который указывает s2, нет нулевого символа, результат не будет завершаться нулевым символом в конце.

Функции конкатенации

Функция strcat()

```
#include <string.h>
char *strcat(char * restrict s1,         //
             const char * restrict s2);  //
```

Функция **strcat** добавляет копию строки, на которую указывает **s2** (включая завершающий нулевой символ), в конец строки, на которую указывает **s1**.

Начальный символ **s2** перезаписывает нулевой символ в конце **s1**.

Если копирование происходит между перекрывающимися объектами, поведение не определено.

Возвращает

Функция **strcat** возвращает значение **s1**.

Функция strncat()

```
#include <string.h>
char *strncat(char * restrict s1,
              const char * restrict s2,
              size_t n);                  // не более, чем
```

Функция **strncat** добавляет не более **n** символов (нулевой символ и следующие за ними символы не добавляются) из массива, на который указывает **s2**, в конец строки, на которую указывает **s1**.

Начальный символ **s2** перезаписывает нулевой символ в конце **s1**.

Завершающий нулевой символ всегда добавляется к результату[10].

Если копирование происходит между перекрывающимися объектами, поведение не определено.

Возвращает

Функция **strncat()** возвращает значение **s1**.

10 Таким образом, максимальное количество символов, которое может оказаться в массиве, на который указывает s1,
равно strlen(s1) + n + 1.

Функции сравнения

memcmp() — сравнить первые n символов объекта
strcmp() — сравнить строки
strncmp() — сравнить n байт в строках

strcoll() — сравнить строки в категории **LC_COLLATE**
strxfrm() — преобразовать строки, учитывая **LC_COLLATE**

Знак ненулевого значения, возвращаемого функциями сравнения **memcmp()**, **strcmp()** и **strncmp()**, определяется знаком разницы между значениями первой пары символов (оба интерпретируются как **unsigned char**), которые различаются в сравниваемых объектах.

Функция memcmp()

```
#include <string.h>
int memcmp(const void *s1, // первый объект
           const void *s2, // второй объект
           size_t n);      // длина сравнения
```

Функция **memcmp( )** сравнивает первые **n** байт объекта, на который указывает **s1**, с первыми[11] **n** байтами объекта, на который указывает **s2**.

Возвращает

Функция **memcmp** возвращает целое число, большее, равное или меньшее нуля в соответствии с тем, больше, равна или меньше строка, на которую указывает **s1**, строки, на которую указывает **s2**.

11 Содержимое «дырок», используемых в качестве заполнения для целей выравнивания внутри структурных объектов, не определено. Строки короче, чем выделенное им пространство, и объединения также могут вызывать проблемы при сравнении.

Функция strcmp()

```
#include <string.h>
int strcmp(const char *s1,  // первая строка
           const char *s2); // вторая строка
```

Функция **strcmp** сравнивает строку, на которую указывает **s1**, со строкой, на которую указывает **s2**.

Возвращает

Функция **strcmp** возвращает целое число, большее, равное или меньшее нуля в соответствии с тем, больше, равна или меньше строка, на которую указывает **s1**, строки, на которую указывает **s2**.

Функция strncmp()

```
#include <string.h>
int strncmp(const char *s1, // первая строка
            const char *s2, // вторая строка
            size_t n);      // не более, чем
```

Функция **strncmp** сравнивает не более **n** символов (символы, следующие за нулевым символом, не сравниваются) из массива, на который указывает **s1**, с массивом, на который указывает **s2**.

Возвращает

Функция **strncmp** возвращает целое число, большее, равное или меньшее нуля, соответственно тому, больше, равен или меньше поскольку массив с возможным завершением нулем, на который указывает **s1**, массива с возможным завершением нулем, на который указывает **s2**.

Функция strcoll()

```
#include <string.h>
int strcoll(const char *s1,  // первая строка
            const char *s2); // вторая строка
```

Функция **strcoll()** сравнивает строку, на которую указывает **s1**, со строкой, на которую указывает **s2**, при этом обе интерпретируются как соответствующие категории **LC_COLLATE** текущей локали.

Возвращает

Функция **strcoll** возвращает целое число, большее, равное или меньшее нуля в соответствии с тем, больше, равна или меньше строка, на которую указывает **s1**, строки, на которую указывает **s2**, когда обе интерпретируются как соответствующие текущей локали.

Функция strxfrm()

```
#include <string.h>
size_t strxfrm(char * restrict s1,       // куда
               const char * restrict s2, // откуда
               size_t n);                // сколько на выходе
```

Функция **strxfrm** преобразует строку, на которую указывает **s2**, и помещает полученную строку в массив, на который указывает **s1**.

Преобразование таково, что если функция **strcmp** применяется к двум преобразованным строкам, она возвращает значение больше, равное или меньше нуля, соответствующее результату функции **strcoll**, примененной к тем же двум исходным строкам.

В результирующий массив, на который указывает **s1**, помещается не более **n** символов, включая завершающий нулевой символ.

Если **n** равно нулю, **s1** может быть нулевым указателем.

Если копирование происходит между перекрывающимися объектами, поведение не определено.

Возвращает

Функция **strxfrm** возвращает длину преобразованной строки (не включая завершающий нулевой символ).

Если возвращаемое значение равно **n** или более, содержимое массива, на которое указывает **s1**, является неопределенным.

Пример

Значение следующего выражения — это размер массива, необходимого для хранения преобразования строки, на которую указывает **s**.

```
1 + strxfrm(NULL, s, 0)
```

Функции поиска

`memchr()` — находит первое вхождение в объекте
`strchr()` — находит первое вхождение символа в строке
`strrchr()` — находит последнее вхождение символа
`strspn()` — вычисляет длину начального сегмента строки, содержащего символы из набора
`strcspn()` — вычисляет длину начального сегмента строки, не содержащего символов из набора
`strpbrk()` — находит в строке первое вхождение из набора
`strstr()` — поиск подстроки
`strtok()` — разбить строку на лексемы

Функция `memchr()`

```
#include <string.h>
void *memchr(const void *s, // где ищем
             int c,         // что ищем
             size_t n);     // размер области поиска в байтах
```

Функция **memchr** находит в начальных **n** символах объекта, на который указывает **s**, первое вхождение **c**, преобразованного в **unsigned char**. Каждый символ объекта, на который указывает **s**, так же интерпретируется как **unsigned char**,

Функция ведет себя так, как если бы символы считывались последовательно, и как только будет найден соответствующий символ, считывание останавливается.

Возвращает

Функция **memchr** возвращает указатель на обнаруженный символ или нулевой указатель, если символ в объекте не встречается.

Функция `strchr()`

```
#include <string.h>
char *strchr(const char *s,  // где ищем
             int c);         // что ищем
```

Функция **strchr** находит в строке, на которую указывает **s**, первое вхождение **c**, преобразованного в **char**.

Завершающий нулевой символ считается частью строки.

Возвращает

Функция **strchr** возвращает указатель на найденный символ или нулевой указатель, если символ в строке не встречается.

Функция `strrchr()`

```
#include <string.h>
char *strrchr(const char *s,  // где ищем
              int c);         // что ищем
```

Функция **`strrchr`** находит в строке, на которую указывает **`s`**, последнее вхождение **`c`**, преобразованного в **`char`**.

Завершающий нулевой символ считается частью строки.

Возвращает

Функция **`strrchr`** возвращает указатель на символ или нулевой указатель, если в строке **`c`** не встречается.

Функции `strspn()` и `strcspn()` — определение длины префикса подстроки

```
#include <string.h>

size_t strspn(const char *s1,   // строка
              const char *s2);  // подстрока

size_t strcspn(const char *s1,  // строка
               const char *s2); // подстрока
```

Функция **strspn** вычисляет длину максимального начального сегмента строки, на которую указывает **s1**, состоящего полностью из символов строки, на которую указывает **s2**.

Функция **strcspn** вычисляет длину максимального начального сегмента строки, на которую указывает **s1**, который полностью состоит из символов, не входящих в строку, на которую указывает **s2**.

Возвращает

Функции возвращают длину сегмента.

Функция `strpbrk()` — ищет в строке любой символ из набора байтов

```
#include <string.h>
char *strpbrk(const char *s1,  // строка поиска
              const char *s2); // набор поиска
```

Функция **strpbrk** находит первое вхождение в строке, на которую указывает **s1**, любого символа из строки, на которую указывает **s2**.

Возвращает

Функция **strpbrk** возвращает указатель на символ или нулевой указатель, если в **s1** не встречается символ из **s2**.

Функция `strstr()` — поиск подстроки

```
#include <string.h>
char *strstr(const char *s1,  // где искать
             const char *s2); // что искать
```

Функция **strstr** находит первое вхождение в строке, на которую указывает **s1**, последовательности символов (исключая завершающий нулевой символ) в строке, на которую указывает **s2**.

Возвращает

Функция **strstr** возвращает указатель на найденную строку или нулевой указатель, если строка не найдена.

Если **s2** указывает на строку нулевой длины, функция возвращает **s1**.

Функция `strtok()` — извлечение лексем из строки

```
#include <string.h>
char *strtok(char * restrict s1,           // строка с лексемамий
             const char * restrict delim); // строка с символами-разделителями
```

Последовательность вызовов функции **strtok** разбивает строку, на которую указывает **s1**, на последовательность лексем (токенов), каждый из которых ограничивается символом из строки, на которую указывает **s2**.

Первый вызов в последовательности имеет ненулевой первый аргумент.

Последующие вызовы в последовательности имеют нулевой первый аргумент.

Строка с разделителями, на которую указывает **s2**, может отличаться от вызова к вызову.

Первый вызов в последовательности ищет в строке, на которую указывает **s1**, первый символ, который не содержится в текущей строке разделителей, на которую указывает **s2**.

Если такой символ не найден, значит, в строке, на которую указывает **s1**, нет лексем, и функция **strtok** возвращает нулевой указатель.

Если такой символ найден, это начало первой лексемы.

Затем с этого места **strtok** ищет символ, который содержится в текущей строке разделителей.

Если такой символ не найден, текущая лексема расширяется до конца строки, на которую указывает **s1**, и последующие поиски лексемы вернут нулевой указатель.

Если такой символ найден, он перезаписывается нулевым символом, который завершает текущую лексему.

Функция **strtok** сохраняет указатель на следующий символ, с которого начнется следующий поиск лексем.

Каждый последующий вызов с нулевым указателем в качестве значения первого аргумента запускает поиск с сохраненного указателя и ведет себя, как описано выше.

Функция **strtok** не требует каких-либо действий, чтобы избежать гонок данных с другими вызовами функции **strtok**.

Функция ведет себя так, как если бы ни одна библиотечная функция не вызывала функцию **strtok**.

Возвращает

Функция **strtok** возвращает указатель на первый символ лексемы или нулевой указатель, если лексемы не найдено.

Пример

```
#include <string.h>
static char str[] = "?a???b,,,#c";
char *t;
t = strtok(str, "?");   // t указывает на лексему "a"
t = strtok(NULL, ",");  // t указывает на лексему "??b"
t = strtok(NULL, "#,"); // t указывает на лексему "c"
t = strtok(NULL, "?");  // t является NULL-указателем
```

Эти функции:

- изменяют свой первый аргумент.
- не могут использоваться со строками-константами.
- теряется идентичность байта-разделителя.
- при анализе функция **strtok()** использует статический буфер, поэтому не является безопасной для нитей. В этом случае можно использовать **strtok_r()**.

Разные функции

`memset()` — установить байт в массиве в указанное значение
`strerror()` — преобразовать код в строку
`strlen()` — вычислить длину строки

Функция memset()

```
#include <string.h>
void *memset(void *s, int c, size_t n);
```

Функция **`memset`** копирует значение **`c`** (преобразованное в **`unsigned char`**) в каждый из первых **`n`** символов объекта, на который указывает **`s`**.

Возвращает

Функция **`memset`** возвращает значение **`s`**.

Функция strerror()

```
#include <string.h>
char *strerror(int errnum);
```

Функция **strerror()** отображает число в **errnum** на строку сообщения.

Обычно значения **errnum** поступают из **errno**, но **strerror** отображает любое значение типа **int** в соответствующее сообщение или в «**Unknown error nnn**».

Возвращает

Функция **strerror** возвращает указатель на строку, содержимое которой зависит от локали (**LC_MESSAGES**). Указанный массив не должен изменяться программой, однако может быть перезаписан последующим вызовом функции **strerror**.

Функция strlen()

```
#include <string.h>
size_t strlen(const char *s);
```

Функция **strlen** вычисляет длину строки, на которую указывает **s**.

Возвращает

Функция **strlen** возвращает количество символов, предшествующих завершающему нулевому символу.