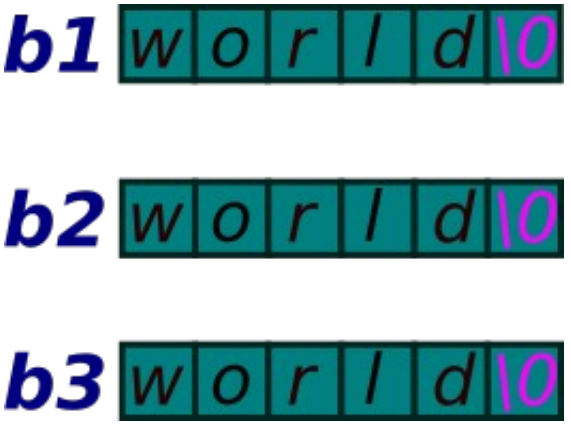
Работа со строками

Основы языка C, лекция 8

Что такое строка

- Строка — это последовательность символов с **'\0'** в конце
- Константы
  - **'0'** символьная, численное значение 48, **'\0'** == 0
  - **"**hello**"** строковая
- Нет специального типа для работы со строками
  - char str[10];
  - char * s;

Массив char



- Одинаковые массивы:
- char b1[6] = {'w', 'o', 'r', 'l', 'd', '\0'};
- char b2[ ] = {'w', 'o', 'r', 'l', 'd', '\0'};
  // размер массива вычисляется автоматически
- char b3[ ] = "world";
  // стандарт позволяет написать при инициализации не каждый символ по отдельности, а строковую константу.
- каждый массив хранится единым куском в памяти

Массив и указатель

- Хранятся по-разному
- // массив символов
  char s1 [ ] = "world";

- char * s2 = "world";
  // указатель на строковую константу
- Размер:
  sizeof (s1) == 6
  sizeof (s2) == sizeof (void*) // 4 или 8
- Можно изменять ?
  s1[0] = 'W'; // Ok
  s2[0] = 'W'; // Segmentation Fault read only memory

== - это НЕ сравнение строк

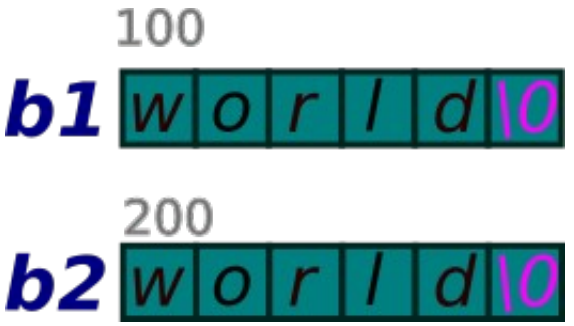
- s1 == s2

  s1 и s2 указывают на одну и ту же область памяти

- strcmp

- b1 == b2 // ложь

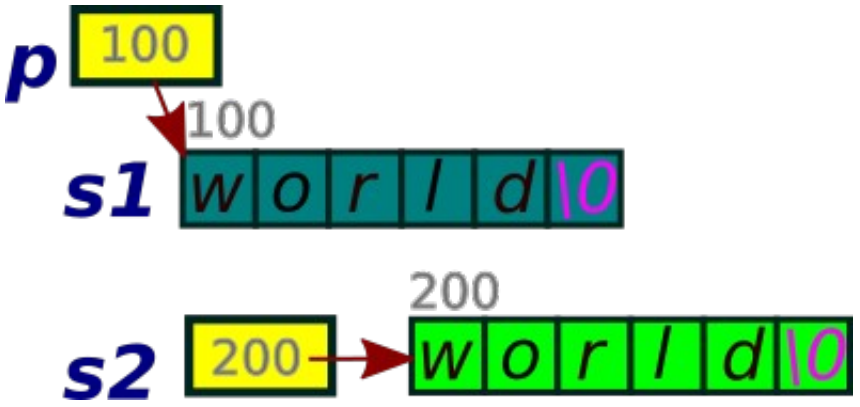
  strcmp(b1, b2) == 0 // истина



- s1 == s2 // ложь

  p == s2 // ложь

  p == s1 // истина



- strcmp (s1, s2)==0 // истина

  strcmp (s1+3, s2+3)==0

Печать строки

- char str[ ] = "world";
  char * str2 = "hello";
- for (int i = 0; str[i] != '\0'; i++)
  printf("%c", str[i]); // fputc(str[i], stdout)
- printf("**%s**", str); // от указанного адреса до \0
  printf("%s", str2); // hello

  printf ("%s\n", str+1); // orld
  printf ("%s\n", str2+1); // ello
- fputs (str, stdout);

Чтение

- Нужно выделить место в памяти (любым способом)
- char s [100];
- char * s = malloc (100);
- scanf("**%s**", s); // плохо, возможно переполнение
- printf("(%s)\n", s); // печать прочитанного в (..)

Input	Output	Что
qaz wsxedc	(qaz)	1 слово, а не строка
qaz123. wsx	(qaz123.)	слово — до пробельного символа

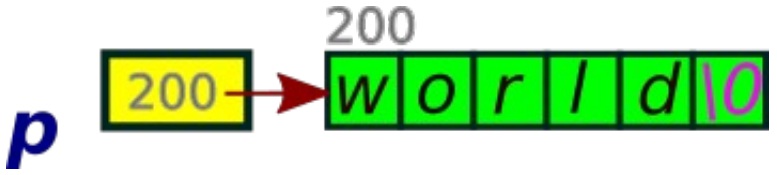
Контроль переполнения

- char s [10];
- scanf("**%5s**", s); // не более 5 символов
- printf("(%s)\n", s); // печать до \0
- не более n символов + '\0'
  char a[**10**]; scanf("%**9**s", a);

Input	Output	Что
qazwsxedc	(qazws)	5 символов + \0
ab	(ab)	или меньше до \0

Выделяем память %ms

- char * p;
- scanf("**%ms**", &p);
  // p будет указывать на выделенную память
  // неявный вызов **malloc**
- p[0] = 'W'; // работаем как с массивом
- printf("(%s)\n", p); // печать до \0
- **free**(p); // когда станет не нужной



Чтение строки

- char * **fgets** (char ***s**, int **size**, FILE ***stream**)
- char a [10];
  fgets (a, 5, stdin); // 1234567890
  printf("%s\n", a); // 1234
- то есть 5 символов, считая \0
- #define STRSIZE 1024
  char a[STRSIZE];
  fgets(a, STRSIZE, stdin); // sizeof(a) == STRSIZE
- char * **gets** (char *s); // нельзя,
  // нет контроля переполнения,
  // удалена из современного C

Читать текст до конца

- По словам:

```
char s[1001];
while (1 == scanf("%1000s", s))
    printf("%s\n", s);
// можно scanf("%*s", sizeof(s)-1, s)
```

- по строкам

```
char s[1000];
while (fgets(s, 1000, stdin) != NULL)
    printf("%s\n", s);
```

- Закончить поток: hello.exe < data.txt

  ввести с клавиатуры:

  UNIX: Ctrl+D Windows: Ctrl+Z

Стандартные функции языка C по работе со строками

Основы языка C, лекция 8

#include <ctype.h>

- int **isalnum** (int c);
  int **isalpha** (int c);
  int **iscntrl** (int c);
  int **isdigit** (int c);
  int **isgraph** (int c);
  int **islower** (int c);
  int **isprint** (int c);
  int **ispunct** (int c);
  int **isspace** (int c);
  int **isupper** (int c);
  int **isxdigit** (int c);

  int **tolower** (int c);
  int **toupper** (int c);

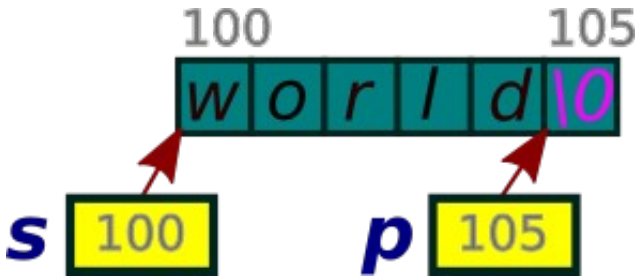
- int **isalnum_l** (int c, locale_t locale);

#include <string.h>

- size_t **strlen** (const char * s);
  длину строки до \0 (не считая \0)
- printf("%**z**d\n", **strlen** ("abc")); // 3
- char a [10] = "abc";
  printf("%**z**d\n", **strlen** ("abc")); // 3
  printf("%**z**d\n", **sizeof** (a)); // 10

Напишем сами

- size_t mystrlen (const char * s) {
  for (size_t i = 0; s[i] != '\0'; i++)
  ;
  return i; // проверьте для строки из 3 букв
  }
- size_t mystrlen (const char *s) {
  const char * p;
  for (p = s; *p != '\0'; p++)
  ;
  return p — s;
  }



Еще компактнее

- size_t mystrlen (const char *s) {
  const char * p = s;
  for (; *p; p++)   // '\0' — это ноль
  ;
  return p — s;
  }
- size_t mystrlen (const char *s) {
  const char * p = s;
  while (*p++)   // приоритет ++ выше
  ;
  return p - s - 1;
  }

Сравнение строк

- int **strcmp** (const char ***s1**, const char ***s2**);
- int **strncmp** (const char ***s1**, const char ***s2**, size_t **n**);
- strcmp сравнивает лексикографически строки: s1 s2
- strncmp - не более n символов
- if (0 == strcmp(s, "Treasure!"))

Вернет		Пример
0	одинаковые	strcmp("abc", "abc")
< 0	s1 < s2	strcmp ("abc", "xyz")
> 0	s1 > s2	strcmp ("abc", "aBcd")

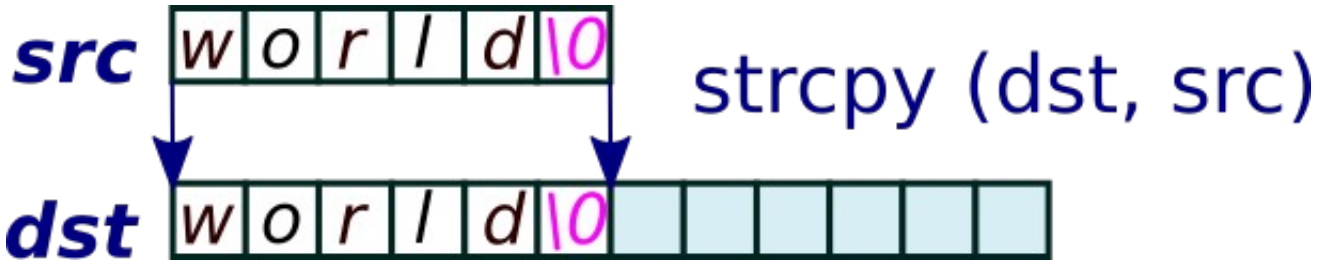
strncmp

- Сравнение посимвольное ASCII-кодов до
  - первого различия
  - конца строки или до конца
  - n

Вернет	Пример
0	strncmp("abcx", "abcA", 3)
> 0	strncmp("abcx", "abcABC", 10)
< 0	strncmp("abcx", "abcxABC", 10)

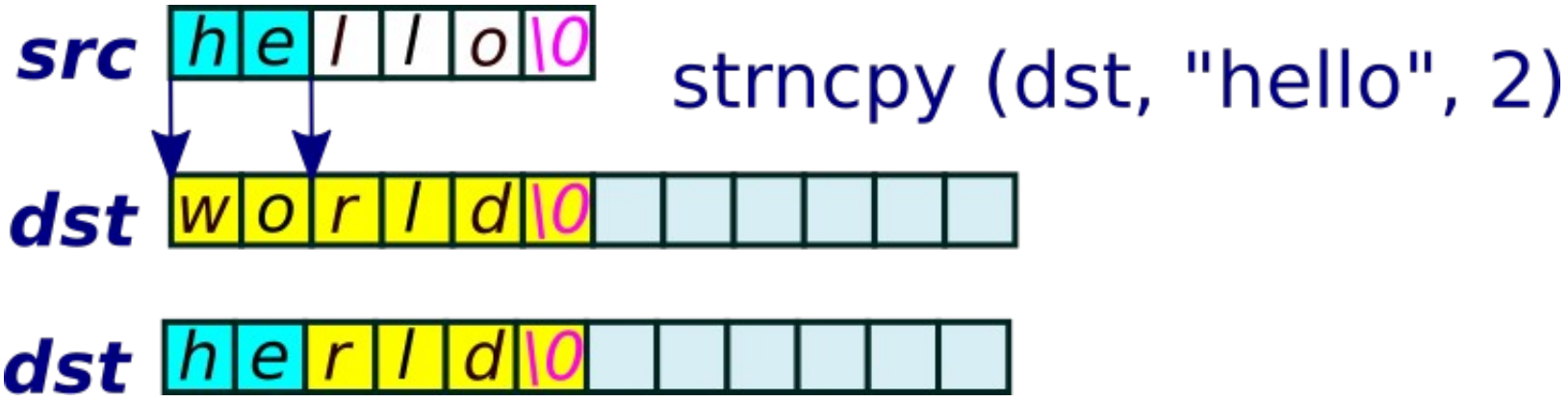
Копирование строк

- char ***strcpy**(char ***dest**, const char ***src**);
- char ***strncpy**(char ***dest**, const char ***src**, size_t **n**);
- src — source (откуда) x = 5
  dest — destination (куда) — его вернут
  n — не более n символов (\0 может не ставить)



Копирование строк

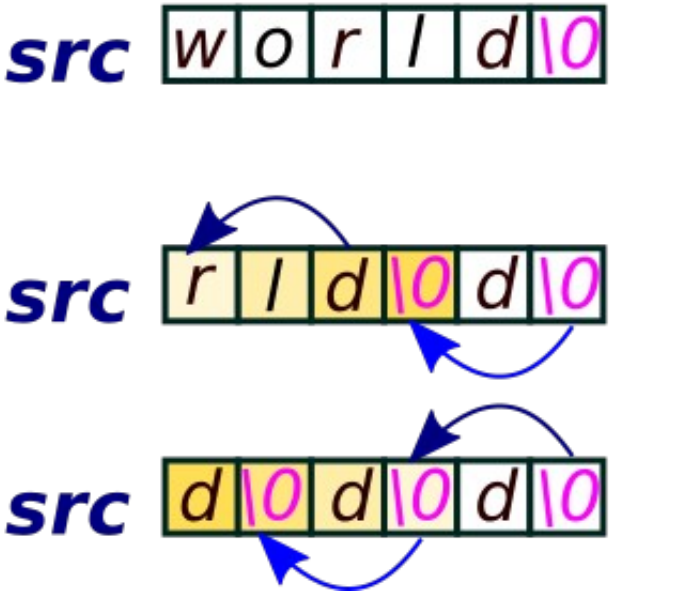
- char a [100]; // сами заботимся о месте
  strcpy (a, **"hello"**);
  printf("%s\n", a); // hello
  strncpy (a, **"abc"**, **2**);
  printf("%s\n", a); // abllo a[2] = '\0';
  strncpy (a, **"abc"**, **10**); // abc остальное нулями



Пересечениям - нет

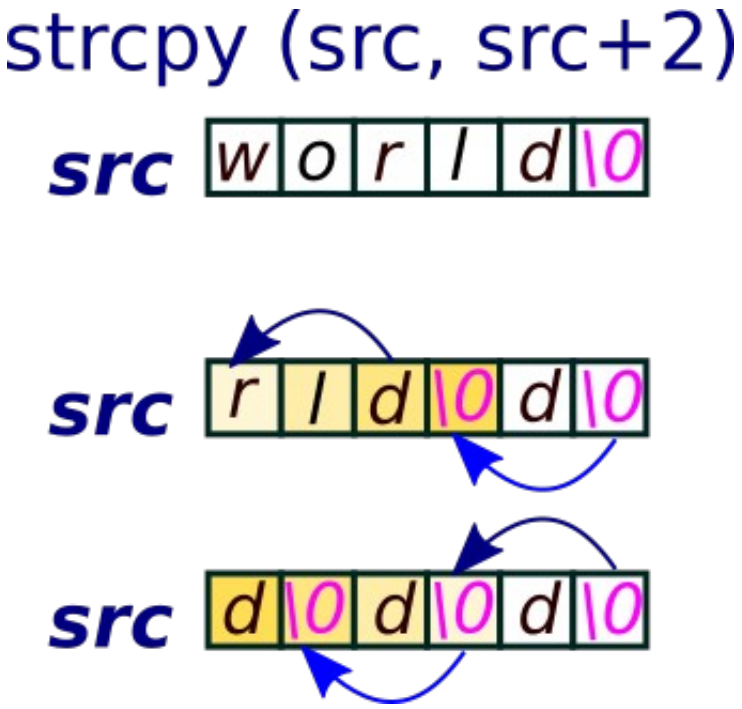
- char * mystrcpy (char * dst, const char * src) {
  for (int i = 0; src[i] != '\0'; i++)
  dst[i] = src[i] ;
  dst[i] = '\0'; // не забыть \0 в конце
  return dst;
  }
-

strcpy (src, src+2)



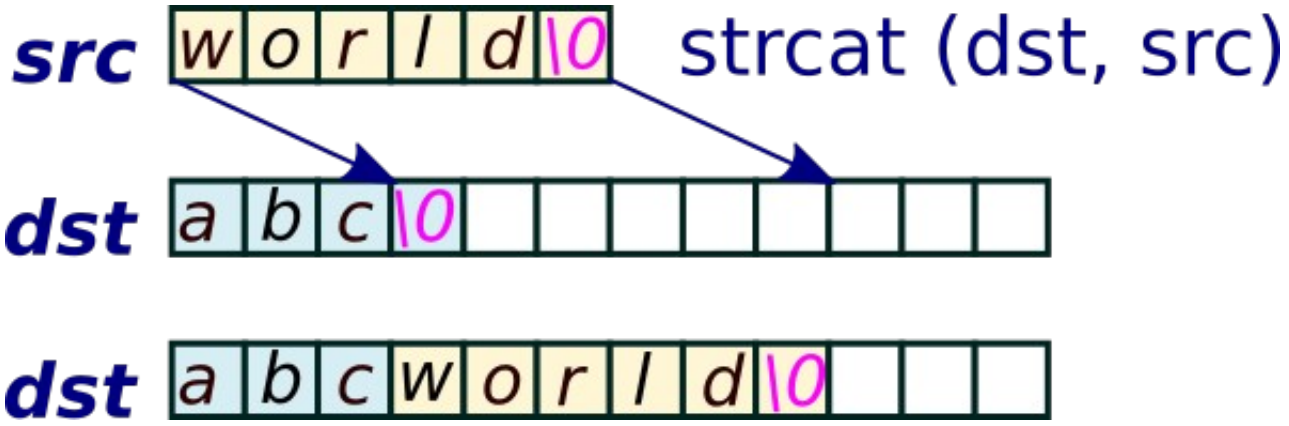
Пересечениям - нет

- char * mystrcpy (char * dst, const char * src) {
  for (int i = strlen(str); i>=0; i--)
  dst[i] = src[i] ;
  return dst;
  }



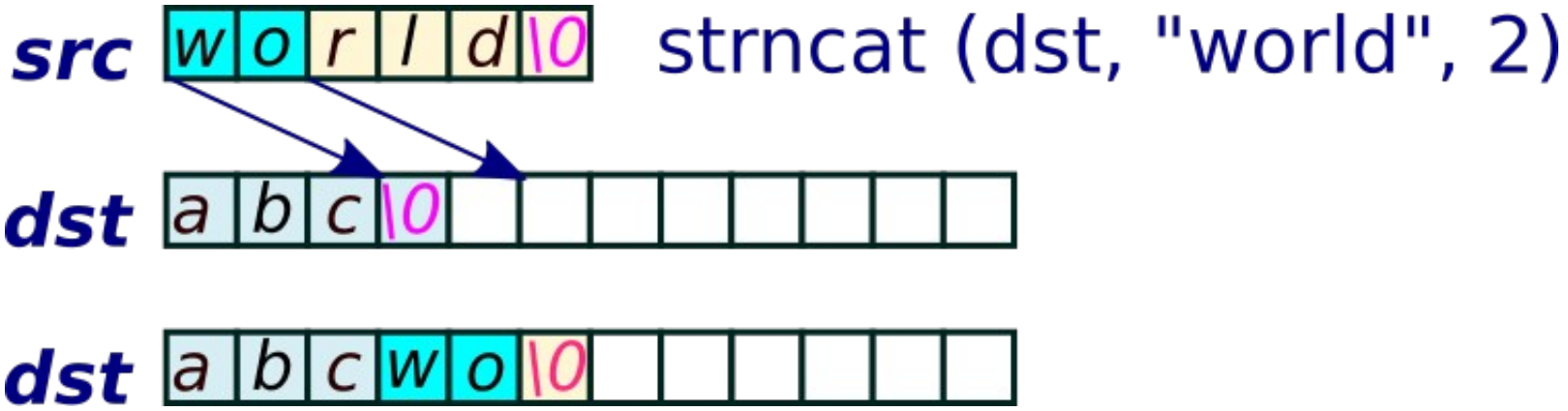
Конкатенация (склеить)

- char ***strcat**(char ***dest**, const char ***src**);
- char ***strncat**(char ***dest**, const char ***src**, size_t **n**);
- src — source (откуда) x = 5
  dest — destination (куда) — его вернут
  n — не более n символов (\0 ставит всегда)



Конкатенация (склеить)

- char a [100] = "abc"; // сами заботимся о месте
  strcat (a, **"hello"**);
  printf("%s\n", a); // abchello
  strncat (a, **"xyz"**, **2**);
  printf("%s\n", a); // abchelloxy \0 записан
  strncat (a, **"END"**, **10**); // abchelloxyEND



Поиск символа

- char ***strchr**(const char ***s**, int **c**);
- char ***strrchr**(const char ***s**, int **c**);
- в строке s ищем символ c, возвращаем его:
  - указатель на первое вхождение — strchr
  - указатель на последнее вхождение — strrchr
  - NULL — символа в строке нет
- char * a = "Hello, world!";
  char * p1 = **strchr** (a, **"l"**);
  char * p2 = **strrchr** (a, **"l"**);
  printf("%s\n", p1); // llo, world!
  printf("%s\n", p2); // d!

Принадлежит алфавиту

- ```
  const char * s = "({[<";
  int c = getchar();
  char * p = strchr(s, c);

  if (p != NULL)
      printf("Символ %c открывающая скобка\n", c);
  ```
- ```
  char * path = "/home/student/hello.c";
  char * file = strrchr(path, '/');
  if (file)
       file ++;
  printf("filename is %s\n", file);  // filename is hello.c
  ```

Поиск подстроки

- char ***strstr**(const char ***s**, const char * **substr**);
- в строке s ищем подстроку substr, возвращает
  - указатель на первое вхождение
  - NULL — подстрока не входит в строку
- char * text = "I have a dog. I have a bomb. I have a cat";
  if (NULL != **strstr**(text, "bomb"))
  printf("WAAA! BOMB!!!!\n");

strtok — разбить на токены

- char ***strtok**(char ***s**, const char * **delim**);
- разбивает строку на подстроки по разделителю delim
- модифицирует строку s, записывая \0 в места, где находятся delim
- возвращает указатель на очередную подстроку после каждой модификации
- Для каждого последующего вызова, кроме первого, указываем NULL вместо s
- Последний раз (нет больше delim) вернет NULL

strtok пример

- ```
  char s[] = "a,bb; cccc?dd";
  const char * delim = ",;?!. \t";
  for (char * p = strtok(s, delim);
          p != NULL;
          p = strtok (NULL, delim) )
      printf("%s\n", p);
  ```

- ```
  a
  bb
  ccc
  dd
  ```

Из числа в строку

- int **sprintf**(char ***s**, const char * **format, ...**);
- Преобразуем часы h и минуты m в строку
- int h = 21, m = 7;
  // память для строки должна быть выделена
  char s[100];
- // тут никто не печатает на экран, а заполняет массив s
  sprintf (s, "%02d:%02d", h, m);
- // тут печатаем эту строку на экран (если нужно)
  printf("%s\n", s);

Из строки в число

- int **sscanf**(char ***s**, const char * **format, ...**);
- Преобразуем строку в часы h и минуты m
- int h, m;
  char s[100] = "21:07";
  sscanf (s, "%d:%d", &h, &m);
- Простые преобразования stdlib.h
  int **atoi** (const char ***str**);
  long **atol** (const char ***str**);
  long long **atoll** (const char ***str**);
  double **atof** (const char ***str**);
- int x = atoi("123");

С контролем ошибок

- long int **strtol** (const char ***s**, char ** **endptr,** int **base**);
  long long **strtoll** (const char ***s**, char ** **endptr,** int **base**);
- конвертирует начальную часть строки s в long int по основанию base (от 2 до 36 или 0).
  В endptr записывается указатель на первый невалидный символ строки (или не записывается, если endptr == NULL).
- В начале строки могут быть пробельные символы isspace, далее [+|-], 0x или 0X — если base=16.
- Потом цифры.
  Если base > 10, то **a** или **A** — это 10, **z** или **Z** — это 35.

strtol - пример

- long int **strtol** (const char ***s**, char ** **endptr,** int **base**);
  atol это strtol (s, NULL, 10)

- base = 0:
  s = "123" → 123, base = 10 или 0
  s = "0123" → $123_8 = 83_{10}$, base = 8 или 0

- long int x;

```
x = strtol ("123", NULL, 10);     // x = atol("123")
x = strtol ("    456", NULL, 10); // x = 456
x = strtol ("    75", NULL, 0);   // x = 75
x = strtol ("020", NULL, 0);      // x = 16
x = strtol ("abc", NULL, 32);     // 10*32^2 + 11*32 + 12
```

strtol — обработка ошибок

- long int **strtol** (const char ***s**, char ** **endptr,** int **base**);

- char * end;
  x = **strtol** ("123abc", **&**end, 10);
  printf("x=%ld end=%s\n", x, end);
  // x = 123 end=abc

- char * str = "hello";
  char * end;
  long int x = **strtol** (str, **&**end, 10);
  if (str == end)
  printf("Это не число\n");

man 3 strtol

- errno = 0;
  val = strtol(str, &endptr, base);
- if ((errno == ERANGE && (val == LONG_MAX ||
  val == LONG_MIN)) || (errno != 0 && val == 0)) {
  perror("strtol"); exit(EXIT_FAILURE);
  }
- if (endptr == str) {
  fprintf(stderr, "No digits\n"); exit(EXIT_FAILURE);
  }
- printf("strtol() returned %ld\n", val);
- if (*endptr != '\0') /* Not necessarily an error... */
  printf("tail after number: %s\n", endptr);

другие функции

- unsigned long int **strtoul** (const char ***s**,
  char ** **endptr**, int **base**);
  unsigned long long int **strtoull** (const char ***s**,
  char ** **endptr**, int **base**);
- double **strtod** (const char ***s**, char ** **endptr**);
  double **strtof** (const char ***s**, char ** **endptr**);
  long double **strtold** (const char ***s**, char ** **endptr**);
  - без учета регистра:
  - "INF", "INFINITY" — бесконечность
  - "NAN" — Not a Number

const

- const — read-only, подсказка оптимизатору
- **const int** x = 12;
  x = 3; // error: assignment of read-only variable ‘x’
- char a[100] = "hello";
  const char * a1 = a;
  a1 = a1 + 2; // ok
  a1[2] = 'q';
  // assignment of read-only location ‘*(a1 + 2)’
- **char const *** b1 = a; // то же самое
  b1 = b1 + 2; // ok
  b1[2] = 'q';
  // assignment of read-only location ‘*(b1 + 2)’

char * const

- char s[100] = "hello";
  char * const s1 = s;
  s1 = s1 + 2; // assignment of read-only variable ‘s1’
  s1[2] = 'q'; // ok