**Работа со строками**

###

**Строки. Ввод-вывод строк. Форматированный ввод-вывод. Обработка строк с использованием стандартных функций языка С. Работа с памятью.**

**1.1. Объявление и инициализация строк.**

                Строкой называется массив символов, который заканчивается пустым символом ‘\0’. Строка объявляется как обычный символьный массив, например,

                **char  s1[10];         // строка длиной в девять символов**

**char  \*s2;              // указатель на строку**

Различие между указателями s1 и s2 заключается в том, что указатель s1 является именованной константой, а указатель s2 – переменной.

**Строковые константы заключаются в двойные кавычки в отличие от символов, которые заключаются в одинарные кавычки. Например,**

**“This is a string.”**

**Длина строковой константы не может превышать 509 символов** по стандарту. Однако, многие реализации допускают строки большей длины.

**При инициализации строк размерность массива лучше не указывать, это выполнит компилятор, подсчитав длину строки и добавив к ней единицу. Например,**

**char  s1[ ] = “This is a string.”;**

                В языке программирования С для работы со строками существует большое количество функций, прототипы которых описаны в заголовочных файлах stdlib.h и string.h. Работа с этими функциями будет рассмотрена в следующих параграфах.

**1.2. Ввод-вывод строк.**

**Для ввода строки с консоли служит функция**

**char\*  gets ( char  \*str );**

которая записывает строку по адресу str и возвращает адрес введенной строки. Функция прекращает ввод, если встретит символ ‘\n’ или EOF (конец файла). Символ перехода на новую строку не копируется. В конец прочитанной строки помещается нулевой байт. В случае успеха функция возвращает указатель на прочитанную строку, а в случае неудачи NULL.

**Для вывода строки на консоль служит стандартная функция**

**int  puts ( const  char  \*s );**

**которая в случае удачи возвращает неотрицательное число, а в случае неудачи – EOF.**

**Прототипы функций gets и puts описаны в заголовочном файле stdio.h.**

**Пример.**

**#include <stdio.h>**

**int main()**

**{**

**char str[80];**

**printf("Input String: ");**

**gets(str);**

**puts(str);**

**return 1;**

**}**

**1.3. Форматированный ввод-вывод.**

**Для форматированного ввода данных с консоли используется функция**

**int  scanf ( const char \*format, …);**

которая в случае успешного завершения возвращает количество единиц прочитанных данных, а в случае неудачи – EOF. Параметр format должен указывать на форматируемую строку, которая содержит спецификации форматов ввода. Количество и типы аргументов, которые следуют после строки форматирования, должны соответствовать количеству и типам форматов ввода, заданным в строке форматирования. Если это условие не выполняется, то результат работы функции непредсказуем.

                Пробел, символы '\t' или '\n' в форматной строке описывают один или более пустых символов во входном потоке, к которым относятся символы: пробел, ‘\t’, ‘\n’, ‘\v’, ‘\f’. Функция scanf пропускает пустые символы во входном потоке.

                Литеральные символы в форматной строке, за исключением символа %, требуют, чтобы во входном потоке появились точно такие же символы. Если такого символа нет, то функция scanf прекращает ввод. Функция scanf пропускает литеральные символы.

                В общем случае спецификация формата ввода имеет вид:

                %[\*] [ширина] [модификаторы] тип

где

                - символ ‘\*’ обозначает пропуск при вводе поля, определенного данной спецификацией;

                - ‘ширина’ определяет максимальное число символов, вводимых по данной спецификации;

                - ‘модификаторы’ уточняют тип аргументов;

                - ‘тип’ определяет тип аргумента.

                Тип может принимать следующие значения:

                c – символьный массив,

                s – строка символов, строки разделяются пустыми символами,

                d – целое число со знаком в 10 с/c,

                i – целое число со знаком, система счисления завит от двух первых цифр,

                u – целое число без знака в 10 с/с,

                o – целое число без знака в 8 с/c,

                х, Х – целое число без знака в 16 с/с,

                e, E, f, g, G – плавающее число,

                p – указатель на указатель,

                n – указатель на целое,

                […] – массив сканируемых символов, например, [A321].

В последнем случае из входного потока будут вводиться только символы, заключенные в квадратные скобки. Если первый символ внутри квадратных скобок равен ‘^’, то вводятся только те символы, которые не входят в массив. Диапазон символов в массиве задается через символ ‘-‘. При вводе символов ведущие пустые символы и завершающий нулевой байт строки также вводятся.

                **Модификаторы могут принимать следующие значения:**

**h – короткое целое,**

**l, L – длинное целое или плавающее,**

**и используются только для целых или плавающих чисел.**

                В следующем примере показаны варианты использования функции scanf. Обратите внимание, что перед спецификатором формата, начиная с ввода плавающего числа, стоит символ пробел.

**#include <stdio.h>**

**int main()**

**{**

**int           n;**

**double   d = 0.0;**

**char        c;**

**char\*     s;**

**printf("Input an integer: ");**

**scanf("%d", &n);**

**printf("Input a double: ");**

**scanf(" %lf", &d);**

**printf("Input a char: ");**

**scanf(" %c", &c);**

**printf("Input a string: ");**

**scanf(" %s", &s);**

**return 1;**

**}**

Обратите внимание, что в этой программе число с плавающей точкой проинициализировано. Это сделано для того, чтобы компилятор подключил библиотеку для поддержки работы с плавающими числами. Если этого не сделать, то на этапе выполнения при вводе плавающего числа произойдет ошибка.

                Для форматированного вывода данных на консоль используется функция

                int  printf ( const char \*format, …);

которая в случае успешного завершения возвращает количество единиц выведенных данных, а в случае неудачи – EOF. Параметр format представляет собой форматируемую строку, которая содержит спецификации форматов вывода. Количество и типы аргументов, которые следуют после строки форматирования, должны соответствовать количеству и типам спецификациям формата вывода, заданным в строке форматирования. **В общем случае спецификация формата вывода имеет вид:**

**%[флаги] [ширина] [.точность] [модификаторы] тип**

**где**

**- ‘флаги’ – это различные символы, уточняющие формат вывода;**

**- ‘ширина’ определяет минимальное количество символов, выводимых по данной спецификации;**

**- ‘.точность’ определяет максимальное число выводимых символов;**

**- ‘модификаторы’ уточняют тип аргументов;**

**- ‘тип’ определяет тип аргумента.**

**Для вывода целых чисел со знаком используется следующий формат вывода:**

**%[-] [+ | пробел] [ширина] [l] d**

где

                - – выравнивание влево, по умолчанию – вправо;

                + – выводится знак ‘+’, заметим, что для отрицательных чисел всегда выводится знак ‘-‘;

                ‘пробел’ – в позиции знака выводится пробел;

                l – модификатор типа данных long;

                d – тип данных int.

                Для вывода целых чисел без знака используется следующий формат вывода:

                %[-] [#] [ширина] [l] [u | o | x | X]

где

                # – выводится начальный 0 для чисел в 8 c/c или начальные 0x или 0X для чисел в 16 c/c,

                l – модификатор типа данных long;

                u – целое число в 10c/c,

                o – целое число в 8 c/c,

                x, X – целое число в 16 c/c.

                Для вывода чисел с плавающей точкой используется следующий формат вывода:

                %[-] [+ | пробел] [ширина] [.точность] [f | e | E | g | G]

где

                'точность' – обозначает число цифр после десятичной точки для форматов f, e и E или число значащих цифр для форматов g и G. Числа округляются отбрасыванием. По умолчанию принимается точность в шесть десятичных цифр;

                f – число с фиксированной точкой,

                e – число в экспоненциальной форме, экспонента обозначается буквой 'e',

                E – число в экспоненциальной форме, экспонента обозначается буквой 'E',

                g – наиболее короткий из форматов f или g,

                G – наиболее короткий из форматов f или G.

                Пример.

                printf ("n = %d\n  f = %f\n  e = %e\n  E = %E\n  f = %.2f", -123, 12.34, 12.34, 12.34, 12.34);

                               // печатает : n = 123  f = 12.340000  e = 1.234000e+001  E = 1.234000E+001  f = 12.34

**1.4. Форматирование строк.**

**Существуют варианты функций scanf и printf, которые предназначены для форматирования строк и называются соответственно sscanf и sprintf.**

**Функция**

**int  sscanf ( const  char  \*str, const  char  \*format, …);**

читает данные из строки, заданной параметром str, в соответствии с форматной строкой, заданной параметром format. В случае удачи возвращает количество прочитанных данных, а в случае неудачи – EOF. Например,

**#include <stdio.h>**

**int main()**

**{**

**char str[] = "a 10 1.2 String No input";**

**char c;**

**int n;**

**double d;**

**char s[80];**

**sscanf(str, "%c %d %lf %s", &c, &n, &d, s);**

**printf("%c\n", c);                // печатает: a**

**printf("%d\n", n);                               // печатает: 10**

**printf("%f\n", d);                // печатает: 1.200000**

**printf("%s\n", s);                 // печатает: String**

**return 1;**

**}**

               Функция

                int  sprintf (char  \*buffer, const  char  \*format, …);

форматирует строку в соответствии с форматом, который задан параметром format и записывает полученный результат в символьный массив buffer. Возвращает функция количество символов, записанных в символьный массив buffer, исключая завершающий нулевой байт. Например,

#include <stdio.h>

int main()

{

                char buffer[80];

                char str[] = "c = %c, n = %d, d = %f, s = %s";

                char c = 'c';

                int n = 10;

                double d = 1.2;

                char s[] = "This is a string.";

                sprintf(buffer, str, c, n, d, s);

                printf("%s\n", buffer);       // печатает: c = c, n = 10, d = 1.200000, s = This is a string

                return 1;

}

**1.5. Преобразование строк в числовые данные.**

                Прототипы функций преобразования строк в числовые данные приведены в заголовочном файле stdlib.h, который нужно включить в программу.

                Для преобразования строки в целое число используется функция

                int  atoi ( const  char  \*str);

которая в случае успешного завершения возвращает целое число, в которое преобразована строка str, а в случае – неудачи 0. Например,

                int  n;

                char  \*str = “-123”;

                n = atoi ( str );                      // n = -123

                Для преобразования строки в длинное целое число используется функция

                long  int  atol ( const  char  \*str);

которая в случае успешного завершения возвращает целое число, в которое преобразована строка str, а в случае – неудачи 0. Например,

                long  int  n;

                char  \*str = “-123”;

                n = atol ( str );                      // n = -123

                Для преобразования строки в число типа double используется функция

                double  atof ( const  char  \*str);

которая в случае успешного завершения возвращает плавающее число типа double, в которое преобразована строка str, а в случае – неудачи 0. Например,

                double  n;

                char  \*str = “-123.321”;

                n = atof ( str );                     // n = -123.321

                Следующие функции выполняют действия, аналогичные функциям atoi, atol, atof, но предоставляют более широкие возможности.

                Функция

                long  int  strtol ( const  char  \*str, char  \*\*endptr, int  base);

преобразует строку str в число типа long int, которое и возвращает. Параметры этой функции имеют следующее назначение.

                Если аргумент base равен 0, то преобразование зависит от первых двух символов строки str:

                - если первый символ – цифра от 1 до 9, то предполагается, что число представлено в 10 c/c;

                - если первый символ – цифра 0, а второй – цифра от 1 до 7, то предполагается, что число представлено в 8 c/c;

                - если первый символ 0, а второй – ‘Х’ или ‘х’, то предполагается, что число представлено в 16 c/c.

                Если аргумент base равен числу от 2 до 36, то это значение принимается за основание системы счисления и любой символ, выходящий за рамки этой системы, прекращает преобразование. В системах счисления с основанием от 11 до 36 для обозначения цифр используются символы от ‘A’ до ‘Z’ или от ‘a’ до ‘z’.

                Значение аргумента endptr устанавливается функцией strtol. Это значение содержит указатель на символ, который остановил преобразование строки str. В случае успешного завершения функция strtol возвращает преобразованное число, а в случае неудачи – 0. Например,

                int  main ( )

                {

                               long int  n;

                               char        \*p;

                               n = strtol (“12a”, &p, 0);

                               printf ( “ n = %ld, %stop = %c, n, \*p );          // n = 12, stop = a

                               n = strtol (“012b”, &p, 0);

                               printf ( “ n = %ld, %stop = %c, n, \*p );          // n = 10, stop = b

                               n = strtol (“0x12z”, &p, 0);

                               printf ( “ n = %ld, %stop = %c, n, \*p );          // n = 18, stop = z

                               n = strtol (“01117”, &p, 0);

                               printf ( “ n = %ld, %stop = %c, n, \*p );          // n = 7, stop = 7

                               return 1;

                }

                Функция

                unsigned  long  int  strtol ( const  char  \*str, char  \*\*endptr, int  base);

работает аналогично функции strtol, но преобразует символьное представление числа в число типа unsigned long int.

                Функция

                double  strtod ( const  char  \*str, char  \*\*endptr);

преобразует символьное представление числа в число типа double.

                Все функции, перечисленные в этом параграфе, прекращают свою работу при встрече первого символа, который не подходит под формат рассматриваемого числа.

                Кроме того, в случае если символьное значение числа превосходит диапазон допустимых значений для соответствующего типа данных, то функции atof, strtol, strtoul, strtod устанавливают значение переменной errno в ERANGE. Переменная errno и константа ERANGE определены в заголовочном файле math.h. При этом функции atof и strtod возвращают значение HUGE\_VAL, функция strtol возвращает значение LONG\_MAX или LONG\_MIN, а функция strtoul – значение ULONG\_MAX.

                Для преобразования числовых данных в символьные строки могут использоваться нестандартные функции itoa, ltoa, utoa, ecvt, fcvt и gcvt. Но лучше для этих целей использовать стандартную функцию sprintf.

**1.6. Стандартные функции для работы со строками.**

                В этом параграфе рассмотрены функции для работы со строками, прототипы которых описаны в заголовочном файле string.h.

                **1. Сравнение строк. Для сравнения строк используются функции strcmp и strncmp.**

                **Функция**

**int  strcmp ( const  char  \*str1, const  char  \*str2);**

лексикографически сравнивает строки str1, str2 и возвращает –1, 0 или 1, если строка str1 соответственно меньше, равна или больше строки str2.

                Функция

                int  strncmp ( const  char  \*str1, const  char  \*str2, size\_t  n);

лексикографически сравнивает не более чем n первых символов из строк str1 и str2. Функция возвращает –1, 0 или 1, если первые n символов из строки str1 соответственно меньше, равны или больше первых n символов из строки str2.

                Пример.

// **пример сравнения строк**

**#include <stdio.h>**

**#include <string.h>**

**int main()**

**{**

**char str1[] = "aa bb";**

**char str2[] = "aa aa";**

**char str3[] = "aa bb cc";**

**printf("%d\n", strcmp(str1, str3));                   // печатает: -1**

**printf("%d\n", strcmp(str1, str1));                   // печатает: -0**

**printf("%d\n", strcmp(str1, str2));                   // печатает: 1**

**printf("%d\n", strncmp(str1, str3, 5));            // печатает: 0**

**return 1;**

**}**

**2. Копирование строк. Для копирования строк используются функции strcpy и strncpy.**

**Функция**

**char  \*strcpy ( char  \*str1, const  char  \*str2 );**

копирует строку str2 в строку str1. Строка str2 копируется полностью, включая завершающий нулевой байт. Функция возвращает указатель на str1. Если строки перекрываются, то результат непредсказуем.

**Функция**

**char  \*strncpy ( char  \*str1, const  char  \*str2, size\_t  n );**

копирует n символов из строки str2 в строку str1. Если строка str2 содержит меньше чем n символов, то последний нулевой байт копируется столько раз, сколько нужно для **расширения строки str2 до n символов. Функция возвращает указатель на строку str1.**

**Пример.**

**char  str1[80];**

**char  str2 = "Copy string.";**

**strcpy ( str1, str2 );**

**printf ( str1 );                                       // печатает: Copy string**.

**4. Соединение строк. Для соединения строк в одну строку используются функции strcat и strncat.**

**Функция**

**char\*  strcat ( char  \*str1, const  char  \*str2 );**

присоединяет строку str2 к строке str1, причем завершающий **нулевой байт строки str1 стирается.** Функция возвращает указатель на строку str1.

                Функция

                char\*  strncat ( char  \*str1, const  char  \*str2, size\_t  n );

присоединяет n символов из строки str2 к строке str1, причем завершающий нулевой байт строки str1 стирается. Функция возвращает указатель на строку str1. если длина строки str2 меньше n, то присоединяются только символы, входящие в строку str2. После соединения строк к строке str1 всегда добавляется нулевой байт. Функция возвращает указатель на строку str1.

                Пример.

#include <stdio.h>

#include <string.h>

                int  main ( )

                {

**char  str1[80] = "String ";**

**char  str2 = "catenation ";**

**char  str3 = "Yes No";**

**strcat  ( str1, str2 );**

**printf ("%s\n", str1 );                          // печатает: String catenation**

**strncat ( str1, str3, 3 );**

**printf ("%s\n", str1 );                          // печатает: String catenation Yes**

**return 1;**

                }

**5. Поиск символа в строке. Для поиска символа в строке используются функции strchr, strrchr, strspn, strcspn и strpbrk.**

**Функция**

**char\*  strchr ( const  char  \*str, int  c );**

**ищет первое вхождение символа, заданного параметром c, в строку str. В случае успеха функция возвращает указатель на первый найденный символ, а в случае неудачи – NULL.**

**Функция**

**char\*  strrchr ( const  char  \*str, int  c );**

**ищет последнее вхождение символа, заданного параметром c, в строку str. В случае успеха функция возвращает указатель на последний найденный символ, а в случае неудачи – NULL.**

                Пример.

#include <stdio.h>

#include <string.h>

                int  main ( )

                {

**char  str[80] = "Char search";**

**printf ("%s\n", strchr ( str, 'r' ));                        // печатает: r search**

**printf ("%s\n", strrchr ( str, 'r' ));                       // печатает: rch**

**return 1;**

                }

                Функция

                size\_t  strspn ( const  char  \*str1, const  char  \*str2 );

возвращает индекс первого символа из строки  str1, который не входит в строку str2.

                Функция

                size\_t  strcspn ( const  char  \*str1, const  char  \*str2 );

возвращает индекс первого символа из строки  str1, который входит в строку str2.

                Пример.

                int  main ( )

                {

                               char  str[80] = "123 abc";

                               printf ("n = %d\n", strspn ( str, "321" );                          // печатает: n = 3

                               printf ("n = %d\n", strcspn ( str, "cba" );                        // печатает: n = 4

                               return 1;

                }

                Функция

                char\*  strpbrk ( const  char  \*str1, const  char  \*str2 );

находит первый символ в строке str1, который равен одному из символов в строке str2. В случае успеха функция возвращает указатель на этот символ, а в случае неудачи – NULL.

                Пример.

                char  str[80] = "123 abc";

                printf ("%s\n", strpbrk ( str, "bca" ));              // печатает: abc

                6. Сравнение строк. Для сравнения строк используются функция strstr.

                Функция

                char\*  strstr ( const  char  \*str1, const  char  \*str2 );

находит первое вхождение строки str2 (без конечного нулевого байта) в строку str1. В случае успеха функция возвращает указатель на найденную подстроку, а в случае неудачи – NULL. Если указатель str1 указывает на строку нулевой длины, то функция возвращает указатель str1.

                Пример.

                char  str[80] = "123 abc 456;

                printf ("%s\n", strstr ( str, "abc");                     // печать: abc 456

                7. Разбор строки на лексемы. Для разбора строки на лексемы используется функция strtok.

                Функция

                char\*  strtok ( char  \*str1, const  char  \*str2 );

возвращает указатель на следующую лексему (слово) в строке str1, в которой разделителями лексем являются символы из строки str2. В случае если лексемы закончились, то функция возвращает NULL. При первом вызове функции strtok параметр str1 должен указывать на строку, которая разбирается на лексемы, а при последующих вызовах этот параметр должен быть установлен в NULL. После нахождения лексемы функция strtok записывает после этой лексемы на место разделителя нулевой байт.

                Пример.

#include <stdio.h>

#include <string.h>

                int  main( )

                {

                               char  str[ ] = "12 34 ab cd";

                               char \*p;

                               p = strtok ( str, " " );

                               while (p)

                               {

                                               printf ( "%s\n", p );                             // печатает в столбик значения: 12 34 ab cd

                                               p = strtok ( NULL, " " );

                               }

                               return 1;

                }

                **8. Определение длины строки. Для определения длины строки используется функция strlen.**

**Функция**

**size\_t  strlen ( const  char  \*str);**

**возвращает длину строки, не учитывая последний нулевой байт. Например,**

**char  str[] = "123";**

**printf ("len = %d\n", strlen ( str ));**

 **// печатает: len** = 3

**1.7. Функции для работы с памятью.**

**В заголовочном файле string.h описаны также функции для работы с блоками  памяти, которые аналогичны  соответствующим функциям для работы со строками.**

**Функция**

**void\*  memchr ( const  void  \*str, int  c,  size\_t  n );**

**ищет первое вхождение символа, заданного параметром c, в n байтах строки str.**

                Функция

                int  memcmp ( const  void  \*str1, const  void  \*str2, size\_t  n);

сравнивает первые n байт строк str1 и str2.

                Функция

                void\*  memcpy ( const  void  \*str1, const  void  \*str2, size\_t  n);

копирует первые n байт из строки str1 в строку str2.

                Функция

                void\*  memmove ( const  void  \*str1, const  void  \*str2, size\_t  n);

копирует первые n байт из строки str1 в строку str2, обеспечивая корректную обработку перекрывающихся строк.

                Функция

                void\*  memset ( const  void  \*str, int  c, size\_t  n);

копирует символ, заданный параметром c, в первые n байтов строки str.

**Смотрите:**

1. [C++. Функции для работы с String (AnsiString)](http://www.itmathrepetitor.ru/c-funkcii-dlya-raboty-s-string-ansistring/%22%20%5Co%20%22C%2B%2B.%20%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D0%B8%20%D0%B4%D0%BB%D1%8F%20%D1%80%D0%B0%D0%B1%D0%BE%D1%82%D1%8B%20%D1%81%20String%20%28AnsiString%29)
2. [С++. Задачи с решениями. Строки](http://www.itmathrepetitor.ru/s-zadachi-s-resheniyami-stroki/%22%20%5Co%20%22%D0%A1%2B%2B.%20%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B8%20%D1%81%20%D1%80%D0%B5%D1%88%D0%B5%D0%BD%D0%B8%D1%8F%D0%BC%D0%B8.%20%D0%A1%D1%82%D1%80%D0%BE%D0%BA%D0%B8)