Strings em C

Rafael Lima de Carvalho

1 Visão geral

- As strings são arrays de caracteres.
- A biblioteca string.h possui funções especializadas para lidar com strings
- A entrada e saída de strings geralmente se difere de variáveis convencionais.

2 Objetivos de aprendizagem

Ao final deste módulo, o aluno deverá ser capaz de:

1. Ler corretamente uma string;
2. Processar strings e manipular as funções especializadas em strings.

3 Conteúdo

3.1 Locale

Cansado de ter que imprimir coisas sem acento? É possível modificar a variável de localização para que tais acentos sejam aceitos. Veja o exemplo abaixo:

```
1  #include <stdio.h>
2  #include <locale.h>
3
4  int main(int argc, char* argv[])
5  {
6    printf ("Localidade atual: %s\n", setlocale(LC_ALL,NULL) );
7    printf("Caso não seja PT_BR ou Portuguese, os acentos aparecerao
       desconfigurados\n");
8    printf("Nova localidade: %s \n",setlocale(LC_ALL,""));
9    printf("Poder usar ç cedilha e outros acentos é muito bom,  não é ?!\n");
10   return 0;
11 }
```

Cuidados ao mudar a localidade. No programa acima, se fores imprimir ou ler um float ou doble, provavelmente o caractere de separação da parte inteira e flutuante será uma vírgula e não um ponto, como estamos acostumados.

3.2 Declarando, inicializando e funções de I/O

Strings são arrays do tipo char, exemplo: `char nome[128];` este código prepara uma string de 127 caracteres, sendo que o último (de índice 127) é reservado para o caractere de fim da string `'\0'`.

Outro exemplo é: `char nome[] = "Teste de string";`. Note que a string pode ser inicializada de forma distinta do array comum, porém também aceita a inicialização com caracteres separados por vírgula.

As funções para I/O de strings são:

- `char *gets(char *s);` e `char *fgets(char *s, int size, FILE *stream);`
- `int sprintf(char *str, const char *format, ...);` e `int snprintf(char *str, size_t size, const char *format, ...);`

O manual do linux para a função gets() é "never use this function" pois é impossível saber de antemão a quantidade de dados. Esta função continua a ler e gravar caracteres a partir da memória apontada por s. Portanto, a sugestão é usar fgets. Existe um file stream especial chamado **stdin** que representa a entrada padrão (neste caso o teclado). Desta forma, usaremos fgets para ler do teclado os caracteres que precisarmos.

A função sprintf se comporta semelhantemente à printf convencional. A diferença é que a string formatada fica armazenada no ponteiro para a string receptora (indicado por str) no primeiro argumento da função. Portanto, se quisermos passar qualquer número para string formatada igual ao printf, segue o exemplo abaixo. O mesmo ocorre para a função snprintf, porém com a diferença de que snprintf vai escrever no máximo n bytes na string destino.

```
1  #include<stdio.h>
2
3  int main(){
4    float a = 0.5606;
5    char stringA[32];
6    int i;
7    sprintf(stringA, "%f", a);
8    puts(stringA);
9    snprintf(stringA, 6, "%f", a); //5 na verdade, pois o ultimo eh o \0
10   puts(stringA);
11   return 0;
12 }
```

3.3 Manipulação de strings

Na biblioteca padrão string.h temos o seguinte conjunto básico de funções:

- `size_t strlen(const char *s);` calcula o tamanho de uma string (em termos de caracteres) excluindo o `'\0'`.
- `char *strcpy(char *dest, const char *src);` e `char *strncpy(char *dest, const char *src, size_t n);` copia uma string (com especificação de tamanho ou não). Caso n seja maior que o tamanho de src, a função strncpy completa o restante com `'\0'`.
- `char *strcat(char *dest, const char *src);` e `char *strncat(char *dest, const char *src, size_t n);` concatena src em dest (substituindo o caractere `'\0'` de dest). A strncat concatena no máximo os n primeiros caracteres de src. As duas sempre adicionam o caractere nulo ao final.

- `int strcmp(const char *s1, const char *s2);` e `int strncmp(const char *s1, const char *s2, size_t n);` comparam duas strings ou no máximo n caracteres das mesmas. Retornam um inteiro menor que zero, igual a zero e maior que zero, se $s1 < s2$, $s1 = s2$ e $s1 > s2$, respectivamente.

- `char *strpbrk(const char *s, const char *accept);` Retorna um ponteiro par a primeira ocorrência na string s, de algum dos caracteres na string accept.

- `char *strchr(const char *s, int c);` e `char *strrchr(const char *s, int c);`retorna um ponteiro para a primeira (e respectivamente a última) ocorrência do caractere c na string.

- `char *strsep(char **stringp, const char *delim);` extrai o primeiro token em stringp que é delimitado por um dos caracteres em delim.

- `size_t strspn(const char *s, const char *accept);` calcula o tamanho do segmento inicial na string s que consiste inteiramente de caracteres em accept.

- `char *strstr(const char *haystack, const char *needle);` encontra a primeira ocorrência da substring need na string haystack. Retorna um ponteiro para a substring encontrada.

- `char *strtok(char *s, const char *delim);` extrai tokens da string s que são delimitadas por um dos caracteres em delim.

- `size_t strxfrm(char *dest, const char *src, size_t n);` transforma src para o locale atual e copia os primeiros caracteres para dest.

- `char *strlwr(char *string);` e `char *strupr(char *string);` passam a string para lowercase e uppercase. OBS: estas funções **não fazem parte do padrão** e podem não estar disponíveis na sua implementação do libc.

```c
#include <string.h>
char *strtok(char *str, const char *delim);
char *strtok_r(char *str, const char *delim, char **saveptr);
```

A primeira chamada a strtok deve ser passada a string. As chamadas subsequentes, se a mesma string está para ser processada, então deve-se passar NULL no lugar do primeiro argumento. Cada chamada retorna um ponteiro para uma string terminada com `'\0'`. Caso nenhum delimitador seja encontrado em qualquer ponto da chamada, o valor NULL é retornado.

```c
//Exemplo de tokenizacao de dois niveis.
/*
  Exemplo de uso:
  $ ./a.out 'a/bbb///cc;xxx:yyy:' ':;' '/'
  1: a/bbb///cc
  --> a
  --> bbb
  --> cc
  2: xxx
  --> xxx
  3: yyy
  --> yyy
*/
```

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

int main(int argc, char *argv[])
{
  char *str1, *str2, *token, *subtoken;
  char *saveptr1, *saveptr2;
  int j;

  if (argc != 4) {
    fprintf(stderr, "Usage: %s string delim subdelim\n",
    argv[0]);
    exit(EXIT_FAILURE);
  }

  for (j = 1, str1 = argv[1]; ; j++, str1 = NULL) {
    token = strtok_r(str1, argv[2], &saveptr1);
    if (token == NULL)
      break;
    printf("%d: %s\n", j, token);

    for (str2 = token; ; str2 = NULL) {
      subtoken = strtok_r(str2, argv[3], &saveptr2);
      if (subtoken == NULL)
        break;
      printf(" --> %s\n", subtoken);
    }
  }
  exit(EXIT_SUCCESS);
}
```

```c
#include <string.h>
char *strsep(char **stringp, const char *delim);
```

Se *stringp for NULL, strsep() retorna NULL e finaliza seu processamento. Caso contrário, est afunção encontra o primeiro token na string *stringp que é delimitada por um dos caracters da string delim. Este token é terminado por sobrescrever o delimitador com o byte null `'\0'` e *stringp é atualizada para uma posição depois do delimitador. Caso nenhum delimitador for encontrado o token se torna toda a string em *stringp e *stringp é igualada a NULL.

Cuidado

As funções `strtok, strtok_r e strsep` modificam a string passada como parâmetro. Portanto, esta string não pode ser constante.

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

int main()
{
  char *string,*found;
```

```
7
8   string = strdup("Hello there, peasants!");
9   printf("Original string: '%s'\n",string);
10
11  while( (found = strsep(&string," ")) != NULL )
12  printf("%s\n",found);
13
14  return(0);
15 }
```

Em strings.h encontramos:

1. `int strcasecmp(const char *s1, const char *s2);` e `int strncasecmp(const char *s1, const char *s2, size_t n);` Compara as strings s1 e s2 ignorando se está em caixa alta ou caixa baixa. Segue o mesmo padrão de retorno de strcmp.
2. `char *index(const char *s, int c);` e `char *rindex(const char *s, int c);` retornam um ponteiro para a primeira (e respectivamente a última) ocorrência) do caractere *c* na string s. NULL é retornado caso o caractere não se encontrar na string. O caractere `'\0'` também é considerado para ser encontrado neste caso.

As funções de conversão são:

1. `double atof(const char *str)` Converte a string apontada por str para um número double.
2. `int atoi(const char *str)` Converte a string apontada por str para um número inteiro.
3. `long int atol(const char *str)` Converte a string apontada por str para um número inteiro longo.
4. `double strtod(const char *str, char **endptr)` Converts the string pointed to, by the argument str to a floating-point number (type double).
5. `long int strtol(const char *str, char **endptr, int base)` Converts the string pointed to, by the argument str to a long integer (type long int).
6. `unsigned long int strtoul(const char *str, char **endptr, int base)` Converts the string pointed to, by the argument str to an unsigned long integer (type unsigned long int).

```
1 #include <stdlib.h>
2 double strtod(const char *nptr, char **endptr);
3 float strtof(const char *nptr, char **endptr);
4 long double strtold(const char *nptr, char **endptr);
```

Tentam converter a porção inicial da string apontada por nptr para double, float e long double, respectivamente.

A forma esperada da porção inicial da string pode conter espaços (reconhecidos por uma função chamada `isspace()` que serão desconsiderados na conversão), um sinal opcional de mais(+) ou menos(-) e o número decimal, hexadecimal, infitito ou NAN (not-a-number).

Infinito é descrito por INF ou INFINITY (em caixa alta ou baixa).

Retorno: Se o ponteiro endptr não for nulo (diferente de NULL), um ponteiro para o último caractere usado na conversão é armazenado em endptr.

Se a conversão não for executada, o valor zero é retornado e o valor de nptr é copiado para endptr.

Exercícios de Fixação

1. Faça um código em C que leia uma string e troque suas vogais para a ordem reversa em que aparecem. Ex.: se a string for hello world, a saída deverá ser hollo werld.

2. Faça um código em C que verifica se uma data string é palíndromo.

3. Escreva uma função que verifica se um dado vetor de string está em ordem lexicográfica.

4. Faça uma função que receba uma string representando um número natural em base binária (ou seja, apenas contendo os caracteres '1' e '0') e converta esta string para um número.

5. Escreva uma função que receba um número natural na base decimal e transforme-o em uma string binária (ou seja, apenas contendo os caracteres '1' e '0') que represente o número dado.

6. Escreva uma função que receba duas strings s e t e calcule quantas vezes t é substring de s.