Apêndice B. Cadeias de Caracteres (Strings)

Até agora, quando desejávamos indicar o número de elementos existentes em um vetor qualquer, utilizávamos uma variável inteira, digamos **tam**, para armazenar tal informação. Logo, quando era necessário verificar se um dado valor estava presente em um vetor executávamos um trecho de código semelhante ao descrito abaixo:

Exemplo

```
int vet[10], i, tam=10, valor;

i = 0;
while (i < tam && vet[i] != valor)
   i++;
if(i != tam)
   /* codigo executado se o valor e encontrado no vetor */
else
   /* codigo executado se o valor não e encontrado no vetor*/
```

Uma outra abordagem possível seria indicarmos o término do vetor lógico através da presença de um valor que não pertença ao domínio dos valores presentes no vetor. Seja, por exemplo, um vetor de inteiros que armazene as idades dos alunos de uma turma. A presença de um valor negativo poderia ser usada para indicar o fim do vetor de idades, já que não há sentido em uma idade negativa. Note que para essa abordagem funcionar seria necessário definir o vetor de idades com pelo menos um elemento a mais do que seria necessário para armazenar todas as idades dos alunos da turma. Logo, se quisermos verificar se uma dada idade existe em uma turma de no máximo 30 alunos, o trecho de código a ser executado será o seguinte:

Exemplo

```
int vetIdade[31],i,idade;

i = 0;
while(vetIdade[i] >= 0 && vetIdade[i] != idade)
   i++;
if(vetIdade[i] >= 0)
   /* codigo executado se o valor e encontrado no vetor */
else
  /* codigo executado se o valor não e encontrado no vetor*/
```

B.1 Cadeias de Caracteres (Strings) em C

A linguagem C não possui nenhum tipo primitivo que permita a manipulação de cadeias de caracteres (strings). Desta forma, uma cadeia de caracteres em C é implementada através do uso de um vetor de caracteres terminado pelo caractere **nulo**. O caractere **nulo** tem configuração binária **00000000** (o seu código ASCII é 0), e é representado pela seqüência **'\0'** ou pela macro **NULL**.

Para armazenarmos uma cadeia de caracteres de até 10 caracteres escrevemos o comando:

```
char str[11];
```

O vetor **str** pode ser inicializado de duas maneiras distintas:

1) listando os caracteres um a um, e acrescentando o caractere **nulo** no final.

```
char str[11]={'a','l','o',' ','m','u','n','d','o','\0'};
```

2) usando uma constante do tipo cadeia de caracteres. Neste caso, o compilador acrescenta o caractere **nulo** automaticamente.

```
char str[11]="alo mundo";
```

B.2 Lendo e Escrevendo Strings

A biblioteca padrão de E/S, **stdio.h**, fornece duas funções para E/S com strings. A função **gets()** lê uma cadeia de caracteres do teclado e o coloca **na cadeia passada como argumento.** Os caracteres digitados são transferidos para a memória após um **Enter**. O **Enter** não se torna parte do string; em seu lugar é colocado o caractere **nulo** (**'\0'**).

A função **puts()** escreve o seu argumento (uma cadeia de caracteres) na tela, seguida por um caractere de nova linha. A função **puts()** reconhece os mesmos caracteres de controle que a função **printf()** (por exemplo, **'\n'**). Uma chamada à função **puts()** requer bem menos tempo do que a mesma chamada à **printf()**, porque **puts()** pode escrever apenas strings de caracteres, não podendo escrever números ou fazer conversões de formato. Portanto, **puts()** ocupa menos espaço na memória e é executada mais rapidamente do que **printf()**. O comando a seguir exibe a frase **Alo Mundo!** no monitor.

```
puts("Alo Mundo!");
```

A função **scanf()** pode ser utilizada para ler uma cadeia de caracteres do teclado usando o especificador de formato **%s**. O **%s** faz com que **scanf()** leia caracteres até que seja encontrado **um caractere de espaço em branco**. Os caracteres lidos são colocados em um vetor de caracteres apontado pelo argumento correspondente, e o resultado tem terminação nula **('\0')**. Para **scanf()**, um caractere de espaço em branco é um espaço, um retorno de carro (**Enter**), ou uma tabulação. Logo, uma cadeia de caracteres como **Alo Mundo!** não pode ser lida com **scanf()** (apenas o substring **Alo** será carregado no vetor).

A função **printf()** escreve uma string através do **%s**. Ao contrário da função **puts()**, não existe mudança automática de linha na exibição de uma cadeia de caracteres no vídeo.

Podemos também ler uma cadeia de caracteres, caractere a caractere, através das funções **getchar()**, **getch()** ou **getche() (para uso das 2 últimas funções, é necessário especificar #include <conio.h>)**. Neste caso, é função do programador

acrescentar um caractere **'\0'** após a string. O código a seguir exemplifica o que foi dito acima.

Exemplo

```
#include <stdio.h>

void main(void)
{
 char nome[81];
 int i=0;

 printf("Informe um Nome\n");
 nome[i]=getchar();
 while( i < 80 && nome[i] != '\n')
 {
  i++;
  nome[i] = getchar();
 }
 nome[i] = '\0';
 puts(nome);
}
```

É importante lembrar que as funções **gets()** e **scanf()** recebem como parâmetro um **ponteiro** para uma área de memória. Logo, não temos controle sobre a quantidade de caracteres que será digitada pelos usuários, e sobre a área de memória que será utilizada para armazenar os caracteres lidos; já que o tamanho do vetor é estabelecido estaticamente.

B.3 Algumas Funções de Strings

A biblioteca **string.h**, fornece uma gama variada de funções para a manipulação de strings. As mais comuns são fornecidas pela maioria dos compiladores C. São elas:

B.3.1 strlen()

A função **strlen()** retorna o número de caracteres em uma cadeia de caracteres (o caractere `'\0'` não é contado). O cabeçalho da função **strlen()** é o seguinte:

```
int strlen(char str[]);
```

Exemplo

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

void main(void)
{
  char nome[]="Joao da Silva";
  printf("%s contem %d caracteres",nome,strlen(nome));
}
```

Ao ser executado, este programa exibirá no monitor o texto:

Joao da Silva contem 13 caracteres

B.3.2 strcpy()

A função **strcpy()** é usada para copiar o conteúdo de uma cadeia de caracteres (fonte) para outra string (destino). Essa função recebe os seguintes parâmetros:

```
strcpy(char destino[],char fonte[]);
```

Exemplo

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

void main(void)
{
  char nome[]="Joao da Silva",aluno[50];
  strcpy(aluno,nome);
  printf("O nome do aluno e %s",aluno);
}
```

Ao ser executado, este programa exibirá no monitor o texto:

O nome do aluno e Joao da Silva

B.3.3 strcmp()

A função **strcmp()** é usada para comparar lexicograficamente duas strings. O cabeçalho da função **strcmp()** é o seguinte:

```
int strcmp(char str1[],char str2[]);
```

Se as strings forem iguais, isto é, se **str1** e **str2** forem do mesmo comprimento e possuírem caracteres iguais nos elementos de mesmo índice, **strcmp()** retornará **zero**. Se **str1** for lexicograficamente maior do que **str2**, a função **strcmp()** retornará um valor maior do que **zero**. Se **str2** for lexicograficamente maior do que **str1**, a função **strcmp()** retornará um valor menor do que **zero**.

Exemplo

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

void main(void)
{
  char nome1[]="Joao da Silva",nome2[]="Maria Fernanda";

  if(!strcmp(nome1,nome2))
   printf("%s e igual a %s",nome1,nome2);
```

```
  else
   if(strcmp(nome1,nome2)>0)
    printf("%s vem depois de %s",nome1,nome2);
   else
    printf("%s vem depois de %s",nome2,nome1);
}
```

Ao ser executado, este programa exibirá no monitor o texto:

Maria Fernanda vem depois de Joao da Silva

B.3.4 strcat()

A função **strcat()** anexa o conteúdo de uma cadeia de caracteres (fonte) no final de outra string (destino). Este processo é chamado de concatenação de strings. É importante lembrar que o tamanho físico da string de destino ter que ser suficiente para armazenar os caracteres das duas strings.

O primeiro caractere da string de origem é copiado para a posição da string de destino que contém o caractere **NULL**. O processo é repetido até que o último caractere da string de origem seja copiado para a string de destino. A string de destino terá apenas um caractere **NULL** no final do processo. Essa função recebe os seguintes parâmetros:

```
strcat(char destino[],char origem[]);
```

Exemplo

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

void main(void)
{
  char nome[30]="Joao",sobreNome[]=" da Silva";

  printf("O nome do aluno e %s",strcat(nome,sobreNome));
}
```

Ao ser executado, este programa exibirá no monitor o texto:

O nome do aluno e Joao da Silva