Capítulo 5

Cadeias de Caracteres

"As primeiras disciplinas de introdução à ciência de computação, são as mais importantes uma vez que representam a oportunidade para a maioria dos estudantes tomar conhecimento dos primeiros conceitos de computação e de programação, tais como, algoritmos, estrutura de dados e assim por diante"

- Anômino-

Sumário:

5.1- Caracteres

Na linguagem C, os caracteres são representados pelo tipo **char**, que podem armazenar valores inteiros: um **char** ocupa um byte de memória, que é igual a 8 bits e pode representar 256 valores distintos. Como os códigos associados aos caracteres estão dentro deste intervalo, usamos o tipo **char** para representar caracteres.

Na linguagem C, a diferença em caracteres e números inteiros é feita apenas pela forma como esses dados são tratados. Por exemplo, podemos imprimir o mesmo valor de duas formas diferentes, utilizando para esse efeito dois formatos diferentes. O segmento de código:

```
char c = 97;
printf ("\n %d   %c " , c , c );
```

Se tivermos em conta a tabela ASCII, a variável c, que foi inicializada com o valor 97, representa o caracter C. Mas a função **printf()**, mostra na tela o conteúdo

dessa variável, utilizando dois formatos. Com a mascara %d, será impresso o número 97, mas com a mascara %c, será impresso a letra a.

Se prestarmos atenção a tabela ASCII, observamos que os códigos dos dígitos estão agrupados em ordem sequencial. Se o dígito 0 for representado pelo código 48, então o dígito 1 será obrigatoriamente representado pelo código 49. O mesmo acontece com os caracteres do alfabeto. As letras maiúsculas e a lestras minúsculas, estão organizadas em dois grupos distintos. Podemos tirar partido dessa codificação escrever os nossos programas.

5.1.1- Verificar se um Carácter é uma Letra Maiúscula

Pela tabela ASCII, as letras maiúsculas são representadas pelo conjunto de números inteiros que vai de 65 até 90, sendo 'A' = 65 e 'Z'= 90.

Então, esta função recebe como argumento um caracter e retornar 1 se esse carácter for uma letra maiúscula e 0 no caso contrário.

```
int ehMaiuscula ( char c)
{
   if ( (c >= "A") && (C <= "Z") )    return 1;
   return 0;
}
```

5.1.2- Converter uma Letra Minúscula para Maiúscula

Vimos pelo exemplo anterior, que na tabela ASCII, as letras maiúsculas são representadas pelo conjunto de números inteiros que vão de 65 até 90, sendo 'A' = 65 e 'Z'= 90, enquanto as enquanto as letras minúsculas são representadas pelo conjunto dos números inteiros que vai de 97 até 122.

Então podemos calcular a distancia que vai de qualquer minúscula para a correspondente letra maiúscula. Seja c uma variável que representa uma letra minúscula. Então c -'a' representa a distância entre essa letra minúscula para a correspondente letra maiúscula. Isso quer dizer que se somarmos essa distância a letra c teremos a letra maiúscula correspondente.

```
char converteMaiuscula (char c)
 {
    if ( (c >= 'a') &&  (c <= 'z') )   c + = 'A'- 'a';
    return c;
 }
```

Uma outra forma de implementar essa função, consiste em utilizar a função **toupper()** que está definida O arquivo-cabeçalho **<ctype.h>** da biblioteca ANSI.

```
char converteMaiuscula (char c)
{
   if ( (c >= 'a') &&  (c <= 'z') )   c  = toupper (c) ;
   return c;
}
```

5.2 - Cadeias de Caracteres

Cadeias de caracteres (**Strings**), em C, são representadas por vectores do tipo **char** que terminam, obrigatoriamente, um código especial a delimitar o seu término. Estes vectores possuem a seguinte sintaxe:

```
char nome[tamanho];
```

O término de qualquer cadeia de caracteres (strings), é identificado por um caracter especial, denominado por **NULL**, e representado pelo símbolo '\0' da tabela ASCII.

Se pretendermos armazenar a palavra UCANIANO, como uma cadeia de caracteres, num vector com no máximo 16 elementos, esse vector terá a seguinte representação:

U	C	A	N	I	A	N	O	'\0'								

↑

Delimitador da String

Para armazenar uma cadeia de caracteres, devemos declarar um vector do tipo **char**, com uma posição adicional, para armazenar o delimitador de fim de cadeia.

Exemplo 1: A declaração,

```
char cidade[10] = { 'L' , 'u' , 'a', 'n', 'd', 'a' };
```

representa um vector com dez elementos do tipo **char**, que contêm nos seis primeiros elementos a palavra *Luanda*. Na posição seguinte, o compilador armazena o delimitador de fim de cadeia de caracteres.

Exemplo 2: A declaração

```
char vogais[5] = {'a' , 'e', 'i', 'o', 'u'};
```

representa um vector com cinco elementos do tipo **char**. Cada elemento é uma vogal. Como o alfabeto possui apenas cinco vogais, este vector fica completamente preenchido e o compilador C não inclui o caracter nulo. Todo o conjunto de caracteres sem o delimitador **NULL** não é uma cadeia de caracteres.

Exemplo 3: A declaração,

```
char st[ ] = "";
```

representa um vector com um único elemento do tipo caracter. Nesse elemento o compilador C armazena o caracter **NULL**.

Dizemos que uma cadeia de caracteres está vazia quando o primeiro elemento do vector contêm o caracter **NULL**.

5.3 - Comandos de Leitura

Para ler um carácter a partir do teclado, podemos utilizar a função **scanf(),** com o formato **%c**.

```
char  letra;
scanf (" %c". & letra);
```

Se o utilizador digitar um carácter, por exemplo a letra m, o código associado a essa letra, será armazenado na variável letra. Mas esse formato lê os espaços em branco.

Se quisermos pular todos os espaços em branco que antecedem um determinado carácter, basta incluir um espaço em branco antes desse formato.

```
char  letra;
scanf ("  %c", & letra);
```

Para ler uma cadeia de caracteres na linguagem C podemos utilizar a função **scanf()** com o formato **%s**. Mas esse formato %s deixa de ler a cadeia de caracteres quando deteta o primeiro carácter em branco ou o carácter que representa a tecla [ENTER]. Nessa altura essa função armazena o delimitador de fim de uma cadeia de caracteres. Consideremos o seguinte segmento de código:

```
char nome[TAMMAX];
printf (" \n Digite o seu nome:");
scanf ("%s", nome);
```

Se o utilizador digitar a palavra kuando kudango, por exemplo, apenas a palavra kuando será armazenada, porque o formato **%s** deixa de capturar os dados quando for digitado o primeiro branco.

*O formato **"%s"** interpreta o carácter em branco como o término do texto que pretendemos armazenar.*

Para nomes completos como por exemplo, nome de uma pessoa, nome de uma província, endereços de uma rua, etc, utiliza-se em geral, a função **scanf()** com

%[^\n]. Este formato que faz com que o programa leia uma cadeia de caracteres, até que o utilizador pressione a tecla [ENTER]. Para o exemplo anterior teríamos a Vamos estudar em seguida, uma série de operações elementares sobre string's. Desenvolveremos essas operações com base no vector st com MAX elementos.

5.4 - Operações Iterativas com Cadeia de Caracteres

Vamos estudar em seguida, uma série de operações elementares sobre cadeia de caracteres, mas desenvolveremos essas operações com base no vector st com MAX elementos.

5.4.1- Imprimir os Elementos de uma Cadeia de Caracteres

Imprimir os elementos de uma cadeia de caracteres, consiste em percorrer a cadeia até ao carácter **NULL**, e mostrar na tela o conteúdo de cada elemento percorrido na tela.

```
void  imprimir (char st[ ])
 {
   int i;
   for ( i = 0 ; st[i] != '\0' ;  i++  )  printf (" %c ", st[i]);
   printf ("\n");
 }
```

Vamos escrever um procedimento equivalente com o formato %s.

```
void  imprimir (char st[ ] )
 {
    printf ("%s", st);
    printf ( "%n");
 }
```

5.4.2- Determinar o Número de Elementos de uma Caracteres de Caracteres

Determinar o número de elementos de uma cadeia de caracteres, consiste em adicionar a uma variável auxiliar o número de posições do vector que armazena essa cadeia sem ter em conta o carácter de que determina o término dessa cadeia.

```
int  comprimento (char st[ ] )
 {
   int  i, n = 0;
   for ( i = 0 ; st[i] != '\0' ; i++)  n++;
   return n;
 }
```

Podemos reescrever esta função de forma mais compacta. Nesta nova versão, vamos aproveitar o funcionamento da variável que percorre a cadeia de caracteres, para contar o número dos seus elementos. Observe que a variável que conta o número de elementos da cadeia de caracteres, tem um funcionamento análogo a variável que percorre todos os seus elementos, logo pode ser desprezada.

```
int  comprimento (char st[ ] )
 {
    int  i;
    for ( i = 0 ; st[i] != '\0' ; i++);
    return i;
 }
```

A biblioteca ANSI, possui uma função denominada por “string lenght” que recebe como parâmetro uma cadeia de caracteres e retorna o seu número de caracteres. Esta função possui a seguinte sintaxe **int strlen** (**char** *st);

Agora, podemos utilizar esta função, para reescrever um procedimento que irá imprimir na tela, uma cadeia de caracteres com o formato **%c**.

```
void  imprimir (char st[ ] )
{
  int i, k = comprimento (st);
  for (i = 0 ; i < k ;  i++) printf ("  %c ", st[i]);
  printf ("\n");
}
```

Quando utilizamos funções desenvolvidas por nós ou por terceiros, incluindo as funções da biblioteca padrão, devemos ter muito cuidado com a eficiência dos algoritmos.

Por exemplo, a função **imprimir()** que mostramos em seguida,

```
void  imprimir (char st[ ] )
{
   int i;
   for (i = 0 ; i < comprimento (st) ;  i++) printf ("  %c ", st[i]);
  printf ("\n");
}
```

Embora seja muito compacta, não é eficiente. A falta ineficiência deve-se ao facto de invocarmos a função **tamanho()** dentre de um ciclo. Suponhamos sem perda da generalidade que a cadeia possui 10 elementos. Na função anterior esse ciclo é percorrido 10 vezes, mas nesta função esse ciclo é percorrido 100 vezes. Observe que em cada iteração a função **tamanho()** é invocada, e ao ser invocada, essa função percorre outra vez essa cadeia. Logo o ciclo dessa função percorre essa cadeia 10x10 vezes. Isso é uma prova inequívoca que este algoritmo não serve.

5.4.3- Copiar os Elementos de uma Cadeia de Caracteres

Copiar os elementos de uma cadeia de caracteres consiste em armazenar os seus elementos, incluindo o carácter '\0' numa outra cadeia. Vamos assumir que a cadeia destino, denominada por st2, possui um espaço suficiente para realizar essa cópia. Seja st1 a cadeia origem.
A estratégia para implementar esta função é simples. Ela consiste em percorrer os elementos da cadeia st1 até ao carácter '\0'. Durante o processo de percurso, copiar cada elemento de st1 para st2.

```
void  copiar (char st1[ ] , char st2[ ] )
 {
   int i;
   for (i = 0; st1[i] != '\0' ; i++)  st2[i] = st1[i];
   st2[i] = '\0';
 }
```

Apresentamos em seguida, uma versão compactada da função copiar(), que utiliza a mesma estratégia.

```
 void copiar (char st1[ ],char st2[ ])
  {
    int i = 0;
    while ((st2[i] = st1[i]) != '\0') i++;
  }
```

A biblioteca ANSI possui uma função

A biblioteca ANSI, possui uma função denominada por "String Copy" que tem por finalidade copiar a string origem para a string destino. Esta função possui a seguinte sintaxe: **char ***strcpy (**char** *dest, **char** *orig);

Podemos utilizar este procedimento para desenvolver um subprograma que recebe o número do mês e devolve o mês por extenso.

```
void mesExtenso (int mes , char nome[ ] )
{
  switch (mes)
    {
      case 1:
              copiar (nome, "Janeiro");
              break;
      case 2:
              copiar (nome, "Fevereiro");
               break;
      case 3:
              copiar (nome, "Marco");
```

```
            break;
        case 4:
            copiar (nome, "Abril");
            break;
        case 5:
            copiar (nome, "Maio");
            break;
        case 6:
            copiar (nome, "Junho");
            break;
        case 7:
            copiar (nome, "Julho");
            break;
        case 8:
            copiar (nome, "Agosto");
            break;
        case 9:
            copiar (nome, "Setembro");
            break;
        case 10:
            copiar (nome, "Outubro");
            break;
        case 11:
            copiar (nome, "Novembro");
            break;
        case 12:
            copiar (nome, "Dezembro");
            break;
    }
}
```

5.4.4- Concatenar duas Cadeias de Caracteres

Concatenar duas cadeias de caracteres, consiste em copiar os caracteres de uma cadeia para o final da outra. Por exemplo, se uma cadeia contém a palavra *Ana* e a concatenarmos com a palavra *Bela,* teremos como resultado a palavra AnaBela.

A estratégia para implementação desta operação, consiste em percorrer em primeiro lugar a cadeia destino (st2) até ao fim, em seguida, copiar cada elemento da cadeia origem (st1) para a cadeira st2.

```
void  concatenar (char st2[ ] , char st1[ ] )
{
  int i, j;
  for (i = 0 ; st2[i] != '\0' ; i++);
  for (j = 0 ; st1[j] != '\0' ; j++)
   {
```

```
        st2[i] = st1[j];
        i++;
    }
   st2[i] = '\0';
}
```

Vamos apresentar em seguida, uma versão compacta dessa função.

```
void concatenar (char st2[ ] , char st1[ ] )
{
  int i = 0 , j = 0;
   while (st2[i]!='\0')  i++;
   while (( st2[i++] = st1[j++] ) !=  '\0');
}
```

A biblioteca ANSI, possui uma função denominada por "String Concat" que tem por finalidade colocar a string origem imediatamente a seguir da string destino e devolver em seguida a string destino. Esta função possui a seguinte sintaxe: **char ***strcat (**char** *dest, **char** *orig);

É oportuno salientar que antes de concatenarmos duas cadeias de caracteres, necessitamos de verificar se pelo menos uma dessas cadeias não está vazia, e se o vector destino (st2) possui espaço suficiente para realizar essa operação.

5.4.5- Comparar duas Cadeias de Caracteres

Comparar duas cadeias de caracteres, consiste em analisar a relação que existe entre os elementos que ocupam a mesma posição no vector, ou seja: se numa determinada posição, o elemento da cadeia st1 for alfabeticamente menor do que o elemento da cadeia st2, retornar um número inteiro menor do que zero; Se o elemento da cadeia st1 for alfabeticamente maior do que o elemento da cadeia st2, retornar um número inteiro maior do que zero; se todos os elementos de st1 forem alfabeticamente iguais ao elemento de st2 retornar um número igual a zero.
Então, a estratégia para implementa-la função consiste em percorrer as cadeias de caracteres e comparar os elementos que ocupam a mesma posição relativa.

```
int comparar (char st1[ ] , char st2[ ])
{
   int i;
   for (i = 0; st1[i] != '\0'; i++)
   if (st1[i] < st2[i])    return -1;
    if ( st1[i] > st2[i] )  return 1;
    return 0;
}
```

Mostraremos a seguir, uma outra função, mais difícil de ler e entender, para comparar duas cadeias de caracteres.

```
int comparar (char st1[ ] , char st2[ ])
{
   int i = 0;
   while (st1[i] != 0 && st2[i] != 0)
       if (st1[i] != st2[i])
           if (st1[i] > st2[i]) return 1;
           else  return -1;
   if (st1[i] == st2[i])  return 0;
   else if (st1[i] > st2[i])  return 1;
        else  return -1;
  }
```

Devemos salientar que esta operação só pode ser realizada se as cadeias st1 e st2 possuírem o mesmo número de elementos. Como na operação anterior, a verificação dessa condição, deve ser feita uma função que irá invocar esta função quando a condição de igualdade for verdadeira.

Veremos agora uma variante para essa operação. A função que descrevemos em seguida, devolve zeros se st1 = st2; um número negativo se st1 < st2 e um número positivo se st1 > st2.

```
int comparar (char st1[ ] , char st2[ ])
{
   int i;
   for ( i = 0; st1[i] == st2[i] ;  i ++)
   if (st1[i] == '\0')  return 0;
   return st1[i] – st2[i];
 }
```

que na forma compacta pode ser descrito por:

```
int comparar (char st1[ ] , char st2[ ])
 {
   int i;
   for (i = 0; s[i] == t[i]; i++)
       if (s[i]=='\0')  return 0;
    return s[i] - t[i];
  }
```

A biblioteca ANSI, possui uma função denominada por "String Compare" que tem por finalidade comparar alfabeticamente a string st1 com a st2. Esta função possui a seguinte sintaxe: **int** strcmp (**char** *st1, **char** *st2);

5.5 - Operações Recursivas com Cadeias de Caracteres

Por definição, uma cadeia de caracteres pode ser representada por:

a) Uma cadeia de caracteres vazia; ou
b) Um carácter seguido de uma subcadeia de caracteres.

Em função desta definição recursiva, vamos desenvolver algumas operações implementadas anteriormente.

5.5.1- Imprimir uma Cadeia de Caracteres, Elemento por Elemento

```
void imprimirRec ( char st[ ] , int i )
 {
   if (st[i] != '\0')
    {
      printf ("%c", st[i]);
      imprimirRec (st , i + 1);
    }
 }
```

Está função só funciona de forma correcta, se estiverem garantidas as condições iniciais para a recursão (base da recursão). Como o processo recursivo termina quando o índice de recursão apontar para o elemento que contêm o delimitador de fim de cadeia de caracteres, então no início da recursão esse índice deve apontar para a primeira posição do vector. Para além disso, é necessário garantir que o vector contenha pelo menos um elemento.

5.5.2- Determinar o número de Elementos de Cadeia de Caracteres

Este algoritmo é muito simples e consiste em:

```
int comprimentoRec ( char st[ ], int i )
{
  if (st[i] != '\0') return 0;
  return 1 + comprimentoRec (st, i +1);
}
```

Mais uma vez salientamos que é necessário garantir as condições iniciais de recursão e que essas condições são programadas numa função que irá invocar a função recursiva.

5.5.3- Copiar os Elementos de uma Cadeia de Caracteres

```
void copiarRec ( char st1[ ] , char st2[ ] , int i )
 {
    if (orig[i] != '\0')  st2[i] = st1[i];
```

```
    else
      {
         st2[i] = st1[i];
         copiarRec (st1 , st2 ,  i +1);
      }
  }
```

Como em ambos os membros do comando condicional temos a atribuição do elemento da cadeia de caracteres origem (st1) para a cadeia de caracteres destino (st2), podemos compactar este código com a seguinte função:

```
void copiarRec ( char st1[ ] , char st2[ ] , int i , int t)
{
    st2[i] = st1[t];
    if (orig[i] != '\0')    copiarRec (st1 , st2 ,  i +1, t +1);
 }
```

5.5.4- Comparar duas Cadeias de Caracteres

```
int compararRec (char st1[ ] , char st2 [ ] , int i )
 {
    if ((st1[i]  == '\0') || ( st1[i] != st2[i] ))   return st1[i] – st2[i];
    return compararRec (st1, st2, i + 1);
 }
```

5.6 - Recomendações Bibliográficas

Para o leitor aprofundar os seus conhecimentos sobre caracteres e cadeia de caracteres, recomendamos os seguintes livros:

K. N. King. - *C Programming – A Modern Approach*, W. W. Norton & Company, Inc., 2nd edition, 2008.

Celes W., Cerqueira R., Rangel J. L., - *Introdução a Estruturas de Dados com Técnicas de Programação em C*, 2ª Edição, Editora Elsevier, Brasil, 2016

5.7 - Exercícios

5.7.1-Desenvolva uma função iterativa e uma função recursiva, para contar o número de vogais existentes numa cadeia de caracteres, passada como parâmetro.

5.7.2-Desenvolva uma função iterativa e uma função recursiva, que converte todos as letras minúsculas em letras minúsculas numa cadeia e de caracteres passada como parâmetro.

5.7.3-Desenvolva uma função iterativa e uma função recursiva, para encontrar a primeira ocorrência de um determinado carácter numa cadeia de caracteres, passada como parâmetro. Essa função deve retornar à posição relativa dessa ocorrência, ou -1 se ela não estiver nessa cadeia.

5.7.4-Desenvolva uma função iterativa e uma função recursiva, para apagar de uma cadeia de caracteres passada como parâmetro, todas as letras minúsculas.

5.7.5-Desenvolva uma função iterativa e uma função recursiva, para substituir numa cadeia de caracteres passada como parâmetro, todas as letras do alfabeto pelo seu oposto, ou seja, a recebe z, b recebe x, etc.

5.7.6-Desenvolva uma função iterativa e uma função recursiva, para deslocar todos os caracteres de uma cadeia de caracteres para uma posição à direita e deslocar o último carácter dessa cadeia para a primeira posição do vector.

5.7.7-Desenvolva uma função iterativa e uma função recursiva, para concatenar os n primeiros elementos da cadeia de caracteres st1 na cadeia de caracteres st2.

5.7.8-Desenvolva uma função iterativa e uma função recursiva, para comparar os n primeiros dígitos das duas cadeias e retornar: devolver um número menor do que zero se a primeira cadeia é menor do que a segunda; 0 se as cdeias forem iguais; e um número positivo se a primeira cadeia é maior do que a segunda.

5.7.9-Desenvolva uma função iterativa, para verificar se uma cadeia está contida na outra. Essa função deve retornar 1 se a condição for verdadeira e 0 no caso contrário.

5.7.10-Desenvolva uma função iterativa e uma função recursiva, para eliminar todos os caracteres de branco de uma cadeia de caracteres e devolver essa cadeia.

5.7.11-Desenvolva uma função iterativa e uma função recursiva, para verificar se duas cadeias de caracteres passadas como parâmetros, são opostas, ou seja, os elementos de uma estão na ordem inversa de outra.

5.7.12-Desenvolva uma função iterativa para remover todos os elementos repetidos de uma cadeia de caracteres e devolver essa cadeia.s

5.7.13-Desenvolva uma função iterativa e uma função recursiva, para verificar se uma cadeia de caracteres é do tipo capicua. Por exemplo, as palavras osso, ovo são do tipo capicua.