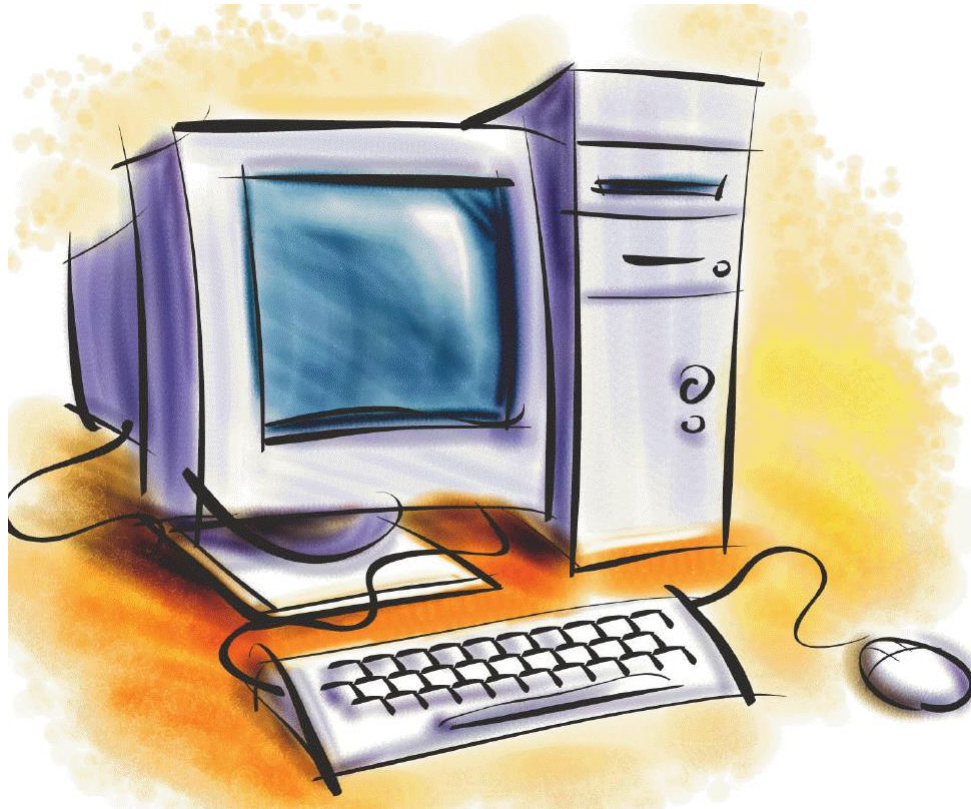


ALGORITMOS E ESTRUTURAS DE DADOS I



UNIDADE 5

ARQUIVOS E STRINGS

PROF. NAÍSSSES ZÓIA LIMA

BASEADO NO MATERIAL DA PROFA. ROSILANE MOTA

Strings

String

Strings (cadeias de caracteres) são muito utilizadas para guardar:

- nomes de arquivos
- nomes de usuários
- qualquer informação baseada em caracteres.

A linguagem C utiliza **vetores** de char para armazenar uma cadeia de caracteres, onde cada posição representa um caractere

A diferença básica entre strings e outros vetores é que a linguagem de programação C indica o fim do vetor de strings através do acréscimo do caractere NULL ('\0') no final do String.

Deve-se declarar sempre o vetor com uma posição a mais para armazenar o caractere nulo ('\0'), que não precisa ser armazenado manualmente. Isso é feito automaticamente pelo compilador.

String

Para armazenar a palavra CADEIA deve-se declarar um vetor do tipo char com 7 posições (que ocuparão posições contíguas na memória)

char palavra[7]

índice	0	1	2	3	4	5	6	...
valor	C	A	D	E	I	A	\0	...
Posição Memória	863	864	865	866	867	868	869	...

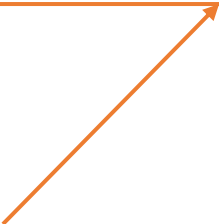
A variável palavra, quando é declarada pode ocupar qualquer posição na memória.

Entretanto, todas as posições do vetor ocupam espaços de memória adjacentes, sendo que cada caractere ocupa 1 byte.

Inicialização

```
char nome1[ ]= {'P', 'r', 'o', 'g', 'r', 'a', 'm', 'a', '\0'};
```

Inicialização no momento da declaração



A variável **nome1** recebeu as letras separadamente (inclusive o caractere nulo). Por isso, cada uma das letras está envolvida por apóstrofes (' '). Essa é a maneira de identificar um caractere isoladamente.

```
char nome2[ ]= "Programa";
```

A variável **nome2** recebeu uma palavra, recebendo automaticamente o caractere nulo. Por isso, a palavra Programa está entre aspas (" "). Esta é a maneira de identificar uma cadeia de caracteres.

Inicialização

Inicialização por meio da atribuição
(depois da declaração)

```
char vet1[10], vet2[5];  
strcpy(vet1, "Programa");
```

A variável vet1 recebeu um valor constante (a palavra Programa).

Usando duas strings, uma será copiada na outra. Ou seja, o conteúdo da variável vet2 foi copiado na variável vet1.

```
char vet1[10], vet2[5];  
strcpy(vet1, vet2);
```

Inicialização

```
char frase[100];  
printf("Digite um texto: ");  
scanf("%s", frase);
```

Inicialização por meio do teclado

O comando scanf consegue armazenar valores vindos do teclado na variável frase. No caso de uma cadeia de caracteres , esse comando consegue armazenar todos os símbolos digitados até a primeira ocorrência do espaço em branco OU <ENTER>.

Solução: ditar conjunto de código de conversão %[..] que pode ser usado para ler uma linha até o ENTER, contendo uma variedade de caracteres, incluindo espaços em branco.

```
char str[20];  
printf("Digite uma string:");  
scanf("%[^\n]", str);  
printf("%s", str);
```

Inicialização

A função `gets()` armazena na variável frase todos os símbolos digitados até a ocorrência do ENTER.

Esta função exige a utilização da biblioteca `stdio.h`.

```
char frase[100];  
  
printf("Digite um texto: ");  
  
gets(frase);
```

Exibição

A função `puts()` é usada para imprimir uma cadeia de caracteres inicializada com o uso da função `gets()`.

```
char frase[100];  
  
printf("\n Digite um texto: ");  
  
gets(frase);  
  
printf("\n A frase digitada foi: ");  
  
puts(frase);
```

Inicialização

O uso da função `gets()` não é seguro. Recomenda-se a utilização da função `fgets()`, que limita a quantidade de caracteres a serem lidos.

```
char frase[100];  
printf("Digite um texto: ");  
fgets(frase, 100, stdin);
```

Exibição

Também é possível imprimir a cadeia de caracteres com o comando printf, nesse caso, vamos precisar usar o %s (string)

```
char frase[100];  
printf("\n Digite um texto: ");  
gets(frase);  
printf("\n A frase digitada foi: %s", frase);
```

Funções para manipulação

Como todas as cadeias de caracteres são variáveis compostas homogêneas (vetor ou matriz) deve-se utilizar funções específicas.

Essas funções fazem parte da biblioteca string.h

Algumas delas:

- strlen()
- strcpy()
- strcat()
- strcmp()
- strupr()
- strlwr()

Faça uma pesquisa sobre as funções disponíveis!

Funções

A função `strlen` retorna para a variável `Tamanho` o número de caracteres da cadeia `str1`.

```
int tamanho;  
  
char str1[20] = "abc";  
  
tamanho = strlen(str1);
```

tamanho é 3

A função `strcpy` copia a cadeia `str2` na cadeia `str1`, inclusive o `'\0'` (NULL). Sendo assim, a cadeia `str1` será substituída pela cadeia `str2`.

```
strcpy (str1, str2);
```

Funções

A função `strncpy` copia os `n` primeiros caracteres da cadeia `str2` para a cadeia `str1` sem incluir o `'\0'` (NULL).

```
strncpy (str1, str2, n);
```

A função `strcat` concatena a cadeia `str2` na cadeia `str1`, ou seja, acrescenta a cadeia `str2` à cadeia `str1`.

```
strcat(str1, str2);
```

Funções

Compara duas cadeias de caracteres e retorna um número inteiro para a variável Resultado, que pode ser:

- Zero: se as duas cadeias forem iguais
- Um número menor que 0: se a cadeia cadeia1 for alfabeticamente menor que cadeia2
- Um número maior que 0: se a cadeia cadeia1 for alfabeticamente maior que cadeia2

Essa função strcmp considera letras maiúsculas como sendo símbolos diferentes de letras minúsculas.

```
resultado = strcmp (cadeia1, cadeia2);
```

Essa função stricmp considera letras maiúsculas e minúsculas como sendo símbolos iguais.

```
resultado = stricmp (cadeia1, cadeia2);
```

Outras Funções (string.h)

- `toupper(caracter)` – converte o caracter em maiúsculo.
- `strupr(string)` – converte a string para maiúsculo
- `tolower(caracter)` – converte o caracter em minúsculo.
- `strlwr(string)` – converte a string para minúsculo

Outras Funções (ctype.h)

A função `toascii` retorna para a variável “valor” o valor numérico que representa o caractere na tabela ASCII. Essa função exige a utilização da biblioteca `ctype.h`.

`valor = toascii(caractere);`

Outras Funções (string.h)

Separa uma string em *substrings*.

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

int main ()
{
    char str[] = "- This, a sample string.";
    char * pch;
    printf ("Splitting string \"%s\" into tokens:\n",str);
    pch = strtok (str," ,.-");
    while (pch != NULL)
    {
        printf ("%s\n",pch);
        pch = strtok (NULL, " ,.-");
    }
    return 0;
}
```

Exemplos

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

int main()
{
    char nome1[100], nome2[100];
    printf("Digite seu nome completo: ");
    scanf(" %s", nome1);
    printf("Digite seu nome completo: ");
    gets(nome2);
    gets(nome2); //usado duas vezes para resolver o
    problema de pular essa entrada de dados
    printf("\n*Resultado com PRINTF*****");
    printf("\nNome1: %s ", nome1);
    printf("\nNome2: %s ", nome2);
    printf("\n\n* Resultado com PUTS*****");
    printf("\n");
    puts(nome1);
    printf("\n");
    puts(nome2);
    return 0;
}
```

Exemplos

```
int main()
{
    char nome[40] = "Jose",
        sobrenome[30] = "Maria";
    strcat(nome, sobrenome);
    printf(" Sobrenome %s", sobrenome);
    printf("\n Nome %s", nome);
    return 0;
}
```


Exemplos

```
int main ()
{
    char nome[40] = "Jose",
          sobrenome[30] = "Jose";
    int teste;
    teste = strcmp (nome, sobrenome);
    if (teste != 0)
    {
        printf("As strings sao diferentes");
    }
    else
        printf("As strings sao identicas");
    return 0;
}
```

Exemplos

```
int main()
{
    char letra, maiuscula, minuscula;
    printf("\n\nDigite uma letra: ");
    scanf("%c",&letra);
    //toupper transforma em maiuscula
    maiuscula = toupper(letra);
    printf("\nMaiuscula: %c",maiuscula);
    //tolower transforma em minuscula
    minuscula = tolower(letra);
    printf("\nMinuscula: %c",minuscula);
    return 0;
}
```

Arquivos

Conceitos Básicos

Conjuntos de informações mantidas no disco (memória secundária)

Informações são “persistidas” em comparação com a memória RAM (memória primária) que eram “temporárias”

Sistema Operacional (OS) faz um “buffer” das informações lidas/gravadas

Em C, arquivos podem ser gravados de duas maneiras:

- Modo texto (conjunto de caracteres)
Arquivo texto pode ser tratado por qualquer editor (e.g., bloco de notas, terminal, word, etc.)
 - Modo binário (conjunto de bytes)
Ex: Grandes quantidades de informações de forma eficiente
-

Operações

- Abertura do arquivo (localização, alocação do buffer)
- Leitura do arquivo (informações do buffer são disponibilizadas)
- Gravação do arquivo (alteração de dados preexistentes ou acréscimo de novos dados)
- Fechamento do arquivo (atualização do buffer e liberação de memória alocada)

Para trabalhar com arquivos, a linguagem C oferece um pacote de funções de biblioteca divididas em grupos:

- Grupo1: Ler e escrever um caractere por vez (funções `fputc()` e `fgetc()`)
 - Grupo2: Ler e escrever linha a linha (funções `fputs()` e `fgets()`)
 - Grupo3: Ler e escrever dados formatados (funções `fprintf()` e `fscanf()`)
-

Modos de Acesso

modo_de_acesso	Significado
r	Abre o arquivo somente para leitura. O arquivo deve existir.
r+	Abre o arquivo para leitura e escrita. O arquivo deve existir.
w	Abre o arquivo somente para escrita no início do arquivo. Apagará o conteúdo do arquivo se ele já existir, criará um arquivo novo se ele não existir.
w+	Abre o arquivo para escrita e leitura, apagando o conteúdo pré-existente.
a	Abre o arquivo para escrita no final do arquivo. Não apagará o conteúdo pré-existente.
a+	Abre o arquivo para escrita no final do arquivo e leitura.

Abrindo arquivos

A função para abrir arquivos é a `fopen()`
Essa função executa duas tarefas:

- Cria e preenche uma estrutura `FILE`, com as informações necessárias
- Retorna um ponteiro do tipo `FILE` que aponta para a localização na memória dessa estrutura criada

Resumindo, a função `fopen()` abre um arquivo, retornando o ponteiro associado ao arquivo

```
FILE * arq = fopen (nome_arquivo, modo_abertura);
```

nome_arquivo: representa o nome do arquivo que se deseja abrir, podendo conter inclusive o caminho.
modo_abertura: representa como o arquivo será aberto.

Exemplo 1

Programa que cria o arquivo1.txt para escrita no mesmo diretório em que o projeto está sendo executado.

```
#include <stdio.h>

int main()
{
    FILE *arquivo;
    arquivo = fopen("arquivo1.txt","w");
    printf("Arquivo criado.");
    return 0;
}
```

Exemplo 2

Programa que cria o arquivo2.txt para escrita no diretório c:\temp\

Se diretório temp não existe, fopen não indica erro ao criar e não cria o arquivo.

O diretório pode ser criado com o comando system e com os argumentos contendo os comandos do prompt como o “make dir” (md).

```
system("md c:\\temp");
```

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

int main()
{
    FILE *arquivo;
    system("md c:\\temp"); // Tem que ser \\
    arquivo = fopen("c:/temp/arquivo2.txt","w");
    printf("Arquivo 2 criado.");
    return 0;
}
```


Exemplo 3

Programa que cria o arquivo3.txt para escrita no diretório c:\temp\

Pode-se utilizar \\ para indicar apenas uma \ e não confundir com caracteres especiais.

```
#include <stdio.h>

int main()
{
    FILE *arquivo;
    arquivo = fopen("c:\\temp\\arquivo3.txt","w");
    printf("Arquivo 3 criado.");
    return 0;
}
```

Um caractere por vez

A função `fputc()` escreve um caractere por vez. Essa função retorna o caractere gravado ou EOF se acontecer algum erro.

```
int fputc (char ch, FILE *arq);
```

A função `fgetc()` realiza a leitura a partir de um arquivo texto ou EOF se não for possível ler.

```
int fgetc (FILE *arq);
```

Exemplo 4

Programa que cria o arquivo4.txt para escrita no diretório do código

Grava “teste” no arquivo e “1” em outra linha, encerrando a manipulação deste modo de escrita.

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

int main()
{
    FILE *arquivo;
    arquivo = fopen("arquivo4.txt","w");
    fputc('t',arquivo);
    fputc('e',arquivo);
    fputc('s',arquivo);
    fputc('t',arquivo);
    fputc('e',arquivo);
    fputc('\n',arquivo);
    fputc('1',arquivo);
    fclose(arquivo);

    printf("Arquivo 4 criado.");
    return 0;
}
```

Exemplo 5

Programa que lê o arquivo4.txt existente no diretório do código

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

int main()
{
    FILE *arquivo;
    char letra;
    arquivo = fopen("arquivo4.txt","r");
    while ((letra = fgetc(arquivo))!= EOF)
    {
        printf("%c",letra);
    }
    fclose(arquivo);
    return 0;
}
```

Exemplo 6

Programa que lê o arquivo4.txt existente no diretório do código

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

int main()
{
    FILE *arquivo;
    char letra;
    arquivo = fopen("arquivo4.txt","r");
    while (!feof(arquivo))
    {
        letra = fgetc(arquivo);
        printf("%c",letra);
    }
    fclose(arquivo);
    return 0;
}
```

Dados formatados

A função `fprintf()` escreve no arquivo uma sequência de dados formatados, retornando a quantidade de caracteres escritos.

É similar à função `printf()`, exceto pelo fato de que um ponteiro para `FILE` deverá ser incluído como primeiro argumento.

```
int fprintf ( FILE * f, const char * formato,  
             [argumentos] );
```

A função `fscanf()` faz a leitura do arquivo uma sequência de dados formatados, retornando a quantidade de itens lidos.

É similar à função `scanf()`, exceto pelo fato de que um ponteiro para `FILE` deverá ser incluído como primeiro argumento.

```
int fscanf ( FILE * f, const char * formato,  
            [argumentos] );
```

Exemplo 7

Programa que escreve no arquivo5.txt
no diretório do código

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

int main()
{
    FILE *arquivo;
    int a = 5, b = 6;
    arquivo = fopen("arquivo5.txt","w");
    fprintf(arquivo,"%d %d\n",a,b);
    fclose(arquivo);
    return 0;
}
```

Exemplo 8

Programa que lê do arquivo5.txt
existente no diretório do código

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

int main()
{
    FILE *arquivo;
    int a, b;
    arquivo = fopen("arquivo5.txt","r");
    while(fscanf(arquivo,"%d %d",&a,&b)!=EOF)
    {
        printf("Leu %d e %d\n",a,b);
    }
    fclose(arquivo);
    return 0;
}
```