

## Aula 6

### Linguagem de Programação

Prof. Sandro de Araujo

### Arquivos e operações com bits

### Conversa Inicial

- Essa aula apresenta a seguinte estrutura de conteúdo:
- Operações em memória – parte 1
- Operações em memória – parte 2
- Arquivos em c
- Modos de abertura: read (r), write (w) e append(a)
- Gravação e leitura de arquivos

- O objetivo dessa aula é conhecer os principais conceitos e aplicações de operações em memória, arquivos em c, modos de leituras de arquivos e gravação e leitura de arquivos para resolver problemas computacionais

### Operações em memória – parte 1

- As sub-rotinas de memória operam diretamente em áreas de memória e a linguagem de programação C possui algumas funções para manipulação dessas sub-rotinas
- Essas funções pertencem à biblioteca "string.h". Nesse primeiro momento, veremos:
- memset – Usada para preenchimento de memória
- memcpy – Faz cópia de memória

- A função memset() preenche (inicializa) uma quantidade de memória (variável, constante, vetor, estrutura, entre outros) com um determinado valor de Byte
- Sintaxe:
- void \* memset(void \* nPonteiro , int nValor , size\_t nBytes)

- memset()
- Retorno:
- Retorna uma cópia do ponteiro "nPonteiro"
- Retorna "NULL" em caso de erro

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main() {
5
6      //array de 40 posições
7      char exMemset[40] = "Exemplo memset em C\n";
8
9      //função para "colocar" uma string na saída de dados
10     puts(exMemset);
11
12     //Substitui os seis primeiros caracteres
13     memset(exMemset, '*', 6);
14     puts(exMemset);
15
16     system("pause");
17     return 0;
18 }
```

```

Exemplo memset em C
*****o memset em C
Pressione qualquer tecla para continuar. . .
```

- A função memcpy() copia uma quantidade de Bytes de uma área de memória para outra. Ambas as regiões de memória são tratadas com unsigned char
- Sintaxe:
- void\* memcpy(void\* pDestino, void\* pOrigem, size\_t num)

- `memcpy()`
- **Retorno:**
- Retorna uma cópia do ponteiro "`pDestino`"
- Retorna "`NULL`" em caso de erro

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  char nome[7] = "aabbcc";
5
6  int main() {
7
8      printf("String origem: %s\n", nome);
9      memcpy(nome + 2, nome, 4);
10     printf("Nova string: %s\n", nome);
11
12     system("pause");
13     return 0;
14 }
```

```

String origem: aabbcc
Nova string: aaaabb
Pressione qualquer tecla para continuar. . .
```

## Operações em memória – parte 2

- Nesse momento, veremos:
- `memmove` – Também faz cópia de memória, só que de uma forma mais segura
- `memcmp` – Faz comparação de memória

- A função `memmove()` copia uma quantidade de Bytes de uma área de memória para outra. Ambas as regiões de memória são tratadas com `unsigned char`.
- **Sintaxe:**
- `Void* memmove(void* pDestino, void* pOrigem, size_t num)`

- `memmove()`
- **Retorno:**
- Retorna uma cópia do ponteiro "`pDestino`";
- Retorna "`NULL`" em caso de erro

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  char nome1[7] = "aabbcc";
5  char nome2[7] = "ddeeff";
6
7  int main() {
8
9      printf("String origem.....: %s\n", nome1);
10     printf("String antes da funcao...: %s\n", nome2);
11     memmove(nome2, nome1, 3);
12     printf("String depois da funcao..: %s\n\n", nome2);
13
14     system("pause");
15     return 0;
16 }
```

```

String origem.....: aabbcc
String antes da funcao...: ddeeff
String depois da funcao..: aabeff

Pressione qualquer tecla para continuar. . .
```

- A função `memcmp()` é usada para saber se uma string é maior, menor ou igual a outra. Essa função compara as 'n' primeiras posições de duas strings, ou seja, de 0 até n-1
- **Sintaxe:**
- `Int memcmp(void* pRegiao1, void* pRegiao2, size_t N)`

- `memcmp()`
- **Retorno:**
- Se o valor de retorno `< 0` – `pRegiao1` menor que `pRegiao2`
- Se o valor de retorno `== 0` – Blocos de memória são iguais
- Se o valor de retorno `> 0` – `pRegiao1` maior que `pRegiao2`

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  char nome1[15];
5  char nome2[15];
6  int ret;
7
8  int main() {
9
10     memcpy(nome1, "ABCDEF", 6);
11     memcpy(nome2, "abcdef", 6);
12
13     ret = memcmp(nome1, nome2, 6);
```

```

14
15     if (ret > 0) {
16         printf("nome1 eh MAIOR que nome2\n\n");
17     }
18     else if (ret < 0) {
19         printf("nome1 eh MENOR que nome2\n\n");
20     }
21     else {
22         printf("nome1 em igual ao nome2\n\n");
23     }
24     system("pause");
25     return 0;
26 }

```

```

nome1 eh MENOR que nome2

Pressione qualquer tecla para continuar. . .

```

## Arquivos em c

### Manipulação se dá em três etapas:

- 1) abrir o arquivo
- 2) ler e/ou gravar os dados
- 3) fechar o arquivo

- Para manipular arquivos, a linguagem C disponibiliza biblioteca stdio.h e um tipo especial de ponteiro
- Sintaxe:
- FILE \*nomePonteiro
- Sintaxe:
- fopen(nome\_do\_arquivo, modo)
- ou
- fopen\_s(ponteiro\_do\_arquivo, nome\_do\_arquivo, modo)

```

int main() {

    FILE *arq;
    arq = fopen("uninter.txt", "r");
    if((arq=fopen("uninter.txt", "r"))==NULL){
        printf("ERRO na abertura do arquivo\n\n");
        system("pause");
        exit(1);
    }

    fclose(arq);
    system("pause");
    return 0;
}

```

```
ERRO na abertura do arquivo
Pressione qualquer tecla para continuar. . .
```

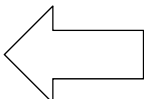
```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5
6     FILE *arquivo;
7     errno_t err;
8
9     err = fopen_s(&arquivo, "uninter.txt", "w");
10
11     if (err == 0) {
12         printf("Arquivo ABERTO\n\n");
13     }
14     else {
15         printf("Arquivo Nao foi ABERTO\n\n");
16     }
17 }
18
```

```
19     if (arquivo) {
20         err = fclose(arquivo);
21         if (err == 0) {
22             printf("Arquivo FECHADO!\n\n");
23         }
24         else {
25             printf("Arquivo nao foi FECHADO!\n\n");
26         }
27     }
28     system("pause");
29     return 0;
30 }
```

```
Arquivo ABERTO
Arquivo FECHADO!
Pressione qualquer tecla para continuar. . .
```

Nome

- ☐ Debug
- ☐ fopen
- ☒ fopen.sln
- ☐ uninter



- Para saber o final do arquivo, a linguagem C procura um sinal, uma constante conhecida por EOF, que sinaliza o fim do arquivo
- Se o byte lido pelo algoritmo representa o EOF, a função `fclose()` "fecha" a abertura do arquivo. Ou seja, libera a memória associado ao ponteiro do `FILE*`
- Assim como em ponteiros, quando usamos a função `free()` para liberar memória alocada, fechar os arquivos que não estão sendo mais usados é uma boa prática de programação

**Modos de abertura:**  
read (r), write (w) e append(a)

- O modo de acesso é uma string que contém uma sequência de caracteres que informam se o arquivo será aberto para escrita ou leitura. Depois que abrir o arquivo, podemos executar os tipos de ação previstos pelo modo de acesso. Assim, não será possível ler um arquivo que foi aberto somente para escrita

- read(r) - Leitura de arquivo
- r – Para ler um arquivo.
  - Exemplo: FILE \*arquivo = fopen("uninter.txt", "r")
- r+ – O "+" no "r", abre o arquivo para leitura e escrita.
  - Exemplo : FILE \*arquivo = fopen("uninter.txt", "r+")
- rb – Abre o arquivo em modo binário para leitura
  - Exemplo: FILE \*arquivo = fopen("uninter.txt", "rb")

- write(w) - Escrita em arquivo
- w – Para abrir um arquivo no modo de escrita. Esse modo cria automaticamente o arquivo ou substitui seu conteúdo anterior
  - Exemplo: FILE \*arquivo = fopen("uninter.txt", "w")

- w+ – Abre um arquivo tanto para leitura quanto para escrita. Se o arquivo já existir terá seu conteúdo substituído
  - Exemplo: FILE \*arquivo = fopen("uninter.txt", "w+")
- wb – Usado para escrita em arquivos no modo binário
  - Exemplo: FILE \*arquivo = fopen("uninter.txt", "wb")

- append(a) - Escrita no final do arquivo (anexando)
- a – Usado para ANEXAR, informações no arquivo
  - Exemplo: FILE \*arquivo = fopen("uninter.txt", "a")

- a+ – Abre um arquivo no modo de leitura ou no modo de escrita ao final do arquivo (anexar)
  - Exemplo: FILE \*arquivo = fopen("arquivo.txt", "a+")
- ab – Do mesmo modo da leitura binária "rb" e da escrita binária "wb", podemos anexar informações ao final do arquivo
  - Exemplo: FILE \*arquivo = fopen("uninter.txt", "ab")

```

int main() {
    FILE *arq;
    arq = fopen("uninter.txt", "r");
    if((arq=fopen("uninter.txt", "r"))==NULL){
        printf("ERRO na abertura do arquivo\n\n");
        system("pause");
        exit(1);
    }
    system("pause");
    return 0;
}

int main() {
    FILE *arquivo;
    errno_t err;
    err = fopen_s(&arquivo, "uninter.txt", "r");
    if (err == 0) {
        printf("arquivo ABERTO\n\n");
    }
    else {
        printf("arquivo nao foi ABERTO\n\n");
    }
    system("pause");
    return 0;
}

```

## Gravação e leitura de arquivos

- Existem várias funções em C para a operação de gravação e leitura de dados em arquivos. Agora, vamos trabalhar com duas funções, da biblioteca stdio.h, que grava e lê um arquivo txt caractere por caractere
- A função fputc() que é usada para escrever um caractere de cada vez em um determinado arquivo
- A função fgetc() que é usada para obter entrada de um único caractere de arquivo por vez

### fputc()

- Grava o caractere fornecido na posição indicada pelo ponteiro do arquivo e, em seguida, avança o ponteiro do arquivo
- Sintaxe:
- int fputc(char c, FILE \*arquivo)

### fputc()

- Retorno:
  - Se houver erro, a função retorna a constante EOF
  - Se o algoritmo tiver sucesso, retornará o próprio caractere

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5
6     FILE *arquivo;
7     errno_t err;
8     int numero;
9
10    err = fopen_s(&arquivo, "uninter.txt", "w");
11
12    if (err == 0) {
13        printf("Digite um numero inteiro: ");
14        do {
15            numero = getchar();
16            fputc(numero, arquivo);
17        } while (numero != '\n');
18    }
19 }

```



```

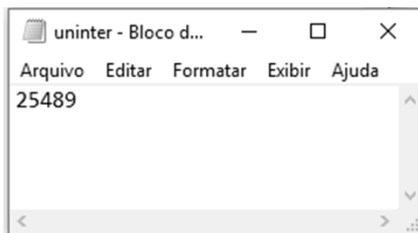
20     else
21     {
22         printf("Arquivo Nao foi ABERTO\n\n");
23     }
24
25     if (arquivo) {
26         err = fclose(arquivo);
27         if (err == 0) {
28             printf("\nArquivo FECHADO!\n\n");
29         }
30         else {
31             printf("Arquivo nao foi FECHADO!\n\n");
32         }
33     }
34     system("pause");
35     return 0;
36 }

```

```

Digite um numero inteiro: 25489
Arquivo FECHADO!
Pressione qualquer tecla para continuar. . .

```



## fgetc()

- Lê o caractere presente na posição indicada pelo ponteiro do arquivo e automaticamente já se posiciona no próximo campo e assim segue lendo até encontrar a constante EOF
- Sintaxe:
- `int fgetc(FILE *arquivo)`

```

1  #include<stdio.h>
2  #include<stdlib.h>
3
4  int main(void)
5  {
6      FILE *arquivo;
7      char localMemoria[81];
8      int i, ch;
9
10     // Abrir o arquivo para ler a linha:
11     fopen_s(&arquivo, "uninter.txt", "r");
12     if (arquivo == NULL)
13         exit(0);
14

```

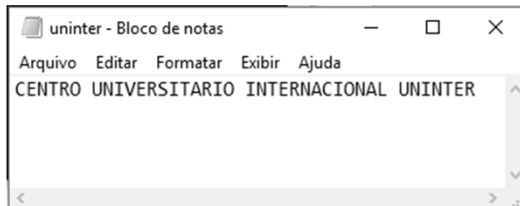
```

15     // Leia os primeiros 80 caracteres e coloque-os em "localMemoria":
16     ch = fgetc(arquivo);
17     for (i = 0; (i < 80) && (feof(arquivo) == 0); i++)
18     {
19         localMemoria[i] = (char)ch;
20         ch = fgetc(arquivo);
21     }
22
23     // Adicionar NULL ao final da String
24     localMemoria[i] = '\0';
25     printf("%s\n\n", localMemoria);
26     fclose(arquivo);
27     system("pause");
28 }

```

```
CENTRO UNIVERSITARIO INTERNACIONAL UNINTER
```

```
Pressione qualquer tecla para continuar. . . _
```



FUNÇÃO	Usada para?
fscanf()	Lê um arquivo
fprintf()	Escreve textos (strings) em arquivos
fgets()	Lê uma string em um arquivo
fputs()	Insere uma string no arquivo
fread()	Lê um bloco de dados do arquivo
fwrite()	Escreve um bloco de dados no arquivo
fseek()	Reposiciona o ponteiro
rewind()	Reposiciona o ponteiro para o início do arquivo
ftell()	Retorna à posição do ponteiro.

Fonte: Mizrahi (2008)