Programação 2

Manipulação de Arquivos

Instituto Federal do Espírito Santo - Ifes
Campus Colatina
Professor: Giovany Frossard Teixeira

Tipo Arquivo

- Um arquivo é uma forma alternativa à entrada/saída de dados via teclado, para representar um arquivo utilizamos a palavra FILE (arquivo em inglês).

- As funções de manipulação de arquivos utilizam ponteiros, daí normalmente criamos variáveis do tipo FILE * e não FILE.

Funções - fopen

- fopen: é a função que abre ou cria o arquivo, é a partir dela que toda a manipulação começa. Sua assinatura é da seguinte forma:

```
FILE *fopen (char *nome_do_arquivo, char *modo);
```

Onde:

nome_do_arquivo: nome do arquivo aberto, junto com seu caminho na estrutura de diretórios.

modo: forma ou permissão que o arquivo será aberto

Funções - fopen

- A função fopen, pode abrir um arquivo de muitas formas, o quadro ao lado ilustra essas possibilidades:

Modo	Significado
"r"	Abre um arquivo texto para leitura. O arquivo deve existir antes de ser aberto.
"w"	Abrir um arquivo texto para gravação. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído.
"a"	Abrir um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo ("append"), se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"rb"	Abre um arquivo binário para leitura. Igual ao modo "r" anterior, só que o arquivo é binário.
"wb"	Cria um arquivo binário para escrita, como no modo "w" anterior, só que o arquivo é binário.
"ab"	Acrescenta dados binários no fim do arquivo, como no modo "a" anterior, só que o arquivo é binário.
"r+"	Abre um arquivo texto para leitura e gravação. O arquivo deve existir e pode ser modificado.
"w+"	Cria um arquivo texto para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado.
"a+"	Abre um arquivo texto para gravação e leitura. Os dados serão adicionados no fim do arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"r+b"	Abre um arquivo binário para leitura e escrita. O mesmo que "r+" acima, só que o arquivo é binário.
"w+b"	Cria um arquivo binário para leitura e escrita. O mesmo que "w+" acima, só que o arquivo é binário.
"a+b"	Acrescenta dados ou cria uma arquivo binário para leitura e escrita. O mesmo que "a+" acima, só que o arquivo é binário

Instituto Federal do Espírito Santo - Ifes
Campus Colatina
Professor: Giovany Frossard Teixeira

Funções - fopen

- Exemplo:

```
int main()
{
    FILE *fp; // Declaração da estrutura
    fp = fopen ("exemplo.bin","wb"); /* o
        arquivo se chama
    exemplo.bin e está localizado no diretório
        corrente */
    if (!fp)
    {
        printf ("Erro na abertura do
            arquivo.");
    }
    else
    {
        printf("Arquivo aberto com
            sucesso");
    }
}
```

Funções - fclose

- fclose: é a função que fecha o arquivo, após o uso dessa função o arquivo não será mais acessado, a não ser que seja novamente aberto. Sua assinatura é da seguinte forma:

```
int fclose (FILE *fp);
```

Obs. A função retorna 0 (zero) em caso de sucesso (se conseguir fechar o arquivo)

Funções - fclose

- Exemplo:

```
int main()
{
	FILE *fp; // Declaração da estrutura
	fp = fopen ("exemplo.bin","wb"); /* o arquivo se chama
	exemplo.bin e está localizado no diretório corrente */
	if (fclose(fp) == 0)
		printf ("Arquivo fechado com sucesso.");
	else
		printf("Erro no fechamento do arquivo");
}
```

Funções - fprintf

- fprintf: A função **fprintf()** funciona como a função **printf()**. A diferença é que a saída de **fprintf()** é um arquivo e não a tela do computador. Sua assinatura pode ser definida da seguinte forma:

```
int fprintf (FILE *, <mensagem formatada>,  parametros da mensagem);
```

- Como já poderíamos esperar, a única diferença do protótipo de **fprintf()** para o de **printf()** é a especificação do arquivo destino através do ponteiro de arquivo.

Funções - fprintf

- Exemplo:

```
int main()
{
	FILE *fp; // Declaração da estrutura
	fp = fopen ("exemplo.txt","w"); /* o arquivo se chama
	exemplo.txt e está localizado no diretório corrente */
	fprintf(fp, "Olá mundo.\n");
	fclose(fp);
}
```

Será escrito no arquivo exemplo.txt a string: “Olá mundo.”

Funções - fscanf

- fscanf: A função **fscanf()** funciona como a função **scanf()**. A diferença é que **fscanf()** lê de um arquivo e não do teclado do computador. Sua assinatura pode ser definida da seguinte forma:

```
int fscanf (FILE *fp, < identificação dos tipos das variáveis >, variáveis
    precedidas pelo & );
```

- Como já poderíamos esperar, a única diferença do protótipo de **fscanf()** para o de **scanf()** é a especificação do arquivo destino através do ponteiro de arquivo.

Funções - fscanf

- Exemplo:

```
int main()
{
        FILE *fp; // Declaração da estrutura
        char str[50];
        fp = fopen ("exemplo.txt", "r"); /* o arquivo se chama
        exemplo.txt e está localizado no diretório corrente */
        fscanf(fp, "%s", str);
        printf("A string lida é: %s ", str);
        fclose(fp);
}
```

Será mostrada a string: "Olá". Isso ocorre porque o fscanf considera o espaço em branco um caracter de término de leitura.

Funções - feof

- feof: EOF ("End of file") indica o fim de um arquivo. Às vezes, é necessário verificar se um arquivo chegou ao fim. Para isto podemos usar a função **feof()**. Ela retorna não-zero se o arquivo chegou ao EOF, caso contrário retorna zero. A assinatura da função feof pode é definida da seguinte forma:

```
int feof (FILE *fp);
```

Funções - feof

- Exemplo:

```
int main()
{
        FILE *fp; // Declaração da estrutura
        char str[50];
        int cont;
        fp = fopen ("exemplo.txt", "r");
        cont = 0;
        while(!feof(fp))
        {
                fscanf(fp, "%c", &str[cont]);
                cont++;
        }
        str[cont] = '\0';
        printf("A string lida é: %s ", str);
        fclose(fp);
}
```

Funções - fread

- fread: Podemos escrever e ler blocos de dados. A função fread faz a leitura de blocos de dados. Sua assinatura é da seguinte forma:

```
unsigned fread (void *buffer, int numero_de_bytes, int count, FILE *fp);
```

Onde:

buffer: é a região de memória na qual serão armazenados os dados lidos.

numero_de_bytes: é o tamanho da unidade a ser lida.

count: indica quantas unidades devem ser lidas. Isto significa que o número total de bytes lidos é: *numero_de_bytes*count*

A função retorna o número de unidades efetivamente lidas. Este número pode ser menor que *count* quando o fim do arquivo for encontrado ou ocorrer algum erro.

Funções - fwrite

- fwrite: A função **fwrite()** funciona como a sua companheira **fread()**, porém escrevendo no arquivo. Sua assinatura é:

```
unsigned fwrite(void *buffer,int numero_de_bytes,int count,FILE *fp);
```

- A função retorna o número de itens escritos. Este valor será igual a count a menos que ocorra algum erro.

Exemplo fread e fwrite

- Exemplo:

```
#include <stdio.h>

#include <stdlib.h>

int main()

{

        FILE *pf;

        float pi = 3.1415, pilido;

        fopen("arquivo.bin", "wb");

        if(fwrite(&pi, sizeof(float), 1,pf) != 1) /*
            Escreve a variável pi */

                printf("Erro na escrita do arquivo");

        fclose(pf); /* Fecha o arquivo*/

        pf = fopen("arquivo.bin", "rb");

        if(fread(&pilido, sizeof(float), 1,pf) != 1)

                printf("Erro na leitura do arquivo");

        else

                printf("\nO valor de PI, lido do
                    arquivo e': %f", pilido);

        fclose(pf);

        return(0);

}
```

Funções - fseek

- fseek: Para se fazer procuras e acessos randômicos em arquivos usa-se a função **fseek()**. Esta move a posição corrente de leitura ou escrita no arquivo de um valor especificado, a partir de um ponto especificado. A assinatura da função fseek é:

```
int fseek (FILE *fp,long numbytes,int origem);
```

O parâmetro *origem* determina a partir de onde os *numbytes* de movimentação serão contados.

Funções - fseek

- O parâmetro origem é definido de acordo com a seguinte tabela:

Nome	Valor	Significado
SEEK_SET	0	Início do arquivo
SEEK_CUR	1	Ponto corrente no arquivo
SEEK_END	2	Fim do arquivo

Funções - fseek

- Exemplo:

```
int main()
{
        FILE *fp;
        char str[50];
        int cont = 0, encontrou_espaco = 0;
        fp = fopen ("exemplo.txt", "r");  // Supor no
             arquivo: “Ola mundo”
        while((!feof(fp)) && (!encontrou_espaco))
        {
                fscanf(fp, "%c", &str[cont]);
                if (str[cont] == ' ') // Vai encontrar o
                   espaço depois do Ola
                        encontrou_espaco = 1; //
                           verdadeiro
                cont++;
        }
        fseek(fp, sizeof(char), SEEK_CUR); // Avança
             um char ( m )
        fscanf(fp, "%s", str); // Lê: ”undo”
        printf("A string lida é: %s ", str);
        fclose(fp);
}
```

Funções - rewind

- **rewind**: retorna a posição corrente do arquivo para o início. Equivale ao fseek com origem = SEEK_SET e num_bytes = 0. Sua assinatura é:

```
void rewind (FILE *fp);
```

Funções - rewind

- Exemplo:

```
int main()
{
        FILE *fp;
        char str[50];
        int cont = 0, encontrou_espaco = 0;
        fp = fopen ("exemplo.txt", "r");  // Supor no
             arquivo: “Ola mundo”
        while((!feof(fp)) && (!encontrou_espaco))
        {
                fscanf(fp, "%c", &str[cont]);
                if (str[cont] == ' ') // Vai encontrar o
                   espaço depois do Ola
                        encontrou_espaco = 1; //
                           verdadeiro
                cont++;
        }
        rewind(fp);
        fscanf(fp, "%s", str); // Lê: ”Ola”
        printf("A string lida é: %s ", str);
        fclose(fp);
}
```

Funções - remove

- **remove**: Apaga um arquivo especificado. Sua assinatura é:

```
int remove (char *nome_do_arquivo);
```

Funções - remove

- Exemplo:

```
int main()
{
   remove("exemplo.txt“ )
}
```

Bibliografia

- SANTOS, Henrique José. **Curso de Linguagem C da UFMG**, apostila.
- FORBELLONE, André Luiz. **Lógica de Programação – A Construção de Algoritmos e Estruturas de Dados.** São Paulo: MAKRON, 1993.