Engenharia de Computação

Fundamentos de Programação

Aula 22 – Arquivos – Parte 2

Prof. Muriel de Souza Godoi
muriel@utfpr.edu.br

Arquivos

- Já vimos:
  - Leitura e escrita de caracteres
  - Leitura e escrita de sequências de caracteres

- Agora vamos aprender a:
  - Leitura e escrita de blocos de dados.
  - Para tanto, temos duas funções
    - `fwrite e fread()`
  - Quando o arquivo for aberto para dados binários, fwrite e fread podem manipular qualquer tipo de dado.
    - Int, char, float, vetores, structs, ...

Função `fwrite` – Escrita binária

```
unsigned fwrite(void *buffer, int nro_bytes,
                int quantidade, FILE* arq);
```

- **Argumentos**
  - **buffer**: ponteiro para a região de memória na qual estão os dados a serem escritos;
  - **nro_bytes**: tamanho de cada posição de memória a ser escrita;
  - **quantidade**: total de unidades de memória que devem ser escritas;
  - **arq**: ponteiro associado ao arquivo onde os dados serão escritos
- **Retorno**
  - O número de unidades efetivamente escritas.
  - Este número pode ser menor que quantidade quando ocorrer algum erro.

Função fwrite – Escrita binária

- Exemplo da função **fwrite**

```
char str[20]="Ola arquivo";
float numero = 5.23;
int vetor[5] = {10, 20, 30, 40, 50};

FILE *arq = fopen("arquivo.bin","wb");
  ...
fwrite(str,sizeof(char),20,arq);
fwrite(&numero,sizeof(float),1,arq);
fwrite(vetor,sizeof(float),5,arq);

fclose(arq);
```

Função fread – Leitura binária

```
unsigned fread(void *buffer, int nro_bytes,
               int quantidade, FILE* arq);
```

- **Argumentos**
  - **buffer**: ponteiro para a região de memória na qual serão salvos os dados lidos;
  - **nro_bytes**: tamanho de cada posição de memória a ser lida;
  - **quantidade**: total de unidades de memória que devem ser lidas;
  - **arq**: ponteiro associado ao arquivo onde os dados serão lidos
- **Retorno**
  - O número de unidades efetivamente lidos.
  - Este número pode ser menor que quantidade quando ocorrer algum erro.

Função fread – Leitura binária

- Exemplo da função **fread**

```
char str[20];
float numero ;
int vetor[5];
FILE *arq = fopen("arquivo.bin","rb"); //Abre o arquivo
   ...
fread(str,sizeof(char),20,arq);// Lê 20 caracteres
fread(&numero,sizeof(float),1,arq);// Lê um número real
fread(vetor,sizeof(float),5,arq);// Lê o vetor de tamanho 5

// Exibe os dados lidos
printf("Texto: %s\n",str);
printf("Float: %.2f\n",numero);
for (int i = 0; i < 5; i++) {
  printf("%i ",vetor[i]);
}//for
fclose(arq);
```

Escrita/Leitura de Dados Formatados

- Permitem ao programador ler e escrever em arquivos da maneira padrão com a qual o já líamos e escrevíamos na tela.
- As funções **fprintf** e **fscanf** funcionam de maneiras semelhantes a printf e scanf, respectivamente
- A diferença é que elas direcionam os dados para arquivos.

Escrita/Leitura de Dados Formatados

- `fprintf` - Escrita formatada
  - Utiliza quase o mesmo formato do `printf`
  - Apenas adiciona o argumento do arquivo
  - Exemplo:
    - Impressão comum na tela:

```
printf("Total = %d \n", valor);
```

    - Impressão no arquivo `arq`

```
fprintf(arq, "Total = %d \n", valor);
```

Escrita/Leitura de Dados Formatados

- Exemplo da função **fprintf**

```
char texto[20] = "Apucarana";
float real = 5.26;
int inteiro = 42;
FILE *arq = fopen("arquivo.txt","w"); //Abre o arquivo
   ...
//Realiza a escrita utilizando o fprintf
fprintf(arq,"Texto: %s\n", texto);
fprintf(arq,"Real: %f", real);
fprintf(arq,"Inteiro: %i", inteiro);

fclose(arq);
```

Escrita/Leitura de Dados Formatados

- `fscanf` - Leitura formatada
  - Utiliza quase o mesmo formato do `scanf`
  - Apenas adiciona o argumento do arquivo
  - Exemplo:
    - Leitura de um inteiro no teclado:

```
scanf("%d", &valor);
```

    - Leitura de um inteiro no arquivo `arq`

```
fscanf(arq, "%d", &valor);
```

Escrita/Leitura de Dados Formatados

- Exemplo da função **fscanf**

```
char texto[20] = "Apucarana", label[20];
float real = 5.26;
int inteiro = 42;
FILE *arq = fopen("arquivo.txt","w"); //Abre o arquivo
   ...
//Realiza a leitura utilizando o fscanf
fscanf(arq,"%s %s\n", label, texto);
fscanf(arq,"%s %f", label, &real);
fscanf(arq,"%s %i", label, &inteiro);

//Exibe os dados lidos
printf("Texto: %s\n", texto);
printf("Real: %f", real);
printf("Inteiro: %i", inteiro);

fclose(arq);
```

Escrita/Leitura de Dados Formatados

- **Atenção**
  - Embora `fprintf` e `fscanf` sejam mais fáceis de ler/escrever dados em arquivos, nem sempre elas são as escolhas mais apropriadas.
  - Como os dados são escritos em ASCII e formatados como apareceriam em tela, um tempo extra é perdido.
  - Se a intenção é velocidade ou tamanho do arquivo, utilize as funções fread e fwrite.

Posição do arquivo

- Ao se trabalhar com arquivos, existe uma espécie de posição onde estamos dentro do arquivo. É nessa posição onde será lido ou escrito o próximo caractere.
  - Quando utilizando o acesso sequencial, raramente é necessário modificar essa posição.
  - Isso por que, quando lemos um caractere, a posição no arquivo é automaticamente atualizada.
  - Leitura e escrita em arquivos são parecidos com escrever em uma máquina de escrever

Movendo-se pelo arquivo

- A função permite movimentar o ponteiro de leitura do arquivo

```
int fseek(FILE* arq, int deslocamento, int origem);
```

  - **arq**: o ponteiro para o arquivo;
  - **deslocamento**: é o total de bytes a partir de origem a ser pulado;
  - **origem**: determina a partir de onde o **deslocamento** de movimentação será contado.
- A função devolve o valor **0** quando bem sucedida

Movendo-se pelo arquivo

- Os valores possíveis para origem são definidos por macros em stdio.h e são:

Nome	Valor	Significado
SEEK_SET	0	Início do arquivo
SEEK_CUR	1	Ponto corrente do arquivo
SEEK_END	2	Fim do arquivo

- Portanto, para mover **deslocamento** bytes a partir:
  - do início do arquivo, origem deve ser SEEK_SET
  - da posição atual, origem deve ser SEEK_CUR
  - do final do arquivo, origem deve ser SEEK_END
- **deslocamento** pode ser negativo quando usado com SEEK_CUR e SEEK_END

Funções Adicionais

- Remover (excluir!) um arquivo:

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

- Argumento
  - **nome_arquivo** recebe o caminho e nome do arquivo a ser excluído, e não um ponteiro para FILE.
- Retorno
  - Um valor inteiro, o qual será igual a 0 se o arquivo for excluído com sucesso.

Funções Adicionais

- Renomear um arquivo:

```
int rename(char* nome_antigo, char* nome_novo);
```

- Argumento
  - **nome_antigo** recebe o caminho e nome do arquivo antigo.
  - **nome_novo** recebe o caminho e nome do arquivo novo.
- Retorno
  - Um valor inteiro, o qual será igual a 0 se o arquivo for renomeado com sucesso.

Funções Adicionais

- Saber a posição de leitura/escrita em um arquivo:

```
long ftell(FILE* arq);
```

- Argumento
  - **arq** ponteiro para o arquivo
- Retorno
  - A posição do ponteiro de leitura e escrita dentro de um arquivo. **Ex.:**
    - 0 se estiver no começo do aquivo
    - tamanho do arquivo (em bytes), se estiver no final

Exercícios

- **1)** Faça um programa que gere 100 números aleatórios. Esse programa deverá, em seguida,armazenar esses números em um arquivo binário.
- **2)** Elabore um programa que leia um arquivo binário contendo 100 números. Mostre na tela a soma desses números.
- **3)** Crie uma estrutura representando um atleta. Essa estrutura deve conter o nome do atleta, seu esporte, idade e altura. Agora, escreva um programa que leia os dados de cinco atletas e os armazene em um arquivo binário.
- **4)** Considerando a estrutura atleta do exercício anterior, escreva um programa que leia um arquivo binário contendo os dados de cinco atletas. Calcule e exiba o nome do atleta mais alto e do mais velho.

Exercícios

- **5)** Crie uma estrutura representando um ranking. Essa estrutura deve conter o nome do jogador e sua pontuação. Em seguida, escreva um programa que leia os dados de 5 jogadores e os armazene em um arquivo utilizando a técnica de Dados formatados.
- **6)** Considerando a estrutura ranking do exercício anterior, escreva um programa que leia o arquivo gerado contendo os dados dos 5 jogadores.