FACULDADE DE ENGENHARIA DE SOROCABA

ENGENHARIA DA COMPUTAÇÃO

PROGRAMAÇÃO ESTRUTURADA II

Módulo 2 – Arquivos em I/O

PROFª. ANDRÉA

4. ARQUIVOS EM I/O 3

4.1 - INTRODUÇÃO 3
4.2 – ALTO-NÍVEL 3
4.3 – BAIXO-NÍVEL 3
4.4 – TEXTO X BINÁRIO 3
4.5 – OPERAÇÕES COM ARQUIVOS EM DISCO – ALTO-NÍVEL 4
4.5.1 – PONTEIRO PARA ARQUIVO 4
4.5.2 – FUNÇÃO FOPEN( ) 4
4.5.3 – FUNÇÃO FCLOSE( ) 5
4.5.4 – FIM DE ARQUIVO (EOF) 5
4.5.5 – LEITURA E GRAVAÇÃO – CARACTER POR CARACTER 5
4.5.6 – LEITURA E GRAVAÇÃO – LINHA A LINHA 6
4.5.7 – LEITURA E GRAVAÇÃO – MODO FORMATADO 6
4.5.8 – LEITURA E GRAVAÇÃO – REGISTROS 7
4.5.9 – EXEMPLOS 7
4.6 – ACESSO ALEATÓRIO – ALTO-NÍVEL 10
4.6.1 – PONTEIRO PARA ARQUIVO 10
4.6.2 – FUNÇÃO FSEEK( ) 10
4.6.3 – FUNÇÃO FTELL( ) 11
4.6.4 – FUNÇÃO REWIND( ) 11
4.6.5 – EXEMPLO 11
4.6.6 – EXERCÍCIOS 12

4. ARQUIVOS EM I/O

4.1 - Introdução

Operações em discos são executadas em entidades chamadas "**arquivos**". Arquivo é uma coleção de bytes referenciados por um nome único.

A linguagem C divide as categorias de acesso a disco em 2 grupos: alto-nível e baixo-nível.

4.2 – Alto-Nível

Também chamada de **bufferizada**, permite acessar arquivos de 4 modos diferentes:

- Os dados são lidos e escritos **um caracter por vez** – semelhante ao getch() e putch()
- Os dados são lidos e escritos como **strings** – semelhante ao gets() e puts()
- Os dados são lidos e escritos de **modo formatado** – semelhante ao scanf() e printf()
- Os dados são lidos e escritos como **registro** ou **bloco** – os dados tem tamanho fixo e é utilizado para armazenar matrizes e/ou estruturas

4.3 – Baixo-Nível

Também chamada de **não bufferizada**, permite acessar arquivos de um único modo:

- Os dados são lidos e escritos através de **um buffer cheio de dados.** E o programador é quem deverá criar e manter o buffer, pois as funções não fazem esta manutenção.

Este modo é usado para criar as funções em alto-nível e não é portável, ou seja, não é reconhecido pelo padrão ANSI.

4.4 – Texto X Binário

É a forma como os arquivos são abertos e, constitui outra maneira de classificar o acesso à arquivos.

- **Modo Texto** – imita arquivos UNIX - o arquivo aberto deste modo é interpretado como seqüências de caracteres agrupados em linhas, e é reconhecida a indicação de nova linha e de fim de arquivo. Neste modo os números são guardados como cadeias de caracteres.
- **Modo Binário** – imita arquivos MS-DOS - é mais simples que o modo texto, não há conversão, ou seja, qualquer caracter é lido ou gravado sem alteração e não temos indicação de fim de arquivo. Neste modo os números são guardados como estão na memória (respeitando o tipo básico e a quantidade de bytes).

4.5 – Operações com Arquivos em Disco – Alto-Nível

4.5.1 – Ponteiro para arquivo

Para trabalharmos com arquivos, precisamos declarar um ponteiro para o tipo FILE (arquivo).

Arquivo de cabeçalho: stdio.h
sintaxe:

```
FILE   *fptr ;
```

onde:
FILE – estrutura que contém informações específicas sobre o arquivo (tamanho, se de leitura ou escrita)
fptr – corresponde ao nome de um ponteiro para arquivo (FILE)

4.5.2 – Função fopen( )

Esta função é utilizada para criar (abrir) um arquivo e, retorna um ponteiro para o tipo FILE.

Arquivo de cabeçalho: stdio.h
sintaxe:

```
fptr = fopen ("xxxxxxxx.xxx", "tipo_abertura");
```

onde:
fptr – corresponde a um ponteiro para arquivo (FILE) a ser aberto ou criado
xxxxxxxx.xxx – nome e extensão do arquivo a ser aberto – respeitar 8 e 3 caracteres respectivamente
tipo_abertura – contém o tipo de abertura do arquivo:

"r" – leitura **"w" – escrita** **"a"- adicionar dados**
além de 2 modificadores: **"+" – atualização** **"b"- modo binário**

Tipo	**Ação**
"r"	Abre um arquivo texto para leitura – o arquivo deve existir
"w"	Cria um arquivo texto para gravação – se o arquivo existir será destruído e reinicializado – se o arquivo não existir será criado
"a"	Cria um arquivo texto para gravação – se o arquivo existir, os dados serão adicionados no fim – se o arquivo não existir será criado
"r+"	Abre um arquivo texto para leitura e gravação – o arquivo deve existir, sendo atualizado
"w+"	Cria um arquivo texto para leitura e gravação – se o arquivo existir será destruído e reinicializado – se o arquivo não existir será criado
"a+"	Cria um arquivo texto para atualizações – se o arquivo existir, os dados serão adicionados no fim – se o arquivo não existir será criado
"rb"	Abre um arquivo binário para leitura – o arquivo deve existir
"wb"	Cria um arquivo binário para gravação – se o arquivo existir será destruído e reinicializado – se o arquivo não existir será criado
"ab"	Cria um arquivo binário para gravação – se o arquivo existir, os dados serão adicionados no fim – se o arquivo não existir será criado
"rb+"	Abre um arquivo binário para leitura e gravação – o arquivo deve existir, sendo atualizado
"wb+"	Cria um arquivo binário para leitura e gravação – se o arquivo existir será destruído e reinicializado – se o arquivo não existir será criado
"ab+"	Cria um arquivo binário para atualizações – se o arquivo existir, os dados serão adicionados no fim – se o arquivo não existir será criado

Cuidado: o fopen( ) dá erro:

- se tentarmos abrir um arquivo inexistente com o tipo **"r"**, **"r+"**, **"rb"**ou **"rb+"**
- se o diretório for inexistente ou inválido
- se nome do arquivo inválido

Portanto, deve-se sempre fazer um teste, e caso o arquivo não possa ser aberto, a função fopen() devolve 0 (NULL).

4.5.3 – Função fclose( )

Esta função é utilizada para fechar um arquivo.

Arquivo de cabeçalho: stdio.h
sintaxe:

```
fclose (fptr);
```

onde:
fptr – corresponde ao ponteiro do arquivo a ser fechado

4.5.4 – Fim de Arquivo (EOF)

Este sinal nos indica o último dado contido no arquivo, e é enviado ao programa pelo sistema operacional e definido no arquivo <stdio.h>.

4.5.5 – Leitura e Gravação – Caracter por Caracter

Lê e grava os dados no **modo texto**.

Arquivo de cabeçalho: stdio.h

Leitura do arquivo – recebe informações do arquivo
sintaxe:

```
ch = getc (fptr);          // lê um caracter por vez no arquivo
```

onde:
ch – armazena o caracter lido
fptr – corresponde ao ponteiro do arquivo a ser lido

Escrita no arquivo – grava informações no arquivo
sintaxe:

```
putc (ch, fptr);           // armazena um caracter por vez no arquivo
```

onde:
ch – caracter a ser armazenado
fptr – corresponde ao ponteiro do arquivo a ser gravado

4.5.6 – Leitura e Gravação – Linha a Linha

Lê e grava os dados no **modo texto**.

Arquivo de cabeçalho: stdio.h

Leitura do arquivo – recebe informações do arquivo
sintaxe:

```
fgets (str, num, fptr);          // lê uma linha por vez no arquivo
```

onde:
str – string auxiliar onde a linha lida será armazenada
num – inteiro que indica o limite de caracteres a serem lidos, não esquecer do '\0' (NULL).
fptr – corresponde ao ponteiro do arquivo a ser lido

Escrita no arquivo – grava informações no arquivo
sintaxe:

```
fputs (str, fptr);               // armazena um caracter por vez no arquivo
```

onde:
str – string a ser armazenado
fptr – corresponde ao ponteiro do arquivo a ser gravado

4.5.7 – Leitura e Gravação – Modo Formatado

Lê e grava os dados no **modo texto**.

Arquivo de cabeçalho: stdio.h

Leitura do arquivo – recebe informações do arquivo
sintaxe:

```
fscanf (fptr, "especificador de formato", &variáveis);      // lê dados do arquivo
```

onde:
fptr – corresponde ao ponteiro do arquivo a ser lido
especificador de formato - %c, %i, %f, %s (referente aos tipos básicos e modificados)
&variáveis – endereço das variáveis onde serão armazenados os dados lidos

Escrita no arquivo – grava informações no arquivo
sintaxe:

```
fprintf (fptr, "especificador de formato", variáveis);      // armazena dados no arquivo
```

onde:
fptr – corresponde ao ponteiro do arquivo a ser gravado
especificador de formato - %c, %i, %f, %s (referente aos tipos básicos e modificados)
variáveis – dados a serem armazenados

4.5.8 – Leitura e Gravação – Registros

Lê e grava os dados complexos como matrizes e estruturas, porém no **modo binário**.

Arquivo de cabeçalho: stdio.h

Leitura do arquivo – recebe informações do arquivo
sintaxe:

```
fread (p, tam, qtde, fptr);          // lê dados do arquivo
```

onde:
p – ponteiro do tipo void para o endereço da memória onde serão armazenados os dados lidos
tam – tamanho em bytes do tipo de dados a ser lido
qtde – quantidade de itens a serem lidos
fptr – corresponde ao ponteiro do arquivo a ser lido

Escrita no arquivo – grava informações no arquivo
sintaxe:

```
fwrite (p, tam, qtde, fptr);         // grava dados no arquivo
```

onde:
p – ponteiro do tipo void para o endereço da memória do dado a ser armazenado
tam – tamanho em bytes do tipo de dados a ser armazenado
qtde – quantidade de itens a serem gravados
fptr – corresponde ao ponteiro do arquivo a ser gravado

4.5.9 – Exemplos

Exemplo 1: Gravação no modo Formatado – Matriz

```
#include <stdio.h>
main()
{
int     mat[10], arq[10],i;
FILE *fptr=NULL;                        //declara ponteiro para arquivo

for(i=0;i<10;i++)
    {
    printf("\nmat[%i] = ",i);
    scanf("%i", mat+i);                 // armazena na memória RAM
    }

// rotina para gravar
if ((fptr = fopen("arqtext.txt", "w")) == NULL)
    printf("Erro ao abrir o arquivo");
else
    for(i=0;i<10;i++)                   //armazena no arquivo – 1 por vez
        fprintf(fptr, "%i   ", *(mat+i));   //cuidado - deixar espaço entre especificadores
fclose (fptr);
```

```
// rotina para ler
if ((fptr = fopen("arqtext.txt", "r")) == NULL)
    printf("Erro ao abrir o arquivo");
else
    {
    for(i=0;i<10;i++)
         fscanf(fptr, "%i", (arq+i));                    // retira do arquivo – 1 por vez
    fclose(fptr);
   }

//mostra que leu e armazenou corretamente
for(i=0;i<10;i++)
    printf("\n mat[%i] = %i\tarq[%i] = %i",i,*(mat+i),i,*(arq+i));
}//main
```

Exemplo 2 Gravação no modo Registro – Matriz

```
#include <stdio.h>
main()
{
int       mat[10],arq[10],i;
FILE *fptr=NULL;                                        //declara ponteiro para arquivo

for(i=0;i<10;i++)
    {
    printf("\nmat[%i] = ",i);
    scanf("%i", mat+i);                                 // armazena na memória RAM
    }

// rotina para gravar
if ((fptr = fopen("arqbin.bin", "wb")) == NULL)
    printf("Erro ao abrir o arquivo");
else
    fwrite(mat,sizeof(int),10,fptr);                    // armazena no arquivo – todos de 1 vez
fclose (fptr);

// rotina para ler
if ((fptr = fopen("arqbin.bin", "rb")) == NULL)
    printf("Erro ao abrir o arquivo");
else
    {
    fread(arq,sizeof(int),10,fptr);                     // retira do arquivo – todos de 1 vez
    fclose(fptr);
   }

//mostra que leu e armazenou corretamente
for(i=0;i<10;i++)
    printf("\n mat[%i] = %i\tarq[%i] = %i",i,*(mat+i),i,*(arq+i));
}//main
```

Exemplo 3: Gravação no modo Formatado – Estrutura

```
#include <stdio.h>
typedef     struct dados{
    char    produto[30];
    int     num;
    float   preco;
};

main()
{
dados  p1,p2;
FILE  *fptr=NULL                                        // ponteiro para arquivo

printf("\n Digite produto, registro e preco:");
scanf("%s    %i    %f", p1.produto, &(p1.num), &(p1.preco));        // armazena na RAM

// rotina para gravar
if ((fptr = fopen("arqtext.txt", "w")) == NULL)
        printf("Erro ao abrir o arquivo");
else
        fprintf(fptr, "%s    %i    %f", p1.produto, p1.num, p1.preco);     // armazena no arquivo
fclose (fptr);

// rotina para ler
if ((fptr = fopen("arqtext.txt", "r")) == NULL)
        printf("Erro ao abrir o arquivo");
else
        {
        fscanf(fptr, "%s    %i    %f", p2.produto, &(p2.num), &(p2.preco));     // retira do arquivo
        fclose(fptr);
        }

//mostra que leu e armazenou corretamente
printf("\n %s = %s",p1.produto, p2.produto);
printf("\n %i = %i", p1.num, p2.num);
printf("\n %f = %f", p1.preco, p2.preco);
}//main
```

Exemplo 4 Gravação no modo Registro – Estrutura

```
#include <stdio.h>
typedef     struct dados{
    char    produto[30];
    int     num;
    float   preco;
};

main()
{
dados  p1,p2;
```

```
FILE *fptr=NULL;                                        //declara ponteiro para arquivo

printf("\n Digite produto, registro e preco:");
scanf("%s      %i      %f", p1.produto, &(p1.num), &(p1.preco));        // armazena na RAM

// rotina para gravar
if ((fptr = fopen("arqbin.bin", "wb")) == NULL)
        printf("Erro ao abrir o arquivo");
else
        fwrite(&p1, sizeof (dados), 1, fptr);                    // armazena no arquivo
fclose (fptr);

// rotina para ler
if ((fptr = fopen("arqbin.bin", "rb")) == NULL)
        printf("Erro ao abrir o arquivo");
else
        {
        fread (&p2, sizeof (dados), 1, fptr);                    // retira do arquivo
        fclose(fptr);
        }

//mostra que leu e armazenou corretamente
printf("\n %s = %s", p1.produto, p2.produto);
printf("\n %i = %i", p1.num, p2.num);
printf("\n %f = %f", p1.preco, p2.preco);
}//main
```

4.6 – Acesso Aleatório – Alto-Nível

4.6.1 – Ponteiro para arquivo

Um ponteiro para o tipo FILE (arquivo) aponta para um byte particular chamado **posição atual.** Cada vez que lemos ou gravamos qualquer coisa no arquivo, o ponteiro é movido para o fim dessa informação, portanto a leitura ou armazenamento de novo dado, começará deste ponto.

» **Quando o arquivo é aberto, o ponteiro é fixado no início do mesmo. Apenas se abrirmos com a opção "a" (append), é que ele será posicionado no fim do arquivo.**

4.6.2 – Função fseek( )

Permite a movimentação do ponteiro para arquivo (posição atual).

Arquivo de cabeçalho: stdio.h

sintaxe:

```
fseek (fptr, offset, modo);
```

onde:
fptr – corresponde ao ponteiro para arquivo
offset – consiste no número de bytes de deslocamento, a partir do "modo" – deve ser do tipo **long int**
modo – especifica a posição desejada

Modo	Ação
0 ou SEEK_SET	Começo do arquivo
1 ou SEEK_CUR	Posição corrente do ponteiro
2 ou SEEK_END	Fim do arquivo

4.6.3 – Função ftell( )

Retorna a posição do ponteiro de um arquivo **binário** em relação ao seu começo.

Arquivo de cabeçalho: stdio.h

sintaxe:

```
pos = ftell (fptr);
```

onde:
pos – número de bytes do começo do arquivo até a posição atual – deve ser do tipo **long**
fptr – corresponde ao ponteiro para arquivo

4.6.4 – Função rewind( )

Reposiciona o ponteiro de um arquivo no início.

Arquivo de cabeçalho: stdio.h

sintaxe:

```
void    rewind (fptr);
```

onde:
fptr – corresponde ao ponteiro para arquivo

4.6.5 – Exemplo

```
#include <stdio.h>

typedef     struct  liv  {
        char    titulo[50];
        int     regnum;
        float   preço;
}liv;
```

```
main( )
{
liv             *livro=NULL;
FILE            *fptr=NULL;
int             numreg;
long int        offset;

if((livro = (liv) realloc (livro, 1*sizeof(liv))) == NULL)     //aloca espaço na memória
        {
        printf("\n Erro impossível alocar memória");
        return;
        }
if ((fptr = fopen(“livros.rec”, “rb”)) = = NULL)
        {
        printf("\n Impossível abrir o arquivo");
        exit();
        }

printf(“\n Digite o número do registro:”);
scanf(“%i”, &numreg);
offset = numreg * sizeof(struct liv);
fseek(fptr, offset, 0)
fread(livro, sizeof(struct liv), 1, fptr);                  //posso ler a mesma qtde alocada de memória
printf("\n   Titulo: %s",livro->titulo);
printf("\n   Numero do registro: %i",livro->regnum);
printf("\n   Preço: %.2f\n",livro->preco);
fclose(fptr);
}// main
```

4.6.6 – Exercícios

1. Dado a estrutura abaixo, implementar uma rotina de cadastro com alocação dinâmica, deve-se consultar o usuário para continuar. Caso não deseje mais cadastrar, salve todos os itens no arquivo. O registro deve ser gerado automaticamente pelo sistema. Não esquecer de conferir se já existem elementos armazenados em arquivo, se sim, calcular o número do próximo registro.

```
struct agenda
        {
        int reg;
        char nome[80];
        float nota;
        };
```

2. Idem ao anterior, porém o registro deve ser gravado cada vez que é criado, ou seja, a cada item cadastrado, deve-se salvar em arquivo.

3. Fazer um programa que aloca espaço para todos os itens gravados no arquivo anterior, lê todos na memória e busca qual será alterado (registro deve ser escolhido pelo usuário). Após a alteração grava os itens novamente. **DICA:** utilizar o modo de abertura do arquivo: **rb+**

4. Idem ao anterior, porém, o programa deve alocar espaço para apenas 1 item. Deve-se buscar o registro a ser alterado (que deve ser escolhido pelo usuário). Após a alteração gravar apenas o registro alterado.

5. Refazer o programa para **Sistema de Conta Bancária**, com gravação em arquivo.

6. Refazer o programa para **Diário Eletrônico**, com gravação em arquivo.

7. Refazer o programa para **Controle de Hotel,** com gravação em arquivo.