Introdução à Programação
AULA 11 – Arquivos

Profª. Glaucia M. M. Campos
glauciamelissa@uern.br

Universidade do Estado do Rio Grande do Norte - CaN

Arquivos

- Por que usar arquivos?
  - Permitem armazenar grande quantidade de informação
  - Persistência dos dados (disco)
  - Acesso aos dados poder ser não seqüencial
  - Acesso concorrente aos dados (mais de um programa pode usar os dados ao mesmo tempo).

Arquivos

- Basicamente, a linguagem C trabalha com dois tipos de arquivos: texto e binários.
- Arquivo texto
  - Armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de textos simples como o Bloco de Notas.
  - Os dados são gravados como caracteres de 8 bits. Ex.: Um número inteiro de 32 bits com 8 dígitos ocupará 64 bits no arquivo (8 bits por dígito).

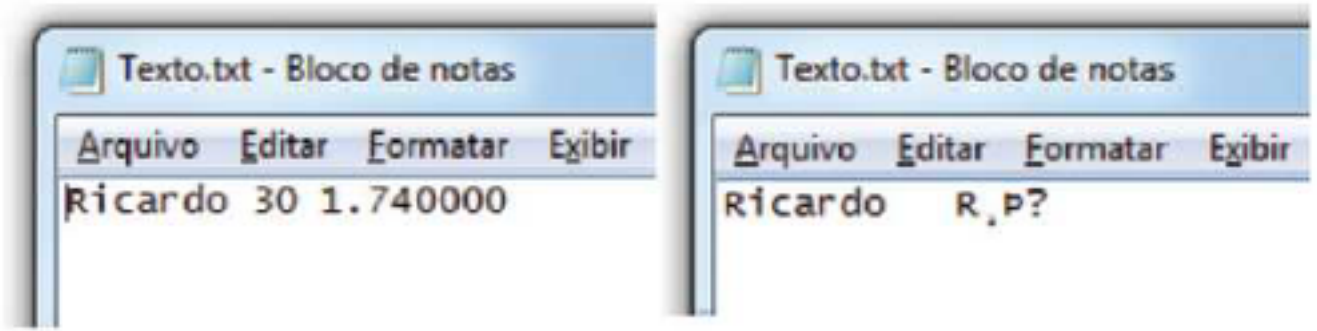
Arquivos

- Arquivo binário
  - Armazena uma sequência de bits que está sujeita as convenções dos programas que o gerou. Ex: arquivos executáveis, arquivos compactados, arquivos de registros, etc.
  - Os dados são gravados na forma binária (do mesmo modo que estão na memória). Ex.: um número inteiro de 32 bits com 8 dígitos ocupará 32 bits no arquivo.

Tipos de Arquivos

- Os dois tipos de arquivos abaixo possuem os mesmos dados

```
char nome[20] = "Ricardo";
int i = 30;
float a = 1.74;
```



Arquivo Texto **Arquivo Binário**

Tipos de Arquivos

- A linguagem C possui uma série de funções para manipulação de arquivos, cujos protótipos estão reunidos na biblioteca padrão de estrada e saída:

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

Manipulando Arquivos

- A linguagem C não possui funções que automaticamente leiam todas as informações de um arquivo
  - Suas funções se limitam a abrir/fechar e ler caracteres/bytes
  - É tarefa do programador criar a função que lerá um arquivo de uma maneira específica.

Manipulando Arquivos

- Todas as funções de manipulação de arquivos trabalham com o conceito de "ponteiro de arquivo". Podemos declarar um ponteiro de arquivo da seguinte maneira:

```
FILE *p;
```

- p é o ponteiro para arquivos que nos permitirá manipular arquivos no C.

Abrindo Arquivos

- Para a abertura de um arquivo, usa-se a função fopen

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

- O parâmetro **nome_arquivo** determina qual arquivo deverá ser aberto, sendo que o mesmo deve ser válido no sistema operacional que estiver sendo utilizado.

Abrindo Arquivos

- No parâmetro nome_arquivo pode-se trabalhar com
- caminhos absolutos ou relativos.
  - Caminho absoluto: descrição de um caminho desde o diretório raiz. (C:\\Projetos\\dados.txt)
  - Caminho relativo: descrição de um caminho desde o diretório corrente (onde o programa está salvo) (arq.txt; ..\\dados.txt)

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

Abrindo Arquivos

Modo	Arquivo	Função
"r"	Texto	Leitura. Arquivo deve existir.
"w"	Texto	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a"	Texto	Escrita. Os dados serão adicionados no fim do arquivo ("append").
"rb"	Binário	Leitura. Arquivo deve existir.
"wb"	Binário	Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"ab"	Binário	Escrita. Os dados serão adicionados no fim do arquivo ("append").
"r+"	Texto	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
"w+"	Texto	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a+"	Texto	Leitura/Escrita. Os dados serão adicionados no fim do arquivo ("append").
"r+b"	Binário	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
"w+b"	Binário	Leitura/Escrita. Cria arquivo se não houver. Apaga o anterior se ele existir.
"a+b"	Binário	Leitura/Escrita. Os dados serão adicionados no fim do arquivo ("append").

Abrindo Arquivos

- Um arquivo binário pode ser aberto para escrita utilizando o seguinte conjunto de comandos:
  - A condição fp==NULL testa se o arquivo foi aberto com sucesso. No caso de erro, a função fopen() retorna um ponteiro nulo (NULL).

```
int main(){
    FILE *fp;
    fp = fopen("exemplo.bin","wb");
    if(fp == NULL)
        printf("Erro na abertura do arquivo.\n");

    fclose(fp);

    return 0;
}
```

Erro ao abrir Arquivos

- Caso o arquivo não tenha sido aberto com sucesso
  - Provavelmente o programa não poderá continuar a executar;
  - Nesse caso, utilizamos a função exit(), presente na biblioteca <stdlib.h>, para abortar o programa

```
void exit(int codigo_de_retorno);
```

Erro ao abrir Arquivos

- A função exit() pode ser chamada de qualquer ponto no programa e faz com que o programa termine e retorne, para o sistema operacional, o código_de_retorno.
- A convenção mais usada é que um programa retorne
  - zero no caso de um término normal
  - um número diferente de zero, no caso de ter ocorrido um problema

Erro ao abrir Arquivos

```
int main(){
    FILE *fp;
    fp = fopen("exemplo.bin","wb");
    if(fp == NULL){
        printf("Erro na abertura do arquivo\n");
        system("pause");
        exit(1);
    }
    fclose(fp);

    return 0;
}
```

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 seqüencial, 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

Fechando Arquivo

- Sempre que terminamos de usar um arquivo que abrimos, devemos fechá-lo. Para isso usa-se a função fclose()
- O ponteiro fp passado à função fclose() determina o arquivo a ser fechado. A função retorna zero no caso de sucesso.

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

Fechando Arquivo

- Por que devemos fechar o arquivo?
  - Ao fechar um arquivo, todo caractere que tenha permanecido no "buffer" é gravado.
  - O "buffer“ é uma região de memória que armazena temporariamente os caracteres a serem gravados em disco imediatamente. Apenas quando o "buffer" está cheio é que seu conteúdo é escrito no disco.

Fechando Arquivo

- Por que utilizar um "buffer"?? Eficiência!
  - Para ler e escrever arquivos no disco temos que posicionar a cabeça de gravação em um ponto específico do disco.
  - Se tivéssemos que fazer isso para cada caractere lido/ escrito, a leitura/escrita de um arquivo seria uma operação muita lenta.
  - Assim a gravação só é realizada quando há um volume razoável de informações a serem gravadas ou quando o arquivo for fechado.
- A função exit() fecha todos os arquivos que um programa tiver aberto.

Escrita/Leitura de Arquivo

- Uma vez aberto um arquivo, podemos ler ou escrever nele.
- Para tanto, a linguagem C conta com uma série de funções de leitura/escrita que variam de funcionalidade para atender as diversas aplicações.

Escrita/Leitura de Caractere

- A maneira mais fácil de se trabalhar com um arquivo é a leitura/escrita de um único caractere.
- A função mais básica de entrada de dados é a função **fputc** (put character).

```
int fputc (int ch, FILE *fp);
```

- Cada invocação dessa função grava um único caractere ch no arquivo especificado por fp.

Escrita/Leitura de Caractere

```c
int main(){
    FILE *arq;
    char string[100];
    int i;
    arq = fopen("arquivo.txt","w");
    if(arq == NULL){
        printf("Erro na abertura do arquivo");
        system("pause");
        exit(1);
    }
    printf("Entre com a string a ser gravada no arquivo:");
    gets(string);
    //Grava a string, caractere a caractere
    for(i = 0; i < strlen(string); i++)
        fputc(string[i], arq);
    fclose(arq);

    return 0;
}
```

Escrita/Leitura de Caractere

- A função fputc também pode ser utilizada para escrever um caractere na tela.
- Nesse caso, é necessário mudar a variável que aponta para o local onde será gravado o caractere:
- Por exemplo, fputc ('*', stdout) exibe um * na tela do monitor (dispositivo de saída padrão).

```
int main(){
    fputc ('*',stdout);

    return 0;
}
```

Escrita/Leitura de Caractere

- Da mesma maneira que gravamos um único caractere no arquivo, também podemos ler um único caractere.
- A função correspondente de leitura de caracteres é fgetc (get character).

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

Escrita/Leitura de Caractere

- Cada chamada da função fgetc lê um único caractere do arquivo especificado
  - Se fp aponta para um arquivo, então fgetc(fp) lê o caractere atual no arquivo e se posiciona para ler o próximo caractere do arquivo.
  - Lembre-se, a leitura em arquivos é parecida com escrever em uma máquina de escrever

```
char c;
c = fgetc(fp);
```

Escrita/Leitura de Caractere

```c
int main(){
    FILE *arq;
    char c;
    arq = fopen("arquivo.txt","r");
    if(arq == NULL){
        printf("Erro na abertura do arquivo");
        system("pause");
        exit(1);
    }
    int i;
    for(i = 0; i < 5; i++){
        c = fgetc(arq);
        printf("%c",c);
    }
    fclose(arq);

    return 0;
}
```

Escrita/Leitura de Caractere

- Similar ao que acontece com a função fputc, a função fgetc também pode ser utilizada para a leitura do teclado (dispositivo de entrada padrão):
- Nesse caso, fgetc(stdin) lê o próximo caractere digitado no teclado.

```c
int main(){
    char ch;
    ch = fgetc(stdin);

    printf("%c\n",ch);

    return 0;
}
```

Escrita/Leitura de Caractere

- O que acontece quando fgetc tenta ler o próximo caractere de um arquivo que já acabou?
- Precisamos que a função retorne algo indicando o arquivo acabou.
- Porém, todos os 256 caracteres são "válidos"!

Escrita/Leitura de Caractere

- Para evitar esse tipo de situação, fgetc não devolve um char mas um int:

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

- O conjunto de valores do char está contido dentro do conjunto do int.
- Se o arquivo tiver acabado, fgetc devolve um valor int que não possa ser confundido com um char

Escrita/Leitura de Caractere

- Assim, se o arquivo não tiver mais caracteres, fgetc devolve o valor −1
- Mais exatamente, fgetc devolve a constante EOF (end of file), que está definida na biblioteca stdio.h. Em muitos computadores o valor de EOF é −1.

```
char c;
c = fgetc(fp);
if (c == EOF)
   printf ("O arquivo terminou!\n");
```

Escrita/Leitura de Caractere

```c
int main(){
    FILE *arq;
    char c;
    arq = fopen("arquivo.txt","r");
    if(arq == NULL){
        printf("Erro na abertura do arquivo");
        system("pause");
        exit(1);
    }
    while((c = fgetc(arq)) != EOF)
        printf("%c",c);

    fclose(arq);

    return 0;
}
```

Fim do Arquivo

- Assim, se o arquivo não tiver mais caracteres, fgetc devolve o valor −1
- Mais exatamente, fgetc devolve a constante EOF (end of file), que está definida na biblioteca stdio.h. Em muitos computadores o valor de EOF é −1.

```
char c;
c = fgetc(fp);
if (c == EOF)
   printf ("O arquivo terminou!\n");
```

Fim do Arquivo

- Como visto, EOF ("End of file") indica o fim de um arquivo.
- No entanto, podemos também utilizar a função feof para verificar se um arquivo chegou ao fim, cujo protótipo é

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

- No entanto, é muito comum fazer mau uso dessa função!

Fim do Arquivo

- Um mau uso muito comum da função feof() é usá-la para terminar um loop
- Mas por que isso é um mau uso??

```
int main{
    int i, n;
    FILE *F = fopen("teste.txt","r");
    if(arq == NULL) {
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    while(!feof(F)){
        fscanf(F,"%d",&n);
        printf("%d\n",n);
    }
    fclose(F);

    return 0;
}
```

Fim do Arquivo

- Vamos ver a descrição da função feof()
  - A função feof() testa o indicador de fim de arquivo para o fluxo apontado por fp
  - A função retona um valor inteiro diferente de zero se, e somente se, o indicador de fim de arquivo está marcado para fp
- Ou seja, a função testa o indicador de fim de arquivo, não o próprio arquivo

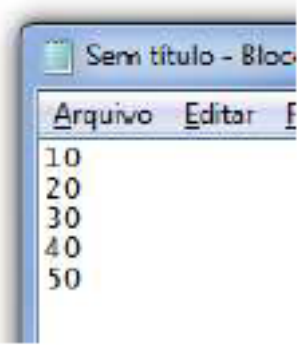
```
int feof(FILE *fp);
```

Fim do Arquivo

- Isso significa que outra função é responsável por alterar o indicador para indicar que o EOF foi alcançado
  - A maioria das funções de leitura irá alterar o indicador após ler todos os dados, e então realizar uma nova leitura resultando em nenhum dado, apenas o EOF
- Como resolver isso devemos evitar o uso da função feof() para testar um loop e usá-la para testar se uma leitura alterou o indicador de fim de arquivo

Fim do Arquivo

- Para entender esse problema do mau uso da funções feof(), considere que queiramos ler todos os números contidos em um arquivo texto como o mostrado abaixo



Fim do Arquivo

Mau uso da função feof()

```
int main{
    int i, n;
    FILE *F = fopen("teste.txt","r");
    if(arq == NULL) {
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    while(!feof(F)){
        fscanf(F,"%d",&n);
        printf("%d\n",n);
    }
    fclose(F);

    return 0;
}

Saída: 10 20 30 40 50 50
```

Bom uso da função feof()

```
int main{
    int i, n;
    FILE *F = fopen("teste.txt","r");
    if(arq == NULL) {
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    while(1){
        fscanf(F,"%d",&n);
        if(feof(F))
            break;
        printf("%d ",n);
    }
    fclose(F);

    return 0;
}

Saída: 10 20 30 40 50
```

Escrita/Leitura de Strings

- Até o momento, apenas caracteres isolados puderam ser escritos em um arquivo.
- Porém, existem funções na linguagem C que permitem ler/escrever uma seqüência de caracteres, isto é, uma string.
  - fputs()
  - fgets()

Escrita/Leitura de Strings

- Basicamente, para se escrever uma string em um arquivo usamos a função fputs:

```
int fputs(char *str, FILE *fp);
```

- Esta função recebe como parâmetro um array de caracteres (string) e um ponteiro para o arquivo no qual queremos escrever.

Escrita/Leitura de Strings

- Retorno da função
  - Se o texto for escrito com sucesso um valor inteiro diferente de zero é retornado.
  - Se houver erro na escrita, o valor EOF é retornado.
- Como a função fputc, fputs também pode ser utilizada para escrever uma string na tela:

```
int main(){
    char texto[30] = "Hello World\n";
    fputs(texto, stdout);

    return 0;
}
```

Escrita/Leitura de Strings

```
int main(){
    char str[20] = "Hello World!";
    int result;
    FILE *arq;
    arq = fopen("ArqGrav.txt","w");
    if(arq == NULL) {
        printf("Problemas na CRIACAO do arquivo\n");
        system("pause");
        exit(1);
    }
    result = fputs(str,arq);
    if(result == EOF)
        printf("Erro na Gravacao\n");
    fclose(arq);

    return 0;
}
```

Escrita/Leitura de Strings

- Da mesma maneira que gravamos uma cadeia de caracteres no arquivo, a sua leitura também é possível.
- Para se ler uma string de um arquivo podemos usar a função fgets() cujo protótipo é:

```
char *fgets(char *str, int tamanho,FILE *fp);
```

Escrita/Leitura de Strings

- A função fgets recebe 3 parâmetros
  - str: onde o valor lido será armazenado, str
  - tamanho: o número máximo de caracteres a serem lidos
  - fp: ponteiro que está associado ao arquivo de onde a string será lida.
- E retorna
  - NULL em caso de erro ou fim do arquivo
  - O ponteiro para o primeiro caractere recuperado em str

```
char *fgets(char *str, int tamanho,FILE *fp);
```

Escrita/Leitura de Strings

- Funcionamento da função fgets
  - A função lê a string até que um caractere de nova linha seja lido ou tamanho-1 caracteres tenham sido lidos.
  - Se o caractere de nova linha ('\n') for lido, ele fará parte da string, o que não acontecia com gets.
  - A string resultante sempre terminará com '\0' (por isto somente tamanho-1 caracteres, no máximo, serão lidos).
  - Se ocorrer algum erro, a função devolverá um ponteiro nulo em str.

Escrita/Leitura de Strings

- A função fgets é semelhante à função gets, porém, com as seguintes vantagens:
  - Pode fazer a leitura a partir de um arquivo de dados e incluir o caractere de nova linha "\n" na string
  - Específica o tamanho máximo da string de entrada
  - Isso evita estouro no buffer

Escrita/Leitura de Strings

```
int main(){
    char str[20];
    char *result;
    FILE *arq;
    arq = fopen("ArqGrav.txt","r");
    if(arq == NULL) {
        printf("Problemas na ABERTURA do arquivo\n");
        system("pause");
        exit(1);
    }
    result = fgets(str,13,arq);
    if(result == NULL)
        printf("Erro na leitura\n");
    else
        printf("%s",str);
    fclose(arq);

    return 0;
}
```

Escrita/Leitura de Strings

- Vale lembrar que o ponteiro fp pode ser substituído por stdin, para se fazer a leitura do teclado:

```
int main(){
    char nome[30];
    printf("Digite um nome: ");
    fgets(nome, 30, stdin);
    printf("O nome digitado foi: %s",nome);

    return 0;
}
```

Escrita/Leitura de Blocos de Dados

- Além da leitura/escrita de caracteres e seqüências de caracteres, podemos ler/escrever blocos de dados.
- Para tanto, temos duas funções
  - **fwrite()**
  - **fread()**

Escrita/Leitura de Blocos de Dados

- A função **fwrite** é responsável pela escrita de um bloco de dados da memória em um arquivo
- Seu protótipo é:

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

Escrita/Leitura de Blocos de Dados

- A função **fwrite** recebe 4 argumentos:
  - **buffer**: ponteiro para a região de memória na qual estão os dados;
  - **numero_de_bytes**: tamanho de cada posição de memória a ser escrita;
  - **count**: total de unidades de memória que devem ser escritas;
  - **fp**: ponteiro associado ao arquivo onde os dados serão escritos.

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

Escrita/Leitura de Blocos de Dados

- Note que temos dois valores numéricos
  - numero_de_bytes
  - count
- Isto significa que o número total de bytes escritos é:
  - numero_de_bytes * count
- Como retorno, temos o número de unidades efetivamente escritas.
- Este número pode ser menor que count quando ocorrer algum erro.

Escrita/Leitura de Blocos de Dados

```
int main(){
    FILE *arq;
    arq = fopen("ArqGrav.txt","wb");

    char str[20] = "Hello World!";
    float x = 5;
    int v[5] = {1,2,3,4,5};
    //grava a string toda no arquivo
    fwrite(str,sizeof(char),strlen(str),arq);
    //grava apenas os 5 primeiros caracteres da string
    fwrite(str,sizeof(char),5,arq);
    //grava o valor de x no arquivo
    fwrite(&x,sizeof(float),1,arq);
    //grava todo o array no arquivo (5 posições)
    fwrite(v,sizeof(int),5,arq);
    //grava apenas as 2 primeiras posições do array
    fwrite(v,sizeof(int),2,arq);
    fclose(arq);

    return 0;
}
```

Escrita/Leitura de Blocos de Dados

- A função **fread** é responsável pela leitura de um bloco de dados de um arquivo
- Seu protótipo é:

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

Escrita/Leitura de Blocos de Dados

- Como na função **fwrite**, **fread** retorna o número de itens escritos. Este valor será igual a **count** a menos que ocorra algum erro.

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

Escrita/Leitura de Blocos de Dados

```
char str1[20],str2[20];
float x;
int i,v1[5],v2[2];
//lê a string toda do arquivo
fread(str1,sizeof(char),12,arq);
str1[12] = '\0';
printf("%s\n",str1);
//lê apenas os 5 primeiros caracteres da string
fread(str2,sizeof(char),5,arq);
str2[5] = '\0';
printf("%s\n",str2);
//lê o valor de x do arquivo
fread(&x,sizeof(float),1,arq);
printf("%f\n",x);
//lê todo o array do arquivo (5 posições)
fread(v1,sizeof(int),5,arq);
for(i = 0; i < 5; i++)
    printf("v1[%d] = %d\n",i,v1[i]);
//lê apenas as 2 primeiras posições do array
fread(v2,sizeof(int),2,arq);
for(i = 0; i < 2; i++)
    printf("v2[%d] = %d\n",i,v2[i]);
```

Escrita/Leitura de Blocos de Dados

- Quando o arquivo for aberto para dados binários, **fwrite** e fread podem manipular qualquer tipo de dado:
  - int
  - float
  - double
  - array
  - struct

Escrita/Leitura por fluxo padrão

- As funções de fluxos padrão 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 por fluxo padrão

Ex: **fprintf**

```
printf("Total = %d",x);//escreve na tela
fprintf(fp,"Total = %d",x);//grava no arquivo fp
```

Ex: **fscanf**

```
scanf("%d",&x);//lê do teclado
fscanf(fp,"%d",&x);//lê do arquivo fp
```

Escrita/Leitura por fluxo padrão

- 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, deve-se utilizar as funções fread e fwrite.

Escrita/Leitura por fluxo padrão

```
int main(){
    FILE *arq;
    char nome[20] = "Ricardo";
    int I = 30;
    float a = 1.74;
    int result;
    arq = fopen("ArqGrav.txt","w");
    if(arq == NULL) {
        printf("Problemas na ABERTURA do arquivo\n");
        system("pause");
        exit(1);
    }
    fprintf(arq,"Nome: %s\n",nome);
    fprintf(arq,"Idade: %d\n",i);
    fprintf(arq,"Altura: %f\n",a);
    fclose(arq);

    return 0;
}
```

Escrita/Leitura por fluxo padrão

```c
int main(){
    FILE *arq;
    char texto[20], nome[20];
    int i;
    float a;
    int result;
    arq = fopen("ArqGrav.txt","r");
    if(arq == NULL) {
        printf("Problemas na ABERTURA do arquivo\n");
        system("pause");
        exit(1);
    }
    fscanf(arq,"%s%s",texto,nome);
    printf("%s %s\n",texto,nome);
    fscanf(arq,"%s %d",texto,&i);
    printf("%s %d\n",texto,i);
    fscanf(arq,"%s%f",texto,&a);
    printf("%s %f\n",texto,a);
    fclose(arq);

    return 0;
}
```

Movendo-se pelo arquivo

- De modo geral, o acesso a um arquivo é seqüencial. Porém, é possível fazer buscas e acessos randômicos em arquivos. Para isso, existe a função fseek:

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

- Basicamente, esta função move a posição corrente de leitura ou escrita no arquivo em tantos bytes, a partir de um ponto especificado.

Movendo-se pelo arquivo

- A função fseek recebe 3 parâmetros
  - fp: o ponteiro para o arquivo;
  - numbytes: é o total de bytes a partir de origem a ser pulado;
  - origem: determina a partir de onde os numbytes de movimentação serão contados.
- A função devolve o valor 0 quando bem sucedida

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

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

Movendo-se pelo arquivo

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 numbytes 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
- numbytes pode ser negativo quando usado com SEEK_CUR e SEEK_END

Movendo-se pelo arquivo

```
struct cadastro{ char nome[20], rua[20]; int idade;};
int main(){
    FILE *f = fopen("arquivo.txt","wb");

    struct cadastro c,cad[4] = {"Ricardo","Rua 1",31,
                                "Carlos","Rua 2",28,
                                "Ana","Rua 3",45,
                                "Bianca","Rua 4",32};
    fwrite(cad,sizeof(struct cadastro),4,f);
    fclose(f);

    f = fopen("arquivo.txt","rb");
    fseek(f,2*sizeof(struct cadastro),SEEK _ SET);
    fread(&c,sizeof(struct cadastro),1,f);
    printf("%s\n%s\n%d\n",c.nome,c.rua,c.idade);
    fclose(f);

    return 0;
}
```

Movendo-se pelo arquivo

- Outra opção de movimentação pelo arquivo é simplesmente retornar para o seu início.
- Para tanto, usa-se a função rewind:

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

Apagando um arquivo

- Além de permitir manipular arquivos, a linguagem C também permite apagá-lo do disco. Isso pode ser feito utilizando a função remove:

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

- Diferente das funções vistas até aqui, esta função recebe o caminho e nome do arquivo a ser excluído, e não um ponteiro para FILE.
- Como retorno temos um valor inteiro, o qual será igual a 0 se o arquivo for excluído com sucesso.

Apagando um arquivo

```c
int main(){
    int status;
    status = remove("ArqGrav.txt");
    if(status != 0){
        printf("Erro na remocao do arquivo.\n");
        system("pause");
        exit(1);
    }else
        printf("Arquivo removido com sucesso.\n");

    return 0;
}
```

Exercícios

1. Escreva um programa que:

   (a) Crie/abra um arquivo texto de nome “arq.txt”

   (b) Permita que o usuário grave diversos caracteres nesse arquivo, até que o usuário entre com o caractere ‘0’

   (c) Feche o arquivo

   Agora, abra e leia o arquivo, caractere por caractere, e escreva na tela todos os caracteres armazenados.

2. Faça um programa que receba do usuário um arquivo texto e mostre na tela quantas linhas esse arquivo possui.

3. Faça um programa que receba do usuário um arquivo texto e mostre na tela quantas letras são vogais.

4. Faça um programa que receba do usuário um arquivo texto e mostre na tela quantas letras são vogais e quantas são consoantes.

5. Faça um programa que receba do usuário um arquivo texto e um caracter. Mostre na tela quantas vezes aquele caractere ocorre dentro do arquivo.

Exercícios

6. Faça um programa que receba do usuário um arquivo texto e mostre na tela quantas vezes cada letra do alfabeto aparece dentro do arquivo.

7. Faça um programa que receba do usuário um arquivo texto. Crie outro arquivo texto contendo o texto do arquivo de entrada, mas com as vogais substituídas por '*'.

8. Faça um programa que leia o conteúdo de um arquivo e crie um arquivo com o mesmo conteúdo, mas com todas as letras minúsculas convertidas para maiúsculas. Os nomes dos arquivos serão fornecidos, via teclado, pelo usuário. A função que converte maiúscula para minúscula é o toupper(). Ela é aplicada em cada caractere da string.

9. Faça um programa que receba dois arquivos do usuário, e crie um terceiro arquivo com o conteúdo dos dois primeiros juntos (o conteúdo do primeiro seguido do conteúdo do segundo).

10. Faça um programa que receba o nome de um arquivo de entrada e outro de saída. O arquivo de entrada contém em cada linha o nome de uma cidade (ocupando 40 caracteres) e o seu número de habitantes. O programa deverá ler o arquivo de entrada e gerar um arquivo de saída onde aparece o nome da cidade mais populosa seguida pelo seu número de habitantes.

11. Faça um programa no qual o usuário informa o nome do arquivo e uma palavra, e retorne o número de vezes que aquela palavra aparece no arquivo.

Exercícios

12. Abra um arquivo texto, calcule e escreva o número de caracteres, o número de linhas e o número de palavras neste arquivo. Escreva também quantas vezes cada letra ocorre no arquivo (ignorando letras com acento). Obs.: palavras são separadas por um ou mais caracteres espaço, tabulação ou nova linha.

13. Faça um programa que permita que o usuário entre com diversos nomes e telefone para cadastro, e crie um arquivo com essas informações, uma por linha. O usuário finaliza a entrada com '0' para o telefone.

14. Dado um arquivo contendo um conjunto de nome e data de nascimento (DD MM AAAA, isto é, 3 inteiros em sequência), faça um programa que leia o nome do arquivo e a data de hoje e construa outro arquivo contendo o nome e a idade de cada pessoa do primeiro arquivo.

15. Faça um programa que receba como entrada o ano corrente e o nome de dois arquivos: um de entrada e outro de saída. Cada linha do arquivo de entrada contém o nome de uma pessoa (ocupando 40 caracteres) e o seu ano de nascimento. O programa deverá ler o arquivo de entrada e gerar um arquivo de saída onde aparece o nome da pessoa seguida por uma string que representa a sua idade.

    - Se a idade for menor do que 18 anos, escreva "menor de idade"
    - Se a idade for maior do que 18 anos, escreva "maior de idade"
    - Se a idade for igual a 18 anos, escreva "entrando na maior idade"

Exercícios

16. Faça um programa que recebe um vetor de 10 números, converta cada um desses números para binário e grave a sequência de 0s e 1s em um arquivo texto. Cada número deve ser gravado em uma linha.

17. Faça um programa que leia um arquivo que contenha as dimensões de uma matriz (linha e coluna), a quantidade de posições que serão anuladas, e as posições a serem anuladas (linha e coluna). O programa lê esse arquivo e, em seguida, produz um novo arquivo com a matriz com as dimensões dadas no arquivo lido, e todas as posições especificadas no arquivo ZERADAS e o restante recebendo o valor 1.
    Ex: arquivo "matriz.txt"

```
3 3 2 /*3 e 3 dimensões da matriz e 2 posições que serão anuladas*/
1 0
1 2  /*Posições da matriz que serão anuladas.
```

    arquivo "matriz_saida.txt"
    saída:

```
1 1 1
0 1 0
1 1 1
```

Exercícios

18. Faça um programa que leia um arquivo contendo o nome e o preço de diversos produtos (separados por linha), e calcule o total da compra.

19. Faça um programa que receba do usuário um arquivo que contenha o nome e a nota de diversos alunos (da seguinte forma: NOME: JOÃO NOTA: 8), um aluno por linha. Mostre na tela o nome e a nota do aluno que possui a maior nota.

20. Crie um programa que receba como entrada o número de alunos de uma disciplina. Aloque dinamicamente dois vetores para armazenar as informações a respeito desses alunos. O primeiro vetor contém o nome dos alunos e o segundo contém suas notas finais. Crie um arquivo que armazene, a cada linha, o nome do aluno e sua nota final. Use nomes com no máximo 40 caracteres. Se o nome não contém 40 caracteres, complete com espaço em branco.

21. Crie um programa que receba como entrada o número de alunos de uma disciplina. Aloque dinamicamente em uma estrutura para armazenar as informações a respeito desses alunos: nome do aluno e sua nota final. Use nomes com no máximo 40 caracteres. Em seguida, salve os dados dos alunos em um arquivo binário. Por fim, leia o arquivo e mostre o nome do aluno com a maior nota.

22. Faça um programa que recebe como entrada o nome de um arquivo de entrada e o nome de um arquivo de saída. O arquivo de entrada contém o nome de um aluno ocupando 40 caracteres e três inteiros que indicam suas notas. O programa deverá ler o arquivo de entrada e gerar um arquivo de saída onde aparece o nome do aluno e as suas notas em ordem crescente.

Exercícios

23. Escreva um programa que leia a profissão e o tempo de serviço (em anos) de cada um dos 5 funcionários de uma empresa e armazene-os no arquivo "emp.txt". Cada linha do arquivo corresponde aos dados de um funcionário. Utilize o comando fprintf(). Em seguida, leia o mesmo arquivo utilizando fscanf(). Apresente os dados na tela.

24. Implemente um controle simples de mercadorias em uma despensa doméstica. Para cada produto armazene um código numérico, descrição e quantidade atual. O programa deve ter opções para entrada e retirada de produtos, bem como um relatório geral e um de produtos não disponíveis. Armazene os dados em arquivo binário.

25. Faça um programa gerenciar uma agenda de contatos. Para cada contato armazene o nome, o telefone e o aniversário (dia e mês). O programa deve permitir

    (a) inserir contato
    (b) remover contato
    (c) pesquisar um contato pelo nome
    (d) listar todos os contatos
    (e) listar os contatos cujo nome inicia com uma dada letra
    (f) imprimir os aniversariantes do mês.

    Sempre que o programa for encerrado, os contatos devem ser armazenados em um arquivo binário. Quando o programa iniciar, os contatos devem ser inicializados com os dados contidos neste arquivo binário.

Exercícios

26. Crie um programa que declare uma estrutura para o cadastro de alunos.

    (a) Deverão ser armazenados, para cada aluno: matrícula, sobrenome (apenas um), e ano de nascimento.

    (b) Ao início do programa, o usuário deverá informar o número de alunos que serão armazenados

    (c) O programa deverá alocar dinamicamente a quantidade necessária de memória para armazenar os registros dos alunos.

    (d) O programa deverá pedir ao usuário que entre com as informações dos alunos.

    (e) Em seguida, essas informações deverão ser gravadas em um arquivo

    (f) Ao final, mostrar os dados armazenados e liberar a memória alocada.

    Ao iniciar o programa, forneça ao usuário uma opção para carregar os registros do arquivo para a memória do computador alocando dinamicamente a quantidade de memória necessária.
    Dica: para que o usuário possa entrar com novos dados, além dos que foram obtidos a partir do arquivo, use a função realloc() para realocar a quantidade de memória usada.

Exercícios

27. Faça um programa para gerenciar as notas dos alunos de uma turma salva em um arquivo. O programa deverá ter um menu contendo as seguintes opções:

    (a) Definir informações da turma;
    (b) Inserir aluno e notas;
    (c) Exibir alunos e médias;
    (d) Exibir alunos aprovados;
    (e) Exibir alunos reprovados;
    (f) Salvar dados em Disco;
    (g) Sair do programa (fim).

    Faça a rotina que gerencia o menu dentro do main, e para cada uma das opções deste menu, crie uma função específica.

28. Faça um programa que recebe como entrada o nome de um arquivo de entrada e o nome de um arquivo de saída. Cada linha do arquivo de entrada possui colunas de tamanho de 30 caracteres. No arquivo de saída deverá ser escrito o arquivo de entrada de forma inversa. Veja um exemplo:

    Arquivo de entrada:
    Hoje é dia de prova de AP
    A prova está muito fácil
    Vou tirar uma boa nota

    Arquivo de saída: Aton aob amu rarit uov
    Licáf otium átse avorp A
    PA ed avorp ed aid é ejoH

Exercícios

29. Codifique um programa que manipule um arquivo contendo registros descritos pelos seguintes campos: codigo_vendedor, nome_vendedor, valor_da_venda e mes.
A manipulação do arquivo em questão é feita através da execução das operações disponibilizadas pelo seguinte menu:

    - Criar o arquivo de dados;
    - Incluir um determinado registro no arquivo;
    - Excluir um determinado vendedor no arquivo;
    - Alterar o valor de uma venda no arquivo;
    - Imprimir os registros na saída padrão;
    - Excluir o arquivo de dados;
    - Finalizar o programa.

    Os registros devem estar ordenados no arquivo, de forma crescente, de acordo com as informações contidas nos campos codigo_vendedor e mes. Não deve existir mais de um registro no arquivo com mesmos valores nos campos codigo_vendedor e mês.

Referências

- Medina, Marco; Fertig, Cristina. Algoritmos e Programação: teoria e prática. São Paulo: Novatec Editora, 2006.
- Lopes, Anita; Garcia, Guto. Introdução à Programação: 500 algoritmos resolvidos. Rio de Janeiro: Editora Campus, 2002.
- Mizrani, Victorine Viviane. Treinamento em Linguagem C, Módulo 1. Editora Makron Books.
- Transparências modificadas do professor Dr. Flavio Luiz Cardeal Pádua, do Centro Federal de Educação Tecnológica de Minas Gerais
- Transparências modificadas do professor Robson Fidalgo, da UFRPE.
- Material disponível em: https://programacaodescomplicada.wordpress.com/complementar/
- Transparências de arquivos foram retiradas do material do professor André Backes, bem como os exercícios.