Disciplina:

Programação Computacional

Prof. Fernando Rodrigues
e-mail: fernandorodrigues@sobral.ufc.br

Aula 14: Programação em C

- Arquivos: Conceito de arquivos;
- Funções de manipulação de arquivos em C.

Definição

- Um arquivo, de modo abstrato, nada mais é do que uma coleção de bytes armazenados em um dispositivo de armazenamento secundário, que é geralmente, um HD, um CD, um DVD etc.
- Essa coleção de bytes pode ser interpretada como: Caracteres, palavras ou frases de um documento de texto; Campos e registros de uma tabela de banco de dados; Pixels de uma imagem etc.
- O que define o significado de um arquivo em particular é a maneira como as estruturas de dados estão organizadas e as operações são usadas por um programa para processar (ler ou escrever) esse arquivo.

Stream

- Embora cada dispositivo seja muito diferente, o sistema de arquivo com buffer trasforma-os em um dispositivo lógico chamado de buffer.

- Todas as streams comportam-se de forma semelhante. Mesmo trabalhando com arquivos ou dispositivos diferentes.

- Existem dois tipos de streams
    - Binária
    - Texto

Stream

- De texto:
  - É uma sequência de caracteres;
  - Um arquivo texto armazena caracteres que podem ser mostrados diretamente na tela ou modificados por um editor de texto simples como o Bloco de Notas;
  - A “conversão” dos dados em texto faz com que os arquivos texto sejam maiores. Além disso, suas operações de escrita e leitura consomem mais tempo em comparação às dos arquivos binários.
- Binária:
  - É uma sequência de bits / bytes com uma correspondência de um para um, com aqueles encontrados no dispositivo externo.
  - Um arquivo binário armazena uma sequência de bits que está sujeita às convenções dos programas que o gerou.

Arquivos Texto

- Os dados gravados em um arquivo texto são gravados exatamente como seriam impressos na tela. Por isso eles podem ser modificados por um editor de texto simples como o Bloco de Notas.

- No entanto, para que isso ocorra, os dados são gravados como caracteres de 8 bits utilizando a tabela ASCII. Ou seja, durante a gravação dos dados existe uma etapa de “conversão”.

Arquivos Texto (Exemplo)

- Para entender essa conversão dos dados em arquivos texto, imagine um número inteiro com oito dígitos: 12345678.

- Esse número ocupa 32 bits (4 bytes) na memória. Porém, quando for gravado em um arquivo texto, cada dígito será convertido em seu caractere ASCII, ou seja, 8 bits por dígito.

- Como resultado final, esse número ocupará 64 bits no arquivo texto, o dobro do seu tamanho na memória como binário.

Arquivos Binários

- Os dados gravados em um arquivo binário são gravados exatamente como estão organizados na memória do computador.

- Isso significa que não existe uma etapa de “conversão” dos dados.

- Portanto, suas operações de escrita e leitura são mais rápidas do que as realizadas em arquivos texto.

Arquivos Binários (Exemplo)

- Voltemos ao nosso número inteiro com 8 dígitos: 12345678.

- Esse número ocupa 32 bits (ou 4 bytes) na memória. Quando for gravado em um arquivo binário, o conteúdo da memória será copiado diretamente para o arquivo, sem conversão.

- Como resultado final, esse número ocupará os mesmos 32 bits (4 bytes) no arquivo binário.

Stream e arquivo

- O sistema de E/S de C provê um nível de abstração entre o programador e o dispositivo acessado.

- Essa abstração é chamada de stream, e o dispositivo real é chamado de arquivo.

Arquivos

- Em C, um “arquivo” pode ser qualquer coisa, desde um arquivo em disco a uma impressora ou terminal.
- Você deve associar uma stream com um arquivo específico, realizando uma abertura de arquivo.
- Uma vez aberto, informações podem ser trocadas entre o arquivo e seu programa.

Fundamentos do sistema de arquivos

- O sistema de arquivos é formado por um conjunto de diversas funções inter-relacionadas.

- Essas funções estão contidas na biblioteca “stdio.h”.

Fundamentos do sistema de arquivos

- Já vimos como podemos exibir e ler dados para / do usuário através do teclado e da tela; (printf, scanf etc)

- Agora veremos também como ler e gravar dados em arquivos.

- Assim como as funções de entrada/saída padrão (teclado e tela), as funções de entrada/saída em arquivos estão declaradas no cabeçalho *stdio.h.*
  - "STanDard Input-Output".

Fundamentos do sistema de arquivos

- Na manipulação de um arquivo, há basicamente três etapas que precisam ser realizadas:

    1. Abrir o arquivo;
    2. Ler e/ou gravar os dados desejados;
    3. Fechar o arquivo.

- Em C, todas as operações realizadas com arquivos envolvem seu *identificador de fluxo*, que é uma variável do tipo FILE *.
- Todas as funções de manipulação de arquivos trabalham com o conceito de “ponteiro de arquivo”.

Ponteiro de Arquivo: FILE *

- No exemplo abaixo, “fp” é um ponteiro de arquivo, ou seja, é o ponteiro que nos permitirá manipular arquivos na linguagem C.

```
FILE *fp;          // não se esqueça do asterisco!
```

- Um ponteiro de arquivo nada mais é do que um ponteiro para uma área na memória chamada de “buffer”. Nela se encontram vários dados sobre o arquivo aberto, como o seu nome e a posição atual.

Abrindo e fechando um arquivo

```
*fopen (char *nome_do_arquivo, char *modo_de_acesso);
```

- O nome do arquivo deve ser uma string com o caminho completo:
  - por exemplo, /usr/share/appname/app.conf
- ou C:\Documentos\nomes.txt
- ou o caminho em relação ao diretório atual (nomes.txt, ../app.conf) do arquivo que se deseja abrir ou criar.

Abrindo e fechando um arquivo

```
*fopen (char *nome_do_arquivo, char *modo_de_acesso);
```

- O modo de acesso é uma string que contém uma sequência de caracteres que dizem se o arquivo será aberto para gravação ou leitura.
- Depois de aberto o arquivo, você só poderá executar os tipos de ação previstos pelo modo de acesso: não poderá ler de um arquivo que foi aberto somente para

Modo	Significado
r	Abre o arquivo somente para leitura. O arquivo deve existir. (O *r* vem do inglês *read*, ler)
r+	Abre o arquivo para leitura e escrita. O arquivo deve existir.
w	Abre o arquivo somente para escrita no início do arquivo. Apagará o conteúdo do arquivo se ele já existir, criará um arquivo novo se não existir. (O *w* vem do inglês *write*, escrever)
w+	Abre o arquivo para escrita e leitura, apagando o conteúdo pré-existente.
a	Abre o arquivo para escrita no final do arquivo. Não apaga o conteúdo pré-existente. (O *a* vem do inglês *append*, adicionar, apender)
a+	Abre o arquivo para escrita no final do arquivo e leitura.

Abrindo e fechando um arquivo

- Modo de acesso:

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").

- O arquivo deve sempre ser aberto em um modo que permita executar as operações desejadas.

Abrindo e fechando um arquivo

- Abrindo / criando um arquivo binário para escrita e depois fechando o mesmo.

Exemplo: abrir um arquivo binário para escrita

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    FILE *fp;
05    fp = fopen("exemplo.bin","wb");
06    if(fp == NULL)
07        printf("Erro na abertura do arquivo.\n");
08
09    fclose(fp);
10    system("pause");
11    return 0;
12  }
```

Abrindo e fechando um arquivo

```
*fopen (char *nome_do_arquivo, char *modo_de_acesso);
```

- O valor de retorno da função fopen() é muito importante! Ele é o identificador do fluxo que você abriu e é só com ele que você conseguirá ler e escrever no arquivo aberto.
- Se houver um erro na abertura/criação do arquivo, a função retornará o valor **NULL**. O erro geralmente acontece por duas razões:
  - O arquivo não existe, caso tenha sido requisitado para leitura.
  - O usuário atual não tem permissão para abrir o arquivo com o modo de acesso pedido. Por exemplo, o arquivo é somente-leitura, ou está bloqueado para gravação por outro programa, ou pertence a outro usuário e não tem permissão para ser lido por outros.

Abrindo e fechando um arquivo

- Ao terminar de usar um arquivo, você deve fechá-lo. Isso é feito pela função fclose( ):

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

- O único argumento é o identificador do fluxo (retornado por fopen).

- O valor de retorno indica o sucesso da operação com o valor zero.

Abrindo e fechando um arquivo

- Por que devemos fechar um arquivo aberto?
  - Ao fechar um arquivo, todo caractere que tenha permanecido no buffer é gravado.
  - O buffer é uma área intermediária entre o arquivo no disco e o programa em execução. Trata-se de uma região de memória que armazena temporariamente os caracteres a serem gravados em disco.
  - Apenas quando o buffer está cheio é que seu conteúdo é escrito (automaticamente) no disco.
- A função exit() fecha todos os arquivos que um programa tiver aberto.

Abrindo e fechando um arquivo

- Por que utiliza-se buffer para escrever em arquivos?
  - O uso de um buffer é uma questão de eficiência.
  - Para ler e escrever arquivos no disco rígido é preciso posicionar a cabeça de gravação em um ponto específico do disco rígido. E isso consome tempo.
  - Se tivéssemos que fazer isso para cada caractere lido ou escrito, as operações de leitura e escrita de um arquivo seriam extremamente lentas.
  - Assim, a gravação só é realizada quando há um volume razoável de informações a serem gravadas ou quando o arquivo for fechado.

Abrindo e fechando um arquivo

- Abrindo / criando um arquivo texto para escrita.

```c
#include <stdio.h>

int main()
{
    FILE *fp;
    fp = fopen ("README", "w");
    if (fp == NULL) {
       printf ("Houve um erro ao abrir o arquivo.\n");
       return 1;
    }
    printf ("Arquivo README criado com sucesso.\n");
    fclose (fp);
    return 0;
}
```

Arquivos pré-definidos

- Na biblioteca padrão do C, existem alguns fluxos pré-definidos que não precisam (nem devem) ser abertos nem fechados:
  - **stdin**: dispositivo de entrada padrão (geralmente o teclado)
  - **stdout**: dispositivo de saída padrão (geralmente o vídeo)
  - **stderr**: dispositivo de saída de erro padrão (geralmente o vídeo)
  - **stdaux**: dispositivo de saída auxiliar (em muitos sistemas, associado à porta serial)
  - **stdprn**: dispositivo de impressão padrão (em muitos sistemas, associado à porta paralela)

Disciplina:

Programação Computacional

Prof. Fernando Rodrigues
e-mail: fernandorodrigues@sobral.ufc.br

Aula 14.1: Programação em C

- Funções de manipulação de arquivos.

Arquivos: Principais funções

Nome	Função
fopen()	Abre um arquivo
fclose()	Fecha um arquivo
fputc()	Escreve um caractere em um arquivo
fgetc()	Lê um caractere de um arquivo
fseek()	Posiciona o arquivo em um byte específico
fprintf()	É para um arquivo o que **printf()** é para o console
fscanf()	É para um arquivo o que **scanf()** é para o console
feof()	Devolve **verdadeiro** se o fim de arquivo for atingido
ferror()	Devolve **verdadeiro** se ocorreu um erro
rewind()	Recoloca o indicador de posição de arquivo no início do arquivo
remove()	Apaga um arquivo
fflush()	Descarrega um arquivo

Escrevendo em arquivos

- Para escrever em arquivos, há quatro funções, das quais três são análogas às usadas para saída padrão:

Saída padrão	Arquivos	Explicação
putchar	fputc	Imprime apenas um caractere.
puts	fputs	Imprime uma string diretamente, sem nenhuma formatação.
printf	fprintf	Imprime uma string formatada.
N/A	fwrite	Grava dados binários para um arquivo.

```
void fputc (int caractere, FILE *fluxo);
```

```
void fputs (char *string, FILE *fluxo);
```

```
void fprintf (FILE *fluxo, char *formatação, ...);
```

```
int fwrite (void *dados, int tamanho_do_elemento, int num_elementos, FILE *fluxo);
```

Escrevendo em arquivos - fputc()

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include <string.h>
04  int main(){
05    FILE *arq;
06    char string[100];
07    int i;
08    arq = fopen("arquivo.txt","w");
09    if(arq == NULL){
10      printf("Erro na abertura do arquivo");
11      system("pause");
12      exit(1);
13    }
14    printf("Entre com a string a ser gravada no arquivo:");
15    gets(string);
16    for(i = 0; i < strlen(string); i++)
17      fputc(string[i], arq);
18
19    fflush(arq);
20    fclose(arq);
21    system("pause");
22    return 0;
23  }
```

Escrevendo em arquivos - fputs()

Exemplo: escrevendo uma string em um arquivo com fputs()

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    char str[20] = "Hello World!";
05    int result;
06    FILE *arq;
07    arq = fopen("ArqGrav.txt","w");
08    if(arq == NULL) {
09      printf("Problemas na CRIACAO do arquivo\n");
10      system("pause");
11      exit(1);
12    }
13    result = fputs(str,arq);
14    if(result == EOF)
15      printf("Erro na Gravacao\n");
16
17    fclose(arq);
18    system("pause");
19    return 0;
20  }
```

Escrevendo em arquivos - fprintf()

Exemplo: usando a função fprintf()

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    FILE *arq;
05    char nome[20] = "Ricardo";
06    int I = 30;
07    float a = 1.74;
08    int result;
09    arq = fopen("ArqGrav.txt","w");
10    if(arq == NULL) {
11      printf("Problemas na ABERTURA do arquivo\n");
12      system("pause");
13      exit(1);
14    }
15    result = fprintf(arq,"Nome: %s\nIdade: %d\nAltura: %f\n",nome,i,a);
16    if(result < 0)
17      printf("Erro na escrita\n");
18    fclose(arq);
19    system("pause");
20    return 0;
21  }
```

Escrevendo em arquivos - fwrite()

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    FILE *arq;
05    arq = fopen("ArqGrav.bin","wb");
06    if(arq == NULL){
07      printf("Problemas na CRIACAO do arquivo\n");
08      system("pause");
09      exit(1);
10    }
11    int total_gravado, v[5] = {1,2,3,4,5};
12    //grava todo o array no arquivo (5 posições)
13    total_gravado = fwrite(v,sizeof(int),5,arq);
14    if(total_gravado != 5){
15      printf("Erro na escrita do arquivo!");
16      system("pause");
17      exit(1);
18    }
19    fclose(arq);
20    system("pause");
21    return 0;
22  }
```

Lendo de arquivos

- Novamente, há quatro funções, das quais três se assemelham às usadas para a saída padrão:

Saída padrão	Arquivos	Explicação
getchar	fgetc	Recebe apenas um caractere.
gets	fgets	Lê uma string (geralmente uma linha inteira).
scanf	fscanf	Recebe uma string formatada.
N/A	fread	Lê dados binários de um arquivo.

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

```
void fgets (char *string, int tamanho, FILE *fluxo);
```

```
void fscanf (FILE *fluxo, char *formatação, ...);
```

```
int fread (void *dados, int tamanho_do_elemento, int num_elementos, FILE *fluxo);
```

Lendo de arquivos - fgetc()

Cada chamada da função **fgetc()** lê um único caractere do arquivo especificado.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    FILE *arq;
05    char c;
06    arq = fopen("arquivo.txt","r");
07    if(arq == NULL){
08      printf("Erro na abertura do arquivo");
09      system("pause");
10      exit(1);
11    }
12    int i;
13    for(i = 0; i < 5; i++){
14      c = fgetc(arq);
15      printf("%c",c);
16    }
17    fclose(arq);
18    system("pause");
19    return 0;
20  }
```

Lendo de arquivos – fgetc() e EOF

Quando atinge o final de um arquivo, a função **fgetc()** devolve a constante **EOF** (**End Of File**), que está definida na biblioteca **stdio.h**. Em muitos computadores, o valor de **EOF** é definido como –1.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    FILE *arq;
05    char c;
06    arq = fopen("arquivo.txt","r");
07    if(arq == NULL){
08      printf("Erro na abertura do arquivo");
09      system("pause");
10      exit(1);
11    }
12    while((c = fgetc(arq)) != EOF)
13      printf("%c",c);
14    fclose(arq);
15    system("pause");
16    return 0;
17  }
```

Lendo de arquivos - fgets()

Exemplo: lendo uma string de um arquivo com fgets()

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    char str[20];
05    char *result;
06    FILE *arq;
07    arq = fopen("ArqGrav.txt","r");
08    if(arq == NULL) {
09      printf("Problemas na ABERTURA do arquivo\n");
10      system("pause");
11      exit(1);
12    }
13    result = fgets(str,13,arq);
14    if(result == NULL)
15      printf("Erro na leitura\n");
16    else
17      printf("%s",str);
18
19    fclose(arq);
20    system("pause");
21    return 0;
22  }
```

Lendo de arquivos - fgets()

- A função fgets() lê uma string do arquivo até que um caractere de nova linha (\n) seja lido ou “tamanho-1” caracteres tenham sido lidos.

- A string resultante de uma operação de leitura usando a função fgets() sempre terminará com a constante “\0” (por isso somente “tamanho-1” caracteres, no máximo, serão lidos).

- No caso de um caractere de nova linha (\n ou ENTER) ser lido antes de “tamanho-1” caracteres, ele fará parte da string.

Lendo de arquivos - fscanf()

Exemplo: usando a função fscanf()

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    FILE *arq;
05    char texto[20], nome[20];
06    int i;
07    float a;
08    int result;
09    arq = fopen("ArqGrav.txt","r");
10    if(arq == NULL) {
11      printf("Problemas na ABERTURA do arquivo\n");
12      system("pause");
13      exit(1);
14    }
15    fscanf(arq,"%s%s",texto,nome);
16    printf("%s %s\n",texto,nome);
17    fscanf(arq,"%s %d",texto,&i);
18    printf("%s %d\n",texto,i);
19    fscanf(arq,"%s%f",texto,&a);
20    printf("%s %f\n",texto,a);
21    fclose(arq);
22    system("pause");
23    return 0;
24  }
```

Lendo de arquivos - fread()

O valor do retorno da função **fread()** será igual ao valor de **count**, a menos que ocorra algum erro na leitura dos dados.

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    FILE *arq;
05    arq = fopen("ArqGrav.bin","rb");
06    if(arq == NULL){
07      printf("Problemas na ABERTURA do arquivo\n");
08      system("pause");
09      exit(1);
10    }
11    int i,total _ lido, v[5];
12    //lê 5 posições inteiras do arquivos
13    total _ lido = fread(v,sizeof(int),5,arq);
14    if(total _ lido != 5){
15      printf("Erro na leitura do arquivo!");
16      system("pause");
17      exit(1);
18    }else{
19      for(i = 0; i < 5; i++)
20        printf("v[%d] = %d\n",i,v[i]);
21    }
22    fclose(arq);
23    system("pause");
24    return 0;
25  }
```

Exercícios I

1) Escreva um programa que leia do usuário o nome de um arquivo texto. Em seguida, mostre na tela quantas linhas esse arquivo possui.

2) Escreva um programa que leia do usuário os nomes de dois arquivos texto. Crie um terceiro arquivo texto com o conteúdo dos dois primeiros juntos (o conteúdo do primeiro seguido do conteúdo do segundo).

3) Crie um programa para calcular e exibir o número de palavras contido em um arquivo texto. O usuário deverá informar o nome do arquivo.

4) Faça um programa que leia 100 números. Esse programa deverá, em seguida, armazenar esses números em um arquivo binário.

5) Elabore um programa que leia um arquivo binário contendo 100 números. Mostre na tela a soma desses números.

6) Crie um programa que leia um arquivo binário contendo uma quantidade qualquer de números. O primeiro número lido indica quantos valores existem no arquivo. Mostre na tela o maior e o menor valor lido.

Disciplina:

Programação Computacional

Prof. Fernando Rodrigues
e-mail: fernandorodrigues@sobral.ufc.br

Aula 14.2: Programação em C

- Funções Avançadas de manipulação de arquivos.

Forçando a escrita dos dados do buffer – fflush() (Exemplo)

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include <string.h>
04  int main(){
05    FILE *arq;
06    char string[100];
07    int i;
08    arq = fopen("arquivo.txt","w");
09    if(arq == NULL){
10      printf("Erro na abertura do arquivo");
11      system("pause");
12      exit(1);
13    }
14    printf("Entre com a string a ser gravada no arquivo:");
15    gets(string);
16    for(i = 0; i < strlen(string); i++)
17      fputc(string[i], arq);
18
19    fflush(arq);
20    fclose(arq);
21    system("pause");
22    return 0;
23  }
```

Gravando uma string e o seu tamanho

Exemplo: gravando uma string e o seu tamanho

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include <string.h>
04  int main(){
05    FILE *arq;
06    arq = fopen("ArqGrav.txt","wb");
07    if(arq == NULL){
08      printf("Erro\n");
09      system("pause");
10      exit(1);
11    }
12    char str[20] = "Hello World!";
13    int t = strlen(str);
14    fwrite(&t,sizeof(int),1,arq);
15    fwrite(str,sizeof(char),t,arq);
16    fclose(arq);
17    system("pause");
18    return 0;
19  }
```

Lendo uma string e o seu tamanho

Exemplo: lendo uma string e o seu tamanho

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    FILE *arq;
05    arq = fopen("ArqGrav.txt","rb");
06    if(arq == NULL){
07      printf("Erro\n");
08      system("pause");
09      exit(1);
10    }
11    char str[20];
12    int t;
13    fread(&t,sizeof(int),1,arq);
14    fread(str,sizeof(char),t,arq);
15    str[t] = '\0';
16    printf("%s\n",str);
17    fclose(arq);
18    system("pause");
19    return 0;
20  }
```

Gravando uma matriz usando fprintf()

Exemplo: gravando uma matriz

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    FILE *arq;
05    arq = fopen("matriz.txt","w");
06    if(arq == NULL){
07      printf("Erro\n");
08      system("pause");
09      exit(1);
10    }
11    int mat[2][2] = {{1,2},{3,4}};
12    int i,j;
13    for(i = 0; i < 2; i++)
14      for(j = 0; j < 2; j++)
15        fprintf(arq,"%d\n",mat[i][j]);
16    fclose(arq);
17    system("pause");
18    return 0;
19  }
```

Lendo uma matriz: feof() e fscanf()

Exemplo: lendo uma matriz

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04    FILE *arq;
05    arq = fopen("matriz.txt","r");
06    if(arq == NULL){
07      printf("Erro\n");
08      system("pause");
09      exit(1);
10    }
11    int v,soma=0;
12    while(!feof(arq)){
13      fscanf(arq,"%d",&v);
14      soma += v;
15    }
16    printf("Soma = %d\n",soma);
17    fclose(arq);
18    system("pause");
19    return 0;
20  }
```

Fim do arquivo - feof()

- A constante EOF (“End Of File”) indica o fim de um arquivo.
- Porém, quando manipulando dados binários, um valor inteiro igual ao valor da constante EOF pode ser lido.
- Nesse caso, se utilizarmos a constante EOF para verificar se chegamos ao final do arquivo, vamos receber a confirmação quando, na verdade, isso ainda não aconteceu.
- Para evitar esse tipo de situação, a linguagem C inclui a função feof(), que determina quando o final de um arquivo foi atingido. Seu protótipo é:
  - int feof(FILE *fp)
- Basicamente, a função feof() recebe como parâmetro o ponteiro fp, que determina o arquivo a ser verificado. Como resultado, a função retorna um valor inteiro igual a ZERO se ainda não tiver atingido o final do arquivo.
- Um valor de retorno diferente de zero significa que o final do arquivo já foi atingido.

Erro ao acessar um arquivo - ferror()

- Ao se trabalhar com arquivos, diversos tipos de erros podem ocorrer: um comando de leitura pode falhar, pode não haver espaço suficiente em disco para gravar o arquivo etc.
- Para determinar se uma operação realizada com o arquivo produziu algum erro, existe a função ferror(), cujo protótipo é:
  - int ferror(FILE *fp)
- Basicamente, a função ferror() recebe como parâmetro o ponteiro fp, que determina o arquivo que se quer verificar.
- A função verifica se o indicador de erro associado ao arquivo está marcado e retorna um valor igual a zero se nenhum erro ocorreu. Do contrário, a função retorna um número diferente de zero.
- Como cada operação modifica a condição de erro do arquivo, a função ferror() deve ser chamada logo após cada operação realizada com o arquivo.

Movendo-se dentro do arquivo - fseek()

- De modo geral, o acesso a um arquivo é quase sempre feito de modo sequencial.
- Porém, a linguagem C permite realizar operações de leitura e escrita randômica. Para isso, usa-se a função fseek(), cujo protótipo é:
  - int fseek(FILE *fp, long numbytes, int origem)
- Basicamente, a função fseek() move a posição atual de leitura ou escrita no arquivo para um byte específico, a partir de um ponto especificado.
- A função fseek() recebe três parâmetros de entrada:
  - **fp**: o ponteiro para o arquivo em que se deseja trabalhar.
  - **numbytes**: é o total de bytes a partir de “origem” a ser pulado.
  - **origem**: determina a partir de onde os “numbytes” de

O valor do parâmetro **numbytes** pode ser negativo, dependendo do tipo de movimentação que formos realizar.

Usado a função fseek()

Exemplo: usando a função fseek()

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  struct cadastro{ char nome[20], rua[20]; int idade;};
04  int main(){
05    FILE *f = fopen("arquivo.txt","wb");
06    if(f == NULL){
07       printf("Erro na abertura\n");
08       system("pause");
09       exit(1);
10    }
11    struct cadastro c,cad[4] = {"Ricardo","Rua 1",31,
12                                "Carlos","Rua 2",28,
13                                "Ana","Rua 3",45,
14                                "Bianca","Rua 4",32};
15    fwrite(cad,sizeof(struct cadastro),4,f);
16    fclose(f);
17    f = fopen("arquivo.txt","rb");
18    if(f == NULL){
19       printf("Erro na abertura\n");
20       system("pause");
21       exit(1);
22    }
23    fseek(f,2*sizeof(struct cadastro),SEEK _ SET);
24    fread(&c,sizeof(struct cadastro),1,f);
25    printf("%s\n%s\n%d\n",c.nome,c.rua,c.idade);
26    fclose(f);
27    system("pause");
28    return 0;
29  }
```

Valores para a função fseek()

- A função fseek() retorna um valor inteiro igual a ZERO quando a movimentação dentro do arquivo for bem-sucedida. Um valor de retorno diferente de zero significa que houve um erro durante a movimentação.
- Os valores possíveis para o parâmetro origem são definidos por constante na biblioteca stdio.h e são:

Constante	Valor	Significado
SEEK_SET	0	Início do arquivo
SEEK_CUR	1	Ponto atual no arquivo
SEEK_END	2	Fim do arquivo

- Portanto, para movermos numbytes a partir do início do arquivo, a origem deve ser SEEK_SET. Se quisermos mover a partir da posição atual em que estamos no arquivo, devemos usar a constante SEEK_CUR. Por fim, se quisermos mover a partir do final do arquivo, a constante SEEK_END deverá ser usada.

Voltando ao começo do arquivo - rewind()

- Outra opção de movimentação dentro do arquivo é simplesmente retornar para o seu início. Para tanto, usa-se a função rewind(), cujo protótipo é:

  - void rewind(FILE *fp)

- A função rewind() recebe como parâmetro de entrada apenas o ponteiro para o arquivo que se deseja retornar para o início.

Excluindo um arquivo - remove()

- Além de permitir manipular arquivos, a linguagem C também permite excluí-los do disco rígido. Isso pode ser feito facilmente utilizando a função remove(), cujo protótipo é:
  - int remove(char *nome_do_arquivo)

- Diferentemente das funções vistas até aqui, a função remove() recebe como parâmetro de entrada o caminho e o nome do arquivo a ser excluído do disco rígido, e não um ponteiro para FILE.

- Como resultado, essa função retorna um valor inteiro igual a ZERO quando houver sucesso na exclusão do arquivo. Um valor de retorno diferente de zero significa que houve um erro durante a sua exclusão.

Renomeando um arquivo - rename()

- A função rename(), cujo protótipo é:
  - int rename(const char *,const char *)
- É útil quando se precisa renomear um arquivo.

- Esta função recebe como parâmetros de entrada o nome atual do arquivo a ser renomeado e o novo nome a ser dado a tal arquivo.

- Como resultado, essa função retorna um valor inteiro igual a ZERO quando houver sucesso na alteração do nome do arquivo. Um valor de retorno diferente de zero indica que houve um erro durante a renomeação.

Sabendo a posição atual dentro do arquivo - ftell()

- Outra operação bastante comum é saber onde estamos dentro de um arquivo.
- Para realizar essa tarefa, usamos a função ftell(), cujo protótipo é:
  - long int ftell(FILE *fp)
- Basicamente, a função ftell() recebe como parâmetro o ponteiro fp, que determina o arquivo a ser manipulado. Como resultado, a função ftell() retorna a posição atual dentro do fluxo de dados do arquivo:
  - Para arquivos binários, o valor retornado indica o número de bytes lidos a partir do início do arquivo.
  - Para arquivos texto, não existe garantia de que o valor retornado seja o número exato de bytes lidos a partir do início do arquivo.
  - Se ocorrer um erro, o valor −1 no formato long é retornado.

ftell() - Exemplo: descobrindo o tamanho de um arquivo

Exemplo: descobrindo o tamanho de um arquivo

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  #include <string.h>
04  int main(){
05    FILE *arq;
06    arq = fopen("arquivo.bin","rb");
07    if(arq == NULL){
08      printf("Erro na abertura do arquivo");
09      system("pause");
10      exit(1);
11    }
12    int tamanho;
13    fseek(arq,0,SEEK _ END);
14    tamanho = ftell(arq);
15    fclose(arq);
16    printf("Tamanho do arquivo em bytes: %d:",tamanho);
17    system("pause");
18    return 0;
19  }
```

Exercícios II

1) Escreva um programa que receba via linha de comando dois nomes de arquivos: um atual (existente) e um outro como sendo o novo nome do arquivo. Em seguida, renomeie o arquivo de acordo com os parâmetros passados.

2) Escreva um programa que receba via linha de comando os nomes de dois arquivos texto. Crie um terceiro arquivo texto com o conteúdo dos dois primeiros juntos (o conteúdo do primeiro seguido do conteúdo do segundo). O nome a ser dado ao novo arquivo deve ser formado pela concatenação dos nomes dos dois arquivos dados que devem ser ligados por um “underline” (_), retirando-se a extensão dos mesmos e incluindo a extensão “.txt” ao final.
P.Ex: Caso os nomes dos arquivos sejam: ArqTexto1.doc e ArqTexto2.rtf, o nome do novo arquivo criado deve ser: “ArqTexto1_ArqTexto2.txt”

3) Crie um programa que defina três constantes: N, M e A (via diretivas de compilação), onde N é o número de valores que deverão ser gerados aleatoriamente no intervalo de 0 até M e A é o nome de um arquivo (que deverá ter formato binário) e que armazenará um vetor com os N valores inteiros aleatórios que serão gerados.

FIM