AED

Arquivos

Profa.: Márcia Sampaio Lima

EST - UEA

Arquivos

- As funções de entrada/saída em arquivos estão declaradas no cabeçalho *stdio.h. A* manipulação de arquivos também se dá por meio de **fluxos** (*streams*).
- Etapas para manipulação de um arquivo:
  - abrir o arquivo;
  - ler e/ou gravar os dados desejados;
  - fechar o arquivo.

Arquivos

- Em C, todas as operações realizadas com arquivos envolvem seu *identificador de fluxo*, que é uma variável do tipo FILE * .
- Declaração:

```
FILE *fp;
```

Arquivos

- Para trabalhar em um arquivo, precisamos abri-lo, associando-o a uma variável interna do programa.
- Usamos uma variáveis do tipo FILE *:

```
FILE *entrada;
FILE *saida;
```

Arquivos

FILE *entrada;

FILE *saida;

- Pela declaração desses ponteiros, passa a existir variáveis de nome entrada e saida, que são ponteiros para um arquivo a ser manipulado.
- O ponteiro de arquivo une o sistema de E/S a um buffer e não aponta diretamente para o arquivo em disco, contendo informações sobre o arquivo, incluindo nome, status (aberto, fechado e outros) e posição atual sobre o arquivo

Arquivos

- A primeira coisa que se deve fazer para manipular um arquivo é abri-lo.
  - Para isso, usamos a função fopen().
  - A função que abre um arquivo em C, que devolve o valor NULL (nulo) ou um ponteiro associado ao arquivo.
  - Deve ser passado para função o nome físico do arquivo e o modo como este arquivo deve ser aberto

Arquivos

- A primeira coisa que se deve fazer para manipular um arquivo é abri-lo.
  - Sua sintaxe é:

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

```
FILE * Arquivo;
Arquivo = fopen ("texto.txt","w");
```

Arquivos

- Nome do arquivo:
  - uma string ou com o caminho completo:
    - /usr/share/appname/app.conf
    - C:\Documentos\nomes.txt
    - diretório atual (nomes.txt, ../app.conf)

Arquivos

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

Arquivos

- A associação entre variável e arquivo ainda é feita através da função **fopen()**:

```
FILE *entrada;
FILE *saída;
entrada = fopen("arquivo_entrada.txt", "r");
saida = fopen("arquivo_saida.txt", "w");
```

Arquivos

- A **fopen()** precisa de dois parâmetros:
  - o nome do arquivo.
  - Modo de abertura do arquivo:
    - gravar (“w”, de write)
    - ler dados (“r”, de read).
  - No final, essas variáveis conterão um tipo de referência aos arquivos que abrimos.
  - Serão usadas para efetuar a leitura e gravação dos dados nos arquivos.

```
entrada = fopen("arquivo_entrada.txt", "r");
saida = fopen("arquivo_saida.txt", "w");
```

Arquivos

- **fopen()** :
  - Ao abrir um arquivo para gravação, pode acontecer de já existir um arquivo com o mesmo nome que você pediu. Se isso ocorrer, o arquivo existente será apagado, e o que você gravar ficará no lugar do arquivo antigo.
  - Caso contrário, o programa simplesmente criará um arquivo novo, com o nome solicitado.
  - Se o objetivo for adicionar dados ao final do arquivo, sem apagar nada, deve-se usar, no lugar da **letra w, a letra "a" (de append).**

Arquivos

- Modos de abertura de arquivo:

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.

Arquivos

- Problemas de Abertura de Arquivos:
  - O arquivo não existe e você tentou abri-lo para leitura;
  - Você não tem permissão para ler ou gravar o arquivo pedido;
  - O arquivo já está aberto e/ou bloqueado por outro programa.

  - Nesses casos, quando a função **fopen()** retornará o valor NULL

Arquivos

- Problemas de Abertura de Arquivos:
  - Para verificar se isso ocorreu:

```
FILE *entrada;
entrada = fopen("arquivo_entrada.txt", "r");
if (entrada == NULL)
 { printf("Arq. não pôde ser aberto! \n ");
 }
```

Arquivos

- Fechar arquivo:
  - Para o esvaziamento da memória de um arquivo é utilizada a função fclose(), que associa-se diretamente ao nome lógico do arquivo (STREAM):

```
fclose (Arquivo);
fclose (entrada);
fclose (saida);
```

Gravando e lendo Dados em Arquivos

- Funções para a operação de gravação e leitura de dados em arquivos.
  - putc() ou fputc(): Grava um único caractere no arquivo
  - fprintf() : Grava dados formatados no arquivo, de acordo com o tipo de dados (float, int, ...).
    - Similar ao printf(), porém ao invés de imprimir na tela, grava em arquivo

Gravando e lendo Dados em Arquivos

- Funções para a operação de gravação e leitura de dados em arquivos.
  - fwrite() : Grava um conjunto de dados heterogêneos (struct) no arquivo
  - fscanf(): retorna a quantidade variáveis lidas com sucesso

Arquivos

- Sintaxe das funções para gravação
  - Grava o conteúdo da variável caracter no arquivo
    - putc (caractere, arquivo);
  - A função mais básica de entrada de dados é **putc** (= *put character*).
  - Cada invocação da função grava um caractere no arquivo especificado.
  - Por exemplo:
    - `putc ('*', arquivo)`

Arquivos

```
#include<stdio.h>

int main(){

   FILE * arquivo;
   arquivo = fopen("texto.txt","a");
   if (arquivo == NULL){
     printf("\nErro na abertura do arquivo!!");}

    putc('f',arquivo);

    fclose(arquivo);
   return 0;
}
```

Arquivos

- Sintaxe das funções para gravação
  - Grava dados formatados no arquivo, de acordo com o tipo de dados (float, int, ...)
    - fprintf(arquivo, "formatos", var1, var2 ...);

Arquivos

- Gravando em arquivos:
  - Usa a função **fprintf()**.
    - É precisa informar o arquivo no qual os dados devem ser escritos.
    - Informar a variável que foi associada ao arquivo.
    - Exemplos simples:

```
fprintf(saida, "Bom dia!\n");
fprintf(saida, "Resulta: %d\n", resultado);
fprintf(saida, "soma = %10.6f quociente =
  %10.6f\n", soma, quociente);
```

```c
#include<stdio.h>

int main(){

   FILE * arquivo;

   arquivo = fopen("texto.txt","a");
   if (arquivo == NULL){
       printf("\nErro na abertura do arquivo!!");
   }
   putc('f',arquivo);
   fprintf(arquivo,"\n%d %d %s",12,12,"maria");
   fprintf(arquivo,"\n%d %d %s",222,32,"joao");
   fclose(arquivo);
   return 1;
}
```

Arquivos

- Lendo arquivos:
  - A função correspondente de *leitura* de caracteres é **getc** (= *get character*).
  - Cada chamada da função lê um caractere do arquivo especificado.

```
c = getc (entrada);
```

```
#include<stdio.h>
int main(){

   FILE * arquivo;
   char caractere;

   arquivo = fopen("texto.txt","r");
   if (arquivo == NULL){
       printf("\nErro na abertura do arquivo!!");
   }
   caractere= getc(arquivo);
   while(caractere != EOF){
       printf("%c",caractere);
       caractere= getc(arquivo);
   }
   fclose(arquivo);
   return (0);
```

Arquivos

- Lendo arquivos:
  - Usar a função **fscanf():**
  - Retorna a quantidade variáveis lidas com sucesso

```
fscanf(arquivo, "formatos", &var1, &var2 ...);
```

  - Por exemplo:
    - int n1, n2;
    - ...
    - fscanf(entrada, "%d %d", &n1, &n2);

```
#include<stdio.h>
int main(){

   FILE * arquivo;
   char arq_aux, nome[30];
   int idade1, idade2;
   arquivo = fopen("texto.txt","r");
   if (arquivo == NULL){
       printf("\nErro na abertura do arquivo!!");}
   arq_aux = fscanf(arquivo,"%d %d %s",&idade1,&idade2,&nome);
   while(arq_aux != EOF){
       printf("%d",idade1);
       printf("%d",idade2);
       printf("%s",nome);
   arq_aux = fscanf(arquivo,"%d %d %s",&idade1,&idade2,&nome);
    }
    fclose(arquivo);
   return 1;
```

Arquivos

- Exemplo fscanf();
  - Arquivo banco.cpp

Lendo e Gravando Estruturas

- Além da manipulação de arquivos do tipo texto, pode-se ler e escrever estruturas usando as funções fread() e fwrite().
- Sintaxe:

```
fread (buffer, tamanhoembytes, quantidade,
                                ponteirodearquivo)

fwrite(buffer, tamanhoembytes, quantidade,
                                ponteirodearquivo)
```

Arquivos

- **buffer** é um endereço de memória da estrutura de onde deve ser lido ou onde devem ser escritos os valores.
- **tamanhoembytes** é um valor numérico que define o número de bytes da estrutura que deve ser lida/escrita.
- **quantidade** é o número de estruturas que devem ser lidas ou escritas em cada processo de fread ou fwrite.
- **ponteirodearquivo** é o ponteiro do arquivo de onde deve ser lida ou escrita uma estrutura

Lendo e Gravando Estruturas

- As funções **fread** e **fwrite** são empregadas para leitura e escrita de dados em modo binário.
- Essas funções são usadas na leitura e escrita de estruturas criadas pelos usuários.
- A gravação em binário da estrutura permite que o programador ao escrever ou ler do arquivo se preocupe somente com **a estrutura** como um todo e não com **cada elemento** que a compõe.

Lendo e Gravando Estruturas

```
fread (buffer, tamanhoembytes, quantidade,
                                ponteirodearquivo);
```

- **fread()** lê `quantidade` objetos, cada um com `tamanhoembytes` bytes de comprimento do fluxo apontado por `ponteirodearquivo` e os coloca na localização apontada por `buffer`.

Lendo e Gravando Estruturas

```
fread (buffer, tamanhoembytes, quantidade,
                                ponteirodearquivo);
```

- **fread()** retorna o número de itens que foram lidos com sucesso. Caso ocorra um erro, ou o fim do arquivo foi atingido o valor de retorno é menor do que `quantidade` ou zero.

Lendo e Gravando Estruturas

- **fread()**
  - Tal função é importante para que o programa de manipulação de arquivos possa saber se ainda existem registros para serem lidos.
  - Por exemplo, enquanto o retorno da instrução abaixo for igual a 1, o programa continua lendo registros:

```
retorno = fread(&Vcli, sizeof(struct
                        Tcliente), 1, cliente);
```

Lendo e Gravando Estruturas

- `fwrite(buffer, tamanhoembytes, quantidade, ponteirodearquivo)`

- **Fwrite()** escreve `quantidade` elementos de dados, cada um com `tamanhoembytes` bytes de comprimento, para o fluxo apontado por `buffer` obtendo-os da localização apontada por `ponteirodearquivo`.

Lendo e Gravando Estruturas

- `fwrite(buffer, tamanhoembytes, quantidade, ponteirodearquivo);`

- **fwrite()** retorna o número de itens que foram lidos com sucesso. Caso ocorra um erro, ou o fim do arquivo foi atingido o valor de retorno é menor do que `quantidade` ou zero.

Arquivos

- Normalmente é necessário manipular arquivos por meio de estruturas de dados ou arquivos de estruturas (struct).
- Podemos por exemplo falar num arquivo de CLIENTES, onde cada cliente possui NOME, RG, ENDERECO E TELEFONE

- Escrever a estrutura

```
typedef struct {
  char nome[30];
  char end[40];
  char  rg[10];
  char fone[12];
}tCliente;
```

Essencial ...

- **sizeof ()** retorna a quantidade de bytes de um determinado tipo ou variável .
  - Não é possível pedir para que se posicione no segundo, terceiro ou último registro.
  - Para isso, programador em C deve saber o tamanho em bytes de cada registro, e posicionar-se de acordo com este tamanho.

Exemplo

- Clientes.cpp

```
#include<stdio.h>

#define MAX 2

typedef struct {
   char nome[30];
   char end[40];
   char  rg[10];
   char fone[12];
}tCliente;
```

Exemplo

```
int main(){


   FILE * arqCliente;
   tCliente vetCli[MAX], vetLeCli[MAX];
   int retorno;
   //Abrindo o arquivo binário
   arqCliente = fopen("dadosCli.xxx","w+b");
   if (arqCliente == NULL){
       printf("\nErro na abertura do arquivo!!");
   }
```

Exemplo

```
//lendo uma coleção de clientes do teclado
for(int i = 0; i < MAX; i++){
    puts("Nome ? ");
    gets(vetCli[i].nome);
    puts("Endereco  ? ");
    gets(vetCli[i].end);
    puts("Telefone ? ");
    gets(vetCli[i].fone);
    puts("RG  ? ");
    gets(vetCli[i].rg);

}
```

Exemplo

```
puts("Gravando .....");

for(int i = 0; i < MAX; i++){

    retorno =
fwrite(&vetCli[i],sizeof(tCliente),1,arqCliente);

    if (retorno != 1){
          puts("Erro na escrita do arquivo!!!");
    }
}
```

Exemplo

```
rewind(arqCliente);

puts("Lendo .....");
for(int i = 0; i < MAX; i++){

    retorno =
fread(&vetLeCli[i],sizeof(tCliente),1,arqCliente);
    if (retorno != 1){
          if (feof(arqCliente)){
                break;
          }
          puts("Erro na escrita do leitura!!!");
    }
}
```

Exemplo

```
puts("Mostrando dados .....");
for(int i = 0; i < MAX; i++){
    printf("\nNome: %s",vetLeCli[i].nome);
    printf("\nEndereco: %s",vetLeCli[i].end);
    printf("\nFone: %s",vetLeCli[i].fone);
    printf("\nRG: %s",vetLeCli[i].rg);

}


fclose(arqCliente);
```

Posicionando em um registro

- A linguagem C não é possível saber qual é a posição de cada registro no arquivo.
- Em outras linguagens, a movimentação em registros é feita por meio de funções que fazem a leitura da linha do registro.
  - Em C esta posição pode ser calculada pelo tamanho do registro

Posicionando em um registro

- Não é possível pedir para que se posicione no segundo, terceiro ou último registro.
- Para isso, programador em C deve saber o tamanho em bytes de cada registro, e posicionar-se de acordo com este tamanho.
- A função seek() movimento de byte em byte.

```
seek(<referencia arquivo>, <n> , <modo> );
```

Posicionando em um registro

- O parâmetro **<n>** indica quantos bytes devem ser avançados ou retrocedidos.
- O exemplo a seguir posiciona-se no 4 registro do arquivo de cliente:

```
fseek(ArqCli,4 *sizeof(tCliente), SEEK_SET);
```

- sizeof() indica quantos bytes possui o registro a ser inserido (ou a estrutura definida para o registro)

Posicionando em um registro

- Neste caso o tipo Cliente, que é o registro, foi utilizado para indicar o tamanho de cada registro.
- Multiplicando-se o valor retornado por quatro obtém-se o local do quarto registro do arquivo.
- Caso o local (o registro) solicitado não exista não será feito o posicionamento e o registro atual continuará sendo o mesmo

Posicionando em um registro

- Outros parâmetros usados pela função seek():
  - SEEK_SET - Parte do início do arquivo e avança **<n>** bytes.
  - SEEK_END - Parte do final do arquivo e retrocede **<n>** bytes.
  - SEEK_CUR - Parte do local atual e avança **<n>** bytes.