LINGUAGEM C: ARQUIVOS

Prof. André Backes

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).

Tipos de Arquivos

3

- Basicamente, a linguagem C trabalha com dois tipos de arquivos: de 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).

Tipos de Arquivos

4

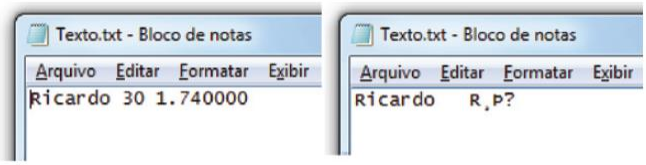
- Arquivo binário
  - armazena uma seqüência de bits que está sujeita as convenções dos programas que o gerou. Ex: arquivos executáveis, arquivos compactados, arquivos de registros, etc.
  - os dados são gravados na forma binária (do mesmo modo que estão na memória). Ex.: um número inteiro de 32 bits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

Tipos de Arquivos

5

- Ex: Os dois trechos de arquivo abaixo possuem os mesmo dados :

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



Arquivo Texto **Arquivo Binário**

Manipulando arquivos em C

6

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

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

Manipulando arquivos em C

7

- 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 em C

8

- 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 um arquivo

9

- 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 um arquivo

10

- 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 um arquivo

11

- O modo de abertura determina que tipo de uso será feito do arquivo.
- A tabela a seguir mostra os modo válidos de abertura de um arquivo.

Abrindo um arquivo

12

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 um arquivo

13

- 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

```
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 um arquivo

14

- 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 um arquivo

15

- 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 um arquivo

16

- Exemplo

```
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

17

- 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 um arquivo

18

- 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 um arquivo

19

- 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 um arquivo

20

- 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 em Arquivos

21

- 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 Strings

22

- Existem funções na linguagem C que permitem ler/escrever uma seqüência de caracteres, isto é, uma string.
  - **fputs()**
  - **fgets()**

Escrita/Leitura de Strings

23

- 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

24

- 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.

- A função **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

25

- Exemplo da função **fputs:**

```
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

26

- 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

27

- A função **fgets** recebe 3 parâmetros
  - **str**: aonde a lida será armazenada, **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

28

- 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

29

- 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

30

- Exemplo da função **fgets**

```
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 por fluxo padrão

31

- 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

32

- 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

33

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

Escrita/Leitura por fluxo padrão

34

- Exemplo da funções **fprintf**

```
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

35

- Exemplo da funções **fscanf**

```
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;
}
```

Fim do arquivo

36

- A constante **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

37

- 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

38

- 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

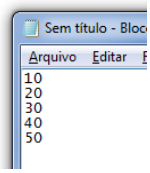
39

- 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

40

- 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

41

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
```

Apagando um arquivo

42

- 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

- Exemplo da função **remove**

```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;
}
```

MATERIAL COMPLEMENTAR

- Vídeo Aulas
  - Aula 66: Arquivos pt.1 – Introdução:
  - youtu.be/LNu-0bzxpos
  - Aula 67: Arquivos pt.2 – Arquivos Texto e Binário:
  - youtu.be/ueg-IE8cZH4
  - Aula 68: Arquivos pt.3 – Abrir e Fechar:
  - youtu.be/uYymG_oUPeY
  - Aula 69: Arquivos pt.4 – fputc:
  - youtu.be/X6BcBhRCR8M
  - Aula 70: Arquivos pt.5 – fgetc:
  - youtu.be/FwW2T3jGvdg
  - Aula 71: Arquivos pt.6 - Trabalhando com Arquivos:
  - youtu.be/WdZv1gCpDjg
  - Aula 72: Arquivos pt.7 – EOF **(contém erros!):**
  - youtu.be/xN61MLUgkSg
  - Aula 73: Arquivos pt.8 - fputs:
  - youtu.be/ODjgyg6WbPk

Material Complementar

- Vídeo Aulas
  - Aula 74: Arquivos pt.9 – fgets:
  - youtu.be/GDVPYnD-T_w
  - Aula 75: Arquivos pt.10 – fwrite:
  - youtu.be/rBnZTxbWqZQ
  - Aula 76: Arquivos pt.11 – fread:
  - youtu.be/ZxuacsaCdaI
  - Aula 77: Arquivos pt.12 – fprintf:
  - youtu.be/4WlsKHHVda0
  - Aula 78: Arquivos pt.13 – fscanf:
  - youtu.be/jnotzdaKjOI
  - Aula 79: Arquivos pt.14 - fseek e rewind:
  - youtu.be/cdXGEy-6jMU
  - Aula 90: Mau uso da função FEOF():
  - youtu.be/2pNlbC_VN94