Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

FACOM32201 - Algoritmos e Programação II

Prof. Thiago Pirola Ribeiro

ARQUIVOS

Parte 2

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 se lia e escrevia na tela.

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 se lia e escrevia na tela.
- As funções `fprintf` e `fscanf` funcionam de maneiras semelhantes a `printf` e `scanf`, respectivamente

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 se lia e escrevia 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.

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 `fread` e `fwrite`.

Escrita por Fluxo Padrão

```c
#include <stdio.h>
#include <stdlib.h>
void main(){
    FILE *arq;
    char nome[20] = "Thiago";
    int i = 30;
    float a = 1.83;
    int result;
    arq = fopen("arquivo.txt", "w");
    if (arq == NULL) {
        printf("Erro na abertura do arquivo\n");
        exit(1);
    }
    result = fprintf(arq,"Nome: %s\nIdade: %d\nAltura:%f\n",nome,i,a);
    if(result < 0)
        printf("Erro na escrita\n");
    fclose(arq);
}
```

Leitura por Fluxo Padrão

```c
void main(){
    FILE *arq;
    char nome[20], texto[20];
    int i;
    float a;
    arq = fopen("arquivo.txt", "r");
    if (arq == NULL) {
        printf("Erro na abertura do arquivo\n");
        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);
}
```

Escrita/Leitura de Bloco de Dados

- Além da leitura/escrita de caracteres e sequências de caracteres, pode-se ler/escrever blocos de dados.

Escrita/Leitura de Bloco de Dados

- Além da leitura/escrita de caracteres e sequências de caracteres, pode-se ler/escrever blocos de dados.

- Para tanto, se tem duas funções

  `fwrite()` e `fread()`

Escrita de Bloco 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);
```

– `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.

Escrita de Bloco de Dados

- Note que se tem dois valores numéricos: `numero_de_bytes` e `count`. Isto significa que o número total de bytes escritos é:

```
numero_de_bytes * count
```

Escrita de Bloco de Dados

- Note que se tem dois valores numéricos: `numero_de_bytes` e `count`. Isto significa que o número total de bytes escritos é:

  ```
  numero_de_bytes * count
  ```

- Como retorno, se tem o número de unidades efetivamente escritas.

Escrita de Bloco de Dados

- Note que se tem dois valores numéricos: `numero_de_bytes` e `count`. Isto significa que o número total de bytes escritos é:

  `numero_de_bytes * count`

- Como retorno, se tem o número de unidades efetivamente escritas.
- Este número pode ser menor que `count` quando ocorrer algum erro.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void main(){
    FILE *arq;
    arq = fopen("arquivo.txt", "wb");
    if (arq == NULL) {
        printf("Erro na criacao do arquivo\n");
        exit(1);    }
    char str[20] = "Hello World!";
    float x = 5;
    int v[5] = {1,2,3,4,5};
// grava a string toda do 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 do arquivo
    fwrite(&x,sizeof(float),1,arq);
// grava todo o array do arquivo (5 posicoes)
    fwrite(v,sizeof(int),5,arq);
// grava apenas 2 primeiras posicoes do array
    fwrite(v,sizeof(int),2,arq);
    fclose(arq);
}
```

Leitura de Bloco de Dados

- A função `fread` é responsável pela leitura de um bloco de dados de um arquivo

Leitura de Bloco 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);
```

Leitura de Bloco 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);
```

- A função **fread** funciona como a sua companheira **fwrite**, porém lendo do arquivo.

Leitura de Bloco 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);
```

- A função **fread** funciona como a sua companheira **fwrite**, porém lendo do arquivo.
- Como na função **fwrite**, **fread** retorna o número de itens lidos. Este valor será igual a `count` a menos que ocorra algum erro.

Leitura de Bloco de Dados

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void main(){
4      FILE *arq;
5      arq = fopen("arquivo.txt", "rb");
6      if (arq == NULL) {
7          printf("Erro na abertura do arquivo\n");
8          exit(1);
9      }
10     char str1[20],str2[20];
11     float x;
12     int i, v1[5], v2[5];
13
14     // le a string toda do arquivo
15     fread(str1,sizeof(char),12,arq);
16     str1[12] = '\0';
17     printf("%s\n",str1);
```

Leitura de Bloco de Dados

```
1  // le apenas os 5 primeiros caracteres da string
2      fread(str2,sizeof(char),5,arq);
3      str1[5] = '\0';
4      printf("%s\n",str2);
5  // le o valor de x do arquivo
6      fread(&x,sizeof(float),1,arq);
7      printf("%f\n",x);
8  // le todo o array do arquivo (5 posicoes)
9      fread(v1,sizeof(int),5,arq);
10     for(i = 0; i < 5; i++)
11         printf("v1[%d] = %d\n",i,v1[i]);
12 // le apenas 2 primeiras posicoes do array
13     fread(v2,sizeof(int),2,arq);
14     for(i = 0; i < 2; i++)
15         printf("v2[%d] = %d\n",i,v2[i]);
16
17     fclose(arq);
18 }
```

Escrita/Leitura de Bloco de Dados

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

Modos de abertura - Arq. Binário

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”).

```c
struct pessoa {
  char nome[30];
   int idade;
  float peso;
};
void main() {
  struct pessoa pes;
  FILE *fp;
  int erro;
  fp= fopen("arquivo.bin","wb");
  if (fp==NULL) exit(1);
  printf("\nNome: "); gets(pes.nome);
  printf("\nIdade:" ); scanf("%d",&pes.idade);
  printf("\nPeso:" ); scanf("%f",&pes.peso);

  fwrite(&pes, sizeof(struct pessoa), 1, fp);

  erro=fclose(fp);
}
```

```c
struct pessoa {
  char nome[30];
   int idade;
  float peso;
};
void main() {
  struct pessoa pes;
  FILE *fp;
  int erro;
  fp= fopen("arquivo.bin","rb");
  if (fp==NULL) exit(1);
  while(fread(&pes,sizeof(struct pessoa),1,fp)==1){
     printf("\nNome: %s", pes.nome);
     printf("\nIdade: %d", pes.idade);
     printf("\nPeso: %0.2f\n", pes.peso);
  }
  erro=fclose(fp);
}
```

Movendo-se pelo arquivo

- De modo geral, o acesso a um arquivo é sequencial. Porém, é possível fazer buscas e acessos aleatórios 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.
- 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.

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 no arquivo
SEEK_END	2	Fim do arquivo

- Portanto, para mover `numbytes` a partir do início do arquivo, origem deve ser `SEEK_SET`. Para mover da posição atual, `SEEK_CUR`, e a partir do final do arquivo, `SEEK_END`.
- A função devolve 0 quando bem sucedida.

```c
#include <stdio.h>
#include <stdlib.h>
struct cadastro{ char nome[20], rua[20]; int idade;};
int main() {
    FILE *f;
    f = fopen("arquivo.txt", "wb");
    if (f == NULL) {
        printf("Erro na abertura do arquivo\n");
        exit(1);
    }
    struct cadastro c;
    struct cadastro cad[4]= {"Thiago","Rua 1",46,
                             "Carlos","Rua 2",31,
                             "Ana","Rua 3",27,
                             "Bruna","Rua 4",32};
    fwrite(cad,sizeof(struct cadastro),4,f);
    fclose(f);
```

```
1
2       f = fopen("arquivo.txt", "rb");
3       if (f == NULL) {
4           printf("Erro na abertura do arquivo\n");
5           exit(1);
6       }
7
8       fseek(f,2*sizeof(struct cadastro),SEEK_SET);
9
10      fread(&c,sizeof(struct cadastro),1,f);
11
12      printf("%s\n%s\n%d\n",c.nome,c.rua,c.idade);
13
14      fclose(f);
15      return 0;
16  }
```

```c
#include <stdio.h>
#include <stdlib.h>
struct cadastro{ char nome[20], rua[20]; int idade;};
int main() {
    FILE *f;
    f = fopen("arquivo.txt", "wb");
    if (f == NULL) {
        printf("Erro na abertura do arquivo\n");
        exit(1);
    }
    struct cadastro c;
    struct cadastro cad[4]= {"Thiago","Rua 1",46,
                             "Carlos","Rua 2",31,
                             "Ana","Rua 3",27,
                             "Bruna","Rua 4",32};
    fwrite(cad,sizeof(struct cadastro),4,f);
    fclose(f);
    f = fopen("arquivo.txt", "rb");
    if (f == NULL) {
        printf("Erro na abertura do arquivo\n");
        exit(1);
    }
    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);
```

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.

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 se tem um valor inteiro, o qual será igual a 0 se o arquivo for excluído com sucesso.

Apagando um arquivo

```c
#include <stdio.h>
#include <stdlib.h>
int main() {
    int status;

    status = remove("Arquivo.txt");

    if(status != 0){
        printf("Erro na remocao.\n");
        exit(1);
    }
    else
        printf("Arquivo removido com sucesso!\n");

    return 0;
}
```

Faculdade de Computação - FACOM

Bacharelado em Sistemas de Informação

Prof. Thiago Pirola Ribeiro