Unoeste

Algoritmos e Estruturas de Dados

Arquivos

Profª. Me. Carla Plantier Message
Profª. Me. Dalila Espinhosa

2024

Introdução

- Arquivos são coleções de bytes identificados por um nome único.
- Operações em discos são realizadas utilizando-se arquivos.

Arquivos

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

Arquivos do tipo texto e binário

- Uma maneira de classificar operações de acesso a arquivos é de acordo com a forma como estes são abertos: em modo texto ou em modo binário.

- Um arquivo aberto em modo texto é interpretado em C como sequências de caracteres agrupadas em linhas.

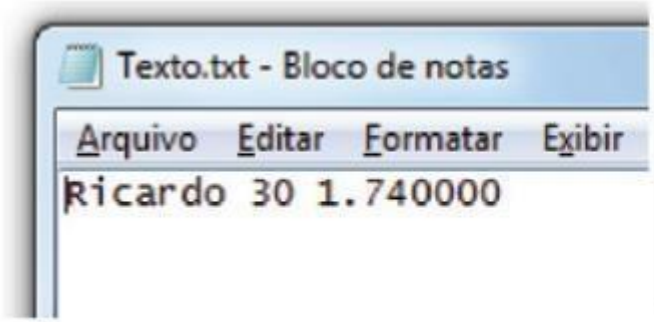
- As linhas são separadas por dois caracteres (13 decimal e 10 decimal), que são convertidos para um caractere único quando o arquivo é gravado (caractere de nova linha).

- Em um arquivo aberto em modo binário qualquer caractere é lido ou gravado sem alterações.

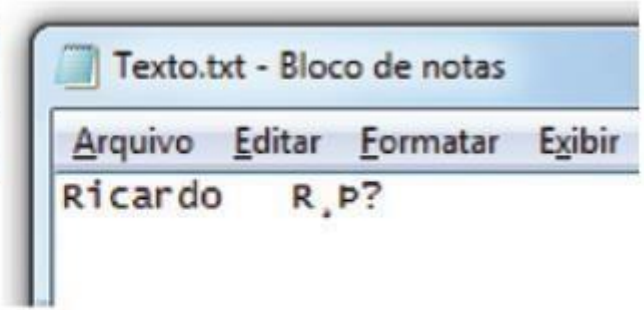
- Na forma de texto os números são armazenados como cadeias de caracteres e na forma binária estes são armazenados como estão na memória.

Os trechos abaixo possuem os mesmos dados

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



Arquivo Texto



Arquivo Binário

Manipulando arquivo em C

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

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

- 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.

Utilização de ponteiros na declaração de arquivos

- Um ponteiro de arquivo é um apontador para suas informações (como nome, status e posição no disco).

- Para ler ou escrever em arquivos, um programa em C precisa utilizar ponteiros de arquivo, que devem ser declarados como variáveis ponteiros do tipo FILE.

- A palavra FILE refere-se a uma estrutura de arquivos definida na biblioteca stdio.h.
- Exemplo de declaração:
- FILE *fp;
- **fp** é o ponteiro para arquivos que nos permitirá manipular arquivos no C.

Função fopen()

- A função fopen( ) associa um arquivo a um buffer de memória. Em sua chamada devem ser utilizados dois parâmetros.
- O primeiro parâmetro refere-se ao nome do arquivo a ser aberto (que pode incluir uma especificação do caminho de pesquisa).
- O segundo parâmetro refere-se ao modo como o arquivo será aberto (para leitura ou escrita, em formato binário ou texto, etc).

Há três opções de abertura:

- “w” – para gravação
- “r” – para leitura
- “a” – para adição de dados
- Estas opções podem ser combinadas com dois modificadores:
- “b” – para abertura no modo binário
- “+” – para indicar que o arquivo já existe e deve ser atualizado.

Exemplo

- FILE *fp;
- fp = fopen(“teste.txt”,”wb”);
- A função fopen( ) retorna um ponteiro para a estrutura FILE, onde estão armazenadas informações sobre o arquivo.
- Caso ocorra um erro na abertura, este ponteiro será nulo.

- 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;
}
```

Fechando um arquivo

- Após a gravação de um arquivo, é necessário fechá-lo.
- Fechar um arquivo faz com que qualquer caractere que tenha permanecido no buffer de memória seja gravado no disco.
- Além disso as áreas de comunicação (estrutura FILE e buffer) são liberadas.

Função fclose()

- A função fclose( ) simplesmente fecha o arquivo associado ao ponteiro FILE utilizado como argumento.
- Exemplo:
- fclose (fp);

Fim de arquivo

- A marca EOF (End of File) é um sinal enviado pelo sistema operacional ao programa em C para indicar o final de um arquivo.

- A marca de fim de arquivo pode ser diferente para diferentes sistemas operacionais.

- Assim, o valor de EOF pode ser qualquer. A biblioteca stdio.h define EOF com o valor correto para o sistema operacional sendo utilizado. Assim, no programa em C, a constante EOF deve ser utilizada para verificação de fim de arquivo.

Leitura e escrita de dados em arquivos

- Em linguagem C é possível ler e gravar arquivos de diferentes maneiras:

- Leitura e escrita de um caractere por vez - funções **getc( )** e **putc( )**;
- Leitura e escrita de strings – funções **fgets( )** e **fputs( )**;
- Leitura e escrita de dados formatados – funções **fscanf( )** e **fprintf( )**;
- Leitura e escrita de registros ou blocos – funções **fread( )** e **fwrite( )**;

Lendo um caractere de um arquivo

- A função getc( ) lê caracteres de um arquivo aberto no modo leitura. Utiliza como parâmetro o ponteiro para o arquivo declarado.

- A função getc( ) devolve EOF quando o fim do arquivo é alcançado.

Exemplo

```c
#include<stdio.h>
main()
{
FILE *fp;
int ch;
fp = fopen("arquivo.txt","r");
do
{
    ch = getc(fp);
    if (ch!=EOF)
    {
        printf("%c",ch); // apresenta na tela
    }
} while (ch != EOF);
  fclose(fp);
}
```

Escrevendo em um arquivo

- A função putc( ) escreve caracteres em um arquivo que foi previamente aberto no modo de escrita.
- Utiliza como parâmetros o caractere a ser escrito e o ponteiro para o arquivo declarado. Se a operação putc( ) for bem sucedida, devolve o caractere escrito. Caso contrário, devolve EOF.

Exemplo

```c
#include<stdio.h>
#include<stdlib.h>
main()
{
    FILE *fp;
    char ch;
    fp = fopen("arquivo.txt","w");
    do
    {
        printf ("Digite um caractere <espaco finalizar> ");
        fflush(stdin);
        scanf("%c",&ch);
        if (ch !=' ') // espaco finaliza a leitura
        {
            putc(ch,fp); // colocar o caractere no arquivo
        }
    }while (ch != ' ');
    fclose(fp);
}
```

Continuando...

Leitura/Escrita de string

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

- A função fgets( ) lê uma string especificada até que um caractere de nova linha seja lido ou que n-1 caracteres tenham sido lidos. Se um caractere de nova linha é lido, este fará parte da string.
- A função retorna um ponteiro para a string (ou um ponteiro nulo se ocorrer um erro).

- A função fgets( ) utiliza 3 argumentos.
- O primeiro é um ponteiro para o buffer onde será armazenada a linha lida. O segundo é um número inteiro que indica o limite máximo de caracteres a serem lidos. Este número deve ser pelo menos 1 maior que o número de caracteres lidos, pois um caractere NULL (‘\0’) é acrescentado na próxima posição livre. O terceiro argumento é um ponteiro para a estrutura FILE do arquivo que está sendo lido.
- A função termina a leitura após ler um caractere de nova linha (‘\n’) ou um caractere de fim de arquivo (EOF).

- Escrevendo uma string em um arquivo.
- A função fputs( ) efetua a gravação de uma string de caracteres em um arquivo.
- Utiliza como parâmetros a string a ser gravada e o ponteiro para o arquivo onde a string será armazenada.

Exemplo

```c
#include <stdio.h>
#include <stdlib.h>
main()
{
    FILE *arq;
    char str[81] = ""; // inicializando a string com vazio
    arq = fopen("texto.txt","a"); // se quiser adicionar troca "w" por "a"
    printf("Digite uma string: ");
    fgets(str,80,stdin);
    fputs(str,arq); // coloca (grava) a string digitada no arquivo
    fputs("\n",arq); // coloca (grava) um pula linha no arquivo
    fclose(arq);
}
```

Funções de entrada e saída

. fscanf()

. fprintf()

Lendo e escrevendo dados formatados

- As funções fprintf( ) e fscanf( ) são utilizadas para ler e armazenar dados formatados de diferentes maneiras (inteiros, reais, caracteres).
- 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.

- 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
```

Exemplo fprintf()

```
main()
{
    FILE *arq;
    char nome[20]="Ricardo";
    int i = 30;
    float a = 1.74;
    arq=fopen("arqgrav.txt","w");
    if(arq==NULL)
      printf("Erro no arquivo.");
    fprintf(arq,"Nome: %s\n",nome);
    fprintf(arq,"Idade: %d\n",i);
    fprintf(arq,"Altura: %f\n",a);
    fclose(arq);
}
```

Exemplo fscanf()

```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;
}
```

Agora vamos começar a trabalhar com os arquivos binários...

Leitura de registros

Além da leitura/escrita de caracteres e sequências de caracteres, podemos ler/escrever blocos de dados (registros). Para tanto, temos duas funções :

– fwrite()
– fread()

Exemplo de gravação

```c
#include <stdio.h>
#include <stdlib.h>
struct funcionarios
{
    char nome[30];
    int id;
    float salario;
};
main()
{
    struct funcionarios cadastro[10];
    FILE *p;
    int cont,x;
    p=fopen("arq2.rec","wb");
    for (cont=0; cont<3; cont++)
    {
        printf("Digite o nome do funcionario: ");
        scanf("%s",cadastro[cont].nome);
        printf("Digite o número de identificação: ");
        scanf("%d",&cadastro[cont].id);
        printf("Digite o salario: ");
        scanf("%f",&cadastro[cont].salario);
    }
    fwrite(&cadastro,sizeof(cadastro),3,p);
    fclose(p);
}
```

A função fwrite( ) utiliza 4 parâmetros:
O primeiro é um ponteiro para a localização de memória do dado a ser gravado.
O segundo é um número inteiro que indica o tamanho do tipo de dado a ser gravado. Pode-se utilizar a função sizeof( ), tamanho de um que retorna o dado.
O terceiro parâmetro é um número inteiro que informa quantos itens do mesmo tipo serão gravados.
O quarto parâmetro é um ponteiro para a estrutura FILE do arquivo que se quer gravar.

Exemplo de leitura

```c
#include <stdio.h>
#include <stdlib.h>
struct funcionarios {
char nome[30];
int id;
float salario;
};

void main()
{
    struct funcionarios cadastro[10];
    FILE *p;
    int cont;
    p=fopen("arq.rec","rb");
    fread(&cadastro,sizeof(cadastro),3,p);
    for (cont=0; cont<3; cont++)
    {
        printf("Nome: %s\t",cadastro[cont].nome);
        printf("Identificacao: %d\t",cadastro[cont].id);
        printf("Salario: %.2f",cadastro[cont].salario);
        printf("\n");
    }
    fclose(p);
}
```

Unoeste