UnidadeVI – Gravar e Mostrar dados Arquivo em Linguagem C

Sumário

3 Operação de Gravação e Busca em arquivos

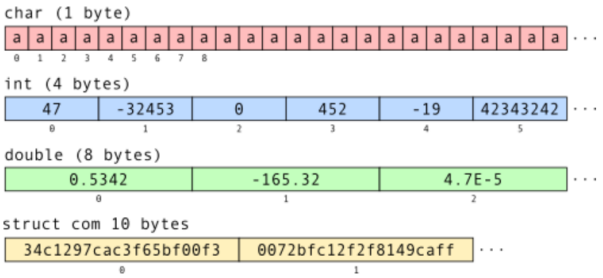
Operações de gravação e busca de dados em arquivos de texto são atividades fundamentais na programação em C, especialmente quando se trata de armazenar e recuperar informações de forma persistente. Para tratar destas operações com arquivos de texto em C, tem-se à disposição um conjunto de funções que permitem a manipulação eficaz desses arquivos. Essas operações incluem a escrita de dados em arquivos de texto e a busca ou leitura desses dados para realizar operações subsequentes.

A gravação de dados em arquivos binários em C envolve a escrita de informações em sua forma bruta, sem formatação adicional como é o caso dos arquivos de texto. Essa abordagem é comumente usada quando se deseja preservar a estrutura interna de dados de forma exata, como em arquivos de imagem, áudio, ou em qualquer situação em que a preservação dos dados em sua forma original seja crucial.

Ao contrário dos arquivos de texto, onde a formatação e a legibilidade humana são prioritárias, os arquivos binários armazenam os dados exatamente como são representados na memória do computador. Isso significa que não há delimitadores de linhas ou caracteres de terminação específicos, e a escrita e leitura são realizadas diretamente em blocos de bytes.

Todos os arquivos contêm sequências de bytes, mas um arquivo é comumente classificado como "binário" quando seu conteúdo não é textual, ou seja, não é representado por caracteres usando codificações como ASCII, UTF-8 ou similares. Arquivos binários são empregados para armazenar dados mais complexos, como imagens, música, código executável, entre outros.

Em C, um arquivo binário é visto como uma sequência de blocos de mesmo tamanho. O tamanho dos blocos depende do tipo de informação armazenada no arquivo. Por exemplo, um arquivo de números reais **double** terá blocos de 8 bytes, enquanto um arquivo de caracteres (**char**) terá blocos de 1 byte.



https://wiki.inf.ufpr.br/maziero/doku.php?id=c:arquivos_binarios

É importante salientar que o sistema operacional apenas armazena a sequência de bytes, sem levar em conta ou registrar o tamanho dos blocos. É responsabilidade da aplicação determinar o tamanho do bloco que deseja utilizar em cada arquivo.

3.1 Arquivo texto x Arquivo binário

Existem dois tipos de streams (sequência de bytes de dados – caracteres) em C:

- o Texto – composto apenas de caracteres organizados em linhas de no máximo 255 caracteres. Cada linha é terminada com um caractere de fim de linha (ex.: CR-LF no DOS);
- o Binário – qualquer tipo de dado, incluindo texto. Bytes de dados em um stream binário não são traduzidos ou interpretados de forma diferente.

Por exemplo, no modo texto, a combinação dos caracteres correspondentes ao carriage return (CR) e line feed (LF) em um arquivo são interpretados como um "\n" ao imprimir na tela. Em modo binário, não recebem tratamento específico, sendo impresso os caracteres "especiais" como parte da mesma linha.

Tipo de Arquivo	**Vantagens**	**Desvantagens**
Texto	- *Facilidade de leitura:* os dados podem ser lidos por qualquer programa, caractere por caractere.	- *Maior gasto de memória:* gasta 1 byte por caractere, exigindo, por exemplo, 10 bytes para armazenar o número 123456.789 que gastaria 4 bytes em um `float`); - *Maior gasto de tempo em buscas:* para acessar o *n*-ésimo elemento, exige uma busca seqüencial acessando todos os elementos anteriores no arquivo.
Binário	- *Menor gasto de memória*: similar à forma como é armazenado em memória RAM, onde o número 123456.789 gastaria 4 bytes equivalente a um `float`; - *Menor gasto de tempo em buscas*: para saber a posição do *n*-ésimo número fracionário de uma lista de números fracionários, bastaria localizar a posição `n*sizeof(float)` movendo o cursor do arquivo.	*Dificuldade de leitura:* apenas o criador do arquivo sabe como manipulá-lo.

Fonte: https://www.ic.unicamp.br – acessado em Março/2024

Cada arquivo deverá possuir um nome com primeiro nome e a extensão:

Nomes de arquivos:

- Cada arquivo armazenado em memória secundária (disco rígido, disquete, CD etc.) possui um nome.
- As regras para a composição desses nomes podem variar entre os diferentes sistemas operacionais.

- No DOS e Windows 3.x, os nomes têm até 8 caracteres seguidos opcionalmente por um ponto e uma extensão de até 3 caracteres.
- No Windows 9x, 2000, ME, XP, NT e na maioria dos sistemas UNIX, os nomes de arquivos podem ter até 256 caracteres.
- Certos caracteres não são permitidos na composição de nomes de arquivo no Windows 9x. Cada sistema operacional tem suas próprias regras de composição de caracteres para nomes de arquivos, que podem incluir os caracteres: **/ \ : * ? " < >**.
- Em C, um nome de arquivo pode incluir informações sobre o seu caminho (drive e/ou diretório onde o arquivo está localizado no disco). Por exemplo, o arquivo "contato.txt" no diretório "c:\dados\cadastros" pode ser referenciado em C da seguinte maneira no Windows:

  ```
  char *arquivo = "c:\\dados\\cadastros\\contato.txt";
  ```

3.2 Funções de Leitura e Escrita de blocos

A linguagem C oferece funções para ler e escrever blocos de bytes em arquivos, que efetuam a cópia desses bytes da memória para o arquivo ou vice-versa.

A funções a seguir permitem ler/escrever blocos de bytes em arquivos binários. Todas essas funções estão definidas no arquivo **stdio.h** ou **cstdio**.

As funções **fwrite()** e **fread()** são as principais funções usadas para gravação e leitura de dados em arquivos binários, respectivamente. Aqui está um breve resumo de cada uma delas:

- **fwrite():**
  - A função **fwrite()** é usada para gravar blocos de dados em um arquivo binário.
  - Sua assinatura é: **size_t fwrite(const void *ptr, size_t size, size_t count, FILE *stream);**
    - **ptr**: Ponteiro para o bloco de memória que contém os dados a serem gravados.
    - **size**: Tamanho em bytes de cada elemento a ser gravado.
    - **count**: Número de elementos a serem gravados.
    - **stream**: Ponteiro para o arquivo onde os dados serão gravados.
  - Retorna o número total de elementos gravados com sucesso.
- **fread():**

- A função **fread()** é usada para ler blocos de dados de um arquivo binário.
- Sua assinatura é: **size_t fread(void *ptr, size_t size, size_t count, FILE *stream);**
    - **ptr**: Ponteiro para o local na memória onde os dados lidos serão armazenados.
    - **size**: Tamanho em bytes de cada elemento a ser lido.
    - **count**: Número de elementos a ser lido.
    - **stream**: Ponteiro para o arquivo de onde os dados serão lidos.
- Retorna o número total de elementos lidos com sucesso.

Ao trabalhar com arquivos binários, é importante lembrar que os dados são escritos e lidos exatamente como estão na memória. Portanto, é essencial garantir que a estrutura dos dados seja mantida corretamente para que a leitura e a gravação ocorram conforme o esperado. Além disso, ao usar arquivos binários, não há formatação para fins de legibilidade humana, o que pode dificultar a depuração ou a inspeção manual do conteúdo do arquivo.

Além dessas operações básicas, o deslocamento dentro do arquivo pode ser necessário para acessar dados específicos. Para isso, as funções **ftell()** e **fseek()** são utilizadas. A função **ftell()** retorna a posição atual do cursor no arquivo, enquanto **fseek()** permite mover o cursor para uma posição específica dentro do arquivo. Isso é particularmente útil ao implementar operações de busca mais complexas.

- **ftell():**
    - A função **ftell()** é usada para determinar a posição atual do cursor de arquivo em relação ao início do arquivo.
    - Ela retorna um valor do tipo **long**, que representa o deslocamento em bytes do início do arquivo até a posição atual do cursor.
    - É comumente usada para saber onde estamos dentro do arquivo, especialmente ao trabalhar com operações de leitura ou gravação de arquivos.
- **fseek():**
    - A função **fseek()** é usada para mover o cursor de arquivo para uma posição específica dentro do arquivo.
    - Ela aceita três argumentos: um ponteiro para o arquivo, um deslocamento em bytes (que pode ser positivo ou negativo) e uma origem de onde começar a contagem do deslocamento.
    - A origem pode ser:

1. **SEEK_SET** (início do arquivo),
2. **SEEK_CUR** (posição atual) ou
3. **SEEK_END** (final do arquivo).

- Sintaxe da função:
  fseek(FILE *arquivo, int deslocamento, int pontoInicio);
- É comumente usada para navegar dentro de arquivos e realizar operações de leitura ou gravação em posições específicas.

Abaixo, estão relacionadas outras funções/operadores que complementam a programação das funcionalidades de leitura e escrita em arquivos binário.

- **sizeof():**
  - A função **sizeof()** é um operador em C que retorna o tamanho em bytes de um tipo de dado ou de uma variável.
  - Não é realmente uma função, mas um operador do compilador que é avaliado em tempo de compilação.
  - É frequentemente usado para determinar o tamanho de um tipo de dado, especialmente ao alocar memória dinamicamente ou ao calcular o tamanho necessário para armazenar dados em um arquivo.
  -
- **feof():**
  - A função **feof()** é usada para verificar se o indicador de final de arquivo (EOF) foi ativado para um arquivo.
  - Ela aceita um ponteiro para um arquivo como argumento e retorna um valor diferente de zero se o indicador de final de arquivo estiver definido para o arquivo, indicando que a próxima operação de leitura atingirá o final do arquivo.
  - É frequentemente usada em loops de leitura para verificar se todas as informações no arquivo foram lidas.

É importante ressaltar que ao lidar com arquivos de texto em C, é essencial considerar questões de segurança, como manipulação adequada de erros e validação de entradas do usuário para evitar vulnerabilidades, como estouro de buffer e ataques de injeção.

3.2.1 Exemplo de uso das funções fread(), fwrite() e fseek()

O exemplo que exploraremos manipula um arquivo binário arqcad.dat contendo dados de contato, ou seja, o nome e o ano. Em C, por padrão, as operações em um arquivo ocorrem em posições sucessivas dentro do arquivo, o que significa que cada operação de leitura ou escrita segue a operação anterior, progredindo até o final do arquivo. Esse modo de acesso ao arquivo é conhecido como acesso sequencial.

Frequentemente, um programa precisa ler ou escrever em posições específicas de um arquivo, ou retomar a leitura de uma posição anteriormente visitada. Isso, por exemplo, é comum em aplicativos que lidam com arquivos muito grandes, como vídeos ou bancos de dados. Para realizar essas operações, é necessário ter um método para acessar diretamente posições específicas do arquivo.

Em C, o acesso direto a posições específicas de um arquivo é realizado por meio de funções de posicionamento de ponteiro. Essas funções permitem modificar a posição do ponteiro do arquivo antes de executar a operação de leitura/escrita desejada. Todas as funções que manipulam o ponteiro de arquivo consideram as posições em bytes a partir do início do arquivo e nunca em termos do número de blocos. Todas estas funcionalidades e as funções e operadores necessários estão descritas e programadas nos trechos de códigos abaixo:

3.2.1.1 Gravar dados no arquivo binário:-

O trecho de código, abaixo, permite ao usuário cadastrar contatos em um arquivo binário ("arqcad.dat"), com a opção de continuar cadastrando novos contatos até que ele decida parar. Cada contato é composto por um nome e um ano e os dados são armazenados em uma estrutura (struct) **cad** e gravados no arquivo.

Vamos analisar o que o trecho de código faz:

1. **Abertura do arquivo:**
    - O trecho **fp = fopen("arqcad.dat", "ab")** tenta abrir o arquivo "arqcad.dat" no modo de abertura binária para adicionar dados ao final do arquivo (**"ab"**). Se a abertura falhar (ou seja, se **fp** for **NULL**), uma mensagem de erro é exibida e o programa é encerrado.
2. **Cadastramento de dados:**
    - Em seguida, o programa limpa a tela e exibe "Cadastrando".
    - O usuário é solicitado a fornecer um nome (**gets(aux)**) e um ano (**cin >> cad.ano**).

- Os dados são então gravados no arquivo usando a função **fwrite()**, que escreve a estrutura **cad** no arquivo **fp**.

3. **Pergunta para cadastrar novo contato:**
   - Após o cadastramento, o programa pergunta se deseja cadastrar um novo contato, lendo a resposta do usuário e convertendo-a para maiúsculas com **strupr(&opc)**.
   - O loop **do-while** continua até que o usuário responda "N" (indicando que não deseja cadastrar outro contato).
4. **Fechamento do arquivo:**
   - Quando o usuário decide não cadastrar mais contatos, o arquivo é fechado com **fclose(fp)**.
5. **Loop principal:**
   - O código todo está envolvido em um loop principal (**do-while**), que permite que o processo de cadastramento seja repetido enquanto o usuário desejar.

Trecho de código para a gravar dados no arquivo binário:

```
//cadastro
   do {
      if ((fp = fopen ("arqcad.dat","ab")) == NULL) {
         cout << "Erro ao abrir arquivo para gravação!";
         exit(0);
      }
      system("cls");
      cout << "Cadastrando  ";

      fflush(stdin);
      cout<<"Nome: ";
      gets(aux);
      strcpy(cad.nome,aux);
      cout<<"Ano: ";
      cin>>cad.ano;
      fwrite (&cad, sizeof (cad), 1, fp);

      do {
         cout << endl << "Cadastrar novo produto? [S/N]" << endl;
         cin >> opc;
         strupr(&opc);
      } while ((opc != 'N') && (opc != 'S'));
      fclose(fp);
   } while (opc != 'N');
```

3.2.2.2 Mostrar dados gravados no arquivo binário:-

O trecho de código, abaixo, abre um arquivo binário que armazena dados de contatos, posiciona o ponteiro de arquivo no último registro, lê e exibe todos os registros de contatos do arquivo na tela, e finalmente fecha o arquivo.

Vamos analisar o que o trecho de código faz:

1. **Abertura do arquivo para leitura e escrita:**
    - O trecho **fp = fopen("arqcad.dat", "r+b")** tenta abrir o arquivo "arqcad.dat" no modo de abertura binária para leitura e escrita. Se a abertura falhar (ou seja, se **fp** for **NULL**), uma mensagem de erro é exibida e o programa é encerrado.
2. **Posicionamento no final do arquivo:**
    - A função **fseek(fp, ftell(fp) - sizeof(cad), SEEK_SET)** move o ponteiro de posição do arquivo para o final menos o tamanho da estrutura **cad**. Isso é feito para que o ponteiro esteja posicionado antes do último registro no arquivo.
3. **Leitura do último registro:**
    - A função **fread(&cad, sizeof(cad), 1, fp)** lê o último registro do arquivo e armazena-o na variável **cad**.
4. **Exibição dos dados cadastrados:**
    - O programa exibe a mensagem "Segue lista de todos os contatos cadastrados:" para indicar que está prestes a mostrar os dados cadastrados.
    - Em seguida, entra em um loop **while** que continua até que o indicador de final de arquivo (**feof(fp)**) seja verdadeiro, o que significa que o final do arquivo foi alcançado.
    - Durante cada iteração do loop, os dados do registro atual (armazenados na variável **cad**) são exibidos na tela, mostrando o nome e o ano do contato.
5. **Leitura do próximo registro:**
    - Após exibir os dados do registro atual, a função **fread(&cad, sizeof(cad), 1, fp)** é chamada novamente para ler o próximo registro no arquivo. Isso permite que o programa continue exibindo os dados dos registros subsequentes até chegar ao final do arquivo.
6. **Fechamento do arquivo:**

- Após a conclusão do loop, o arquivo é fechado com **fclose(fp)**.

```
//mostrar todos
if ((fp = fopen ("arqcad.dat", "r+b")) == NULL) {
        cout << "Erro ao abrir arquivo para leitura!";
        exit (0);
        system ("pause");
        return 0;
    }
        fseek(fp, ftell(fp) - sizeof(cad), SEEK_SET);
        fread(&cad, sizeof(cad), 1, fp);
        cout << "Segue lista de todos os contatos cadastrados: " << endl;
        while (!feof(fp)) {
            cout<<"Nome: ";
            cout<<cad.nome;
            cout<<"     Ano: ";
            cout<<cad.ano<<endl;
            fread(&cad, sizeof(cad), 1, fp);
        }
    fclose (fp);
```

Em resumo, as operações de gravação e busca de dados em arquivos de texto em C fornecem aos programadores ferramentas poderosas para armazenar e recuperar informações de forma eficiente. Com um entendimento sólido dessas operações e boas práticas de programação, é possível desenvolver aplicativos robustos e seguros que interajam com arquivos de texto de maneira confiável e eficaz.