Manipulação de arquivos

Linguagem de programação
Prof. Allan Rodrigo Leite

Arquivos

- Objetivos de um programa manipular arquivos
  - Permitem armazenar grande quantidade de informação
    - Custo da memória secundária é muito inferior à memória primária
  - Persistência dos dados em uma memória não volátil
    - Ao finalizar o programa ou desligar o computador, os dados serão mantidos
  - Diferentes formas de acesso aos dados
    - Os acessos podem ser aleatório ou sequencial
  - Acesso concorrente aos dados
    - Mais de um programa pode usar os mesmos dados ao mesmo tempo
    - Inclusive programas em computadores distintos

Tipos de arquivos

- Em geral, as linguagens de programação suportam dois tipos
  - Arquivos texto (*plain-text*) ou binário (*binary*)

- Arquivo texto
  - Armazena caracteres que podem ser
    - Exibidos diretamente na tela
    - Modificados por editores de textos simples

  - Os dados são gravados como caracteres de 8 bits
    - Exemplo: um número inteiro com 8 dígitos ocupará 64 bits no arquivo
    - Embora o mesmo número poderia ser armazenado como um `int` (32 bits)

Tipos de arquivos

- Arquivo binário
  - Sequência de bits sujeita às convenções do programa que o gerou
  - A diferença é que os dados são gravados na forma binária
    - Isto é, do mesmo modo que estão na memória
    - Exemplo: um número inteiro com 8 dígitos ocupará 32 bits no arquivo

  - Exemplos
    - Arquivos executáveis
    - Arquivos compactados
    - Arquivos de registros

Manipulando arquivos

- A linguagem C possui diversas funções para manipular arquivos
  - Estas funções encontram-se na biblioteca padrão `stdio.h`

- No entanto, não há funções de alto nível para manipular arquivo
  - Suas funções se limitam a abrir, fechar e ler caracteres/bytes
  - Portanto, é tarefa do desenvolvedor a criação da função que irá ler um arquivo de uma maneira específica

- Estas funções utilizam uma abstração de ponteiro de arquivo
  - Exemplo de declaração um ponteiro de arquivo

```
FILE *p; //p é o ponteiro para arquivos que permitirá a manipulação de arquivos
```

Abrindo arquivos

- Para a abertura de um arquivo, usa-se a função `fopen`

```
FILE* fopen(char nome[], char modo[]);
```

- O parâmetro `nome` determina qual arquivo deverá ser aberto
  - Deve ser um caminho válido para o sistema operacional em uso
  - É possível trabalhar com caminhos absolutos ou relativos
  - Caminho absoluto
    - Descrição de um caminho desde o diretório raiz
    - Exemplo: `/home/allan/dados.txt`
  - Caminho relativo
    - Descrição de um caminho desde o diretório corrente (local do programa)
    - Exemplo: `./arquivo/dados.txt`

Abrindo arquivos

- O parâmetro modo determina o tipo de abertura de uso
  - A tabela a seguir mostra os modo válidos de abertura de um arquivo

Modo	Tipo	Função
`"r"`	Texto	Leitura, o arquivo deve existir.
`"w"`	Texto	Escrita, cria o arquivo se não houver e apaga o anterior se existir.
`"a"`	Texto	Escrita, os dados serão adicionados no fim do arquivo (*append*).
`"rb"`	Binário	Leitura, o arquivo deve existir.
`"wb"`	Binário	Escrita, cria arquivo se não houver e apaga o anterior se existir.
`"ab"`	Binário	Escrita, os dados serão adicionados no fim do arquivo (*append*).
`"r+"`	Texto	Leitura e escrita, o arquivo deve existir e pode ser modificado.
`"w+"`	Texto	Leitura e escrita, cria arquivo se não houver e apaga o anterior se existir.
`"a+"`	Texto	Leitura e escrita, os dados serão adicionados no fim do arquivo (*append*).
`"r+b"`	Binário	Leitura e escrita, o arquivo deve existir e pode ser modificado.
`"w+b"`	Binário	Leitura e escrita, cria o arquivo se não houver e apaga o anterior se existir.
`"a+b"`	Binário	Leitura e escrita, os dados serão adicionados no fim do arquivo (*append*).

Abrindo arquivos

- Exemplo de um arquivo binário pode ser aberto para escrita
  - Caso a função `fopen` retorne NULL, não foi possível abrir o arquivo

```c
int main() {
     FILE *arquivo = fopen("exemplo.bin","wb");

     if (arquivo == NULL) {
          printf("Erro na abertura do arquivo.\n");
     } else {
          printf("Arquivo aberto.\n");
     }

     fclose(arquivo);
}
```

Posição do arquivo

- Existe uma noção de posição no arquivo para leitura ou gravação
  - Esta posição indica onde será lido ou escrito o próximo caractere
  - As operações de leitura e escrita em arquivos são análogas à digitação em uma máquina de escrever

- Acesso sequencial ou aleatório
  - Em acesso sequencial raramente é necessário modificar essa posição
    - Ao ler um caractere, a posição no arquivo é automaticamente atualizada
  - Em acesso aleatório, frequentemente é necessário percorrer o arquivo
    - Encontrar uma determinada informação, seja para leitura ou para gravação

Fechando um arquivo

- Recomenda-se fechar o arquivo após concluir a manipulação dele
  - Para esta finalidade, usa-se a função `fclose`

    ```
    fclose(FILE *arquivo);
    ```

  - O ponteiro `arquivo` indica o arquivo a ser fechado
    - A função retorna `0` em caso de sucesso

- Ao fechar um arquivo, todo caractere contido no *buffer* será gravado
  - *Buffer* é uma região de memória que mantém temporariamente os dados
  - A técnica é utilizada para aumentar a eficiência do acesso ao dispositivo
  - O conteúdo do *buffer* é gravado em memória só quando estiver cheio

Lendo e escrevendo em arquivo

- Após abrir um arquivo, é possível ler ou escrever nele
  - A linguagem C conta com um conjunto de funções de leitura/escrita
    - Possuem diferentes funcionalidade para atender diferentes aplicações

- A maneira mais simples de manipular arquivos é a partir de uma *string*
  - Isto é, considerando que os dados são uma cadeia de caracteres

- Para se escrever uma *string* em um arquivo usa-se a função `fputs`
  - Recebe como parâmetro uma *string* e o ponteiro do arquivo a ser escrito

```
int fputs(char string[], FILE *arquivo);
```

Lendo e escrevendo em arquivo

- A função `fputs` retorna
  - `0` se a escrita ocorreu com sucesso
  - EOF se ocorreu algum problema durante a escrita
  - Recebe como parâmetro uma *string* e o ponteiro do arquivo a ser escrito
- A função `fputs` pode ser utilizada para escrever uma *string* na tela

```
int main() {
     char frase[20] = "Olá mundo";
     fputs(frase, stdout);

     return 0;
}
```

Lendo e escrevendo em arquivo

- Exemplo da função `fputs`

```
int main() {
     char texto[20] = "Olá mundo!";

     FILE *arquivo = fopen("gravar.txt","w");

     if (arquivo != NULL) {
          int retorno = fputs(texto,arquivo);

          if (retorno == EOF) {
               printf("Erro na gravação do arquivo.");
          }

          fclose(arquivo);
     } else {
          printf("Erro na criação do arquivo.\n");
     }
}
```

Lendo e escrevendo em arquivo

- Similar ao `fputs`, é possível realizar a leitura de uma cadeia de caracteres com a função `fgets`

```
char* fgets(char texto[], int tamanho, FILE *arquivo);
```

  - A função fgets recebe três parâmetros
    - `texto`: variável onde será armazenado o conteúdo lido do arquivo
    - `tamanho`: o número máximo de caracteres a serem lidos
    - `arquivo`: ponteiro do arquivo cujo conteúdo será lido
  - O retorno da função fgets será:
    - `NULL`: em caso de erro ou fim do arquivo
    - `ponteiro`: se há conteúdo, retorna o ponteiro para o primeiro caractere lido

Lendo e escrevendo em arquivo

- Funcionamento da função `fgets`
  - A partir de um ponteiro de arquivo, esta função lê uma *string* até
    - Encontrar um caractere de nova linha seja lido, ou
    - Atingir `tamanho - 1` caracteres lidos
  - Se o caractere de nova linha (`'\n'`) for lido, ele fará parte da *string*
    - Portanto, a *string* resultante sempre terminará com `'\0'`
    - Assim, no máximo `tamanho - 1` caracteres serão lidos
  - Se ocorrer algum erro, a função `fgets` devolverá um ponteiro nulo

Lendo e escrevendo em arquivo

- Exemplo da função `fgets`

```
int main() {
     char texto[20];
     FILE *arquivo = fopen("gravar.txt","r");

     if (arquivo != NULL) {
          char *retorno = fgets(texto,20,arquivo);

          if (retorno != NULL)
               printf("Lido %s\n",texto);
          else
               printf("Erro na leitura do arquivo.");

          fclose(arquivo);
     } else {
          printf("Erro na abertura do arquivo.\n");
     }
}
```

Lendo e escrevendo em arquivo

- Funções de fluxo padrão
  - Permitem ler/escrever em arquivos da forma padrão (`printf` e `scanf`)
  - As funções fprintf e fscanf funcionam de maneiras semelhantes
    - Só que elas direcionam os dados para arquivos, ao invés do `stdin`/`stdout`

  - Função `fprintf`

```
printf("Nome: %s", nome); //exibe na tela (console)
fprintf(arquivo, "Nome: %s", nome); //grava no arquivo
```

  - Função `fscanf`

```
scanf("%s", &nome); //le do teclado
fscanf(arquivo, "%s", &nome); //le do arquivo
```

Fim de arquivo

- A constante EOF (*End Of File*) indica o fim de um arquivo
  - No entanto, é possível utilizar a função `feof` para verificar se um arquivo chegou ao fim

```c
int feof(FILE *arquivo);
```

- Comportamento da função `feof`
  - Testa o indicador de fim de arquivo para o fluxo apontado pelo ponteiro
  - A função retorna um valor inteiro diferente de zero se, e somente se, o indicador de fim de arquivo está marcado para o ponteiro
    - A função testa o indicador de fim de arquivo, mas não o próprio arquivo

Fim de arquivo

- Comportamento da função `feof` (cont.)
  - Isso significa que outra função é responsável por alterar o indicador para informar que o EOF foi alcançado
  - A maioria das funções de leitura altera o indicador ao ler todos os dados
    - E então realizar uma nova leitura resultando em nenhum dado, apenas EOF
  - Assim devemos evitar o uso da função feof no teste lógico de um laço
    - Usa-se para testar se uma leitura alterou o indicador de fim de arquivo

Exercícios

1. Faça um programa que leia um arquivo texto contendo valores numéricos separados por `<tab>`. Em seguida, o programa deve alimentar uma matriz $A_{M \times N}$ gerada dinamicamente a partir do número *M* de linhas e *N* colunas existentes no arquivo texto. Por fim, deve ser gerada uma matriz transposta $M^t_{N \times M}$ e gravá-la em outro arquivo.
   - Exemplo, dado o arquivo `"1 2 3\n4 5 6"`, a saída esperada é uma matriz `"1 4\n2 5\n3\n6"`.

2. Faça um programa que leia um arquivo texto e, em seguida, leia uma palavra para ser buscada dentro do arquivo texto. Por fim, o programa deve retornar o número de vezes que a palavra foi encontrada dentro do arquivo.
   - Exemplo, dada o arquivo `"abcd\nbabcde\n"` e a *string* `"ab"`, a saída esperada é 2.

3. Faça um programa para manter as informações de uma agenda de contatos utilizando estruturas. O programa deve conter um menu inicial com 5 opções: i) incluir um novo contato; ii) excluir um contato existente; iii) alterar um contato existente; iv) listar todos os contatos cadastrados; e v) finalizar o programa. A estrutura do contato deve conter um código de identificação, nome, e-mail e celular. Os dados da agenda de contatos devem ser salvos em arquivo, garantindo que ao fechar o programa, os dados serão mantidos.

Manipulação de arquivos

Linguagem de programação
Prof. Allan Rodrigo Leite