Arquivos

Aula 08

Ivone P. Matsuno Yugoshi
ivone.matsuno@ufms.br

Ronaldo Fiorilo dos Santos
ronaldo.santos@ufms.br

Universidade Federal de Mato Grosso do Sul
Câmpus de Três Lagoas
Bacharelado em Sistemas de Informação

Algoritmos e Programação II

Sequências de caracteres

- Na linguagem C, o termo **sequência de caracteres**, do inglês *stream*, significa qualquer fonte de entrada ou qualquer destinação para saída de informações
- Os programas que produzimos até aqui sempre obtiveram toda sua entrada a partir de uma sequência de caracteres, em geral associada ao teclado, e escreveram sua saída em outra sequência de caracteres, associada com o monitor
- Programas maiores podem necessitar de sequências de caracteres adicionais, associadas a arquivos armazenados em uma variedade de meios físicos tais como discos e memórias ou ainda portas de rede e impressoras.

Sequências de caracteres

- As funções de entrada e saída mantidas em `stdio.h` trabalham do mesmo modo com todas as sequências de caracteres, mesmo aquelas que não representam arquivos físicos.
- O acesso a uma sequência de caracteres na linguagem C se dá através de um ponteiro de arquivo, cujo tipo é `FILE *`, declarado em `stdio.h` .
- Podemos ainda declarar ponteiros de arquivos conforme nossas necessidades, como fazemos abaixo:

```
FILE *pt1, *pt2;
```

Sequência de caracteres

- A biblioteca representada pelo arquivo-cabeçalho `stdio.h` suporta dois tipos de arquivos:
  - Arquivo texto: Os bytes em um arquivo-texto representam caracteres, fazendo que seja possível examinar e editar o seu conteúdo. O arquivo-fonte de um programa na linguagem C, por exemplo, é armazenado em um arquivo-texto.
  - Arquivo binário: Por outro lado, os bytes em um arquivo-binário não representam necessariamente caracteres. Grupos de bytes podem representar outros tipos de dados tais como inteiros e números com ponto flutuante. Um programa executável, por exemplo, é armazenado em um arquivo-binário.
- Arquivos-texto possuem duas principais características que os diferem dos arquivos-binários: são divididos em linhas e podem conter um marcador especial de fim de arquivo.

Funções de entrada e saída da linguagem C

- Para que se possa realizar qualquer operação sobre um arquivo é necessário, antes de tudo, abrir esse arquivo.
- Como um programa pode ter de trabalhar com diversos arquivos, então todos eles deverão estar abertos durante a execução desse programa.
- Dessa forma, há necessidade de identificação de cada arquivo, o que é implementado na linguagem C com o uso de um ponteiro para arquivo.

Funções de abertura e fechamento

- A função `fopen` da biblioteca padrão de entrada e saída da linguagem C é uma função que realiza a abertura de um arquivo no sistema.
- A função recebe como parâmetros duas cadeias de caracteres: a primeira é o identificador/nome do arquivo a ser aberto e a segunda determina o modo no qual o arquivo será aberto.
  - O nome do arquivo deve constar no sistema de arquivos do sistema operacional.

Funções de abertura e fechamento

- A função `fopen` devolve um ponteiro único para o arquivo que pode ser usado para identificar esse arquivo a partir desse ponto do programa.
  - Esse ponteiro é posicionado no início ou no final do arquivo, dependendo do modo como o arquivo foi aberto.
- Se o arquivo não puder ser aberto por algum motivo, a função devolve o ponteiro com valor `NULL`.
- Um ponteiro para um arquivo deve ser declarado com um tipo pré-definido `FILE`, também incluso no arquivo-cabeçalho `stdio.h`.
- A interface da função fopen é então descrita da seguinte forma:

```
FILE *fopen(char *nome, char *modo)
```

Funções de abertura e fechamento

- As opções para a cadeia `modo`, parâmetro da função `fopen`, são:

modo	Descrição
r	modo de leitura de texto
w	modo de escrita de texto †
a	modo de adicionar texto ‡
r+	modo de leitura e escrita de texto
w+	modo de leitura e escrita de texto †
a+	modo de leitura e escrita de texto ‡
rb	modo de leitura em binário
wb	modo de escrita em binário †
ab	modo de adicionar em binário ‡
r+b ou rb+	modo de leitura e escrita em binário
w+b ou wb+	modo de leitura e escrita em binário †
a+b ou ab+	modo de leitura e escrita em binário ‡

† trunca o arquivo existente com tamanho 0 ou cria novo arquivo;
‡ abre ou cria o arquivo e posiciona o ponteiro no final do arquivo.

Funções de abertura e fechamento

- Algumas observações sobre os modos de abertura de arquivos se fazem necessárias. Primeiro, observe que se o arquivo não existe e é aberto com o modo de leitura (`r`) então a abertura falha.
- Também, se o arquivo é aberto com o modo de adicionar (`a`), então todas as operações de escrita ocorrem o final do arquivo, desconsiderando a posição atual do ponteiro do arquivo.
- Por fim, se o arquivo é aberto no modo de atualização (`+`) então a operação de escrita não pode ser imediatamente seguida pela operação de leitura, e vice-versa, a menos que uma operação de reposicionamento do ponteiro do arquivo seja executada, tal como uma chamada a qualquer uma das funções `fseek` , `fsetpos` , `rewind` , etc.

Funções de abertura e fechamento

- Por exemplo, o comando de atribuição a seguir

```
ptarq = fopen("entrada", "r");
```

  - tem o efeito de abrir um arquivo com nome `entrada` no modo de leitura.
- A chamada à função `fopen` devolve um identificador para o arquivo aberto que é atribuído ao ponteiro `ptarq` do tipo `FILE`. Então, esse ponteiro é posicionado no primeiro caracter do arquivo.
- A declaração prévia do ponteiro `ptarq` deve ser feita da seguinte forma: `FILE *ptarq;`

Funções de abertura e fechamento

- A função `fclose` faz o oposto que a função `fopen` faz, ou seja, informa o sistema que o programa não precisa mais usar o arquivo
- Quando um arquivo é fechado, o sistema realiza algumas tarefas importantes, especialmente a escrita de quaisquer dados que o sistema possa ter mantido na memória principal para o arquivo na memória secundária, e então dissocia o identificador do arquivo
- Depois de fechado, não podemos realizar tarefas de leitura ou escrita no arquivo, a menos que seja reaberto. A função `fclose` tem a seguinte interface:

```
int fclose(FILE *ptarq)
```

- Se a operação de fechamento do arquivo apontado por `ptarq` obtém sucesso, a função `fclose` devolve o valor 0 (zero). Caso contrário, o valor `EOF` é devolvido.

Funções de entrada e saída

- A função `fgetc` da biblioteca padrão de entrada e saída da linguagem C permite que um único caracter seja lido de um arquivo. A interface dessa função é apresentada a seguir:

```
int fgetc(FILE *ptarq)
```

- A função `fgetc` lê o próximo caracter do arquivo apontado por `ptarq`, avançando esse ponteiro em uma posição.
- Se a leitura é realizada com sucesso, o caracter lido é devolvido pela função.

Funções de entrada e saída

- Note, no entanto, que a função, ao invés de especificar o valor de devolução como sendo do tipo `unsigned char`, especifica-o como sendo do tipo `int`.
- Isso se deve ao fato de que a leitura pode falhar e, nesse caso, o valor devolvido é o valor armazenado na constante simbólica `EOF`, definida no arquivo-cabeçalho `stdio.h`.
  - O valor correspondente à constante simbólica `EOF` é obviamente um valor diferente do valor de qualquer caracter e, portanto, um valor negativo.
- Do mesmo modo, se o fim do arquivo é encontrado, a função `fgetc` também devolve `EOF`.

Funções de entrada e saída

- A função `fputc` permite que um único caracter seja escrito em um arquivo. A interface dessa função é apresentada a seguir:

```
int fputc(int caracter, FILE *ptarq)
```

- Se a função `fputc` tem sucesso, o ponteiro `ptarq` é incrementado e o caracter escrito é devolvido. Caso contrário, isto é, se ocorre um erro, o valor `EOF` é devolvido.

Funções de entrada e saída

- Existe outro par de funções de leitura e escrita em arquivos com identificadores `fscanf` e `fprintf`.
- As interfaces dessas funções são apresentadas a seguir:

```
int fscanf(FILE *ptarq, char *formato, ...)
```

e

```
int fprintf(FILE *ptarq, char *formato, ...)
```

- Essas funções são semelhantes às respectivas funções `scanf` e `printf` que conhecemos bem, a menos de um parâmetro a mais que é informado, justamente o primeiro, que é o ponteiro para o arquivo que ser quer realizar as operações de entrada e saída formatadas.

Funções de entrada e saída

- Dessa forma, o exemplo de chamada a seguir

```
fprintf(ptarq, "O número %d é primo\n", numero);
```

  - realiza a escrita no arquivo apontado por `ptarq` da mensagem entre aspas duplas, substituindo o valor numérico correspondente armazenado na variável numero.
- O número de caracteres escritos no arquivo é devolvido pela função `fprintf`.
- Se um erro ocorrer, então o valor -1 é devolvido.

Funções de entrada e saída

- Do mesmo modo,

```
fscanf(ptarq, "%d", &numero);
```

  - realiza a leitura de um valor que será armazenado na variável numero a partir de um arquivo identificado pelo ponteiro `ptarq`.
- Se a leitura for realizada com sucesso, o número de valores lidos pela função `fscanf` é devolvido.
- Caso contrário, isto é, se houver falha na leitura, o valor `EOF` é devolvido.

Funções de entrada e saída

- Há outras funções para entrada e saída de dados a partir de arquivos, como as funções `fread` e `fwrite`, que auxiliam na manipulação de *structs* em arquivos.

```
int fread(void *ptr, int size, int nmemb, FILE *ptarq);
```

  - `*ptr` : ponteiro para o bloco de memória que vai armazenar os dados lidos
  - `size` : tamanho em bytes de cada elemento a ser lido
  - `nmeb` : número de elementos (de tamanho `size`) a serem lidos do arquivos
  - `*ptarq` : ponteiro para o arquivo

- Se a leitura for realizada com sucesso, o número de elementos lidos pela função é devolvido. Caso contrário, o valor devolvido será diferente de `nmemb` .

Funções de entrada e saída

- A função `fwrite` funciona de forma parecida:

```
int fwrite(void *ptr, int size, int nmemb, FILE *ptarq);
```

- Se a escrita for feita corretamente no arquivo, a função devolve a quantidade de elementos escritos. Caso contrário é devolvido um valor diferente de `nmemb`.

Funções de entrada e saída

- Suponha a existência da seguinte *struct*:

```
typedef struct {
   char nome[50];
   double salario;
   int matricula;
} func;
```

- O trecho de código a seguir representa a leitura de um registro do tipo `func` e em seguida a escrita no arquivo do mesmo registro.

```
func f;
fread(&f, sizeof(func), 1, ptarq);
.
.
.
fwrite(&f, sizeof(func), 1, ptarq);
```

Funções de controle

- Sobre as funções que tratam do posicionamento do ponteiro de um arquivo, existe uma função específica que realiza um teste de final de arquivo.

```
int feof(FILE *ptarq)
```

- O argumento da função `feof` é um ponteiro para um arquivo do tipo `FILE`.
- A função devolve um valor inteiro diferente de 0 (zero) se o ponteiro `ptarq` está posicionado no final do arquivo. Caso contrário, a função devolve o valor 0 (zero).

Funções de controle

- A função `fgetpos` determina a posição atual do ponteiro do arquivo e tem a seguinte interface:

```
int fgetpos(FILE *ptarq, fpos_t *pos)
```

  - onde `ptarq` é o ponteiro associado a um arquivo e `pos` é uma variável que, após a execução dessa função, conterá o valor da posição atual do ponteiro do arquivo.
- Observe que `fpos_t` é um novo tipo de dado, definido no arquivo-cabeçalho `stdio.h`, adequado para armazenamento de uma posiçã qualquer de um arquivo.
- Se a obtenção dessa posição for realizada com sucesso, a função devolve 0 (zero). Caso contrário, a função devolve um valor diferente de 0 (zero).

Funções de controle

- A função `fsetpos` posiciona o ponteiro de um arquivo em alguma posição escolhida e tem a seguinte interface:

```
int fsetpos(FILE *ptarq, fpos_t *pos)
```

  - onde `ptarq` é o ponteiro para um arquivo e `pos` é uma variável que contém a posição para onde o ponteiro do arquivo será deslocada.

- Se a determinação dessa posição for realizada com sucesso, a função devolve 0 (zero). Caso contrário, a função devolve um valor diferente de 0 (zero).

Funções de controle

- A função `ftell` determina a posição atual de um ponteiro em um dado arquivo. Sua interface é apresentada a seguir:

```
long int ftell(FILE *ptarq)
```

- Essa função devolve a posição atual no arquivo apontado por `ptarq`.
  - Se o arquivo é binário, então o valor é o número de bytes a partir do início do arquivo.
  - Se o arquivo é de texto, então esse valor pode ser usado pela função `fseek`, como veremos a seguir.
- Se há sucesso na sua execução, a função devolve a posição atual no arquivo. Caso contrário, a função devolve o valor -1L.

Funções de controle

- A função **`fseek`** posiciona o ponteiro de um arquivo para uma posição determinada por um deslocamento:

```
int fseek(FILE *ptarq, long int desloca, int a_partir)
```

  - O argumento **`ptarq`** é o ponteiro para um arquivo.
  - O argumento **`desloca`** é o número de bytes a serem saltados a partir do conteúdo do argumento **`a_partir`**, que pode ser:

SEEK_SET	A partir do início do arquivo
SEEK_CUR	A partir da posição atual
SEEK_END	A partir do fim do arquivo

    - Em um arquivo de texto, **`a_partir`** dever ser **`SEEK_SET`** e o conteúdo de **`desloca`** deve ser 0 (zero) ou um valor devolvido pela função **`ftell`**.

- Se a função é executada com sucesso, o valor 0 (zero) é devolvido. Caso contrário, um valor diferente de 0 (zero) é devolvido.

Funções de controle

- A função `rewind` faz com que o ponteiro de um arquivo seja posicionado para o início desse arquivo:

```
void rewind(FILE *ptarq)
```

Funções sobre arquivos

- Há funções na linguagem C que permitem que um programador remova um arquivo do disco ou troque o nome de um arquivo.

```
int remove(char *nome)
```

- A função `remove` elimina um arquivo, com nome armazenado na cadeia de caracteres `nome`, do sistema de arquivos do sistema computacional.
  - O arquivo não deve estar aberto no programa.
- Se a remoção é realizada, a função devolve o valor 0 (zero). Caso contrário, um valor diferente de 0 (zero) é devolvido.

Funções sobre arquivos

- A função `rename` tem a seguinte interface:

```
int rename(char *antigo, char *novo)
```

- A função `rename` faz com que o arquivo com nome armazenado na cadeia de caracteres `antigo` tenha seu nome trocado pelo nome armazenado na cadeia de caracteres `novo`.
- Se a função realiza a tarefa de troca de nome, então `rename` devolve o valor 0 (zero). Caso contrário, um valor diferente de 0 (zero) é devolvido e o arquivo ainda pode ser identificado por seu nome antigo.

Arquivos do sistema

- Sempre que um programa na linguagem C é executado, três arquivos ou sequências de caracteres são automaticamente abertos pelo sistema, identificados pelos ponteiros do tipo `FILE`:
  - `stdin`: identifica a entrada padrão do programa e é normalmente associado ao teclado.
  - `stdout`: identifica a saída padrão do programa e é normalmente associado ao monitor (terminal).
  - `stderr`: identifica a saída de erros padrão do programa e é normalmente associado ao monitor (terminal).

Arquivos do sistema

- Todas as funções de entrada definidas na linguagem C que executam entrada de dados e não têm um ponteiro do tipo `FILE` como um argumento tomam a entrada a partir do arquivo apontado por `stdin`.
- Assim, ambas as chamadas a seguir:

```
scanf("%d", &numero);
```

e

```
fscanf(stdin, "%d", &numero);
```

- são equivalentes e lêem um número do tipo inteiro da entrada padrão, que é normalmente o teclado.

Arquivos do sistema

- Do mesmo modo, o ponteiro `stdout` se refere à saída padrão, que também é associada ao terminal. Assim, as chamadas a seguir:

```
printf("Programar é bacana!\n");
```

e

```
fprintf(stdout, "Programa é bacana!\n");
```

- são equivalentes e imprimem a mensagem acima entre as aspas duplas na saída padrão, que é normalmente o terminal.

Arquivos do sistema

- O ponteiro `stderr` se refere ao arquivo padrão de erro, onde muitas das mensagens de erro produzidas pelo sistema são armazenadas e também é normalmente associado ao terminal.
- Uma justificativa para existência de tal arquivo é, por exemplo, quando as saídas todas do programa são direcionadas para um arquivo.
- Assim, as saídas do programa são escritas em um arquivo e as mensagens de erro são escritas na saída padrão, isto é, no terminal.
- Ainda há a possibilidade de escrever nossas próprias mensagens de erro no arquivo apontado por `stderr`.

Redirecionamento de entrada e saída

- Vimos na última aula sobre arquivos que sempre que um programa na linguagem C é executado, três arquivos ou sequências de caracteres são automaticamente abertos pelo sistema, identificados pelos ponteiros do tipo `FILE`:
  - `stdin`: identifica a entrada padrão do programa e é normalmente associado ao teclado.
  - `stdout`: identifica a saída padrão do programa e é normalmente associado ao monitor (terminal).
  - `stderr`: identifica a saída de erros padrão do programa e é normalmente associado ao monitor (terminal).
- Porém é possível usar operadores de redirecionamento para redirecionar os fluxos de entrada e saída dos locais padrão para locais diferentes.

Redirecionamento de entrada e saída

- Tomemos como um exemplo simples, o programa abaixo, onde um número inteiro na base decimal é fornecido como entrada e na saída é apresentado o mesmo número na base binária.

```
#include <stdio.h>
int main(void)
{
   int pot10, numdec, numbin;
   scanf("%d", &numdec);
   pot10 = 1;
   numbin = 0;
   while (numdec > 0)
   {
      numbin = numbin + (numdec % 2) * pot10;
      numdec = numdec / 2;
      pot10 = pot10 * 10;
   }
   printf("%d\n", numbin);
   return 0;
}
```

Redirecionamento de entrada e saída

- Supondo que o programa executável equivalente tenha sido criado após a compilação com o nome **`decbin`**
- Se queremos que a saída do programa executável **`decbin`** seja armazenada no arquivo **`resultado`**, podemos digitar em uma linha de terminal o seguinte:

```
prompt$ ./decbin > resultado
```

- Dessa forma, qualquer informação a ser apresentada por uma função de saída, como **`printf`**, não será mostrada no terminal, mas será escrita no arquivo **`resultado`**.

Redirecionamento de entrada e saída

- O operador $>$
  - O arquivo sempre será sobrescrito!
  - Caso o arquivo não exista, ele será criado.

```
prompt$ ./decbin > resultado
prompt$ ./decbin > resultado
```

    - Ao final das duas execuções, apenas o resultado da última execução estará no arquivo `resultado`.

- O operador $>>$
  - Semelhante ao $>$, com a diferença de não sobrescrever o arquivo.

```
prompt$ ./decbin » resultado
prompt$ ./decbin » resultado
```

    - Ao final das duas execuções, os dois resultados estarão no arquivo `resultado`, na ordem em que os programas foram executados.

Redirecionamento de entrada e saída

- O operador 2>
  - Supondo que o arquivo **`inexistente`** não exista, qual o resultado de digitar em uma linha de terminal o seguinte:

```
prompt$ cat inexistente > teste
```

    - Será exibida uma mensagem de erro e o arquivo **`teste`** será criado em branco.
  - O operador 2> tem a função de redirecionar somente as mensagens de erro para um arquivo.

```
prompt$ cat inexistente 2> erros
```

    - A mensagem de erro é redirecionada para o arquivo **`erros`**.
- O operador 2>>
  - Semelhante ao 2>>, com a diferença de não sobrescrever o arquivo.

Redirecionamento de entrada e saída

- Podemos também redirecionar a entrada de um programa executável, de tal forma que chamadas a funções que realizam entrada de dados as obtenha a partir de um arquivo.
- Por exemplo, se temos um arquivo com nome `numero` que contém um número inteiro, podemos digitar o seguinte em uma linha de terminal:

```
prompt$ ./decbin < numero
```

- Com esse redirecionamento, o programa `decbin`, que solicita um número a ser informado pelo usuário, não espera até que um número seja digitado.
- Ou seja, a chamada à função `scanf` lê um valor do arquivo `numero` e não do terminal, embora a função `scanf` não "saiba" disso.

Redirecionamento de entrada e saída

- É possivel combinar os operadores de redirecionamento de entrada, saída e erros de um programa simultaneamente da seguinte maneira:

```
prompt$ ./decbin < numero > resultado 2> erros
```

- Esse comando faz com que o programa **`decbin`** seja executado tomando a entrada de dados a partir do arquivo **`numero`**, escrevendo a saída de dados no arquivo **`resultado`** e caso haja alguma mensagem de erro ela será escrita no arquivo **`erros`**.
- Os operadores podem ser usados separadamente, combinados aos pares ou até mesmo usados os três ao mesmo tempo, como no exemplo acima.