Estruturas de Dados

Módulo 15 - Arquivos

Referências

Waldemar Celes, Renato Cerqueira, José Lucas Rangel, *Introdução a Estruturas de Dados*, Editora Campus (2004)

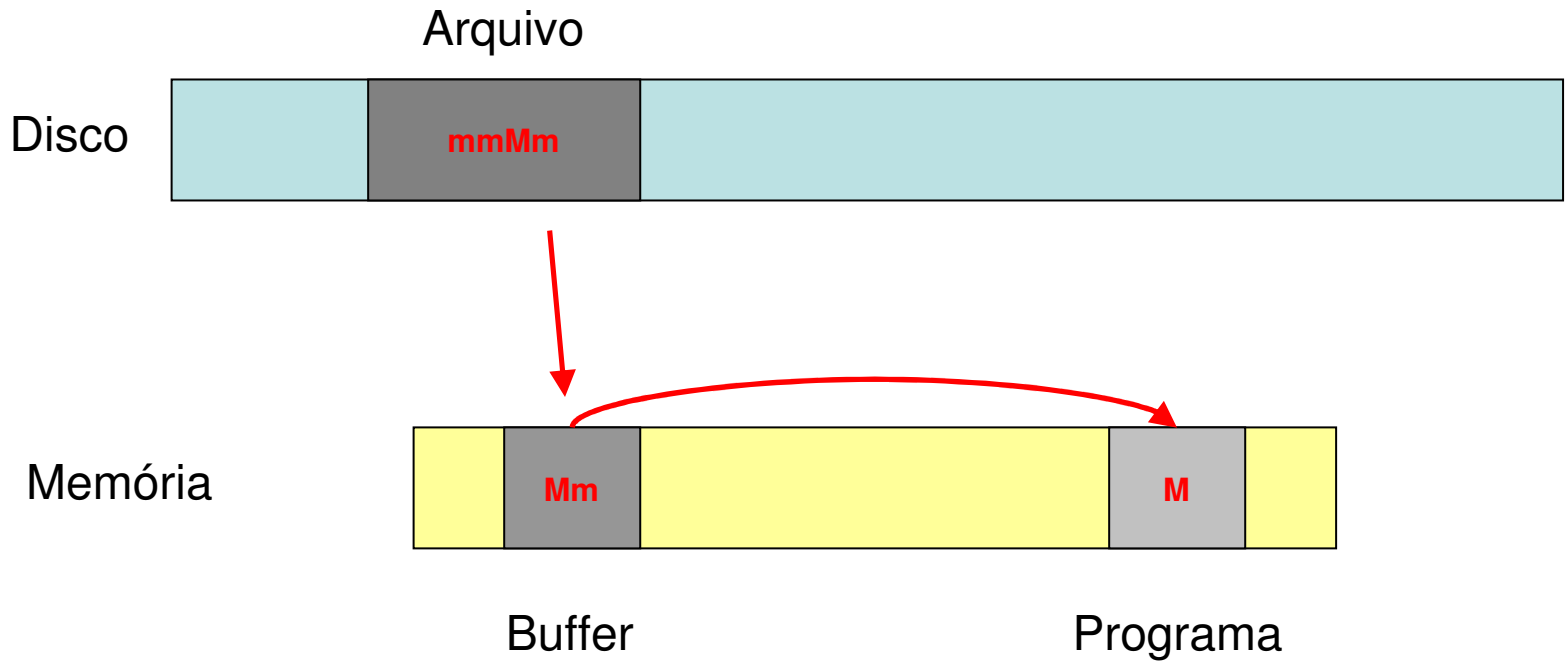
Capítulo 15 – Arquivos

Tópicos

- Introdução
- Funções para abrir e fechar arquivos
- Arquivos em modo texto
- Estruturação de dados em arquivos texto
- Arquivos em modo binário

Introdução

- Serviços oferecidos pelos sistema operacional (SO):



Introdução

- Serviços oferecidos pelos sistema operacional (SO):
  - Abertura de arquivo:
    - SO encontra o arquivo com o nome dado e prepara o buffer na memória
  - Leitura do arquivo:
    - SO recupera o trecho solicitado do arquivo
    - SO obtém os dados, na sua totalidade ou em parte, do buffer
  - Escrita no arquivo:
    - SO acrescenta ou altera o conteúdo do arquivo
    - SO altera dados no buffer para posteriormente transferi-los para disco
  - Fechamento de arquivo:
    - SO transfere todos os dados do buffer para disco
    - SO libera área do buffer

Funções para abrir e fechar arquivos

FILE* fopen (char* nome_arquivo, char* modo);

- valor de retorno:
  - ponteiro para o tipo FILE
    - tipo FILE definido pela biblioteca padrão para representar arquivos
    - todas as operações subseqüentes no arquivo receberão este endereço como parâmetro de entrada
  - NULL, se o arquivo não puder ser aberto

Funções para abrir e fechar arquivos

FILE* fopen (char* nome_arquivo, char* modo);

- parâmetro nome_arquivo
  - nome do arquivo a ser aberto
  - pode ser relativo ao diretório de trabalho do programa, ou
  - pode ser absoluto, incluindo os diretórios, desde o diretório raiz

Funções para abrir e fechar arquivos

FILE* fopen (char* nome_arquivo, char* modo);

- parâmetro modo

r	leitura (*read)*
w	escrita (*write)*
a	acréscimo ao final do arquivo (*append*)
t	texto (*text)*
b	binário (*binary)*
r+	leitura e escrita num arquivo já existente
w+	leitura e escrita num novo arquivo

Funções para abrir e fechar arquivos

- Comentários sobre os modo de abertura:
  - modos b e t:
    - podem ser combinados com os demais
  - modo w:
    - se o arquivo não existe, um novo é criado, inicialmente vazio
    - se o arquivo já existe, ele é destruído e um novo é criado, inicialmente vazio
    - fopen retorna NULL se o programa não tem acesso de escrita ao diretório
  - modo a:
    - arquivo é preservado e novos dados podem ser escritos no final do arquivo
  - modo r:
    - arquivo já deve existir, caso contrário a fopen falha e retorna NULL

Funções para abrir e fechar arquivos

- Exemplo:
  - solicita abertura de *entrada.txt* para leitura em modo texto
  - testa se a abertura do arquivo foi realizada com sucesso

```
...
FILE* fp;
fp = fopen("entrada.txt","rt");
if (fp == NULL) {
   printf("Erro na abertura do arquivo!\n");
   exit(1);
}
...
```

Funções para abrir e fechar arquivos

int fclose (FILE* fp);

- parâmetro:
  - ponteiro do arquivo que se deseja fechar
- valor de retorno:
  - a constante 0, se o arquivo for fechado com sucesso
  - a constante EOF (definida na biblioteca), se houve erro

Arquivos em modo texto

- Leitura de arquivos:
  - cada arquivo possui um cursor indicando a posição corrente
  - quando o arquivo é aberto para leitura:
    - o cursor é posicionado no início do arquivo
  - a cada leitura:
    - o dado lido é sempre aquele apontado pelo cursor do arquivo
    - o cursor avança e passa a apontar para a posição imediatamente após o dado lido
      - a próxima leitura captura o próximo dado do arquivo

Arquivos em modo texto

- Funções para ler dados de arquivos em modo texto:

```
int fscanf (FILE* fp, char* formato, ...);
int fgetc (FILE* fp);
char* fgets (char* s, int n, FILE* fp);
```

Arquivos em modo texto

int fscanf (FILE* fp, char* formato, ...);

- parâmetros:
  - ponteiro para o arquivo do qual os dados serão lidos
  - formato e lista de endereços de variáveis que armazenarão os valores lidos (como na função scanf)
- valor de retorno:
  - número de dados lidos com sucesso
- definição:
  - transfere dados para a memória
  - avança o cursor para o próximo dado

Arquivos em modo texto

int fgetc (FILE* fp);

- parâmetro:
  - ponteiro para o arquivo do qual os dados serão lidos
- valor de retorno:
  - código do caractere lido
  - constante EOF (*end of file*), se o fim do arquivo for alcançado
- definição:
  - captura o próximo caractere do arquivo
  - avança o cursor para o próximo caractere

Arquivos em modo texto

char* fgets (char* s, int n, FILE* fp);

- parâmetros:
  - cadeia de caracteres que armazenará o conteúdo lido do arquivo
  - número máximo de caracteres que deve ser lido
  - ponteiro para o arquivo do qual os dados serão lidos
- valor de retorno:
  - ponteiro da própria cadeia de caracteres passada como parâmetro
  - NULL, em caso de erro de leitura (ex: quando alcançar o final do arquivo)
- definição:
  - lê uma seqüência de caracteres, até que um caractere '\n' seja encontrado ou que o máximo de caracteres especificado seja alcançado
  - especificação do número máximo de caracteres evita invasão de memória quando a linha do arquivo for maior do que suposto

Arquivos em modo texto

- Funções para escrever dados em arquivos em modo texto:

```
int fprintf(FILE* fp, char* formato, ...);
int fputc (int c, FILE* fp);
```

Arquivos em modo texto

int fprintf(FILE* fp, char* formato, ...);

- parâmetro:
  - ponteiro para o arquivo no qual o dado será salvo
  - formato e lista de endereços de variáveis que fornecerão os dados a serem escritos no arquivo (como na função printf)
- valor de retorno:
  - representa o número de bytes escritos no arquivo
- definição:
  - (similar à função printf)

Arquivos em modo texto

int fputc (int c, FILE* fp);

- parâmetros:
  - código do caractere a ser escrito (salvo)
  - ponteiro para o arquivo no qual o caractere será escrito
- valor de retorno:
  - o próprio caractere escrito
  - EOF, se ocorrer um erro na escrita
- definição:
  - escreve um caractere no arquivo

Estruturação de dados em arquivos textos

- Formas para representar e acessar dados em arquivos:
  - caractere a caractere
  - linha a linha
  - usando palavras-chaves

Estruturação de dados em arquivos textos

- Exemplo de acesso caractere a caractere:
  - programa para contar o número de linhas de um arquivo (de nome “entrada.txt”)
    - leia, caractere a caractere, todo o conteúdo do arquivo, contando o número de ocorrências do caractere '\n‘ (que indica mudança de linha)

```c
/* Conta número de linhas de um arquivo */

#include <stdio.h>

int main (void)
{
   int c;
   int nlinhas = 0;     /* contador do número de linhas */
   FILE *fp;

   /* abre arquivo para leitura */
   fp = fopen("entrada.txt","rt");

   if (fp==NULL) {
      printf("Não foi possivel abrir arquivo.\n");
      return 1;
   }
```

```
/* lê caractere a caractere */
   while ((c = fgetc(fp)) != EOF)
   {
      if (c == '\n')
         nlinhas++;
   }

   /* fecha arquivo */
   fclose(fp);

   /* exibe resultado na tela */
   printf("Numero de linhas = %d\n", nlinhas);

   return 0;
}
```

Estruturação de dados em arquivos textos

- Exemplo de acesso caractere a caractere:
  - programa para:
    - ler o conteúdo do arquivo
    - criar um arquivo com o mesmo conteúdo,
      mas com todas as letras minúsculas convertidas para maiúsculas
  - nomes dos arquivos fornecidos, via teclado, pelo usuário

```c
/* Converte arquivo para maiúsculas */

#include <stdio.h>
#include <ctype.h>          /* função toupper : converte para maiúsculas */

int main (void)
{
  int c;
  char entrada[121];        /* armazena nome do arquivo de entrada      */
  char saida[121];          /* armazena nome do arquivo de saída        */
  FILE* e;                  /* ponteiro do arquivo de entrada           */
  FILE* s;                  /* ponteiro do arquivo de saída             */

  /* pede ao usuário os nomes dos arquivos */
  printf("Digite o nome do arquivo de entrada: ");
  scanf("%120s",entrada);
  printf("Digite o nome do arquivo de saida: ");
  scanf("%120s",saida);
```

```
/* abre arquivos para leitura e para escrita */
   e = fopen(entrada,"rt");
   if (e == NULL) {
      printf("Não foi possível abrir arquivo de entrada.\n");
      return 1;
   }
   s = fopen(saida,"wt");
   if (s == NULL) {
      printf("Não foi possível abrir arquivo de saida.\n");
      fclose(e);
      return 1;
   }
   /* lê da entrada e escreve na saída */
   while ((c = fgetc(e)) != EOF)
      fputc(toupper(c),s);
   /* fecha arquivos */
   fclose(e);
   fclose(s);
   return 0;
}
```

Estruturação de dados em arquivos textos

- Exemplo de acesso linha a linha:
  - programa para procurar a ocorrência de uma sub-cadeia de caracteres dentro de um arquivo:
    - se a sub-cadeia for encontrada,
      retorne o número da linha da primeira ocorrência
      - leia, linha a linha, o conteúdo do arquivo, contanto o número da linha
      - para cada linha, verifique a ocorrência da sub-cadeia,
        interrompendo a leitura em caso afirmativo
  - função strstr
    - procura a ocorrência de uma sub-cadeia
      numa cadeia de caracteres maior
    - retorna o endereço da primeira ocorrência ou
      NULL, se a sub-cadeia não for encontrada

```
/* Procura ocorrência de sub-cadeia em arquivo com linhas de 121 posições*/
#include <stdio.h>
#include <string.h>          /* função strstr */
int main (void)
{
  int n = 0;                 /* número da linha corrente                  */
  int achou = 0;             /* indica se achou sub-cadeia                */
  char entrada[121];         /* armazena nome do arquivo de entrada       */
  char subcadeia[121];       /* armazena sub-cadeia                       */
  char linha[121];           /* armazena cada linha do arquivo            */
  FILE* fp;                  /* ponteiro do arquivo de entrada            */
  /* pede ao usuário o nome do arquivo e a sub-cadeia                     */
  printf("Digite o nome do arquivo de entrada: ");
  scanf("%120s",entrada);
  printf("Digite a sub-cadeia: ");
  scanf("%120s",subcadeia);
  /* abre arquivo para leitura */
  fp = fopen(entrada,"rt");
  if (fp == NULL)
    { printf("Não foi possível abrir arquivo de entrada.\n");
      return 1; }
```

```
/* lê linha a linha */
   while (fgets(linha,121,fp) != NULL)
   {
      n++;
      if (strstr(linha,subcadeia) != NULL)
        { achou = 1; break; }
    }

   /* fecha arquivo */
   fclose(fp);

   /* exibe saída */
   if (achou)
       printf("Achou na linha %d.\n", n);
   else
       printf("Nao achou.");

   return 0;
}
```

Estruturação de dados em arquivos textos

- Exemplo de acesso linha a linha:
  - arquivo com uma lista de retângulos, triângulos e círculos
  - formato das linhas do arquivo:
    - linha com dados de figura:
      - caractere indicando o tipo de figura (r, t ou c)
      - parâmetros que definem a figura:

        retângulos e triângulos: base e altura

        círculos: raio
    - linha iniciada com o caractere #
      - representa comentário e deve ser desconsiderada
    - linha em branco
      - permitida, mas deve ser desprezada

```
# Lista de figuras

r 2.0 1.2
c 5.8
# t 1.23 12
t 4 1.02

c 5.1
```

Estruturação de dados em arquivos textos

int sscanf (char* s, char* formato, ...);

- parâmetros:
  - cadeia de caracteres representando a string
  - formato e lista de endereços de variáveis que fornecerão os dados a serem capturados da string (como na função scanf)
- valor de retorno:
  - número de dados lidos com sucesso
- definição:
  - similar às funções scanf e fscanf
  - captura os valores armazenados numa *string*

Estruturação de dados em arquivos textos

- Programa para interpretar o arquivo:
  - para cada linha lida do arquivo,
    - leia um caractere do conteúdo da linha (desprezando eventuais caracteres brancos iniciais) seguido de dois números reais
  - se nenhum dado for lido com sucesso,
    - há uma linha vazia, que deve ser desprezada
  - se pelo menos um caractere for lido com sucesso,
    - é possível interpretar o tipo da figura geométrica armazenada na linha ou detectar a ocorrência de um comentário
  - se for um retângulo ou um triângulo,
    - os dois valores reais também deverão ter sido lidos com sucesso
  - se for um círculo,
    - apenas um valor real deverá ter sido lido com sucesso

```
 char c;
float v1, v2;
FILE* fp;
char linha[121];
...
while (fgets(linha,121,fp))
{  int n = sscanf(linha," %c %f %f",&c,&v1,&v2);
   if (n>0)
   {  switch(c)
      {  case '#': /* desprezar linha de comentário */
         break;
         case 'r':  if (n!=3) { /* tratar erro de formato do arquivo */  ... }
                          else  { /* tratar retângulo: base = v1, altura = v2 */ ... }
         break;
         case 't':  if (n!=3) { /* tratar erro de formato do arquivo */ ...  }
                          else  { /* tratar triângulo: base = v1, altura = v2 */ ... }
         break;
         case 'c': if (n!=2) { /* tratar erro de formato do arquivo */ ... }
                          else { /* tratar círculo: raio = v1 */ ... }
         break;
         default:  /* tratar erro de formato do arquivo */   ...
         break;
      }
   }
}
...
```

Estruturação de dados em arquivos textos

- Acesso via palavras-chaves:
  - adotado quando os objetos num arquivo têm descrições de tamanhos variados
  - cada objeto é precedido por uma palavra-chave que o identifica
  - interpretação do arquivo pode ser feita lendo-se as palavras-chaves e interpretando a descrição do objeto correspondente

Estruturação de dados em arquivos textos

- Exemplo de acesso via palavras-chaves:
  - arquivo contendo uma lista de retângulos, triângulos, círculos e polígonos quaisquer
  - cada polígono é descrito pelo número de vértices que o compõe, seguido das coordenadas dos vértices

```
RETANGULO
 b   h

TRIANGULO
 b   h

CIRCULO
 r

POLIGONO
 n
 x1  y1
 x2  y2
 …
 xn  yn
```

```
 ...
FILE* fp;
char palavra[121];
...
while (fscanf(fp,"%120s",palavra) == 1)
{
   if (strcmp(palavra,"RETANGULO")==0) {
      /* interpreta retângulo */
   }
   else if (strcmp(palavra,"TRIANGULO")==0) {
      /* interpreta triângulo */
   }
   else if (strcmp(palavra,"CIRCULO")==0) {
      /* interpreta círculo */
   }
   else if (strcmp(palavra,"POLIGONO")==0) {
      /* interpreta polígono */
   }
   else {
      /* trata erro de formato */
   }
}
```

strcmp(s,t)==0		, se c=t
	<0	, se c<t
	>0	, se c>t
(a ordem é lexicográfica)		

Arquivos em modo binário

- Arquivo em modo binário:
  - utilizado para salvar (e depois recuperar) o conteúdo da memória principal diretamente no disco
    - cada byte da memória é copiado para o arquivo
- Funções:

```
int fwrite (void* p, int tam, int nelem, FILE* fp);

int fread (void* p, int tam, int nelem, FILE* fp);

int fseek (FILE* fp, long offset, int origem);
```

Arquivos em modo binário

int fwrite (void* p, int tam, int nelem, FILE* fp);

- parâmetros:

p	representa o endereço de memória a partir do qual bytes devem ser copiados para o arquivo
tam	indica o tamanho, em bytes, de cada elemento
nelem	indica o número de elementos
fp	ponteiro do arquivo binário para o qual o conteúdo da memória será copiado

- definição:
  - escreve (salva) dados em arquivos binários

Arquivos em modo binário

int fread (void* p, int tam, int nelem, FILE* fp);

- parâmetros:

p	representa o endereço de memória a partir do qual os bytes lidos para o arquivo serão colocados
tam	indica o tamanho, em bytes, de cada elemento
nelem	indica o número de elementos
fp	ponteiro do arquivo binário do qual dados serão lidos

- definição:
  - lê (recupera) dados de arquivos binários

Arquivos em modo binário

- Exemplo:
  - aplicação com conjunto de pontos armazenados num vetor
  - tipo definindo o ponto:

```
struct ponto {
   float x, y, z;
};
typedef struct ponto Ponto;
```

Arquivos em modo binário

- Exemplo (cont.):
  - função para salvar o conteúdo de um vetor de pontos:
    - recebe como parâmetros o nome do arquivo, o número de pontos no vetor e o ponteiro para o vetor

```
void salva (char* arquivo, int n, Ponto* vet)
{
  FILE* fp = fopen(arquivo,"wb");
  if (fp==NULL) {
    printf("Erro na abertura do arquivo.\n");
    exit(1);
  }
  fwrite(vet, sizeof(Ponto), n, fp);
  fclose(fp);
}
```

Arquivos em modo binário

- Exemplo (cont.):
  - função para recuperar os dados salvos

```
void carrega (char* arquivo, int n, Ponto* vet)
{
   FILE* fp = fopen(arquivo,"rb");
   if (fp==NULL) {
      printf("Erro na abertura do arquivo.\n");
      exit(1);
   }
   fread(vet, sizeof(Ponto), n, fp);
   fclose(fp);
}
```

Arquivos em modo binário

int fseek (FILE* fp, long offset, int origem);

- parâmetros:

  fp — arquivo cujo cursor será re-posicionando

  offset — de quantos bytes o cursor deve avançar

  origem — em relação a que posição o cursor deve ser avançado

    SEEK_CUR — em relação à posição corrente

    SEEK_SET — em relação ao início do arquivo

    SEEK_END — em relação ao final do arquivo

- definição:
  - altera a posição do cursor do arquivo

Arquivos em modo binário

- Exemplo (cont.):
  - função para capturar o i-ésimo ponto armazenado em um arquivo de pontos no espaço 3D salvo como definido anteriormente

```
Ponto le_ponto (FILE* fp, int i)
{
   Ponto p;
   fseek(fp, i*sizeof(Ponto), SEEK_SET);
   fread(&p, sizeof(Ponto), 1, fp);
   return p;
}
```

Resumo

- Funções para abrir e fechar arquivos:

```
FILE* fopen (char* nome_arquivo, char* modo);
int fclose (FILE* fp);
```

- Funções para arquivo em modo texto:

```
int fscanf (FILE* fp, char* formato, ...);
int fgetc (FILE* fp);
char* fgets (char* s, int n, FILE* fp);
int fprintf(FILE* fp, char* formato, ...);
int fputc (int c, FILE* fp);
```

- Funções para arquivos em modo binário:

```
int fwrite (void* p, int tam, int nelem, FILE* fp);
int fread (void* p, int tam, int nelem, FILE* fp);
int fseek (FILE* fp, long offset, int origem);
```