MC-102 – Aula 24
Arquivos Binários

Eduardo C. Xavier

Instituto de Computação – Unicamp

28 de Junho de 2017

Roteiro

Motivação

- Vimos que existem dois tipos de arquivos: textos e binários.
- Variáveis **int** ou **float** têm tamanho fixo na memória. Por exemplo, um **int** ocupa 4 bytes.
  - Representação em texto precisa de um número variável de dígitos (`10`, `5.673`, `100.340`), logo de um tamanho variável.
  - Lembre-se que cada letra/dígito é um **char** e usa 1 byte de memória.
- Armazenar dados em arquivos de forma análoga a utilizada em memória permite:
  - Reduzir o tamanho do arquivo.
  - Guardar estruturas complicadas tendo acesso simples.

Arquivos Binário em C

- Assim como em arquivos texto, para trabalharmos com arquivos binários devemos criar um ponteiro para arquivos.

```
FILE *nome_variavel;
```

- Podemos então associar o ponteiro com um arquivo real do computador usando o comando **fopen**.

```
FILE *arq1;
arq1 = fopen("teste.bin","rb");
```

Abrindo um Arquivo Binário: **fopen**

Um pouco mais sobre a função **fopen()** para arquivos binário.

```
FILE* fopen(const char *caminho, char *modo);
```

Modos de abertura de arquivo binário

modo	operações
rb	leitura
wb	escrita
r+b	leitura e escrita
w+b	escrita e leitura

Abrindo um Arquivo Binário: **fopen**

- Se um arquivo for aberto para leitura (**rb**) e não existir a função devolve **NULL**.
- Se um arquivo for aberto para escrita (**wb**) e não existir um novo arquivo é criado. Se ele existir, é sobrescrito.
- Se um arquivo for aberto para leitura/gravação (**r+b**) e existir ele NÃO é sobrescrito;
  Se o arquivo não existir a função devolve **NULL**.
- Se um arquivo for aberto para gravação/escrita (**w+b**) e existir ele é sobrescrito;
  Se o arquivo não existir um novo arquivo é criado.

Lendo e Escrevendo com **fread** e **fwrite**

- As funções **fread** e **fwrite** permitem a leitura e escrita de blocos de dados.
- Devemos determinar o número de elementos a serem lidos ou gravados e o tamanho de cada um.

Lendo e Escrevendo com **fread** e **fwrite**

Para escrever em um arquivo binário usamos a função **fwrite**:

```
size_t fwrite(void *pt_mem, size_t size,
              size_t num_items, FILE *pt_arq);
```

- **pt-mem:** Ponteiro para região da memória contendo os itens que devem ser escritos em arquivo.
- **size:** Número de bytes de um item.
- **num-items:** Número de itens que devem ser gravados.
- **pt-arq:** Ponteiro para o arquivo.

O retorno da função é o número de itens escritos corretamente.

Lendo e Escrevendo com **fread** e **fwrite**

Podemos por exemplo gravar um double em formato binário como abaixo:

```
#include <stdio.h>

int main(){
  double aux = 2.5;
  FILE *arq = fopen("teste.bin", "wb");

  if(arq == NULL){
    printf("Erro no arquivo.\n"); return 0;
  }

  fwrite(&aux, sizeof(double), 1, arq);

  fclose(arq);

}
```

Lendo e Escrevendo com **fread** e **fwrite**

Para ler de um arquivo binário usamos a função **fread**:

```
size_t fread(void *pt-mem, size_t size,
             size_t num-items, FILE *pt-arq);
```

- **pt-mem:** Ponteiro para região da memória (já alocada) para onde os dados serão lidos.
- **size:** Número de bytes de um item a ser lido.
- **num-items:** Número de itens que devem ser lidos.
- **pt-arq:** Ponteiro para o arquivo.

O retorno da função é o número de itens lidos corretamente.

Lendo e Escrevendo com **fread** e **fwrite**

Usando o exemplo anterior podemos ler um double em formato binário como segue:

```
#include <stdio.h>

int main(){
  double aux;
  FILE *arq = fopen("teste.bin", "rb");

  if(arq == NULL){
    printf("Erro no arquivo.\n"); return 0;
  }

  fread(&aux, sizeof(double), 1, arq);

  printf("Lido: %lf\n", aux);

  fclose(arq);

}
```

Lendo e Escrevendo com **fread** e **fwrite**

Podemos por exemplo gravar um vetor de doubles em formato binário:

```
#include <stdio.h>

int main(){
  double aux[]={2.5, 1.4, 3.6};

  FILE *arq = fopen("teste.bin", "w+b");
  if(arq == NULL){
    printf("Erro no arquivo.\n"); return 0;
  }

  fwrite(aux, sizeof(double), 3, arq);

  fclose(arq);

}
```

Lendo e Escrevendo com **fread** e **fwrite**

Usando o exemplo visto, podemos ler um vetor de doubles em formato binário como segue:

```
#include <stdio.h>

int main(){
  double aux[3];
  int i;
  FILE *arq = fopen("teste.bin", "r+b");
  if(arq == NULL){
    printf("Erro no arquivo.\n"); return 0;
  }


  fread(aux, sizeof(double), 3, arq);
  for(i=0; i<3; i++)
    printf("%lf, ", aux[i]);
  printf("\n");

  fclose(arq);

}
```

Lendo e Escrevendo com **fread** e **fwrite**

- Lembre-se do **indicador de posição** de um arquivo, que assim que é aberto é apontado para o início do arquivo.
- Quando lemos uma determinada quantidade de itens, o indicador de posição automaticamente avança para o próximo item não lido.
- Quando escrevemos algum item, o indicador de posição automaticamente avança para a posição seguinte ao item escrito.

Lendo e Escrevendo com **fread** e **fwrite**

- Se na leitura não sabemos exatamente quantos itens estão gravados, podemos usar o que é devolvido pela função **fread**:
  - Esta função devolve o número de itens corretamente lidos.
  - Se alcançarmos o final do arquivo e tentarmos ler algo, ela devolve 0.

No exemplo do vetor poderíamos ter lido os dados como segue:

```
double aux2;
while( fread(&aux2, sizeof(double), 1, arq) != 0){
  printf("%.2lf, ", aux2);
}
```

Lendo dados do arquivo:

```c
#include <stdio.h>

int main(void){
   double aux2;
   FILE *arq = fopen("teste.bin", "r+b");
   if(arq == NULL){
      printf("Erro"); return 1;
   }
   while( fread(&aux2, sizeof(double), 1, arq) != 0){
      printf("%.2lf, ", aux2);
   }
   printf("\n");
   fclose(arq);
}
```

Acesso Não Seqüencial: **fseek**

- Fazemos o acesso não seqüencial usando a função **fseek**.
- Esta função altera a posição de leitura/escrita no arquivo.
- O deslocamento pode ser relativo ao:
  - início do arquivo (SEEK_SET)
  - ponto atual (SEEK_CUR)
  - final do arquivo (SEEK_END)

Acesso Não Seqüencial: **fseek**

```
int fseek(FILE *pt-arq, long num-bytes, int origem);
```

- **pt-arq:** ponteiro para arquivo.
- **num-bytes:** quantidade de bytes para se deslocar.
- **origem:** posição de início do deslocamento (SEEK_SET, SEEK_CUR, SEEK_END).

Por exemplo, se quisermos alterar o segundo **double** de um vetor escrito fazemos:

```
double aux[]={2.5, 1.4, 3.6};
double aux3=5.0;

arq = fopen("teste.bin", "w+b");
fwrite(aux, sizeof(double), 3, arq);

fseek(arq, 1*sizeof(double), SEEK_SET); //a partir do inicio pula um double
fwrite(&aux3, sizeof(double), 1, arq);
```

Programa que escreve vetor de 3 números do tipo **double**:

```
#include <stdio.h>

int main(){
  double aux[]={2.5, 1.4, 3.6};

  FILE *arq = fopen("teste.bin", "w+b");
  if(arq == NULL){
    printf("Erro no arquivo.\n"); return 0;
  }

  fwrite(aux, sizeof(double), 3, arq);

  fclose(arq);

}
```

Programa que altera o arquivo:

```c
#include <stdio.h>

int main(void){
   double aux=104.98;
   FILE *arq = fopen("teste.bin", "r+b");
   if(arq == NULL){
      printf("Erro"); return 1;
   }

   //seta o indicador de posição do arquivo para o início do
   //segundo número
   fseek(arq, 1*sizeof(double), SEEK_SET);

   fwrite(&aux, sizeof(double), 1, arq);

   fclose(arq);

}
```

Exemplo: Arquivo de Registros

- Um arquivo pode armazenar registros (como um banco de dados).
- Isso pode ser feito de forma bem fácil se lembrarmos que um registro, como qualquer variável em C, tem um tamanho fixo.
- O acesso a cada registro pode ser direto, usando a função **fseek**.
- A leitura ou escrita do registro pode ser feita usando as funções **fread** e **fwrite**.

Exemplo: Arquivo de Registros

Vamos fazer uma aplicação para um cadastro de alunos:

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

//nome do arquivo que contém o cadastro
#define FILE_NAME  "alunos.bin"

struct Aluno{
        char nome[100];
        int RA;
};
typedef struct Aluno  Aluno;

//Esta função imprime todo o conteúdo do cadastro em arquivo
void imprimeArquivo();

//Dado um ra passado por parâmetro, a função altera o nome da pessoa com este ra
void alteraNome(int ra, char nome[]);
```

Exemplo: Função que imprime arquivo

```
void imprimeArquivo(){
  Aluno cadastro;
  FILE *arq = fopen(FILE_NAME, "r+b"); //Note que usamos r e não w
  if(arq == NULL){
    printf("Erro Arquivo (imprime).\n");
    return;
  }

  printf(" ----- Imprimindo Dados -----\n");
  while( fread(&cadastro, sizeof(Aluno), 1, arq) != 0){
    printf("Nome: %s,    RA: %d \n", cadastro.nome, cadastro.RA);
  }
  printf("\n");
  fclose(arq);
}
```

Exemplo: Função que Altera um Registro

```
void alteraNome(int ra, char nome[]){
  Aluno aluno;
  FILE *arq = fopen(FILE_NAME, "r+b");
  if(arq == NULL){
    printf("Erro Arquivo (altera).\n");
    return;
  }

  while(fread(&aluno, sizeof(Aluno), 1, arq) != 0){
    if(aluno.RA == ra){ //Encontramos o Aluno
      strcpy(aluno.nome, nome); //Altera nome
      fseek(arq, -1*sizeof(Aluno), SEEK_CUR);//Volta um item
      fwrite(&aluno, sizeof(Aluno), 1, arq);//Sobreescreve Reg. antigo
      break;
    }
  }

  fclose(arq);
}
```

Exemplo: Função Principal

```
int main(){
  Aluno cadastro[] = { {"Joao", 1}, {"Batata", 2}, {"Ze", 3}, {"Malu", 4}, {"Ju", 5} };
  FILE *arq = fopen(FILE_NAME, "w+b");
  if(arq == NULL){
    printf("Erro Arquivo (main).\n");
    return 0;
  }
  fwrite(cadastro, sizeof(Aluno), 5, arq);
  fclose(arq);

  //Após criado o arquivo aqui em cima, vamos alterá-lo
  //chamando a função alteraNome
  imprimeArquivo();
  alteraNome(4,"Malu Mader");
  imprimeArquivo();
}
```

Exercício

- Escreva um programa que leia dois arquivos de inteiros ordenados e escreva um arquivo cuja saída é um único arquivo ordenado.
  - Vale a pena colocar o conteúdo dos arquivos de entrada em dois vetores?
  - Escreva duas versões deste programa, uma para arquivos texto e outra para arquivos binários.