MC-102 – Aula 28
Arquivos Binários e Parâmetros do Programa

Alexandre M. Ferreira

IC – Unicamp

14/06/2017

Roteiro

1. Arquivos Binários
   - Abrindo um Arquivo Binário: **fopen**
   - Lendo e Escrevendo com **fread** e **fwrite**
   - Acesso Não Seqüencial: **fseek**
2. Exemplo: Arquivo de Registros
3. Parâmetros do programa: argc e argv

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.

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

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

```
int main(void){
  FILE *arq;
  double aux=2.5;

  arq = fopen("teste.bin", "w+b");
  if(arq == NULL){
     printf("Erro"); return 1;
  }
  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.

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

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

```
int main(void){
  FILE *arq;
  double aux=0;

  arq = fopen("teste.bin", "r+b");
  if(arq == NULL){
     printf("Erro"); return 1;
  }
  fread(&aux, sizeof(double), 1, arq);
  printf("Numero lido: %.2lf\n", aux);

  fclose(arq);

}
```

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

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

```
int main(void){
  FILE *arq;
  double aux[]={1.4, 2.4, 3.6};

  arq = fopen("teste.bin", "w+b");
  if(arq == NULL){
     printf("Erro"); return 1;
  }
  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:

```
int main(void){
   FILE *arq;
   double aux[3];

   arq = fopen("teste.bin", "r+b");
   if(arq == NULL){
       printf("Erro"); return 1;
   }
   fread(aux, sizeof(double), 3, arq);

   int i;
   for(i=0; i<3; i++){
      printf("Numero lido: %.2lf\n", aux[i]);
   }
   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:

```
i=0;
while(fread(&aux2[i], sizeof(double), 1, arq) != 0){
      i++;
}
```

Lendo dados do arquivo:

```
int main(void){
  FILE *arq;
  double aux[3], aux2;

  arq = fopen("teste.bin", "r+b");
  if(arq == NULL){
     printf("Erro"); return 1;
  }

  int i=0;
  while( fread(&aux2, sizeof(double), 1, arq) != 0){
    aux[i] = aux2;
    i++;
  }

  int j;
  for(j=0; j<i; j++){
    printf("Numero lido: %.2lf\n", aux[j]);
  }
  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 terceiro **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, 2*sizeof(double), SEEK_SET); //a partir do inicio pula dois doubles
fwrite(&aux3, sizeof(double), 1, arq);
```

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

```
int main(void){
  FILE *arq;
  double aux[]={1.4, 2.4, 3.6};

  arq = fopen("teste.bin", "w+b");
  if(arq == NULL){
     printf("Erro"); return 1;
  }
  fwrite(aux, sizeof(double), 3, arq);

  fclose(arq);
}
```

Programa que altera o arquivo:

```
int main(void){
  FILE *arq;
  double aux2=104.98;

  arq = fopen("teste.bin", "r+b");
  if(arq == NULL){
     printf("Erro"); return 1;
  }

  //seta o indicador de posição do arquivo para o início do
  //terceiro número
  fseek(arq, 2*sizeof(double), SEEK_SET);

  fwrite(&aux2, sizeof(double), 1, arq);

  fclose(arq);

}
```

Programa que mostra conteúdo do arquivo:

```
int main(void){
  FILE *arq;
  double aux[3], aux2;

  arq = fopen("teste.bin", "r+b");
  if(arq == NULL){
     printf("Erro"); return 1;
  }

  int i=0;
  while( fread(&aux2, sizeof(double), 1, arq) != 0){
    aux[i] = aux2;
    i++;
  }

  int j;
  for(j=0; j<i; j++){
    printf("Numero lido: %.2lf\n", aux[j]);
  }
  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>

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


void imprimeArquivo(char nomeArq[]); //Função que imprime arquivo com o cadastro
void alteraNome(char nomeArq[], int ra, char nome[]); // Função que atualiza um
                                                      // cadastro
```

Exemplo: Função Principal

```
int main(){
  char nomeArq[] = "alunos.bin";
  Aluno alunos[] = { {"Batata", 1}, {"Isa", 2}, {"Malu", 3}, {"Beto", 4} };

  FILE *arq = fopen(nomeArq, "w+b");
  if(arq == NULL){
    printf("Erro ao criar arquivo\n"); return 1;
  }
  fwrite(alunos, sizeof(Aluno), 4, arq);
  fclose(arq);
  //Após criado o arquivo com o cadastro
  //vamos altera-lo conforme abaixo

  imprimeArquivo(nomeArq);
  alteraNome(nomeArq, 2, "Isabela");
  imprimeArquivo(nomeArq);

}
```

Exemplo: Função que imprime arquivo

```
void imprimeArquivo(char nomeArq[]){
  FILE *arq = fopen(nomeArq, "r+b");

  if(arq == NULL){
    printf("Arquivo não existe!\n");
    return;
  }

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

  fclose(arq);
}
```

Exemplo: Função que Altera um Registro

```
void alteraNome(char nomeArq[], int ra, char nome[]){
  FILE *arq = fopen(nomeArq, "r+b");

  if(arq == NULL){
    printf("Arquivo não existe!\n");     return;
  }

  Aluno aux; int achou=0;
  while( fread(&aux, sizeof(Aluno), 1, arq) != 0){
    if(aux.RA == ra){
      achou = 1;
      break;
    }
  }

  //Volta uma posição para sobreescrever registro
  if(achou){
    fseek(arq, -1 * sizeof(Aluno), SEEK_CUR);
    strcpy(aux.nome, nome); //atualiza o nome
    fwrite(&aux, sizeof(Aluno), 1, arq);
  }
  fclose(arq);
}
```

Argc e Argv

- Até então temos criado programas onde a função `main()` não tem parâmetros.
- Mas esta função pode receber dois parâmetros: `main(int argc, char *argv[])`.
  - `argc` (*argument counter*): indica o número de argumentos na linha de comando ao se executar o programa.
  - `*argv[]` (*argument vector*): é um vetor de ponteiros para caracteres (ou seja vetor de strings) que contém os argumentos da linha de comando, um em cada posição do vetor.

Argc e Argv

O programa abaixo imprime cada um dos parâmetros passados na linha de comando:

```
#include <stdio.h>

int main(int argc, char *argv[]){
   int i;

   for(i=0; i<argc; i++){
      printf("%s\n", argv[i]);
   }

}
```

Argc e Argv

- Seu uso é útil em programas onde dados de entrada são passados via linha de comando.
- Exemplo: dados a serem processados estão em um arquivo, cujo nome é passado na linha de comando.

```
//Este programa mostra o conteúdo de um arquivo texto cujo nome é passado como
//parâmetro do programa
int main(int argc, char *argv[]){
  int i;
  FILE *fp=NULL;

  fp = fopen(argv[1], "r+");
  if(fp == NULL){
    printf("Arquivo não existe!\n");
    return 1;
  }

  char aux;
  while(fscanf(fp, "%c", &aux) != EOF){
    printf("%c", aux);
  }

}
```