

# MC-102 – Aula 26

## Arquivos Binários

Instituto de Computação – Unicamp

4 de Junho de 2014

# Motivação

- Vimos que existem dois tipos de arquivos: texto e binário.
- 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 Binários em C

- Assim como em arquivos texto, devemos criar um ponteiro especial: **um ponteiro para arquivos.**

```
FILE *nome_variavel;
```

- Podemos então associa-lo com um arquivo real do computador usando o comando **fopen.**

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

# fopen

Um pouco mais sobre a função `fopen()` para arquivos binários.

```
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

# fopen

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

## 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.

## 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 gravados.
- **size:** Número de bytes de um item.
- **num-items:** Número de itens que devem ser gravados.
- **pt-arq:** Ponteiro para o arquivo.

## fread e fwrite

Podemos por exemplo gravar um double em formato binário como no exemplo:

```
FILE *arq;  
double aux=2.5;  
  
arq = fopen("teste.bin", "w+b");  
fwrite(&aux, sizeof(double), 1, arq);
```

## fread e fwrite

Podemos por exemplo gravar um vetor de doubles em formato binário no exemplo:

```
FILE *arq;  
double aux[]={2.5, 1.4, 3.6};  
  
arq = fopen("teste.bin", "w+b");  
fwrite(aux, sizeof(double), 3, arq);
```

## 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 deve ser lido.
- **pt-arq:** Ponteiro para o arquivo.

## fread e fwrite

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

```
#include <stdio.h>

int main(){
    FILE *arq;
    double aux=2.5;
    double aux2=0;

    arq = fopen("teste.bin", "w+b");
    fwrite(&aux, sizeof(double), 1, arq);

    rewind(arq);
    fread(&aux2, sizeof(double), 1, arq);
    printf("Conteudo de aux2: %lf, \n", aux2);

    fclose(arq);
}
```

## fread e fwrite

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

```
#include <stdio.h>

int main(){
    FILE *arq;
    double aux[]={2.5, 1.4, 3.6};
    double aux2[3];
    int i;

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

    rewind(arq);

    fread(aux2, sizeof(double), 3, arq);
    for(i=0; i<3; i++)
        printf("Conteudo de aux2[%d]: %lf\n", i, aux2[i]);

    fclose(arq);
}
```

## 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.

## 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:

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

ou de forma equivalente:

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

```
#include <stdio.h>

int main(){
    FILE *arq;
    double aux[]={2.5, 1.4, 3.6};
    double aux2[3];
    int i;

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

    rewind(arq);

    for(i=0; fread(&aux2[i], sizeof(double), 1, arq) != 0; i++)
        ;

    for(i=0; i<3; i++)
        printf("Conteudo de aux2[%d]: %lf\n", i, aux2[i]);

    fclose(arq);
}
```

# Acesso não seqüencial

- 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

```
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:

```
double aux[]={2.5, 1.4, 3.6};  
double aux3=5.0;  
  
arq = fopen("teste.bin", "w+b");  
fwrite(aux, sizeof(double), 3, arq);  
  
rewind(arq);  
fseek(arq, 2*sizeof(double), SEEK_SET);  
fwrite(&aux3, sizeof(double), 1, arq);
```

```
#include <stdio.h>

int main(){
    FILE *arq;
    double aux[]={2.5, 1.4, 3.6};
    double aux2[3];
    double aux3=5.0;
    int i;

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

    rewind(arq);
    fseek(arq, 2*sizeof(double), SEEK_SET);
    fwrite(&aux3, sizeof(double), 1, arq);

    fseek(arq, 0, SEEK_SET); //isto é equivalente a rewind(arq). Por que?
    fread(aux2, sizeof(double), 3, arq);

    for(i=0; i<3; i++)
        printf("Conteudo de aux2[%d]: %lf\n", i, aux2[i]);

    fclose(arq);
}
```

# 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 com Registros

Vamos fazer uma aplicação para um cadastro de alunos:

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

#define TAM 5 //tamanho do vetor usado como cadastro

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

typedef struct Aluno Aluno;

void imprimeArquivo(); //Esta função imprime todo o conteúdo
                        // do cadastro em arquivo
void alteraNome(int ra, char nome[]); //Dado um ra passado por
                                        //parâmetro, a função altera o nome da pessoa com este ra

char nomeArq[] = "alunos.bin"; //Nome do arquivo que contém o cadastro
```

# Exemplo: Função Principal

```
int main(){
    FILE *arq;
    Aluno cadastro[TAM] = {
        {"Joao", 1}, {"Batata", 2}, {"Ze", 3}, {"Malu", 4}, {"Ju", 5} };

    arq = fopen(nomeArq, "w+b");
    if(arq == NULL){
        printf("Erro: Main!\n");
        return 0;
    }
    fwrite(cadastro, sizeof(Aluno), TAM, arq);
    fclose(arq);
    //Após criado o arquivo aqui em cima, vamos alterá-lo
    //chamando a função alteraNome

    imprimeArquivo();
    alteraNome(4,"Malu Mader");
    imprimeArquivo();
}
```

## Exemplo: Função que imprime arquivo

```
void imprimeArquivo(){
    Aluno cadastro[TAM];
    FILE *arq = fopen(nomeArq, "r+b"); //Note que usamos r e não w
    int i;

    if(arq == NULL){
        printf("Erro: Imprime Arquivo!\n");
        return;
    }

    fread(cadastro, sizeof(Aluno), TAM, arq);
    printf(" ---- Imprimindo Dados ----\n");
    for(i=0; i<TAM; i++){
        printf("Nome: %s,   RA: %d \n", cadastro[i].nome, cadastro[i].RA);
    }
    printf("\n");
    fclose(arq);
}
```

## Exemplo: Função que Altera um Registro

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

    if(arq == NULL){
        printf("Erro: Altera nome!\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 da posição corrente
            fwrite(&aluno, sizeof(Aluno), 1, arq); //Sobreescreve Reg. antigo
            break;
        }
    }

    fclose(arq);
}
```

# Exercício

- Considere o programa de cadastro de alunos.
  - ▶ Altere-o para que mostre um menu de opções: 1. Criar cadastro; 2. Ler cadastro do arquivo e imprimir na tela; 3. Alterar cadastro, onde:
    - ★ Na opção 1, ao criar cadastro, seu programa deve ler alunos até que o RA digitado seja -1 (isto é, deve funcionar com qualquer quantidade de alunos).
    - ★ Na opção 2, seu programa deve ler o arquivo de cadastro e imprimir as informações na tela.
    - ★ Na opção 3, seu programa deve ler um RA cujo cadastro será alterado, e o novo nome a ser escrito no cadastro do RA lido. Por fim, o cadastro no arquivo deve ser atualizado com o novo nome.
  - ▶ Crie uma função que ordena o cadastro (primeiro lê o cadastro, depois ordena usando uma função de ordenação, e depois escreve o cadastro novamente no arquivo).