BCC 201 - Introdução à Programação I

Arquivos Binários

Guillermo Cámara-Chávez
UFOP

Arquivos Binários: typedef, structs em Arquivos I

```
typedef struct Dados
{
    int dia, mes, ano;
    double temp_min, temp_max;
}meusDados;

meusDados medidas[100];

FILE *Arquivo;
```

Arquivos Binários: typedef, structs em Arquivos II

- Arquivos de texto:
  - Precisa salvar cada dado da estrutura

```
    fprintf (Arquivo, "%d\n", Medidas[i].dia);
    fprintf (Arquivo, "%d\n", Medidas[i].mes);
    fprintf (Arquivo, "%d\n", Medidas[i].ano);
    fprintf (Arquivo, "%lf\n", Medidas[i].temp_min);
    fprintf (Arquivo, "%lf\n", Medidas[i].temp_max);
```

Arquivos Binários: typedef, structs em Arquivos III

- Arquivos binários:

  `fprintf "%???"` => Que tipos de dados devo usar?
  => fwrite

  - Pode escrever a estrutura inteira de uma só vez:

```
fwrite(medidas, sizeof(meusDados),
       100, Arquivo);
```

  - Pode escrever a estrutura com um registro de dados por vez:

```
fwrite(&medidas[indice], sizeof(meusDados),
       1, Arquivo);
```

Arquivos Binários: typedef, structs em Arquivos IV

```
FILE *Arquivo;
typedef struct Ponto{
    int x,y;
}myPonto;
myPonto vet[10];
```

- Arquivos Binários:
  - Pode escrever a estrutura inteira de uma só vez

```
fwrite(vet, sizeof(myPonto), 10, Arquivo);
```

- Arquivos de Texto:
  - Precisa salvar cada dado da estrutura

```
fprintf(Arquivo, "%d", vet[i].x);
fprintf(Arquivo, "%d", vet[i].y);
```

Modo Texto I

- É interpretado como uma seqüência de caracteres agrupadas em linhas
- Linhas são separadas por um caracter de nova linha
- Vantagens:
  - Pode ser lido facilmente por uma pessoa
  - Editado por editores de texto convencionais
- Desvantagens
  - Codificação dos caracteres pode variar (ASCII, UTF-8, ISSO-8859, etc)
  - Arquivos tendem a ser maiores (todas os dados são convertidos para caracteres)

Modo Binário I

- Dados são armazenados da mesma forma que são armazenados na memória principal
- Vantagens:
  - Facilmente interpretados por programas
  - Maior velocidade de manipulação
  - Arquivos são, geralmente, mais compactos
- Desvantagens:
  - Difícil de serem entendidos por pessoas
  - Dependentes da máquina onde foram gerados

Leitura (Modo Binário) I

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

- *p* é o endereço de memória em que vai ser armazenado o que for lido
- *tam* é o tamanho em bytes de cada elemento lido
- *nelem* é o número de elementos de tamanho tam lidos
- Retorna a quantidade de bytes lidos com sucesso (tam * nelem)

Escrita (Modo Binário) I

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

- *p* é o endereço de memória em que vai ser armazenado o que for lido
- *tam* é o tamanho em bytes de cada elemento lido
- *nelem* é o número de elementos de tamanho tam lidos
- Retorna a quantidade de bytes lidos com sucesso (tam * nelem)

Verificando o Final do Arquivo I

- Em operações de leitura do arquivo, é comum verificarmos se o final do arquivo já foi atingido.
- Função de verificação de fim de arquivo

```
int feof(FILE *fp);
```

  - Retorna 1 se o fim do arquivo é atingido
  - Retorna 0 caso contrario

Usando fwrite na Escrita I

Criar um programa que salva n pontos (composto de coordenas x, y) em um arquivo binário

Usando fwrite na Escrita II

```c
#include <stdio.h>
typedef struct ponto {
    float x,y;
} Ponto;

int main () {
    int i,n;
    Ponto p;
    FILE* fp = fopen("arquivoStruct.txt", "wb");
    if (fp == NULL) {
        printf("Erro na abertura do arquivo.\n");
        exit(1);
    }
    ...
}
```

Usando fwrite na Escrita III

```
int main () {
    . . .
    printf("Digite numero de pontos a gravar\n");
    scanf("%d",&n);
    for (i = 0; i < n; i++) {
        scanf("%f %f",&p.x,&p.y);
        fwrite(&p, sizeof(Ponto), 1, fp);
    }
    fclose(fp);
    return 0;
}
```

Usando fread na Leitura I

Criar um programa que lê todos os n pontos em um arquivo binário

Usando fread na Leitura II

```c
#include <stdio.h>
typedef struct ponto {
    float x,y;
} Ponto;

int main () {
    int i,n, numBytes;
    Ponto p;
    FILE* fp = fopen(arquivo, "rb");
    if (fp == NULL) {
        printf("Erro na abertura do arquivo.\n");
        exit(1);
    }
    while(!feof(fp)){
        numBytes = fread(&p, sizeof(Ponto), 1, fp);
        if (numBytes == 0) break;
        printf("Ponto lido: (%d,%d)",p.x,p.y);
    }
    fclose(fp);
}
```

Leitura/Escrita de Blocos de Dados I

- As funções fread/fwrite permitem ler/escrever grandes blocos de dados em um arquivo
  - Um dos parâmetros indica qual é a quantidade de dados de um determinado tipo a ser lido/escrito
- Portanto podem ser úteis para ler/escrever estruturas ou vetores em um arquivo numa única chamada de função

Usando fwrite na Escrita I

```c
#include <stdio.h>
typedef struct ponto {
    float x,y;
} Ponto;

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);
}
```

Usando fread na Leitura I

```c
#include <stdio.h>
typedef struct ponto {
    float x,y;
} Ponto;

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);
}
```

Usando as Funções Definidas Anteriormente I

```
int main() {
    Ponto *entrada, *saida; int nPontos, cont,pos ;
    FILE *arquivo;
    char nome_arquivo[121];
    printf("Digite o nome do arquivo:\n");
    scanf("%120s",nome_arquivo);
    printf("\nDigite o número de pontos:\n");
    scanf("%d",&nPontos);
    entrada = (Ponto *) malloc(nPontos*sizeof(Ponto));
    for (cont = 0; cont < nPontos;cont++) {
        printf("Digite coordenadas x,y:\n");
        scanf("%f%f",&entrada[cont].x,&entrada[cont].y);
    }
    salva(nome_arquivo, nPontos, entrada);
    . . .
}
```

Usando as Funções Definidas Anteriormente II

```
int main() {
    . . .
    do {
        printf("Digite agora a posição do ponto que
                deseja ver: \n");
        scanf("%d",&pos);
    } while (pos > nPontos || pos<=0 );
    saida = (Ponto *) malloc (nPontos*sizeof(Ponto));
    carrega(nome_arquivo, nPontos, saida);
    printf("O ponto na posicao %d é {%f,%f}\n", pos,
            saida[pos-1].x, saida[pos-1].y);
    return 0;
}
```

Acesso não sequencial I

- Fazemos o acesso não sequencial 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_SET`)

Acesso não sequencial II

```
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`).

Acesso não sequencial III

Alterar o terceiro elemento de um vetor escrito em um arquivo binário

Acesso não sequencial IV

```
int main(){
    float x[] = {5.6, 6, 9.8, 4.2};
    float nx[4], tmp = 7.2;
    int i;  FILE* pfile;
    pfile = fopen("test.txt", "w+b");
    fwrite(x, sizeof(float), 4, pfile);
    rewind(pfile); // retorna ao inicio
    fseek(pfile, 2*sizeof(float), SEEK_SET);
    fwrite(&tmp, sizeof(float), 1, pfile);
    fseek(pfile, 0, SEEK_SET); //equivalente a rewind
    fread(nx, sizeof(float), 4, pfile);
    for (i = 0; i < 4; i++)
        printf("%f ", nx[i]);
    fclose(pfile);
    return 0;
}
```

FIM