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];

ofstream Arquivo;
```

Arquivos Binários: typedef, structs em Arquivos II

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

```
Arquivo << Medidas[i].dia << " ";
Arquivo << Medidas[i].mes << " ";
Arquivo << Medidas[i].ano << " ";
Arquivo << Medidas[i].temp_min << " ";
Arquivo << Medidas[i].temp_max << endl;
```

Arquivos Binários: typedef, structs em Arquivos III

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

```
arquivo.write( (char*)medidas,
               sizeof(meusDados)*10);
```

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

```
arquivo.write( (char*)&medidas[indice],
               sizeof(meusDados));
```

Arquivos Binários: typedef, structs em Arquivos IV

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

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

```
Arquivo.write( (char*)vet, sizeof(myPonto)*10 );
```

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

```
Arquivo << vet[i].x << " ";
Arquivo << 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

```
istream& read(const char*, int);
```

- primeiro parâmetro: é o endereço de memória em que vai ser armazenado o que for lido
- segundo parâmetro: o tamanho em bytes do dado a ser lido

Escrita (Modo Binário) I

```
ostream& write(const char*, int);
```

- primeiro parâmetro: é o endereço de memória onde se encontram os dados
- segundo parâmetro: o tamanho em bytes do dado a ser escrito

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

```
bool eof();
```

  - Retorna true se o fim do arquivo é atingido, ou seja, se o flag de estado de erro `eofbit` está ligado
  - Retorna false caso contrario

Usando write na Escrita I

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

Usando write na Escrita II

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

int main () {
    int i,n;
    Ponto p;
    ofstream fp;
    fp.open("arquivoStruct.txt", ios::binary);
    if ( fp.fail() ) {
        cout << "Erro na abertura do arquivo.\n";
        exit(1);
    }
    . . .
}
```

Usando write na Escrita III

```
int main () {
    . . .
    cout << "Digite numero de pontos a gravar\n";
    cin >> n;
    for (i = 0; i < n; i++) {
        cin >> p.x >> p.y;
        fp.write((char*)&p, sizeof(Ponto));
    }
    fp.close();
    return 0;
}
```

Usando read na Leitura I

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

Usando read na Leitura II

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

int main () {
    ifstream nfp;
    Ponto P;
    nfp.open("arquivoStruct.txt", ios::binary);
    if (nfp.fail()) {
        cout << "Erro na abertura do arquivo.\n";
        exit(1);
    }
    while ( nfp.read( (char*)&P1, sizeof(Ponto) ) )
    {
        cout << P1.x << " " << P1.y << endl;
    }
    nfp.close();
    return 0;
}
```

Leitura/Escrita de Blocos de Dados I

- As funções read/write 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 write na Escrita I

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

void salva (string arquivo, int n, Ponto* vet) {
    ofstream fp;
    fp.open(arquivo, ios::binary);
    if ( fp.fail() ) {
        cout << "Erro na abertura do arquivo.\n";
        exit(1);
    }
    fp.write((char*)vet, sizeof(Ponto) * n);
    fp.close();
}
```

Usando fread na Leitura I

```
void carrega (string arquivo, int n, Ponto* vet) {
    ifstream fp;
    fp.open(arquivo, ios::binary);
    if ( fp.fail() ) {
        cout << Erro na abertura do arquivo.\n";
        exit(1);
    }
    fp.read((char*)vet, sizeof(Ponto) * n);
    fp.close();
}
```

Usando as Funções Definidas Anteriormente I

```
int main() {
    Ponto *entrada, *saida; int nPontos, cont,pos ;
    string nome_arquivo;
    cout << "Digite o nome do arquivo:\n";
    cin >> nome_arquivo;
    cout << "\n Digite o número de pontos:\n");
    cin >> nPontos;
    entrada = new Ponto[nPontos];
    for (cont = 0; cont < nPontos; cont++) {
        cout << "Digite coordenadas x,y:\n";
        cin >> entrada[cont].x >> entrada[cont].y;
    }
    salva(nome_arquivo, nPontos, entrada);
    . . .
}
```

Usando as Funções Definidas Anteriormente II

```
int main() {
    . . .
    do {
        cout << "Digite agora a posição do ponto que
                deseja ver: \n");
        cin >> pos;
    } while (pos > nPontos || pos<=0 );
    saida = new Ponto[nPontos];
    carrega(nome_arquivo, nPontos, saida);
    cout << "O ponto na posicao " << pos << " eh "
        <<saida[pos-1].x <<" " << saida[pos-1].y);
    return 0;
}
```

Acesso não sequencial I

- Fazemos o acesso não sequencial usando a função `seekg`.
- Esta função altera a posição de leitura/escrita no arquivo.
- O deslocamento pode ser relativo ao:
  - início do arquivo (`ios::beg`)
  - ponto atual (`ios::cur`)
  - final do arquivo (`ios::end`)

Acesso não sequencial II

```
istream& seekg(streampos off, ios_base::seekdir way)
```

- **off**: quantidade de bytes para se deslocar.
- **way**: posição de início do deslocamento (`ios::beg`, `ios::cur`, `ios::end`).

Acesso não sequencial III

Para obter a posição, em bytes, onde se encontra localizado o ponteiro que realiza a leitura do arquivo

```
streampos tellg()
```

Acesso não sequencial IV

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

Acesso não sequencial V

```
int main(){
    ifstream nfp;  Ponto *v;
    nfp.open("arquivoStruct.txt", ios::binary);
    // conta a quantidade de bytes que contem o arquivo
    nfp.seekg(0, ios::end);
    streampos tam = nfp.tellg() / sizeof(Ponto);
    nfp.seekg(0, ios::beg);

    v = new Ponto[tam];

    nfp.read((char*)v, sizeof(Ponto)*tam);

    for (int i = 0; i < tam; i++)
        cout << v[i].x << " " << v[i].y << endl;

    if (v != nullptr) delete[] v;
    return 0;
}
```

Acesso não sequencial VI

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

Acesso não sequencial VII

```
int main(){
    float x[4] = { 5.6, 6, 9.8, 4.2 };
    float nx[4], tmp = 7.2;
    fstream pfile;
    pfile.open("num.bin", ios::binary|ios::in|ios::out);
    pfile.write((char*)x, sizeof(float)* 4);

    pfile.seekg(sizeof(float)*2, ios::beg);
    pfile.write((char*) &tmp, sizeof(float));
    pfile.seekg(0, ios::beg);

    pfile.read((char*)nx, sizeof(float)* 4);
    for (int i = 0; i < 4; i++)
        cout << nx[i] << " ";

    pfile.close();
    return 0;
}
```

FIM