Unidade 7

Arquivos
Texto e Binário

DEC0012 - Linguagem de Programação I
Curso de Engenharia de Computação
Profa. Andréa Sabedra Bordin

Introdução

Quando necessitamos armazenar dados de forma persistente, utilizamos **ARQUIVOS**.
Ao armazenar dados em ARQUIVOS, podemos acessar/recuperar esses dados sempre que houver necessidade.

Dados podem ser armazenados em ARQUIVOS, em duas formas: **TEXTO** e **BINÁRIO**.

Introdução

Arquivo texto - É um arquivo cujo conteúdo é baseado em uma sequência de caracteres que formam linhas determinadas por um caractere de nova linha (“\ n”). Dentro destes arquivos podem ser gravados apenas dados em forma de texto.

Arquivo binário – É um arquivo que tem o seu conteúdo baseado em uma estrutura ou dado que respeita um determinado tipo de dado. Este tipo de dado pode ser um tipo primitivo/simples (int, float, char) ou um tipo estruturado como o registro (struct). Portanto, é construído como uma sequência de bytes respeitando uma determinada estrutura.

Introdução

Um arquivo é manipulado com uma **variável do tipo ponteiro para arquivo**, que é declarada da seguinte forma:

```
#include<stdio.h>

int main()
{
FILE *arq;
}
```

Introdução

- Para se trabalhar com **arquivos** devemos ter sempre em mente:
- Se o arquivo **não existe**, devo **criar** o arquivo.
  - Uma vez criado, este arquivo está aberto e pronto para ser preenchido por dados.
- Se o arquivo **já existe**, então devo **abrir** este arquivo para ler os dados e imprimir ou ler e adicionar novos dados.

Abrir ou Criar um Arquivo

Para **abrir** um arquivo ou **criar** um arquivo novo é usada função **fopen().**
Quando chamada, essa função inicializa um objeto do tipo **FILE**, que contem a informação necessária para controle de entrada/saída de dados.

O protótipo da função **fopen()**:

FILE *fopen(const char * filename, const char * mode);
onde:
filename – é uma **string** que contem o nome do arquivo
mode – é uma variável que define a forma de acesso (as operações permitidas)

Fechar um Arquivo

O arquivo deve ser fechado usando a função **fclose().**

O protótipo dessa função tem o seguinte formato:
int fclose(FILE *fp);

Essa função retorna zero em caso de sucesso ou **EOF** se ocorrer um erro na hora de fechar o arquivo.
A função passa todos os dados pendentes para o arquivo, fecha o arquivo e libera a memória.

EOF – é uma constante definida em **stdio.h.**

Arquivo Texto – modos de abertura

Modo	Descrição
r	read – abre um arquivo texto existente para leitura
w	write – abre um arquivo texto para entrada de dados. Caso o arquivo não existe – ele é criado. O programa começa a escrever os dados no início do arquivo.
a	append – abre o arquivo para acrescentar os dados. Caso o arquivo não existe – ele é criado. O programa começa a escrever os dados no final do arquivo.
r+	abre o arquivo texto para entrada e saída de dados.
w+	abre o arquivo texto para entrada e saída de dados. Primeiro o tamanho da arquivo é zerado (caso o arquivo tinha tamanho diferente de zero), se o arquivo não existia ele é criado.
a+	abre o arquivo texto para entrada e saída de dados. A leitura começa do inicio do arquivo, mas os dados serão acrescentados no final do arquivo.

Arquivo Texto

Escrita de dados para arquivo

Para escrever os caracteres individuais pode ser usada a função:

```
int fputc( int c, FILE *fp );
```

Essa função escreve o valor do **c** para o arquivo referenciado pelo **fp**.
Retorna o valor do caractere escrito em caso de sucesso ou **EOF** em caso de falha.

Para escrever uma sequência de caracteres (que termina com **null**) pode ser usada a função:

```
int fputs( const char *s, FILE *fp );
```

Essa função escreve o **string s** para o arquivo referenciado pelo **fp**.
Vai retornar o valor não nulo em caso de sucesso ou **EOF** em caso de falha.

A função **fprintf()** também pode ser usada para gravar uma sequência de caracteres para o arquivo.

```
int fprintf(FILE *fp, const char *format, ...)
```

Arquivo Texto

Leitura de dados do arquivo

Para ler um caractere do arquivo é usada a função **fgetc():**

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

A função retorna o caractere lido ou **EOF** em caso de falha.

Para fazer a leitura de uma **string** de caracteres:

```
char *fgets( char *buf, int n, FILE *fp );
```

A função vai tentar fazer a leitura de **n-1** caracteres do arquivo referenciado pelo **fp**. Ela copia a sequência lida para **buf** e acrescenta o caractere **null** para indicar o fim da sequência de caracteres.

Se a função encontrar o símbolo de nova linha **'\n'** ou o fim do arquivo **EOF** antes de processar **n-1** caracteres, ela vai copiar somente caracteres encontrados até esse momento.

Outra função que pode ser usada para leitura de dados é:

```
int fscanf(FILE *fp, const char *format, ...)
```

Essa função vai fazer a leitura de dados de forma similar, porém vai parar a leitura se encontrar um espaço em branco(' ').

Exemplo 1: Criação do arquivo **output.txt**

```c
#include<stdio.h>
//criação do arquivo output.txt

int main()
{
    FILE *filePtr;
    int i;

    // criar um arquivo
    filePtr =fopen("output.txt", "w");

    if (filePtr == NULL)
     {
       printf("\n Erro! Não foi possivel criar arquivo! \n ");
       return 1; // código de eero
     }

    // gravar dados para arquivo
    for (i = 1; i <= 10; i++)
       fprintf(filePtr,"%d\n", i);

    // fechar o arquivo
    printf("\n O programa gravou números de [1,10] para arquivo output.txt \n ");
    fclose(filePtr);

    return  0;
}
```

O programa gravou números de [1,10] para arquivo output.txt

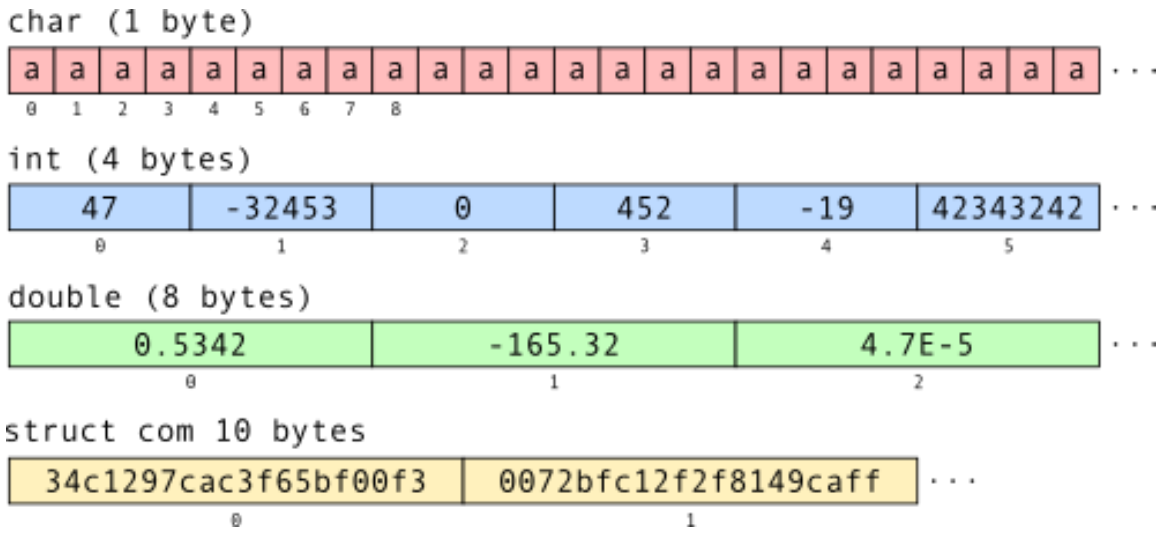
Exemplo 2: Leitura de dados do arquivo **output.txt**

```c
#include<stdio.h>
// leitura de dados do arquivo output.txt
// o arquivo output.txt deve estar presente na pasta atual
int main()
{
    FILE *in_fPtr;
    int num;

    // abertura do arquivo para leitura
    in_fPtr =fopen("output.txt", "r");

    if (in_fPtr == NULL)
     {
       printf("\n Erro! Não foi possivel abrir o arquivo! \n ");
       return 1; // código de eero
     }

    // leitura de dados do arquivo
    printf("\n Leitura de dados do arquivo output.txt \n");
    while( fscanf(in_fPtr, "%d", &num ) == 1 )
       printf("Número: %d\n", num);

    // fechar o arquivo
    fclose(in_fPtr);

    return  0;
}
```

```
 Leitura de dados do arquivo output.txt
Número: 1
Número: 2
Número: 3
Número: 4
Número: 5
Número: 6
Número: 7
Número: 8
Número: 9
Número: 10
```

Arquivo binário

Em C, um arquivo binário é visto como uma sequência de blocos de mesmo tamanho.
O tamanho dos blocos depende do tipo de informação armazenada no arquivo.
Por exemplo, um arquivo de números reais *double* terá blocos de 8 *bytes*, enquanto um arquivo de caracteres (*char*) terá blocos de 1 *byte*, como mostra a figura:



Arquivo binário

- Os arquivos binários têm **três recursos** que os distinguem dos arquivos de texto:

1. Você pode pular instantaneamente para qualquer estrutura no arquivo (acesso aleatório).
2. Você pode alterar o conteúdo de uma estrutura em qualquer lugar do arquivo a qualquer momento.
3. Arquivos binários costumam ter tempos de leitura e gravação mais rápidos do que arquivos de texto, porque uma imagem binária do registro é armazenada diretamente da memória para o disco (ou vice-versa).

Arquivos binários

- A representação binária de tipos numéricos simples como int e float costuma ser mais **compacta** do que sua representação como texto.
- Por exemplo, o número 12345678 é representado com 8 caracteres (**8 bytes**) em modo texto, mas com somente **4 bytes** de forma binária.
- Por isso, para gravar grandes quantidades de números em arquivos é preferível usar o formato binário.

```c
#include <stdio.h>

void main(){
   int vet[4]={1,2,3,4};

   FILE *arq ;
   // Abre o arquivo com o nome no modo de escrita binaria
   arq = fopen ("vetor" , "wb") ;

   // Grava o vetor vet, que tem tam elementos int, no arquivo arq
   fwrite ( vet , sizeof (int) , 4 , arq ) ;
   // Fecha o arquivo
   fclose(arq);

   //----------

   // Abre o arquivo com o nome pedido no modo de leitura
   arq = fopen (nomearq , "rb") ;

   //Lê o bloco  para vetor
   int vetleitura[4];
   if (arq != NULL)
      fread(vetleitura, sizeof(int), 4, arq);

   //Escreve o vetor lido
   for (int i=0;i<4;i++)
      printf("Valor: %d \n", vetleitura[i]);

   // Fecha o arquivo
   fclose(arq);
}
```

fopen(nome_arquivo, modo)

- Abre um arquivo qualquer para ser manipulado pelo seu programa.
- Retorna um ponteiro para este arquivo ou NULL em caso de erros.
- **nome_arquivo**: é o caminho do sistema para o arquivo que se deseja abrir. Exemplo: “C:\Users\matheus\arquivo.txt” OU “/home/matheus/arquivo.txt”
- **modo**: é o modo em que o arquivo será manipulado. Veja os tipos de modo no próximo slide.

Modo	Significado
"r"	Abre um arquivo texto para leitura. O arquivo deve existir antes de ser aberto.
"w"	Abrir um arquivo texto para gravação. Se o arquivo não existir, ele será criado. Se já existir, o conteúdo anterior será destruído.
"a"	Abrir um arquivo texto para gravação. Os dados serão adicionados no fim do arquivo ("append"), se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"rb"	Abre um arquivo binário para leitura. Igual ao modo "r" anterior, só que o arquivo é binário.
"wb"	Cria um arquivo binário para escrita, como no modo "w" anterior, só que o arquivo é binário.
"ab"	Acrescenta dados binários no fim do arquivo, como no modo "a" anterior, só que o arquivo é binário.
"r+"	Abre um arquivo texto para leitura e gravação. O arquivo deve existir e pode ser modificado.
"w+"	Cria um arquivo texto para leitura e gravação. Se o arquivo existir, o conteúdo anterior será destruído. Se não existir, será criado.
"a+"	Abre um arquivo texto para gravação e leitura. Os dados serão adicionados no fim do arquivo se ele já existir, ou um novo arquivo será criado, no caso de arquivo não existente anteriormente.
"r+b"	Abre um arquivo binário para leitura e escrita. O mesmo que "r+" acima, só que o arquivo é binário.
"w+b"	Cria um arquivo binário para leitura e escrita. O mesmo que "w+" acima, só que o arquivo é binário.
"a+b"	Acrescenta dados ou cria uma arquivo binário para leitura e escrita. O mesmo que "a+" acima, só que o arquivo é binário

Leitura/escrita de blocos

- fread(buffer, sizeof, número, ponteiro)
- fwrite(buffer, sizeof, número, ponteiro)

fread(buffer, sizeof, número, ponteiro)

- Faz a leitura de (sizeof bytes * número) do arquivo e salva o conteúdo em buffer. Retorna o próprio número, ou algo diferente se deu erro pra ler os dados.
- **buffer**: uma posição de memória para salvar os dados lidos. Tem que já estar alocado dinamicamente (*malloc*) ou estaticamente com o tamanho mínimo que caiba os bytes lidos.
- **sizeof**: o tamanho em bytes de cada elemento lido.
- **número**: o número de elementos a serem lidos.
- **ponteiro**: o ponteiro para o arquivo retornado por “fopen”.

fread(buffer, sizeof, número, ponteiro)

- **Exemplos**:
- ler 1 byte do arquivo: fread(byte_lido, **1**, 1, ponteiro);
- ler um inteiro do arquivo:
  - fread(valor, **4**, 1, ponteiro) ou
  - fread(valor, sizeof(int), 1, ponteiro);
- ler 10 inteiros do arquivo:
  - fread(vetor, **4**, 10, ponteiro) ou
  - fread(vetor, sizeof(int), 10, ponteiro);
- ler uma string de tamanho 40 do arquivo:
  - fread(str, 1, 40, ponteiro) ou
  - fread(str, sizeof(char), 40, ponteiro);
- ler uma struct inteira de uma vez do arquivo:
  - fread(struct, sizeof(struct), 1, ponteiro);

fwrite(buffer, sizeof, número, ponteiro)

- Faz a escrita de sizeof * número bytes no arquivo que estão salvos em buffer.
  - Se não tiver no fim do arquivo, vai sobrescrever os dados existentes (exceção se foi aberto com “a”).
  - Se tiver no fim, aumenta o tamanho do arquivo.
- Retorna o próprio número, ou algo diferente se deu erro pra escrever os dados.
- **buffer**: uma posição de memória para salvar os dados lidos. Tem que já estar alocado dinamicamente (malloc) ou estaticamente com o tamanho mínimo que caiba os bytes lidos.
- **sizeof**: o tamanho em **bytes** de cada elemento lido.
- **número**: o número de elementos a serem lidos.
- **ponteiro**: o ponteiro retornado por “fopen”.

fwrite(buffer, sizeof, número, ponteiro)

- **Exemplos**:
- escrever 1 byte no arquivo:
  - fwrite(byte_pra_escrever, **1**, 1, ponteiro);
- escrever um inteiro no arquivo:
  - fwrite(valor, 4, 1, ponteiro) ou fwrite(valor, sizeof(int), 1, ponteiro);
- escrever 10 inteiros no arquivo:
  - fwrite(vetor, 4, 10, ponteiro) ou fwrite(vetor, sizeof(int), 10, ponteiro);
- escrever uma string de tamanho 40 no arquivo:
  - fwrite(str, 1, 40, ponteiro) ou fwrite(str, sizeof(char), 40, ponteiro);
- escrever uma struct inteira de uma vez no arquivo:
  - fwrite(struct, sizeof(struct), 1, ponteiro);

fseek(ponteiro, offset, origem)

- Uma vantagem do formato binário é que o tamanho ocupado por cada elemento é conhecido.
- Isso significa que é possível calcular a posição de cada um no arquivo.
- Uma vez sabida a posição de um elemento que se deseja acessar, o arquivo não precisa ser lido por completo.
- A função **fseek** pode ser usada para acessar diretamente qualquer ponto do arquivo.

fseek(ponteiro, offset, origem)

- Ajusta o ponteiro do arquivo para a posição "*offset*" a partir de "origem".
- **ponteiro**: o ponteiro retornado por "fopen".
- **offset**: o número de bytes que vai pular, ou melhor, o *byteoffset*.
- **origem**:
  - SEEK_SET para início do arquivo.
  - SEEK_CUR para posição atual do ponteiro.
  - SEEK_END para fim do arquivo.
- Exemplos:
  - ir para o início do arquivo: fseek(ponteiro, 0, SEEK_SET);
  - ir para o final do arquivo: fseek(ponteiro, 0, SEEK_END);
  - ir para o *byteoffset* 1024 do arquivo: fseek(ponteiro, 1024, SEEK_SET);
  - voltar 4 bytes da posição atual:
    - fseek(ponteiro, -4, SEEK_CUR) ou
    - fseek(ponteiro, -sizeof(int), SEEK_CUR);
  - ir para o offset do último byte do arquivo: fseek(ponteiro, -1, SEEK_END);

fclose(ponteiro)

- Fecha um arquivo que estava em uso e o libera para que outros programas usem.
- Sempre que finalizar a manipulação de um arquivo, não esqueça de usar essa função para fechar ele.
- Isso diz pro sistema operacional: “*Terminei de mexer nesse arquivo. Se outro programa quiser usar ele, agora pode*.”
- Vale lembrar que não é possível abrir o mesmo arquivo sem antes ter fechado ele.
  - Mas se o seu programa se encerrar, o sistema operacional automaticamente fecha o arquivo.
- **ponteiro**: o ponteiro retornado pelo “fopen”.

Referências

- Notas de aula profa. Olga. UFSC.
- Notas de aula prof. Maziero. UTFPR.