

Organização de Arquivos

Prof. Jefferson T. Oliva

Algoritmos e Estrutura de Dados I (AE22CP)
Engenharia de Computação
Departamento Acadêmico de Informática (Dainf)
Universidade Tecnológica Federal do Paraná (UTFPR)
Campus Pato Branco

- *Stream*
- Arquivos
- Arquivos Binários
- Arquivos: Outras Funções
- Fluxos Padrão

- O armazenamento de dados na memória RAM é temporário
- Arquivos são utilizados para o armazenamento de dados em dispositivos secundários (HD, pendrive, cartão de memória, CD, DVD, etc)
- Sistema de arquivos¹



- Arquivos:
 - Coleção de informações armazenadas em memória secundária
 - Dados, programas, etc
 - Arquivos de texto e binários
 - Gerenciados pelo sistema operacional

- Atributos de arquivos
 - Nome
 - Identificador
 - Tipo
 - Posição
 - Tamanho
 - Proteção
 - Hora, data e identificação do usuário
- Uma biblioteca da linguagem C define um conjunto completo de funções de entrada/saída
 - *#define <stdio.h>*

Stream

- O sistema de arquivos C é projetado para possibilitar o uso de vários dispositivos (discos, impressoras, teclados)
- O sistema de arquivos transforma os dados carregados em um dispositivo lógico denominado *stream*
- Apesar dos dispositivos de entrada/saída serem distintos, todas as *streams* comportam-se de forma similar, ou seja, são amplamente independentes de dispositivo
- Existem dois tipos de *streams*
 - *Stream* do tipo texto: sequência de caracteres
 - *Stream* binária: sequência de bytes

Arquivos

- Na linguagem C, um arquivo é uma sequência de *bytes*
- Quanto um arquivo é aberto, um ponteiro é associado à sua respectiva *stream*
- Os arquivos são manipulados por meio de ponteiros (`FILE *`)
- `FILE * arquivo = fopen(char * nome_arquivo, char * modo)`
 - A função `fopen` retorna um ponteiro do tipo `FILE`
 - Caso ocorra um erro na operação, a função retorna `NULL`
 - `nome_arquivo`: é o nome e a localização do arquivo.
Exemplos `"teste1.txt"`, `"/pasta/teste.txt"`
 - `modo`: determina a operação que deverá ser aplicada no arquivo (leitura, escrita, etc)

- Modos de processamento de arquivos do tipo texto:
 - `"r"`: abre um arquivo texto para leitura
 - `"w"`: abre um arquivo texto para escrita. Caso o arquivo não exista, o mesmo será criado, caso contrário, o conteúdo do mesmo é apagado
 - `"a"`: abre um arquivo texto para escrita no final
 - `"r+"`: abre um arquivo texto para leitura e gravação (o arquivo deve existir e pode ser alterado)
 - `"w+"`: abre um arquivo texto para leitura e gravação. Se o arquivo não existir, o mesmo é criado, caso contrário o conteúdo do mesmo é apagado
 - `"a+"`: abre um arquivo texto para leitura e gravação. Os dados são adicionados no final do arquivo

- Após o uso do arquivo, o mesmo deve ser fechado por meio da função *fclose*
 - *int resultado = fclose(FILE * arquivo)*
- A função *fclose* retorna 0, caso a operação seja bem-sucedida, ou 1, caso ocorra um erro na operação

- Funções principais para o processamento de arquivos texto:
 - *int getc(FILE * arquivo)* ou *int fgetc(FILE * arquivo)*: lê caracteres no arquivo texto
 - *int putc(int ch, FILE * arquivo)* ou *int fputc(int ch, FILE * arquivo)*: escreve caracteres no arquivo texto. Retorna 0 caso a operação seja bem-sucedida, ou 1, caso ocorra um erro
 - *char * fgets(char * string, int n, FILE * arquivo)*: lê uma string de tamanho *n* no arquivo
 - *char * fputs(char * string, FILE * arquivo)*: escreve uma string no arquivo texto
 - *int feof(FILE * arquivo)*: retorna um valor diferente de 0 se o final do arquivo foi atingido

- Exemplo:

```
void print_file_content(char * path){
    FILE * arq = fopen(path, "r");
    char str[100];

    if (arq){
        while (!feof(arq)){
            fgets(str, 100, arq);
            printf("%s", str);
        }

        printf("\n");

        int close = fclose(arq);

        if (close == 0)
            printf("Operacao bem sucedida!\n");
        else
            printf("Falha ao fechar o arquivo.\n");
    }
}
```

- Outros exemplos

```
void write_file(char * path, char * content){  
    FILE * arq = fopen(path, "w");  
  
    fputs(content, arq);  
  
    fclose(arq);  
}
```

```
void append_file(char * path, char * content, unsigned int times){  
    FILE * arq = fopen(path, "a");  
    int i;  
  
    if (arq){  
        for (i = 0; i < times; i++){  
            fputs("\n", arq);  
            fputs(content, arq);  
        }  
  
        fclose(arq);  
    }  
}
```

Arquivos Binários

- Arquivos texto são simples de serem manipulados e podem ser facilmente compreendido por humanos
- Desvantagens dos arquivos texto
 - Os registros devem ser separados por algum caractere ou identificador
 - O acesso ao conteúdo é sequencial
 - Os caracteres numéricos são armazenados em formato ASCII
 - Variáveis do tipo *short*, *int*, *long*, *float* e *double* possuem tamanho fixo na memória
 - Para representação de números em arquivos texto, cada dígito é representado por um caractere equivalente (exemplo: 300000.123)

Arquivos Binários

- Em arquivos binários, o conteúdo é armazenado em formato binário, possibilitando o uso de menor quantidade de espaço em relação aos arquivos do tipo texto
- Não é necessário o uso de caracteres ou identificadores para separar cada registro
- O acesso ao conteúdo em arquivos binários não é sequencial, isto é, basta o conhecimento da posição em que um determinado registro esteja inserido
- Arquivos binários podem ter diversas extensões (de acordo com o programa que os lê), por exemplo ".dat"
- Na linguagem C, para o processamento de arquivos binários deve ser utilizada uma biblioteca apropriada
 - `#define <stdlib.h>`

Arquivos Binários

- Os arquivos binários também são manipulados por meio de ponteiros (FILE *)
 - *FILE * arquivo = fopen(char * nome_arquivo, char * modo)*
- Modos de processamento de arquivos binários:
 - "rb": abre um arquivo binário para leitura
 - "wb": abre um arquivo binário para escrita
 - "ab": abre um arquivo binário para escrita no final
 - "r+b" ou "rb+": abre um arquivo binário para leitura e gravação (o arquivo deve existir e pode ser alterado)
 - "w+b" ou "wb+": abre um arquivo binário para leitura e gravação
 - "a+b" ou "ab+": abre um arquivo binário para leitura e gravação. Os dados são adicionados no final do arquivo

- Funções principais para o processamento de arquivos binários:
 - *int fread(void * buffer, int qtd_bytes, int n_unidades, FILE * arquivo)*: lê o conteúdo de um arquivo binário
 - *buffer*: região da memória onde os dados são armazenados
 - *qtd_bytes*: número de bytes que deverão ser lidos por unidade
 - *n_unidade*: quantidade de unidades que deverão ser lidas
 - Essa função retorna a quantidade de unidades lidas efetivamente
 - *int fwrite(void * buffer, int qtd_bytes, int n_unidades, FILE * arquivo)*: escreve dados em um arquivo binário

- Funções principais para o processamento de arquivos binários:
 - *int fseek(FILE * arquivo, int qtd_bytes, int posicao_origem)*: move a posição do cursor (*qtd_bytes*) a partir de um localização específica (*posicao_origem*)
 - Na variável *posicao_origem* pode ser atribuído os seguintes valores
 - *SEEK_SET*: a partir da posição inicial do arquivo (corresponde ao valor 0)
 - *SEEK_CUR*: a partir da posição atual (corresponde ao valor 1)
 - *SEEK_END*: a partir do final do arquivo (corresponde ao valor 2)
 - *void rewind(FILE * arquivo)*: retorna o cursor ao início do arquivo

- Exemplo

```
typedef struct{
    long RA;
    char nome[70];
    int codigo_curso;
    float coef;
}Aluno;

void write_bin_file(char * path, Aluno content[], int n){
    FILE * arq = fopen(path, "wb");
    int i = 0;

    if (arq){
        fwrite(content, sizeof(Aluno), n, arq);

        fclose(arq);
    }else
        printf("Erro ao gerar o arquivo");
}
```

- Exemplo

```
void read_bin_file(char * path, Aluno content[], int n){  
    FILE * arq = fopen(path, "rb");  
  
    if (arq){  
        fread(content, sizeof(Aluno), n, arq);  
  
        fclose(arq);  
    }else  
        printf("Erro ao abrir o arquivo");  
}
```

- Exemplo

```
int getPosicaoAluno(Aluno Aluno[], int n, char * nome){
    int i;

    for (i = 0; i < n; i++)
        if (strcmp(Aluno[i].nome, nome) == 0)
            return i;

    return -1;
}

void setAluno(char * path, int position, Aluno *aluno){
    FILE * arq = fopen(path, "r+b");

    if (arq){
        fseek(arq, sizeof(Aluno) * (position), SEEK_SET);

        fwrite(aluno, sizeof(Aluno), 1, arq);

        fclose(arq);
    }
}
```

Arquivos: Outras Funções

- *int fprintf(FILE * arquivo, char * string, ...)*: possui a mesma finalidade que a função *printf()*, porém escreve em um arquivo texto
- *int fscanf(FILE * arquivo, char * string, ...)*: possui a mesma finalidade que a função *scanf()*, porém lê de um arquivo texto
- *int ferror(FILE * arquivo)*: verifica se ocorreu um erro no *arquivo* apontado por *FILE*
- *int remove(FILE * arquivo)*: deleta o arquivo
- *ftell(FILE * arquivo)*: retorna a posição do ponteiro

Fluxos Padrão

Fluxos Padrão

- *stdin*: entrada padrão
- *stdout*: saída padrão
- *stderr*: saída de erro padrão
- *stdaux*: dispositivo de saída auxiliar
- *stdprn*: dispositivo de impressão padrão

- Exemplo:

```
void main(){
    char str[20];

    printf("Digite algo: ");
    fgets(str, 20, stdin);

    printf("\n\nVc digitou: %s", str);
}
```

- Reimplemente a função **write_bin_file** apresentado em sala de aula para a gravação dos registros em arquivos no formato texto
- Reimplemente a função **read_bin_file** apresentado em sala de aula para a leitura dos registros em arquivos no formato texto



Deitel, H. M. e and Deitel, P. J.
C: Como Programar.
Pearson, 2011.



Schidildt, H.
C Completo e Total.
Pearson, 2011.