

Grupo 1 – Abertura e criação de arquivos

1) Como verificar se um arquivo existe no Sistema Operacional?

A função `fopen` é utilizada para abrir, criar e recriar arquivos. A combinação de valores no segundo parâmetro, “mode” define a forma de abertura do arquivo.

```
#include <stdio.h>
int main(int argc, char *argv[]) {
    FILE *fp;
    char nomeArquivo[] = "nome_do_arquivo.txt";

    if( (fp=fopen(nomeArquivo,"r")) != NULL ) {
        printf("O arquivo %s existe.\n\n",nomeArquivo);
        fclose(fp);
    } else {
        printf("O arquivo %s não foi localizado.\n\n",nomeArquivo);
    }
    return 0;
}
```

O primeiro parâmetro “r” abre o arquivo apenas para leitura, se o arquivo existir. Caso contrário, a função retorna o valor NULL.

2) Como criar um arquivo texto?

```
include <stdio.h>
int main(int argc, char *argv[]) {
    FILE *fp;
    char nomeArquivo[] = "um_arquivo_texto.txt";

    if( (fp=fopen(nomeArquivo,"wt")) != NULL ) {
        printf("O arquivo %s foi criado.\n\n",nomeArquivo);
        fclose(fp);
    } else {
        printf("Nao foi posivel criar o arquivo %s.\n\n",nomeArquivo);
    }
    return 0;
}
```

Observe a presença do “t” no parâmetro modo do `fopen`, ele tem efeito apenas no Windows, que distingue arquivos textos de arquivos binários. O parâmetro é ignorado no Linux.

3) Como criar um arquivo binário?

Modifique o parâmetro “wt” para “wb” no exemplo anterior.

4) Quais são os “modos”?

Modo	Função
r	Abrir um arquivo existe, apenas para leitura
w	Abre o arquivo para escrita, se o arquivo existir ele será truncado (zerado), caso contrário ele será criado.
a	Abre o arquivo em modo de “append”, apenas para escrita no final do arquivo.
r+	Abre um arquivo existente para leitura e escrita. O cursor é posicionado no início do arquivo.
w+	Abre um arquivo para leitura e escrita. Se o arquivo existir ele será truncado, caso contrário será criado.
a+	Abre ou cria o arquivo para leitura ou append (anexo). Se o arquivo existir o conteúdo não é modificado, caso contrário o arquivo é criado. O cursor de leitura é posicionado no início do arquivo e o cursor de escrita no final do arquivo.

Fonte: https://www.gnu.org/software/libc/manual/html_node/Opening-Streams.html

Grupo 2 – Ler dados de arquivo

1) Como ler um arquivo símbolo a símbolo?

```
#include <stdio.h>

int main(int argc, char *argv[]) {
    FILE *fp;
    char nomeArquivo[] = "genes_aa.fa";
    int ch;
    if( (fp=fopen(nomeArquivo,"rt")) == NULL ) {
        printf("O arquivo %s não foi localizado.\n\n",nomeArquivo);
        fclose(fp);
        return -1;
    }

    while( !feof(fp) ) {
        ch = fgetc(fp);
        putchar(ch);
    }

    fclose(fp);
    return 0;
}
```

No código acima, a função `feof()` retorna um valor diferente de zero se o final do *stream* foi alcançado. A função `fgetc` lê um símbolo do arquivo, desloca o cursor para a próxima posição e retorna o código ASCII do símbolo. Em arquivos codificados em UTF o valor retornado pode ser maior que 255 (limite do unsigned char).

2) Como ler um arquivo texto linha a linha?

O código a seguir faz uso da função `puts` para imprimir no stream padrão de saída (`stdout`, geralmente a tela) a linha lida. Mas a função `fgets` inclui a quebra de linha no buffer e a função `puts` imprime o buffer e mais uma quebra de linha, com isso o programa precisa “limpar” uma das quebras de linha para não imprimir duas vezes. Essa limpeza ocorre a partir da linha “`p = buffer`” que obtém o endereço de memória de buffer e o percorre até encontrar uma quebra de linha ou um nulo (`\0`).

Observe também que `fgets` recebe três parâmetros, sendo o endereço de memória (RAM) onde são copiados os dados lidos do arquivo para a memória RAM, o limite máximo que pode ser lido e que corresponde com a capacidade da variável passada no primeiro parâmetro e no terceiro parâmetro é informado o handle do arquivo (file pointer ou stream).

A função `fgets` só tem sentido em ser utilizado em arquivos textos, onde as quebras de linhas (`\n`) são interpretadas como delimitadores das linhas dos textos.

```
#include <stdio.h>
#define MAX_LINE_BUFFER 32768
int main(int argc, char *argv[]) {
    FILE *fp;
    char nomeArquivo[] = " genes_aa.fa";
    char buffer[MAX_LINE_BUFFER],*p;

    if( (fp=fopen(nomeArquivo,"rt")) == NULL ) {
        printf("O arquivo %s não foi localizado.\n\n",nomeArquivo);
        fclose(fp);
        return -1;
    }

    while( !feof(fp) ) {
        if( fgets(buffer, MAX_LINE_BUFFER, fp) != NULL ) {
            // remove a quebra de linha
            p = buffer;
            while( *p != '\0' ) {
                if( *p == '\n' ) {
                    *p = '\0';
                } else {
                    p++;
                }
            }
            puts( buffer );
        }
    }
    fclose(fp);
    return 0;
}
```

3) Como ler um arquivo binário?

Os arquivos binários geralmente seguem uma estrutura padrão conhecida, tais como WAV para som, BMP ou JPEG para imagens, DBF para dados etc. Mas existem programas que geram seus próprios formatos e são chamados de “formatos proprietários”, onde apenas o programa que gerou os arquivos conhece sua organização interna. No exemplo a seguir, o arquivo de dados foi criado pelo programa descrito na questão 2 do grupo 3.

```
#include <stdio.h>

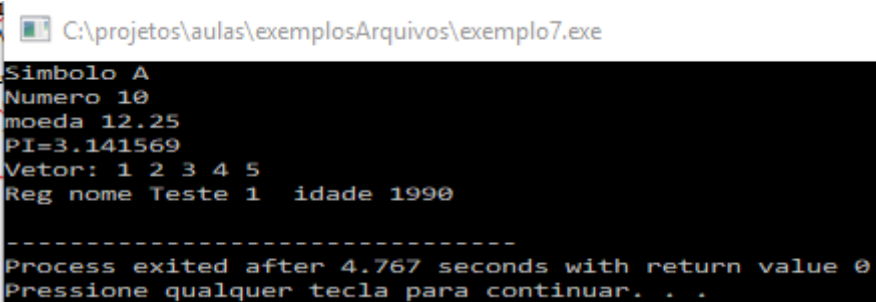
int main(int argc, char *argv[]) {
    FILE *fp;
    char nomeArquivo[] = "gerandoArquivoBinario.dat";
    char simbolo;          // = 'A';
    int numero;             // = 10;
    float moeda;            // = 12.25f;
    double pi;              // = 3.141569;
    int vetor[5];           // = {1,2,3,4,5};
    struct _REG {
        char nome[40];
        int ano_nascimento;
    } r;                    // nome: Teste 1, nascimento: 1990
```

```

if( (fp=fopen(nomeArquivo,"rb")) == NULL ) {
    printf("O arquivo %s não foi localizado.\n\n",nomeArquivo);
    fclose(fp);
    return -1;
}
// grava os dados de forma binaria - tipos primitivos
fread((void *)&simbolo, sizeof(char), 1, fp);
fread((void *)&numero, sizeof(int), 1, fp);
fread((void *)&moeda, sizeof(float), 1, fp);
fread((void *)&pi, sizeof(double), 1, fp);
// vetor
fread((void *)vetor, sizeof(int), 5, fp);
// struct
fread((void *)&r, sizeof(struct _REG), 1, fp);
printf("Simbolo %c \nNumero %d\nmoeda %5.2f\nPI=%f\nVetor: %d %d %d %d %d\nReg nome %s idade %d\n",
    simbolo,numero,moeda,pi,
    vetor[0],vetor[1],vetor[2],vetor[3],vetor[4],
    r.nome, r.ano_nascimento
);
fclose(fp);
return 0;
}

```

Nesse caso, como os dados são estaticamente definidos no programa que gera o arquivo de dados binários, a saída do programa deve ser:



```

C:\projetos\aulas\exemplosArquivos\exemplo7.exe
Simbolo A
Numero 10
moeda 12.25
PI=3.141569
Vetor: 1 2 3 4 5
Reg nome Teste 1 idade 1990
-----
Process exited after 4.767 seconds with return value 0
Pressione qualquer tecla para continuar. . .

```

Observe que na função fopen foi utilizado o modo "rb", onde o "b" de binário só tem sentido no Sistema Operacional MS Windows, o Unix/Linux ignora essa informação. A sequência dos fread's devem ser obrigatoriamente a mesma sequência dos fwrite's do programa que criou e salvo os dados no arquivo gerandoArquivoBinario.dat.

Grupo 3 – Escrever dados em um arquivo

1) Como escrever em um arquivo texto?

As funções fputc, fputs e fprintf podem ser utilizadas para escrita de textos em um arquivo, sendo que a primeira escreve símbolo a símbolo, a segunda escreve uma linha e fprintf permite gerar uma saída formatada. Segue um exemplo:

```

#include <stdio.h>
int main(int argc, char *argv[]) {
    FILE *fp;
    char nomeArquivo[] = "gerandoArquivoTexto.txt";

    if( (fp=fopen(nomeArquivo,"wt")) == NULL ) {
        printf("O arquivo %s não foi localizado.\n\n",nomeArquivo);
    }
}

```

```

        fclose(fp);
        return -1;
    }
    // escrita simbolo a simbolo - gerando a 1a linha do arquivo
    fputc(67, fp);
    fputc(65, fp);
    fputc((int)'S', fp);
    fputc((int)'A', fp);
    fputc((int)'\n', fp);
    // escrevendo a 2a linha do arquivo
    fputs("Era uma vez...\n", fp);
    // escrevendo a 3a e 4a linha do arquivo com funcao formatada
    fprintf(fp, "Nome: %-50s\tMatricula: %6d\n    Data do cadastro: %s\n",
        "Fulano de tal", 123, "27/11/2017");
    fclose(fp);
    return 0;
}

```

Os valores 67 e 65 correspondem aos símbolos ASCII C e A, os casts utilizados em S, A e \n são recomendados para evitar avisos do compilador.

2) Como escrever dados em um arquivo binário?

A função fwrite é utilizada para escrever um “buffer” de dados de qualquer tipo para um arquivo de forma binária, ou seja, copiando os bytes da RAM para o disco. Os dados podem ser tipos primitivos (char, int, float, double) ou mais complexos como vetores e estruturas. O importante é lembrar que a informação tem que estar em uma região continua da RAM, nesse sentido, tome muito cuidado com vetor de ponteiros ou estruturas com campos de ponteiros.

```

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

int main(int argc, char *argv[]) {
    FILE *fp;
    char nomeArquivo[] = "gerandoArquivoBinario.dat";
    char simbolo = 'A';
    int numero = 10;
    float moeda = 12.25f;
    double pi = 3.141569;
    int vetor[] = {1,2,3,4,5};
    struct _REG {
        char nome[40];
        int ano_nascimento;
    } r;
    if( (fp=fopen(nomeArquivo,"wb")) == NULL ) {
        printf("O arquivo %s não foi criado.\n\n", nomeArquivo);
        fclose(fp);
        return -1;
    }
    strcpy(r.nome, "Teste 1");
    r.ano_nascimento = 1990;
    // grava os dados de forma binaria - tipos primitivos
    fwrite((void *)&simbolo, sizeof(char), 1, fp);
    fwrite((void *)&numero, sizeof(int), 1, fp);
    fwrite((void *)&moeda, sizeof(float), 1, fp);
    fwrite((void *)&pi, sizeof(double), 1, fp);
    // vetor
    fwrite((void *)vetor, sizeof(int), 5, fp);
    // struct
    fwrite((void *)&r, sizeof(struct _REG), 1, fp);
}

```

```

        fclose(fp);
        return 0;
    }

```

No código acima, deve-se observar o cast (void *) utilizado para evitar avisos do compilador, bem como a ausência do operador & que obtém o endereço de memória das variáveis. No caso do vetor (variável vetor) como não foi informado a posição de um elemento específico, a linguagem compreende que é o endereço de memória da “base do vetor” ou do elemento inicial (posição 0). Nessa linha tem outra diferença, a quantidade de dados que será salva e que se refere ao número de elementos do vetor, ou seja, 5 elementos. Na estrutura o campo nome com 40 posições e o campo do ano de nascimento serão escritos em uma única chamada da função fwrite.

A função fwrite tem como parâmetros:

```

fwrite( void * buffer, int size, int count, FILE *stream)

```

Onde buffer é o endereço de memória da posição inicial onde são lidos os bytes a serem copiados para o disco, size é o tamanho do tipo do dado, count o número de elementos e stream o handle do arquivo onde são escritos os dados.

Observe ainda que esse arquivo só pode ser lido por um programa que carregue os dados na mesma ordem que foi escrita, uma vez que o mesmo não segue nenhum padrão conhecido. O programa correspondente encontra-se descrito na seção anterior, questão 3.

Grupo 4 – Posição do “cursor”

Para cada arquivo aberto existe pelo menos um cursor associado e gerenciado pelo Sistema Operacional, ao abrir o arquivo o cursor é posicionado “antes do início do arquivo”, ou na posição ZERO (em algumas linguagens, existe a função bof – acrônimo de begin of file que retorna true nessa posição). A exceção é no modo “a” em que o cursor é posicionado no final do arquivo. Após qualquer operação de leitura ou escrita o cursor é deslocado em bytes que corresponde ao tamanho da informação lida ou gravada. A função ftell é utilizada para obter a posição atual do cursor e a função fseek para deslocamento arbitrário.

1) Como encontrar a posição física do início de cada linha em um arquivo texto?

```

#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
    char arquivo[] = "exe8main.c";
    char buffer[512];
    FILE *fp;
    long pos;
    int numLinha = 1;
    if( (fp=fopen(arquivo,"rt")) == NULL ) {
        printf("Arquivo %s nao encontrado.\n\n",arquivo);
        exit(-1);
    }
    pos = ftell(fp);
    while( fgets(buffer, 512, fp) != NULL ) {
        printf("linha %d inicia em %ld\n",numLinha,pos);
        numLinha++;
        pos = ftell(fp);
    }
    fclose(fp);
    return 0;
}

```

```
}
```

O arquivo `exe8main.c` contém o código acima, e exceto se ocorrer alguma diferença proporcionada por espaços em branco ou outros símbolos, a saída do programa deve ser:


```
linha 1 inicia em 0
linha 2 inicia em 20
linha 3 inicia em 41
linha 4 inicia em 43
linha 5 inicia em 79
linha 6 inicia em 112
linha 7 inicia em 132
linha 8 inicia em 144
linha 9 inicia em 156
linha 10 inicia em 176
linha 11 inicia em 219
linha 12 inicia em 272
linha 13 inicia em 285
linha 14 inicia em 289
linha 15 inicia em 308
linha 16 inicia em 352
linha 17 inicia em 404
linha 18 inicia em 419
linha 19 inicia em 439
linha 20 inicia em 443
linha 21 inicia em 457
linha 22 inicia em 469
```

2) Com base da informação do programa anterior, como podemos ler apenas a quarta linha do programa, sem ler as linhas anteriores?

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

int main(int argc, char *argv[]) {
    char arquivo[] = "exe8main.c";
    char buffer[512];
    FILE *fp;
    long pos = 43;
    if( (fp=fopen(arquivo,"rt")) == NULL ) {
        printf("Arquivo %s nao encontrado.\n\n",arquivo);
        exit(-1);
    }
    fseek(fp, pos, SEEK_SET);
    if( fgetc(buffer, 512, fp) != NULL ) {
        printf("%s",buffer);
    }
    fclose(fp);
    return 0;
}
```

Saída esperada:

 C:\projetos\aulas\exemplosArquivos\exemplo9.exe

```
>int main(int argc, char *argv[]) {
```

Observe que a função `fgetc` é chamada apenas uma vez, portanto ela não leu as linhas anteriores, isso ocorreu porque a função `fseek` deslocou o cursor para a posição 43 (ou byte 43), posição onde `fgetc` iniciou a leitura dos dados.

