



# Estruturas de Dados 1

## 06 – Manipulação de arquivos

Antonio Angelo de Souza Tartaglia  
angelot@ifsp.edu.br

# Estrutura de Dados 1

## Arquivos

- Até aqui, quase todos os programas que implementamos, solicitavam ao usuário os dados que eram então manipulados;
- Uma vez terminado o programa, todos os dados introduzidos, ou mesmo os resultados do programa, eram perdidos, pois não eram armazenados de forma definitiva em um repositório permanente;
- Ao contrário de outras linguagens em que os arquivos são vistos como tendo uma estrutura interna ou registro associado, em “C” um arquivo é apenas um conjunto de **bytes** colocado uns após outros de forma sequencial;
- A utilização de arquivos em “C” pode ser vista em dois sentidos distintos:
  - O arquivo é fonte de dados para o programa: nesse caso trata-se de um arquivo de entrada de dados (*input*);
  - O arquivo é o destino dos dados gerados pelo programa: nesse caso trata-se de um arquivo de saída de dados (*output*).

# Estrutura de Dados 1

## Arquivos

- Definição de Arquivo:
  - Coleção de bytes armazenados em dispositivo de armazenamento secundário.
    - Disco Rígido;
    - CD;
    - DVD;
    - Pendrive;
    - Disquete (será que existe ainda?);
    - etc.

Qualquer dispositivo onde os dados fiquem armazenados, ou salvos, mesmo depois de sua aplicação ser encerrada.

# Estrutura de Dados 1

## Arquivos

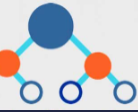


- Vantagens de se usar arquivos:
  - Armazenamento durável;
  - Permitem armazenar uma grande quantidade de informação;
  - Acesso concorrente aos dados.
- Cuidado: A extensão do arquivo não define o seu tipo.

A extensão de um arquivo serve somente para o sistema operacional vincular o programa mais indicado para abrir tal arquivo
- O que define um arquivo é a maneira como os dados estão organizados e as operações usadas por um programa para processar (ler e escrever), esse tipo de arquivo.

# Estrutura de Dados 1

## Arquivos

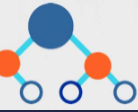


- Para que seja possível o processamento de um arquivo, a primeira operação a ser realizada é ligar uma variável do programa a esse arquivo;
- A esta operação, dá-se o nome de **Abertura de Arquivo**, que consiste em associar a variável do programa ao arquivo que se pretende processar, ou em outras palavras, representar internamente o nome físico do arquivo através de um nome lógico, que corresponde à variável do programa que irá representa-lo;
- Dessa forma evita-se estar permanentemente escrevendo o nome completo do arquivo sempre que for necessário se referir à ele

# Estrutura de Dados 1

## Arquivos

- Depois de aberto, é possível a realização de todas as operações pretendidas sobre o seu conteúdo – ler dados, escrever dados, etc. - o que corresponde, na realidade, ao processamento a se realizar sobre o arquivo;
- Depois de processado o arquivo, se este já não for mais necessário, a ligação que existe entre a variável do programa e o arquivo que esta representa, deve ser retirada, a este processo damos o nome de **Fechar o Arquivo**.
- As operações de abertura e fechamento traduzem-se em inglês por ***open*** e ***close***. No entanto, em “C” todas as operações de processamento de arquivos são precedidas por um “**f**”, de forma a indicar que se tratam de operações sobre arquivos .



# Estrutura de Dados 1

## Arquivos



- Para trabalharmos com arquivos utilizaremos a biblioteca `stdio.h`
- A Linguagem C utiliza um tipo especial de ponteiro para manipular arquivos, um ponteiro do tipo “FILE”, e sua forma geral é:

```
FILE *nome_ponteiro;
```

**FILE** é escrito em maiúsculas para reforçar a ideia de que não se trata de um tipo básico da própria linguagem, e trata-se de um tipo de objeto adequado para armazenar informações para controle de um fluxo de dados para arquivos.

- É este ponteiro que controla o fluxo de leitura e escrita dentro de um arquivo, e será a variável associada ao arquivo que será processado.
- Os membros da estrutura FILE contêm informações sobre o arquivo a ser usado, tais como: seu atual tamanho, a localização de seus buffers de dados, se o arquivo está sendo lido ou gravado, etc.

# Estrutura de Dados 1

## Arquivos

- Trecho da biblioteca `stdio.h`, que contém a estrutura do tipo `FILE`:

```
/*
 * The structure underlying the FILE type.
 *
 * Some believe that nobody in their right mind should make use of the
 * internals of this structure. Provided by Pedro A. Aranda Gutierrez
 * <paag@tid.es>.
 */
#ifndef _FILE_DEFINED
#define _FILE_DEFINED
typedef struct _iobuf
{
    char*    _ptr;
    int      _cnt;
    char*    _base;
    int      _flag;
    int      _file;
    int      _charbuf;
    int      _bufsiz;
    char*    _tmpfname;
} FILE;
#endif /* Not _FILE_DEFINED */
```

Próximo caractere de/para o *buffer*

Caracteres disponíveis no *buffer*

O *buffer*

O estado do fluxo (*stream*)

Mussum Ipsum, cacilds vidis litro abertis.

In elementis mé pra quem é amistosis

quis leo. Nullam volutpat risus nec leo

commodo, ut interdum diam laoreet.

Sed non consequat odio.

Mauris nec dolor in eros commodo tempor.

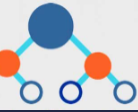




# Estrutura de Dados 1

## Arquivos

- A Linguagem C trabalha com apenas dois tipos de arquivos:
  - Arquivos texto: podem ser editados no bloco de notas;
  - Arquivos binários : não podem ser editados no bloco de notas.
- Arquivo Texto:
  - Os dados são gravados exatamente como seriam impressos na tela;
  - Os dados são gravados como caracteres de 8 bits utilizando a tabela ASCII, para isso,
  - existe uma etapa de “conversão” dos dados.



# Estrutura de Dados 1

## Arquivos



- Problemas com a conversão:
  - Os arquivos gerados são maiores;
  - Leitura e escrita são operações lentas.
- Exemplo: vamos considerar um número inteiro com 8 dígitos:

```
int n = 12345678; //32 bits na memória.
```

- Em um arquivo texto, cada dígito será convertido para seu caractere ASCII, ou seja, 8 bits por dígito.

```
12345678; //64 bits no arquivo.
```

# Estrutura de Dados 1

## Arquivos de Texto e Binários

- Arquivo Binário:
  - Os dados são gravados exatamente como estão organizados na memória do computador;
  - Não existe a etapa de conversão dos dados.
- Em consequência temos:
  - Arquivos geralmente menores;
  - E leituras e escritas mais rápidas.
- Utilizando o exemplo anterior, o número com 8 dígitos:

```
int n = 12345678; //32 bits na memória.
```

- Em um Arquivo binário, o conteúdo da memória será copiado diretamente para o arquivo, sem conversão:

```
12345678; //32 bits no arquivo de forma codificada.
```

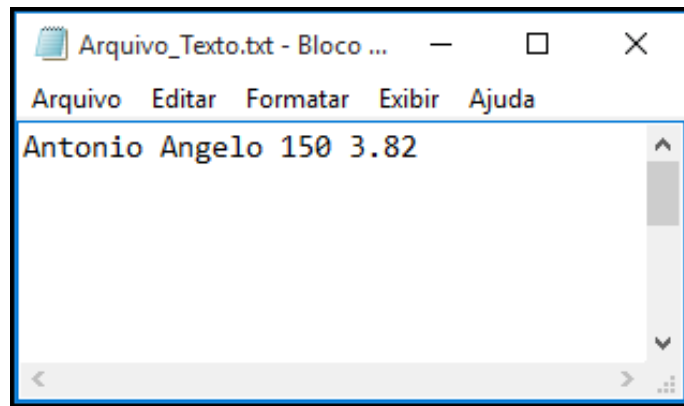


# Estrutura de Dados 1

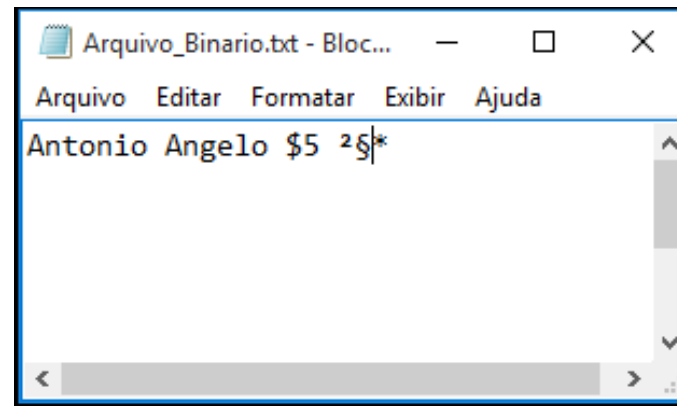
## Arquivos de Texto e Binários

- Para entender melhor a diferença entre esses dois arquivos, imagine os seguintes dados a serem gravados:

```
char nome[41] = "Antonio Angelo";  
int    i = 150;  
float  f = 3.82;
```



Arquivo Texto



Arquivo Binário



# Estrutura de Dados 1

## Abrindo e Fechando Arquivos

- Função `fopen()` permite abrir um arquivo em um determinado modo de leitura ou escrita, e sua forma geral é:

```
FILE *fopen(char *nome, char *modo);
```

- Exemplo:

```
typedef struct _iobuf
{
    char* _ptr;
    int _cnt;
    char* _base;
    int _flag;
    int _file;
    int _charbuf;
    int _bufsiz;
    char* _tmpfname;
} FILE;
```

```
FILE *f;
f = fopen("arquivo.txt", "w");
```

Mussum Ipsum, cacilds vidis litro abertis.

In elementis mé pra quem é amistosis quis leo. Nullam volutpat risus nec leo commodo, ut interdum diam laoreet.

Sed non consequat odio.

Mauris nec dolor in eros commodo tempor.



# Estrutura de Dados 1

## Abrindo e Fechando Arquivos

- Se a função `fopen()` não conseguir abrir o arquivo, ela irá retornar `NULL`, que é um valor especial e significa: **endereço de lugar nenhum**.

- Exemplo:

```
FILE *f;  
f = fopen("H:\arquivo.txt", "w");  
if(f == NULL){  
    printf("Erro na abertura!\n");  
    system("pause");  
    exit(1); //Aborta o programa.  
}
```

Se o arquivo não puder ser aberto, por qualquer razão, e a função devolver `NULL`, o programa não será abortado, como acontece em outras linguagens. É de responsabilidade do programador o tratamento a ser aplicado nos possíveis erros que possam aparecer quando se processa arquivos.

Se `fopen()`, conseguir abrir com sucesso um arquivo, ela cria em memória uma estrutura, do tipo `FILE`, que representa toda a informação necessária relativa ao arquivo, retornando o endereço em que essa estrutura foi criada em memória.

`NULL` é uma constante simbólica e está definida na biblioteca `stdlib.h`. Trata-se de um valor reservado que indica que aquele ponteiro aponta para uma região de memória inexistente. O valor da constante `NULL` é **ZERO**, na maioria dos computadores.



# Estrutura de Dados 1

## Abrindo e Fechando Arquivos

- Para o “nome” do arquivo, podemos usar o caminho:
  - Absoluto, onde temos o endereço completo desde a raiz do drive;
  - Relativo, onde este caminho é relativo à pasta onde está sendo executado o programa.
- Exemplo:

```
//caminho absoluto  
f = fopen("C:\\Projetos\\arquivo.txt", "w");  
//caminho relativo  
f = fopen("arquivo.txt", "w");  
f = fopen("../Novo\\arquivo2.txt", "w");
```

Sequência de escape  
para a barra invertida



# Estrutura de Dados 1

## Abrindo e Fechando Arquivos

- O modo de abertura determina que tipo de uso será feito do arquivo:

```
FILE *f;  
//leitura de Arquivo Texto  
f = fopen("arquivo.txt", "r");  
//escrita em Arquivo Texto  
f = fopen("arquivo.txt", "w");  
//Leitura de Arquivo Binário  
f = fopen("arquivo.txt", "rb");  
//Escrita de Arquivo Binário  
f = fopen("arquivo.txt", "wb");
```





# Estrutura de Dados 1

## Abrindo e Fechando Arquivos



► Modos de abertura possíveis:

Modo	Arquivo	Função
"r"	Texto	Leitura. Arquivo deve existir.
"w"	Texto	Escrita. Cria o Arquivo se não houver. Apaga o anterior se ele existir.
"a"	Texto	Escrita. Os dados serão adicionados no fim do Arquivo ("append").
"rb"	Binário	Leitura. Arquivo deve existir.
"wb"	Binário	Escrita. Cria o Arquivo se não houver. Apaga o anterior se ele existir.
"ab"	Binário	Escrita. Os dados serão adicionados no fim do Arquivo ("append").
"r+"	Texto	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
"w+"	Texto	Leitura/Escrita. Cria o Arquivo se não houver. Apaga o anterior se ele existir.
"a+"	Texto	Leitura/Escrita. Os dados serão adicionados no fim do Arquivo ("append").
"r+b"	Binário	Leitura/Escrita. O arquivo deve existir e pode ser modificado.
"w+b"	Binário	Leitura/Escrita. Cria o Arquivo se não houver. Apaga o anterior se ele existir.
"a+b"	Binário	Leitura/Escrita. Os dados serão adicionados no fim do Arquivo ("append").



## Abrindo e Fechando Arquivos

- Sempre que terminamos de usar um arquivo, devemos fechá-lo. Para isso, usamos a função `fclose()`. Sua forma geral é:

```
int fclose(FILE *f);
```

- `fclose()` retorna 0 (zero), no caso de sucesso no fechamento do arquivo.

```
//Exemplo:
FILE *f;
f = (fopen("C:\\arquivo.txt", "w"));
if(f == NULL){
    printf("Erro ao abrir arquivo!\n");
    system("pause");
    exit(1); //aborta o programa
}
/*Fechar o arquivo garante que todos os
dados foram gravados*/
fclose(f);
```

Quando se fecha um arquivo, os dados que possam existir no **buffer** de memória, são escritos fisicamente no disco, só então é desfeita a ligação entre o arquivo físico e a variável que o representa no programa.



## Gravando um caractere por vez em Arquivos – função fputc()

- Para escrever um caractere em um arquivo usamos a função `fputc()`. Sua forma geral é:

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

- Ela retorna:
  - Em caso de erro: a constante EOF;
  - Em caso de sucesso: o próprio caractere.

**EOF (*End-Of-File*):** constante simbólica que serve como indicador de final de arquivo. Normalmente o valor “-1”. Como os valores válidos da tabela ASCII podem estar dentro do conteúdo do arquivo, é devolvido então um valor fora da faixa de valores válidos para a tabela, que são de 0 à 255.

Mussum Ipsum, cacilds vidis litro abertis.

In elementis mé pra quem é amistosis

quis leo. Nullam volutpat risus nec leo

commodo, ut interdum diam laoreet.

Sed non consequat odio.

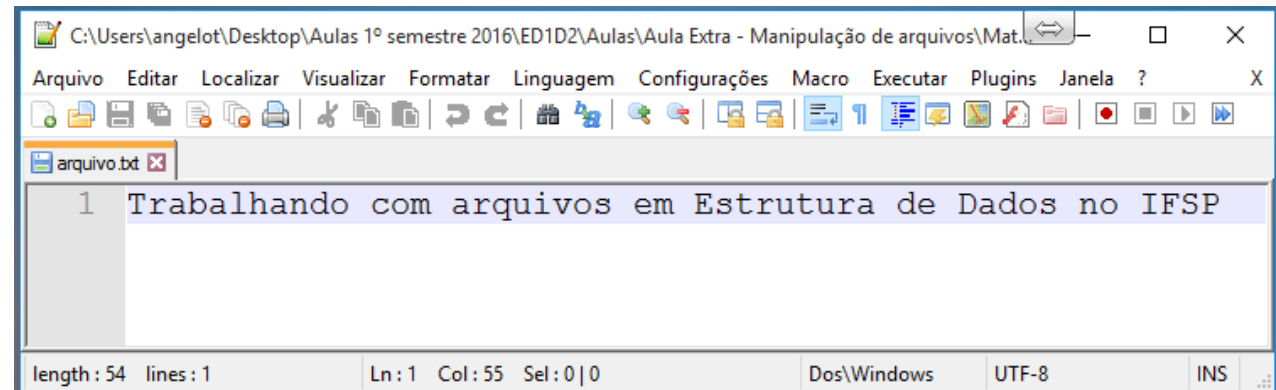
Mauris nec dolor in eros commodo tempor.

# Estrutura de Dados 1

## Gravando um caractere por vez com fputc()

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

int main(){
    FILE *f;
    f = fopen("arquivo.txt", "w");
    if(f == NULL){
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char texto[60] = "Trabalhando com arquivos em Estrutura de Dados no IFSP";
    int i;
    //grava a string, caractere a caractere
    for(i = 0; i < strlen(texto); i++){
        fputc(texto[i], f);
    }
    fclose(f);
    system("pause");
    return 0;
}
```

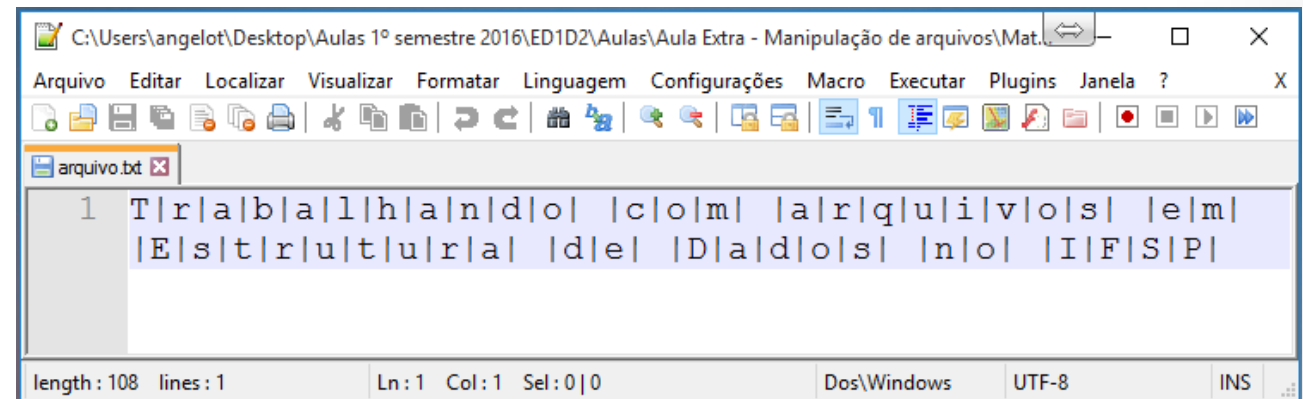


# Estrutura de Dados 1

## Gravando um caractere por vez com fputc()

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

int main(){
    FILE *f;
    f = fopen("arquivo.txt", "w");
    if(f == NULL){
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char texto[60] = "Trabalhando com arquivos em Estrutura de Dados no IFSP";
    int i;
    //grava a string, caractere a caractere
    for(i = 0; i < strlen(texto); i++){
        fputc(texto[i], f);
        fputc('|', f);
    }
    fclose(f);
    system("pause");
    return 0;
}
```



# Estrutura de Dados 1

## Lendo um caractere por vez com `fgetc()`



- Para ler um caractere de um arquivo, utilizamos a função `fgetc()`. Sua forma geral é:

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

- Ela retorna:
  - Em caso de erro: a constante `EOF`;
  - Em caso de sucesso: o caractere lido do arquivo.

# Estrutura de Dados 1

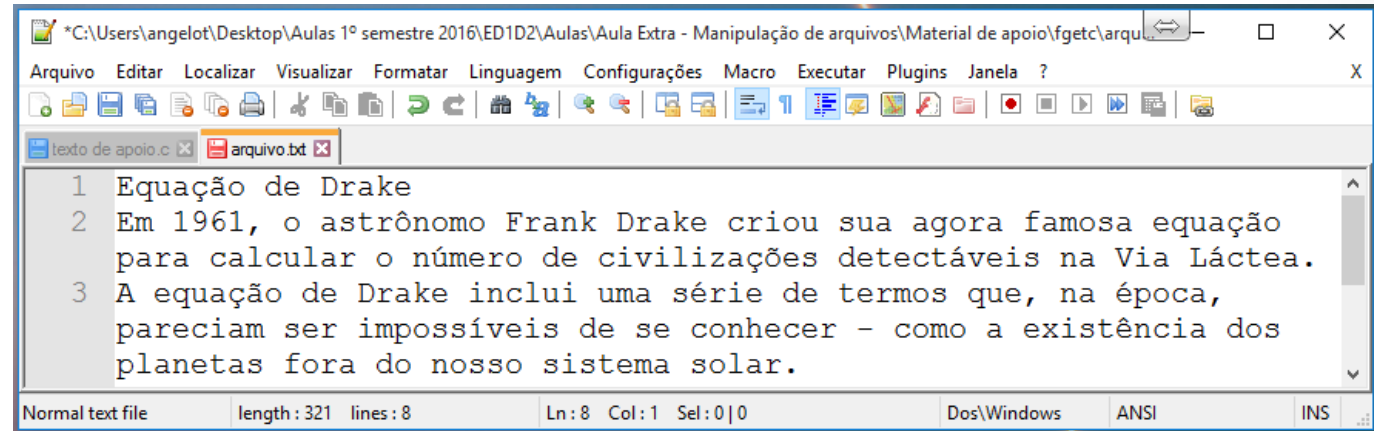


## Lendo um caractere por vez com fgetc()

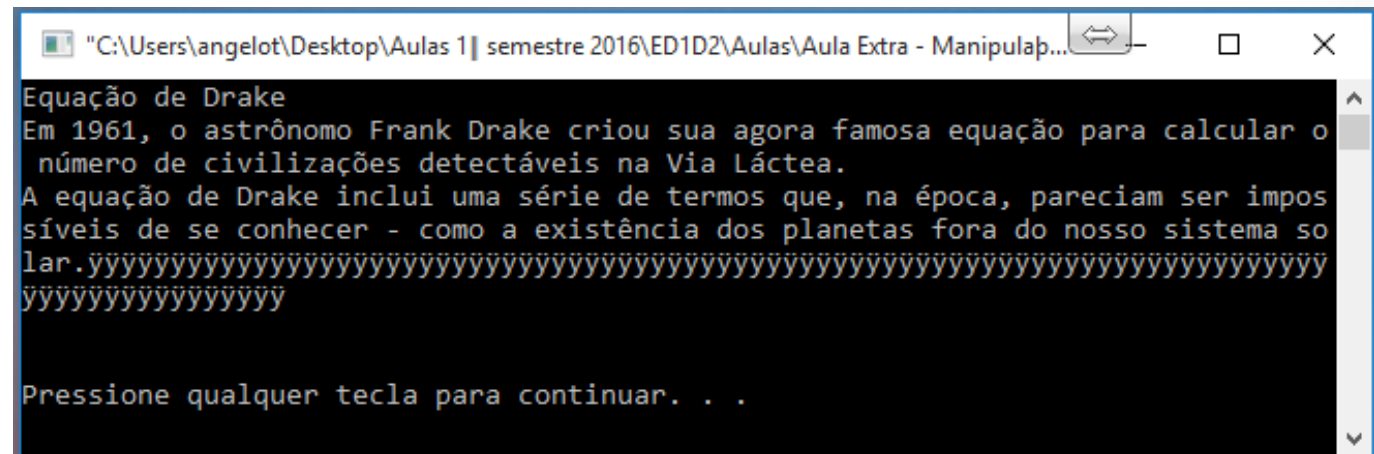
```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <locale.h>
```

```
int main(){
    setlocale(LC_ALL, "");
    FILE *f;
    f = fopen("arquivo.txt", "r");
    if(f == NULL){
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char c;
    int i;
    //lê o arquivo um caractere por vez
    for(i = 0; i < 400; i++){
        c = fgetc(f);
        printf("%c", c);
    }
    printf("\n\n\n");
    fclose(f);
    system("pause");
    return 0;
}
```

Modo  
leitura!



The screenshot shows a text editor window titled "C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\fgetc\arquivo.txt". The editor contains three lines of text: "1 Equação de Drake", "2 Em 1961, o astrônomo Frank Drake criou sua agora famosa equação para calcular o número de civilizações detectáveis na Via Láctea.", and "3 A equação de Drake inclui uma série de termos que, na época, pareciam ser impossíveis de se conhecer - como a existência dos planetas fora do nosso sistema solar." The status bar at the bottom indicates "Normal text file", "length: 321 lines: 8", "Ln: 8 Col: 1 Sel: 0|0", "Dos\Windows", "ANSI", and "INS".



The screenshot shows a command prompt window titled "C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\fgetc\arquivo.txt". The output of the program is displayed: "Equação de Drake", "Em 1961, o astrônomo Frank Drake criou sua agora famosa equação para calcular o número de civilizações detectáveis na Via Láctea.", and "A equação de Drake inclui uma série de termos que, na época, pareciam ser impossíveis de se conhecer - como a existência dos planetas fora do nosso sistema solar." followed by a large block of "y" characters. The prompt "Pressione qualquer tecla para continuar. . ." is visible at the bottom.



## Utilizando a constante EOF para controle de leitura

- No exemplo anterior, a função `fgetc()`, foi utilizada para ler um bloco de 400 caracteres, no entanto o arquivo possui menos caracteres (321). Todos os caracteres extras impressos pelo programa são “sujeira” e foram gerados pelo compilador, já que não existiam no arquivo original.
- Como fazer para ler corretamente todo o conteúdo do arquivo?
- Para resolver este problema temos que utilizar a constante EOF (*End Of File*), que devolve um caractere especial indicativo de “fim de arquivo”. Quando o final do arquivo é atingido, a função `fgetc()` devolve esta constante, que geralmente é -1 (isso depende do compilador).

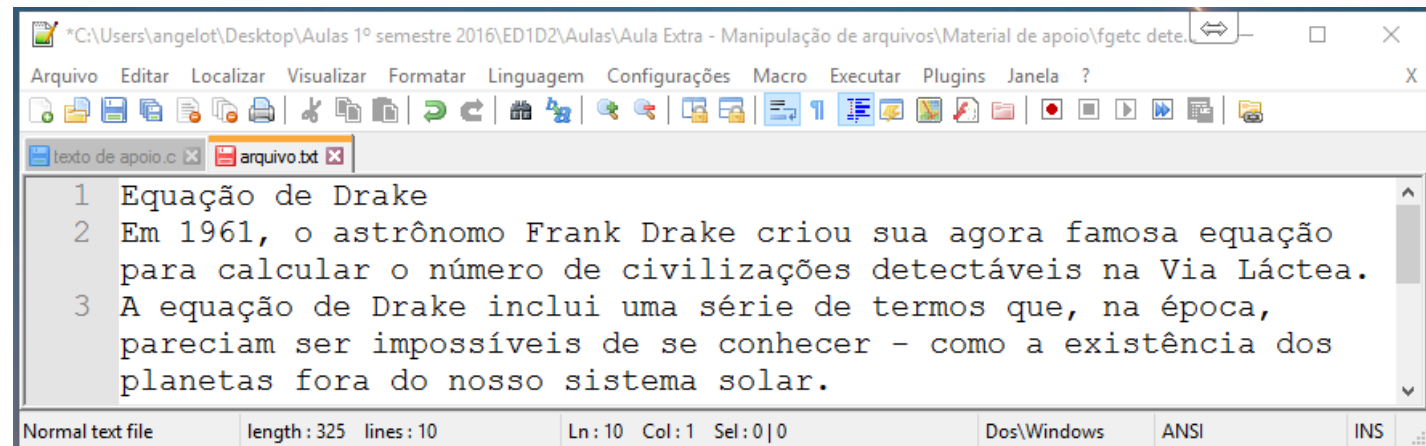


# Estrutura de Dados 1

## Utilizando a constante EOF para controle de leitura

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

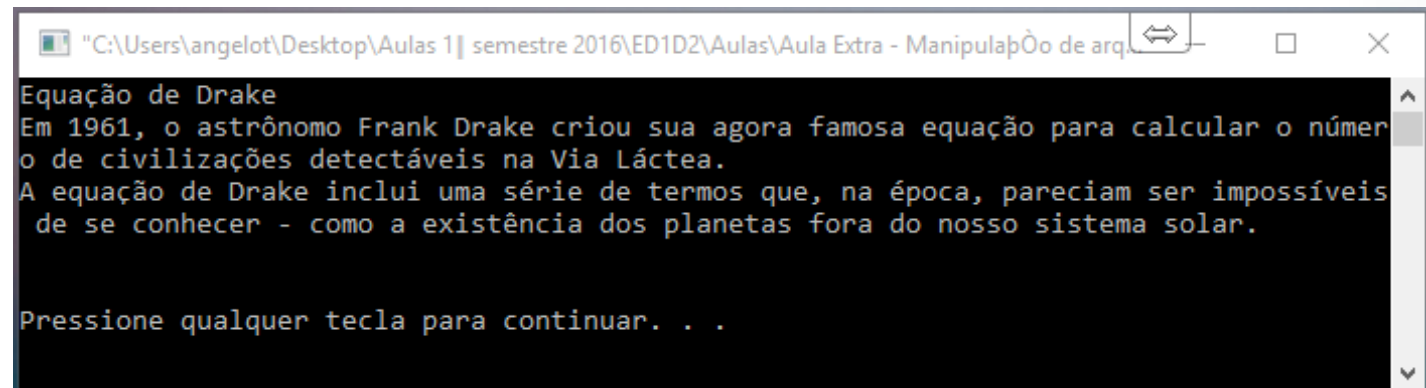
int main(){
    setlocale(LC_ALL, "");
    FILE *f;
    f = fopen("arquivo.txt", "r");
    if(f == NULL){
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char c = fgetc(f);
    while(c != EOF){
        printf("%c", c);
        c = fgetc(f);
    }
    printf("\n\n\n");
    fclose(f);
    system("pause");
    return 0;
}
```



A screenshot of a text editor window titled "C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\fgetc dete...". The editor shows the content of a file named "arquivo.txt". The text is as follows:

```
1 Equação de Drake
2 Em 1961, o astrônomo Frank Drake criou sua agora famosa equação
  para calcular o número de civilizações detectáveis na Via Láctea.
3 A equação de Drake inclui uma série de termos que, na época,
  pareciam ser impossíveis de se conhecer - como a existência dos
  planetas fora do nosso sistema solar.
```

The status bar at the bottom indicates "Normal text file", "length: 325 lines: 10", "Ln: 10 Col: 1 Sel: 0|0", "Dos\Windows", "ANSI", and "INS".



A screenshot of a terminal window titled "C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\fgetc dete...". The terminal shows the output of the program, which is the content of "arquivo.txt" displayed line by line:

```
Equação de Drake
Em 1961, o astrônomo Frank Drake criou sua agora famosa equação para calcular o número
o de civilizações detectáveis na Via Láctea.
A equação de Drake inclui uma série de termos que, na época, pareciam ser impossíveis
de se conhecer - como a existência dos planetas fora do nosso sistema solar.
```

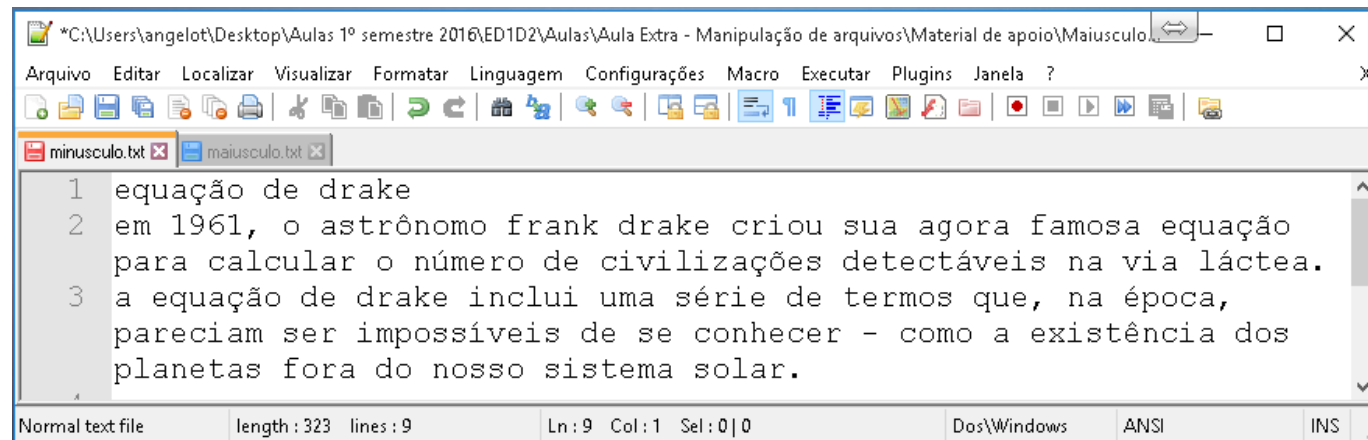
Below the text, the prompt "Pressione qualquer tecla para continuar. . ." is visible.



# Estrutura de Dados 1

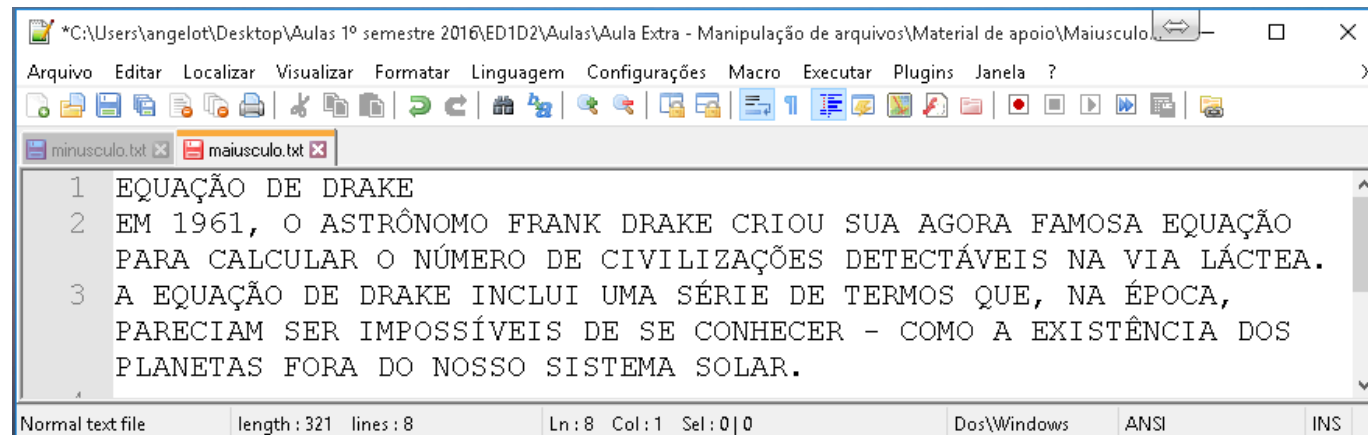
## Exemplo: manipulando conteúdos de arquivos

- Como exemplo, utilizaremos o arquivo texto anterior onde todos os caracteres foram alterados para minúsculo. Em nosso programa agora, criaremos uma versão deste arquivo com todas as letras em maiúsculo.



The screenshot shows a text editor window titled "\*C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\Maiusculo...". The menu bar includes Arquivo, Editar, Localizar, Visualizar, Formatar, Linguagem, Configurações, Macro, Executar, Plugins, Janela, and ?. The toolbar contains various icons for file operations. The text area shows three lines of lowercase text: "1 equação de drake", "2 em 1961, o astrônomo frank drake criou sua agora famosa equação para calcular o número de civilizações detectáveis na via láctea.", and "3 a equação de drake inclui uma série de termos que, na época, pareciam ser impossíveis de se conhecer - como a existência dos planetas fora do nosso sistema solar." The status bar at the bottom indicates "Normal text file", "length : 323 lines : 9", "Ln : 9 Col : 1 Sel : 0 | 0", "Dos\Windows", "ANSI", and "INS".

```
1 equação de drake
2 em 1961, o astrônomo frank drake criou sua agora famosa equação
  para calcular o número de civilizações detectáveis na via láctea.
3 a equação de drake inclui uma série de termos que, na época,
  pareciam ser impossíveis de se conhecer - como a existência dos
  planetas fora do nosso sistema solar.
```



The screenshot shows the same text editor window after converting the text to uppercase. The text area now shows: "1 EQUAÇÃO DE DRAKE", "2 EM 1961, O ASTRÔNOMO FRANK DRAKE CRIOU SUA AGORA FAMOSA EQUAÇÃO PARA CALCULAR O NÚMERO DE CIVILIZAÇÕES DETECTÁVEIS NA VIA LÁCTEA.", and "3 A EQUAÇÃO DE DRAKE INCLUI UMA SÉRIE DE TERMOS QUE, NA ÉPOCA, PARECIAM SER IMPOSSÍVEIS DE SE CONHECER - COMO A EXISTÊNCIA DOS PLANETAS FORA DO NOSSO SISTEMA SOLAR." The status bar at the bottom indicates "Normal text file", "length : 321 lines : 8", "Ln : 8 Col : 1 Sel : 0 | 0", "Dos\Windows", "ANSI", and "INS".

```
1 EQUAÇÃO DE DRAKE
2 EM 1961, O ASTRÔNOMO FRANK DRAKE CRIOU SUA AGORA FAMOSA EQUAÇÃO
  PARA CALCULAR O NÚMERO DE CIVILIZAÇÕES DETECTÁVEIS NA VIA LÁCTEA.
3 A EQUAÇÃO DE DRAKE INCLUI UMA SÉRIE DE TERMOS QUE, NA ÉPOCA,
  PARECIAM SER IMPOSSÍVEIS DE SE CONHECER - COMO A EXISTÊNCIA DOS
  PLANETAS FORA DO NOSSO SISTEMA SOLAR.
```



# Estrutura de Dados 1

## Exemplo: manipulando conteúdos de arquivos

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

```
int main() {
    setlocale(LC_ALL, "");
    FILE *f1, *f2;
    f1 = fopen("minusculo.txt", "r");
    f2 = fopen("maiusculo.txt", "w");
    if (f1 == NULL || f2 == NULL) {
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char c = fgetc(f1);
    while (c != EOF) {
        fputc(toupper(c), f2);
        c = fgetc(f1);
    }
}
```

```
fclose(f1);
fclose(f2);
f2 = fopen("maiusculo.txt", "r");
char b = fgetc(f2);
while (b != EOF) {
    printf("%c", b);
    b = fgetc(f2);
}
printf("\n\n\n");
fclose(f2);
system("pause");
return 0;
}
```



# Estrutura de Dados 1

## Atividade 1



- Utilizando exemplos anteriores, codifique um programa que receba via teclado um pequeno texto e em seguida salve-o gerando um arquivo chamado “arq1.txt”. Visualize-o com um editor de textos (bloco de notas).
- Em seguida baseado no exemplo anterior, modifique seu programa para que depois de recolhido o pequeno texto, todos os seus caracteres sejam modificados para letras maiúsculas e seja salvo com o nome de “arq2.txt”.
- Imprima os dois arquivos em tela.
- Devem ser entregues como atividade 1, os seguintes arquivos (compactados ou “zipados”):
  - Seu programa, somente o código fonte “.c”;
  - Os dois arquivos de texto gerados, “arq1.txt” e “arq2.txt”.

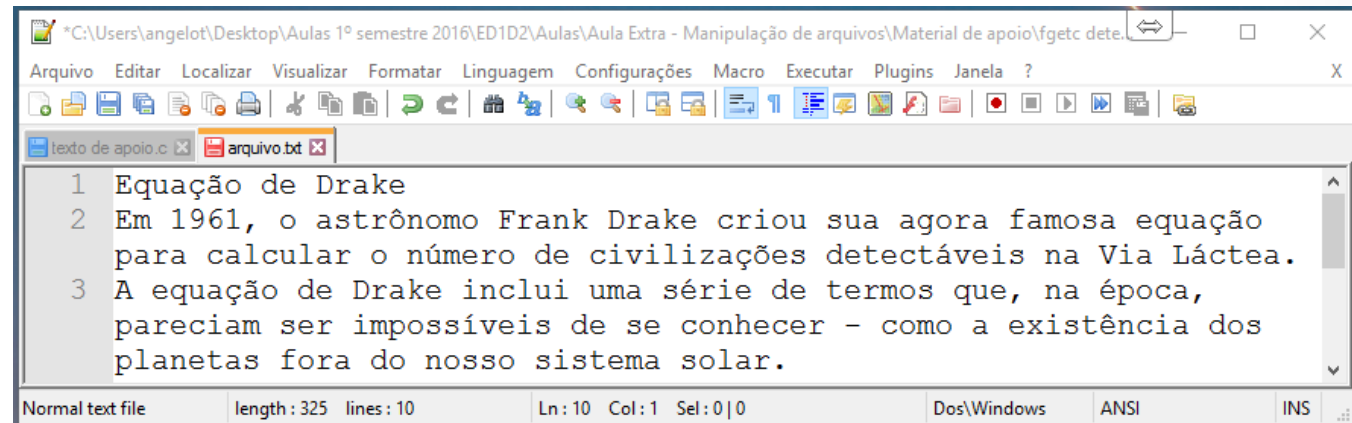
# Estrutura de Dados 1

## Utilizando a função `fgetc()`, para indicar o fim de um arquivo

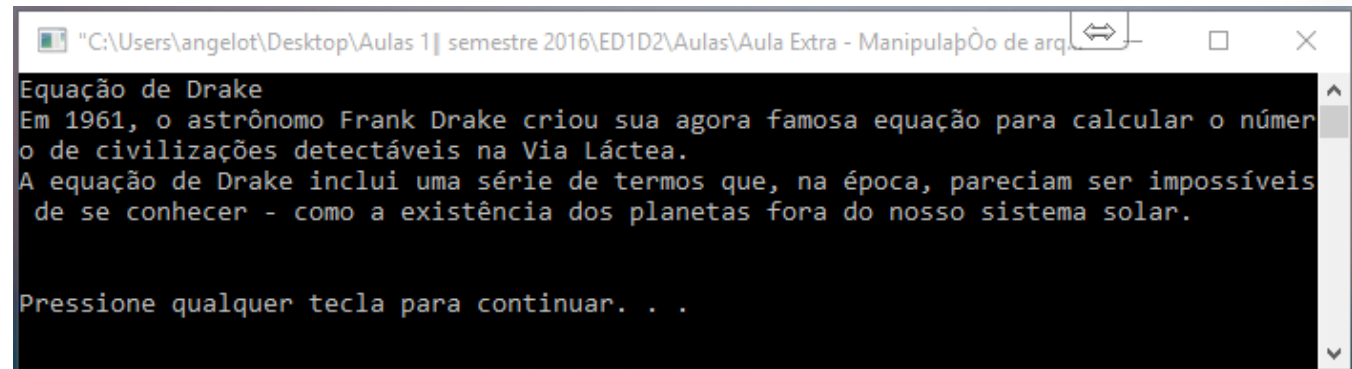
- No programa exemplo anterior, utilizamos a constante EOF (*End Of File*), para indicar o fim de um arquivo como abaixo:

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

int main() {
    setlocale(LC_ALL, "");
    FILE *f;
    f = fopen("arquivo.txt", "r");
    if(f == NULL){
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char c = fgetc(f);
    while(c != EOF){
        printf("%c", c);
        c = fgetc(f);
    }
    printf("\n\n\n");
    fclose(f);
    system("pause");
    return 0;
}
```



A screenshot of a text editor window titled "C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\fgetc dete...". The editor shows the content of "arquivo.txt" with three lines of text: "1 Equação de Drake", "2 Em 1961, o astrônomo Frank Drake criou sua agora famosa equação para calcular o número de civilizações detectáveis na Via Láctea.", and "3 A equação de Drake inclui uma série de termos que, na época, pareciam ser impossíveis de se conhecer - como a existência dos planetas fora do nosso sistema solar." The status bar at the bottom indicates "Normal text file", "length: 325 lines: 10", and "Ln: 10 Col: 1 Sel: 0|0".



A screenshot of a command prompt window titled "C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\fgetc dete...". The prompt displays the output of the program, which is the content of "arquivo.txt" printed line by line. The output is: "Equação de Drake", "Em 1961, o astrônomo Frank Drake criou sua agora famosa equação para calcular o número de civilizações detectáveis na Via Láctea.", and "A equação de Drake inclui uma série de termos que, na época, pareciam ser impossíveis de se conhecer - como a existência dos planetas fora do nosso sistema solar." The prompt ends with "Pressione qualquer tecla para continuar. . .".



# Estrutura de Dados 1



## Utilizando a função `feof()`, para indicar o fim de um arquivo

- O problema na utilização desta abordagem para o fim de um arquivo, surge quando é necessária a manipulação arquivos binários. Neste caso, um valor inteiro igual ao valor da constante EOF (que é -1), pode ser encontrado no conteúdo do arquivo, e assim lido e interpretado, erroneamente como final do arquivo em questão, quando na realidade não é.
- Para evitar este tipo de problema, a Linguagem C dispõe de uma função específica para a detecção do final do arquivo, a função `feof()`. Sua forma geral é:

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

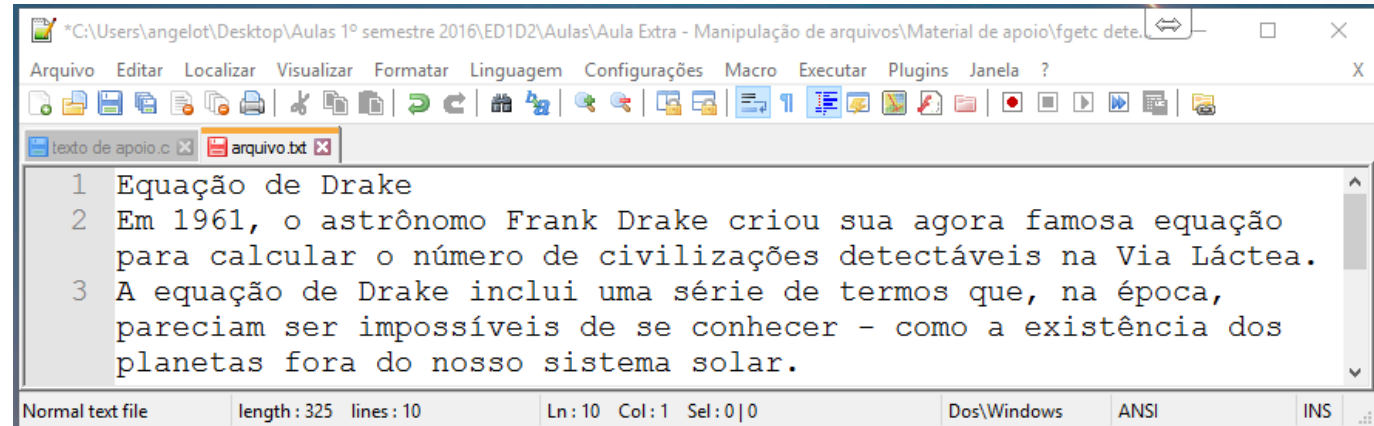
- Essa função retorna um valor inteiro igual a 0 (zero), se ainda não tiver atingido o final do arquivo.

# Estrutura de Dados 1

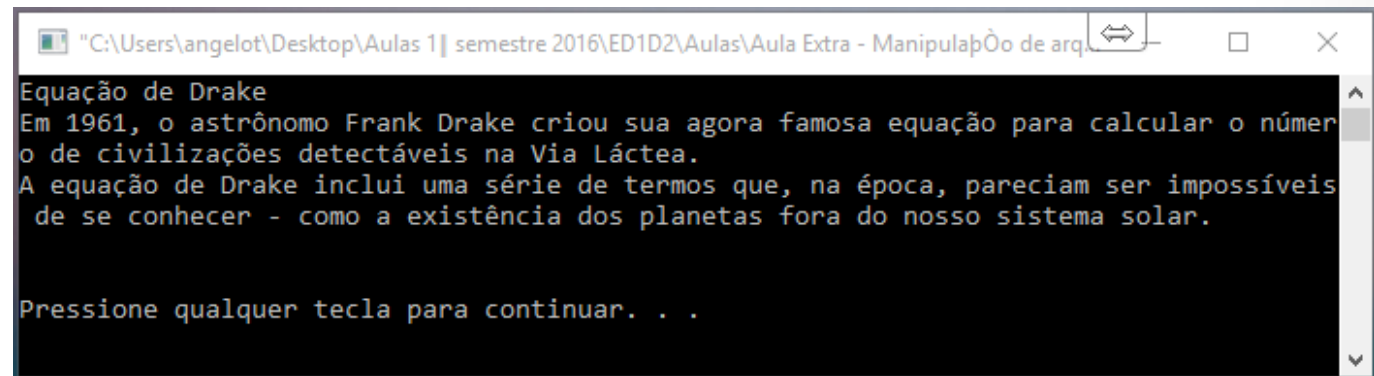
Utilizando a função `feof()`, para indicar o fim de um arquivo

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

int main() {
    setlocale(LC_ALL, "");
    FILE *f;
    f = fopen("arquivo.txt", "r");
    if (f == NULL) {
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char c = fgetc(f);
    while (!feof(f)) {
        printf("%c", c);
        c = fgetc(f);
    }
    printf("\n\n\n");
    fclose(f);
    system("pause");
    return 0;
}
```

A screenshot of a text editor window titled "C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\fgetc dete...". The editor shows the content of a file named "arquivo.txt". The text is as follows:

```
1 Equação de Drake
2 Em 1961, o astrônomo Frank Drake criou sua agora famosa equação
  para calcular o número de civilizações detectáveis na Via Láctea.
3 A equação de Drake inclui uma série de termos que, na época,
  pareciam ser impossíveis de se conhecer - como a existência dos
  planetas fora do nosso sistema solar.
```

The status bar at the bottom indicates "Normal text file", "length: 325 lines: 10", "Ln: 10 Col: 1 Sel: 0|0", "Dos\Windows", "ANSI", and "INS".A screenshot of a command prompt window titled "C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\fgetc dete...". The output of the program is displayed as follows:

```
Equação de Drake
Em 1961, o astrônomo Frank Drake criou sua agora famosa equação para calcular o número
o de civilizações detectáveis na Via Láctea.
A equação de Drake inclui uma série de termos que, na época, pareciam ser impossíveis
de se conhecer - como a existência dos planetas fora do nosso sistema solar.

Pressione qualquer tecla para continuar. . .
```





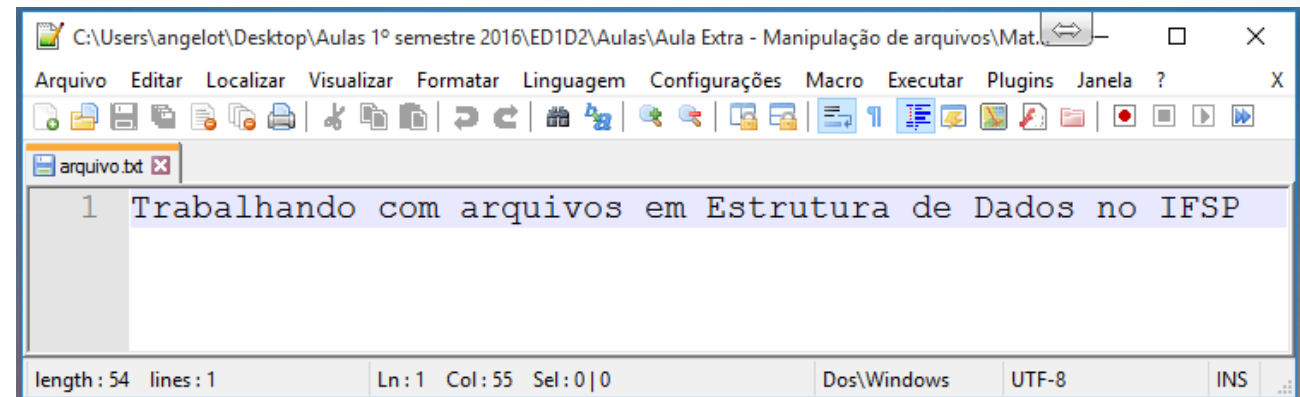
# Estrutura de Dados 1

## Gravando *strings* inteiras com `fputs()`

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

int main() {
    FILE *f;
    f = fopen("arquivo.txt", "w");
    if(f == NULL) {
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char texto[60] = "Trabalhando com arquivos em Estrutura de Dados no IFSP";
    int i;
    //grava a string, caractere a caractere
    for(i = 0; i < strlen(texto); i++) {
        fputc(texto[i], f);
    }
    fclose(f);
    system("pause");
    return 0;
}
```

Lendo um texto no exemplo anterior,  
caractere a caractere:  
Não existe uma maneira mais fácil de  
escrever *strings*?





# Estrutura de Dados 1

## Gravando *strings* inteiras com `fputs()`

- Para se escrever uma *string* em um arquivo utilizamos a função `fputs()`, e sua forma geral é:

```
int fputs(char *str, FILE *fp);
```

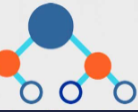
`fputs()` grava toda a *string* até que encontre o caractere de controle `"\0"`.

- Ela retorna:
  - Em caso de erro, a constante `EOF`;
  - Em caso de sucesso, um valor diferente de `0` (zero).



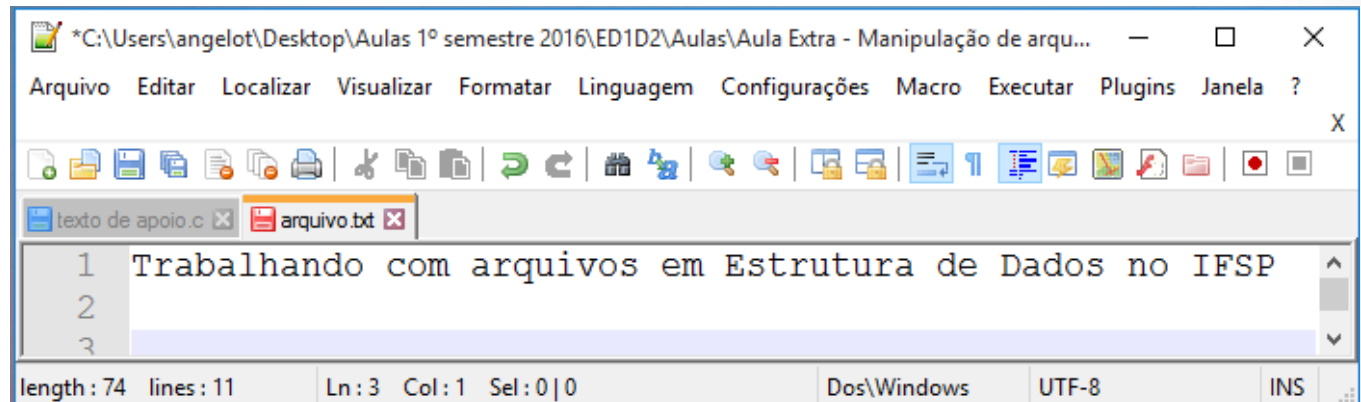
# Estrutura de Dados 1

## Gravando *strings* inteiras com fputs()



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

int main(){
    FILE *f;
    f = fopen("arquivo.txt", "w");
    if(f == NULL){
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char texto[60] = "Trabalhando com arquivos em Estrutura de Dados no IFSP";
    int i;
    //grava toda a string de uma só vez
    int retorno = fputs(texto, f);
    if(retorno == EOF){
        printf("Erro na gravação\n");
    }
    fclose(f);
    system("pause");
    return 0;
}
```



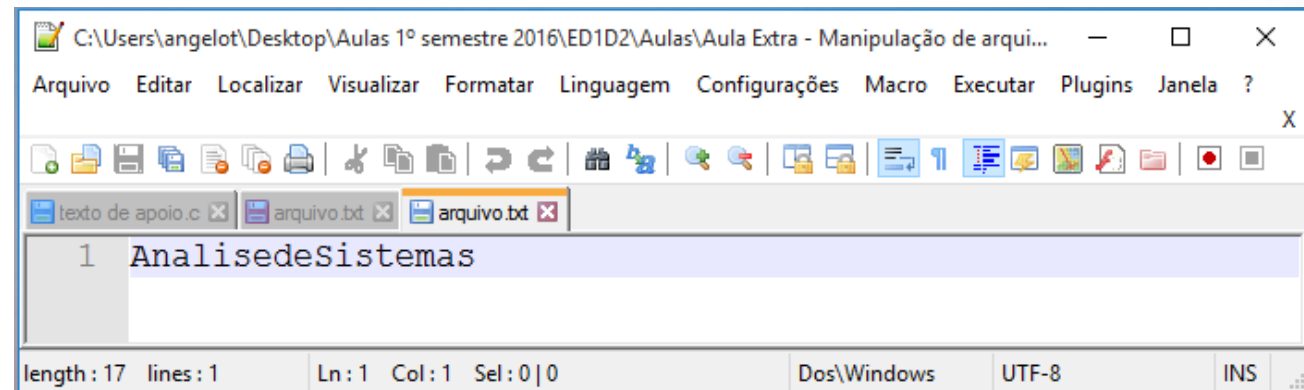
# Estrutura de Dados 1

## Gravando strings inteiras com fputs()

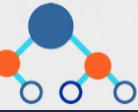
- A função `fputs()`, não coloca o caractere de nova linha “\n”, nem qualquer outro tipo de caractere, no final da *string* escrita, exemplo:

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

int main() {
    FILE *f;
    f = fopen("arquivo.txt", "w");
    if (f == NULL) {
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    fputs("Analise", f);
    fputs("de", f);
    fputs("Sistemas", f);
    fclose(f);
    system("pause");
    return 0;
}
```



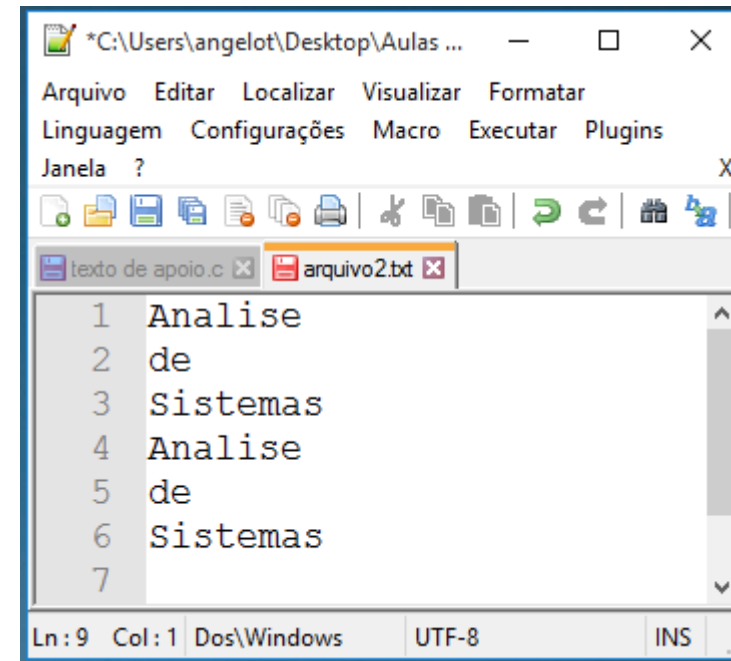
# Estrutura de Dados 1



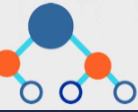
## Gravando *strings* inteiras com `fputs()`

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

int main() {
    FILE *f;
    f = fopen("arquivo2.txt", "w");
    if(f == NULL) {
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    fputs("Analise\n", f);
    fputs("de\n", f);
    fputs("Sistemas\n", f);
    //ou
    fputs("Analise", f);
    fputc('\n', f);
    fputs("de", f);
    fputc('\n', f);
    fputs("Sistemas", f);
    fclose(f);
    system("pause");
    return 0;
}
```



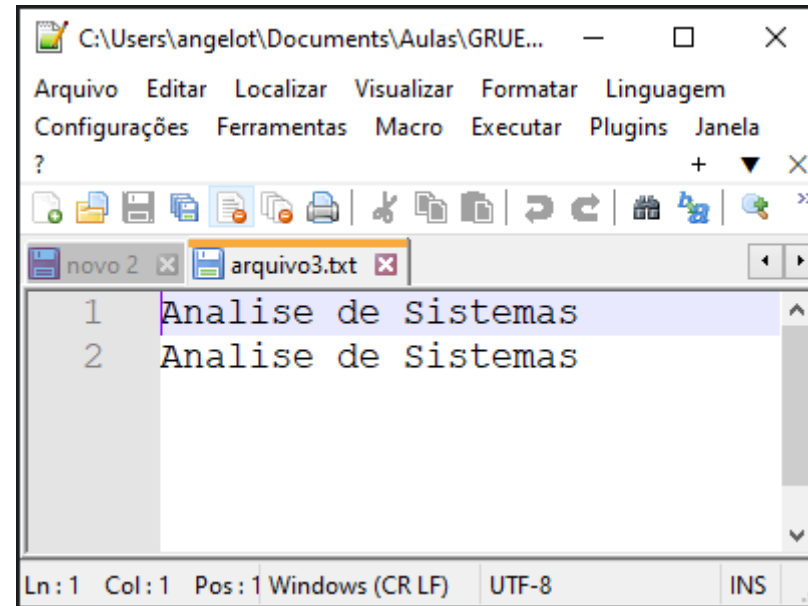
# Estrutura de Dados 1



## Gravando *strings* inteiras com `fputs()`

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

int main(){
    FILE *f;
    f = fopen("arquivo3.txt", "w");
    if(f == NULL){
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    fputs("Analise ", f);
    fputs("de ", f);
    fputs("Sistemas", f);
    //ou
    fputc('\n', f);
    fputs("Analise", f);
    fputc(' ', f);
    fputs("de", f);
    fputc(' ', f);
    fputs("Sistemas", f);
    fclose(f);
    system("pause");
    return 0;
}
```



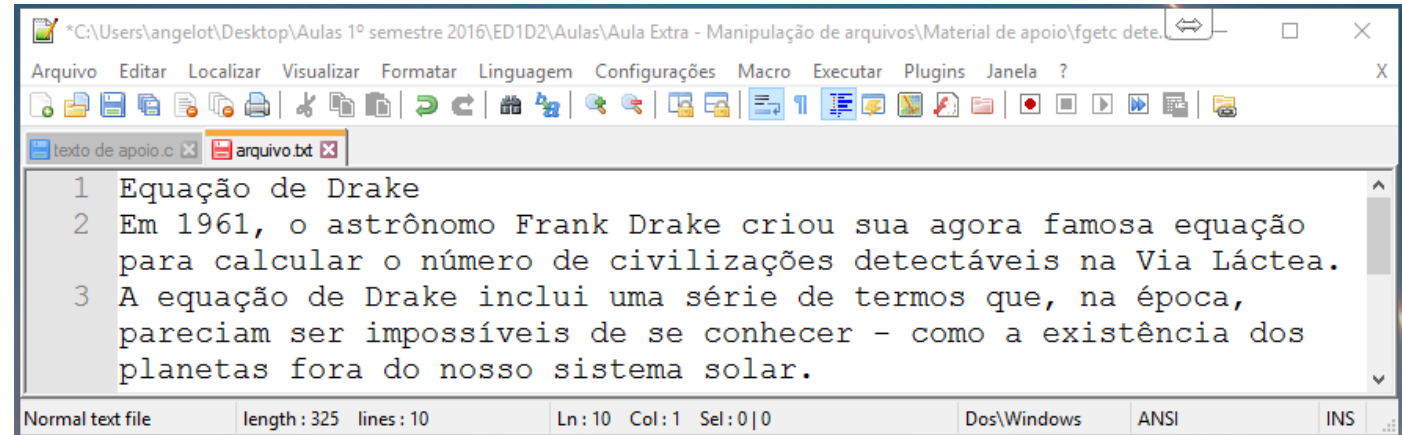
# Estrutura de Dados 1

## Lendo *strings* inteiras com `fgets()`

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

```
int main() {
    setlocale(LC_ALL, "");
    FILE *f;
    f = fopen("arquivo.txt", "r");
    if(f == NULL) {
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char c = fgetc(f);
    while(c != EOF) {
        printf("%c", c);
        c = fgetc(f);
    }
    printf("\n\n\n");
    fclose(f);
    system("pause");
    return 0;
}
```

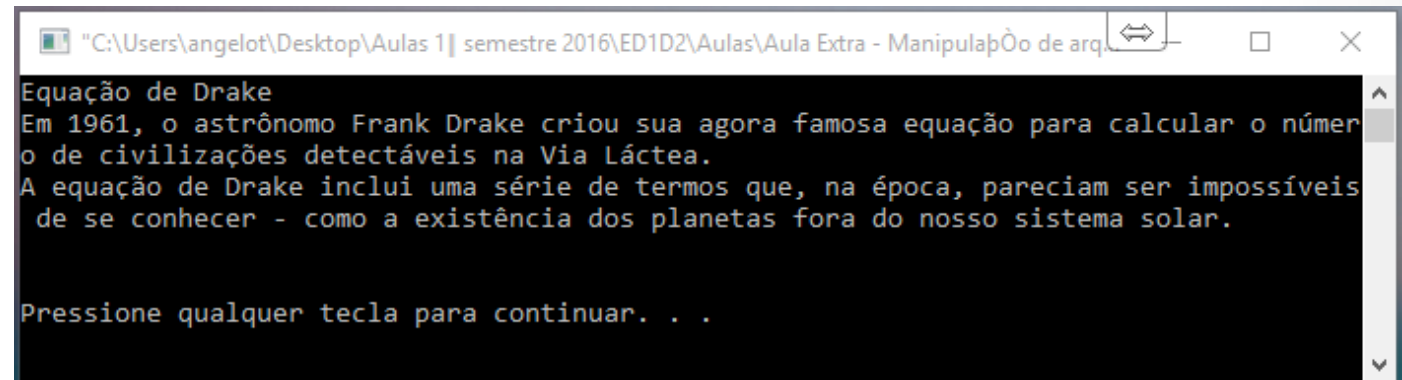
Até aqui manipulávamos  
caractere a caractere...



A screenshot of a text editor window titled "C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\fgets dete...". The editor shows the content of a file named "arquivo.txt". The text is as follows:

```
1 Equação de Drake
2 Em 1961, o astrônomo Frank Drake criou sua agora famosa equação
  para calcular o número de civilizações detectáveis na Via Láctea.
3 A equação de Drake inclui uma série de termos que, na época,
  pareciam ser impossíveis de se conhecer - como a existência dos
  planetas fora do nosso sistema solar.
```

The status bar at the bottom indicates "Normal text file", "length: 325", "lines: 10", "Ln: 10", "Col: 1", "Sel: 0|0", "Dos/Windows", "ANSI", and "INS".



A screenshot of a command prompt window titled "C:\Users\angelot\Desktop\Aulas 1º semestre 2016\ED1D2\Aulas\Aula Extra - Manipulação de arquivos\Material de apoio\fgets dete...". The output of the program is displayed as follows:

```
Equação de Drake
Em 1961, o astrônomo Frank Drake criou sua agora famosa equação para calcular o número
o de civilizações detectáveis na Via Láctea.
A equação de Drake inclui uma série de termos que, na época, pareciam ser impossíveis
de se conhecer - como a existência dos planetas fora do nosso sistema solar.

Pressione qualquer tecla para continuar. . .
```



# Estrutura de Dados 1

## Lendo *strings* inteiras com `fgets()`



- Para se ler uma *string* de um arquivo, utilizamos a função `fgets()`, e sua forma geral é:

```
char* fgets(char *str, int tamanho, FILE *fp);
```

Todo o conteúdo lido, será armazenado nesta *string* que foi passada com argumento

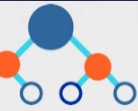
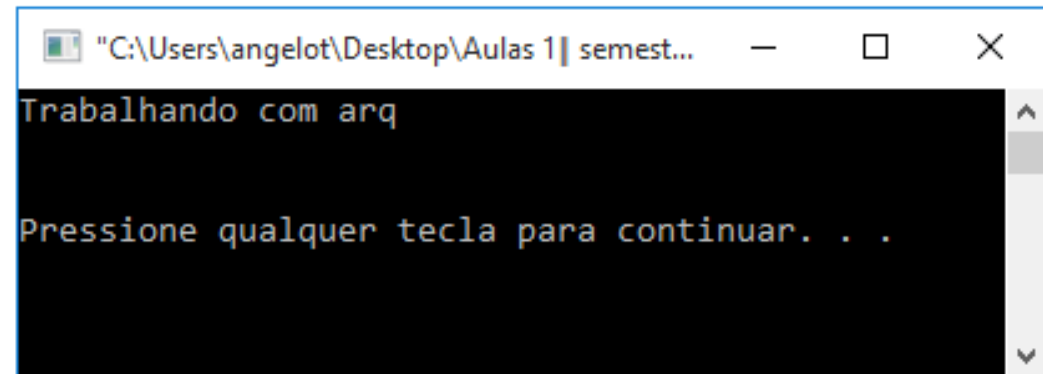
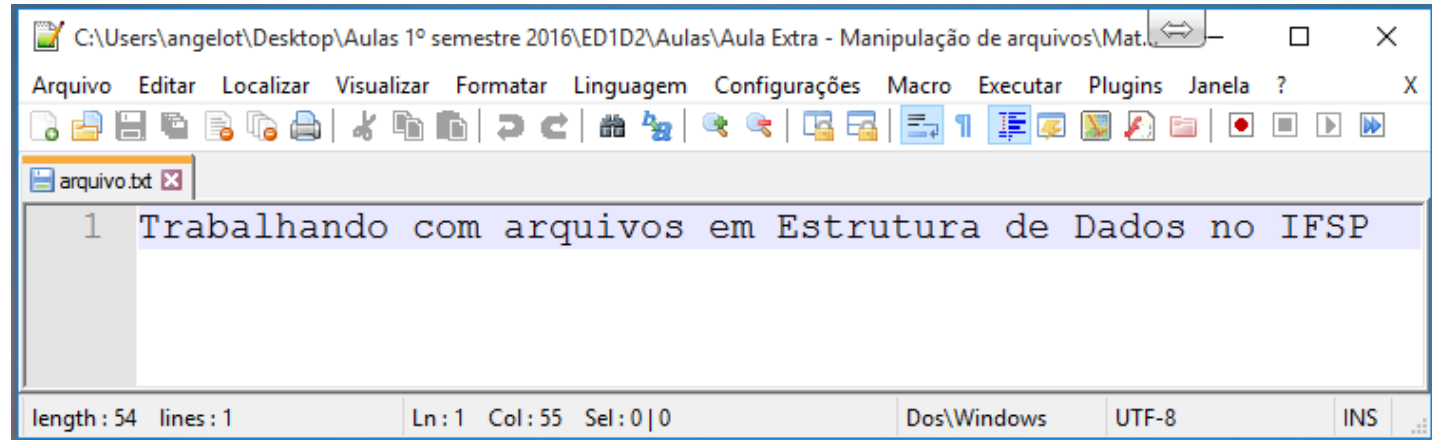
- Ela retorna:
  - Em caso de erro, `NULL`;
  - Em caso de sucesso, um ponteiro para o primeiro caractere da *string* (`str`).

# Estrutura de Dados 1

## Lendo *strings* inteiras com `fgets()`

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

int main() {
    setlocale(LC_ALL, "");
    char str[30];
    FILE *f;
    f = fopen("arquivo.txt", "r");
    if (f == NULL) {
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }
    char *resultado = fgets(str, 20, f);
    if (resultado == NULL) {
        printf("Erro na leitura\n");
    } else {
        printf("%s", str);
    }
    printf("\n\n\n");
    fclose(f);
    system("pause");
    return 0;
}
```





# Estrutura de Dados 1

## Lendo *strings* inteiras com `fgets()`



- A função `fgets()`, lê uma *string* até encontrar um caractere de nova linha, o “\n”, ou tamanho -1 caracteres, o EOF.

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

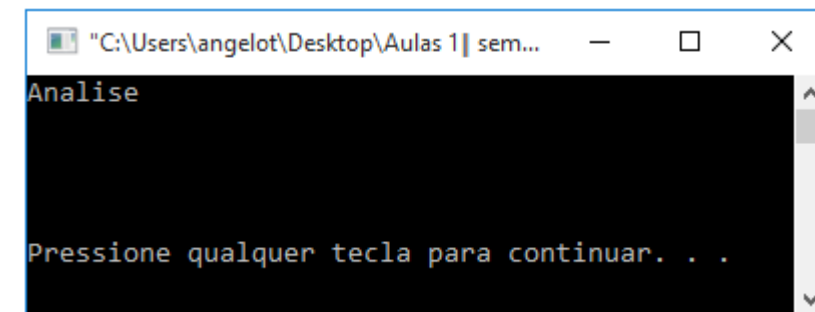
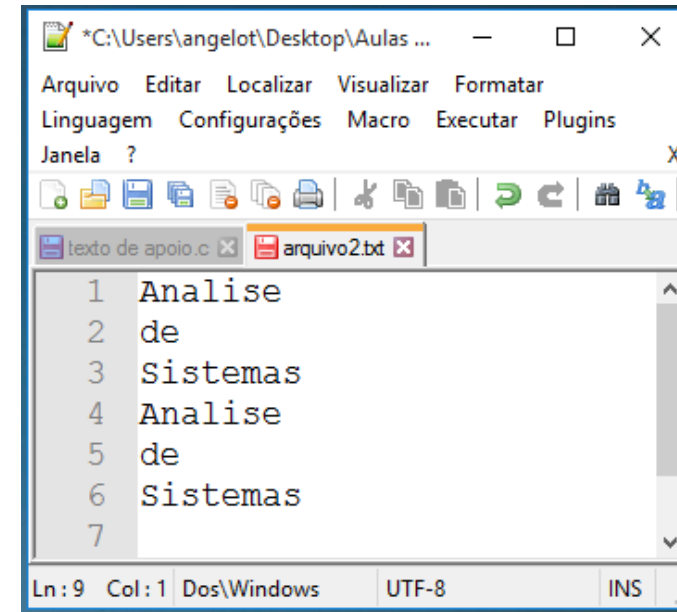
int main() {
    setlocale(LC_ALL, "");
    char str[30];
    FILE *f;

    f = fopen("arquivo2.txt", "r");
    if(f == NULL) {
        printf("Erro na abertura!\n");
        system("pause");
        exit(1);
    }

    fgets(str, 30, f);
    printf("%s\n", str);

    printf("\n\n\n");
    fclose(f);
    system("pause");
    return 0;
}
```

`fgets()` lê uma *string* até encontrar um caractere de nova linha “\n”, ou tamanho -1 caracteres (EOF)



# Estrutura de Dados 1

## Gravando blocos de *bytes* com a função `fwrite()`

- Até aqui vimos como escrever e ler caracteres e sequências de caracteres (*strings*), em arquivos;
- As funções de escrita de blocos de bytes permitem escrever dados mais complexos como os tipos `int`, `float`, `double`, vetores, ou mesmo o tipo definido pelo programador, como por exemplo, suas `structs`.
- Elas devem ser utilizadas **preferencialmente com arquivos binários**.





## Gravando blocos de *bytes* com a função `fwrite()`

- Para escrever um bloco de bytes em um arquivo, usamos a função `fwrite()`, e sua forma geral é:

```
int fwrite(void *buffer, int bytes, int count, FILE *fp);
```

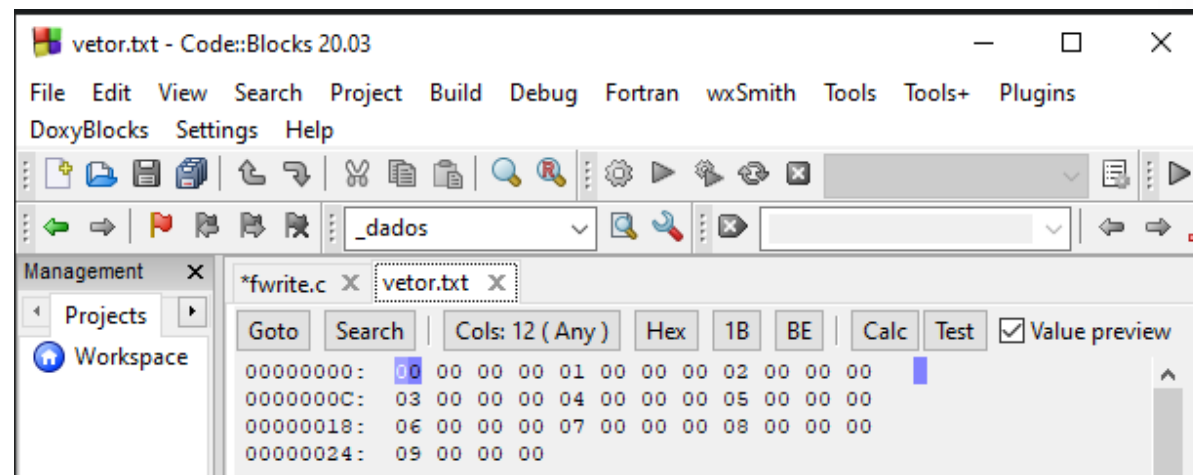
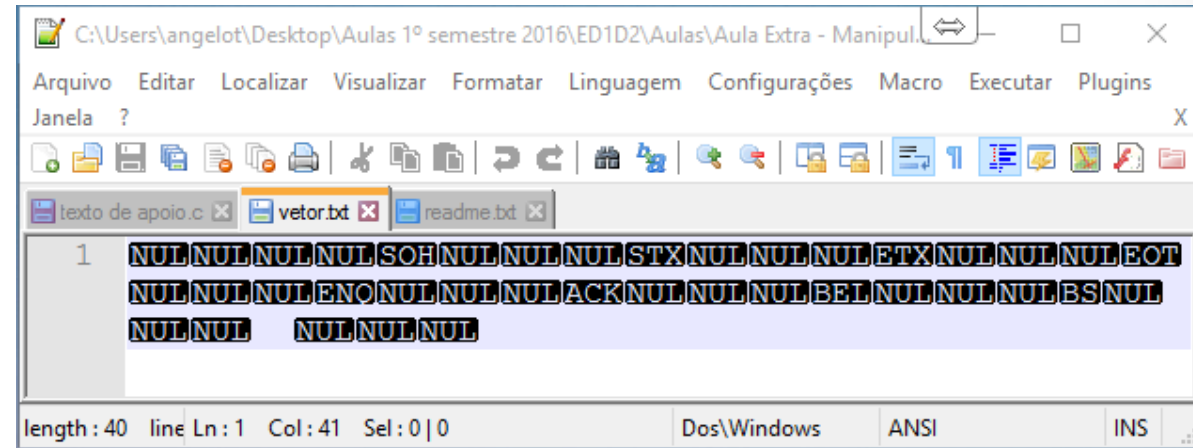
- Em que seus parâmetros são:
  - `buffer` - ponteiro genérico para os dados;
  - `bytes` - tamanho em bytes, de cada unidade de dado a ser gravada;
  - `count` - total de unidades de dados que devem ser gravadas;
  - `fp` – o ponteiro para o arquivo.
- Retorno inteiro: número total de unidades de dados gravada com sucesso.

# Estrutura de Dados 1

## Gravando blocos de *bytes* com a função `fwrite()`

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

int main() {
    FILE *f;
    f = fopen("vetor.txt", "wb");
    if(f == NULL) {
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    int total_gravado, v[10] = {0,1,2,3,4,5,6,7,8,9};
    //grava todo o vetor no arquivo (10 posições)
    total_gravado = fwrite(v, sizeof(int), 10, f);
    if(total_gravado != 10) {
        printf("Erro na escrita do arquivo\n");
        system("pause");
        exit(1);
    } else {
        printf("Arquivo gravado com sucesso!");
    }
    printf("\n\n\n\n");
    fclose(f);
}
```



# Estrutura de Dados 1

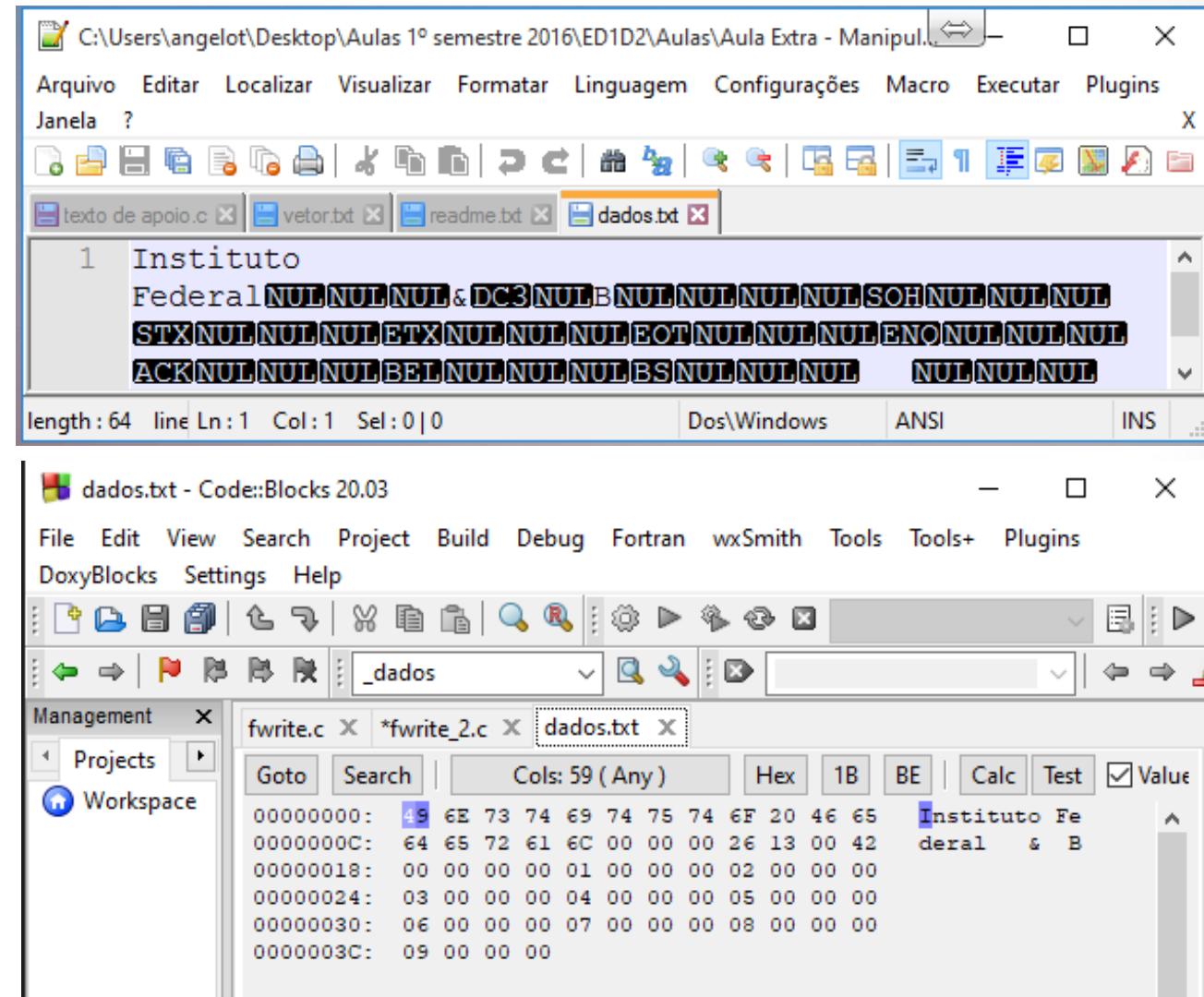
## Gravando blocos de *bytes* com a função `fwrite()`

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

int main() {
    FILE *f;
    f = fopen("dados.txt", "wb");
    if (f == NULL) {
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    char str[20] = "Instituto Federal";
    float x = 32.0187;
    int vetor[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};

    fwrite(str, sizeof(char), 20, f);
    fwrite(&x, sizeof(float), 1, f);
    fwrite(vetor, sizeof(int), 10, f);

    printf("\n\n\n\n");
    fclose(f);
    return 0;
}
```



# Estrutura de Dados 1

## Gravando blocos de *bytes* com a função `fwrite()`

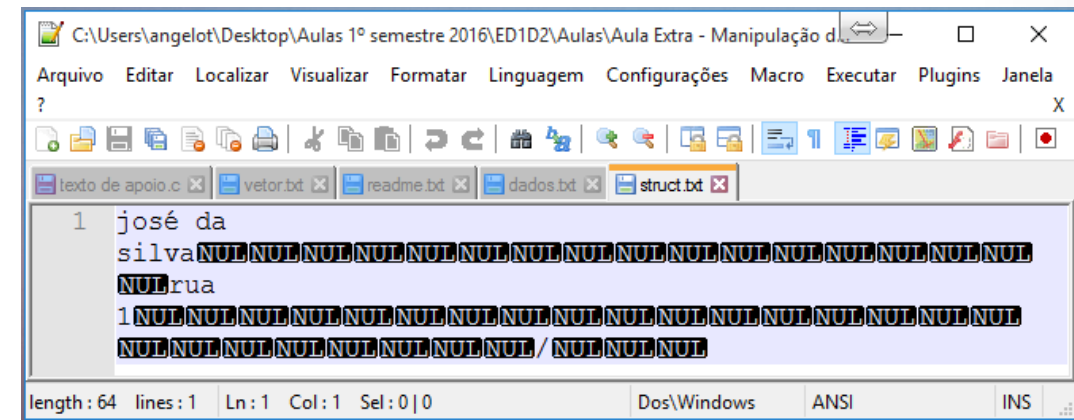
```
#include <stdio.h>
#include <stdlib.h>

struct cadastro{
    char nome[30];
    char endereco[30];
    int idade;
};

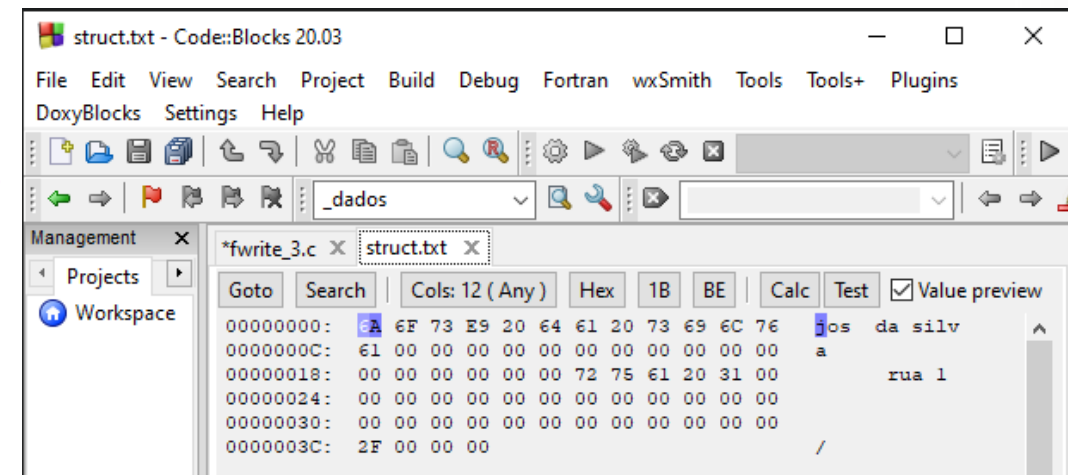
int main(){
    FILE *f;
    f = fopen("struct.txt", "wb");
    if(f == NULL){
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    struct cadastro cliente = {"josé da silva", "rua 1", 47};

    fwrite(&cliente, sizeof(struct cadastro), 1, f);

    printf("\n\n\n\n");
    fclose(f);
    return 0;
}
```



```
1  josé da
silva
rua
1  rua
```



```
00000000: 62 6F 73 E9 20 64 61 20 73 69 6C 76 1  jos da silv
0000000C: 61 00 00 00 00 00 00 00 00 00 00 00 a
00000018: 00 00 00 00 00 00 72 75 61 20 31 00 rua 1
00000024: 00 00 00 00 00 00 00 00 00 00 00 00
00000030: 00 00 00 00 00 00 00 00 00 00 00 00
0000003C: 2F 00 00 00 /
```



# Estrutura de Dados 1

## Lendo blocos de *bytes* com a função `fread()`

- Para ler um bloco de bytes de um arquivo usamos a função `fread()`, sua forma geral é:

```
int fread(void *buffer, int bytes, int count, FILE *fp);
```

- Em que seus parâmetros são:
  - `buffer` – ponteiro genérico para os dados;
  - `bytes` – tamanho, em bytes, de cada unidade de dado a ser lida;
  - `count` - total de unidades de dados que devem ser lidas.
  - `fp` – o ponteiro para o arquivo.
- Retorno: total de unidades de dados lidas com sucesso.



# Estrutura de Dados 1

## Lendo blocos de *bytes* com a função `fread()`



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

int main() {
    FILE *f;
    int i;
    f = fopen("vetor.txt", "rb");
    if(f == NULL){
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    int total_lido, v[10]; // {0,1,2,3,4,5,6,7,8,9};
    //grava todo o vetor no arquivo (10 posições)
    total_lido = fread(v, sizeof(int), 10, f);
    if(total_lido != 10){
        printf("Erro na leitura do arquivo\n");
        system("pause");
        exit(1);
    }else{
        printf("Arquivo lido com sucesso!");
    }
    printf("\n\n\n\n");
    fclose(f);
    printf("conteudo do arquivo: ");
    for(i = 0; i < 10; i++){
        printf(" %d", v[i]);
    }
    printf("\n\n\n\n");
}
```



# Estrutura de Dados 1

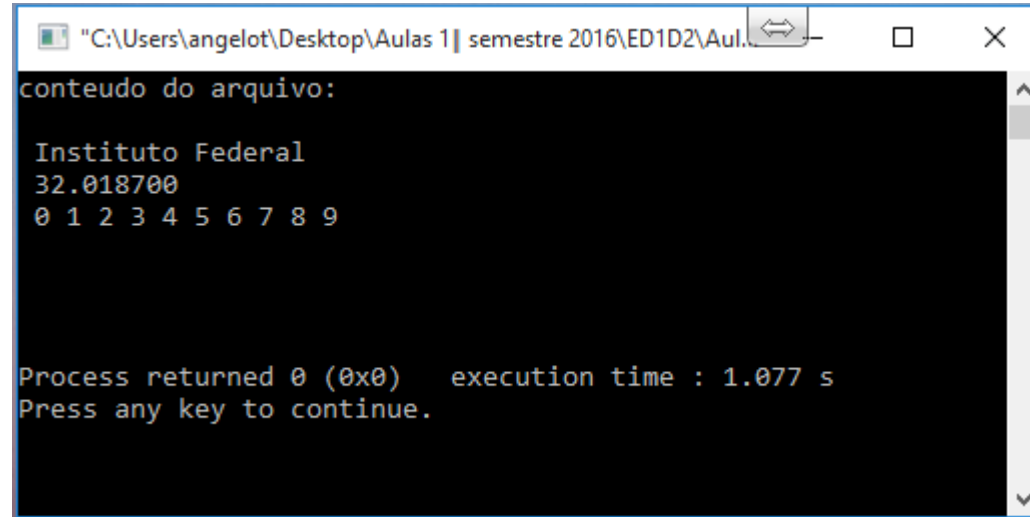
## Lendo blocos de *bytes* com a função `fread()`

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

int main() {
    FILE *f;
    int i;
    f = fopen("dados.txt", "rb");
    if(f == NULL) {
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    char str[20];
    float x;
    int v[10];

    fread(str, sizeof(char), 20, f);
    fread(&x, sizeof(float), 1, f);
    fread(v, sizeof(int), 10, f);

    printf("conteudo do arquivo:\n\n");
    printf("%s\n %f\n", str, x);
    for(i = 0; i < 10; i++){
        printf(" %d", v[i]);
    }
    fclose(f);
    printf("\n\n\n\n");
}
```



```
"C:\Users\angelot\Desktop\Aulas 1\ semestre 2016\ED1D2\Aul...
conteudo do arquivo:

Instituto Federal
32.018700
0 1 2 3 4 5 6 7 8 9

Process returned 0 (0x0) execution time : 1.077 s
Press any key to continue.
```



# Estrutura de Dados 1

## Lendo blocos de *bytes* com a função fread()

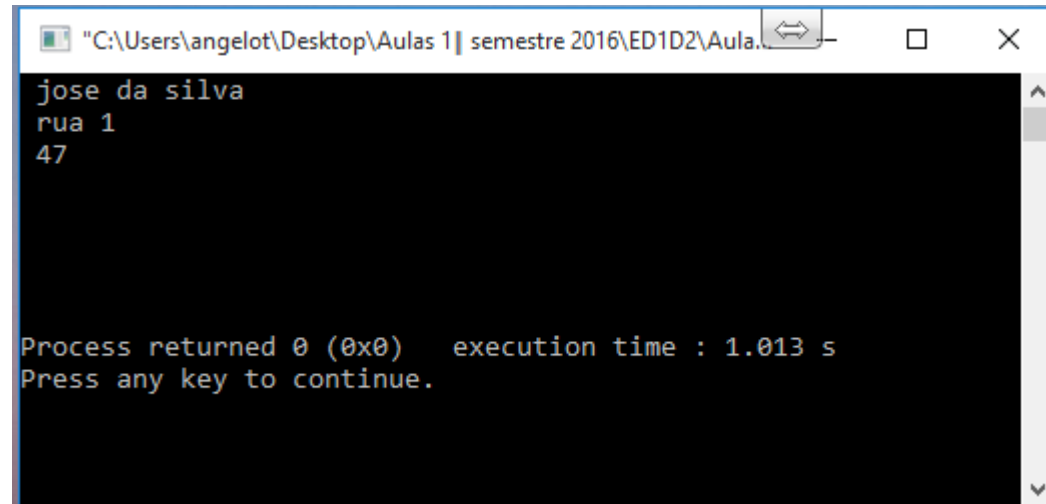
```
#include <stdio.h>
#include <stdlib.h>

struct cadastro{
    char nome[30];
    char endereco[30];
    int idade;
};

int main(){
    FILE *f;
    f = fopen("struct.txt", "rb");
    if(f == NULL){
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    struct cadastro cliente; // {"josé da silva", "rua 1", 47};

    fread(&cliente, sizeof(struct cadastro), 1, f);
    printf(" %s\n %s\n %d\n", cliente.nome, cliente.endereco, cliente.idade);

    printf("\n\n\n\n");
    fclose(f);
    return 0;
}
```



```
"C:\Users\angelot\Desktop\Aulas 1\ semestre 2016\ED1D2\Aula..."
jose da silva
rua 1
47

Process returned 0 (0x0)   execution time : 1.013 s
Press any key to continue.
```





## Gravando dados formatados com a função fprintf()

- Até o momento vimos como ler e escrever em arquivos caracteres, *strings* e blocos de bytes;
- A linguagem C também permite escrever uma lista formatada de variáveis em um arquivo do mesmo modo como é feito na tela do computador com a função printf();
- Para isso usamos a função fprintf().
- Forma geral da função printf();

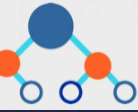
```
printf("tipos de saída", variáveis);
```

- Forma geral da função fprintf();

```
fprintf(FILE *fp, "tipos de saída", variáveis);
```

# Estrutura de Dados 1

## Gravando dados formatados com a função fprintf()



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

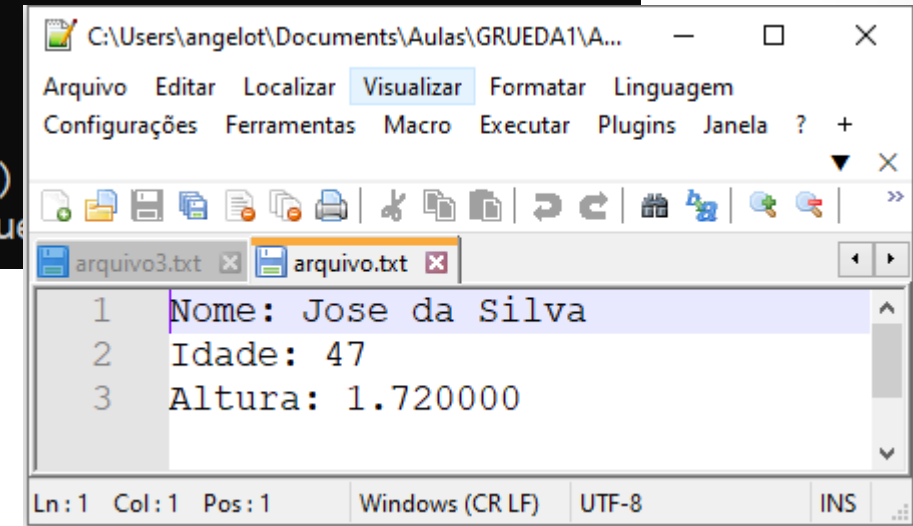
```
int main(){
    char nome[20] = "Jose da Silva";
    int i = 47;
    float altura = 1.72;
    FILE *f = fopen("arquivo.txt", "w");
    if(f == NULL){
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    printf("Nome: %s\nIdade: %d\nAltura: %f", nome, i, altura);

    fprintf(f, "Nome: %s\nIdade: %d\nAltura: %f", nome, i, altura);

    fclose(f);
    printf("\n\n\n");
    return 0;
}
```

```
Nome: Jose da Silva
Idade: 47
Altura: 1.720000
```

```
Process returned 0 (0x0)
Press any key to continue
```





## Lendo uma lista de dados formatados com a função fscanf()

- Assim como podemos gravar em um arquivo dados formatados, a Linguagem C também permite ler uma lista formatada de variáveis do arquivo, de modo análogo ao que é feito quando o computador lê os dados inseridos pelo teclado, com a função scanf();
- Para isso utilizamos a função fscanf();
- Forma geral da função scanf():

```
scanf("tipos de entrada", variáveis);
```

- Forma geral da função fscanf():

```
fscanf(FILE *fp, "tipos de entrada", variáveis);
```

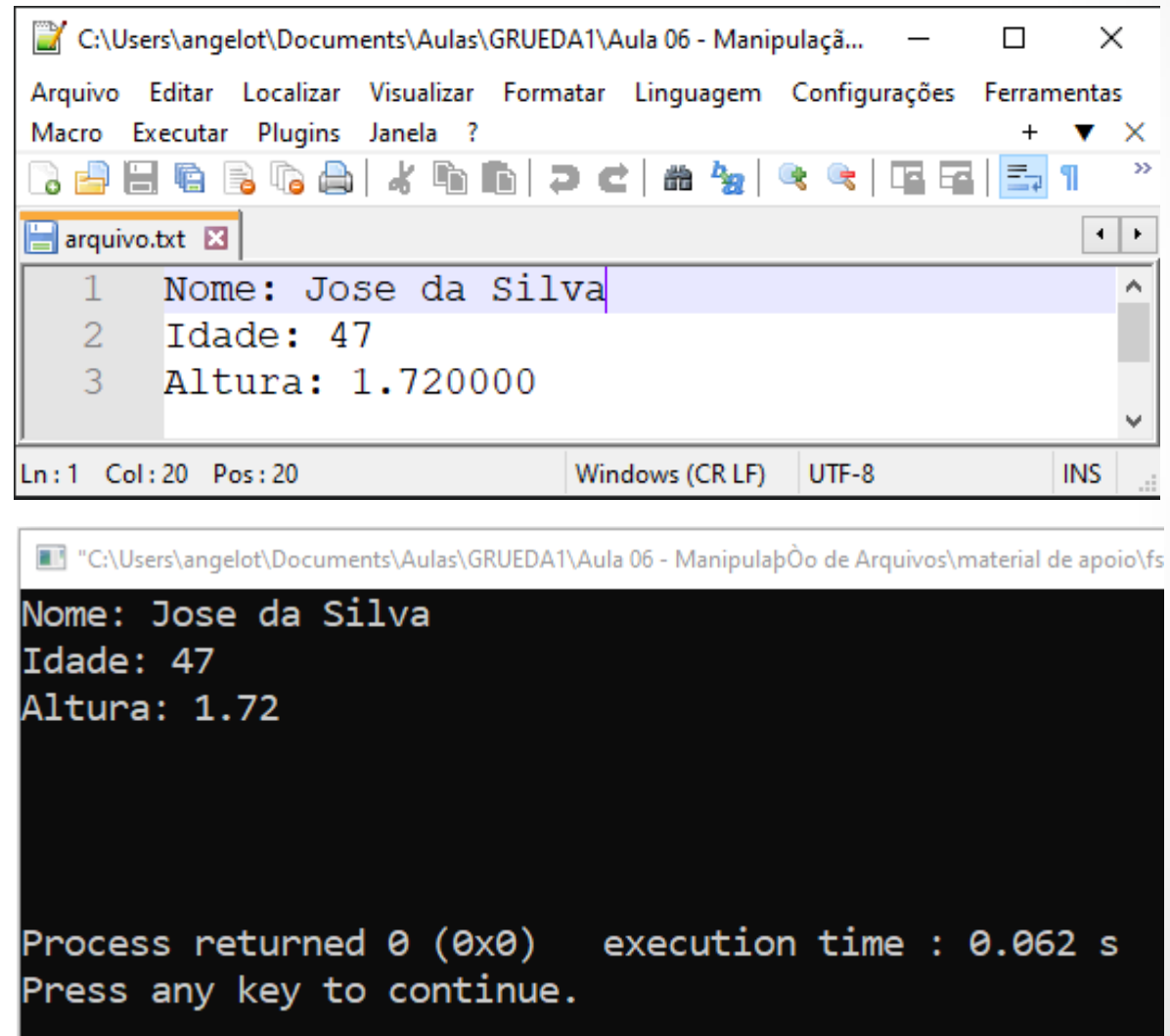
# Estrutura de Dados 1

## Lendo uma lista de dados formatados com a função fscanf()

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

int main() {
    FILE *f = fopen("arquivo.txt", "r");
    if (f == NULL) {
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    char texto[20], nome[20];
    int i;
    float altura;
    fscanf(f, "%s %[^\n]", texto, nome);
    printf("%s %s\n", texto, nome);
    fscanf(f, "%s %d", texto, &i);
    printf("%s %d\n", texto, i);
    fscanf(f, "%s %f", texto, &altura);
    printf("%s %.2f\n", texto, altura);

    fclose(f);
    printf("\n\n\n");
    return 0;
}
```



The screenshot displays two windows from a Windows operating system. The top window is a Notepad++ editor titled "C:\Users\angelot\Documents\Aulas\GRUEDA1\Aula 06 - Manipulaçã...". It shows the contents of a file named "arquivo.txt" with three lines of text: "1 Nome: Jose da Silva", "2 Idade: 47", and "3 Altura: 1.720000". The bottom window is a terminal window titled "C:\Users\angelot\Documents\Aulas\GRUEDA1\Aula 06 - Manipulaçã...". It shows the output of the C program, which reads the file and prints the data in a formatted manner: "Nome: Jose da Silva", "Idade: 47", and "Altura: 1.72". At the bottom of the terminal, it shows "Process returned 0 (0x0) execution time : 0.062 s" and "Press any key to continue."





## Movimentando-se dentro de um arquivo – fseek()

- De uma forma geral, o acesso a um arquivo é quase sempre sequencial. Porém, a linguagem C permite a realização de operações de leitura e escrita de forma randômica , utilizando a função `fseek()`;
- Forma geral:

```
int fseek(FILE *fp, long numbytes, int origem);
```

- Em que:
  - `fp`: é o ponteiro para o arquivo;
  - `numbytes`: é o total de bytes a serem saltados a partir de uma origem;
  - `origem`: ponto a partir do qual os “`numbytes`” serão contados para o salto.
- A função retorna 0 (zero), em caso de sucesso.



## Movimentando-se dentro de um arquivo – `fseek()`

- Os valores possíveis para o parâmetro *origem* são definidos por constantes na biblioteca `stdio.h`, e são:

Constante	Valor	Significado
SEEK_SET	0	Início do arquivo
SEEK_CUR	1	Ponto atual no arquivo
SEEK_END	2	Fim do arquivo

- É possível a utilização de valores negativos de *bytes* para retrocesso à uma determinada posição.



# Estrutura de Dados 1

## Movimentando-se dentro de um arquivo – fseek()



Este programa somente gera o arquivo onde, posteriormente será realizada a operação de busca com a função fseek(), com outro programa.

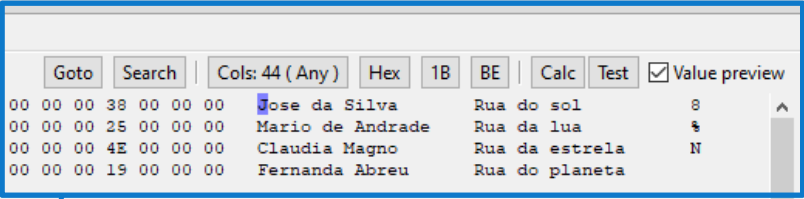
```
#include <stdio.h>
#include <stdlib.h>

typedef struct cadastroCliente{
    char nome[20];
    char rua[20];
    int idade;
}CADCLI;

int main(){
    FILE *fp = fopen("arquivo.txt", "wb");
    if(fp == NULL){
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    //criando o vetor de estruturas já preenchido:
    CADCLI cad[4] = {"Jose da Silva",    "Rua do sol",    56,
                    "Mario de Andrade", "Rua da lua",   37,
                    "Claudia Magno",    "Rua da estrela", 78,
                    "Fernanda Abreu",   "Rua do planeta", 25};

    //criando o arquivo em disco:
    fwrite(cad, sizeof(CADCLI), 4, fp);
    fclose(fp);
    system("pause");
    return 0;
}
```

Jose da Silva	Rua do sol	56
Mario de Andrade	Rua da lua	37
Claudia Magno	Rua da estrela	78
Fernanda Abreu	Rua do planeta	25



# Estrutura de Dados 1

## Movimentando-se dentro de um arquivo – fseek()

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

typedef struct cadastroCliente{
    char nome[20];
    char rua[20];
    int idade;
} CADCLI;

int main(){
    FILE *fp = fopen("arquivo.txt", "rb");
    if(fp == NULL){
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    CADCLI c;

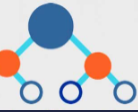
    fseek(fp, 2*sizeof(CADCLI), SEEK_SET);
    fread(&c, sizeof(CADCLI), 1, fp);
    printf(" Nome: %s \n Endereco: %s \n Idade: %d\n", c.nome, c.rua, c.idade);
    system("pause");
    return 0;
}
```

Jose da Silva	Rua do sol	56
Mario de Andrade	Rua da lua	37
Claudia Magno	Rua da estrela	78
Fernanda Abreu	Rua do planeta	25

"C:\Users\angelot\Documents\Aulas\GRUEDA1\Aula 06 - Manipulação de Arquivos\material c

```
Nome: Claudia Magno
Endereco: Rua da estrela
Idade: 78
Pressione qualquer tecla para continuar. . .
```





## Movimentando-se dentro de um arquivo – `rewind()`

- Outra opção de movimentação dentro do arquivo é simplesmente retornar para o seu início usando a função `rewind()`;
- Forma geral:

```
void rewind(FILE *fp);
```

# Estrutura de Dados 1

## Movimentando-se dentro de um arquivo – rewind()

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

typedef struct cadastroCliente{
    char nome[20];
    char rua[20];
    int idade;
} CADCLI;

int main(){
    FILE *fp = fopen("arquivo.txt", "rb");
    if(fp == NULL){
        printf("Erro na abertura\n");
        system("pause");
        exit(1);
    }
    CADCLI c;

    fseek(fp, 2*sizeof(CADCLI), SEEK_SET);
    fread(&c, sizeof(CADCLI), 1, fp);
    printf(" Nome: %s \n Endereco: %s \n Idade: %d\n", c.nome, c.rua, c.idade);
    rewind(fp);
    fread(&c, sizeof(CADCLI), 1, fp);
    printf(" Nome: %s \n Endereco: %s \n Idade: %d\n", c.nome, c.rua, c.idade);
    system("pause");
    return 0;
}
```

↑	Jose da Silva	Rua do sol	56
	Mario de Andrade	Rua da lua	37
↑	Claudia Magno	Rua da estrela	78
	Fernanda Abreu	Rua do planeta	25

```
"C:\Users\angelot\Documents\Aulas\GRUEDA1\Aula 06 - Manipulaç o de Arquivos\material de apoio\
Nome: Claudia Magno
Endereco: Rua da estrela
Idade: 78
Nome: Jose da Silva
Endereco: Rua do sol
Idade: 56
Pressione qualquer tecla para continuar. . .
```



# Estrutura de Dados 1

## Atividade 2

- Utilizando o formato da estrutura dos exercícios de uma aula anterior (abaixo), faça um programa que receba 5 funcionários como entrada, armazene-os em um vetor de estruturas e em seguida salve-o em um arquivo binário.

```
struct funcionario{  
    int ID;  
    char nome[30];  
    int idade;  
    float salario;  
};
```

- Gere um segundo programa que leia no arquivo e imprima apenas o terceiro elemento, lendo diretamente do arquivo salvo anteriormente, e sem carregar o vetor totalmente na memória. Lembre-se você precisará de 2 programas: um para gerar o arquivo e outro para ler o dado do arquivo gerado pelo primeiro.
- Entregue como atividade 2 no Moodle.

