

# Algoritmos e Lógica de Programação I

## Funções e Arquivos

Prof. MSc. Rafael Staiger Bressan  
[rafael.bressan@unicesumar.edu.br](mailto:rafael.bressan@unicesumar.edu.br)

# Cronograma

- Funções
- Arquivos



# Modularização

- **Modularizar** um programa consiste em dividi-lo em partes que serão desenvolvidas em separado. Essas partes executam tarefas menores que depois serão acopladas para formar o programa. A cada uma dessas partes chamamos de subprograma.
- Os subprogramas permitem uma melhor legibilidade e manutenibilidade do programa.



# Modularização

- Facilitar a solução de problemas complexos.

**“A arte de programar consiste na arte de organizar e dominar a complexidade dos sistemas”** (*Dijkstra, 1972*)



# Modularização

- Divisão de um problema original em subproblemas (módulos) mais fáceis de resolver e transformáveis em trechos mais simples, com poucos comandos (subprogramas).



# Modularização

- Trechos de código independentes, com estrutura semelhante àquela de programas, mas executados somente quando chamados por outro(s) trecho(s) de código.
- Devem executar UMA tarefa específica, muito bem identificada (conforme a programação estruturada).



# Modularização

- Ao ser ativado um subprograma, o fluxo de execução desloca-se do fluxo principal para o subprograma. Concluída a execução do subprograma, o fluxo de execução retorna ao ponto imediatamente após onde ocorreu a chamada do subprograma.



# Modularização

## Vantagens

- Maior controle sobre a complexidade.
- Estrutura lógica mais clara.
- Maior facilidade de depuração e teste, já que subprogramas podem ser testados separadamente.
- Possibilidade de reutilização de código.





# Subprogramas em Linguagem C

Implementados através de FUNÇÕES



# Funções

- Funções são segmentos de programa que executam uma determinada tarefa específica.
- Funções (também chamadas de rotinas, ou sub-programas) são a essência da programação estruturada.
  - Ex: `sqrt()`, `strlen()`, etc.



# Funções

- O programador também pode escrever suas próprias funções, chamadas de funções de usuário, que tem uma estrutura muito semelhante a um programa.



# Funções

## Representação Geral ( C )

```
tipo_da_funcao  nome_da_função (lista_de_parâmetros)  
{  
    //declarações locais  
    //comandos  
}
```



# Funções

## Representação Geral ( C )

**tipo\_da\_funcao:** o tipo de valor retornado pela função. Se não especificado, por falta a função é considerada como retornando um inteiro.

**nome\_da\_função:** nome da função conforme as regras do C



# Funções

## Representação Geral ( C )

- **lista\_de\_parâmetros:** tipo de cada parâmetro seguido de seu identificador, com vírgulas entre cada parâmetro. Mesmo se nenhum parâmetro for utilizado, os parênteses são obrigatórios.
  - Os parâmetros da declaração da função são chamados de parâmetros formais.



# Funções

## Representação Geral ( C )

- **lista\_de\_parâmetros:** tipo de cada parâmetro seguido de seu identificador, com vírgulas entre cada parâmetro. Mesmo se nenhum parâmetro for utilizado, os parênteses são obrigatórios.
  - Os parâmetros da declaração da função são chamados de parâmetros formais.



# Funções

```
tipo_da_funcao nome_da_função  
(lista_de_parâmetros)  
{ ... }
```

```
soma_valores (int valor1, int valor2) // por falta é inteira  
void imprime_linhas(int num_lin)  
void apresenta_menu ( )  
float conv_dolar_para_reais(float dolar);
```





# Funções

- Funções void
  - **Void** é um termo que indica ausência.
  - Em linguagem C é um tipo de dados.

# Funções

Observe esse código.

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  main( )
4  {
5      int i;
6      system("color 70");
7      for (i=1;i<20;i++) {
8          printf("*");
9      }
10     printf("\n");
11     printf("Numeros entre 1 e 5\n");
12     for (i=1;i<20;i++){
13         printf("*");
14     }
15     printf("\n");
16     for (i=1;i<=5;i++){
17         printf("%d\n",i);
18     }
19     for (i=1;i<20;i++) {
20         printf("*");
21     }
22     printf("\n");
23     system("pause");
24 }
```

```
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
Pressione qualquer tecla para continuar. . .
```



# Funções

- A repetição de trechos de código idênticos em um programa pode ser um procedimento fácil e rápido, mas facilmente tende a produzir erros.
- Tanto a manutenção quanto a alteração de programas com trechos repetidos tende a ser mais trabalhosa e sujeita a erros.



# Funções

- Com frequência alterações de trechos iguais que se repetem não são realizadas em todas as ocorrências do trecho ou são realizadas de forma incompleta em alguma ocorrência, com resultados bastante danosos.
- A solução para esta questão são os subprogramas.
- A seguir uma versão do programa escreveint onde as linhas de asterisco são produzidas pela função `apresente_linha`.

# Funções

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  main( )
4  {
5      int i;
6      system("color 70");
7      for (i=1;i<20;i++) {
8          printf("*");
9      }
10     printf("\n");
11     printf("Numeros entre 1 e 5\n");
12     for (i=1;i<20;i++){
13         printf("*");
14     }
15     printf("\n");
16     for (i=1;i<=5;i++){
17         printf("%d\n",i);
18     }
19     for (i=1;i<20;i++) {
20         printf("*");
21     }
22     printf("\n");
23     system("pause");
24 }
```

Envia



Recebe



```
void apresente_linha (void)
{
    int i;
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}
```

# Funções

```
1 #include<stdio.h>
2 #include <stdlib.h>
3 void apresente_linha(void);
4 main( )
5 {
6     int i;
7     system("color 70");
8     apresente_linha( );
9     printf("Numeros entre 1 e 5\n");
10    apresente_linha( );
11    for (i=1;i<=5;i++)
12        printf("%d\n",i);
13    apresente_linha( );
14    system("pause");
15 }
16 void apresente_linha (void)
17 {
18     int i;
19     for (i=1;i<20;i++)
20         printf("*");
21     printf("\n");
22 }
```

Carrega a função.

```
*****
Numeros entre 1 e 5
*****
1
2
3
4
5
*****
Pressione qualquer tecla para continuar. . .
```

# Funções

Envia



Recebe

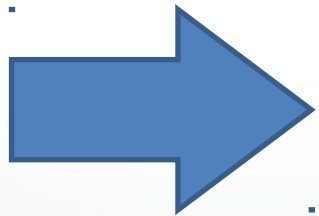


```
void apresente_linha (void)
{
    int i;
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}
```

# Funções

## Variáveis locais

- Os parâmetros que aparecem no cabeçalho das funções e as variáveis e constantes declaradas internamente a funções são locais à função.
- Na função **apresente\_linha**, o **i** é uma variável local a essa função.



```
void apresente_linha (void)
{
    int i;
    for (i=1;i<20;i++)
        printf("*");
    printf("\n");
}
```





# Funções

## Variáveis e constantes locais:

- Importante:
  - **Recomenda-se fazer todas as declarações de uma função no seu início.**
- As variáveis e constantes declaradas em uma função são ditas locais à função porque:
  - só podem ser referenciadas por comandos que estão dentro da função em que foram declaradas;
  - existem apenas enquanto a função em que foram declaradas está sendo executada. São criadas quando a função é ativada e são destruídas quando a função encerra.



# Funções Variáveis

- Variáveis locais :
  - uma função (inclusive a main) tem acesso somente às variáveis locais.
  - não altera valor de variáveis de outras funções.

# Funções - Variáveis

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  void mostra(void);
4  main( )
5  {
6      int a,b,c;
7      a = 10;b = 20; c = 30;
8      printf("Valores A = %d \n",a);
9      printf("Valores B = %d \n",b);
10     printf("Valores C = %d \n",c);
11     mostra();
12     printf("Valores A = %d \n",a);
13     printf("Valores B = %d \n",b);
14     printf("Valores C = %d \n",c);
15     system("pause");
16 }
17 void mostra (void)
18 {
19     int a,b,c;
20     a = 100;b = 200; c = 300;
21 }
```

```
Valores A = 10
Valores B = 20
Valores C = 30
Valores A = 10
Valores B = 20
Valores C = 30
Pressione qualquer tecla para continuar. . .
```

# Funções - Variáveis

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  void mostra(void);
4  main( )
5  {
6      int a,b,c;
7      a = 10;b = 20; c = 30;
8      printf("Valores A = %d \n",a);
9      printf("Valores B = %d \n",b);
10     printf("Valores C = %d \n",c);
11     mostra();
12     printf("Valores A = %d \n",a);
13     printf("Valores B = %d \n",b);
14     printf("Valores C = %d \n",c);
15     system("pause");
16 }
17 void mostra (void)
18 {
19     int a,b,c;
20     a = 100;b = 200; c = 300;
21     printf("Valores A = %d \n",a);
22     printf("Valores B = %d \n",b);
23     printf("Valores C = %d \n",c);
24 }
```

```
Valores A = 10
Valores B = 20
Valores C = 30
Valores A = 100
Valores B = 200
Valores C = 300
Valores A = 10
Valores B = 20
Valores C = 30
Pressione qualquer tecla para continuar. . .
```



# Funções

- Funções com tipo não void e com parâmetros
  - função pré-definida
    - Exemplo: `sqrt()`
    - Para extrair a raiz quadrada dos valores é usada a função pré-definida `sqrt`, da biblioteca `math.h`.

# Funções

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <math.h>
4  main ( )
5  {
6      int seguir;
7      double valor;
8      do
9      {
10         printf("\nValor para extrair raiz: ");
11         scanf("%lf", &valor);
12         printf ("\nRaiz quadrada de %6.2lf = %6.2lf\n", valor, sqrt(valor));
13         printf("\nMais um valor, digite 1, para parar, digite 0: ");
14         scanf("%d", &seguir);
15     }
16     while (seguir);
17     system("pause");
18 }
```

```
Valor para extrair raiz: 10
Raiz quadrada de  10.00 =   3.16
Mais um valor, digite 1, para parar, digite 0: 0
Pressione qualquer tecla para continuar. . .
```



# Funções

- A função ***sqrt*** é do tipo ***double***, isso significa que quando ela é chamada, no lugar de sua chamada retorna um valor ***double***.
- Para executar essa função é necessário fornecer um parâmetro, o valor para o qual se deseja que a raiz quadrada seja calculada.
- No exemplo, está armazenado na variável *valor*.



# Funções

- A seguir um programa que calcula o produto de um número indeterminado de pares de valores informados.
- Para calcular os produtos é usada a função definida pelo usuário `calc_produto`.



# Funções

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 int calc_produto(int, int);
4 main ( )
5 {
6     int seguir;
7     int oper1, oper2, produto;
8     do
9     {
10         printf("\nOperando 1: ");
11         scanf("%d", &oper1);
12         printf("\nOperando 2: ");
13         scanf("%d", &oper2);
14         printf("\nProduto = %d\n", calc_produto(oper1, oper2));
15         printf("\nPara continuar, digite 1, para parar, digite 0: ");
16         scanf("%d", &seguir);
17     }
18     while (seguir);
19     system("pause");
20 }
21 int calc_produto(int valor1, int valor2)
22 {
23     return valor1 * valor2;
24 }
```

Operando 1: 10

Operando 2: 10

Produto = 100

Para continuar, digite 1, para parar, digite 0: 0

Pressione qualquer tecla para continuar. . .



# Funções

```
int calc_produto(int valor1, int valor2)
{
    return valor1 * valor2;
}
```

A função `calc_produto` é do tipo `int`, isso significa que quando ela é chamada, no lugar de sua chamada retorna um valor `int`.

Para executar essa função é necessário fornecer dois parâmetros, os dois valores para cálculo do produto, `oper1` e `oper2`.



## Funções return

- O comando return atribui valor a função.
- Ao ser executado, encerra a execução da função.
- Ao ser executado o return na função `calc_produto`, um valor é atribuído à função e ela encerra sua execução.
- No ponto onde ocorreu a chamada de `calc_produto`, um valor passa a estar disponível para processamento.



## Funções return

- Se uma função é declarada com tipo diferente de void (int, char, float, etc.) significa que ela pretende explorar a possibilidade de retorno de um valor em seu nome, e então pode ser usada em expressões.
- Uma função que retorna um valor em seu nome deve conter pelo menos uma ocorrência do comando return, uma vez que é pela execução de um comando return que um valor é atribuído ao nome de uma função.



# Funções

- Uma função encerra sua execução quando:
  - o fim do seu código é atingido ou um comando `return` é encontrado e executado.
- **Vários comandos `return` podem existir em uma função?**



# Funções

- Sim, embora não seja recomendável.
- Segundo os princípios da programação estruturada seguidos na disciplina, cada função deve ter um único ponto de entrada e um único ponto de saída.
- Se vários returns existirem em uma função, tem-se múltiplos pontos de saída possíveis.
- Mas a função só conclui quando o primeiro return é ativado.



# Funções

- As funções devem ser declaradas de modo a serem o mais independentes possível do mundo externo a elas. Nos códigos das funções devem ser usados sempre que possível tão somente os parâmetros declarados no cabeçalho da função (se existirem) e os demais itens locais à função.



# Funções

- Em grande medida em C a preocupação com a independência das funções é facilitada pelo fato dos parâmetros de chamada e dos parâmetros da declaração da função ocuparem espaços de memória distintos e só existir a chamada passagem de parâmetro por valor entre eles.
- Passagem de parâmetro por valor: os parâmetros de chamada e os parâmetros formais (da declaração da função) só se conectam no momento da chamada da função e então o que há é apenas a transferência de valores entre os parâmetros respectivos.



# Funções

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void soma_dez_a_valor(int);
4  main ( )
5  {
6      int valor;
7      system("color 71");
8      printf("\nValor inteiro: ");
9      scanf("%d", &valor);
10     printf("\nNa Main: valor antes da chamada da funcao: %d\n", valor);
11     soma_dez_a_valor(valor);
12     printf("\nNa Main: valor apos chamada da funcao: %d\n", valor);
13     system("pause");
14 }
15 void soma_dez_a_valor(int valor)
16 {
17     valor = valor + 10;
18     printf("\nNa Funcao: valor dentro da funcao: %d\n", valor);
19 }
```

Valor inteiro: 10

Na Main: valor antes da chamada da funcao: 10

Na Funcao: valor dentro da funcao: 20

Na Main: valor apos chamada da funcao: 10

Pressione qualquer tecla para continuar. . .

# Funções

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  void soma_dez_a_valor(int);
4  main ( )
5  {
6      int valor;
7      system("color 71");
8      printf("\nValor inteiro: ");
9      scanf("%d", &valor);
10     printf("\nNa Main: valor antes da chamada da funcao: %d\n", valor);
11     soma_dez_a_valor(valor);
12     printf("\nNa Main: valor apos chamada da funcao: %d\n", valor);
13     system("pause");
14 }
15 void soma_dez_a_valor(int num)
16 {
17     num = num + 10;
18     printf("\nNa Funcao: valor dentro da funcao: %d\n", num);
19 }
```

Valor inteiro: 10

Na Main: valor antes da chamada da funcao: 10

Na Funcao: valor dentro da funcao: 20

Na Main: valor apos chamada da funcao: 10

Pressione qualquer tecla para continuar. . .



# Funções

- Os **nomes das variáveis declaradas** no cabeçalho de uma função **são independentes dos nomes das variáveis usadas para chamar a mesma função.**
- As **declarações de uma função são locais a essa função.** Os parâmetros declarados no cabeçalho de uma função existem somente dentro da função onde estão declarados.



# Funções

## Passagem de Parâmetros

- Ao ser ativada a função `calc_produto`, `valor1` e `valor2` são criadas.
- E os valores existentes nesse momento em `oper1` e `oper2` são transferidos para `valor1` e `valor2`.
- A conexão entre `oper1` e `valor1` e `oper2` e `valor2` só existe no momento que a função é ativada.
- Fora o momento da ativação as funções `calc_produto` e `main` são mundos independentes.

```
int calc_produto(int valor1, int valor2)  
calc_produto(oper1, oper2)
```



# Funções

```
int calc_produto(int valor1, int valor2)  
calc_produto(oper1, oper2)
```

- valor1 e valor2 existem na função calc\_produto.
- oper1 e oper2 existem na função main.
- Quaisquer modificações de valor1 e valor2 que aconteçam a partir da chamada de calc\_produto só são conhecidas e percebidas dentro da função calc\_produto.

# Funções

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int calc_produto(int, int);
4  main ( )
5  {
6      int seguir;
7      int oper1, oper2, produto;
8      do
9      {
10         printf("\nOperando 1: ");
11         scanf("%d", &oper1);
12         printf("\nOperando 2: ");
13         scanf("%d", &oper2);
14         printf ("\nProduto = %d\n", calc_produto(oper1, oper2));
15         printf("\nMais um valor, digite 1, para parar, digite 0: ");
16         scanf("%d", &seguir);
17     }
18     while (seguir);
19     system("pause");
20 }
21 int calc_produto(int valor1, int valor2)
22 {
23     return valor1 * valor2;
24 }
```

Protótipo

Chamada da função

Declaração da função



# Funções

- **Forma geral de declaração de um protótipo.**
  - **tipo\_da\_funcao nome\_da\_função (lista de tipos dos parâmetros);**
  - **tipo\_da\_funcao:** o tipo de valor retornado pela função.
  - **nome\_da\_função:** nome da função conforme as regras do C.
  - **lista de tipos dos parâmetros:** tipo de cada parâmetro, separados entre si por vírgulas.



## Funções

- **Em C, é possível chamar uma função de dentro de outra função, mas não é possível declarar uma função dentro de outra função!**





## Atividade 1

- Escreva o código de uma função que calcule o fatorial de um número informado como parâmetro.

# Atividade 1

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  int fatorial(int);
4  main() {
5      int N;
6      printf ("Informe o numero: ");
7      scanf ("%d", &N);
8      printf ("fatorial: %d\n", fatorial(N));
9      system("pause");
10 }
11
12 // declaracao da funcao fatorial
13 int fatorial(int n) {
14     int I, fat=1;
15     for (I=1; I<=n; I++) {
16         fat=fat*I;
17     }
18     return (fat);
19 }
```

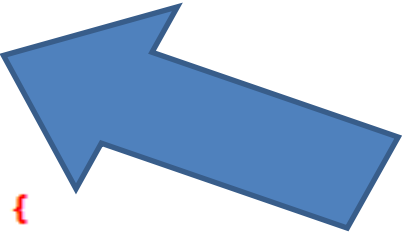


## Atividade 1

- Problema na função fatorial definida: é do tipo inteiro, o que limita muito a sua aplicabilidade, pois o maior número do **tipo inteiro** é relativamente pequeno.
- Solução: definir a função como do **tipo double**

# Atividade 1

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  double fatorial(int);
4  main() {
5      int N;
6      printf ("Informe o numero: ");
7      scanf ("%d",&N);
8      printf ("fatorial: %lf\n",fatorial(N));
9      system("pause");
10 }
11 double fatorial(int n){
12     int I;
13     double fat=1.0;
14     for (I=1;I<=n;I++){
15         fat=fat*I;
16     }
17     return (fat);
18 }
```





## Atividade 2

- **Escreva o código de uma função que calcule a média aritmética de dois valores informados como parâmetros**

## Atividade 2

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  float media(float, float);
4  main() {
5      float v1,v2,m;
6      printf ("Informe os numeros: ");
7      scanf ("%f %f",&v1,&v2);
8      m=media(v1,v2);
9      printf("a media dos numeros e': %.4f\n",m);
10     system("pause");
11 }
12 float media(float n1, float n2){
13     return ((n1+n2)/2);
14 }
```



## Atividade 3

- **Desenvolva um programa utilizando funções que retorne o maior elemento de um vetor.**

# Problematizando

```
1  #include<stdio.h>
2  #include <stdlib.h>
3  void mostra(void);
4  main( )
5  {
6      int a,b,c;
7      a = 10;b = 20; c = 30;
8      printf("Valores A = %d \n",a);
9      printf("Valores B = %d \n",b);
10     printf("Valores C = %d \n",c);
11     mostra();
12     printf("Valores A = %d \n",a);
13     printf("Valores B = %d \n",b);
14     printf("Valores C = %d \n",c);
15     system("pause");
16 }
17 void mostra (void)
18 {
19     int a,b,c;
20     a = 100;b = 200; c = 300;
21     printf("Valores A = %d \n",a);
22     printf("Valores B = %d \n",b);
23     printf("Valores C = %d \n",c);
24 }
```

```
Valores A = 10
Valores B = 20
Valores C = 30
Valores A = 100
Valores B = 200
Valores C = 300
Valores A = 10
Valores B = 20
Valores C = 30
Pressione qualquer tecla para continuar. . .
```





## Problematizando

- **Como fazer para alterar as variáveis a, b e c dentro da função?**

- **MAIN() → Envia A=1, B=2 e C=3**

- **Função ALTERA A=10, B=20 e C=30**

- **MAIN() → A=10, B=20 e C=30**



# Exercícios

- Desenvolva um programa em C que contenha as funções de soma, subtração, multiplicação e divisão de dois números reais.
- Desenvolva uma função para retornar a média dos elementos de um vetor. A função deve ter como entrada um vetor de números inteiros e o tamanho do vetor.
- Escolha 15 exercícios referente ao capítulo de funções do livro “Fundamentos da programação de computadores”.



# Arquivos

- A linguagem C utiliza o conceito de fluxo (stream) de dados para manipular vários tipos de dispositivos de armazenamento.
- Dados podem ser manipulados em dois diferentes tipos de fluxos: fluxos de texto e fluxos binários.
- Um fluxo de texto é composto por uma sequência de caracteres, que pode ou não ser dividida em linhas terminadas por um ***caracter*** de final de linha.



# Arquivos

- Um arquivo pode estar associado a qualquer dispositivo de entrada e saída, como por exemplo: teclado, vídeo, impressora, disco rígido, etc.
- O processo de trabalhar com arquivos em C consiste em três etapas:
  - Abrir o arquivo;
  - Ler e/ou gravar as informações desejadas no arquivo;
  - Fechar o arquivo.



# Arquivos

- As funções mais usadas estão armazenadas na biblioteca `stdio.h`
  - `fopen()` - Abre um arquivo
  - `fputc()` - Escreve um caracter em um arquivo
  - `fgetc()` - Lê um caracter de um arquivo
  - `fputs()` – escreve uma string em um arquivo
  - `fgets()` – lê uma linha de um arquivo
  - `fprintf()`- Equivalente a `printf()`
  - `fscanf()` - Equivalente a `scanf()`
  - `rewind()` - Posiciona o arquivo no início
  - `feof()` - Retorna verdadeiro se chegou ao fim do arquivo
  - `fclose()` – fecha



# Arquivos

- Antes de qualquer operação ser executada com o arquivo, ele deve ser aberto. Esta operação associa um fluxo de dados a um arquivo.
- Um arquivo pode ser aberto de diversas maneiras: leitura, escrita, leitura/escrita, adição de texto, etc.
- A função utilizada para abrir o arquivo é **fopen()**



# A função fopen()

- fopen(nome arquivo, modo de uso). Os modos de uso mais comuns são:
  - r (de read) : abre um arquivo texto para leitura
  - w (de write) : cria um arquivo texto para escrita
  - a (de append) : para adicionar conteúdo no fim de um arquivo texto já existente
  - r+ : abre um arquivo texto para leitura/escrita
  - w+ : cria um arquivo texto para leitura/escrita

# Exemplo

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main() {
4      FILE *arquivo; //vai ser associada ao arquivo
5      arquivo = fopen("c:/aula/aula1.txt","r");
6      if(arquivo==0) {
7          printf("Erro na leitura do arquivo \n");
8      }else{
9          printf("Arquivo aberto com sucesso \n");
10     }
11     fclose(arquivo); //fecha arquivo
12 }
```






# Leitura e gravação

- fgets (string, tamanho, arquivo)
- fputs(string, arquivo)
- fgetc(arquivo)
- fputc(char, arquivo)
  
- **fprintf(arquivo, ...)**
- **fscanf(arquivo, ...)**



# Exemplo

- Escreva um programa que leia os dados do seguinte arquivo, onde a primeira linha contém o nome, a segunda o sobrenome, a terceira o ano de nascimento e a quarta o peso (dados verídicos!):
  - Marcela
  - Snifer
  - 1984
  - 65.4




```
1  #include <stdio.h>
2  int main() {
3      int i;
4      float f;
5      char nome[20];
6      char nome2[20];
7      FILE *arquivo;
8      arquivo = fopen("c:/aula/aula1.txt", "r");
9      fscanf(arquivo, "%s", nome);
10     fscanf(arquivo, "%s", nome2);
11     fscanf(arquivo, "%i", &i);
12     fscanf(arquivo, "%f", &f);
13     printf("\n %i \n %f \n %s \n %s", i, f, nome, nome2);
14 }
```




# Exemplo

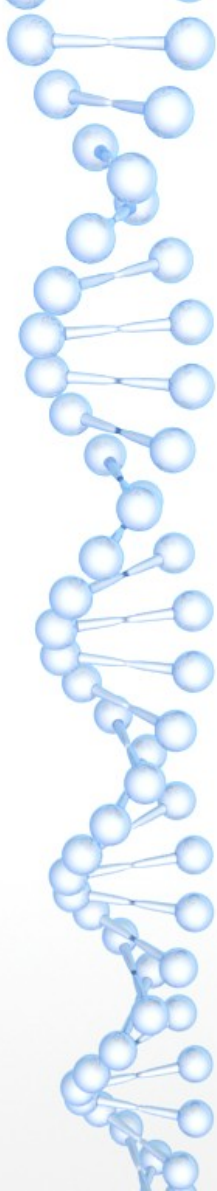
- Escreva um programa que leia os dados do seguinte arquivo, cada linha contém o número em ponto flutuante.
- Detalhe: não sabemos o número total de linhas! Imprima os números na tela.

# Exemplo



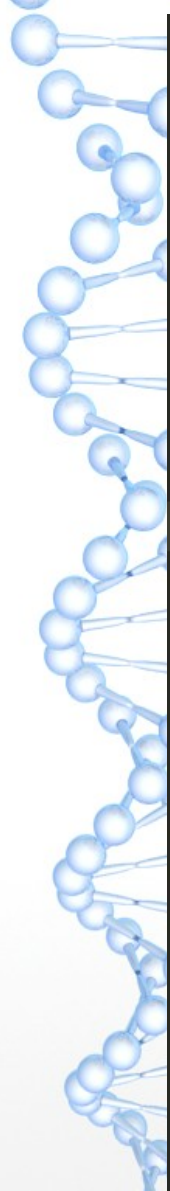
```
1 #include <stdio.h>
2 int main() {
3     float f;
4     FILE *arquivo;
5     arquivo = fopen("c:/aula/aula2.txt", "r");
6     while(fscanf(arquivo, "%f", &f) != EOF) {
7         printf("%f\n", f);
8     }
9 }
```





2020

```
1  /*
2     gera 26000 números inteiros
3     MSc. Prof. Rafael S. Bressan
4     2019
5  */
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <locale.h>
9  #define max 26000
10
11  main(){
12      int i;
13      setlocale(LC_ALL, "Portuguese");
14      FILE *arquivo;
15      arquivo = fopen("numbers.txt", "w");
16      if(arquivo == NULL) {
17          printf("Erro na abertura do arquivo!");
18          return 1;
19      }
20      for(i=0; i<max; i++){
21          fprintf(arquivo, "%d \n", rand() % (max*2));
22      }
23      fclose(arquivo);
24      return 0;
25  }
```



```
1  /*
2     Realiza a leitura de uma arquivo contendo números inteiros
3     MSc. Prof. Rafael S. Bressan
4     2019
5  */
6  #include <stdio.h>
7  #include <stdlib.h>
8  #include <locale.h>
9
10 main(){
11     int i, number;
12     setlocale(LC_ALL, "Portuguese");
13     FILE *arquivo;
14     arquivo = fopen("numbers.txt", "r");
15     if(arquivo == NULL) {
16         printf("Erro na abertura do arquivo!");
17         return 1;
18     }
19     while(fscanf(arquivo, "%d \n", &number) != EOF){
20         printf("%d \n", number);
21     }
22     fclose(arquivo);
23     return 0;
24 }
```





# Exercício

- Faça um programa em c que leia um arquivo TXT contendo nomes, apresente os nomes lidos.
- Desenvolva códigos em C que:
  - Abra | Leia | Escreva | Feche um arquivo .logs
- Desenvolva um código em C que leia e grave em um arquivo os registros de um produto de uma loja contendo as seguintes informações: nome, descrição e valor.





# Exercício

- Crie um arquivo que contenha vários nomes e telefones
  - Rafael
  - 22562235
  - Fulano
  - 11545525
  - .....
- Desenvolva um programa em C que registre os nomes lidos em uma (struct), apresente os dados de cada estrutura. Desenvolva também uma opção de busca por nome.

