Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

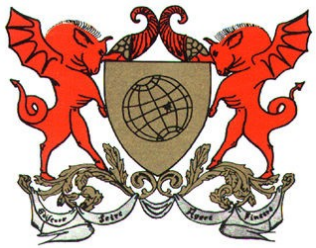
SIN 110

Programação

Curso de Sistemas de Informação

Prof. Rodrigo Smarzaro

smarzaro@ufv.br



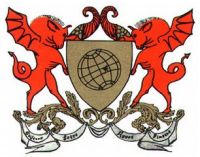
Universidade Federal de Viçosa
Campus Rio Paranaíba
Instituto de Ciências Exatas e Tecnológicas

Aula de Hoje

Modularização (Funções)

O que é modularização?

- No século XIX, **Henry Ford**, para baratear e massificar a montagem de carros, criou uma base modular.
- Esta base era usada para montar diversos tipos de automóveis.
- Ele batizou de modelo T (Ford Modelo T).



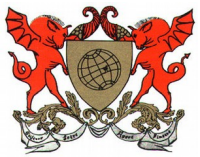
O que é modularização?

O que é modularização?

- Atualmente, encontram-se diversos produtos modulares.
- Isto significa que peças menores podem se encaixar de diversas formas, com o objetivo de formar algo maior.

O que é modularização?

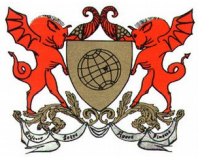
- Até agora, foi utilizado apenas o programa principal (**main**) para executar todas as tarefas.
- No entanto, esta não é a melhor solução para o desenvolvimento de código estruturado.
- A modularização do código visa separar as tarefas em **funções** ou **procedimentos**, cada um deles realizando uma parte do programa.



O que é modularização?

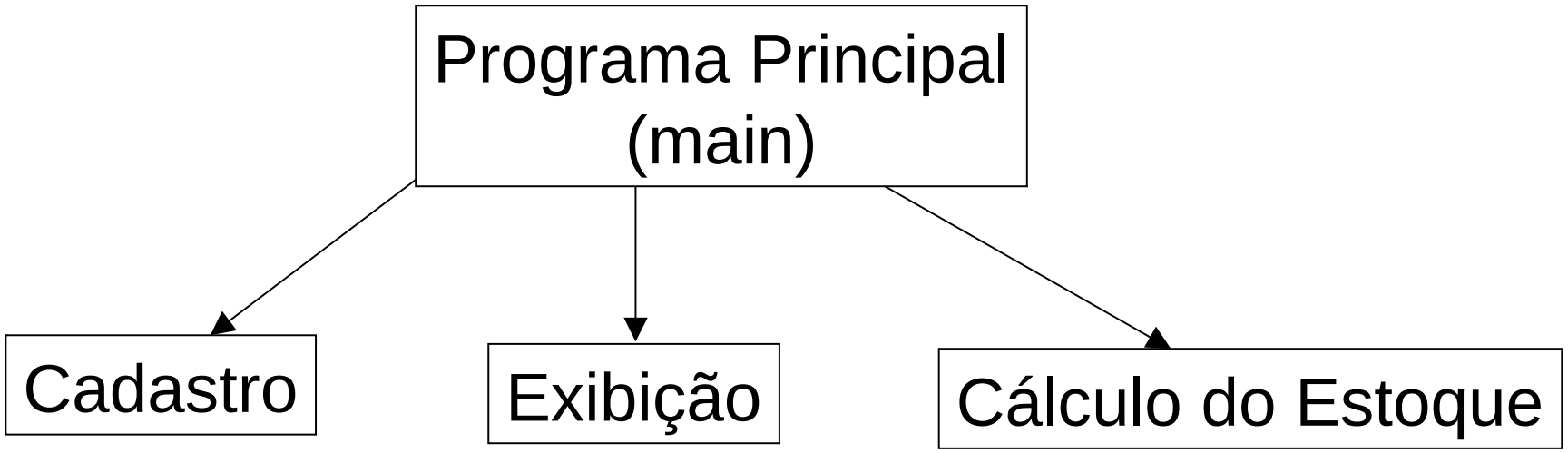
- Por exemplo, suponha que você tenha que fazer um programa que permita cadastrar produtos, exibir produtos e calcular o total do estoque.

```
Programa Principal
(main)
{
    Cadastro
    Exibição
    Cálculo do Estoque
}
```



O que é modularização?

- Em termos de módulos, cada módulo pode ser utilizado de forma independente tanto pelo programa principal quanto por outros módulos ...

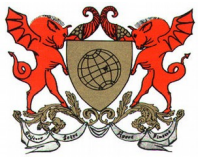


... a qualquer momento e quantas vezes for necessário.

O que é modularização?

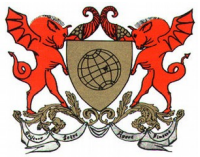
- **Modularizar** significa:

  Quebrar um problema em pequenas partes, sendo **cada uma destas**, responsável pela realização de **uma etapa do problema**.



Funções em C

- Na linguagem C, a modularização é implementada por **funções**.
- A **função main** (programa principal) é uma função especial. Ela é sempre a primeira a ser executada.
- Já usamos funções em C sem perceber e outras que sabíamos.

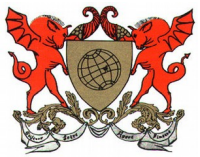


Funções em C

- O que faz o código abaixo?

```
int main()
{
   char prim_nome[20];
   printf("Digite seu primeiro nome:\n");
   scanf("%s",prim_nome);
   printf("Seu nome é %s e possui %d letras.\n",
   prim_nome, strlen(prim_nome));
   system("pause");
   return 0;
}
```

- Quantas funções ele utiliza?

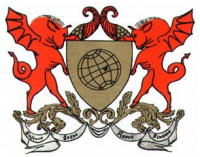


Funções em C

- O que faz o código abaixo?

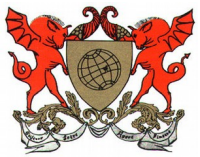
```
int main()
{
   char prim_nome[20];
   printf("Digite seu primeiro nome:\n");
   scanf("%s",prim_nome);
   printf("Seu nome é %s e possui %d letras.\n",
   prim_nome, strlen(prim_nome));
   system("pause");
   return 0;
}
```

- Quantas funções ele utiliza? 6 funções



Funções Matemáticas

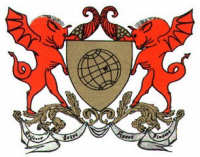
Função	descrição	Exemplo
ceil(x)	"teto" – arredonda para o menor inteiro não menor que x	ceil(10.3) é 11.0 ceil(-7.8) é -7.0
floor(x)	"piso" – arredonda para o maior inteiro não maior que x	floor(5.3) é 5.0 floor (-4.2) é -5.0
cos(x)	Cosseno de x em radianos	Cos(0.0) é 1.0
sin(x)	Seno de x em radianos	Sin(0.0) é 0
exp(x)	Exponencial e^x	exp(1) é 2.71828
fabs(x)	Valor absoluto de x	fabs(5.3) é 5.3 fabs(-4.2) é 4.2
Log(x)	Logaritmo natural de x	log(2.71828) é 1.0
Log10(x)	Logaritmo na base 10	Log10(10.0) é 1.0



Funções Matemáticas

Função	Descrição	Exemplo
pow(x,y)	X elevado a Y (x^y)	pow(2,7) é 128
sqrt(x)	Raiz quadrada de x (x deve ser um valor não negativo)	sqrt(9.0) é 3.0
fmod(x,y)	Retorna o resto da divisão de x por y em ponto flutuante	fmod(2.6, 1.2) é 0.2
tan(x)	Tangente de x (x em radianos)	tan(0) é)

Para se usar as funções matemáticas deve-se usar a biblioteca #include<math.h>



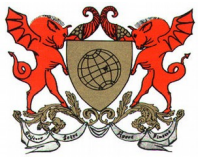
Função de números aleatórios

Na linguagem C, existe uma função para gerar números aleatórios que pode ser usada em inúmeras aplicações. Biblioteca: #include <stdlib.h>

A função é:

int rand();

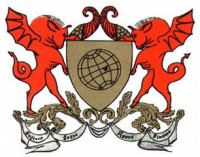
Que não recebe parâmetros e retorna um número inteiro aleatório.



Função de números aleatórios

Em geral, usa-se uma função de **tempo**, para sempre trocar a semente de aleatoriedade necessária para o retorno da **rand()** ser diferente em diferentes execuções.

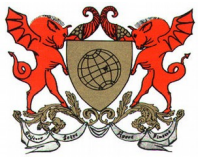
```
int main(){
    int i, n = 5;
    /* Inicializa a semente de aleatoriedade */
    srand(time(NULL));
    /* Mostra 5 números aleatórios de 0 a 49 */
    for( i = 0 ; i < n ; i++ )
       printf("%d\n", rand() % 50);
    return 0;
}
```



Funções em C

- A forma geral de uma função em C é:

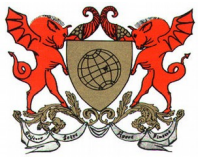
```
tipo_da_função nome_da_função(lista de parâmetros)
{
    corpo_da_função
}
```

Funções em C

```
tipo_da_função nome_da_função(lista de parâmetros)
{
    corpo_da_função
}
```

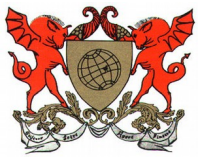
- **Tipo da função**: refere-se ao tipo de resposta que a função devolve (int, float, char, void, etc).
- Se nenhum tipo for especificado, a linguagem C assume que o retorno será do tipo int (inteiro).
- Quando não é necessário um retorno, usa-se **void.**



Funções em C

```
tipo_da_função nome_da_função(lista de parâmetros)
{
    corpo_da_função
}
```

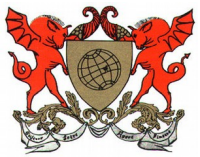
- **Nome da função:** segue as regras de nomenclatura do C, não podendo possuir espaços, caracteres especiais ou acentos e não podem ser palavras reservadas da linguagem.



Funções em C

```
tipo_da_função nome_da_função(lista de parâmetros)
{
    corpo_da_função
}
```

- É **recomendável** que o nome da função seja o mais explicativo possível sobre o seu funcionamento.
  Ex:
  - float divideDoisNumerosInteiros() {...}
  - void AlertaUsuario() { ... }

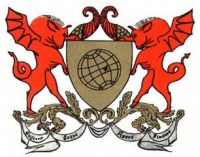


Funções em C

```
tipo_da_função nome_da_função(lista de parâmetros)
{
    corpo_da_função
}
```

- **Lista de parâmetros:** Relação de **nomes de variáveis** e **seus tipos**, para entrada de dados na função.

- Esse é um mecanismo usado para transmitir informações para uma função.

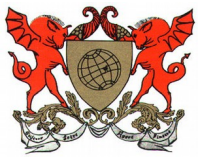


Funções em C – Exemplo 1

- Uma função que mostra uma saudação e que é chamada a partir da função principal.

```
# include <stdio.h>
void saudacao()
{
    printf("Seja bem vindo!");
}
```

// Definição da função saudacao



Funções em C – Exemplo 1

- Uma função que mostra uma saudação e que é chamada a partir da função principal.

```
# include <stdio.h>
void saudacao()
{
    printf("Seja bem vindo!");
}

int main( )
{
    saudacao(); // Chamada da função saudacao
    return 0;
}
```

// Definição da função saudacao

Funções em C – Exemplo 2

- Uma função que soma dois números inteiros.

```
# include <stdio.h>
int soma(int i1, int i2)
{
    int s;
    s = i1 + i2;
    return s;
}
int main( )
{
    int resultado;
    resultado = soma(2, 4);
    printf("\nSoma: %d", resultado);
    return 0;
}
```

// Definição da função soma

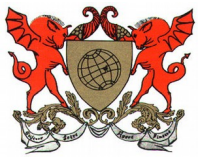
// Chamada da função soma

Definição x Chamada de Funções

- Qual a diferença entre a implementação da definição de uma função e da chamada de uma função?
- Na definição de uma função definimos os tipos de retorno da função, tipos dos parâmetros, corpo da função e o ponto-e-vírgula não é usado.

```
void saudacao()
{
   //corpo da função
}
```

```
int soma(int i1, int i2)
{
   //corpo da função
}
```

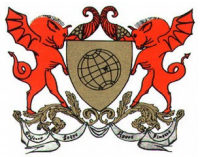


Definição x Chamada de Funções

- Qual a diferença entre a implementação da definição de uma função e da chamada de uma função?
- Já na chamada a uma função só são passados os parâmetros (variáveis ou valores) caso existem e deve ser finalizada por ponto-e-vírgula.

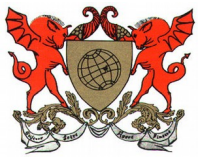
```
saudacao();
```

```
resultado = soma(2, 4);
```



Exercício

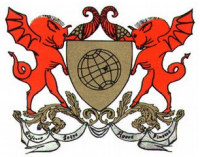
- Faça uma **função** de multiplicação de números reais. A função deve receber como parâmetros dois números **float** e retornar o valor da sua multiplicação.



Exercício

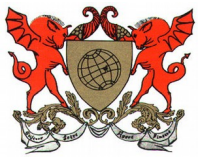
- Faça uma **função** de multiplicação de números reais. A função deve receber como parâmetros dois números **float** e retornar o valor da sua multiplicação.

```
float multiplica(float a, float b){
    return a * b;
}
```



Escopo de Variáveis

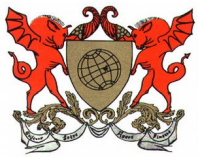
- Variáveis que são criadas dentro de uma função estão no escopo desta função.
- Variáveis no escopo de uma função só podem ser lidas e alteradas dentro da função.
- Sendo assim, uma variável no escopo da função **main** não pode ser usada por outra função.



Escopo de Variáveis

- Variáveis criadas dentro de funções são chamadas de **variáveis locais**.

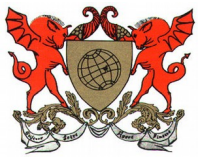
- As variáveis locais existem apenas (criadas e destruídas) durante a execução de sua respectiva função.



Escopo de Variáveis

- Entenda o programa abaixo e identifique as variáveis locais:

```
int quadrado(){
    int num, quad;
    scanf("%d", &num);
    quad = num * num;
    return quad;
}
int main(){
      int result;
      printf("Digite o numero: ");
      result = quadrado();
      printf("Quadrado do numero: %d", result);
}
```

Escopo de Variáveis

- Entenda o programa abaixo e identifique as variáveis locais:

escopo de quadrado

escopo de main

```
int quadrado(){
    int num, quad;
    scanf("%d", &num);
    quad = num * num;
    return quad;
}
int main(){
    int result;
    printf("Digite o numero: ");
    result = quadrado();
    printf("Quadrado do numero: %d", result);
}
```

Variáveis locais.

Variável local

Escopo de Variáveis

- Por que o programa abaixo não funciona?

```
#include <stdio.h>
int quadrado(){
    int num, quad;
    scanf("%d", &num);
    quad = num * num;
    return quad;
}
int main(){
    printf("Digite o numero: ");
    quadrado();
    printf("Quadrado do numero: %d", quad);
}
```

Escopo de Variáveis

- Por que o programa abaixo não funciona?

```
#include <stdio.h>
int quadrado(){
    int num, quad;
    scanf("%d", &num);
    quad = num * num;
    return quad;
}
int main(){
    printf("Digite o numero: ");
    quadrado();
    printf("Quadrado do numero: %d", quad);
}
```

qual o escopo?

não existe "quad" na função main

Escopo de Variáveis

- Por que o programa abaixo não funciona?

```
#include <stdio.h>
void quadrado(int quad){
    int num;
    scanf("%d", &num);
    quad = num * num;
}
int main(){
    int quad;
    printf("Digite o numero: ");
    quadrado(quad);
    printf("Quadrado do numero: %d", quad);
}
```

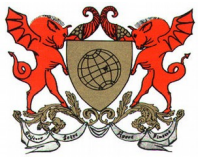
Ok. Agora declaramos a variável quad em main.

Vai funcionar?

Escopo de Variáveis

- É possível declarar **variáveis acessíveis a todas as funções.** Para isto tem que ser **criadas** fora das funções, após os **includes**.

- Estas variáveis, acessíveis a partir de todas as funções, são chamadas **variáveis globais.**



Escopo de Variáveis

- Por que o programa abaixo funciona?

```
#include <stdio.h>
int quad;
void quadrado(){
    int num;
    scanf("%d", &num);
    quad = num * num;
}
int main(){
    printf("Digite o numero: ");
    quadrado();
    printf("Quadrado do numero: %d", quad);
}
```

Variável global

Escopo de Variáveis

- Por que o programa abaixo funciona?

```
#include <stdio.h>
int quad;
void quadrado(){
    int num;
    scanf("%d", &num);
    quad = num * num;
}
int main(){
    printf("Digite o numero: ");
    quadrado();
    printf("Quadrado do numero: %d", quad);
}
```

Variável global

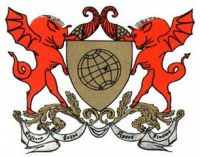
escopo de quad

Outro exemplo com Erro

```
#include <stdio.h>

float media;

void calculaMedia()
{
   media = (valor1+valor2)/2;
}

int main(void)
{
   float valor1, valor2;

   printf("Digite o valor 1 ");
   scanf("%f",&valor1);
   printf("Digite o valor 2");
   scanf("%f",&valor2);

   calculaMedia();

   printf("Media = %.1f",media);

   return 0;
}
```

calculaMedia() não pode acessar valor1 e valor2, apenas media

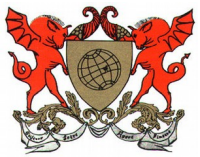
escopo de valor1 e valor2



Outro exemplo - OK

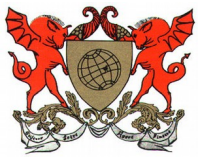
```
#include <stdio.h>

float media, valor1, valor2;

void calculaMedia()
{
   media = (valor1+valor2)/2;
}

int main(void)
{
   printf("Digite o valor 1 ");
   scanf("%f",&valor1);
   printf("Digite o valor 2");
   scanf("%f",&valor2);

   calculaMedia();

   printf("Media = %.1f",media);

   return 0;
}
```

escopo de valor1 e valor2 →



Utilizando retorno

```
#include <stdio.h>

float valor1, valor2;

float calculaMedia() {
   return (valor1+valor2)/2;
}

int main(void) {
   float media;
   printf("Digite o valor 1 ");
   scanf("%f",&valor1);
   printf("Digite o valor 2");
   scanf("%f",&valor2);

   media = calculaMedia();

   printf("Media = %.1f",media);

   return 0;
}
```



Comando **return**

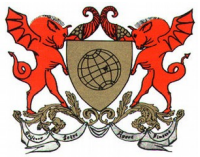
- O comando **return** tem dois usos importantes:

(1) Devolver um valor e retornar para a função que o chamou.

(2) Pode ser usado sem valores para causar uma saída imediata da função em que se encontra.

Limitações do return:

- O comando return pode retornar somente **um único valor**



Comando **return**

Por que a função imprimirMsg() não tem o comando return?

```
#include <stdio.h>
#include <conio.h>
void imprimirMsg()
{
        printf("Seja Bem vindo!");
}
int main()
{
        imprimirMsg();
        getch();
        return (0);
}
```

Comando **return**

```
#include <stdio.h>
#include <conio.h>
int hora;
void saudacoes()
{
   if(hora > 6 && hora < 12 )
   {
        printf(“Bom dia!!!”);
        return;
    }
   else if( hora >= 12 && hora < 18)
   {
        printf(“Bom tarde!!!”);
        return;
    }

   printf(“Sem saudacoes!”);
}
```

O return causa a saída imediata da função em que se encontra

```
main()
{
     printf("Digite a hora: ");
     scanf(“%d”, &hora);
     saudacoes();
}
```

Exercício

1) Faça um programa que crie variáveis para modelar "**pontos**" em coordenadas cartesianas através de structs (x,y).

- O programa deverá pedir para o usuário entrar com as coordenadas de dois pontos

- Construir uma função que calcule a distância euclidiana entre os dois pontos e mostre esta distância na tela onde (x_1,y_1) são as coordenadas do ponto 1 e (x_2,y_2) são as coordenadas do ponto 2.

$$Dist = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

* raiz = **sqrt(**numero**)** é a função para a raiz quadrada

1.1 - crie a struct correspondente

1.2 - Crie a função correspondente

```
19  int main(){
20
21      Ponto P1, P2;
22      float distancia;
23
24      printf("Digite as coordenadas x e y do ponto P1:\n");
25      scanf("%f %f", &P1.x, &P1.y);
26
27      printf("Digite as coordenadas x e y do ponto P2:\n");
28      scanf("%f %f", &P2.x, &P2.y);
29
30      distancia = calculo_distancia(P1, P2);
31
32      printf("Distancia entre os pontos P1(%0.2f, %0.2f) e ", P1.x, P1.y);
33      printf("P2(%0.2f, %0.2f):%0.2f\n", P2.x, P2.y, distancia);
34
35      system("pause");
36      return 0;
37  }
```

```c
#include <stdio.h>
#include <locale.h>
#include <math.h>

//Declarando um tipo Ponto
typedef struct sPonto{
        float x, y;
}Ponto;

//Distancia entre dois pontos
float calculo_distancia(Ponto A, Ponto B){
      float resposta, dx, dy;
      dx = A.x - B.x;//Delta X
      dy = A.y - B.y;//Delta Y
      resposta = sqrt( dx*dx + dy*dy );
      return resposta;
}

int main(){

    Ponto P1, P2;
    float distancia;

    printf("Digite as coordenadas x e y do ponto P1:\n");
    scanf("%f %f", &P1.x, &P1.y);

    printf("Digite as coordenadas x e y do ponto P2:\n");
    scanf("%f %f", &P2.x, &P2.y);

    distancia = calculo_distancia(P1, P2);

    printf("Distancia entre os pontos P1(%0.2f, %0.2f) e ", P1.x, P1.y);
    printf("P2(%0.2f, %0.2f):%0.2f\n", P2.x, P2.y, distancia);

    system("pause");
    return 0;
}
```

struct Ponto

função para calcular a distância

2) Faça um programa completo que contenha uma função que receba um número inteiro positivo **n** e retorne o seu fatorial, n!. Teste sua função.

```
n! = 1 * 2 * 3 * ... * (n-1) * n
```

2) Faça um programa completo que contenha uma função que receba um número inteiro positivo **n** e retorne o seu fatorial, n!. Teste sua função.

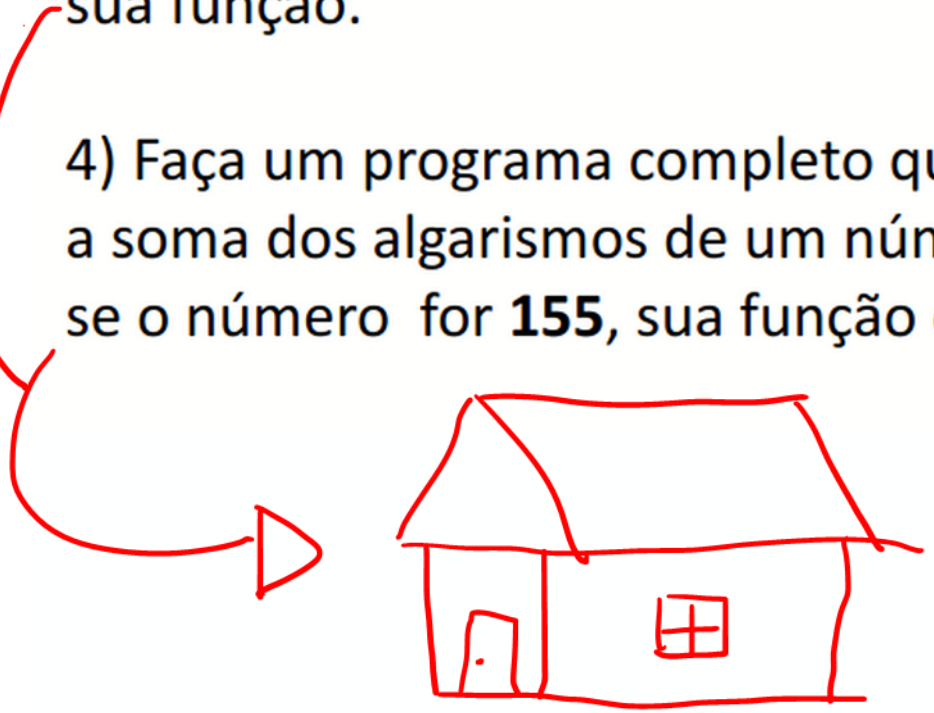
$$n! = 1 * 2 * 3 * \ldots * (n-1) * n$$

```
int fatorial(int n){
    int i, fat;
    fat = 1;
    for(i=1; i<=n; i++){
        fat *= i;
    }
    return fat;
}
```

Exercício

3) Faça um programa completo que contenha uma função que receba um número inteiro positivo n e retorne o somatório de **1 até n**. Teste sua função.

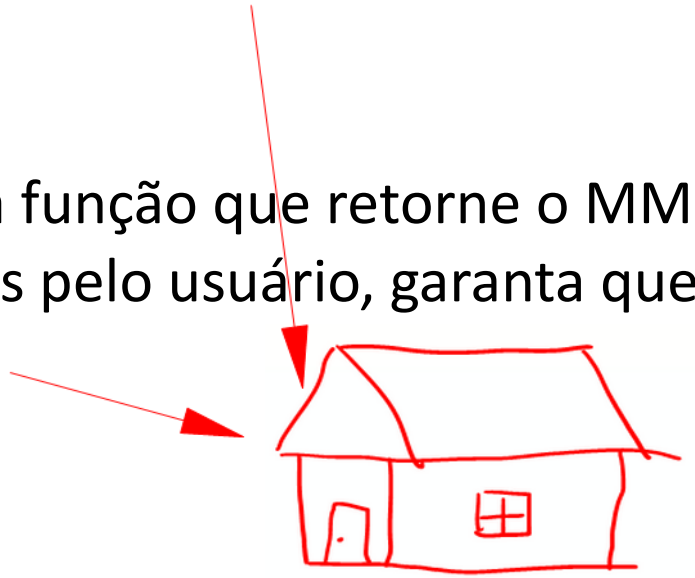
4) Faça um programa completo que contenha uma função que retorne a soma dos algarismos de um número inteiro qualquer, por exemplo, se o número for **155**, sua função deve retornar 1+5+5 = **11**.



Exercício

5) Faça um programa completo que contenha uma função que receba um número inteiro positivo **N** e retorne se o número digitado é primo ou não. De acordo com esta função, permita que o usuário digite um outro inteiro **M** e mostre na tela os M primeiros números primos. Por exemplo, se M for 5, a saída em tela deverá ser: 1, 2, 3, 5 e 7, os 5 primeiros primos.

6) Faça um programa completo com uma função que retorne o MMC e o MDC de dois números inteiros digitados pelo usuário, garanta que os números digitados sejam positivos.



Exercício

7) Faça um programa completo para manipular números racionais isto é, usando a seguinte struct:

```
//Definindo um tipo racional
typedef struct sRacional{
        int n, d; //numerador e denominador
}Racional;
```

De acordo com essa struct faça **4 funções** que recebem 2 racionais X e Y cada e retornando cada uma, a soma dos racionas, a subtração dos racionais, a multiplicação dos racionais e a divisão dos racionais.

Como seria a função para somar dois números racionais?

$$\frac{x}{y} + \frac{a}{b} = \frac{b.x + y.a}{b.y}$$

Como seria a função para somar dois números racionais?

$$\frac{x}{y} + \frac{a}{b} = \frac{b.x + y.a}{b.y}$$

```
Racional soma(Racional X, Racional Y){
    Racional R; // resultado
    R.den = X.d * Y.d;
    R.num = Y.d * X.n + X.d * Y.n;
    return R;
}
```

O resto fica para