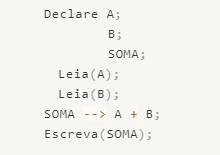
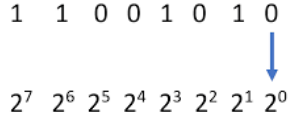
**Dia 21/02/2022**

* Algoritmos
  + Um conjunto finito de instruções que uma vez executado conduz a solução de um problema
  + Não precisa ser uma máquina ‘
* Algoritmos computacional
  + Feito para ser executado no computador
* Ambiente computacional
  + Processador: Executará o algoritmo
    - Antigamente os algoritmos eram armazenados no processador, fato que foi modificado para a memória.
  + Memoria principal (RAM): Armazena dados e algoritmos
    - Quando a máquina está desligada a memória está vazia.
  + Dispositivos de entrada (Input): Conversa entre o mundo externo e computacional - Teclado
  + Dispositivos de saída (Output): Conversa entre o mundo externo e computacional - Tela
  + I/O: Input e Output – HD
  + Existe uma via que liga o processador e a memória para que haja uma comunicação entre eles.
* Primeiro algoritmo – soma de dois reais



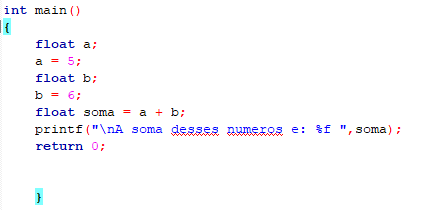
**Aula 23/02**

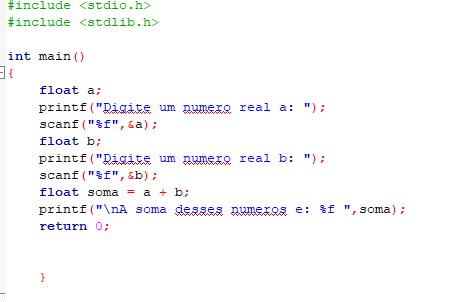
* Memória secundária = disco – armazenamento de dados (qualquer coisa que pode guardar um número binário pode virar uma memória secundária)
* Recursividade = infinito – exemplo da câmera que projeta ela mesmo.
* Para a liberação de um software é necessário uma documentação e testes
* C++ = linguagem c + orientação de objetos – criação nos anos 80.
* Livro: Fundamentos da programação de computadores Ascencio Ana Fernanda Gomes Sp Pearson 2012
* Representação do dado na forma digital:
  + Analógico = ondas
  + Digital = bytes
  + “A” à 01000001 (ASCII) – American Standard Code for Information Interchange – 256 à 0 a 255
  + “B” à 01000010
  + Bit: Binary digit
  + 8 bits = 1 byte (Binary term)
  + Transmissão = bit (internet)
  + Armazenamento = byte(memória)
  + Código binário: 1 2 4 8 16 32 64 128

Ex: 0100001 = 64 + 1 = 65

* Tipo primitivo de dado:
  + Numérico à Inteiro e Real
  + Alfanumérico à Caractere e String(cadeia de caracteres)

**Aula dia 04/03**

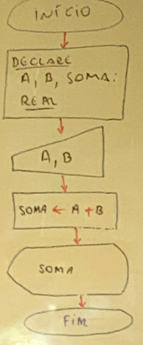
* Uso de bibliotecas são extremamente importantes para o uso de comandos:
  + #include <stdio.h>
  + #include <stdlib.h>
* Leitura de dados em C:
  + HD é tratado em blocos de dados, já a memória RAM é tratada byte por byte
  + Número real na memória = 8 bites
  + Número inteiro na memória = 4 bites
  + Soma de 2 (6 e 5 ) números:
  + Uso de scanf:
    - Scanf(“TIPO VARIAVEL”, &X);
    - &X = local da memória que a variável foi armazenada.
  + Soma de 2 números qualquer: (Uso de scanf):

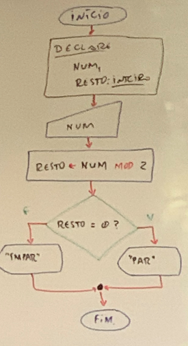


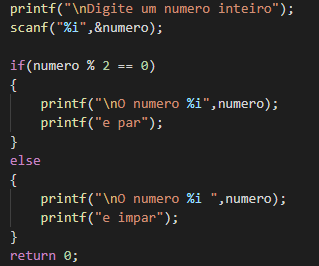
**Aula dia 07/03/2022**

* Para a armazenamento de dados é necessário que o tipo seja especificado além da sua localização na memoria
* Na memória existe uma divisão para dados e a outra para algoritmos.
* Tipos de dados:
  + Float %f
  + Int %i
  + Char %c
* Buffer - \n
* Dica: Todo programa a ser feito é necessário um tempo antes de inicia-lo para arquitetar o código.

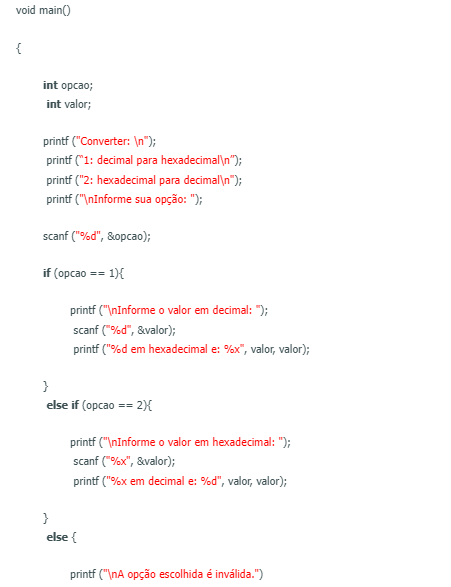
**Aula dia 09/03/2022**

* Fluxograma
  + Sempre terá inicio e fim
  + Fluxo das instruções – reflete a ideia de algoritmos
  + Imagem no celular do fluxograma
  + Mais didático
  + EX:
  + ****
* Estrutura de seleção/decisão
  + Problema: inteiro é par ou impar ímpar – fluxograma no celular:

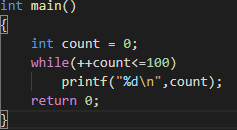
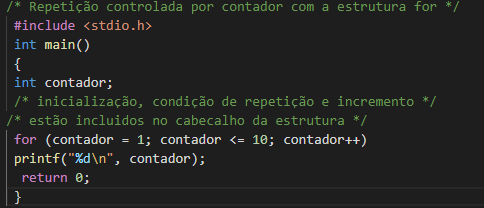


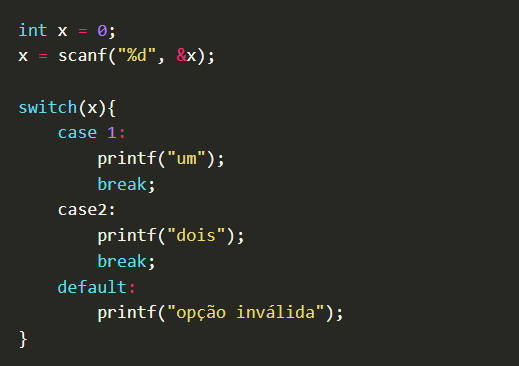
* + Entrada: Valor inteiro qualquer
  + Processamento:
    - Num/2 com resto 0 = PAR
    - Num/2 com resto diferente que 0 = IMPAR
    - MOD = resto da divisão(%)
  + Saída: “Par ou Ímpar”
  + Formato IF and ELSE:
* Operadores Booleados em C:
  + And = &&
  + Or = ||
  + Not = !

**Estudo Individual 13/03/2022**

* Estrutura de if else else if:
* obs: As expressões condicionais serão avaliadas de cima para baixo. Assim que uma condição verdadeira é encontrada, o bloco associado a ela será executado, e o resto do encadeamento é ignorado. Se nenhuma das condições for verdadeira, então o else final será executado.

Se o else final não estiver presente e todas as outras condições forem falsas, então nenhuma ação será realizada

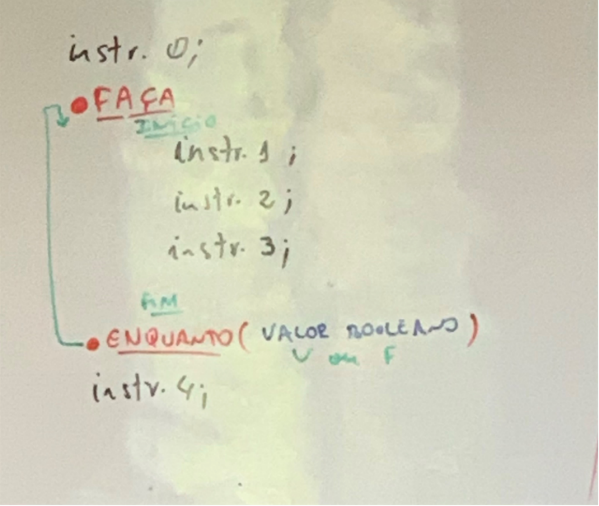
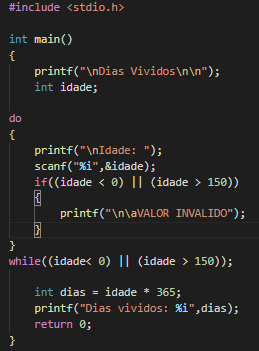
* A estrutura de while:
* OBS: Diferenca de %d e %i(Output):
  + No output não existe diferença, já no input de dados pelo usuário:
  + %i –permite o usuário informar valores na base de 8, 10 e 16.
  + %d implica que o valor digitado pelo usuário será interpretado como um número inteiro decimal.
* Estrutura de for:
* Estrutura de Switch:
  + Melhorar a legibilidade do código, para não ter que ficar repetindo if/else

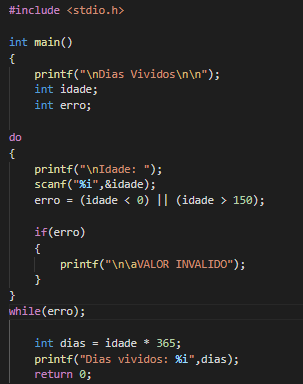


**Aula dia 14/03/2022**

* Valor booleando sempre resultara em V ou F
* Se a instrução do if for unitária não é necessário colocar entre {}.
* Operadores relacionais em C:
  + >
  + >=
  + <
  + <=
  + == ou =(atribuição)
  + <> ou !=
* Operadores aritmética em C:
  + +
  + –
  + /
  + %
  + \*

**Aula dia 16/03/2022**

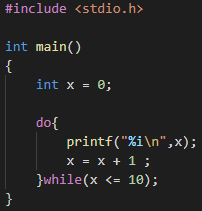
* Sempre antes de começar a programar pensar:
  + Entrada
  + Processamento
  + Saída
* Scanf(“%c”, &x) – Para usar esse comando com um caractere é necessário que haja um espaçamento antes da %. – Scanf(“ %c”,&x). – Erro comum(buffer).
* Pode usar scanf com mais de um valor? Sim. EX:
  + Declarar os tipos dos dados dentro de uma string.
  + Declara os caminhos na memória dos dados, separados por vírgula.
* Estruturas de repetição:
  + OBS: Existiam línguas que usavam o “go to”, o que é extremamente ineficiente, o qual impossibilita a adição de outras linhas de código.
  + **Com teste no final (do....while... ou do/while):**
    - ****Sintaxe:
    - Usada quando queremos que o bloco de código seja executado pelo menos uma vez (grande diferença entre do e while)
    - Exemplo:
    - ****Boas práticas – versão melhor: - Uso de bool – Semântica melhor



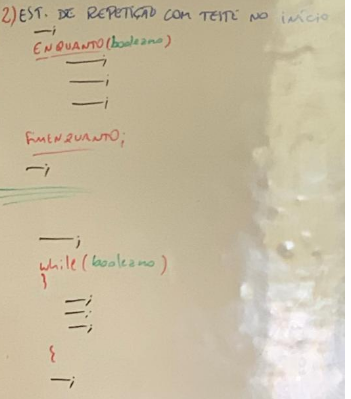
Aula dia 18/03

* Declarando um bool:
  + É necessário usa a biblioteca <stdbool.h>

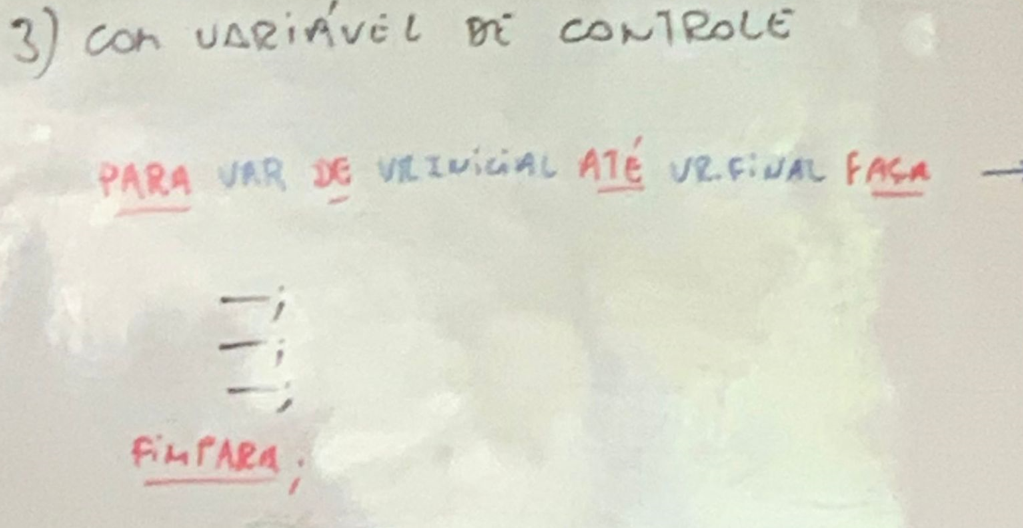
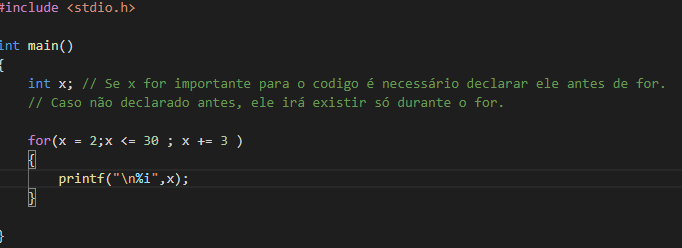
Aula dia 21/03/2022

* Contadores e Acumuladores:
  + Usa-se a ideia de adicionar ou subtrair um certo número da variável, juntamente com alguma estrutura de repetição.
  + Exemplo (contador de 1 a 10):
  + Exemplos para fazer o mesmo:
    - X ++ i;
    - X +=1;
    - X = X + 1;
    - X = 1;

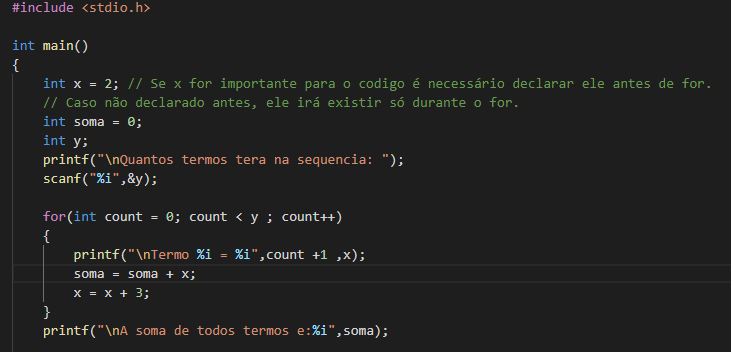
Aula dia 23/03/2022

* Estruturas de repetição:
  + **Com teste no início(while):**
    - Sintaxe
    - Diferença entre while e do/while é o fato de que o while voltará para o teste acima até que ele seja falso. Além disso, o do/while executara o código no mínimo 1 vez.

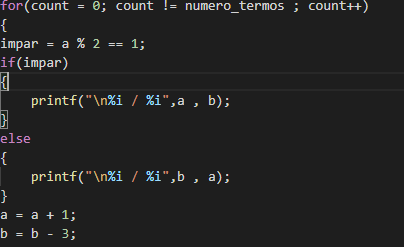
Aula 28/03/2021 – Arquivo corrompido

* **Estrutura de repetição com variável de controle(for)**:
  + Início e fim bem definidos.
  + Sintaxe:
  + Exemplo:

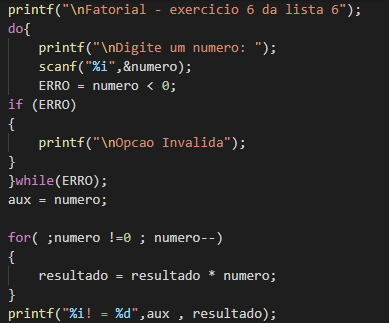
Aula dia 30/03

* Padrão de comportamento.
  + É necessário descobrir o padrão de comportamento para depois codifica-lo.
  + EX:
  + Variável count, no exemplo, será a variável de controle, ou seja, ela so existe durante o bloco de código for. Caso, ela fosse importante para depois, seria necessário declara-la no início do programa.
  + Divisão real:
    - Float/ float – Basta que um dos números seja real(float).
    - Muitas vezes se usa 1.0 para transformar um inteiro definido em float.
    - Caso der erro na divisão (sempre ser 0) é necessário conferir o tipo de dado que está sendo dividido.
  + Biblioteca <math.h>:
    - Funções matemáticas
      * pow(base , expoente); - potencia
      * Sqrt()- raiz quadrada

Aula dia 04/04/2022

* Às vezes é necessário usar a identificação de par ou ímpar para manter o padrão do código desejado. Ex: (ex 2 da lista 6)
* \a – usado para alertar o usuario, normamelte usado na variavel ERRO.
* Para arredondar casa decimal, usa-se %.Xf, sendo que x é o número de casa decimais desejado. Ex:
* FLAG = valor para sair do programa ou sair da função.

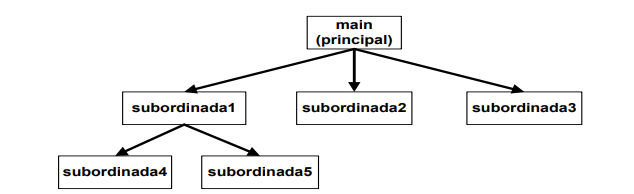
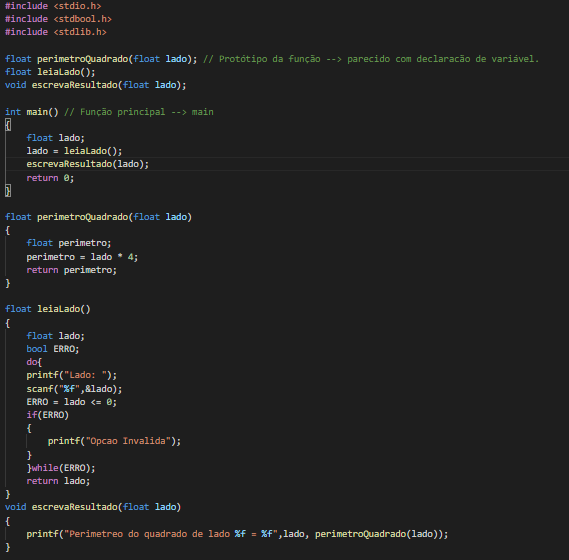
Aula dia 06/04/2022

* O número inteiro tem um limite máximo que será armazenado, caso deseja-se aumentar esse limite é necessário usar DOUBLE ou LONG.
* Sempre lembrar do **alinhamento** correto no código, entre begin e end é necessário jogar para direita. Ex:
* Uso do for à quando o **inicio e o fim estiver bem definido.**
* While à quando você quiser **preservar** uma condição.
* Exemplo acima usou for incorreto, pois na verdade o while nesse caso seria mais adequado.
* Toupper(x) = comando para converter x para maiúscula.

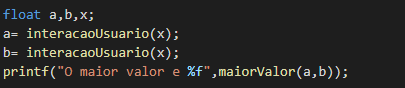
Aula dia 13/04/2022

* Não terá aula no dia 18/04 e 20/04 à reposição 29/04 e 6/05.

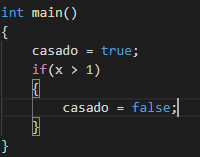
Aula dia 25/04/2022

* Funções
  + No mesmo arquivo existirá diversas funções, que serão chamadas por uma, porém existe a possibilidade de uma função “secundaria” chamar outra do mesmo tipo.
  + Função “main” à Primeira a ser chamada e executada (apenas uma no arquivo).
  + Em computação se a palavra da frente começar com BRH tem hífen.
  + Sub-rotinaà Uma função vai chamando a outra. Não é um modelo **hierárquico** e sim um modelo **democrático** (todo mundo pode chamar todos.
    - Tipos:
      * 1- Procedimentoà É uma sub-rotina que **NÃO** gera valor. Ganho: Reutilização de ações. Facilidade de manutenção.
      * 2- Funçãoà É uma sub-rotina que **GERA** valor, ou seja, é necessário que devolva alguma coisa para o usuário.
    - No mundo C:
      * Procedimento = Função vazia, uso do void. Linguagem C não separa os dois, porém o procedimento vira uma função vazia.
      * As variáveis são associadas as sub-rotinas, ou seja, é possível criar duas variáveis **DIFERENTES** com o mesmo nome.
      * Variável pode ser copiada e passada entre sub-rotinas, chamadas de **VARIAVEIS PARAMÉTRICAS**.
    - Definindo uma função:
      * O formato de uma definição de função é tipo-do-valor-de-retorno nome-da-função (lista-de-parâmetros) {declarações instruções}
      * Exemplo:
* OBS:
  + Sempre será necessário fazer o protótipo.

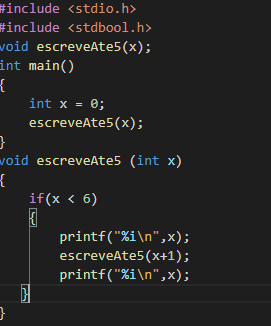
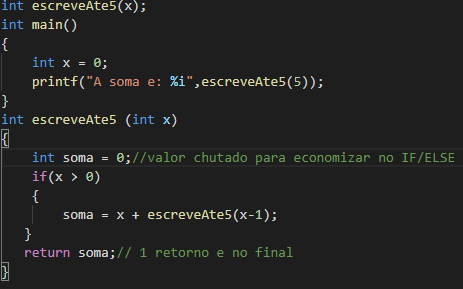
Aula dia 27/04/2022

* A variável local sempre será priorizada, muitas vezes, gerando conflito com as variáveis globaisàDICA: sempre priorizar a variável local.
* Melhor nome para funçãoàQual será o valor retornado por ela.
  + Usar verbo no imperativo **APENAS** quando a função for void.
* Quanto **menos testes é melhor para seu código** (IF/ELSE).
* É possível atribuir uma variável, o qual é definida por uma função.EX:

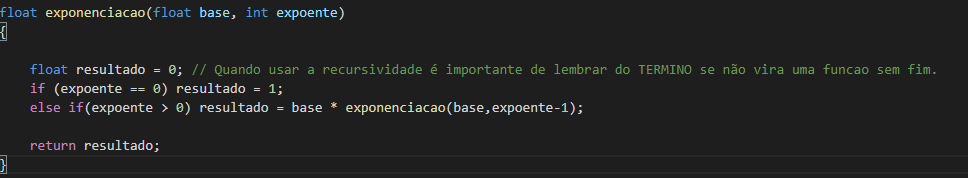
**Aula dia 29/04/2022**

* O nome da variável é importante. **DICA: evitar verbo!**
* Não é certo usar printf e scanf dentro da função, e sim criar funções que façam isso, leiaReal, escrevaReal.
* Quando declarada a função você já pensa na função. Ex:
  + Float pesoIdeal (char sexo , float h)
    - Neste exemplo já foi pensando que a função vai retornar um número real.
    - Sendo que um char e um float foram parâmetros. (mochila do herói)
* Não se pode criar **dependência** entre a função e chamada, ou função e função. Queremos fazer uma função que se encaixará em diferentes cenários, ou seja, não pode depender da int main().
* **DICA: Chutar valor para melhorar sintaxe em condições.EX:**
* Grau de acoplamento à grau de dependência.
* Quando se usa FLAG à Pede número no início e no final. (WHILE).
* Recursividade: à Uma função chama a si mesmo criando um ciclo.
  + Ciclo infinito.
  + Consome muita memória, pois cada função criará um espaço na memória
  + Estoura a memória.
  + Repetição e Recursividade são diferentes, mesmo parecendo serem iguais.
  + X = x++ à x será x (não irá ser somado)
  + X = ++x àx sera x + 1 (obriga a soma)
  + Return àVolta para quem chamou.

**Aula dia 02/05/2022**

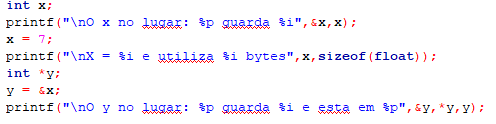
* Todo início de algoritmo devemos pensar se iremos ou não usar recursão.
* Uma **função recursiva** é aquela que tem uma ou mais chamadas a si mesmo.EX:
* A função voltará no escreveAte5, ou seja, o return volta pra quem chamou ele.
* Soma de dois números com uma função recursiva:
* Um **problema recursivo** é aquele cuja solução permite modela-la em termos de si mesma.EX:
  + Somátorio de 1 até n.
    - Caso indutivo à n + somatório(n-1)
    - Caso base à n igual a zero, gera zero.
* Dentro da função é bom ter apenas **1 return e normalmente no final do bloco.**
* Arvore de recorrência:
  + Foto no celular

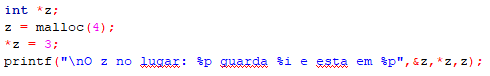
**Aula dia 04/05/2022**

* Exemplo de caso base e caso indutivo:
  + 2^4 = 2 \* 2^3 à caso indutivo
  + 2^3 = 2 \* 2^2
  + 2^2 = 2 \* 2^1
  + 2^1 = 2 \*2^0
  + 2^0 = 1 à caso base
* ****Potenciação recursiva:
* OBS: Lembrar de colocar um fim na recursividade.
* **Passagem de parâmetro por referência:**
  + Variável não recebe o valor e sim a referencia
  + Cada linguagem é diferente.
  + Em C:
    - Realizada por ponteiros. (\* antes do nome da variável)
    - **Ponteiros** à Aponta para um lugar já existente na memória.
      * àPode-se colocar qualquer área da memória (Por esse motivo que a maioria dos vírus são feitos em C)
      * àÉ necessário entregar um endereço e não um valor, usa-se &
      * àPara recuperar o lugar apontado usa-se \* antes da variável.
      * àvoid troca(int \*a , int \*b)
      * Para declarar é preciso indicar que tipo de dado está sendo apontado.

**Aula dia 06/05/2022**

* Usa o %p para recuperar o lugar exato da memória que a variável está armazenada e &x. EX:
* Função **sizeof**() é utilizada para descobrir quantos bytes o **TIPO** do dado ocupa da memória.
  + Tipos de dados têm tamanhos diferentes.
    - Int = 4 bytes
    - Double = 8 bytes
    - Float = 4 bytes
* Exemplo uso ponteiro e posições na memória:



* **OBS:**
  + É necessário declarar o ponteiro, usando \*.
  + &y = Lugar onde a variável está armazenada.
  + \*y = Valor de x, pois y aponta para x.
  + y= Lugar onde X está armazenado. , pois y = &x.
* Para criar área na memória com a função **malloc(numerodebytes)**:
  + Devolve o endereço do **PRIMEIRO** byte.
  + 
  + Pode-se usar malloc juntamente com sizeof.:
* **Introdução a arquivos em C**:
  + Fluxo de entrada(teclado) e saída(tela) à i/o
  + Memoria secundaria à Qualquer dispositivo de armazenamento.
    - Pode ser os dois fluxos.
    - Tudo digitado vai para o buffer, que armazena.
    - É possível mandar valores para uma memória secundaria, para assim ser armazenado.
    - Necessário da biblioteca à #include <stdio.h>, mesma usada para scanf e printf
    - É preciso criar uma variável para apontar para o lugar na memória que o buffer estar, ou seja, um ponteiro para buffer.
      * Não se pode criar uma dependência entre o nome da variável e do arquivo.
    - Para fazer:

FILE \*arq;

Arq = **fopen(“NOME DO AQUIVO”,”MODO”)**

* OBS
  + Fopen() à abrir algum arquivo
  + Uso de aspas duplas, independentemente do tamanho.
  + O nome do arquivo pode ser uma variável.
  + Fprintf(arq, “18”)
  + Fclose(arq) àfecha o fluxo.
* Modos de abertura:

1. W àwrite, apenas escrita a partir do inicio , cria arquivo
2. A àappend, apenas p escrita no final,
3. R àread, apenas para leitura.
4. W+àacrescenta a leitura, cria o arquivo
5. A+àacrescenta a leitura,
6. R+ àler e escrever arquivo existente àsobrepondo informações já existentes .

**Aula dia 09/05/2022**

* Variáveis para alcançar algum arquivo é **declarada em main**, pois se for dentro de outra função ela será temporária, usando letras maiúsculas.
* Desenvolvimento do arquivo media\_idade\_da\_turma.c à questões aula.
* BOA PRATICA àAbrir e fechar o arquivo.

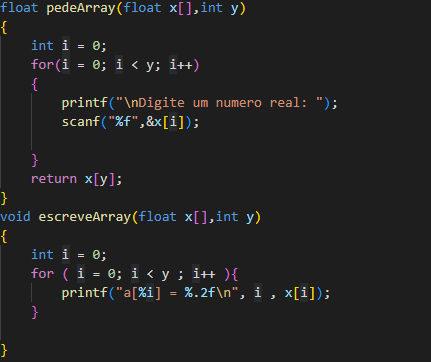
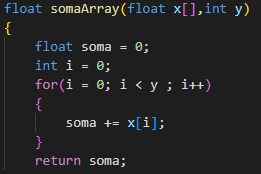
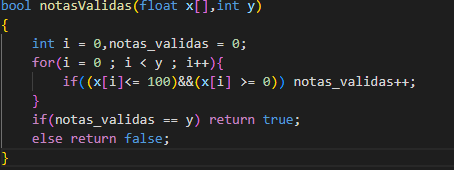
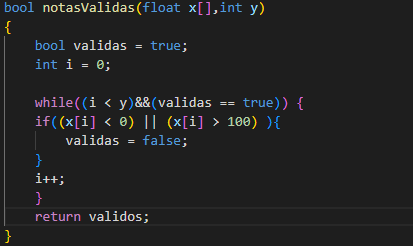
**Aula dia 11/05/2022**

* O arquivo não precisa ser necessariamente criado na linguagem c.
* Arquivo só se passa por **REFERÊNCIA**
* Quando existe a necessidade de voltar para o início:
  + Fechar e abrir o arquivo.
  + Usa-se **rewind**(**NOME DO ARQUIVO**). àVoltar para o inicio
  + ****

**Aula dia 13/05/2022**

* **Arranjos/Vetor/Array em C:**
  + Vizinhos na memória
  + Mesmo tipo – diferentemente do que em outras línguas, assim definindo o tamanho (qntd de bytes)
  + Precisa especificar o tamanho para declara-lo. (prós e contras)
  + Posições: (**SEMPRE COMEÇA EM 0**)
  + EX para declarar:
    - Int a [TAMANHO] à int[3]
    - A[0] = 7;
    - A[1] = 8;
    - A[2] = 9
  + Percorrendo um vetor com for: (foto celular)
  + Para usar em uma função
    - Void leiAarranjo(int x []) à não precisa declarar tamanho/ **passagem por referência**, ou seja, sempre mandara o endereço.
* Desenvolvimento do arquivo:

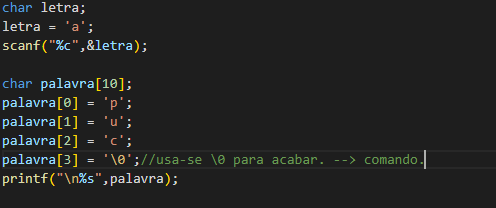
**Aula dia 16/05/2022**

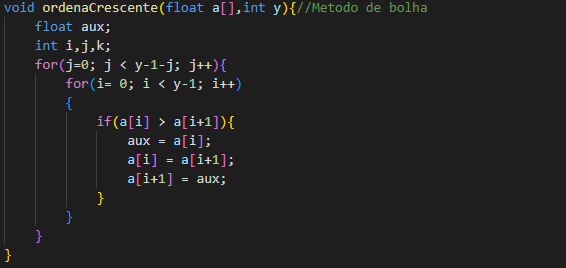
* Correção da lista 11.
* Exemplo de duas funções com arranjo para pedir e escrever um arranjo:
* A linguagem C não importa se tentar alcançar uma posição inexistente, entretanto retornará lixo.
* Vantagem array à Nem sempre usando variável você consegue usar dentro de um laço, pois seriam necessárias várias variáveis, já com array é possível. EX: soma no array.
* **OBS :** Quando for uma função bool, nomear ela pra quando o valro for verdadeiro: notasValidas, valorValidos , maiorQueSete...
  + **Melhorando o código:** àDessa forma a finção testará TODOS os cenarios possiveis
  + No segundo caso, a função testará enquanto achar um valor invalido, assim transformando o codigo em mais barato em questao de memoria
  + Quando tiver que percorrer tudo àusar for ... Quando não tiver àusar while.
* É melhor criar uma dependência entre duas funções do que criar dependência entre a main e uma função.

**Aula dia 18/05/2022**

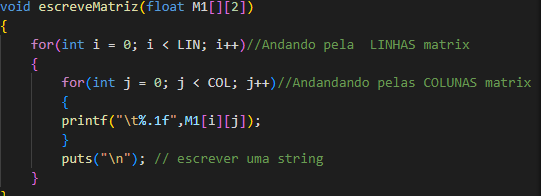
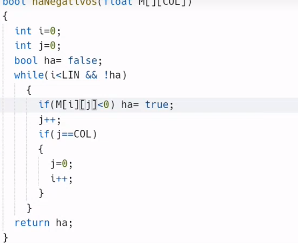
* A variável que está no parâmetro de uma função, é diferente do que a declarada da main, mesmo que elas têm o mesmo nome.

**Aula dia 23/05/2022**

* **Arranjos e strings:**
  + Int a [3] = {7,8,9}
    - A[0] = 7
    - A[1] = 8
    - A[2] = 9
  + Int a [] = {7,8,9} àNão precisa dizer o tamanho
    - A[0] = 7
    - A[1] = 8
    - A[2] = 9
  + Em c a string é um vetor de caracteres
  + ****Para usar no código:
    - **Usa-se:**
      * Scanf(“%s”, palavra) àpara ler .
      * Gets(palavra) àpara ler
      * Fflush(stdin) àLimpar buffer
      * 

****

**Aula dia 27/05**

* **Arranjos bidimensionais – Matrizes:**
  + Feito por colunas e linhas – posição exata
  + Para declarar usa-se [] []. Ex 
  + Não precisa colocar o número de linhas , mas sim o numero de colunas , pois os dados são armazenados um ao lado do outro na memória física.
  + Para percorrer a matrix usa-se-se dois laços for. Ex:
  + Desenvolvimento do arquivo:
  + Percorrendo matriz usando while: àPara economizar potência do programa

**Aula dia 30/05**

* Desenvolvimento e correção da lista 13

**Estudo Individual – 30/05**

* Funções:
  + A chamada da função especifica o nome da mesma e seus argumentos.
  + A função main se comunica com várias funções subordinadas.
  + Todas funções matemáticas retornam um double.
  + Sempre preciso de fazer o **protótipo** da função antes da main à composto por: O formato de uma definição de função é

tipo-do-valor-de-retorno nome-da-função (lista-de-parâmetros) {

declarações instruções

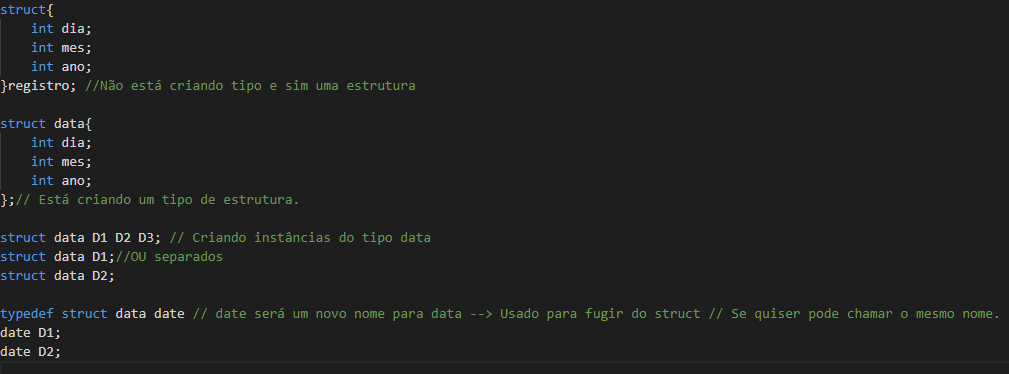
}

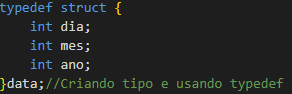
* Arrays:
  + Um array é um grupo de locais da memória relacionados pelo fato de que todos têm o mesmo nome e o mesmo tipo. Para fazer referência a um determinado local ou elemento no array, especificamos o nome do array e o número da posição daquele elemento no array.
  + Declarando: int c[12];
  + Para rodar uma array toda usa-se while ou for.
  + Usa-se #define TAMANHO 10 , para definir antes o tamanho e ficar constante.
  + Uma string em c é na verdade um array de caracteres isolados.
  + Declarando: char string1[] = “primeiro”;

**Aula dia 01/06/2022**

* Variável global à fora da função.(todo mundo acessa) #define ou const NOMEDAVARIAVEL VALOR
* Variável local à dentro de uma função
* Diferença define x const:
  + Tempo de compilação e tempo de execução

**Aula dia 10/06/2022**

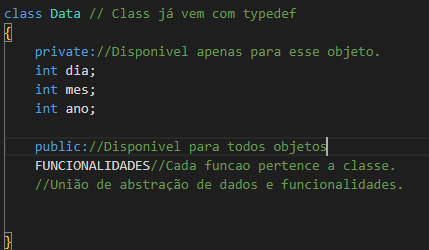
* **Registro e estruturas:**
  + Conjunto de variáveis, denominado Struct(C).
  + Permite subir o grau de abstração e trazer semântica a esse nivel de dados
  + Criar tipos --> Exemplo data DIA MES ANO
  + Normalmente cria-se um arranjo para guardar cada estruturas
  + Para recuperar DIA usa-se . :
    - NOMEARRANJO[POSICAO]**.DIA**
  + Criando estruturas:
  + Normalmente são definidos fora de qualquer função, pois é um tipo.
  + Desenvolvimento do arquivo lista15.c
  + Usando typedef:



Aula dia 13/06/2022

* **Registro e estruturas**:
  + Para organizar o início do programa, cabeçalho, normalmente usa arquivos.c, sem a função main àfacilita a manutenção.
  + #include “data.h” à mesma coisa de copia e cola, diferentemente de <data.h>, que faz referencia ao arquivo que se encontra na library
  + Diferença gets e scanf(%s):
    - Gets à Le até o enter
    - Scanf àLe até o espaço em branco.
  + Ponteiro para estruturas:
    - Pessoa\* P àNão cria a estrutura e sim aponta, desse modo é necessário cria-la com malloc.
    - EX:
    - Como recuperar/definir usando ponteiros:

**Aula dia 15/06/2022**

* **Uma Introdução à Orientação por Objetos(C++)**
  + Software industrial necessita de orientação por objetos, ou seja, programas mais complexos
  + Elementos: Abstração de dados + Abstração de operações(funcionalidades/algoritmos).
  + Classeà Para em um mesmo lugar ser feitos os dois elementos.
  + Objetos:
    - Elementos do mundo real
    - Identificação, descrição, representação controlados por diferentes graus de **abstração,** ou seja, o que muda é GRAU DE ABSTRAÇÃO
    - Ambos elementos.
    - Sempre quando formos construir um programa temos que pensar em identificar os objetos.
    - Todo objeto é uma instância de uma classe.
    - Tem dados + funcionalidades
  + Classe de objetos:
    - Uma classe descreve um objeto por ambos:
      * Dados(atributos)
      * Operações(funcionalidades)
    - Struct x Classe à Struct só é possível colocar os dados.
  + Descrevendo data como uma classe:
  + Princípios da OO:
    - Abstração
    - Encapsulamento à Public e Private
    - Generalização(herança) àGeneralização de pontos comuns EX: pessoas pessoas físicas e pessoas jurídicas(superclasse e subclasse)
    - Polimorfismo à Diferentes formas para um mesmo objetivo. EX área figuras geométricas.

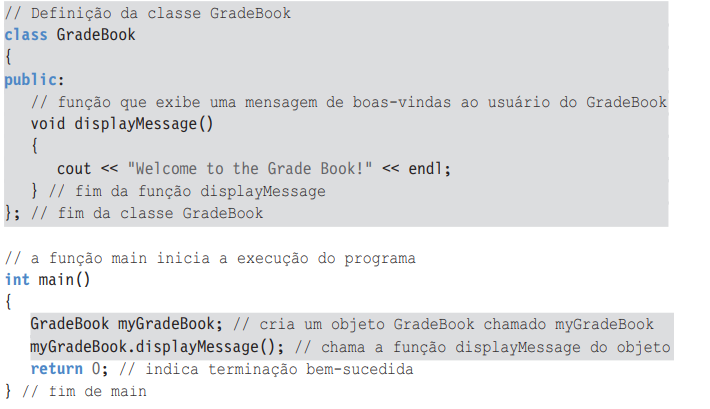
OBS: 3,4 são **exclusivos** da OO

* + Interação entre o público e o privado
    - Mudar tudo pra público (NÃO RECOMENDADO)
    - Funcionalidades atômicas àMandar e recuperar para o privado.
      * Métodos sets e gets.

Anotações Gabi dia 17/06/2022

* O struct, por definição, é público, e a classe é privada.
* Há situações que o set vai ser booleano, caso a operação seja válida (ex: se não tiver espaço na memória a operação vai ser falsa)
* Get sempre retorna algo
* As operações set e get devem ser públicas na própria classe, para que possam ser evocadas depois = Mensagem
* This = posição da memória que o objeto está
* por isso precisa da -> (ponteiro)
* Desenvolvimento do arquivo: 

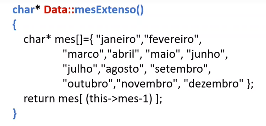
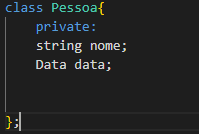
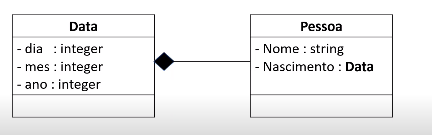
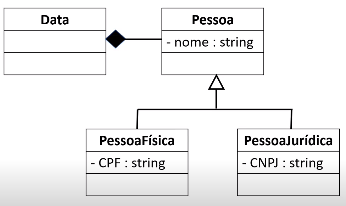
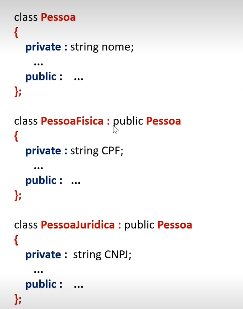
Estudo Individual Introdução a classes e objetos 20/06/2022

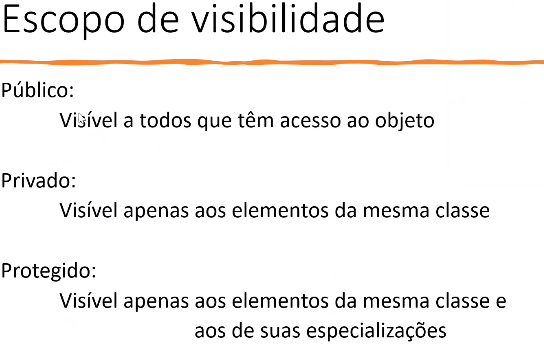
* Definindo uma classe e um objeto:
* As classes costumam fornecer funções-membro public para permitir a clientes da classe configurar (set, isto é, atribuir valores a) ou obter (get, isto é, obter valores de) membros de dados private. Os nomes dessas funções-membro não precisam iniciar com set ou get, mas essa convenção de atribuição de nomes é comum.

Aula dia 22/06/2022 – Objetos com alocação dinâmica

* Função construtora em C++
  + Não precisa de tipo, por ser construtora
  + Para criar uma destrutora apenas se cola o ~
  + Metodo SEM TIPO COM O MESMO NOME DA CLASSE
* C++
  + << = operador de inserção
  + Endl = \n
  + Using namespace std à
    - Toda vez que usar suas funções precisa usar declarar EX: std::cout<<”Oi Tudo bem?”<<end1;
    - Quando se usa essa linha não precisa mais.
  + Pode usar todos comandos de C
* Para usar o ponteiro de uma classe criado é necessário alocar espaço.Em C usamos MALLOC já em C++ usamos NEW. EX:
* Para trabalharmos com ponteiro usamos -> para chamar funções ex:
* Linguagem C++ àhíbrida, une OPP com normal.
* O this ponteiro é um ponteiro acessível somente dentro das funções de membro não estático de um class ou struct union tipo. Ele aponta para o objeto para o qual a função de membro é chamada. As funções de membro estático não têm um this ponteiro. à this evita conflito por nomes

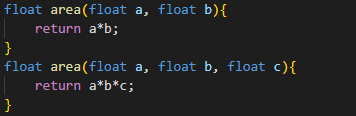
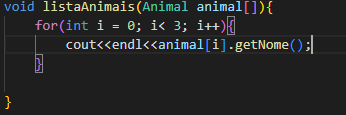
Aula dia 24/06/2022

* C++ permite que as funções-membros possam ser descritas fora do escopo da classe. à para resgar:
* Em C++ , string é tipo primitivo. à usar getline(cin,nome); para guardar duas palavras
* Tamanho padrão da string = 32bytes, memória se aloca automaticamente.
* Composição:
  + Um objeto formado por vários outros.
  + Exemplo: data e pessoa
  + Ganho de OO à pode usar de novo.
  + Relacionamento do topo : has a (UML)
  + - = private
  + + = public
* Generalização:
  + Uma classe pode generalizar as propriedades comuns de outras
  + Uma classe pode se especializar em outras
  + Uma classe pode herdar as propriedades descritas em outras
  + Relacionamento de tipo: is a
* Superclasses e subclasses
* Como herdar as operações à apenas acrescentar o que é novo:

Aula dia 27/06/2022

* Desenvolvimento arquivo : (COLOCAR) **herança.c**
* Herança publica: Tudo herdado vira público na subclasse
* Herança privada: Tudo herdado vira privado na subclasse.

Aula dia 29/06/2022

* Polimorfismo
  + Sobrecarga de funções, operadores, ... ( mesmo nome).
  + Polimorfismo fraco: em tempo de compilação, pois esta brevemente definido.
  + Funções sobrecarregadas: mesmo nome, leve diferença na assinatura.
    - Assinatura à parâmetros, seja o nome ou tipo, ou quantidade.
  + EXEMPLO:
  + Sobrecarga de construtores:
    - Estamos autorizando diferentes maneiras de se construir um objeto.
    - Para isto, basta implementar um construtor para cada maneira que se pretende autorizar a criação.
    - EX:
    - OBS: Duas formas diferentes de se criar um objeto da classe CACHORRO.
  + Se tenho um arranjo de um tipo, podemos usar subtipos nele
  + Ex:
  + OBS à Podemos usar listaAnmais(cachorro), pois é um subtipo de animal --> Polimorfismo FORTE
  + OBS 2 à Existem limites, quando a operação ou atributo for especifico de uma subclasse.
  + Função virtual à
    - pode ser especializada na subclasse e considerada! (No mundo C++ , Já no JAVA todas são virtuais).
    - Aquela que está autorizando especializar. --> prioridade da superclasse
    - EX; falaAnimal()

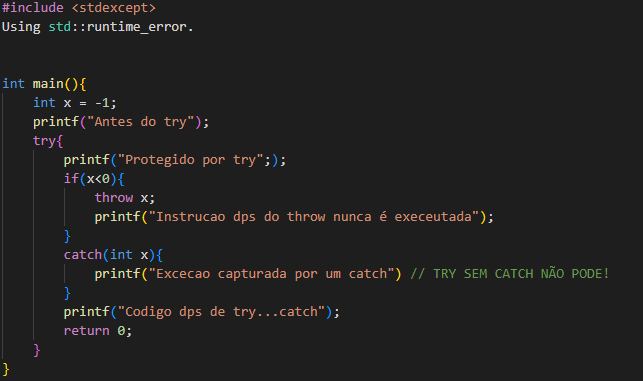
Aula dia 01/07/2022

* 

OBS: Esta função irá funcionar tanto para um arranjo de cachorro, quanto de gatos, pois ambos são sub classes de Animal!

* Função abstrata:
  + Não pode ser instanciada
  + UML --> itálico
* Método abstrato:
  + Não tem corpo, apenas assinatura
  + Vai ganhar corpo apenas quando for especializada
* Classe concreta:
  + Não pode ter nenhum método abstrato.
  + Pode ser instanciada.

Aula 04/07/2022

* Mecanismos para tratamento de exceções:
  + Permite tratatar uma exceção antes de abortar a execução do código. Para isto, ao gerar um erro , uma execução é lançada.
  + Palavras-chaves: **TRY** **THROW** **CATCH**
  + #include <stdexcept>
  + Using std::runtime\_error.
  + EX:
  + Catch(...) 🡪 Pegará qualquer coisa que chegar, sem depender do tipo(default).
  + Runtime\_error é uma classe provida pela linguagem