

Paradigma Imperativo

Integrantes:

- **Pedro Henrique Souza Cravo**
- **Pedro Lopes Monteiro**
- **Pedro Maia Dantas Nunes**
- **Rafael Barcelos de Azevedo**
- **Rafael Christian Silva Wernesbach**
- **Rafael Ferreira Bassul**
- **Rebeca Bravim Garcia**
- **Ricardo Ramalho Marques**
- **Rickson Medice Tomé**
- **Robson Júnior Schultz Dias**
- **Wellington Carvalho Branco Saldanha Junior**
- **William Alexsander Santos Siqueira**

Paradigmas

Definição:

- Algo que ocorreu ou foi descoberto no passado que pode ajudar a guiar o presente

Na computação:

- Categorização das várias linguagens de programação
- Conjunto de comportamentos esperados de uma linguagem

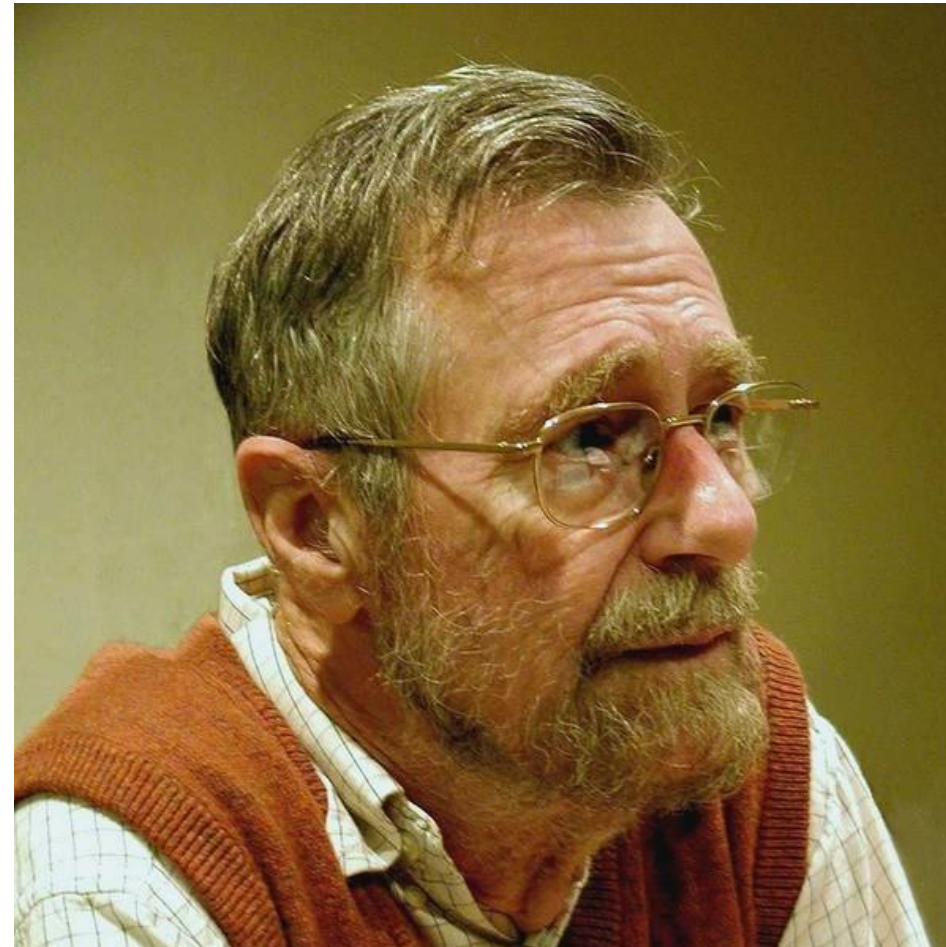


Paradigmas

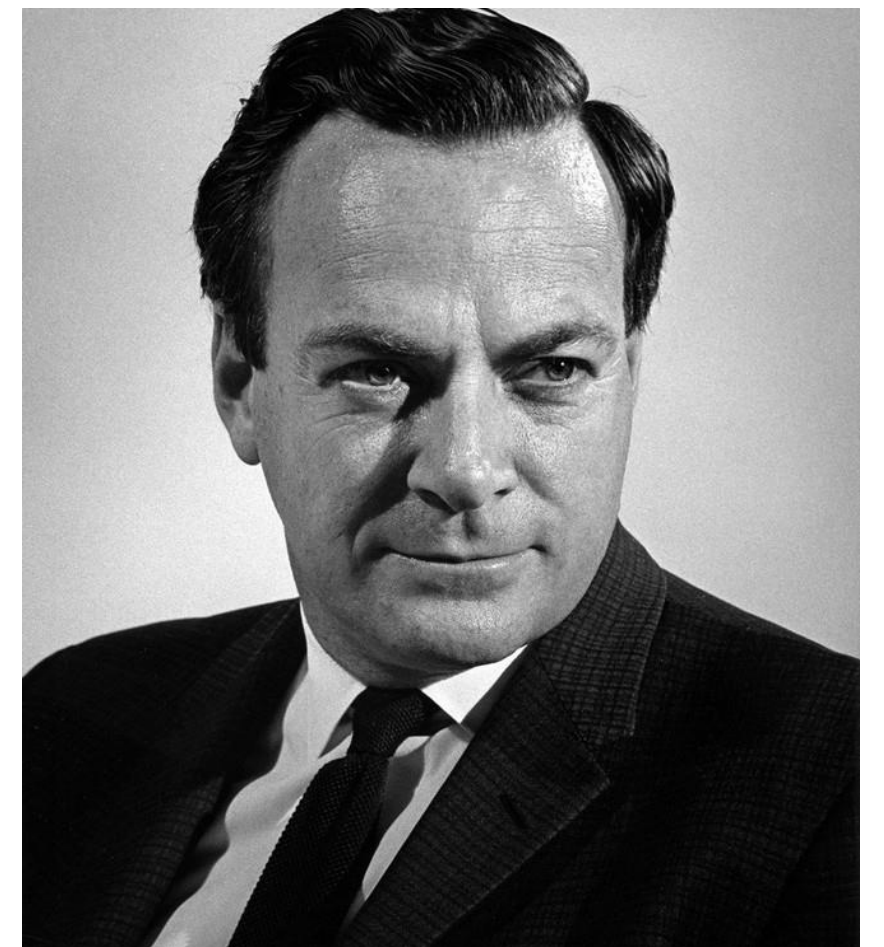
Edsger Dijkstra, em 1968 usou o termo pela primeira vez.

Apesar disso, no futuro criticou o conceito.

Outros grandes cientistas como Hoare e Feynmann, também criticaram o conceito, por tornar coisas simples em coisas complexas



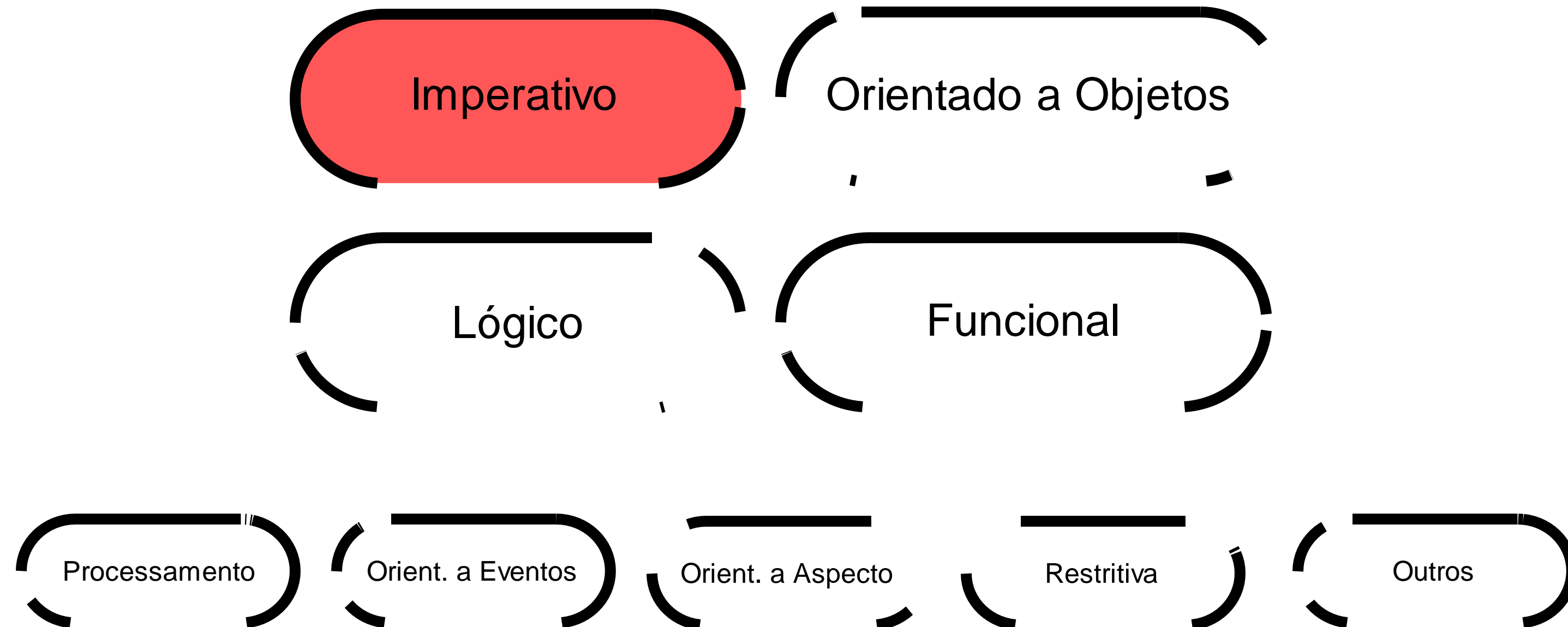
Hoare



Feynmann

Paradigmas

Existem vários:



Paradigmas

Multiparadigma:

- Hoje em dia é raro vermos uma linguagem que não seja multiparadigma
- São linguagens com suporte para o programador trabalhar com o conceito de vários paradigmas

Paradigmas puros:

- Conceito confuso
- Extremamente raro
- São linguagens que só usam um paradigma
- C, Pascal, SmallTalk

Histórico

Acontecimentos marcantes:

- Máquina de Turing: Dispositivo teórico que descreve um modelo abstrato de um computador.
- Arquitetura de Von Neumann;
- IBM 704 (1954): Sua capacidade permitiu a criação de linguagens de programação.

Primeiras linguagens:

- Fortran I (1957): Nomes de variáveis com até seis caracteres, sub-rotinas definidas pelos usuários, sentenças de seleção e repetição.
- Algol 60: As variáveis podem ter qualquer tamanho, sentenças de seleção aninhadas foram permitidas e passagem de parâmetros por nome ou valor.
- COBOL: Divisões de dados fortes.

Paradigma Imperativo

Definição:

- É um estilo de programação onde o programador escreve código que descreve, passo a passo, como o computador deve realizar as operações para atingir o resultado desejado.
- Esse paradigma foca em como as coisas devem ser feitas, ou seja, nas instruções que modificam o estado do programa através de variáveis, e estruturas de controle.



Programação Estruturada

Definição:

- A programação estruturada é uma subcategoria do paradigma imperativo. Ela adota os princípios imperativos, mas com uma ênfase adicional em boas práticas e estruturas de controle específicas que melhoram a clareza e a organização do código.

Baseia-se em três conceitos fundamentais:

- **Sequência:** As instruções do programa SÃO executadas sequencialmente, uma após a outra.
- **Decisão (ou seleção):** O programa pode tomar decisões e executar diferentes blocos de código com base em condições lógicas (como o uso de estruturas if, else, e switch).
- **Repetição (ou iteração):** Permite que certas partes do código sejam repetidas várias vezes, utilizando estruturas de loop como for, while e do-while.

Programação Procedural

- A programação procedural também é uma subcategoria do paradigma imperativo.
- É comum também encontrarmos em literaturas afirmando que é uma subcategoria da programação estrutural, então podemos afirmar que ela também adota as características da programação estrutural, dando ênfase na modularização através da criação de funções e procedimentos reutilizáveis que encapsulam instruções para a execução de tarefas.

Em resumo podemos dizer que ela é uma subcategoria da programação estrutural representada pela criação de sub rotinas(funções e procedimentos) para guiar o fluxo do programa.

Vantagens

- **Clareza Sequencial:** Cada instrução é executada em uma sequência linear, tornando o rastreamento do fluxo de execução mais intuitivo para muitos programadores.
- **Facilidade de Depuração:** Ferramentas de depuração são bem desenvolvidas para essas linguagens, permitindo inspecionar e modificar o estado do programa com precisão.
- **Otimização de Desempenho:** Com acesso direto ao hardware e à memória, os programadores podem otimizar o desempenho de forma mais eficaz, ajustando o código para maximizar a eficiência.
- **Ampla Utilização:** Muitas linguagens de programação populares (como C, C++, Java, e Python) são baseadas no paradigma imperativo. Isso resulta em um ecossistema maduro com muitas bibliotecas, frameworks e ferramentas de desenvolvimento disponíveis.



Desvantagens

- **Erros de Manipulação de Memória:** A manipulação explícita de memória, comum em muitos ambientes imperativos, pode levar a erros como vazamentos de memória, acesso a memória inválida e corrupção de dados.
- **Complexidade com o crescimento do projeto:** À medida que o tamanho do código aumenta, manter a coerência e a clareza pode se tornar desafiador. Gerenciar o estado global e as interações entre diferentes partes do código pode levar a bugs difíceis de rastrear.
- **Possui difícil legibilidade:** Os programas imperativos grandes e complexos inevitavelmente se tornam difíceis de ler e entender, um dos motivos para isso é o fluxo do programa que pode ser espalhado por várias partes do programa.



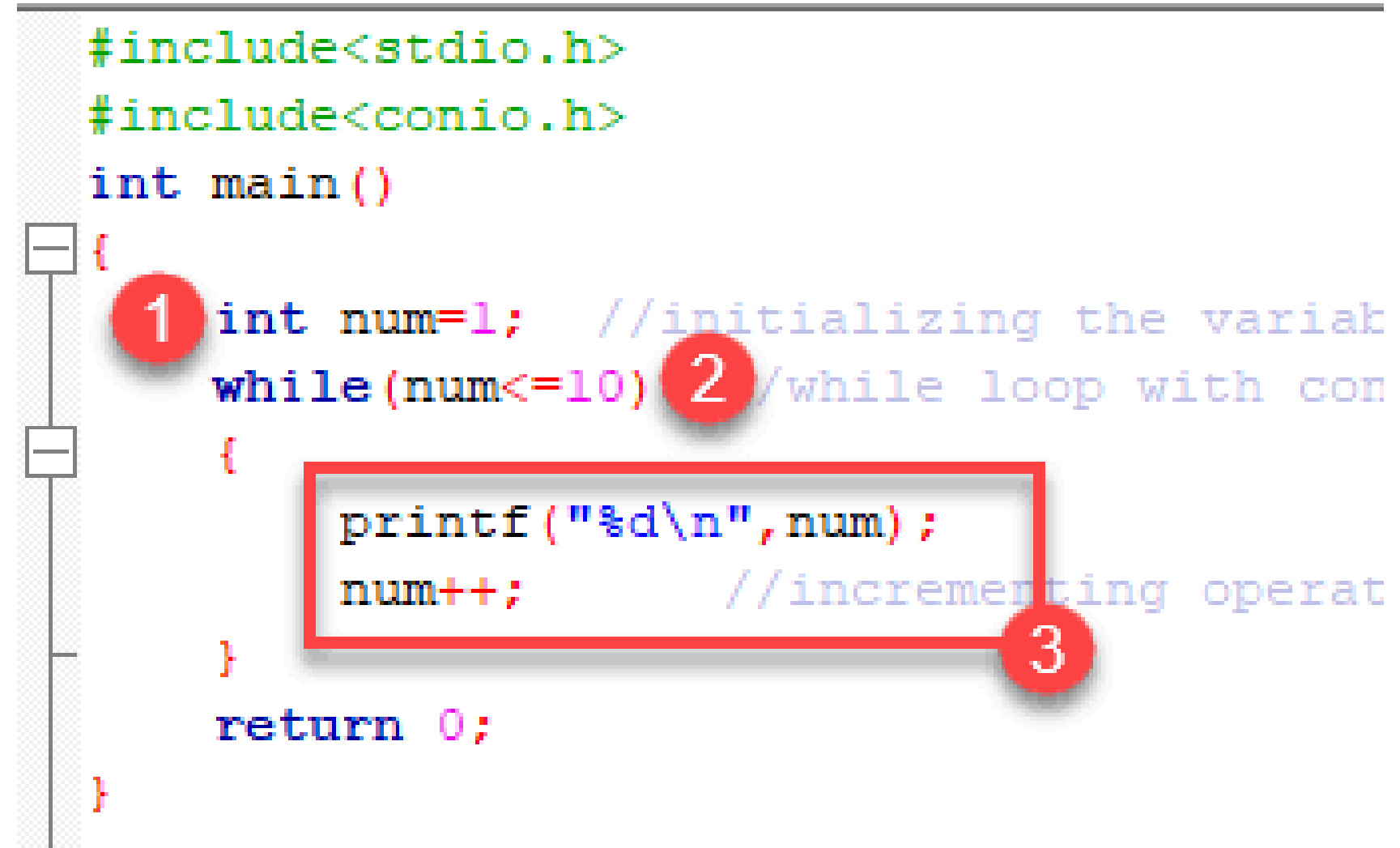
Linguagem C

- A linguagem C é um exemplo de linguagem imperativa.
- Criada na década de 1970, conhecida por sua eficiência e ampla aplicação em baixo e alto nível
- Possui uma sintaxe relativamente simples e concisa, o que torna o aprendizado mais fácil



Exemplo de Código em C

```
#include<stdio.h>
#include<conio.h>
int main()
{
    1 int num=1; //initializing the variak
    while(num<=10) 2 //while loop with con
    {
        printf ("%d\n",num) ;
        num++; //incrementing operat
    }
    return 0;
}
```



The image shows a C program snippet with three annotations. A vertical line on the left side of the code block has two small square boxes, one next to the opening curly brace of the main function and one next to the closing curly brace. A red circle with the number '1' is placed next to the line 'int num=1;'. A red circle with the number '2' is placed next to the line 'while(num<=10)'. A red circle with the number '3' is placed next to the closing curly brace of the while loop. A red rectangular box highlights the two lines of code inside the while loop: 'printf ("%d\n",num) ;' and 'num++; //incrementing operat'.

Comparativo

Breve resumo do Imperativo:

- **Estado Mutável:** O estado do programa é modificado através de atribuições de variáveis.
- **Sequenciamento:** A execução do programa segue uma sequência específica de instruções.
- **Controle Explícito do Fluxo:** Utiliza estruturas de controle como loops e condicionais para gerenciar o fluxo do programa.

Comparativo

Breve resumo do paradigma Funcional:

- **Imutabilidade:** Dados são imutáveis e não mudam após serem criados.
- **Funções Puras:** Funções retornam resultados apenas baseados nos seus argumentos, sem efeitos colaterais.
- **Composição de Funções:** Funções podem ser compostas para formar funções mais complexas.

Comparativo

Imperativo x Funcional

- **Estado:** Imperativo usa estado mutável; funcional usa imutabilidade.
- **Fluxo de Controle:** Imperativo controla o fluxo explicitamente; funcional depende da aplicação de funções e recursão.

Comparativo

Breve resumo do paradigma Lógico

- **Declaração de Conhecimento:** Programas são uma coleção de fatos e regras.
- **Resolução Automática:** O sistema deduz automaticamente as soluções através de inferências lógicas.
- **Declarativo:** Foca no "o que" deve ser resolvido, em vez de "como".

Comparativo

Imperativo x Logico

- **Abordagem:** Imperativo é descritivo (descreve como fazer); lógico é declarativo (descreve o que quer).
- **Fluxo de Controle:** Imperativo é sequencial e controlado explicitamente; lógico é baseado em inferência e busca

Comparativo

Breve resumo do paradigma Orientado a Objetos

- **Encapsulamento:** Dados e métodos são encapsulados dentro de objetos.
- **Herança:** Objetos podem herdar características de outros objetos.
- **Polimorfismo:** Objetos podem ser tratados como instâncias de suas superclasses.

Comparativo

Imperativo x Orientado a Objetos

- **Estrutura de Dados:** Imperativo usa variáveis e procedimentos; orientado a objetos usa objetos e classes.
- **Modularidade:** Orientado a objetos promove modularidade e reutilização através de objetos e herança.
- **Controle de Estado:** Ambos podem ter estados mutáveis, mas orientado a objetos atribui o estado em objetos.

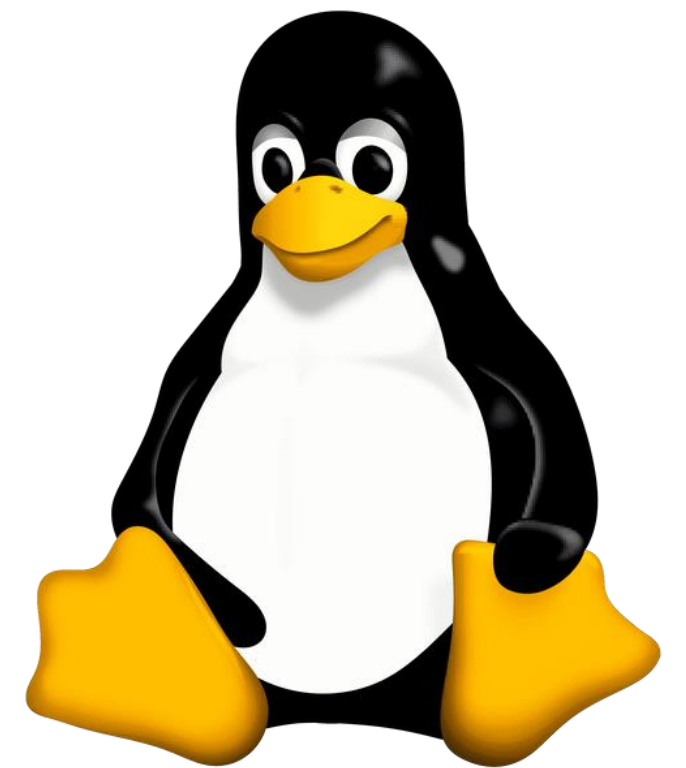
Linguagens Imperativas

- **Segundo o gemini:** C, C++, Java, Python, PHP, JavaScript, Pascal, BASIC, Cobol, Fortran, Assembly. E podem ser consideradas também: Go, Rust e D
- **Segundo a wikipedia br:** Ada, ALGOL, Basic, C, PHP, Java, Cobol, Fortran, Pascal, Python, Lua, Mathematica
- **Segundo a wikipedia ingles:** FORTRAN, COBOL, Algol, Basic, C, C++

Tutorial de instalação

Linux (Ubuntu)

- Antes de iniciar a instalação, primeiro verifique se o GCC já está instalado na máquina
- Para isso podemos abrir um terminal e digitar o comando:
`$gcc --version`



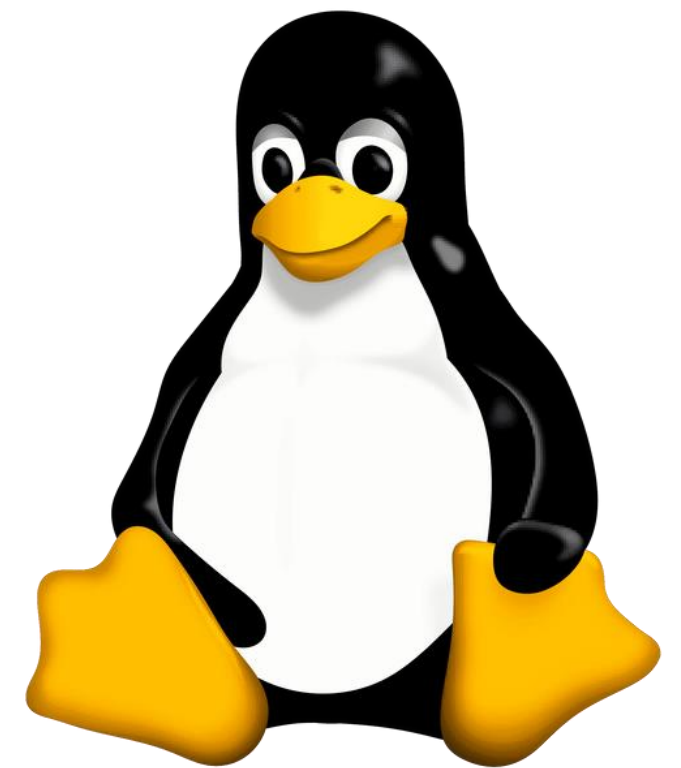
Tutorial de instalação

Linux (Ubuntu)

- Caso o GCC já esteja instalado em sua máquina, serão exibidas informações referentes a versão do mesmo, como mostra a imagem a seguir.

```
phoenixnap@phoenixnap:~$ gcc --version
gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0
Copyright (C) 2021 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

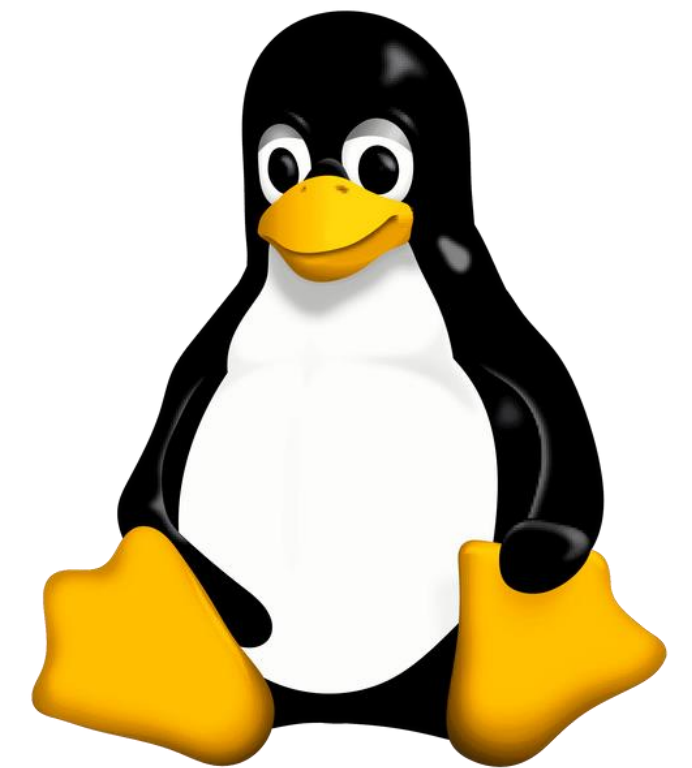
- Caso não esteja, o terminal deve exibir uma mensagem de erro.



Tutorial de instalação

Linux (Ubuntu)

- Para instalar/atualizar o GCC você deve seguir o passo a passo referente a sua distribuição Linux. Aqui vamos exemplificar com o Ubuntu.
- No Ubuntu, você pode usar o apt(advanced package tool) para instalar o GCC.
- Abrindo um terminal você deve executar os seguintes comandos:



Tutorial de instalação

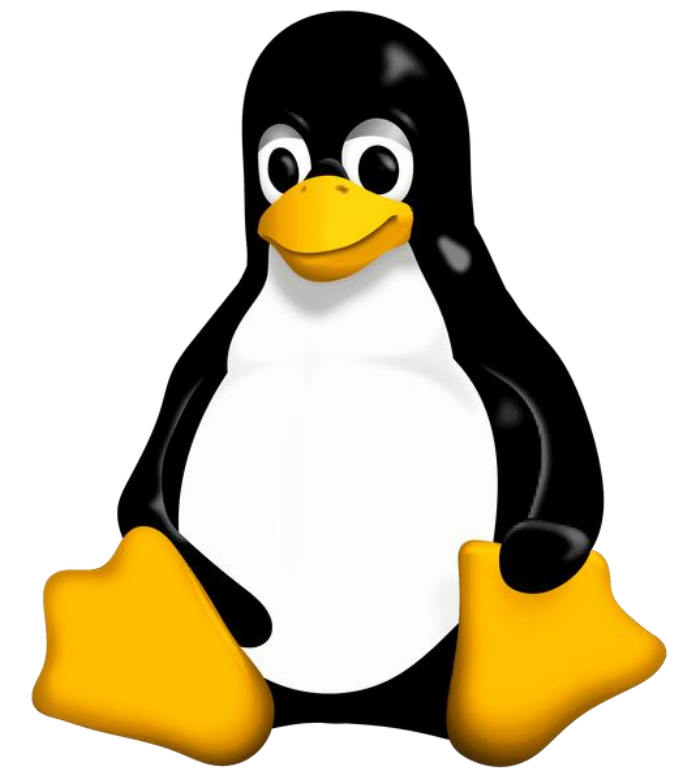
Linux (Ubuntu)

\$ sudo apt update

- Este comando é usado para atualizar a lista de pacotes disponíveis nos repositórios configurados no sistema.

\$ sudo apt install build-essential

- Este comando é usado para instalar um pacote que contém ferramentas de desenvolvimento essenciais, como o GCC e outras ferramentas necessárias para compilar programas em C



Tutorial de instalação

Windows

- Caso você utilize o Windows, uma maneira bem difundida para obter um compilador de C é através do MSYS2.
- Para instalar o GCC no Windows utilizando o MSYS2 siga os seguintes passos:

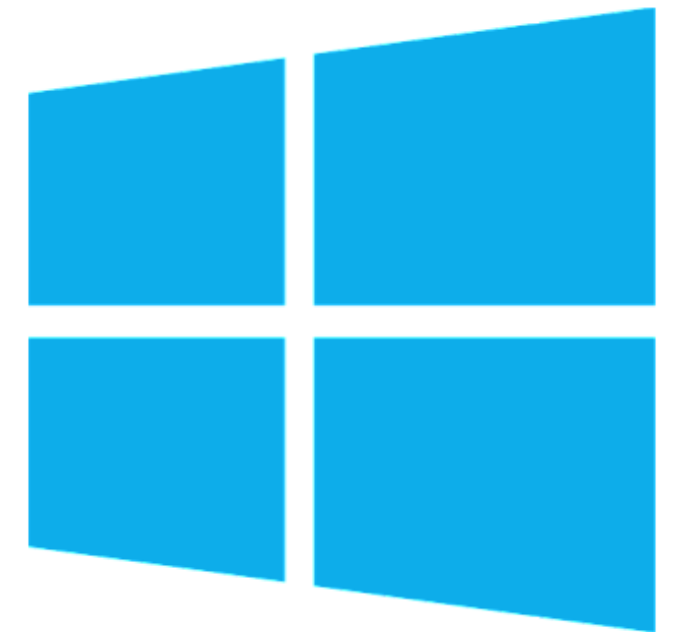
1- Baixar MSYS2

1.1- Vá para a página de downloads do MSYS2

(<https://www.msys2.org/>)

1.2- Baixe o instalador adequado para seu sistema

(provavelmente o arquivo.exe)



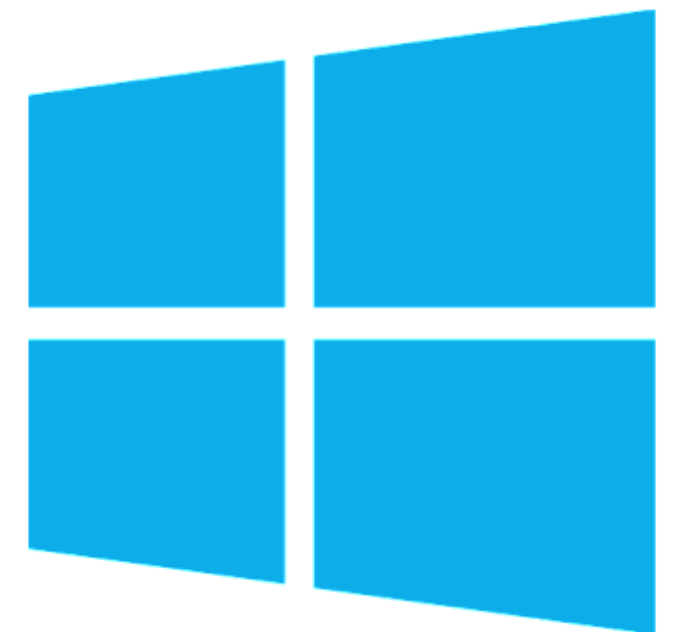
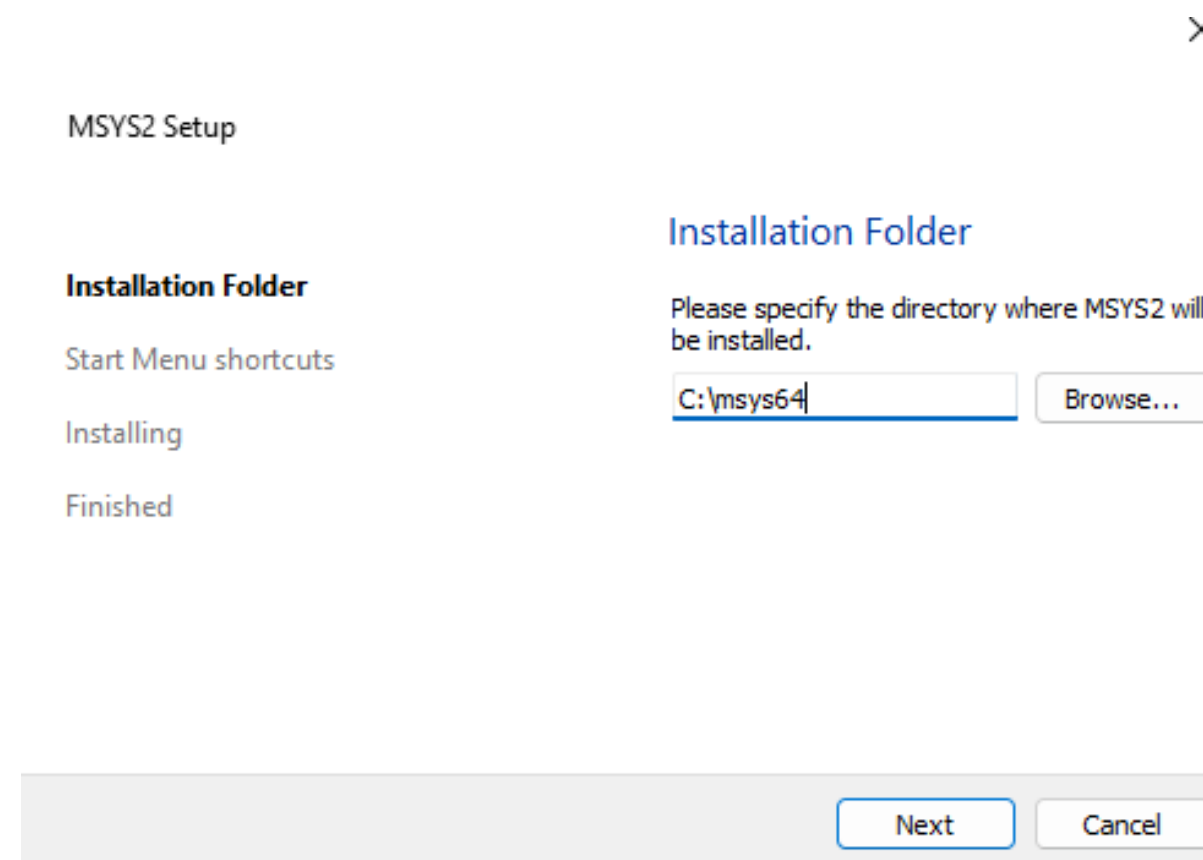
Tutorial de instalação

Windows

2- Instalar MSYS2

2.1- Execute o instalador e siga as instruções na tela para instalar o MSYS2

2.2- Durante a instalação, escolha um diretório de instalação (por exemplo, 'C:\msys64')



Tutorial de instalação

Windows

3- Atualizar o MSYS2

3.1- Após a instalação, abra o terminal MSYS2 (você pode encontrar um atalho no menu iniciar)

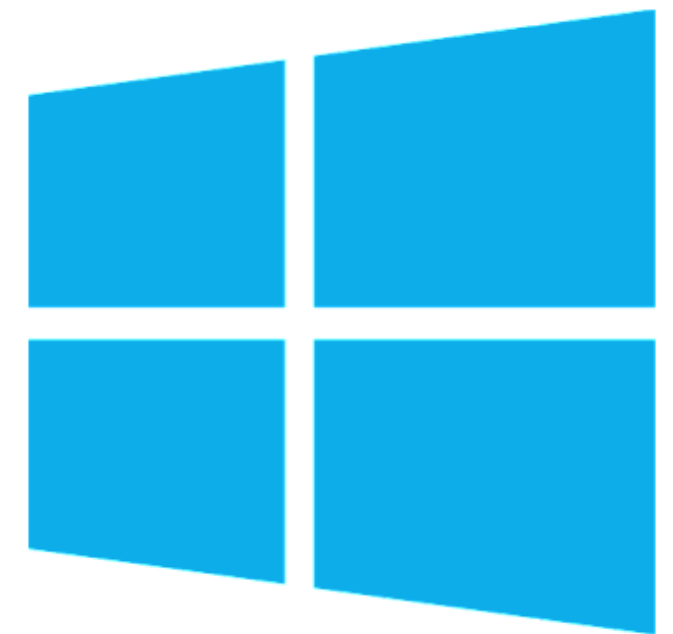
3.2 - Atualize os pacotes e o banco de dados de pacotes. Execute os seguintes comandos no terminal MSYS2:

pacman -Syu

4- Instalar os Pacotes do GCC

4.1- No terminal MSYS2, instale os pacotes necessários para o GCC. Execute o seguinte comando:

pacman -S --needed base-devel mingw-w64-x86_64-toolchain

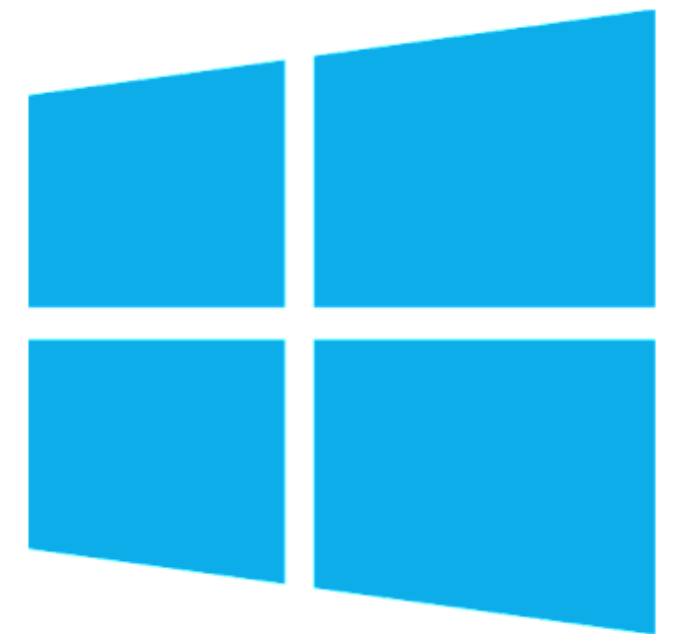


Este comando instala o conjunto básico de ferramentas de desenvolvimento e o toolchain do GCC para Windows 64 bits

Tutorial de instalação

Windows

- Ao terminar a instalação é recomendável que você verifique se o GCC foi instalado corretamente.
- Para isso execute o comando no terminal:
gcc --version
- Então deverá ser exibido a versão do GCC instalada.



Questões Teóricas

- I. É um paradigma de programação que organiza o código em unidades autônomas chamadas objetos, encapsulando dados e comportamentos relacionados, promovendo reutilização, modularidade e facilitando a compreensão e manutenção do sistema.
- II. É um paradigma de programação que enfatiza o uso de estruturas de controle, como sequência, seleção e repetição, para criar algoritmos organizados e eficientes, promovendo uma abordagem procedural e modular que facilita a compreensão, manutenção e depuração do código-fonte.

Assinale a alternativa que apresenta quais são os paradigmas de programação citados.

- A) I. Programação Orientada a Depuração - II. Programação Orientada a Testes
- B) I. Programação Orientada a Testes – II. Programação Orientada a Depuração
- C) I. Programação Orientada a Objetos – II. Programação Estruturada
- D) I. Programação Estruturada – II. Programação Orientada a Objetos

Questões Teóricas

- I. É um paradigma de programação que organiza o código em unidades autônomas chamadas objetos, encapsulando dados e comportamentos relacionados, promovendo reutilização, modularidade e facilitando a compreensão e manutenção do sistema.
- II. É um paradigma de programação que enfatiza o uso de estruturas de controle, como sequência, seleção e repetição, para criar algoritmos organizados e eficientes, promovendo uma abordagem procedural e modular que facilita a compreensão, manutenção e depuração do código-fonte.

Assinale a alternativa que apresenta quais são os paradigmas de programação citados.

- A) I. Programação Orientada a Depuração - II. Programação Orientada a Testes
- B) I. Programação Orientada a Testes – II. Programação Orientada a Depuração
- C) I. Programação Orientada a Objetos – II. Programação Estruturada**
- D) I. Programação Estruturada – II. Programação Orientada a Objetos

Questões Teóricas

Avalie as seguintes afirmativas associadas à programação estruturada:

- I. Uma variável declarada no contexto de uma função é automaticamente acessível às demais funções do programa.
- II. A passagem de variável por valor a uma função permite que a função altere o valor da variável.
- III. Uma estrutura de seleção ou repetição, se fizer parte de outra estrutura de seleção ou repetição, deve estar completamente contida nesta.

Assinale a alternativa que contém a(s) afirmativa(s) CORRETA(S).

- A) I, apenas.
- B) II e III, apenas.
- C) III, apenas.
- D) I e II, apenas.
- E) I, II e III.

Questões Teóricas

Avalie as seguintes afirmativas associadas à programação estruturada:

- I. Uma variável declarada no contexto de uma função é automaticamente acessível às demais funções do programa.
- II. A passagem de variável por valor a uma função permite que a função altere o valor da variável.
- III. Uma estrutura de seleção ou repetição, se fizer parte de outra estrutura de seleção ou repetição, deve estar completamente contida nesta.

Assinale a alternativa que contém a(s) afirmativa(s) CORRETA(S).

A) I, apenas.

B) II e III, apenas.

C) III, apenas.

D) I e II, apenas.

E) I, II e III.

Questões Teóricas

Em uma linguagem de programação estruturada, como a linguagem C, é comum dividir o código em conjuntos de instruções que realizam determinada tarefa e que podem ser reaproveitados em mais de um momento ao longo do código. Estes conjuntos podem ser caracterizados como procedimentos ou funções. São definições de procedimentos e funções, EXCETO:

- A) Funções retornam valor.
- B) Procedimentos não retornam valor.
- C) Funções e procedimentos são sinônimos.
- D) Funções podem ser utilizadas em expressões aritméticas dentro de um código.

Questões Teóricas

Em uma linguagem de programação estruturada, como a linguagem C, é comum dividir o código em conjuntos de instruções que realizam determinada tarefa e que podem ser reaproveitados em mais de um momento ao longo do código. Estes conjuntos podem ser caracterizados como procedimentos ou funções. São definições de procedimentos e funções, EXCETO:

- A) Funções retornam valor.
- B) Procedimentos não retornam valor.
- C) Funções e procedimentos são sinônimos.**
- D) Funções podem ser utilizadas em expressões aritméticas dentro de um código.

Questões Teóricas

Com relação ao paradigma de programação estruturada, analise as afirmativas a seguir.

I. Divide um problema complexo em pequenas partes mais simples que, trabalhadas conjuntamente, permitem solucioná-lo.

II. Enfatiza procedimentos implementados em blocos estruturados, com comunicação por passagem de dados.

III. Pelo paradigma estruturado, também conhecido como iterativo, qualquer problema pode ser resolvido utilizando três estruturas: sequencial, condição e repetição.

Assinale:

A) se somente a afirmativa I estiver correta.

B) se somente as afirmativas I e II estiverem corretas.

C) se somente as afirmativas I e III estiverem corretas.

D) se somente as afirmativas II e III estiverem corretas.

Questões Teóricas

Com relação ao paradigma de programação estruturada, analise as afirmativas a seguir.

I. Divide um problema complexo em pequenas partes mais simples que, trabalhadas conjuntamente, permitem solucioná-lo.

II. Enfatiza procedimentos implementados em blocos estruturados, com comunicação por passagem de dados.

III. Pelo paradigma estruturado, também conhecido como iterativo, qualquer problema pode ser resolvido utilizando três estruturas: sequencial, condição e repetição.

Assinale:

A) se somente a afirmativa I estiver correta.

B) se somente as afirmativas I e II estiverem corretas.

C) se somente as afirmativas I e III estiverem corretas.

D) se somente as afirmativas II e III estiverem corretas.

Questões Teóricas

Na programação estruturada, as funções podem receber parâmetros por valor ou por referência. Sobre passagem de parâmetro por referência, assinale a afirmativa correta.

- A) Somente é possível passar por referência parâmetros do tipo char.
- B) Somente é possível passar por referência parâmetros do tipo inteiro.
- C) Um parâmetro passado por referência é, na verdade, um endereço de memória.
- D) Um parâmetro passado por referência é, na verdade, um valor armazenado na memória.

Questões Teóricas

Na programação estruturada, as funções podem receber parâmetros por valor ou por referência. Sobre passagem de parâmetro por referência, assinale a afirmativa correta.

- A) Somente é possível passar por referência parâmetros do tipo char.
- B) Somente é possível passar por referência parâmetros do tipo inteiro.
- C) Um parâmetro passado por referência é, na verdade, um endereço de memória.**
- D) Um parâmetro passado por referência é, na verdade, um valor armazenado na memória.

Questões Práticas

Quais os valores armazenados no vetor lista ao fim do programa?

```
#include <stdio.h>

int main() {
    int lista[5];

    for (int i = 0; i < 5; i++) {
        lista[i] = i * 2;
        if (i % 3 == 0) {
            lista[i] = 0;
        }
    }

    return 0;
}
```

Questões Práticas

```
#include <stdio.h>

int main() {
    int lista[5];

    for (int i = 0; i < 5; i++) {
        lista[i] = i * 2;
        if (i % 3 == 0) {
            lista[i] = 0;
        }
    }

    return 0;
}
```

Quais os valores armazenados no vetor lista ao fim do programa?

R: 0,2,4,0,8

Questões Práticas

O programa possui um erro em sua lógica, onde ele está?

```
#include <stdio.h>

int primo(int a);
int primo(int a) {
    int divisores = 0;
    for (int i = 1; i <= a / 2; i++) {
        if (a % i == 0) {
            divisores++;
            printf("%i ", i);
        }
    }
    if (divisores == 2) {
        return 0;
    } else {
        return 1;
    }
}

int main(void) {
    int a = primo(4);
    printf("%i", a);
    return 0;
}
```


Questões Práticas

O programa possui um erro em sua lógica, onde ele está?

R: na verificação do valor de divisores no segundo if da função primo, o correto seria ≥ 2

```
#include <stdio.h>

int primo(int a);
int primo(int a) {
    int divisores = 0;
    for (int i = 1; i <= a / 2; i++) {
        if (a % i == 0) {
            divisores++;
            printf("%i ", i);
        }
    }
    if (divisores == 2) {
        return 0;
    } else {
        return 1;
    }
}

int main(void) {
    int a = primo(4);
    printf("%i", a);
    return 0;
}
```

Questões Práticas

O que será impresso ao fim do programa?

```
#include <stdio.h>

int modulo(int a);
int modulo(int a) {
    if (a < 0) {
        return -a;
    } else {
        return a;
    }
}

int main(void) {
    int a = -10;
    printf("%i", modulo(a * 2));
}
```

Questões Práticas

O que será impresso ao fim do programa?

R: 20

```
#include <stdio.h>

int modulo(int a);
int modulo(int a) {
    if (a < 0) {
        return -a;
    } else {
        return a;
    }
}

int main(void) {
    int a = -10;
    printf("%i", modulo(a * 2));
}
```

Questões Práticas

O que será impresso ao fim do programa?

```
#include <stdio.h>

int main() {
    int a = 1, b = 0, c = 1;

    if (a && b || c) {
        printf("Verdadeiro\n");
    } else {
        printf("Falso\n");
    }

    return 0;
}
```

Questões Práticas

O que será impresso ao fim do programa?

R: Verdadeiro

```
#include <stdio.h>

int main() {
    int a = 1, b = 0, c = 1;

    if (a && b || c) {
        printf("Verdadeiro\n");
    } else {
        printf("Falso\n");
    }

    return 0;
}
```

Questões Práticas

```
#include <stdio.h>

int main() {
    int x = 7;
    int y = 3;
    float z == x * y + (float)y / x;

    printf("z = %f\n", z);
    return 0;
}
```

O programa possui um erro, onde ele está?

Questões Práticas

```
#include <stdio.h>

int main() {
    int x = 7;
    int y = 3;
    float z == x * y + (float)y / x;

    printf("z = %f\n", z);
    return 0;
}
```

O programa possui um erro, onde ele está?

R: a variavel Z está sendo comparada com a expressão, e não recebendo um valor calculado pela expressão