

Paradigma Imperativo

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

Orientador: Prof. Abrantes Araujo Silva Filho

1 Paradigmas

1.1 O que é o paradigma

Um paradigma é algo que já aconteceu que pode servir como estudo ou guia para lidar com padrões futuros (exemplo: Vila Velha nunca teve problemas sérios com terremotos, portanto não há um paradigma para lidar com terremotos, ao contrário de alagamentos, que existem paradigmas). Em programação, consideramos que um paradigma é um conjunto de comportamentos esperados para lidar com vários tipos de situação, por exemplo as formas diferentes de atribuições de tipo tais qual tipo de variável ou classe, apesar disso, há grandes nomes que não aprovam o conceito de paradigmas. Existem inúmeros paradigmas, mas alguns receberam mais atenção, sendo os 4 mais importantes: imperativo, orientado a objetos, funcional e lógico. Além deles, podemos mencionar o processamento paralelo (imperativo), orientado a eventos, orientada a aspecto, restritiva, simbólica, reativa, e entre outros.

Apesar de parecerem coisas bem distintas, são raras as linguagens que usam apenas um paradigma, sendo que a maioria é multiparadigma, isto é, permite que o programador utilize recursos para trabalhar com conceitos de vários paradigmas, principalmente entre procedural e POO. Exemplos de linguagens com paradigmas puros: C, Pascal, BASIC e COBOL, todos procedurais e SmallTalk orientada a objetos, apesar de terem bibliotecas que cancelam esse atributo.

Os atributos principais do paradigma imperativo:

- Arquitetura Von Neumann (dados e programas são armazenados na mesma memória e as variáveis são os recursos principais).
- Estruturas de sequência
- Estruturas de decisão ou seleção
- Estruturas de iteração (laços)
- Compartilhamento através de bibliotecas
- Gerenciamento de escopo de variáveis

A programação estruturada é uma forma de escrever códigos que pede que o programador escreva o código na mesma sequência que deve ser executado, sem o uso do comando `goto` (especialmente relevante na época que surgiu). Hoje em dia evoluiu para a modularização.

1.2 O que é um paradigma imperativo?

O paradigma imperativo 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, loops, e condicionais.

1.3 Programação estruturada

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. Esse paradigma se baseia 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`.

A programação estruturada promove o uso de sub-rotinas, funções ou procedimentos, para dividir um programa em partes menores e mais gerenciáveis. Isso facilita a compreensão, manutenção e depuração do código. Além disso, evita o uso excessivo do comando `goto`, que pode tornar o fluxo do programa difícil de seguir e manter.

1.4 Diferença entre Programação Imperativa e Programação Estruturada

Paradigma Imperativo: Inclui todas as formas de programação que expressam o fluxo de controle com comandos que mudam o estado do programa. Pode usar instruções como `goto`, que podem tornar o código difícil de seguir.

Programação Estruturada: Restringe o uso de instruções de salto como `goto` e promove o uso de estruturas de controle mais organizadas e claras, como loops e condicionais, além de funções e procedimentos para modularizar o código.

1.5 Uso de goto

Programação Imperativa: Pode usar `goto` para saltar entre diferentes partes do código, o que pode tornar o fluxo do programa difícil de seguir.

Programação Estruturada: Evita o uso de `goto`, preferindo estruturas de controle mais claras e organizadas.

1.6 Modularidade

Programação Imperativa: Pode ou não promover a modularidade. Em sua forma mais básica, pode consistir em longas sequências de instruções sem uma estrutura clara.

Programação Estruturada: Enfatiza a modularidade, dividindo o código em funções, procedimentos ou módulos menores e mais manejáveis.

1.7 Legibilidade e Manutenção

Programação Imperativa: Pode resultar em código mais difícil de ler e manter, especialmente se abusar de `goto` e não modularizar adequadamente.

Programação Estruturada: Produz código mais legível e fácil de manter devido ao uso de boas práticas e estruturas de controle bem definidas.

1.8 Exemplos em C

Código Imperativo (`goto`):

```
#include <stdio.h>

int main() {
    int a = 10;
    int b = 20;
    int result;

    if (a > b)
        goto grater;
    else
        goto lesser;

grater:
    result = a + b;
    printf("Sum is %d\n", result);
    return 0;
lesser:
    result = b - a;
    printf("Difference is %d\n", result);
    return 0;
}
```

Código estruturado (sem `goto`):

```
#include <stdio.h>

int main() {
    int a = 10;
    int b = 20;
    int result;

    if (a > b) {
        result = a + b;
        printf("Sum is %d\n", result);
    } else
        result = a - b;
        printf("Difference is %d\n", result);
    }
    return 0;
}
```

1.9 Programação Procedural

Traz ênfase na organização do código utilizando procedimentos ou funções, em algumas literaturas ela é considerada uma subcategoria da programação estruturada encapsulando um conjunto de instruções para realizar tarefas específicas.

1.10 Uso de `goto`

Programação Imperativa: Pode usar `goto` para saltar entre diferentes partes do código, o que pode tornar o fluxo do programa difícil de seguir.

Programação Procedural: A princípio, como a programação procedural está quase sempre junta com a estruturada, podendo ser considerada uma subcategoria da mesma, ela evita o uso de `goto`, utilizando procedimentos e funções para guiar o fluxo do programa. Entretanto, é possível encontrar `goto` dentro de procedimentos e funções de códigos mal estruturados.

1.11 Modularidade

Programação Imperativa: Pode ou não promover a modularidade. Em sua forma mais básica, pode consistir em longas sequências de instruções sem uma estrutura clara.

Programação Procedural: Enfatiza a modularidade, dividindo o código em funções e procedimentos.

1.12 Legibilidade e Manutenção

Programação Imperativa: Pode resultar em código mais difícil de ler e manter, especialmente se abusar de `goto` e não modularizar adequadamente.

Programação Procedural: Produz código mais legível e fácil de manter devido à organização em funções e procedimentos, e ao uso de boas práticas presentes também na programação estruturada.

Código Procedural (Estruturado):

```
// Função para calcular a soma de dois números
int soma(int a, int b) {
    return a + b;
}

int main {
    int num1, num2, resultado;

    // Solicita ao usuário que insira dois números
    printf("Digite o primeiro número: ");
    scanf("%d", &num1);
    printf("Digite o segundo número: ");
    scanf("%d", &num2);

    // Calcula a soma dos dois números
    resultado = soma(num1, num2);

    // Exibe o resultado
    printf("A soma é: %d\n", resultado);
```

```
return 0;  
}
```

2 Histórico

Os conceitos principais deste paradigma estão atrelados ao dispositivo teórico da máquina de Turing, que foi pensada na década de 30 e a arquitetura de computadores criada por Von Neumann.

As primeiras linguagens de programação imperativas eram linguagens de baixo nível. A introdução do IBM 704 em 1954 levou ao desenvolvimento do FORTRAN, primeira linguagem imperativa de alto nível. O ambiente no qual FORTRAN foi desenvolvido era o seguinte: os computadores tinham memórias pequenas, eram lentos, uso exclusivo para computações científicas e não era muito eficiente programá-los. A linguagem FORTRAN I incluía formatação de entrada e saída, nomes de variáveis com até seis caracteres (em FORTRAN 0 eram apenas 2), sub-rotinas definidas pelos usuários, sentença de seleção `if` e sentença de repetição `do`.

O paradigma de programação imperativo é um dos estilos de programação mais antigos e fundamentais. Ele se baseia no conceito de comandos explícitos que alteram o estado de um programa.

2.1 Origem e primeiros anos

Década de 1930 – Máquina de Turing: A concepção da máquina de Turing em 1936 posteriormente se tornaria a base teórica para a programação imperativa. Sua ideia de uma máquina universal que executa instruções sequenciais é essencial para esse paradigma.

Década de 1940 – Início da computação: A arquitetura proposta por John Von Neumann, que se baseia em uma unidade central de processamento e memória compartilhada, é fundamental para a execução de programas imperativos. Ela aprimorou a máquina de Turing e permitiu que programas fossem armazenados na memória do computador e executados sequencialmente.

Os primeiros computadores digitais eletrônicos, como o ENIAC, foram programados usando código de máquina e linguagens de montagem. Esses programas eram essencialmente imperativos, consistindo de uma série de instruções diretas que o computador executava sequencialmente.

Década de 1950 – Primeiras linguagens de programação: -

Assembly: As linguagens de montagem surgiram para simplificar a programação direta em código de máquina, mantendo um estilo imperativo.

FORTRAN: Desenvolvida pela IBM, é considerada a primeira linguagem de programação de alto nível. Ela introduziu a estrutura básica de controle imperativa com comandos como loops e condicionais.

ALGOL: Introduzida como uma linguagem de propósito geral, influenciou muitas linguagens subsequentes e estabeleceu a base para a estrutura de programação imperativa, incluindo blocos de código alinhados.

2.2 Desenvolvimento e Evolução

Décadas de 1960 e 1970 – Consolidação e expansão: A década de 60 viu a ascensão da programação estruturada, que promoveu a decomposição de programas em sub-rotinas, estruturas de controle claras e eliminação de desvios incontroláveis (como o uso excessivo de `goto`).

A década de 70 viu o desenvolvimento de sistemas operacionais complexos que exigiam uma programação imperativa eficiente e estruturada, como o Unix.

Criada por Dennis Ritchie, a linguagem C teve um impacto profundo no desenvolvimento de software, combinando eficiência com uma sintaxe expressiva que suportava programação de sistemas e aplicação. C estabeleceu muitos padrões que ainda são usados em linguagens imperativas modernas.

Década de 1980 – Popularização e novos paradigmas: Desenvolvida por Bjarne Stroustrup, C++ adicionou conceitos de programação orientada a objetos ao paradigma imperativo, permitindo uma maior modularidade e reutilização de código.

Décadas de 1990 e 2000 – Híbridização: Desenvolvida pela Sun Microsystems, Java popularizou a combinação dos paradigmas imperativo e orientado a objeto, com um foco em portabilidade e segurança.

2.3 Influência e legado

O paradigma imperativo continua a ser um componente central da maioria das linguagens de programação modernas. Sua abordagem direta e intuitiva para a manipulação de estados e controle de fluxo é fundamental para muitos tipos de desenvolvimento de software, desde sistemas operacionais até aplicações de alto nível. A evolução contínua e a integração com outros paradigmas mostram a sua adaptabilidade e a sua importância duradoura na ciência da computação.

2.4 Linguagens de programação e suas respectivas datas

Fortran (1956): Em 1956 foi implementado por John Backus a primeira linguagem de alto nível que foi projetado para aplicações científicas seu nome foi Fortran (FORmula TRANslation). Fortran ganhou ampla aceitação e continua a ser largamente utilizado na comunidade.

Cobol (1960): Por iniciativa do Departamento de Defesa dos EUA em 1959 foi criada a linguagem COBOL para ser facilmente compreensível e utilizável por pessoas que não são programadores profissionais.

Algol (1960): Houve um esforço conjunto de vários matemáticos e cientistas da computação para criar uma linguagem de programação que fosse mais próxima das notações matemáticas utilizadas para expressar algoritmos. Isso resultou no ALGOL 58.

Basic (1964): A ideia por trás do BASIC era fornecer uma linguagem com uma sintaxe simples e uma estrutura clara, permitindo que os usuários escrevessem programas sem precisar se preocupar com os detalhes complexos da programação.

C (1970): Ela foi criada como uma evolução da linguagem de programação B. C é conhecida por sua eficiência e portabilidade, sendo amplamente utilizada em sistemas operacionais, desenvolvimento de software de sistema e aplicativos.

3 Vantagens e Desvantagens

3.1 Vantagens

Eficiência (embute o modelo Von Neumann): Considera a arquitetura presente nos computadores atuais, aproveitando ao máximo o hardware disponível.

Paradigma dominante e bem estabelecido: Amplamente utilizado ao se aprender programação nos dias atuais sendo bem estabelecido no meio da computação, o que resulta em uma vasta quantidade de recursos, bibliotecas, frameworks e comunidades disponíveis para suporte, tendo também ferramentas e práticas já consolidadas para desenvolvimento, depuração e otimização do código.

Modelagem natural de aplicações do mundo real: É considerada uma maneira mais natural de se programar, pois consiste em um conjunto de ações sequenciais executadas para alcançar um objetivo, assim como qualquer ação do dia a dia.

Clareza Sequencial: Cada instrução é executada em uma sequência linear, tornando o rastreamento do fluxo de execução mais intuitivo para muitos programadores.

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.

3.2 Desvantagens

Possui difícil legibilidade em grandes códigos: 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.

Relacionamento indireto com a E/S (indução a erros/estados): Nos programas imperativos, as operações de entrada e saída são por muitas vezes tratadas de forma indireta por funções ou procedimentos específicos, sendo intercalados com a lógica do programa, torna-se fácil introduzir erros, por conta disso também é fácil acabar gerando erros por conta da sequência das operações.

Ainda se foca em como a tarefa deve ser feita e não em o que deve ser feito
É considerado uma desvantagem por conta do detalhamento excessivo que é necessário, tendo menos abstrações, o que leva o código a ficar de difícil legibilidade.

Difícil paralelização: O controle explícito de estado e a sequência rígida de instruções tornam a paralelização de programas imperativos mais com-

plexa. Em comparação, paradigmas como o funcional são frequentemente mais adequados para execução paralela e concorrente.

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.

4 Comparativo

O paradigma imperativo é caracterizado pela programação baseada em instruções que alteram o estado do programa de forma sequencial. Programas imperativos descrevem como uma tarefa deve ser realizada usando comandos como loops, condicionais e atribuições de variáveis.

Características principais:

- 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.

4.1 Paradigma Funcional

O paradigma funcional enfatiza a avaliação de funções e evita mudanças de estado e dados mutáveis. Programas funcionais são construídos por meio de funções puras, que não têm efeitos colaterais.

Características principais:

- 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.

Comparação:

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

4.2 Paradigma Lógico

O paradigma lógico é baseado na lógica formal e na resolução de problemas através de declarações de fatos e regras. Um exemplo clássico é a linguagem Prolog.

Características principais:

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

Comparação:

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

4.3 Paradigma Orientado a Objetos

O paradigma orientado a objetos organiza o código em torno de objetos, que combinam dados e comportamentos. Exemplos incluem C++, Java e Python.

Características principais:

- 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.

Comparação:

- 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 organiza o estado em objetos.

5 Tutorial

5.1 Linux

Antes de iniciar a instalação, uma boa prática é verificar com antecedência se o GCC já se encontra instalado em sua máquina.

Para isso podemos abrir um terminal e digitar o comando:

```
$ gcc --version
```

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

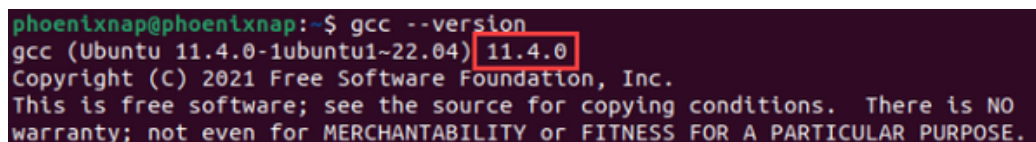
A terminal window with a dark background. The prompt is 'phoenixnap@phoenixnap:~'. The command entered is 'gcc --version'. The output is 'gcc (Ubuntu 11.4.0-1ubuntu1~22.04) 11.4.0'. The version number '11.4.0' is highlighted with a red box. Below the version number, the text 'Copyright (C) 2021 Free Software Foundation, Inc.' is displayed. At the bottom, a disclaimer states: 'This is free software; see the source for copying conditions. There is NO warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.'

Figure 1: Informações Linux

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

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** para instalar o GCC.

Abrindo um terminal você deve executar os seguintes comandos:

```
$ 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.

5.2 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

- (a) Vá para a página de downloads do MSYS2(<https://www.msys2.org/>)
- (b) Baixe o instalador adequado para seu sistema(provavelmente o arquivo.exe)

2. Instalar MSYS2

- (a) Execute o instalador e siga as instruções na tela para instalar o MSYS2.
- (b) Durante a instalação, escolha um diretório de instalação

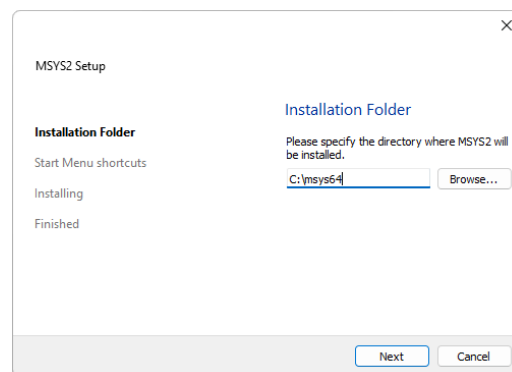


Figure 2: Diretório de instalação

3. Atualizar o MSYS2

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

- (a) 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

- (b) 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.

6 Exercícios

6.1 Questões Teóricas

1. Analise as afirmativas abaixo.

- 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

2. 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.
3. 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.
4. 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 interativo, 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.
 - (e) se todas as afirmativas estiverem corretas.
5. 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.

6.2 Questões de Código

1. Analise o trecho de código abaixo.

```
#include <stdio.h>

int main(void)
{
    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?

2. 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;
}

```

3. 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));
}

```

4. O que será impresso ao fim do programa?

```

#include <stdio.h>

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

    if (a && b || c) {
        printf("Verdadeiro\n");
    } else {
        printf("Falso\n");
    }
    return 0;
}

```

5. O programa possui um erro, onde ele está?

```

#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;  
    }
```

7 Respostas

7.1 Teóricas

1. Letra C
2. Letra C
3. Letra C
4. Letra B
5. Letra C

7.2 Práticas

1. 0, 2, 4, 0, 8
2. Na verificação do valor de divisores no segundo if da função primo, o correto seria maior igual a 2
3. 20
4. Verdadeiro
5. A variável Z está sendo comparada com a expressão, e não recebendo um valor calculado pela expressão

8 Fontes

8.1 Sites

- <https://www.geeksforgeeks.org/introduction-of-programming-paradigms/>
- <https://people.cs.aau.dk/~normark/prog3-03/pdf/paradigms.pdf>
- <https://www.editorasynapse.org/wp-content/uploads/2021/03/paradigmasprogramacaouniversitaria.pdf>
- <https://pt.wikipedia.org/wiki/Programacaoimperativa>

- <https://pt.wikipedia.org/wiki/Paradigmadeprogramacao>
- <https://en.wikipedia.org/wiki/Programmingparadigm>
- <https://pt.wikipedia.org/wiki/Programaçãoprocedural>
- <https://archive.org/details/paradigmasprogramacaoumainducaoV0/page/8/mode/1up>
- [https://pt.wikipedia.org/wiki/Java\(linguagemdeprogramação\)](https://pt.wikipedia.org/wiki/Java(linguagemdeprogramação))
- [https://pt.wikipedia.org/wiki/C\(linguagemdeprogramação\)](https://pt.wikipedia.org/wiki/C(linguagemdeprogramação))
- <https://pt.wikipedia.org/wiki/Linguagemassembly>
- <https://pt.wikipedia.org/wiki/ALGOL>
- <https://pt.wikipedia.org/wiki/Fortran>
- <https://pt.wikipedia.org/wiki/AlanTuring>
- <https://pt.wikipedia.org/wiki/JohnvonNeumann>
- <https://guia.dev/pt/pillars/languages-and-tools/programming-paradigms.html>
- <https://pt.wikipedia.org/wiki/COBOL>
- <https://blog.geekhunter.com.br/quais-sao-os-paradigmas-de-programacao/>
- <https://pt.wikipedia.org/wiki/Programação logica>
- <https://pt.wikipedia.org/wiki/Programação funcional>
- <https://pt.wikipedia.org/wiki/Programação orientada a objetos>
- <https://medium.com/trainingcenter/programação-funcional-para-iniciantes-9e2beddb5b43>
- <https://learn.microsoft.com/pt-br/dotnet/standard/linq/functional-vs-imperative-programming>
- <https://www.linkedin.com/pulse/paradigma-declarativo-vs-imperativo-henrique-santos/>
- <https://guia.dev/pt/pillars/languages-and-tools/programming-paradigms.html>

8.2 Livros

- Abelson, H., & Sussman, G. J. (1985). Structure and Interpretation of Computer Programs.
- Tucker, A. B., & Noonan, R. E. (2002). Programming Languages: Principles and Paradigms.
- Sebesta, R. W. (2012). Concepts of Programming Languages.
- Chiusano, P., & Bjarnason, R. (2014). Functional Programming in Scala.
- Clocksin, W.F., & Mellish, C. S. Programming in Prolog: Using the ISO Standard.

8.3 Documentação

1. Documentação do C(ISO/IEC 9899).

8.4 Conteúdo em vídeo

1. <https://www.youtube.com/playlist?list=PLsri1g4fxrjuf6UCYHqCmqsfXR4gofAFH>