**Resumo Paradigmas de linguagem de programação**

**Paradigma imperativo**

Paradigma imperativo (assemelha-se à máquina de Turing e Von Neumman) é baseado na execução sequencial de comandos e no armazenamento de dados na memória do computador (variáveis).

Variável possui um identificador com um endereço de memória, ela possui tipo, valor e tempo de vida.

**Tipos primitivos:** inteiros, reais, booleanos e caracteres.

**Tipos compostos:** ponteiros, cadeira de caracteres, matrizes e estruturas/registros.

**Linguagem C**

**Tipagem estática:** valor de uma expressão é determinado por tempo de compilação, ou seja, um tipo associado a uma variável declarada, permanece associado até o fim do programa.

**Tipagem dinâmica:** valor de uma expressão é determinado por tempo de execução, ou seja, tipo de uma variável pode ser redefinido cada vez que um novo valor for atribuído.

**Ponteiro:** referência ou endereço de memória.

**& :** operador endereço “de”, que retorna o endereço da variável.

**\*:** recebe uma referência e acessa o valor desta referência (conteúdo).

**Estruct:** conjunto de elementos de tipos diferentes.

data \*p; //p é um ponteiro para data

data hoje;

p = &hoje; //agora p aponta para hoje

(\*p).dia = 31; //agora o conteúdo de p vale 31

**Tempo de vida estático:** variáveis globais, alocadas em memória no início da execução e permanecem lá até que o programa termine de executar.

**Tempo de vida dinâmico:** variáveis alocadas a partir da pilha de execução, e depois de terminar a execução da função elas são liberadas.

**Funções**

void troca (int x, int y) // x e y são parâmetros

troca (a, b); // a e b são argumentos

**Passagem de argumento por valor:** valor do argumento é calculado no tempo da chamada e copiado para o parâmetro correspondente.

**Passagem por referência ou por endereço:** significa que o endereço de memória do argumento é copiado para o parâmetro, de modo que o parâmetro se torna um ponteiro indireto ao argumento real.

**Gerenciamento de memória**

|  |
| --- |
| **Área Estática** |
| **Pilha** |
| **Heap** |

**Memória estática:** contém valores conhecidos antes do tempo de execução e permanecem constantes por toda vida do programa (variáveis globais).

**Pilha:** são armazenadas variáveis declaradas localmente (dentro da função) e a ligação parâmetro-argumento (chamada das funções).

**Memória Heap:** contém valores que são alocados e estruturados dinamicamente, como vetores e matrizes dinâmicas.

A função **calloc()** aloca um bloco contíguo de bytes e os preenche com valor 0.

**void \*calloc(int n\_elementos, int tamanho\_elemento)**

Se quisermos alocar um bloco de endereço para inteiros, colocamos int\* antes do calloc.

Se a função calloc não consegue alocar espaço ela retorna NULL.

A função **free()** desaloca o bloco contíguo de bytes.

**Linguagem Java**

**Encapsulamento:** constantes logicamente relacionadas sejam agrupadas em uma entidade (classe).

**Funções são métodos.**

Cada instância de uma classe é um objeto.

Operador **new** aloca espaço em memória com parâmetros passados no processo de instanciação.

**Setter e getters** caso o objeto seja private.