ALGORITMOS III

Prof. Ms. Ronan Loschi.

ronan.loschi@unifasar.edu.br

31-98759-9555

UNIFASAR
CENTRO UNIVERSITÁRIO SANTA RITA

ALOCAÇÃO DINÂMICA DE MEMÓRIA

C++

DEFINIÇÃO:

- Existem duas maneiras de um programa em **C++** armazenar informações na memória.

- A primeira é através da utilização de **variáveis locais e globais**.

- Como visto anteriormente, uma **variável global**, existe na memória enquanto o programa estiver sendo executado e pode ser acessada ou modificada por qualquer parte do programa.

- Já, a **variável local** é acessível, apenas dentro de um procedimento ou função (ou dentro de um componente de formulário), ou seja, criada e usada no momento em que for necessária.

DEFINIÇÃO:

- A segunda maneira de armazenar informações na memória é utilizar **alocação dinâmica**.

- Desta forma o programa usa **espaços da memória que podem variar de tamanho de acordo com o que se precisa armazenar**.

- Isto ocorre em **tempo de execução** do programa.

- A alocação dinâmica está diretamente ligada ao uso de **ponteiros**, pois a memória alocada não possui um nome específico, sendo referenciada **apenas** pelo ponteiro.

INTRODUÇÃO:

Benefícios:

- Flexibilidade: Alocar memória conforme necessário.
- Eficiência: Evitar desperdício de memória.
- **Controle**: Gerenciar a memória de forma mais precisa.
- Permite **manipulação flexível de estruturas** de dados (listas, árvores, etc.).

PALAVRAS-CHAVE NEW E DELETE

new: Aloca memória dinamicamente .

delete: Libera memória alocada para evitar vazamento.

SINTAXE: **int* ptr = new int; // Aloca memória para um inteiro**

delete ptr; // Libera a memória alocada

EXEMPLO:

```
int* ptr = new int;    // Aloca memória para um inteiro
*ptr = 10;             // Atribui um valor
cout << *ptr;    // Imprime 10
delete ptr;            // Libera a memória alocada
ptr = nullptr;         // Evita ponteiro perdido
```

UNIFASAR
CENTRO UNIVERSITÁRIO SANTA RITA

EXEMPLO DE APLICAÇÃO:

```cpp
#include <iostream>
#include <locale.h>
using namespace std;

int main() {
    setlocale(LC_ALL, "");
    //Declaração dos ponteiros int
    int* n1;
    int* n2;
    int* n3;
    int* soma;
    float* media;

    // Alocação dinâmica do inteiro
    n1 = new int;
    n2 = new int;
    n3 = new int;
    soma = new int;
    media = new float;
```

```cpp
    // Solicita o valor ao usuário
    cout << "Digite o primeiro número: ";
    cin >> *n1;
    cout << "Digite o segundo número: ";
    cin >> *n2;
    cout << "Digite o pterceiro número: ";
    cin >> *n3;
    *soma=*n1+*n2+*n3;
    *media=*soma/3;

    // Imprimindo o valor da média
    cout << "O valor da média é: " << *media << endl;

    // Liberação da memória
    delete n1;
    delete n2;
    delete n3;
    delete soma;
    delete media;

    return 0;
}
```

UNIFASAR
CENTRO UNIVERSITÁRIO SANTA RITA

Alocação de Arrays Dinâmicos:

Arrays também podem ser alocados dinamicamente.

Use new tipo [ ] para **alocar** um array.

Use delete [ ] para **liberar** a memória do array.

Sintaxe para VETORES:

```
int* arr = new int [10]; // Aloca um array de 10 inteiros

delete[ ] arr; // Libera a memória do array
```

UNIFASAR
CENTRO UNIVERSITÁRIO SANTA RITA

Exemplo de VETOR dinâmico:

```
int* arr = new int[ 5 ]; // Aloca um array de 5 inteiros

for (int i = 0; i < 5; i++) {
            arr[ i ] = i * 2;
}
for (int i = 0; i < 5; i++) {
            cout << arr[ i ] << “  ";
}

delete[  ] arr;      // Libera a memória alocada
arr = nullptr;    // Boa prática para evitar acesso a memória liberada
```

EXEMPLO 1 DE APLICAÇÃO COM VETOR:

```cpp
#include <iostream>
using namespace std;

int main() {
    int* ptr = new int; // Aloca um inteiro
    *ptr = 10; // Atribui valor
    cout << "Valor: " << *ptr << endl;
    delete ptr; // Libera a memória

    int* arr = new int[5]; // Aloca um array de 5 inteiros
    for (int i = 0; i < 5; i++) {
        arr[i] = i * 10;
    }
    for (int i = 0; i < 5; i++) {
        cout << "Elemento " << i << ": " << arr[i] << endl;
    }
    delete[] arr; // Libera a memória do array

    return 0;
}
```

EXEMPLO 2 DE APLICAÇÃO COM VETOR:

```cpp
#include <iostream>
#include <locale.h>
using namespace std;

int main() {
    int tam;
     setlocale(LC_ALL, "");

    // Solicita o tamanho do vetor ao usuário
    cout << "Digite o tamanho do vetor: ";
    cin >> tam;

    // Alocação dinâmica do vetor
    int* vetor = new int[tam];

    // Preenchendo o vetor
    for (int i = 0; i < tam; i++) {
        cout << "Digite o "<< i+1 <<"º valor inteiro do vetor: ";
        cin >> vetor[i];
    }

    // Imprimindo o vetor
    cout << "Elementos do vetor:" << endl;
    for (int i = 0; i < tam; i++) {
        cout << "Elemento " << i << ": " << vetor[i] << endl;
    }

    // Liberação da memória
    delete[] vetor;

    return 0;
}
```

UNIFASAR
CENTRO UNIVERSITÁRIO SANTA RITA

Alocação de Matriz Dinâmicos:

Matriz também podem ser alocados dinamicamente.

Use new [ ] para **alocar** uma matriz.

Use delete [ ] para **liberar** a memória da matriz.

Sintaxe para MATRIZ:

int** matriz = **new** int*[LINHAS]; // Aloca as linhas

matriz[i] = **new** int[COLUNAS]; // Aloca as colunas para cada linha dentro do FOR

MATRIZ :

Representação de Matrizes:

Matriz Estática:

int matrix[3][4]; //A memória é contígua e o compilador sabe o tamanho exato.

Matriz Dinâmica:

int** matrix = new int*[LINHAS];

Cada linha é um **ponteiro** para um VETOR de inteiros.

A memória não é necessariamente contígua.

MATRIZ :

Explicando:

Por que int**?

int*: // Um **ponteiro** para **um inteiro**.

Exemplo: int* ptr = new int;

int**: // Um **ponteiro para um ponteiro** para um inteiro.
Exemplo: int** matrix = new int*[LINHAS];.

DEFINIÇÃO:

```
// Alocação dinâmica da matriz

int** matrix = new int*[LINHAS];

for (int i = 0; i < LINHAS; i++) {
        matrix[ i ] = new int [COLUNAS];
}
```

Exemplo de matriz dinâmica::

```
int linhas = 3, colunas = 4;
int** matriz = new int*[linhas]; // Aloca as linhas

for (int i = 0;  i < linhas;  i++) {
    matriz[ i ] = new int [coluna]; // Aloca as colunas para cada linha
}

// Preenchendo a matriz
for (int i = 0; i < linhas; i++) {
     for (int j = 0; j < colunas; j++) {
            matriz[ i ][ j ] = i + j;
    }
}
// Liberando a memória
for (int i = 0; i < linhas; i++) {
      delete[    ] matriz[  i  ];
}
delete[   ] matriz;
```

EXEMPLO 1 DE APLICAÇÃO COM MATRIZ:

```cpp
#include <iostream>
using namespace std;

int main() {
    int rows = 3, columns = 4;

    // Alocação dinâmica da matriz
    int** matrix = new int*[rows];
    for (int i = 0; i < rows; i++) {
        matrix[i] = new int[columns];
    }

    // Preenchendo a matriz
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            matrix[i][j] = i * columns + j;
        }
    }

    // Imprimindo a matriz
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < columns; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    // Liberação da memória
    for (int i = 0; i < rows; i++) {
        delete[] matrix[i];
    }
    delete[] matrix;

    return 0;
}
```

UNIFASAR
CENTRO UNIVERSITÁRIO SANTA RITA

EXEMPLO 2 DE APLICAÇÃO COM MATRIZ:

```cpp
#include <iostream>
using namespace std;

int main() {
    int linhas, colunas;

    // Solicita o número de linhas e colunas ao usuário
    cout << "Digite o número de linhas: ";
    cin >> linhas;
    cout << "Digite o número de colunas: ";
    cin >> colunas;

    // Alocação dinâmica da matriz
    int** matrix = new int*[linhas];
    for (int i = 0; i < colunas; i++) {
        matrix[i] = new int[colunas];
    }

    // Preenchendo a matriz
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            matrix[i][j] = i * colunas + j;
        }
    }

    // Imprimindo a matriz
    cout << "Matriz:" << endl;
    for (int i = 0; i < linhas; i++) {
        for (int j = 0; j < colunas; j++) {
            cout << matrix[i][j] << " ";
        }
        cout << endl;
    }

    // Liberação da memória
    for (int i = 0; i < linhas; i++) {
        delete[] matrix[i];
    }
    delete[] matrix;

    return 0;
}
```

UNIFASAR
CENTRO UNIVERSITÁRIO SANTA RITA

Estruturas e Objetos Dinâmicos:

Podemos alocar dinamicamente estruturas e objetos

EXEMPLO:

```
struct Aluno {
        string nome;
        int idade;
};
Aluno* estudante = new Aluno;
estudante->nome = "Carlos";
estudante->idade = 21;
cout << estudante->nome << " tem " << estudante->idade << " anos.";
delete estudante;
estudante = nullptr;
```

EXEMPLO 1 - Estruturas e Objetos Dinâmicos:

```cpp
#include <iostream>
#include <string>
using namespace std;

struct Aluno {
    string nome;
    int idade;
};

int main() {
    // Alocação dinâmica de um objeto do tipo Aluno
    Aluno* estudante = new Aluno;

    // Atribuindo valores
    estudante->nome = "Carlos";
    estudante->idade = 21;

    // Exibindo os valores
    cout << estudante->nome << " tem " << estudante->idade << " anos." <<endl;

    // Liberando memória
    delete estudante;
    estudante = nullptr; // Boa prática para evitar ponteiro perdido

    return 0;
}
```

```cpp
#include <iostream>
#include <string>
#include <locale.h>>
using namespace std;

struct Aluno {
    string nome, sexo;
    int idade, cpf;
};

int main() {
    setlocale (LC_ALL,"");
    // Alocação dinâmica de um objeto do tipo Aluno
    Aluno* estudante = new Aluno;

    // Solicitando valores
    cout << " Digite o nome: " <<endl;
    cin>> estudante->nome;
    cout << " Digite o sexo: " <<endl;
    cin>> estudante->sexo;
    cout << " Digite a idade: " <<endl;
    cin>> estudante->idade;
    cout << " Digite o CPF (somente números): " <<endl;
    cin>> estudante->cpf;

    // Exibindo os valores
    cout << estudante->nome << " tem " << estudante->idade << " anos."  <<endl;
    cout << "E é do sexo " <<estudante->sexo <<" e tem CPF "<<estudante->cpf <<endl;

    // Liberando memória
    delete estudante;
    estudante = nullptr; // Boa prática para evitar ponteiro perdido

    return 0;
}
```

EXEMPLO 2 - Estruturas e Objetos Dinâmicos:

UNIFASAR
CENTRO UNIVERSITÁRIO SANTA RITA

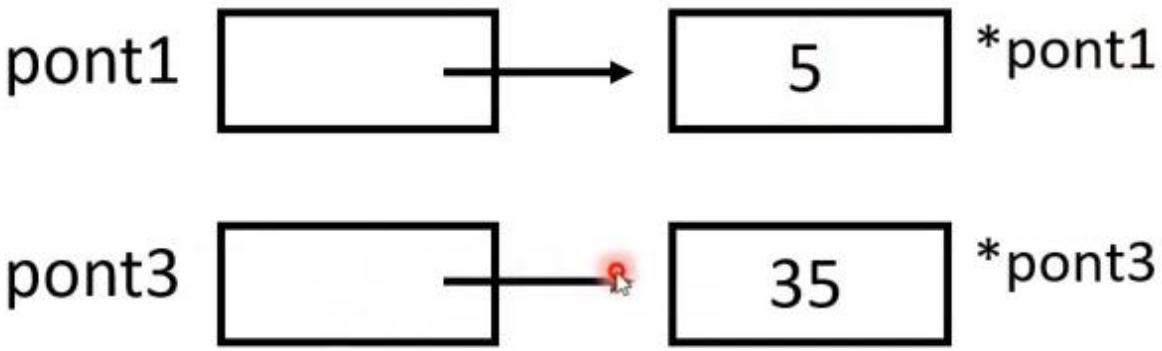
BOAS PRÁTICAS:

- Sempre libere a memória de cada linha antes de liberar o array de ponteiros.
- Evite acessar memória já liberada.
- Use nullptr para inicializar ponteiros não alocados.

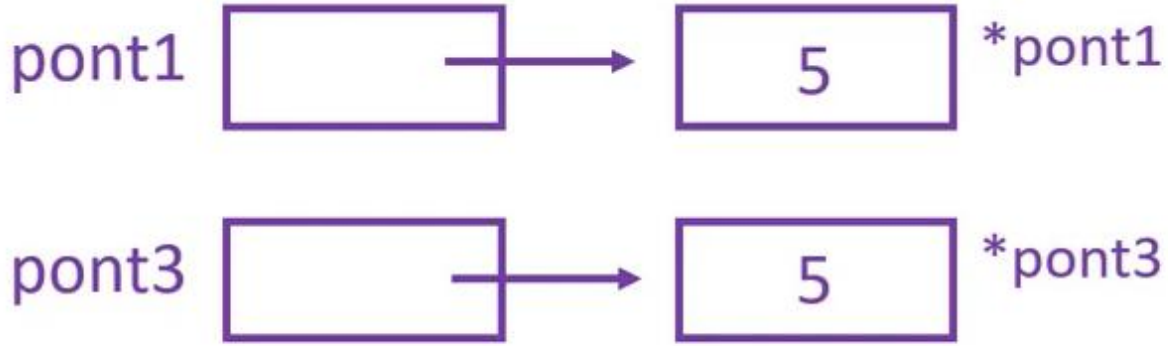
PROBLEMAS COMUNS:

- **Vazamento de memória:** Ocorre quando esquecemos de liberar memória alocada.
- **Acesso a memória liberada:** Tentar acessar um ponteiro deletado pode causar comportamento indefinido.
- **Ponteiro "dangling" (perdido):** Aponta para uma região de memória inválida.

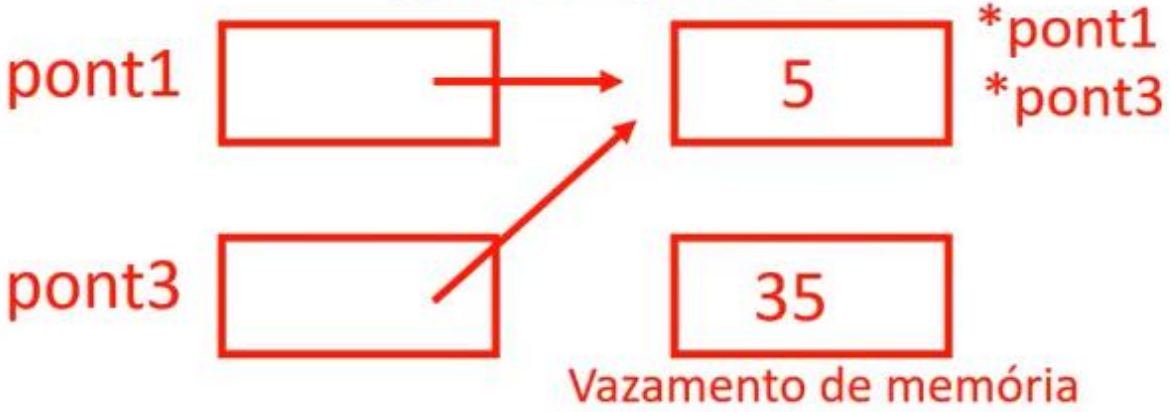
Ponteiros



*pont3 = *pont1



pont3 = pont1



EXRCÍCIOS

UNIFASAR
CENTRO UNIVERSITÁRIO SANTA RITA

Exercício 1: Média das Alturas (Alocação Básica)

Enunciado: Crie um programa que aloque dinamicamente memória para armazenar a altura de 10 pessoas. Solicite ao usuário que insira as alturas, calcule a média e exiba o resultado. Libere a memória alocada ao final.

Detalhes: Usar new e delete para alocação e desalocação.

Exercício 2: Vetor Dinâmico de Números

Enunciado: Implemente um programa que aloque dinamicamente um vetor para armazenar n números inteiros (onde n é definido pelo usuário). Peça ao usuário para inserir os números, em seguida, calcule e exiba a soma dos elementos no vetor. Libere a memória após o uso.

Detalhes: Alocar um array usando new int[n] e liberar com delete[].

Exercício 3: Matriz Dinâmica de Notas

Enunciado: Desenvolva um programa que aloque dinamicamente uma matriz para armazenar as notas de m alunos em n disciplinas (tanto m quanto n são definidos pelo usuário). Solicite ao usuário que insira as notas e, em seguida, calcule e exiba a média de cada aluno. Libere a memória alocada.

Detalhes: Alocação de um array de ponteiros, cada um apontando para uma linha da matriz. Desalocação em ordem inversa.

Exercício 4: Struct com Alocação Dinâmica

Enunciado: Defina uma struct Aluno contendo nome (string alocada dinamicamente) e nota (float). Crie um programa que aloque dinamicamente um array de structs Aluno para k alunos (definido pelo usuário). Solicite ao usuário que insira o nome e a nota de cada aluno. Exiba os dados de cada aluno e libere a memória.

Detalhes: Alocar memória para a struct Aluno e para a string nome dentro da struct.