**Estruturas de Dados**

**Exercícios**

**- Recursividade**

1) Faça um programa que apresente um menu inicial com as seguintes opções:

MENU PRINCIPAL

1 – Funções sem vetor

2 – Funções com vetor

3 - Sair

Digite a opção desejada: \_

Se o usuário escolher a opção 1, solicite um número inteiro inicial e um número inteiro final. O número final deve ser maior que o inicial. Em seguida apresente um segundo menu com as seguintes opções:

MENU SECUNDÁRIO

1 - Inteiros em ordem crescente

2 - Inteiros em ordem decrescente

3 - Inteiros ímpares (crescente)

4 - Somatório dos inteiros

Faça uma função recursiva para cada opção desse segundo menu.

Se o usuário escolher a opção 2 (do menu inicial), solicite vários números inteiros (armazene num vetor) e em seguida faça uma função recursiva que calcule o somatório dos números do vetor.

2) Escreva uma função recursiva chamada potencia(x, y), que retorne a base x elevado ao expoente y.

using System;

public class Potencia {

    public static double potencia(double x, int y) {

        if (y == 0) {

            return 1;

        }

        else if (y == 1) {

            return x;

        }

        else if (y < 0) {

            return 1 / potencia(x, -y);

        }

        else if (y % 2 == 0) {

            return potencia(x\*x, y/2);

        }

        else {

            return x \* potencia(x\*x, (y-1)/2);

        }

    }

    public static void Main() {

        double resultado = potencia(2, 3);

        Console.WriteLine(resultado);

    }

}

3) Implemente uma versão recursiva da seguinte função iterativa. Faça a repetição recursiva somente do for, não precisa fazer da expressão i \* i \* i.

void cubos (int n)

{

for (int i = 1; i <= n; i++)

Controle.WrilteLine(i \* i \* i);

}

using System;

public class Cubos\_recursivos {

    public static void Cubos(int n) {

        CubosRecursivo(n, 1);

    }

    public static void CubosRecursivo(int n, int i) {

        if (i <= n) {

            Console.WriteLine(i \* i \* i);

            CubosRecursivo(n, i + 1);

        }

    }

    public static void Main() {

        Cubos(5);

    }

}

4) Baseado no algoritmo de Euclides, implemente uma função recursiva para determinar o máximo divisor comum (mdc) entre dois números inteiros x e y.

Algoritmo de Euclides:

se (x = y) retorna x

senão se (x < y) retorna mdc(y, x)

senão retorna mdc(x - y, y)

using System;

class Mdc {

    static void Main(string[] args) {

        int x = 12;

        int y = 18;

        int mdc = MDC(x, y);

        Console.WriteLine("O MDC de {0} e {1} é: {2}", x, y, mdc);

    }

    static int MDC(int x, int y) {

        if (x == y) {

            return x;

        }

        else if (x < y) {

            return MDC(y, x);

        }

        else {

            return MDC(x - y, y);

        }

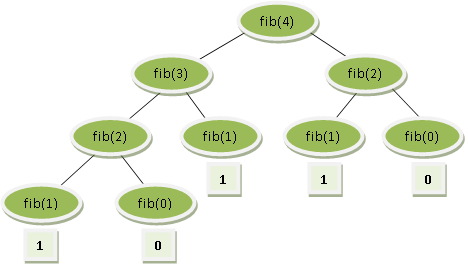
    }

}

5) A Série de Fibonacci pode ser definida da seguinte maneira:

se (n == 0 ou n == 1) retorna n

se (n >= 2) retorna fib(n – 1) + fib(n – 2)



Seja fib(n) uma função que retorna o n-ésimo termo da série de Fibonacci, implemente uma versão recursiva e outra iterativa. Observe que a partir de um determinado número, a função recursiva começa a ficar mais lenta que a iterativa.

Função recursiva:

using System;

class Program {

    static void Main() {

        for (int i = 0; i <= 10; i++) {

            Console.WriteLine($"FibRecursive({i}) = {FibRecursive(i)}");

        }

    }

    public static int FibRecursive(int n) {

        if (n == 0 || n == 1) {

            return n;

        }

        else

            return FibRecursive(n - 1) + FibRecursive(n - 2);

        }

    }

6) Defina uma função recursiva que converte um número inteiro para a base binária.

using System;

class Program

{

    static void Main() {

        Console.Write("Digite um número inteiro: ");

        int num = int.Parse(Console.ReadLine());

        string binary = DecimalToBinary(num);

        Console.WriteLine($"O número {num} em binário é: {binary}");

    }

    public static string DecimalToBinary(int n) {

        if (n == 0) {

            return "0";

        }

        else if (n == 1) {

            return "1";

        }

        else {

            string binary = DecimalToBinary(n / 2);

            if (n % 2 == 0) {

                return binary + "0";

            }

            else {

                return binary + "1";

            }

        }

    }

}

7) Qual o valor de retorno da função a seguir, caso n = 27?

int recursao (int n) {

if (n <= 10) {

return n \* 2;

}

else {

return recursao(recursao(n/3));

}

}

A) 8

B) 9

C) 12

X D) 16

E) 18

**- Lista linear sequencial: Pilha**

8) Ilustre a situação (memória) da lista linear sequencial do tipo pilha, inicialmente vazia, após a execução de cada uma das operações a seguir:

1. Insere(pilha, ref topo, 20);

+---+---+---+---+---+---+---+---+---+---+

| 20| | | | | | | | | |

+---+---+---+---+---+---+---+---+---+---+

^ ^

| |

topo base

1. Insere(pilha, ref topo, 30);
2. Insere(pilha, ref topo, 40);
3. v = Remove(pilha, ref topo);
4. Insere(pilha, ref topo, 50);
5. Insere(pilha, ref topo, v);
6. Remove(pilha, ref topo);
7. Remove(pilha, ref topo);

9) Faça um programa que converta um número inteiro da base decimal para a base binária.

using System;

class Conversao {

    static void Main() {

        Console.Write("Digite um número inteiro em decimal: ");

        int decimalNumber = int.Parse(Console.ReadLine());

        string binaryNumber = "";

        while (decimalNumber > 0) {

            int remainder = decimalNumber % 2;

            binaryNumber = remainder.ToString() + binaryNumber;

            decimalNumber = decimalNumber / 2;

        }

        Console.WriteLine("O número em binário é: " + binaryNumber);

    }

}

10) Escreva um programa para conhecer uma frase e exibi-la com as PALAVRAS invertidas. Por exemplo, a frase “Jose da Silva” deve sair “esoJ ad avliS”.

using System;

class Program {

    static void Main(string[] args) {

        Console.Write("Digite uma frase: ");

        string frase = Console.ReadLine();

        string[] palavras = frase.Split(' ');

        string fraseInvertida = "";

        foreach (string palavra in palavras) {

            string palavraInvertida = "";

            for (int i = palavra.Length - 1; i >= 0; i--) {

                palavraInvertida += palavra[i];

            }

            fraseInvertida += palavraInvertida + " ";

        }

        Console.WriteLine("Frase invertida: " + fraseInvertida.TrimEnd());

    }

}

11) Elabore um programa que verifica se uma dada expressão é válida em relação aos abre e fecha parênteses, colchetes e chaves. Exemplos:

a) a = b + (c -d) \* (e – f) - válida

b) b = [c – d) - inválida

c) while (m < (n[8] + o)) { m = m + 1; } - válida

d) for (i = 1; i <= tl; i++ - inválida

using System;

using System.Collections.Generic;

class Verificando\_expressao\_valida {

    static void Main() {

        string expressao = "while (m < (n[8] + o)) { m = m + 1; }";

        if (VerificarExpressao(expressao)) {

            Console.WriteLine("Expressão válida!");

        } else {

            Console.WriteLine("Expressão inválida!");

        }

    }

    static bool VerificarExpressao(string expressao) {

        Dictionary<char, char> pares = new Dictionary<char, char>();

        pares.Add('(', ')');

        pares.Add('[', ']');

        pares.Add('{', '}');

        Stack<char> pilha = new Stack<char>();

        foreach (char caractere in expressao) {

            if (pares.ContainsKey(caractere)) {

                pilha.Push(caractere);

            }

            if (pares.ContainsValue(caractere))

                if (pilha.Count == 0 || pares[pilha.Peek()] != caractere) {

                    return false;

                }

                pilha.Pop();

            }

        }

        return (pilha.Count == 0);

    }

}

12) Desenvolva um algoritmo para conversão de expressões da notação tradicional (in-fixa) para a notação polonesa (pós-fixa). As expressões são compostas somente por parênteses, operandos de apenas uma letra e os operadores de soma, subtração e multiplicação.

Como entrada de dados o usuário informa a expressão in-fixa a ser analisada, como saída de dados seu algoritmo deve apresentar a mesma expressão na notação pós-fixa.

Entrada

A+B

A+B–C

(A+B)\*(C–D)

(A+B\*C)

Saída

AB+

AB+C–

AB+CD-\*

ABC\*+

**- Lista linear sequencial: Fila**

using System;

using System.Collections.Generic;

class Program {

    static void Main(string[] args) {

        string expressaoInfixa = "(A+B)\*(C-D)";

        string expressaoPosfixa = ConverterParaPosfixa(expressaoInfixa);

        Console.WriteLine(expressaoPosfixa);

    }

    static int ObterPrecedencia(char operador) {

        switch (operador) {

            case '+':

            case '-':

                return 1;

            case '\*':

                return 2;

            default:

                return 0;

        }

    }

    static string ConverterParaPosfixa(string expressaoInfixa) {

        List<char> fila = new List<char>();

        string expressaoPosfixa = "";

        for (int i = 0; i < expressaoInfixa.Length; i++) {

            char caractere = expressaoInfixa[i];

            if (Char.IsLetter(caractere)) {

                expressaoPosfixa += caractere;

            }

            else if (caractere == '(') {

                fila.Add(caractere);

            }

            else if (caractere == ')') {

                while (fila.Count > 0 && fila[fila.Count - 1] != '(') {

                    expressaoPosfixa += fila[fila.Count - 1];

                    fila.RemoveAt(fila.Count - 1);

                }

                fila.RemoveAt(fila.Count - 1);

            }

            else {

                while (fila.Count > 0 && fila[fila.Count - 1] != '(' &&

                       ObterPrecedencia(caractere) <=

ObterPrecedencia(fila[fila.Count - 1])) {

                    expressaoPosfixa += fila[fila.Count - 1];

                    fila.RemoveAt(fila.Count - 1);

                }

                fila.Add(caractere);

            }

        }

        while (fila.Count > 0) {

            expressaoPosfixa += fila[fila.Count - 1];

            fila.RemoveAt(fila.Count - 1);

        }

        return expressaoPosfixa;

    }

}

13) Ilustre a situação (memória) da lista linear sequencial do tipo fila, inicialmente vazia, após a execução de cada uma das seguintes operações:

1. Insere(fila, ref fim, 20);
2. Insere(fila, ref fim, 30);
3. Insere(fila, ref fim, 40);
4. Remove(fila, ref inicio);
5. Remove(fila, ref inicio);
6. Insere(fila, ref fim, 50);
7. Remove(fila, ref inicio);
8. Insere(fila, ref fim, 60);
9. Insere(fila, ref fim, 70);
10. v = Remove(fila, ref inicio)
11. Insere(fila, ref fim, v);

14) Escreva um programa que tenha uma fila cujo valores indicam prioridade, quanto maior o valor, maior a prioridade.

Seu programa deve inserir vários valores numa fila e solicitar um valor divisor. Em seguida, o programa deve dividir a fila inicial em duas novas filas, uma fila com valores menores que o divisor (menor prioridade) e outra fila com os valores maiores que o divisor (maior prioridade).

Por fim, informe os valores das duas novas filas.

using System;

class Fila\_prioridade {

    static void Main(string[] args) {

        int[] filaPrioridade = { 10, 5, 15, 20, 8, 12 };

        Console.WriteLine("Fila original:");

        foreach (int valor in filaPrioridade) {

            Console.WriteLine(valor);

        }

        Console.Write("\nDigite um valor divisor: ");

        int divisor = int.Parse(Console.ReadLine());

        int[] filaMaiorPrioridade = new int[filaPrioridade.Length];

        int[] filaMenorPrioridade = new int[filaPrioridade.Length];

        int tamanhoFilaMaior = 0;

        int tamanhoFilaMenor = 0;

        foreach (int valor in filaPrioridade) {

            if (valor > divisor) {

                filaMaiorPrioridade[tamanhoFilaMaior++] = valor;

            } else {

                filaMenorPrioridade[tamanhoFilaMenor++] = valor;

            }

        }

        Console.WriteLine("\nFila de maior prioridade:");

        for (int i = 0; i < tamanhoFilaMaior; i++) {

            Console.WriteLine(filaMaiorPrioridade[i]);

        }

        Console.WriteLine("\nFila de menor prioridade:");

        for (int i = 0; i < tamanhoFilaMenor; i++) {

            Console.WriteLine(filaMenorPrioridade[i]);

        }

    }

}

15) Escreva um programa que insira vários números numa fila. Após a digitação dos números, seu programa deve encontrar o maior, o menor e a média aritmética dos números da fila. Por fim, informe os resultados encontrados.

using System;

using System.Collections.Generic;

class Maior\_menor\_media\_aritmetica {

    static void Main(string[] args) {

        List<double> fila = new List<double>();

        while (true) {

            Console.Write("Digite um número (ou 'fim' para encerrar): ");

            string numero = Console.ReadLine();

            if (numero == "fim") {

                break;

            }

            try {

                fila.Add(double.Parse(numero));

            } catch (FormatException) {

                Console.WriteLine("Valor inválido. Digite um número ou 'fim' para encerrar.");

            }

        }

        if (fila.Count == 0) {

            Console.WriteLine("Nenhum número foi digitado.");

        } else {

            double maior = fila.Max();

            double menor = fila.Min();

            double media = fila.Average();

            Console.WriteLine("O maior número é: " + maior);

            Console.WriteLine("O menor número é: " + menor);

            Console.WriteLine("A média aritmética dos números é: " + media);

        }

    }

}

16) Escreva um programa que simule o controle de uma pista de decolagem de aviões em um aeroporto. Os aviões são identificados pelos números digitados pelo usuário. Neste programa, o usuário deve ser capaz de realizar as seguintes operações:

a) Adicionar um avião na fila de espera para decolagem

b) Consultar a quantidade de aviões aguardando na fila

c) Autorizar a decolagem de um avião da fila

d) Listar os números de todos os aviões na fila

e) Consultar o número do primeiro avião da fila

Construa um menu principal para oferecer essas operações ao usuário.

using System;

class Program {

    static void Main(string[] args) {

        const int MAX\_AVIOES = 100;

        string[] filaAvioes = new string[MAX\_AVIOES];

        int inicioFila = 0;

        int fimFila = 0;

        while (true) {

            Console.WriteLine("Escolha uma opção:");

            Console.WriteLine("a) Adicionar um avião na fila de espera");

            Console.WriteLine("b) Consultar a quantidade de aviões aguardando na fila");

            Console.WriteLine("c) Autorizar a decolagem de um avião da fila");

            Console.WriteLine("d) Listar os números de todos os aviões na fila");

            Console.WriteLine("e) Consultar o número do primeiro avião da fila");

            Console.WriteLine("f) Encerrar o programa");

            string opcao = Console.ReadLine();

            switch (opcao.ToLower()) {

                case "a":

                    if (fimFila < MAX\_AVIOES) {

                        Console.WriteLine("Digite o número do avião:");

                        string numeroAviao = Console.ReadLine();

                        filaAvioes[fimFila] = numeroAviao;

                        fimFila++;

                        Console.WriteLine($"Avião {numeroAviao} adicionado à fila de espera.");

                    } else {

                        Console.WriteLine("A fila de espera está cheia.");

                    }

                    break;

                case "b":

                    Console.WriteLine($"Há {fimFila - inicioFila} aviões aguardando na fila de espera.");

                    break;

                case "c":

                    if (inicioFila == fimFila)    {

                        Console.WriteLine("Não há aviões aguardando na fila de espera.");

                    } else {

                        string numeroAviaoDecolado = filaAvioes[inicioFila];

                        inicioFila++;

                        Console.WriteLine($"Avião {numeroAviaoDecolado} decolado.");

                    }

                    break;

                case "d":

                    Console.WriteLine("Aviões na fila de espera:");

                    for (int i = inicioFila; i < fimFila; i++) {

                        Console.WriteLine(filaAvioes[i]);

                    }

                    break;

                case "e":

                    if (inicioFila == fimFila) {

                        Console.WriteLine("Não há aviões aguardando na fila de espera.");

                    } else {

                        string numeroPrimeiroAviao = filaAvioes[inicioFila];

                        Console.WriteLine($"O primeiro avião na fila de espera é o {numeroPrimeiroAviao}.");

                    }

                    break;

                case "f":

                    Console.WriteLine("Encerrando o programa...");

                    return;

                default:

                    Console.WriteLine("Opção inválida. Tente novamente.");

                    break;

            }

            Console.WriteLine();

        }

    }

}

17) Considere que as variáveis pilha e fila correspondem, respectivamente, às estruturas de dados do tipo Pilha e Fila. Para testar as duas estruturas, um programador realizou a série de operações a seguir.

a) push(pilha, ‘A’);

b) push(pilha, ‘B’);

c) push(pilha, ‘C’);

d) enqueue(fila, top(pilha));

e) enqueue(fila, top(pilha));

f) enqueue(fila, ‘D’);

g) push(pilha, dequeue(fila));

h) enqueue(fila, dequeue(fila));

i) enqueue(fila, pop(pilha));

j) push(pilha, ‘E’);

k) enqueue(fila, ‘E’);

l) pop(pilha);

Após essas operações, ao imprimir o conteúdo de pilha e fila, respectivamente, seria exibido:

|  |  |
| --- | --- |
| A)  pilha: topo → C → A → E.  fila: início → D → A → A → E. | B)  pilha: topo →A.  fila: início → D → B → C → E. |
| C)  pilha: topo → C → B → A.  fila: início → D → C → C → E. | D)  pilha: topo → B → A.  fila: início → D → B → C → E. |
| E)  pilha: topo → C → B → A.  fila: início → D → B → C → E. |  |

**- Lista linear encadeada**

18) Ilustre a situação (memória) da lista linear encadeada do tipo pilha, inicialmente vazia, após a execução de cada uma das operações a seguir:

1. Insere(ref topo, 20);

20 (topo)

1. Insere(ref topo, 30);

30 (topo)

20

1. Insere(ref topo, 40);

40 (topo)

30

20

1. no = Remove(ref topo);

30 (topo)

20

1. no = Remove(ref topo);

20 (topo)

19) Faça uma implementação que construa uma lista encadeada. Seu programa deve ter as opções de inserção e remoção dos elementos. Após remover um elemento da lista, exiba-o na tela.

using System;

class Node {

    public int data;

    public Node next;

    public Node(int data) {

        this.data = data;

        this.next = null;

    }

}

class ListaEncadeada {

    private Node head;

    public ListaEncadeada() {

        this.head = null;

    }

    public void AdicionarUltimo(int data) {

        Node novoNo = new Node(data);

        if (this.head == null) {

            this.head = novoNo;

        } else {

            Node temp = this.head;

            while (temp.next != null) {

                temp = temp.next;

            }

            temp.next = novoNo;

        }

    }

    public void Remover(int data) {

        Node temp = this.head;

        Node anterior = null;

        bool encontrado = false;

        while (temp != null) {

            if (temp.data == data) {

                encontrado = true;

                break;

            }

            anterior = temp;

            temp = temp.next;

        }

        if (encontrado) {

            if (anterior == null) {

                this.head = temp.next;

            } else {

                anterior.next = temp.next;

            }

            Console.WriteLine("{0} foi removido da lista.", data);

        } else {

            Console.WriteLine("{0} não foi encontrado na lista.", data);

        }

    }

    public void ImprimirLista() {

        Node temp = this.head;

        while (temp != null) {

            Console.Write("{0} -> ", temp.data);

            temp = temp.next;

        }

        Console.WriteLine("null");

    }

}

class Program {

    static void Main(string[] args) {

        ListaEncadeada minhaLista = new ListaEncadeada();

        int opcao = 0;

        while (opcao != 4) {

            Console.WriteLine("Escolha uma opção:");

            Console.WriteLine("1 - Inserir elemento");

            Console.WriteLine("2 - Remover elemento");

            Console.WriteLine("3 - Imprimir lista");

            Console.WriteLine("4 - Sair");

            opcao = int.Parse(Console.ReadLine());

            switch (opcao) {

                case 1:

                    Console.WriteLine("Digite o valor a ser inserido:");

                    int valorInserir = int.Parse(Console.ReadLine());

                    minhaLista.AdicionarUltimo(valorInserir);

                    break;

                case 2:

                    Console.WriteLine("Digite o valor a ser removido:");

                    int valorRemover = int.Parse(Console.ReadLine());

                    minhaLista.Remover(valorRemover);

                    break;

                case 3:

                    Console.WriteLine("Conteúdo da lista:");

                    minhaLista.ImprimirLista();

                    break;

                case 4:

                    Console.WriteLine("Saindo...");

                    break;

                default:

                    Console.WriteLine("Opção inválida.");

                    break;

            }

        }

    }

}

20) Faça um programa que utilize lista encadeada e que tenha as opções a seguir. O nó deve conter os atributos: nome, idade, whats e prox.

1. Incluir conforme apresentado em aulas
2. Para alterar, consulte pelo nome. Se encontrar, exiba os valores atuais e permita a alteração. Caso não encontre, exiba mensagem de não encontrado.
3. Para excluir, procure pelo nome. Se encontrar, exiba os valores atuais e permita a exclusão. Caso não encontre, exiba mensagem de não encontrado.
4. Na opção exibir, exiba todos os registros.

using System;

class Node {

    public string nome;

    public int idade;

    public string whats;

    public Node prox;

    public Node(string nome, int idade, string whats) {

        this.nome = nome;

        this.idade = idade;

        this.whats = whats;

        this.prox = null;

    }

}

class Lista {

    private Node primeiro;

    public Lista() {

        this.primeiro = null;

    }

    public void Incluir(string nome, int idade, string whats) {

        Node novo = new Node(nome, idade, whats);

        if (this.primeiro == null) {

            this.primeiro = novo;

        }

        else

        {

            Node atual = this.primeiro;

            while (atual.prox != null) {

                atual = atual.prox;

            }

            atual.prox = novo;

        }

    }

    public void Alterar(string nome) {

        Node atual = this.primeiro;

        while (atual != null) {

            if (atual.nome == nome) {

                Console.WriteLine("Nome: " + atual.nome);

                Console.WriteLine("Idade: " + atual.idade);

                Console.WriteLine("WhatsApp: " + atual.whats);

                Console.WriteLine("Informe os novos dados:");

                Console.Write("Idade: ");

                int idade = int.Parse(Console.ReadLine());

                Console.Write("WhatsApp: ");

                string whats = Console.ReadLine();

                atual.idade = idade;

                atual.whats = whats;

                Console.WriteLine("Dados alterados com sucesso!");

                return;

            }

            atual = atual.prox;

        }

        Console.WriteLine("Registro não encontrado!");

    }

    public void Excluir(string nome) {

        if (this.primeiro == null) {

            Console.WriteLine("Lista vazia!");

            return;

        }

        if (this.primeiro.nome == nome) {

            this.primeiro = this.primeiro.prox;

            Console.WriteLine("Registro excluído com sucesso!");

            return;

        }

        Node atual = this.primeiro;

        while (atual.prox != null) {

            if (atual.prox.nome == nome) {

                atual.prox = atual.prox.prox;

                Console.WriteLine("Registro excluído com sucesso!");

                return;

            }

            atual = atual.prox;

        }

        Console.WriteLine("Registro não encontrado!");

    }

    public void Exibir() {

        if (this.primeiro == null) {

            Console.WriteLine("Lista vazia!");

            return;

        }

        Node atual = this.primeiro;

        while (atual != null)

        {

            Console.WriteLine("Nome: " + atual.nome);

            Console.WriteLine("Idade: " + atual.idade);

            Console.WriteLine("WhatsApp: " + atual.whats);

            Console.WriteLine();

            atual = atual.prox;

        }

    }

}

class Exclusao\_alteracao\_e\_exibicao {

    static void Main(string[] args)     {

        Lista lista = new Lista();

        while (true) {

            Console.WriteLine("Escolha uma opção:");

            Console.WriteLine("1 - Incluir");

            Console.WriteLine("2 - Alterar");

            Console.WriteLine("3 - Excluir");

            Console.WriteLine("4 - Exibir");

            Console.WriteLine("0 - Sair");

            int opcao = int.Parse(Console.ReadLine());

            switch (opcao) {

                case 0:

                    return;

                case 1:

                    Console.Write("Nome: ");

                    string nome = Console.ReadLine();

                    Console.Write("Idade: ");

                    int idade = int.Parse(Console.ReadLine());

                    Console.Write("WhatsApp: ");

                    string whats = Console.ReadLine();

                    lista.Incluir(nome, idade, whats);

                    Console.WriteLine("Registro incluído com sucesso!");

                    break;

                case 2:

                    Console.Write("Informe o nome do registro a ser alterado: ");

                    nome = Console.ReadLine();

                    lista.Alterar(nome);

                    break;

                case 3:

                    Console.Write("Informe o nome do registro a ser excluído: ");

                    nome = Console.ReadLine();

                    lista.Excluir(nome);

                    break;

                case 4:

                    lista.Exibir();

                    break;

                default:

                    Console.WriteLine("Opção inválida!");

                    break;

            }

            Console.WriteLine();

        }

    }

}

21) Faça um programa para o usuário digitar vários números, inclua-os num vetor. Em seguida, faça uma função recursiva que copie os números do vetor para uma lista encadeada. Após copiar os números do vetor para a lista, percorra a lista e encontre o menor, o maior e a média dos números.

using System;

class Node {

    public int value;

    public Node next;

    public Node(int value) {

        this.value = value;

        this.next = null;

    }

}

class Lista\_encadeada\_recursiva {

    static void Main(string[] args) {

        Console.WriteLine("Digite vários números separados por espaço:");

        string input = Console.ReadLine();

        string[] numbers = input.Split(' ');

        Node head = null;

        int count = 0;

        foreach (string number in numbers) {

            int value = int.Parse(number);

            Node newNode = new Node(value);

            if (head == null) {

                head = newNode;

            } else {

                Node current = head;

                while (current.next != null) {

                    current = current.next;

                }

                current.next = newNode;

            }

            count++;

        }

        int min = head.value;

        int max = head.value;

        int sum = head.value;

        Node current2 = head.next;

        while (current2 != null) {

            if (current2.value < min) {

                min = current2.value;

            }

            if (current2.value > max) {

                max = current2.value;

            }

            sum += current2.value;

            current2 = current2.next;

        }

        double average = (double)sum / count;

        Console.WriteLine("Menor número: " + min);

        Console.WriteLine("Maior número: " + max);

        Console.WriteLine("Média dos números: " + average);

    }

}

22) Implemente a lógica da inserção e remoção em uma lista duplamente encadeada (nó com ponteiro para o nó anterior e o próximo nó). Faça a procura pelo nó a ser excluído.

using System;

class Node {

    public int data;

    public Node next;

    public Node prev;

    public Node(int data) {

        this.data = data;

        this.next = null;

        this.prev = null;

    }

}

class Lista\_duplamente\_encadeada {

    static void Main(string[] args) {

        Node head = null;

        Node newNode = new Node(1);

        if (head == null) {

            head = newNode;

        } else {

            Node current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newNode;

            newNode.prev = current;

        }

        newNode = new Node(2);

        if (head == null) {

            head = newNode;

        } else {

            Node current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newNode;

            newNode.prev = current;

        }

        newNode = new Node(3);

        if (head == null) {

            head = newNode;

        } else {

            Node current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newNode;

            newNode.prev = current;

        }

        newNode = new Node(4);

        if (head == null) {

            head = newNode;

        } else {

            Node current = head;

            while (current.next != null) {

                current = current.next;

            }

            current.next = newNode;

            newNode.prev = current;

        }

        Console.Write("Lista: ");

        Node temp = head;

        while (temp != null) {

            Console.Write(temp.data + " ");

            temp = temp.next;

        }

        Console.WriteLine();

        Node nodeToFind = head;

        while (nodeToFind != null && nodeToFind.data != 3) {

            nodeToFind = nodeToFind.next;

        }

        if (nodeToFind != null) {

            Console.WriteLine("Nó com numero 3 encontrado na lista: ");

        } else {

            Console.WriteLine("Nó com numero 3 não encontrado na lista");

        }

        // Remoção

        Node nodeToRemove = head;

        while (nodeToRemove != null && nodeToRemove.data != 3) {

            nodeToRemove = nodeToRemove.next;

        }

        if (nodeToRemove != null) {

            if (nodeToRemove.prev != null) {

                nodeToRemove.prev.next = nodeToRemove.next;

            } else {

                head = nodeToRemove.next;

            }

            if (nodeToRemove.next != null) {

                nodeToRemove.next.prev = nodeToRemove.prev;

            }

            Console.Write("Lista após remover o nó com numero 3: ");

            temp = head;

            while (temp != null) {

                Console.Write(temp.data + " ");

                temp = temp.next;

            }

            Console.WriteLine();

        } else {

            Console.WriteLine("Nó com numero 3 não encontrado na lista");

        }

    }

}

23) Implemente a lógica de uma lista encadeada com o conceito de fila, ou seja, insira os nós no fim da lista e remova-os do início da lista. Utilize duas variáveis (inicio e fim) para controlar os acessos a lista.

using System;

public class NoFila {

    public int valor;

    public NoFila proximo;

    public NoFila(int valor) {

        this.valor = valor;

        this.proximo = null;

    }

}

public class Fila {

    private NoFila inicio;

    private NoFila fim;

    public Fila() {

        this.inicio = null;

        this.fim = null;

    }

    public void Enfileirar(int valor) {

        NoFila novoNo = new NoFila(valor);

        if (this.fim != null) {

            this.fim.proximo = novoNo;

        } else {

            this.inicio = novoNo;

        }

        this.fim = novoNo;

    }

    public int Desenfileirar() {

        if (this.inicio == null) {

            throw new InvalidOperationException("A fila está vazia.");

        }

        int valor = this.inicio.valor;

        this.inicio = this.inicio.proximo;

        if (this.inicio == null) {

            this.fim = null;

        }

        return valor;

    }

    public void Imprimir() {

        NoFila noAtual = this.inicio;

        while (noAtual != null) {

            Console.Write(noAtual.valor + " ");

            noAtual = noAtual.proximo;

        }

        Console.WriteLine();

    }

}

class Lista\_encadeada\_com\_o\_conceito\_de\_fila {

    static void Main(string[] args) {

        Fila fila = new Fila();

        fila.Enfileirar(10);

        fila.Enfileirar(20);

        fila.Enfileirar(30);

        fila.Enfileirar(40);

        fila.Imprimir();

        int valorRemovido = fila.Desenfileirar();

        Console.WriteLine(valorRemovido);

        fila.Imprimir();

    }

}

24) O código fonte a seguir, contém a implementação de uma pilha utilizando a estratégia encadeada.

|  |  |  |
| --- | --- | --- |
| No topo; // Var global  int top() {  if (topo == null)  return -1;  return topo.dado;  } | int pop() {  if (topo == null)  return -1;  No lixo = topo;  topo = topo.prox;  lixo.prox = null;  return lixo.dado;  } | class No {  int dado;  No prox;  } |

Com base no exposto, assinale a opção que possui a implementação correta da função push para essa pilha.

|  |  |
| --- | --- |
| A)  void push (int elemento) {  No novo = new No();  novo.dado = elemento;  if (topo == null)  topo = novo;  else  topo.prox = novo;  } | B)  void push (int elemento) {  No novo = new No();  novo.dado = elemento;  topo = novo;  } |
| C)  void push (int elemento) {  topo.dado = elemento;  } | D)  void push (int elemento) {  No novo = new No();  novo.dado = elemento;  while (topo != null)  topo = topo.prox;  topo.prox = novo;  } |
| X E)  void push (int elemento) {  No novo = new No();  novo.dado = elemento;  novo.prox = topo;  topo = novo;  } |  |

**- Lista não linear: Árvore Binária**

25) Seja uma árvore binária inicialmente vazia, ilustre seu estado final após terem sido inseridos os elementos a seguir, nesta ordem:

a) Insere(ref raiz, 4);

b) Insere(ref raiz, 1);

c) Insere(ref raiz, 0);

d) Insere(ref raiz, 5);

e) Insere(ref raiz, 3);

f) Insere(ref raiz, 7);

g) Insere(ref raiz, 2);

h) Insere(ref raiz, 6);

i) Insere(ref raiz, 9);

j) Insere(ref raiz, 8);

Agora, redesenhe a árvore original para cada situação:

a) Remove(ref raiz, 8)

b) Remove(ref raiz, 9)

c) Remove(ref raiz, 4)

26) Considerando a árvore esquematizada a seguir, responda:

1. Quais são os nós folhas
2. O nível de cada nó não terminal
3. A altura da árvore
4. O grau da árvore
5. Os descendentes do nó c

27) Desenvolva um programa para inserir, pesquisar, remover e exibir os valores de uma árvore binária. Observe as opções a seguir:

1. Inserir um valor digitado pelo usuário
2. Pesquisar um valor digitado pelo usuário. Exiba uma mensagem informando se encontrou ou não
3. Remover um valor digitado pelo usuário. Exiba a mensagem se removido com sucesso ou não encontrado
4. Exibir todos os valores da árvore em ordem, pré ordem ou pós ordem

using System;

namespace BinaryTree {

    class Node {

        public int data;

        public Node left;

        public Node right;

    }

    class BinaryTree {

        private Node root;

        public BinaryTree() {

            root = null;

        }

        public void Insert(int data) {

            Node newNode = new Node();

            newNode.data = data;

            if (root == null) {

                root = newNode;

            } else {

                Node current = root;

                Node parent;

                while (true) {

                    parent = current;

                    if (data < current.data) {

                        current = current.left;

                        if (current == null) {

                            parent.left = newNode;

                            break;

                        }

                    } else {

                        current = current.right;

                        if (current == null) {

                            parent.right = newNode;

                            break;

                        }

                    }

                }

            }

        }

    }

        public bool Search(int data) {

            Node current = root;

            while (current != null) {

                if (data == current.data) {

                    return true;

                } else if (data < current.data) {

                    current = current.left;

                } else {

                    current = current.right;

                }

            }

            return false;

        }

        public bool Remove(int data) {

            Node current = root;

            Node parent = root;

            bool isLeftChild = true;

            while (current.data != data) {

                parent = current;

                if (data < current.data) {

                    isLeftChild = true;

                    current = current.left;

                } else {

                    isLeftChild = false;

                    current = current.right;

                }

                if (current == null) {

                    return false;

                }

            }

            if (current.left == null && current.right == null) {

                if (current == root) {

                    root = null;

                } else if (isLeftChild) {

                    parent.left = null;

                } else {

                    parent.right = null;

                }

            } else if (current.right == null) {

                if (current == root) {

                    root = current.left;

                } else if (isLeftChild) {

                    parent.left = current.left;

                } else {

                    parent.right = current.left;

                }

            } else if (current.left == null) {

                if (current == root) {

                    root = current.right;

                } else if (isLeftChild) {

                    parent.left = current.right;

                } else {

                    parent.right = current.right;

                }

            } else {

                Node successor = GetSuccessor(current);

                if (current == root) {

                    root = successor;

                } else if (isLeftChild) {

                    parent.left = successor;

                } else {

                    parent.right = successor;

                }

                successor.left = current.left;

            }

            return true;

        }

        private Node GetSuccessor(Node node) {

            Node successorParent = node;

            Node successor = node;

            Node current = node.right;

            while (current != null) {

                successorParent = successor;

                successor = current;

                current = current.left;

            }

            if (successor != node.right) {

                successorParent.left = successor.right;

                successor.right = node;

            }

            return successor;

    }

    public void InOrderTraversal(Node node) {

        if (node != null) {

            InOrderTraversal(node.left);

            Console.Write(node.data + " ");

            InOrderTraversal(node.right);

        }

    }

    public void PreOrderTraversal(Node node) {

        if (node != null) {

            Console.Write(node.data + " ");

            PreOrderTraversal(node.left);

            PreOrderTraversal(node.right);

        }

    }

    public void PostOrderTraversal(Node node) {

        if (node != null) {

            PostOrderTraversal(node.left);

            PostOrderTraversal(node.right);

            Console.Write(node.data + " ");

        }

    }

}

class Program {

    static void Main(string[] args) {

        BinaryTree tree = new BinaryTree();

        while (true) {

            Console.WriteLine("\n1. Inserir um valor");

            Console.WriteLine("2. Pesquisar um valor");

            Console.WriteLine("3. Remover um valor");

            Console.WriteLine("4. Exibir todos os valores em ordem");

            Console.WriteLine("5. Exibir todos os valores em pré-ordem");

            Console.WriteLine("6. Exibir todos os valores em pós-ordem");

            Console.WriteLine("7. Sair\n");

            Console.Write("Digite uma opção: ");

            int option = int.Parse(Console.ReadLine());

            switch (option) {

                case 1:

                    Console.Write("\nDigite um valor para inserir: ");

                    int value = int.Parse(Console.ReadLine());

                    tree.Insert(value);

                    Console.WriteLine("\nValor inserido com sucesso!");

                    break;

                case 2:

                    Console.Write("\nDigite um valor para pesquisar: ");

                    int searchValue = int.Parse(Console.ReadLine());

                    if (tree.Search(searchValue)) {

                        Console.WriteLine("\nValor encontrado!");

                    } else {

                        Console.WriteLine("\nValor não encontrado!");

                    } break;

                case 3:

                    Console.Write("\nDigite um valor para remover: ");

                    int removeValue = int.Parse(Console.ReadLine());

                    if (tree.Remove(removeValue)) {

                        Console.WriteLine("\nValor removido com sucesso!");

                    } else {

                        Console.WriteLine("\nValor não encontrado!");

                    } break;

                case 4:

                    Console.Write("\nValores em ordem: ");

                    tree.InOrderTraversal(tree.root);

                    Console.WriteLine();

                    break;

                case 5:

                    Console.Write("\nValores em pré-ordem: ");

                    tree.PreOrderTraversal(tree.root);

                    Console.WriteLine();

                    break;

                case 6:

                    Console.Write("\nValores em pós-ordem: ");

                    tree.PostOrderTraversal(tree.root);

                    Console.WriteLine();

                    break;

                case 7:

                    Console.WriteLine("\nSaindo...");

                    Environment.Exit(0);

                    break;

                default:

                    Console.WriteLine("\nOpção inválida!");

                    break;

            }

        }

    }

}

28) Baseado no atravessamento “em ordem” desenvolva:

a) um algoritmo para percorrer os nós e exibir apenas os valores armazenados nas folhas

b) uma rotina que determina se uma árvore é estritamente binária. OBS: Dizemos que uma árvore é estritamente binária se todo nó pertencente a ela é folha ou tem os dois filhos

c) um algoritmo para contar o número de nós não-terminais

29) O uso da estrutura de dados tipo Árvore Binária de Busca é uma técnica fundamental de programação. Uma árvore binária é um conjunto finito de elementos que está vazio ou é particionado em três subconjuntos, a saber: 1) raiz da árvore - elemento inicial (único), 2) subárvore da esquerda - se vista isoladamente compõe outra árvore e 3) subárvore da direita - se vista isoladamente compõe outra árvore. A árvore pode não ter qualquer elemento (árvore vazia). A definição de árvore é recursiva e, devido a isso, muitas operações sobre árvores binárias utilizam recursão. Sendo “A” a raiz de uma árvore binária e “B” a raiz de sua subárvore esquerda ou direita, é dito que “A” é pai de “B” e que “B” é filho de “A”. Um elemento sem filhos é chamado de folha. A altura da árvore é o número de elementos encontrados no caminho descendente mais longo que liga a sua raiz até uma folha.

Uma Árvore de Busca Binária é uma árvore binária especializada, na qual a informação que o elemento filho esquerdo possui é numericamente menor que a informação do elemento pai. De forma análoga, a informação que o elemento filho direito possui é numericamente maior ou igual à informação do elemento pai. O objetivo de organizar dados em Árvores Binárias de Busca é facilitar a tarefa de encontrar um determinado elemento. O percurso completo de uma árvore binária consiste em visitar todos os elementos desta árvore, segundo algum critério, a fim de processá-los. Três formas são bem conhecidas para a realização deste percurso: 1) pré-ordem, 2) em-ordem e 3) pós-ordem. A figura a seguir mostra um exemplo de árvore binária.



Considerando o texto e a figura apresentados e que a seguinte lista de elementos numéricos: 27, 34, 40, 18, 23, 5, 25, 36, 10, 7, -2 seja totalmente transferida para uma estrutura de Árvore Binária de Busca, inicialmente vazia, elemento a elemento, da esquerda para a direita, assinale a alternativa correta.

A) A árvore resultante terá 5 níveis de altura, com 6 elementos à esquerda da raiz principal (inicial) e 4 elementos à direita.

B) O percurso da árvore em Pré-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): -2, 7, 10, 5, 25, 23, 18, 36, 40, 34, 27.

C) O percurso da árvore em Em-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): -2, 5, 7, 10, 18, 23, 25, 27, 34, 36, 40.

D) O percurso da árvore em Pós-ordem irá processar os elementos na seguinte ordem (do primeiro ao último): 27, 18, 5, -2, 10, 7, 23, 25, 34, 40, 36.

E) O número máximo de elementos que essa árvore poderá ter com 10 níveis será de 1024 elementos.

**- Hash**

30) Ilustre a situação (memória) do vetor hash, inicialmente vazio, após a execução de cada uma das operações a seguir. Considere um vetor para cada sequência de operações.

1. Insere(vetor, 22);
2. Insere(vetor, 40);
3. Insere(vetor, 34);
4. Insere(vetor, 44);
5. InsereLinear(vetor, 22);
6. InsereLinear (vetor, 40);
7. InsereLinear (vetor, 34);
8. InsereLinear (vetor, 44);
9. InsereEncadeado(vetor, 22);
10. InsereEncadeado (vetor, 40);
11. InsereEncadeado (vetor, 34);
12. InsereEncadeado (vetor, 44);

31) Implemente um programa que conte a quantidade de colisões ocorridas durante o processo de inserção. Utilize o tratamento de colisão linear. O vetor deve ser de um tipo abstrato de dado composto por nota, nome e email. A nota deve ser um número inteiro e corresponderá a chave.

O menu deve conter as seguintes opções: Inserir, Recuperar e Informar. Na opção recuperar, solicite a nota e caso encontre-a no vetor, exiba o nome e o email correspondente, caso contrário, exiba a mensagem de nota não encontrada. A opção informar, informa a quantidade de colisões ocorridas até o momento.

32) Implemente um programa com as seguintes opções: Sem tratamento de colisão, Tratamento de colisão Linear e Tratamento de colisão com Lista Encadeada.

Dentro de cada opção deve haver as funcionalidades: Inserir, Alterar e Relatar.

O vetor deve ser do tipo abstrato de dado composto por idade, nome e whats. Serão necessários 3 vetores, um para cada tipo de tratamento de colisão.

Para inserir um novo registro, solicite a idade, nome e whats. Utilize a idade como chave.

Para alterar solicite a idade (chave) para ser utilizada na busca. Caso encontrada, informe o nome e o whats da pessoa. Após a consulta, o usuário pode atualizar somente o nome e o whats.

Para relatar, percorra o vetor do inicio ao fim e exiba todos os registros.

33) Uma abordagem calcula a posição da chave na tabela com base no valor da chave. Este valor é a única indicação da posição. Quando a chave é conhecida, a posição na tabela pode ser acessada diretamente, sem fazer qualquer outro teste preliminar, conforme exigido na busca binária ou durante a pesquisa em uma árvore. Precisamos encontrar uma função h que possa transformar uma chave particular K — seja ela uma cadeia de caracteres, um número, um registro etc. — em um índice na tabela usada para armazenar itens do mesmo tipo que K. A função h é chamada função de escrutínio (hash) e não retorna valores únicos. Por exemplo, h(“abc”) = h(“acb”). Este problema é chamado colisão.

Uma tabela na qual os dados são inseridos em entradas determinadas por uma função h, conforme descrito no texto, é chamada de tabela de dispersão ou tabela de espalhamento. A tabela de dispersão é representada por meio do vetor tabelaH de T posições, no qual cada posição é uma estrutura ItemDado e o procedimento de inserção em tabelaH está definido a seguir.

ItemDado:

chave: inteiro

dado: inteiro

prox: ItemDado

procedimento insere(chave, dado: inteiro)

início

item, temp: ItemDado

índice: inteiro

índice <- h(chave)

se tabelaH[índice].chave = -1 então

tabelaH[índice].chave <- chave

tabelaH[índice].dado <- dado

tabelaH[índice].prox <- NULO

senão

item.chave <- chave

item.dado <- dado

item.prox <- NULO

se tabelaH[índice].prox = NULO então

tabelaH[índice].prox <- item

senão

temp <- tabelaH[índice].prox

enquanto temp.prox ≠ NULO faça

temp <- temp.prox

fimenquanto

temp.prox <- item

fimse

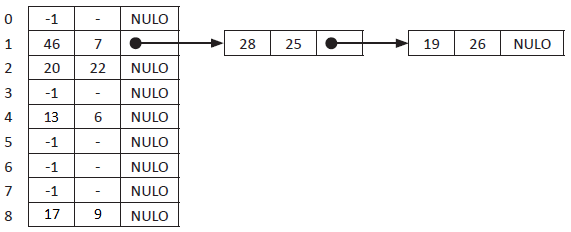
fimse

fimprocedimento

O valor -1 no campo chave de uma entrada na tabela significa que aquela entrada está livre; o campo prox com o valor NULO significa que ele não aponta para nenhum item de dado. A função h(chave) simplesmente retorna o valor do resto da divisão inteira de chave por T.

Considerando as informações apresentadas e supondo que T = 9, assinale a opção que corresponde ao estado da tabelaH após a seguinte sequência de chamadas: insere(13, 6); insere(46, 7); insere(20, 22); insere(28, 25); insere(19, 26); insere(17, 9).

A)



B)



C)



D)



E)

