PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

INF1010

Estruturas de Dados Avançadas

LAB2 - Vetores e Listas

Integrantes:

2320276 Aline Jéssica David Gonçalves

2320869 Lucas Raposo Bastos Araujo

Professor(a):

Luiz Fernando Bessa Seibel

Rio de Janeiro

Março de 2025

1. SAÍDA DO PROGRAMA

```
Retirando 10 da lista de pares
20 -> 30 -> 34 -> 36 -> 38 -> 50 -> 60 -> 62 -> 70 -> NULL
Retirando 11 da lista de ímpares
13 -> 33 -> 35 -> 41 -> 43 -> 55 -> 61 -> 71 -> NULL
Retirando 20 da lista de pares
30 -> 34 -> 36 -> 38 -> 50 -> 60 -> 62 -> 70 -> NULL
Retirando 13 da lista de ímpares
33 -> 35 -> 41 -> 43 -> 55 -> 61 -> 71 -> NULL
Retirando 30 da lista de pares
34 -> 36 -> 38 -> 50 -> 60 -> 62 -> 70 -> NULL
Retirando 33 da lista de ímpares
35 -> 41 -> 43 -> 55 -> 61 -> 71 -> NULL
Retirando 34 da lista de pares
36 -> 38 -> 50 -> 60 -> 62 -> 70 -> NULL
Retirando 35 da lista de ímpares
41 -> 43 -> 55 -> 61 -> 71 -> NULL
Retirando 36 da lista de pares
38 -> 50 -> 60 -> 62 -> 70 -> NULL
Retirando 41 da lista de ímpares
43 -> 55 -> 61 -> 71 -> NULL
Retirando 38 da lista de pares
50 -> 60 -> 62 -> 70 -> NULL
Retirando 43 da lista de ímpares
55 -> 61 -> 71 -> NULL
Retirando 50 da lista de pares
60 -> 62 -> 70 -> NULL
Retirando 55 da lista de ímpares
61 -> 71 -> NULL
Retirando 60 da lista de pares
62 -> 70 -> NULL
Retirando 61 da lista de ímpares
71 -> NULL
Retirando 62 da lista de pares
70 -> NULL
Retirando 71 da lista de ímpares
NULL
Retirando 70 da lista de pares
NULL
Lista intercalada (encadeada):
9 -> 10 -> 11 -> 20 -> 13 -> 30 -> 33 -> 34 -> 35 -> 36 -> 41 -> 38 -> 43 -> 50 -> 55 -> 60 -> 61 -> 62 -> 71 -> 70 -> NULL
```

```
Impressao a partir de vetores:
9 (retirado)
10 (retirado)
11 (retirado)
20 (retirado)
13 (retirado)
30 (retirado)
33 (retirado)
34 (retirado)
35 (retirado)
36 (retirado)
41 (retirado)
38 (retirado)
43 (retirado)
50 (retirado)
55 (retirado)
60 (retirado)
61 (retirado)
62 (retirado)
71 (retirado)
70 (retirado)
Impressao de verificacao de numeros retirados dos vetores:
par 1: -1
impar 1: -1
par 2: -1
impar 2: -1
par 3: -1
impar 3: -1
par 4: -1
impar 4: -1
par 5: -1
impar 5: -1
par 6: -1
impar 6: -1
par 7: -1
impar 7: -1
par 8: -1
impar 8: -1
par 9: -1
impar 9: -1
par 10: -1
impar 10: -1


...Program finished with exit code 0
Press ENTER to exit console.
```

2. OBJETIVO

O objetivo do programa é imprimir os valores pares e ímpares sendo alternados em vetores e a partir de uma lista encadeada ordenada com todos os valores juntos (pares e ímpares). A impressão dos valores deve ser de forma recursiva em ambas as funções de impressão.

3. FUNÇÕES

Bibliotecas utilizadas e definição de constante:

```
#include <stdio.h>
#include <stdlib.h>
#define X 10
```

Declaração de struct:

```
typedef struct node Node;
struct node {
    int num;
    Node* prox;
};
```

A linha typedef struct node Node; cria um atalho, permitindo usar o nome Node em vez de escrever struct node toda vez. A estrutura struct node define um tipo de "nó", que tem dois componentes: um número (num) e um ponteiro (prox) que aponta para o próximo nó na lista. Isso permite criar uma lista de nós, onde cada nó guarda um número e aponta para o próximo nó da lista.

Função para imprimir os vetores intercalados:

```
int imprimeVetoresIntercalados(int * par, int * impar, int indice) {
        if (indice == X) return 0;

        int retirado = -1;

        printf("%d ", impar[indice]);
        impar[indice] = retirado;
```

```
        printf("(retirado)\n");

        printf("%d ", par[indice]);
        par[indice] = retirado;
        printf("(retirado)\n");

        return imprimeVetoresIntercalados(par, impar, indice + 1);
}
```

A função imprimeVetoresIntercalados serve para imprimir os elementos de dois vetores (par e impar) de forma intercalada, ou seja, um elemento do vetor impar seguido de um elemento do vetor par, até o índice atingir um valor máximo (X). A função também "retira" os elementos de ambos os vetores, substituindo-os por -1 para indicar que foram processados. Ela é recursiva, ou seja, ela chama a si mesma para processar o próximo índice até que o índice alcance X, que é a condição de parada. O processo de remoção é simulado pela atribuição de -1 aos elementos dos vetores, e a cada iteração a função imprime o número retirado, seguido de um aviso "(retirado)".

Função para imprimir os vetores vazios:

```
int imprimeVetoresVazios(int * par, int * impar, int indice) {
    if (indice == X) return 0;

        printf("par %d: %d\n", (indice + 1), par[indice]);
    printf("impar %d: %d\n", (indice + 1), impar[indice]);

        return imprimeVetoresVazios(par, impar, indice + 1);
}
```

A função imprimeVetoresVazios tem o objetivo de imprimir os elementos dos vetores par e impar em cada índice, até que o índice atinja o valor X. Ela funciona de forma recursiva, chamando a si mesma para processar o próximo índice até atingir a condição de parada (indice == X). Para cada índice, ela imprime o valor de

par[indice] e impar[indice], junto com o número do índice (iniciado em 1, pois o valor impresso é indice + 1). Ou seja, ela mostra, para cada índice, os números que estão nas posições correspondentes dos vetores par e impar, sem modificar seus valores.

Função para imprimir uma lista encadeada:

```
void imprimeListaEncadeada(Node* n) {
    if (n == NULL) {
        printf("NULL\n");
        return;
    }
    printf("%d -> ", n->num);
    imprimeListaEncadeada(n->prox);
}
```

A função imprimeListaEncadeada tem o objetivo de imprimir os elementos de uma lista encadeada. Ela recebe um ponteiro n, que aponta para o primeiro nó da lista. Se o ponteiro for NULL, significa que a lista está vazia ou atingiu o final, então a função imprime "NULL" e termina. Caso contrário, ela imprime o valor do campo num do nó atual, seguido de ->, indicando que há um próximo nó. Depois, a função chama a si mesma (recursivamente) para imprimir o próximo nó da lista, que é apontado pelo ponteiro prox. Isso continua até que todos os nós da lista sejam impressos, terminando com a impressão de "NULL" quando o final da lista for alcançado.

Função para criar um elemento no formato da struct:

```
Node* CriaStruct(int n, Node* ant) {
    Node* n1 = (Node*)malloc(sizeof(Node));
    if (n1 == NULL) {
        printf("Erro na alocacao de memoria.\n");
        return NULL;
    }
    n1->num = n;
    n1->prox = ant;
```

```
    return n1;
}
```

A função CriaStruct cria um novo nó para uma lista encadeada. Ela recebe um número (n) e um ponteiro para o nó anterior (ant). Primeiro, tenta alocar memória para o novo nó. Se a alocação falhar, retorna NULL. Caso contrário, o número é armazenado no nó e o ponteiro prox é configurado para apontar para o nó anterior. No final, a função retorna o ponteiro para o novo nó criado.

Função para criar uma lista encadeada:

```
Node* CriaLista(int v[], int n) {
    Node* lista = NULL;
    for (int i = n - 1; i >= 0; i--) {
        lista = CriaStruct(v[i], lista);
    }
    return lista;
}
```

A função CriaLista cria uma lista encadeada a partir de um vetor de números. Ela recebe um vetor v e o tamanho n do vetor. A função começa do último elemento do vetor e, para cada elemento, chama a função CriaStruct para criar um nó e adicioná-lo à lista. O novo nó é inserido no início da lista, fazendo com que a lista seja construída de trás para frente. Ao final, a função retorna o ponteiro para o primeiro nó da lista encadeada criada.

Função para intercalar as listas encadeadas par e impar:

```
Node* intercalaListas(Node* par, Node* impar) {
    Node temp;
    Node* atual = &temp;
    temp.prox = NULL;
    Node* aux;

    while (par && impar) {
```

```
        printf("Retirando %d da lista de ímpares\n", impar->num);
        atual->prox = impar;
        impar = impar->prox;
        atual = atual->prox;
        imprimeListaEncadeada(impar);
        printf("Retirando %d da lista de pares\n", par->num);
        atual->prox = par;
        par = par->prox;
        atual = atual->prox;
        imprimeListaEncadeada(par);
    }
    if (impar) {
        atual->prox = impar;
    }
    if (par) {
        atual->prox = par;
    }
    return temp.prox;
}
```

A função intercalaListas intercala duas listas encadeadas de números (pares e ímpares), criando uma nova lista ordenada. Ela recebe como entrada dois ponteiros, par e impar, que apontam para as listas de números pares e ímpares, respectivamente. A função usa uma lista temporária, representada pelo ponteiro temp, para adicionar os elementos de impar e par alternadamente. Enquanto ambas as listas tiverem elementos, ela retira um nó de cada lista, adicionando-o à nova lista encadeada, e imprime o estado das listas após cada remoção. Se uma das listas terminar antes da outra, os elementos restantes da outra lista são adicionados diretamente à lista resultante. No final, a função retorna o ponteiro para a lista intercalada.

Declaração de vetores par e impar:

```
int par[X] = { 10, 20, 30, 34, 36, 38, 50, 60, 62, 70 };
int impar[X] = { 9, 11, 13, 33, 35, 41, 43, 55, 61, 71 };
```

As variáveis par e impar são vetores de inteiros, onde o vetor par contém 10 números pares e o vetor impar contém 10 números ímpares. Ambos os vetores têm o mesmo tamanho (X = 10), e os números estão organizados de forma crescente dentro de cada vetor. O vetor par começa com o número 10 e vai até o 70, enquanto o vetor impar começa com o número 9 e vai até o 71. Essas variáveis são usadas como entrada para funções que operam com esses valores, como a intercalação entre as listas de pares e ímpares.

Main:

```
int main(void) {
    Node* listaPar = CriaLista(par, X);
    Node* listaImpar = CriaLista(impar, X);

    printf("Intercalando listas encadeadas: \n");
    Node* listaIntercalada = intercalaListas(listaPar, listaImpar);

    printf("Lista intercalada (encadeada): \n");
    imprimeListaEncadeada(listaIntercalada);

    printf("\nImpressao a partir de vetores:\n");
    imprimeVetoresIntercalados(par, impar, 0);

    printf("Impressao de verificacao de numeros retirados dos vetores:\n");
    imprimeVetoresVazios(par, impar, 0);

    return 0;
}
```

O código principal (main) realiza a seguinte sequência de operações:

- **Criação das listas encadeadas**: Chama a função CriaLista para criar duas listas encadeadas a partir dos vetores par e impar. As listas resultantes são armazenadas nas variáveis listaPar e listaImpar.
- **Intercalação das listas**: A função intercalaListas é chamada para intercalar as listas listaPar e listaImpar. O resultado é armazenado na lista listaIntercalada.
- **Impressão da lista encadeada**: A função imprimeListaEncadeada é chamada para imprimir a lista intercalada gerada, mostrando os números intercalados da lista encadeada.
- **Impressão a partir de vetores**: A função imprimeVetoresIntercalados é chamada para imprimir os números dos vetores par e impar de forma intercalada, removendo os elementos à medida que são impressos.
- **Impressão de verificação**: A função imprimeVetoresVazios é chamada para imprimir os vetores par e impar, após os elementos terem sido "retirados" (substituídos por -1), confirmando que a remoção foi realizada corretamente.

No final, o programa mostra a intercalação dos números tanto a partir de listas encadeadas quanto de vetores, e verifica a remoção dos elementos dos vetores.

4. DESENVOLVIMENTO

Dificuldades encontradas:

Houve dificuldade para fazer as funções de lista encadeada, uma vez que é um pensamento mais elaborado.

Facilidades encontradas:

As funções de impressão recursivas não foi difícil.

Compilação:

O programa foi compilado e testado em um compilador online da linguagem C.

5. CONCLUSÃO

Em conclusão, foi necessário aplicar conhecimentos e conceitos de manipulação de vetores e listas encadeadas em C. Durante o desenvolvimento, foi preciso criar e intercalar listas encadeadas, além de trabalhar com funções recursivas para impressão e manipulação de dados. A utilização de funções recursivas, como a impressão dos vetores e listas, permite uma maior compreensão

das estruturas e do fluxo de execução do código. Ao final, foi alcançada a implementação e teste das funções com sucesso.

O código apresenta uma boa abordagem para manipulação de listas encadeadas e vetores, utilizando recursão e alocação dinâmica de memória, o que facilita a gestão de dados. Apesar disso, a recursão excessiva pode impactar negativamente a eficiência, especialmente em termos de consumo de memória e tempo de execução.