

Exercício Torre de Hanói

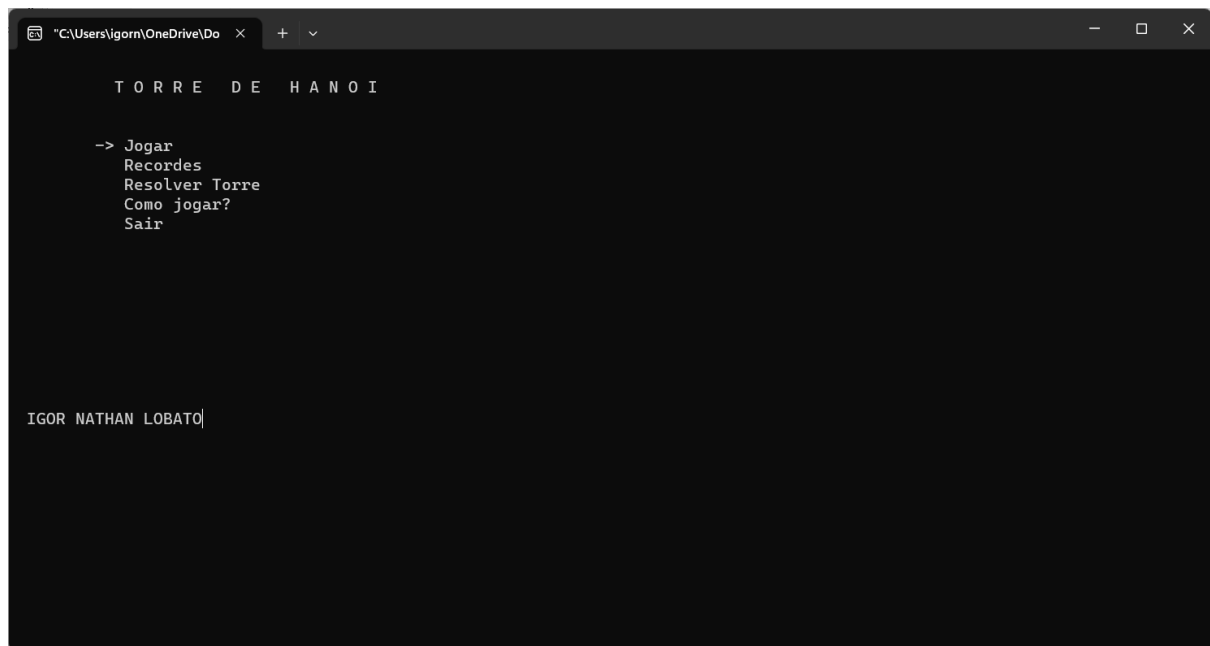
IGOR NATHAN LOBATO GRR20210549

VÍDEO DEMONSTRAÇÃO:

<https://www.youtube.com/watch?v=IHEkdQ0DB4g>

TELAS:

MENU PRINCIPAL:



```

TORRE DE HANOI

-> Jogar
    Recordes
    Resolver Torre
    Como jogar?
    Sair

IGOR NATHAN LOBATO|
```

JOGAR:

```

TORRE DE HANOI

Movimentos: 0

(-)
(--)
(---)
(----)
(-----)
Haste A      Haste B      Haste C

Escolha uma opcao:

1 - Mover disco do pino A para o pino B
2 - Mover disco do pino A para o pino C
3 - Mover disco do pino B para o pino A
4 - Mover disco do pino B para o pino C
5 - Mover disco do pino C para o pino A
6 - Mover disco do pino C para o pino B

9 - Voltar ao Menu
|
```

JOGO EM FUNCIONAMENTO:

```

TORRE DE HANOI

Movimentos: 2

(---)
(----)
(-----)      (-)      (--)
Haste A      Haste B      Haste C

Escolha uma opcao:

1 - Mover disco do pino A para o pino B
2 - Mover disco do pino A para o pino C
3 - Mover disco do pino B para o pino A
4 - Mover disco do pino B para o pino C
5 - Mover disco do pino C para o pino A
6 - Mover disco do pino C para o pino B

9 - Voltar ao Menu
|
```

EXIBIR RECORDES:

```
T O R R E   D E   H A N O I

- RECORDES -

1. igordemonstracao - Movimentos: 32, Tempo: -11910.79s

Pressione Enter para voltar!
```

RESOLVER TORRE DE HANOI ALTOMATICAMENTE:

```
(---)
(----)
(-----)
Haste A      Haste B      Haste C

|
```

TORRE RESOLVIDA! :

```
"C:\Users\igorn\OneDrive\Do x + v

Resolucao da Torre de Hanoi:

                (~)
                (--)
                (---)
                (----)
                (-----)
                (-----)
Haste A      Haste B      Haste C

Tecle Enter para retornar ao menu|
```

COMO JOGAR:

```
"C:\Users\igorn\OneDrive\Do x + v

      T O R R E   D E   H A N O I

INSTRUcoes:

- O objetivo e passar todos os discos de um pino (A) para
  o pino (C), usando o pino (B) como auxiliar!

REGRAS:

- Movimentar uma so peca (disco) de cada vez;
- Uma peca maior nao pode ficar acima de uma menor;
- Nao e permitido movimentar uma peca que esteja abaixo de outra;

Pressione Enter para voltar|
```

ENCERRAR O JOGO:

```
"C:\Users\igorn\OneDrive\Do x + v

TORRE DE HANOI

Jogar
Recordes
Resolver Torre
Como jogar?
-> Sair

IGOR NATHAN LOBATO
Process returned 0 (0x0) execution time : 202.079 s
Press any key to continue.
```

CÓDIGO FONTE DO JOGO:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <windows.h>
#define NUM_RECORDS 5

void posicao(int x, int y) //posiciona o cursor na tela
{
    COORD coord;
    coord.X = x;
    coord.Y = y;
    SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}

// Estrutura de um disco
typedef struct
{
    int tamanho;
} Disco;

// Estrutura de uma haste (pilha)
typedef struct
{
    Disco* discos[5]; // Capacidade máxima de 5 discos
    int topo;
} Haste;

// Função para inicializar uma haste vazia
void inicializarHaste(Haste* haste)
{
    haste->topo = -1;
}

// Função para empilhar um disco em uma haste
void empilharDisco(Haste* haste, Disco* disco)
{
    haste->discos[++haste->topo] = disco;
}

// Função para desempilhar um disco de uma haste
Disco* desempilharDisco(Haste* haste)
{
    return haste->discos[haste->topo--];
}
```

```

// Função para verificar se é válido mover um disco da origem para o destino
int movimentoValido(Haste* origem, Haste* destino)
{
    if (origem->topo == -1)
    {
        // Origem vazia
        return 0;
    }
    else if (destino->topo == -1)
    {
        // Destino vazio
        return 1;
    }
    else
    {
        Disco* discoOrigem = origem->discos[origem->topo];
        Disco* discoDestino = destino->discos[destino->topo];
        if (discoOrigem->tamanho < discoDestino->tamanho)
        {
            // Disco da origem é menor que o disco do destino
            return 1;
        }
        else
        {
            // Disco da origem é maior ou igual ao disco do destino
            return 0;
        }
    }
}

// Função para mover um disco de uma haste para outra
void moverDisco(Haste* origem, Haste* destino)
{
    if (movimentoValido(origem, destino))
    {
        Disco* disco = desempilharDisco(origem);
        empilharDisco(destino, disco);
    }
    else
    {
        printf("Movimento inválido!\n");
    }
}

// Função para exibir o estado atual das hastes
void exibirHastes(Haste* A, Haste* B, Haste* C)
{
    int maxAltura = A->topo;
    if (B->topo > maxAltura)
    {
        maxAltura = B->topo;
    }
    if (C->topo > maxAltura)
    {
        maxAltura = C->topo;
    }

    printf("\n");
    for (int i = maxAltura; i >= 0; i--)
    {
        // Haste A
        if (A->topo >= i)
        {
            int tamanho = A->discos[i]->tamanho;
            printf("(");
            for (int j = 0; j < tamanho; j++)
            {
                printf("-");
            }
            printf(")");
        }
        else
        {
            printf(" ");
        }
        printf("\t\t");

        // Haste B
        if (B->topo >= i)
        {
            int tamanho = B->discos[i]->tamanho;
            printf("(");
            for (int j = 0; j < tamanho; j++)
            {
                printf("-");
            }
        }
    }
}

```

```

        printf(" ");
    }
    else
    {
        printf(" ");
    }
    printf("\t\t");

    // Haste C
    if (C->topo >= i)
    {
        int tamanho = C->discos[i]->tamanho;
        printf("(");
        for (int j = 0; j < tamanho; j++)
        {
            printf("-");
        }
        printf(")");
    }
    else
    {
        printf(" ");
    }
    printf("\n");
}

printf("Haste A\t\tHaste B\t\tHaste C\n");
printf("\n");
}

// Função para verificar se o jogo foi concluído
int jogoConcluido(Haste* haste)
{
    return haste->topo == 4; // A haste "C" está completa com 5 discos
}

// Função para calcular o tempo de execução
double calcularTempo(clock_t inicio, clock_t fim)
{
    return ((double) (fim - inicio)) / CLOCKS_PER_SEC;
}

// Função para resolver a Torre de Hanoi
void resolverTorreHanoi(int n, Haste* origem, Haste* destino, Haste* auxiliar)
{
    if (n == 1)
    {
        moverDisco(origem, destino);
        exibirHastes(origem, auxiliar, destino);
        Sleep(1000); // Pausa de 1 segundo para a animação
        system("cls");
        return;
    }

    resolverTorreHanoi(n - 1, origem, auxiliar, destino);
    moverDisco(origem, destino);
    exibirHastes(origem, auxiliar, destino);
    Sleep(1000);
    system("cls"); // Pausa de 1 segundo para a animação
    resolverTorreHanoi(n - 1, auxiliar, destino, origem);
    exibirHastes(origem, auxiliar, destino);
    Sleep(1000);
    system("cls");
}

typedef struct
{
    char nomes[NUM_RECORDS][50];
    int movimentos[NUM_RECORDS];
    float tempos[NUM_RECORDS];
} Record;

// Função para inicializar os recordes
void inicializarRecordes(Record* recordes)
{
    for (int i = 0; i < NUM_RECORDS; i++)
    {
        recordes->tempos[i] = -1.0f;
    }
}

// Função para verificar se um tempo é um recorde
int verificaRecorde(Record* recordes, float tempo)
{

```

```

    for (int i = 0; i < NUM_RECORDS; i++)
    {
        if (tempo < recordes->tempos[i] || recordes->tempos[i] == -1.0f)
        {
            return 1;
        }
    }
    return 0;
}

// Função para adicionar um novo recorde
void adicionarRecorde(Recordes* recordes, float tempo, int movimentos, char nome[20])
{
    int posicaoInserir = -1;
    for (int i = 0; i < NUM_RECORDS; i++)
    {
        if (tempo < recordes->tempos[i] || recordes->tempos[i] == -1.0f)
        {
            posicaoInserir = i;
            break;
        }
    }
    if (posicaoInserir != -1)
    {
        for (int i = NUM_RECORDS - 1; i > posicaoInserir; i--)
        {
            recordes->tempos[i] = recordes->tempos[i - 1];
            recordes->movimentos[i] = recordes->movimentos[i - 1];
            strcpy(recordes->nomes[i], recordes->nomes[i - 1]);
        }
        recordes->tempos[posicaoInserir] = tempo;
        recordes->movimentos[posicaoInserir] = movimentos;
        strcpy(recordes->nomes[posicaoInserir], nome);
    }
}

// Função para exibir os recordes
void exibirRecordes(Recordes* recordes)
{
    // Ordenar os recordes por tempo usando o algoritmo de ordenação bubble sort
    int i, j;
    for (i = 0; i < NUM_RECORDS - 1; i++)
    {
        for (j = 0; j < NUM_RECORDS - i - 1; j++)
        {
            if (recordes->tempos[j] > recordes->tempos[j + 1])
            {
                // Trocar os tempos
                float tempTempo = recordes->tempos[j];
                recordes->tempos[j] = recordes->tempos[j + 1];
                recordes->tempos[j + 1] = tempTempo;

                // Trocar os movimentos
                int tempMovimentos = recordes->movimentos[j];
                recordes->movimentos[j] = recordes->movimentos[j + 1];
                recordes->movimentos[j + 1] = tempMovimentos;

                // Trocar os nomes
                char tempNome[50];
                strcpy(tempNome, recordes->nomes[j]);
                strcpy(recordes->nomes[j], recordes->nomes[j + 1]);
                strcpy(recordes->nomes[j + 1], tempNome);
            }
        }
    }

    printf("\n- RECORDES -\n\n");
    int numRecordesExibidos = 0;
    for (i = 0; i < NUM_RECORDS; i++)
    {
        if (recordes->tempos[i] != -1.0f)
        {
            printf("%d. %s - Movimentos: %d, Tempo: %.2fs\n", numRecordesExibidos + 1, recordes->nomes[i], recordes->movimentos[i], recordes->tempos[i]);
            numRecordesExibidos++;
            if (numRecordesExibidos == 5)
            {
                break;
            }
        }
    }
}

// Função para salvar os recordes em um arquivo
void salvarRecordes(Recordes* recordes)
{
    FILE* arquivo = fopen("recordes.txt", "w");

```



```

    if (arquivo == NULL)
    {
        printf("Erro ao abrir o arquivo de recordes.\n");
        return;
    }

    for (int i = 0; i < NUM_RECORDS; i++)
    {
        fprintf(arquivo, "%s %d %.2f\n", recordes->nomes[i], recordes->movimentos[i], recordes->tempos[i]);
    }

    fclose(arquivo);
}

// Função para carregar os recordes de um arquivo
void carregarRecordes(Recordes* recordes)
{
    FILE* arquivo = fopen("recordes.txt", "r");
    if (arquivo == NULL)
    {
        printf("Nenhum recorde encontrado.\n");
        return;
    }

    for (int i = 0; i < NUM_RECORDS; i++)
    {
        fscanf(arquivo, "%s %d %f\n", recordes->nomes[i], &recordes->movimentos[i], &recordes->tempos[i]);
    }

    fclose(arquivo);
}

void exibirMenu(Haste* A, Haste* B, Haste* C)
{
    char menus[5][1000] =
    {
        "Jogar",
        "Recordes",
        "Resolver Torre",
        "Como jogar?",
        "Sair"
    };

    int enter = 0;
    int count;
    int opcao = 0;
    int opcaoAnterior = -1;
    Recordes recordes;
    inicializarRecordes(&recordes);

    while(1)
    {
        if (opcao != opcaoAnterior)
        {
            system("cls");

            posicao(10,1);

            printf("%s", "T O R R E   D E   H A N O I");
            printf("\n\n\n");
            for (count=0; count<5; count++)
            {
                if(opcao==count)
                {
                    printf ("\t-> %s\n",menus[count]);
                }
                else
                {
                    printf ("\t   %s\n",menus[count]);
                }
            }
            opcaoAnterior = opcao;

            posicao(1,10);
            printf("IGOR NATHAN LOBATO");

        }
        Sleep(100);
        if(kbhit()) //kbhit verifica se alguma tecla foi pressionada
        {
            char tecla = getch();

            switch (tecla)

```

```

{
case 72:
    if(opcao!=0)
    {
        opcao --;
    }

    break;

case 80:
    if(opcao!=4)
    {
        opcao ++;
    }

    break;

case 13:

    enter = 1;

    switch(opcao)
    {
    case 0:
        system("cls");
        return;

        break;

    case 1:
        system("cls");
        posicao(10,1);

        printf("%s","T O R R E   D E   H A N O I");
        printf("\n\n\n");

        carregarRecordes(&recordes);
        exhibirRecordes(&recordes);

        posicao(10,18);

        printf("Pressione Enter para voltar!");
        getchar();

        system("cls");
        exhibirMenu(&A,&B,&C);
        return;

        break;

    case 2:
        system("cls");

        posicao(10,6);
        printf("Tecle enter para iniciar! ");
        getchar();
        system("cls");

        resolverTorreHanoi(5, A, C, B);

        posicao(1,4);
        printf("Resolucao da Torre de Hanoi:\n ");

        exhibirHastes(A, B,C);

        posicao(1,15);
        printf("\n\n Tecle Enter para retornar ao menu!");

        getchar();

        exhibirMenu(&A,&B,&C);

        return 0;

        break;

    case 3:
        system("cls");

```

```

        posicao(10,1);

        printf("%s", "T O R R E   D E   H A N O I");
        printf("\n\n\n");

        printf("INSTRUÇÕES:\n\n");
        printf("- O objetivo é passar todos os discos de um pino (A) para o pino (C), usando o pino (B) como auxiliar!\n");

        printf("\n\nREGRAS: \n\n");
        printf("- Movimentar uma só peça (disco) de cada vez;\n");
        printf("- Uma peça maior não pode ficar acima de uma menor;\n");
        printf("- Não é permitido movimentar uma peça que esteja abaixo de outra;\n");

        printf("\n\nPressione Enter para voltar!");
        getchar();

        system("cls");
        exibirMenu(&A,&B,&C);
        return 0;
        break;

    case 4:
        exit(0);
        break;

    }

    break;

    }
}

// Função principal
int main()
{
    Haste A, B, C;
    Disco discos[5];
    char nome[20];
    Recordes recordes;
    inicializarRecordes(&recordes);

    while(1)
    {

        // Inicialização das hastes e dos discos
        inicializarHaste(&A);
        inicializarHaste(&B);
        inicializarHaste(&C);

        for (int i = 4; i >= 0; i--)
        {
            discos[i].tamanho = i + 1;
            empilharDisco(&A, &discos[i]);
        }
        exibirMenu(&A,&B,&C);

        // Variáveis para controle dos melhores tempos
        clock_t tempos[5];
        for (int i = 0; i < 5; i++)
        {
            tempos[i] = -1;
        }

        int opcao;
        int movimentos = 0;
        clock_t inicio, fim;

        int voltar = 0;
        do
        {
            posicao(10,1);

            printf("%s", "T O R R E   D E   H A N O I");
            printf("\n\n\n");

            posicao(1,3);

```

```

printf("Movimentos: %d \n\n",movimentos);

exibirHastes(&A, &B, &C);
printf("\n\n Escolha uma opcao:\n\n");
printf(" 1 - Mover disco do pino A para o pino B\n");
printf(" 2 - Mover disco do pino A para o pino C\n");
printf(" 3 - Mover disco do pino B para o pino A\n");
printf(" 4 - Mover disco do pino B para o pino C\n");
printf(" 5 - Mover disco do pino C para o pino A\n");
printf(" 6 - Mover disco do pino C para o pino B\n\n");
printf(" 9 - Voltar ao Menu\n");

scanf("%d", &opcao);

switch (opcao)
{
case 1:
    moverDisco(&A, &B);
    movimentos++;
    break;
case 2:
    moverDisco(&A, &C);
    movimentos++;
    break;
case 3:
    moverDisco(&B, &A);
    movimentos++;
    break;
case 4:
    moverDisco(&B, &C);
    movimentos++;
    break;
case 5:
    moverDisco(&C, &A);
    movimentos++;
    break;
case 6:
    moverDisco(&C, &B);
    movimentos++;
    break;
case 9:
    printf("Saindo do programa...\n");
    voltar = 1;
    break;
default:
    printf("Opcao invalida!\n");
    break;
}
system("cls");

if (jogoConcluido(&C))
{
    fim = clock();
    double tempoTotal = calcularTempo(inicio, fim);
    printf("\nParabens! Voce completou o jogo em %d movimentos \n", movimentos, tempoTotal);
    printf("Digite o seu nome: ");
    scanf("%s", &nome);
    adicionarRecorde(&recordes, tempoTotal, movimentos, nome);
    salvarRecordes(&recordes);
    system("cls");
    exibirMenu(&A,&B,&C);

    // Atualiza os melhores tempos
    int posicaoInserir = -1;
    for (int i = 0; i < 5; i++)
    {
        if (tempos[i] == -1 || tempoTotal < tempos[i])
        {
            posicaoInserir = i;
            break;
        }
    }

    if (posicaoInserir != -1)
    {
        for (int i = 4; i > posicaoInserir; i--)
        {
            tempos[i] = tempos[i - 1];
        }
        tempos[posicaoInserir] = tempoTotal;
    }

    movimentos = 0;
    inicializarHaste(&A);
    inicializarHaste(&B);
    inicializarHaste(&C);
}

```

```

        for (int i = 0; i < 5; i++)
        {
            empilharDisco(&A, &discos[i]);
        }

        printf("\nDigite ENTER para continuar...");
        getchar();
        getchar();

        inicio = clock();
    }
}
while (voltar == 0);
}
return 0;
}

```