UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Torre de Hanoi

Exercício substitutivo

João Pedro Queiroz Deger

- A **pilha** é utilizada pelo seu carater **FILO** que seriam as hastes no jogo
- Utilizamos tambem o ponteiro **p** para acessar/modificar as informações dessa pilha

```c
typedef struct {
    int topo;
    int discos[MAX_DISCOS];
} Pilha;
```

Funções basicas das pilhas

- Aqui temos as funções para iniciar e as outras funções para saber se a pilha está vazia ou cheia estas serão requisitadas por outras funções.

```c
void inicializar_pilha(Pilha *p) {
    p->topo = -1;
}

int pilha_vazia(Pilha *p) {
    return p->topo == -1;
}

int pilha_cheia(Pilha *p) {
    return p->topo == MAX_DISCOS - 1;
}
```

- O **push** tem a função de adicionar o disco no topo da torre se ela não estiver cheia

```c
void push(Pilha *p, int disco) {
    if (!pilha_cheia(p)) {
        p->discos[++(p->topo)] = disco;
    } else {
        printf("Movimento inválido: A pilha está cheia.\n");
    }
}
```

- O **pop** faz o reverso e tira um disco do topo da torre se ela não estiver vazia

```c
int pop(Pilha *p) {
    if (!pilha_vazia(p)) {
        return p->discos[(p->topo)--];
    }
    return -1;
}
```

- **Função de mover os discos no modo manual**
- Primeiro definimos o disco como o topo da **pilha** (com a função **pop** já mostrada) de origem através do **ponteiro** origem

```c
void mover_disco(Pilha *origem, Pilha *destino, char origemT, char destinoT) {
    int disco = pop(origem);
    if (disco != -1) {
        if (!pilha_cheia(destino) && (destino->topo == -1 || destino->discos[destino->topo] > disco)) {
            push(destino, disco);
            printf("Mover disco %d de %c para %c\n", disco, origemT, destinoT);
        } else {
            printf("Movimento inválido: Não é permitido colocar um disco maior sobre um disco menor.\n");
            push(origem, disco);  // Recoloca o disco na pilha de origem se o movimento for inválido
        }
    } else {
        printf("Movimento inválido!\n");
    }
}
```

- Se a **pilha** de destino não estiver cheia e for válida, o disco é movido com uma mensagem informativa. Caso contrário, o **movimento é inválido**, uma mensagem de erro é exibida, e o disco é recolocado na **pilha** de origem.

Parâmetros:

- **origem:** Ponteiro para a pilha de origem do disco
- **destino:** Ponteiro para a pilha de destino do disco.
- **origemT**: Caractere que representa a pilha de origem (por exemplo, 'A' para pilha A).
- **destinoT**: Caractere que representa a pilha de destino (por exemplo, 'C' para pilha C).

- **A função de mostrar as pilhas**
  - Percorre todos os elementos da pilha e exibe ela

```c
void exibir_pilhas(Pilha *A, Pilha *B, Pilha *C) {
    printf("A: ");
    for (int i = 0; i <= A->topo; i++) {
        printf("%d ", A->discos[i]);
    }
    printf("\nB: ");
    for (int i = 0; i <= B->topo; i++) {
        printf("%d ", B->discos[i]);
    }
    printf("\nC: ");
    for (int i = 0; i <= C->topo; i++) {
        printf("%d ", C->discos[i]);
    }
    printf("\n");
}
```

Parâmetros:

- A: Ponteiro para a pilha A.
- B: Ponteiro para a pilha B.
- C: Ponteiro para a pilha C.

- **Função de reiniciar o jogo**
  - Chama a função iniciar pilha fazendo o topo ficar “vazio” e joga todos os elementos e coloca na pilha A(com um push)

```c
void reiniciar_jogo(Pilha *A, Pilha *B, Pilha *C, int num_discos) {
    inicializar_pilha(A);
    inicializar_pilha(B);
    inicializar_pilha(C);
    for (int i = num_discos; i >= 1; i--) {
        push(A, i);
    }
}
```

Parâmetros:

- **A:** Ponteiro para a pilha A.
- **B:** Ponteiro para a pilha B.
- **C:** Ponteiro para a pilha C.
- **num_discos:** Número total de discos no jogo.

- **Função resolver automático**
- Resolve o a torre de Hanói automático utilizando-se de recursão

- **Primeira chamada recursiva:**
  Move discos - 1 discos da **pilha** de origem para a **pilha** auxiliar. Usa a **pilha** de destino como auxiliar temporária.
- **Movimento do maior disco:**
  Move o maior disco restante diretamente da **pilha** de origem para a **pilha** de destino.
- **Segunda chamada recursiva:**
  Move os discos - 1 discos da **pilha** auxiliar para a **pilha** de destino. Usa a **pilha** de origem como auxiliar temporária.

- **Ou seja a recursão é utilizada dividindo o problema em partes menores e movendo os discos conforme necessário até que a solução completa seja alcançada.**

```c
void mover_discos_automatico(int discos, Pilha *origem, Pilha *destino, Pilha *auxiliar, char origemT, char destinoT, char auxT) {
    if (discos == 1) {
        mover_disco(origem, destino, origemT, destinoT);
        return;
    }
    mover_discos_automatico(discos - 1, origem, auxiliar, destino, origemT, auxT, destinoT);
    mover_disco(origem, destino, origemT, destinoT);
    mover_discos_automatico(discos - 1, auxiliar, destino, origem, auxT, destinoT, origemT);
}
```

- **discos:** Número de discos a serem movidos.
- **origem:** Ponteiro para a pilha de origem dos discos.
- **destino:** Ponteiro para a pilha de destino final dos discos.
- **auxiliar:** Ponteiro para a pilha auxiliar utilizada durante a resolução.
- **origemT:** Caractere que representa a pilha de origem (por exemplo, 'A').
- **destinoT:** Caractere que representa a pilha de destino (por exemplo, 'C').
- **auxT**: Caractere que representa a pilha auxiliar (por exemplo, 'B').

- **Tempo no modo manual**
  - Foi utilizado a biblioteca **#time.h** e as funções:
  - **clock_t** é um tipo de dado que armazena ticks de clock.
  - **clock()** retorna o número de ticks de clock desde a inicialização do programa.

```
clock_t inicio = clock();
clock_t fim = clock();
double tempo_jogo = (double)(fim - inicio) / CLOCKS_PER_SEC;
```

- O calculo é feito pela diferença do fim pelo começo do jogo e é convertido para segundos

- **Funções para melhor tempo**

- Temos o vetor **melhores_tempos** que armazena os tempos em ordem crescente

- A função **listar_melhores_tempos** serve justamente para mostrar os melhores tempos, que são organizados pela função **atualizar_melhores_tempos** e que garanti que não passe do maximo de recordes

```
void listar_melhores_tempos(double melhores_tempos[], int num_tempos) {
    printf("Melhores tempos:\n");
    for (int i = 0; i < num_tempos; i++) {
        printf("%d. %.2f segundos\n", i + 1, melhores_tempos[i]);
    }
}

void atualizar_melhores_tempos(double melhores_tempos[], int *num_tempos, double novo_tempo) {
    if (*num_tempos < MAX_RECORDES) {
        melhores_tempos[*num_tempos] = novo_tempo;
        (*num_tempos)++;
    } else {
        for (int i = 0; i < MAX_RECORDES; i++) {
            if (novo_tempo < melhores_tempos[i]) {
                for (int j = MAX_RECORDES - 1; j > i; j--) {
                    melhores_tempos[j] = melhores_tempos[j - 1];
                }
                melhores_tempos[i] = novo_tempo;
                break;
            }
        }
    }
    // Ordena os tempos
    for (int i = 0; i < *num_tempos - 1; i++) {
        for (int j = i + 1; j < *num_tempos; j++) {
            if (melhores_tempos[i] > melhores_tempos[j]) {
                double temp = melhores_tempos[i];
                melhores_tempos[i] = melhores_tempos[j];
                melhores_tempos[j] = temp;
            }
        }
    }
}
```

- **melhores_tempos**: Array do tipo double que armazena os melhores tempos de jogo.
- **num_tempos:** Ponteiro para uma variável do tipo int que indica o número de tempos armazenados no array melhores_tempos.
- **novo_tempo:** Variável do tipo double que representa o novo tempo de jogo a ser adicionado ao array

UFPR
UNIVERSIDADE FEDERAL DO PARANÁ

Iniciação do jogo manual

- Ao começar o jogo o a função **clock_t é** usada e o **loop while 1** para continuar rodando o código até o fim do jogo
- A função **exibir_pilha** e chamada e um **scanf** é utilizado por um **swicht case** para saber qual ação o usuário quer fazer

```c
void iniciar_jogo_manual(Pilha *A, Pilha *B, Pilha *C, int num_discos, double melhores_tempos[], int *num_tempos) {
    int opcao;
    clock_t inicio = clock();

    while (1) {
        exibir_pilhas(A, B, C);
        printf("Digite o próximo movimento:\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");
        printf("7 - Reiniciar jogo\n");
        printf("8 - Sair\n");
        scanf("%d", &opcao);

        switch (opcao) {
            case 1: mover_disco(A, B, 'A', 'B'); break;
            case 2: mover_disco(A, C, 'A', 'C'); break;
            case 3: mover_disco(B, A, 'B', 'A'); break;
            case 4: mover_disco(B, C, 'B', 'C'); break;
            case 5: mover_disco(C, A, 'C', 'A'); break;
            case 6: mover_disco(C, B, 'C', 'B'); break;
            case 7:
                reiniciar_jogo(A, B, C, num_discos);
                inicio = clock();
                break;
            case 8:
                printf("Saindo...\n");
                return;
            default: printf("Opção inválida!\n");
        }

        if (C->topo == num_discos - 1) {
            printf("Parabéns! Você completou o jogo!\n");
            clock_t fim = clock();
            double tempo_jogo = (double)(fim - inicio) / CLOCKS_PER_SEC;
            printf("Tempo: %.2f segundos\n", tempo_jogo);
            atualizar_melhores_tempos(melhores_tempos, num_tempos, tempo_jogo);
            return;
```

- De 1 a 6 estão os movimentos entre “hastes” através da função **mover_discos**
- 7 reinicia o jogo com a função **reiniciar_jogo** e a 8 sai do modo manual
- A verificação da vitoria é feita a cada fim de movimento e verifica se todas as pilhas estão vazias menos a C que deve conter os discos na ordem correta se sim o jogo acaba e o tempo é armazenado e atualizado pela função **atualizar_melhores_tempos**

Função Principal/Menu do jogo

- Declara todas a variáveis necessárias além de obter o numero de discos escolhido pelo usuário e reinicia as pilhas
- O **while(1)** é utilizado para manter o programa rodando e o menu é mostrado com 5 opções, e um **swicht case** é utilizado para executar a ação escolhida pelo usuário
- **Opção 1:** As funções **reiniciar_jogo** e **iniciar_jogo_manual** são chamadas para reiniciar as pilhas e iniciar o jogo no modo manual, respectivamente.
- **Opção 2:** A função **reiniciar_jogo** é chamada para reiniciar as pilhas. A função **mover_discos_automatico** é chamada para resolver o problema da Torre de Hanói de forma automática, utilizando o algoritmo recursivo.
- **Opção 3:** A função **listar_melhores_tempos** é chamada para exibir os melhores tempos armazenados no array melhores_tempos.
- **Opção 4:** A função **printf** exibe uma mensagem solicitando ao usuário que digite o novo número de discos. A função **scanf** é utilizada para ler o valor digitado pelo usuário e armazená-lo na variável **num_discos**. A função **reiniciar_jogo** é chamada para reiniciar as pilhas com o novo número de discos
- **Opção 5:** A função **printf** exibe uma mensagem de despedida. A instrução **return 0;** finaliza o programa.