Introdução:

Este trabalho tem como principal objetivo consolidar os conhecimentos adquiridos em Tipos Abstratos de Dados (TAD), Listas Encadeadas e suas aplicações na linguagem de programação C através do desenvolvimento de uma aplicação prática. O projeto consiste na criação de um jogo de simulação baseado no clássico Monopoly, onde serão implementados e explorados conceitos essenciais de estrutura de dados e algoritmos.

Modelagem e Funcionamento

Estratégia de Desenvolvimento:

Definição dos Tipos Abstratos de Dados (TADs):

Tabuleiro: Estrutura que representa o tabuleiro do jogo, incluindo as localidades e o gerenciamento das propriedades.

Jogador: Estrutura que representa cada jogador, incluindo informações como saldo, posição e propriedades.

Localidade: Estrutura que representa cada propriedade no tabuleiro.

TADjogo: Estrutura que representa a integração das demais funções e estruturas para o funcionamento correto do jogo

Implementação
Estrutura de Dados:

- Tabuleiro
    - Tabuleiro.h

```c
#ifndef TABULEIRO_H
#define TABULEIRO_H

#include "localidade.h"

typedef struct Apontador {
    Localidade *localidade;
    struct Apontador *prox;
} Apontador;

typedef struct {
    Apontador *inicio;
    Apontador *fim;
    int tamanho;
} Tabuleiro;
```

Descrição: A princípio temos a estrutura “Apontador” que representa um nó, cada Apontador possui um ponteiro para uma localidade e um ponteiro para a próxima célula da lista circular. Para facilitar a implementação do código, temos um Apontador* início e Apontador* fim, além de uma variável do tipo Inteiro para manipular o tamanho do tabuleiro.

Funções do Tabuleiro e suas respectivas implementações:

```c
void inicializarTabuleiro(Tabuleiro *tabuleiro);
Apontador* criarApontador(Localidade *localidade);
void inserirLocalidade(Tabuleiro *tabuleiro, Localidade *localidade);
void imprimirTabuleiro(Tabuleiro *tabuleiro);
Apontador* AvancaCasa(Apontador* inicio, int rolagem);

#endif
```

Tabuleiro é inicializado sem nenhuma localidade inserida e com os ponteiros NULL (vazios)

```c
void inicializarTabuleiro(Tabuleiro *tabuleiro) {
    tabuleiro->inicio = NULL;
    tabuleiro->fim = NULL;
    tabuleiro->tamanho = 0;
}
```

```c
Apontador* criarApontador(Localidade *localidade) {
    Apontador *novoApontador = (Apontador*)malloc(sizeof(Apontador));
    if (novoApontador == NULL) {
        perror("Falha ao alocar memória para o Apontador");
        exit(EXIT_FAILURE);
    }

    novoApontador->localidade = (Localidade*)malloc(sizeof(Localidade));
    if (novoApontador->localidade == NULL) {
        perror("Falha ao alocar memória para a Localidade");
        free(novoApontador);
        exit(EXIT_FAILURE);
    }

    memcpy(novoApontador->localidade, localidade, sizeof(Localidade));
    novoApontador->prox = NULL;

    return novoApontador;
}
```

Essa função inicializa um espaço destinado a localidade e um apontador para a próxima localidade que também será inserida na lista circular. Para além disso, é utilizado a função de memory copy (memcpy), pois ele automatiza o processo de copiar manualmente os dados de uma estrutura localidade para o Apontador.

Funções de Imprimir e de Inserir tabuleiro:

```c
void imprimirTabuleiro(Tabuleiro *tabuleiro) {
    if (tabuleiro->inicio == NULL) {
        printf("Tabuleiro vazio.\n");
        return;
    }

    Apontador *atual = tabuleiro->inicio;
    printf("Início -> ");

    do {
        printf("%s -> ", atual->localidade->endereco);
        atual = atual->prox;
    } while (atual != tabuleiro->inicio);

    printf("Início\n");
}
```

```c
void inserirLocalidade(Tabuleiro *tabuleiro, Localidade *localidade) {
    Apontador *novoApontador = criarApontador(localidade);

    if (tabuleiro->inicio == NULL) {
        // Caso Lista esteja vazia
        tabuleiro->inicio = novoApontador;
        tabuleiro->fim = novoApontador;
        novoApontador->prox = novoApontador; // Garantimos a cirularidade
    } else {
        // Inserir antes do início e manter a circularidade
        novoApontador->prox = tabuleiro->inicio;
        tabuleiro->fim->prox = novoApontador;
        tabuleiro->fim = novoApontador;
        tabuleiro->tamanho++;
    }
}
```

Função “AvancaCasa”:

```c
Apontador* AvancaCasa(Apontador* inicio, int rolagem){
    Apontador *casaDeInicio = inicio;
    for(int i = 0; i < rolagem; i++){
        casaDeInicio = casaDeInicio->prox;
    }
    return casaDeInicio;
}
```

O ponteiro para a casa inicial é inicializado, e após isso, a função percorre um loop pelo número de casas correspondente à rolagem dos dados que o jogador realizou. Assim, a função **AvancaCasa** indica em qual casa o jogador "parou"

Resultado do Tabuleiro:

```
In├¡cio -> Savassi -> Pra├ºa Sete -> Av. Afonso Pena -> Pra├ºa da Liberdade -> Rua da Bahia -> Mercado Central ->
 Pal├ácio das Artes -> Parque Municipal -> UFMG -> Mineir├úo -> Lagoa da Pampulha -> In├¡cio
```

- Localidade
    - Localidade.h

```c
typedef struct {
    char endereco[200];
    char cor[20];
    int custoCompra;
    int valorAluguel;
    int construcao; // 0 - sem construção, 1 - Casa, 2 - Hotel
    int proprietario;
} Localidade;

void inicializarLocalidade(Localidade *localidade, const char *endereco, const char *cor,
int custoCompra, int valorAluguel);
```

Para a construção, utilizei os números 0, 1 e 2 para representar os diferentes níveis de construção nas localidades.

Além disso, foi adicionada uma variável do tipo int chamada "Proprietário" na estrutura, que será usada para atribuir um dono à localidade. Cada jogador possui um identificador único (ID), que será associado a essa variável para indicar a propriedade da localidade.

Implementação:

```c
void inicializarLocalidade(Localidade *localidade, const char *endereco, const char *cor,
 int custoCompra, int valorAluguel) {
    strcpy(localidade->endereco, endereco);
    strcpy(localidade->cor, cor);
    localidade->custoCompra = custoCompra;
    localidade->valorAluguel = valorAluguel;
    localidade->construcao = 0;
    localidade->proprietario = -1;
}
```

- Jogador
    - Jogador.h

```c
#include "localidade.h"

#define MAX_PROPRIEDADES 100

typedef struct {
    char nome[100];
    int id;
    int saldo;
    Localidade *propriedades[MAX_PROPRIEDADES];
    int posicao;
} Jogador;
```

- TADjogo
    - TADjogo.h

```c
#include "tabuleiro.h"
#include "jogador.h"
#include "localidade.h"

typedef struct {
    Tabuleiro tabuleiro;
    Jogador *jogadores;
    int numero_jogadores;
    int jogadorAtual;
} Jogo;

void PreencherJogo(Tabuleiro *tabuleiro, const char arq[]);
void adicionar_jogador(Jogador *jogadores, const char arq[], int *num_jogadores);
int rola_dados();
void MovimentoJogadores(Jogo* jogo, int a);
void comprarPropriedade(Jogador* jogador, Localidade* localidade);
void PagamentoAluguel(Jogador* jogador, Localidade* localidade, Jogador* jogadores, int numero_jogadores);
void construir(Localidade* Localidade, Jogador* jogador);
void Falencia(Jogador* jogador);
void proxRodada();
void PrintJogo();
void inicializarJogador(Jogador *jogadorA, char nomeN[], int saldo, int id);
void imprimeJogador(const Jogador *jogadores, int num_jogadores);
```

Principais Funções implementadas:

```c
void PreencherJogo(Tabuleiro *tabuleiro, const char arq[]) {
    FILE *file = fopen(arq, "r");
    if (file == NULL) {
        perror("Erro ao abrir o arquivo");
        exit(EXIT_FAILURE);
    }

    char linha[256];
    while (fgets(linha, sizeof(linha), file)) {
        // Processa cada linha do arquivo
        char endereco[200], cor[20];
        int custoCompra, valorAluguel;

        // Use o formato correto para a leitura dos dados
        if (sscanf(linha, "%199[^;];%19[^;];%d;%d", endereco, cor, &custoCompra, &valorAluguel) != 4) {
            fprintf(stderr, "Erro ao ler os dados da linha: %s", linha);
            continue; // Pula para a próxima linha em caso de erro
        }

        // Cria uma nova localidade e insere no tabuleiro
        Localidade localidade;
        inicializarLocalidade(&localidade, endereco, cor, custoCompra, valorAluguel);
        inserirLocalidade(tabuleiro, &localidade);
    }

    fclose(file);
```

Usa `sscanf` para extrair os dados da linha (endereço, cor, custo de compra e valor de aluguel). Se a leitura falhar, exibe uma mensagem de erro e continua com a próxima linha. Cria uma nova localidade com os dados lidos e a insere no tabuleiro.

A função “adicionar_jogadores” segue a mesma lógica da função anterior, utilizando de sscanf e fgets

Movimenta Jogador:

```c
void MovimentoJogadores(Jogo* jogo, int a) {
    Jogador *jogador = &jogo->jogadores[jogo->jogadorAtual];
    int prePosicao = jogador->posicao;

    jogador->posicao = (jogador->posicao + a) % jogo->tabuleiro.tamanho;
    Apontador *novaLoc = AvancaCasa(jogo->tabuleiro.inicio, jogador->posicao);

    if (jogador->posicao < prePosicao) {
        jogador->saldo += 200;
    }

    if (jogador->posicao == 0) {
        jogador->saldo += 500;
    }

    printf("%s se moveu de %d para %d - %s\n", jogador->nome, prePosicao, jogador->posicao,
     novaLoc->localidade->endereco);


    int idProprietario = novaLoc->localidade->proprietario;
    if (idProprietario == -1) {
        char escolha;
        printf("Você está na propriedade %s. Deseja comprar esta propriedade? (s/n): ",
         novaLoc->localidade->endereco);
        scanf(" %c", &escolha);
```

```
    if (escolha == 's' || escolha == 'S') {
        comprarPropriedade(jogador, novaLoc->localidade);
    } else {
        printf("Você optou por não comprar a propriedade.\n");
    }
} else if (idProprietario == jogador->id) {
    char escolha;
    printf("Você já possui a propriedade %s. Deseja construir algo? (s/n): ",
     novaLoc->localidade->endereco);
    scanf(" %c", &escolha);

    if (escolha == 's' || escolha == 'S') {
        construir(novaLoc->localidade, jogador);
    } else {
        printf("Você optou por não construir.\n");
    }
} else {
    Jogador *proprietario = &jogo->jogadores[idProprietario];
    int aluguel = 0;
if (novaLoc->localidade->construcao == 1) {
    aluguel = novaLoc->localidade->valorAluguel * 5;
}else if (novaLoc->localidade->construcao == 2) {
    aluguel = novaLoc->localidade->valorAluguel * 10;
}
```

Por meio dessa função todas as principais funcionalidades do Monopoly são feitas, como por exemplo:

Movimentação e Atualização de Posição,Compra de Propriedade,Construção em Propriedade do Jogador,Pagamento de Aluguel.

Construir:

```
void construir(Localidade* localidade, Jogador* jogador) {
    if (localidade->proprietario == jogador->id) {
        if (localidade->construcao == 0) {
            if (jogador->saldo >= localidade->custoCompra) {
                jogador->saldo -= localidade->custoCompra;
                localidade->construcao = 1;
                printf("%s construiu uma casa na propriedade %s\n", jogador->nome, localidade->endereco);
            } else {
                printf("%s não tem dinheiro suficiente para construir uma casa em %s\n",
                jogador->nome, localidade->endereco);
            }
        } else if (localidade->construcao == 1) {
            if (jogador->saldo >= localidade->custoCompra*5) {
                jogador->saldo -= localidade->custoCompra*5;
                localidade->construcao = 2;
                printf("%s construiu um hotel na propriedade %s\n", jogador->nome, localidade->endereco);
            } else {
                printf("%s não tem dinheiro suficiente para construir um hotel em %s\n",
                 jogador->nome, localidade->endereco);
            }
        } else {
            printf("Não é possível construir mais na propriedade %s\n", localidade->endereco);
        }
    } else {
        printf("%s não é o proprietário da propriedade %s\n", jogador->nome, localidade->endereco);
    }
}
```

Verifica se o jogador já é proprietário da localidade e faz as operações no saldo com base na alteração no valor de Custo de compra

Funcionalidade:

```
Jogador 1 (Renata), sua vez.
Deseja rolar os dados? (s/n): s

Jogador 1 (Renata) rolou: 2
Renata se moveu de 0 para 2 - Av. Afonso Pena
Voc├¬ est├í na propriedade Av. Afonso Pena. Deseja comprar esta propriedade? (s/n): s
Jogador Renata comprou a propriedade Av. Afonso Pena por 200

Saldo atual de Renata: R$500
Saldo de Renata: R$500
Saldo de Max: R$700
Saldo de Carolina: R$700
Saldo de Eduardo: R$700
Saldo de Marco Antonio: R$700
Saldo de Isabela: R$700
Pressione Enter para continuar...
Jogador 2 (Max), sua vez.
Deseja rolar os dados? (s/n): n
Voc├¬ optou por n├úo rolar os dados.
Pressione Enter para continuar...
Jogador 3 (Carolina), sua vez.
Deseja rolar os dados? (s/n): s

Jogador 3 (Carolina) rolou: 11
Carolina se moveu de 0 para 1 - Pra├ºa Sete
Voc├¬ est├í na propriedade Pra├ºa Sete. Deseja comprar esta propriedade? (s/n):
```

Forma de compilar:

```
PS C:\Users\Usuario\Downloads\TP_almostthere> gcc main.c jogador.c localidade.c tabuleiro.c TADjogo.c -o programa
```

Conclusão:

Durante o desenvolvimento deste trabalho, enfrentei desafios significativos relacionados à organização e estruturação do código no arquivo `main.c`. Inicialmente, a implementação do roteiro do programa apresentou dificuldades, pois muitas funcionalidades não estavam totalmente integradas e coesas. Além disso, ao longo do desenvolvimento, percebi a importância de realizar testes constantes das funcionalidades. A falta de testes adequados levou à necessidade de revisões e refatorações substanciais em várias partes do código. Esses ajustes foram essenciais para garantir que o programa funcionasse conforme o esperado e para resolver problemas de integração que surgiram. A experiência destacou a importância de

uma abordagem metódica e de testes contínuos durante o desenvolvimento de software.

Referências bibliográficas:

https://www.w3schools.com/c/

https://www.youtube.com/watch?v=_dWDe6JnMzg