



**Universidade Federal de Sergipe  
Departamento de Computação  
Docente: Beatriz Trinchão Andrade de Carvalho**

**Programação Imperativa  
Turma 02**

**Discentes:**

**Bruno Carvalho Santana Rocha  
Danilo Campos Deichmann  
Filipe Ciríaco Marcelino do Nascimento  
Luiz Manoel Rosa Nunes Menezes**

**São Cristóvão - SE  
11/04/2025**

# 1 INTRODUÇÃO

Esse relatório tem como objetivo explicar de forma clara e objetiva o simulador de labirinto feito na linguagem C, apresentado pela disciplina de Programação Imperativa. No relatório será apresentado como solucionamos os principais problemas e a nossa trajetória até o fim desse projeto.

## 2 DESENVOLVIMENTO

### 2.1 Leitura do Arquivo

Primeiramente, foi necessário raciocinar sobre como fazer a leitura e guardar em memória o labirinto. Antes de pensar em como armazenar, foi necessário ler a primeira linha do texto para saber qual é o tamanho do labirinto. Em princípio tentamos utilizar a função `fscanf()`, porém, por algum motivo desconhecido, dava algum problema ao receber os valores, diante disso, buscamos outra solução. Logo tentamos utilizar a função `fgets()` para ler a primeira linha e transformar em uma string, após isso, usamos a função `sscanf()` para formatar a string separando ela nos espaços e transformando em números inteiros.

Nosso primeiro pensamento foi utilizar um `fgetc()` para ler cada carácter, porém ao perceber que o labirinto original possui uma quantidade considerável de espaços entre os caracteres, utilizar esta função de forma simples seria inviável. Logo, preferimos utilizar `fscanf()`, pois com ele foi possível ignorar o espaço ao passar como argumento `"%C"`.

### 2.2 Estrutura do Personagem e Labirinto

Para representar o personagem, foi criada uma estrutura homogênea, chamada de atributos, que nela é armazenado a posição do player (`PosI` e `PosJ`), a posição do ponto de chegada (`endI` e `endJ`), o poder (`power`) do personagem que será mais abordado nos próximos parágrafos e os caminhos que o personagem poderá seguir (`N`, `S`, `L` e `O`), o player é do tipo atributos. Para armazenar, o labirinto foi usado uma matriz bidimensional de tipagem `char`.

### 2.3 Combate e Fim de Jogo

Foi adicionado a uma estrutura homogênea o poder do personagem, escrito como `power`, começando o poder em 50 pontos. Logo em seguida, um número aleatório é gerado de 0 a 99, se o número sorteado estiver entre 0 e 49, o personagem ganha, mudando a sua posição atual para (!), se não estiver entre esses números, ele perde, alterando sua posição atual para (+). Caso ele ganhe, o poder (`power`) do player aumentará em 10 pontos, assim, aumentando o número sorteado para entre 0 e 59 e assim por diante. A implementação de toda a luta do personagem está na função `batalha`.

Entretanto, ao se perder ou chegar a uma posição final, a parte do código promoverá isso é a da função `mov_aleatoria`. No momento em que o player tem todos os seus caminhos indisponíveis(sejam eles, o limite o labirinto, a parede(#) ou o caminho já passado anteriormente(\*)), a função `mov_aleatoria` chama a função `rastroPlayer` para adicionar a (?) na posição atual do personagem para simbolizar que ele se perdeu. Para saber que o player chegou até a posição final, o código identifica a posição atual do player e a posição da chegada(?), quando a posição atual e a posição final forem iguais, significa que ele escapou do labirinto.

## 2.4 Movimentação do Jogador

O método implementado para a movimentação do personagem é dito através da geração de um número aleatório que define a direção do próximo movimento, toda a lógica de movimentação está dentro da função `mov_aleatoria`. Primeiro, o código identifica a posição atual do personagem por meio da função `posicao` e avalia os destinos viáveis (norte, sul, leste e oeste) utilizando a função `verificandoArea`. Os destinos são considerados disponíveis se não contiverem uma parede (#), se não forem um ponto já visitado (marcado com \*) e se estiverem dentro dos limites do labirinto. Assim que um destino válido é selecionado, a função `rastroPlayer` atualiza a posição anterior para indicar que aquele caminho já foi percorrido.

Pensando em agilizar a forma que o personagem conclui o labirinto, adicionamos a função `mov_aleatoria_inteligente`. Diferente da função `mov_aleatoria`, o personagem tem mais probabilidade de ir em direção ao seu objetivo final, o ponto de chegada(\$), por exemplo, se o ponto de chegada está na parte sul do labirinto, então o personagem tem maiores chances de ir também para a parte sul do que para alguma outra direção.

# 3 INSTRUÇÕES DE COMPILAÇÃO

## 3.1 Linux

1. É necessário compilar primeiro, utilizando o comando:  
`gcc ./main.c -o main`
2. Agora só rodar o aplicativo utilizando o labirinto como argumento, exemplo:  
`./main lab5x5.txt`  
(Necessário colocar o .txt)

## 3.2 Windows

1. É necessário compilar primeiro, utilizando o comando:  
`gcc ./main.c -o main.exe`
2. Agora só rodar o aplicativo utilizando o labirinto como argumento, exemplo:  
`./main.exe lab5x5.txt`  
(Necessário colocar o .txt)

## **4 Conclusão**

Ao concluir esse projeto percebemos que os ensinamentos em sala de aula, poderia ser colocado de forma prática e adquirir ainda mais conhecimento sobre a linguagem C. Foi de grande alegria participarmos desse projeto sendo ministrado pela honorável docente Beatriz Trinchão Andrade de Carvalho, que nos ajudou imensamente em todo o trajeto e queríamos agradecer todos os ensinamentos que nos tem passado durante esse período.