

## Informações gerais

Neste laboratório vamos exercitar nossos conhecimentos de recursão e de backtracking; para este fim, vamos implementar um programa que utiliza um TAD para manipulação de um objeto que descreve um labirinto e o posicionamento de uma ou mais pessoas perdidas no mesmo. Obviamente, implementar operações para retirar essas pessoas com segurança desse labirinto é uma prioridade nossa! 😊

### Observações importantes:

1. Neste laboratório será permitido o uso apenas das bibliotecas `stdio.h`, `stdlib.h`, `string.h` e `math.h`, além do arquivo de interface do laboratório, que é o `labirinto.h`.
  2. O arquivo `labirinto.h`, que contém os protótipos de tipos e operadores do nosso TAD, não deve ser modificado em hipótese alguma. Alterações devem ser feitas apenas nos arquivos de implementação e de cliente (`labirinto.c` e `cliente.c`).
  3. Para compilar a sua questão, utilize o `Makefile`, chamando no terminal o comando `make`.
- 

## TAD para manipulação de labirintos

Para este trabalho, você terá que lidar com um desafio comum em jogos e simulações: **labirintos**. No contexto deste trabalho, labirintos são representações bidimensionais onde um agente (representado pelo símbolo **P**) tenta encontrar a saída (representada pelo símbolo **S**), navegando por caminhos (representados pelo número **0**) e evitando obstáculos ou paredes (representados pelo número **1**).

O seu objetivo é criar uma **representação abstrata** deste labirinto como um **Tipo Abstrato de Dados (TAD)**. Isso facilitará a manipulação, busca e análise dos labirintos.

A estrutura do labirinto, `struct labirinto`, deve conter os seguintes campos:

- Dimensões do labirinto (número de linhas e colunas);
- Matriz representando o labirinto;
- Posição da saída.

### Para manipular esta estrutura, implemente os seguintes operadores:

- **criar\_labirinto**: aloca e inicializa a estrutura de labirinto com as dimensões especificadas e retorna o ponteiro para ela;
- **ler\_labirinto**: lê os dados de um labirinto da entrada padrão;
- **buscar\_saida\_recursivamente**: função auxiliar recursiva que busca uma saída no labirinto;
- **buscar\_saida**: função que busca a saída do labirinto a partir de um ponto inicial;
- **imprimir\_labirinto**: imprime a matriz do labirinto na saída padrão, mostrando o caminho encontrado, se existir;
- **destruir\_labirinto**: libera a memória alocada para a estrutura do labirinto.

Os operadores mencionados estão definidos como funções no arquivo `labirinto.h`. Junto a essas definições, existe uma documentação detalhada explicando os tipos dos parâmetros, a funcionalidade de cada operador e o que retornam. Além disso, a estrutura do tipo `labirinto` está declarada nesse arquivo, sendo representada como uma `struct`, conforme explanado anteriormente.

Nas próximas etapas, você trabalhará no arquivo `labirinto.c` para implementar esses

operadores. Ao concluir, o programa será capaz de ler, identificar a saída e mostrar labirintos com variadas dimensões e estruturas.

---

## Questão 1 (6 Pontos) - Uma pessoa no labirinto

Nesta questão, será implementada uma solução para ajudar **uma única pessoa** a navegar em um labirinto. O objetivo é criar um programa que permita que essa pessoa encontre o caminho correto para sair do labirinto, lidando com desafios como paredes e possíveis becos sem saída.

### Algumas definições do problema:

1. O labirinto é representado como uma matriz bidimensional onde:
  - 0 representa um caminho livre.
  - 1 representa uma parede ou obstáculo.
  - P representa a posição inicial da pessoa no labirinto.
  - S representa a saída do labirinto.
2. Você receberá uma linha de entrada especificando o tamanho da matriz e, após isso, irá receber a matriz que representa o design do labirinto com as posições da pessoa e saída marcadas.
3. O seu programa deve encontrar a saída e imprimir o labirinto com o **caminho de saída traçado** no mesmo.

Em casos onde houver mais de uma saída possível, a ordem de prioridade das direções possíveis (da mais prioritária para a menos prioritária) é:

→ Direita →  
↓ Baixo ↓  
← Esquerda ←  
↑ Cima ↑

### Exemplo de entrada:

```
7x6
1 1 1 1 1 1
1 P 0 0 0 1
1 1 1 0 1 1
1 1 1 0 1 1
1 0 0 0 1 1
1 1 1 0 S 1
1 1 1 1 1 1
```

### Saída esperada:

```
1 1 1 1 1 1
1 P X X 0 1
1 1 1 X 1 1
1 1 1 X 1 1
1 0 0 X 1 1
1 1 1 X X 1
```

1 1 1 1 1 1

---

## Questão 2 (4 Pontos) - Mais pessoas no labirinto

Nesta questão, teremos a presença de **duas ou mais pessoas** em um labirinto e a tarefa será implementar uma solução que permita encontrar a saída individual para cada uma delas.

Isso significa que cada pessoa no labirinto terá que encontrar seu próprio caminho para escapar, e o programa ou algoritmo desenvolvido deverá imprimir a saída específica para cada uma, considerando as diferentes posições iniciais e obstáculos no labirinto.

**Exemplo de entrada com duas pessoas no labirinto:**

7x6

```
1 1 1 1 1 1
1 P 0 0 0 1
1 1 1 0 1 1
1 1 1 0 1 1
1 P 0 0 1 1
1 1 1 0 S 1
1 1 1 1 1 1
```


**Saída esperada (duas matrizes separadas):**

```
1 1 1 1 1 1
1 P X X 0 1
1 1 1 X 1 1
1 1 1 X 1 1
1 P 0 X 1 1
1 1 1 X X 1
1 1 1 1 1 1
```

```
1 1 1 1 1 1
1 0 0 0 0 1
1 1 1 0 1 1
1 1 1 0 1 1
1 P X X 1 1
1 1 1 X X 1
1 1 1 1 1 1
```

**Observação:**

A ordem de impressão das saídas das pessoas deve seguir a **ordem das posições delas na matriz**, percorrendo da **esquerda para a direita e de cima para baixo**.

 **Cuidado para não passar por uma posição que já tem alguém!**