MC202 — ESTRUTURAS DE DADOS

Laboratório 07 — Labirinto

Tarefa

Nesta tarefa, o aluno deve exercitar os conceitos de "backtracking" implementando um programa que, dado um labirinto, retorna o caminho que leva da entrada até a saída. O labirinto será especificado como uma matriz M por N, com uma posição de entrada e uma de saída. As demais posições vão ser ou posições livres (por onde se pode passar), ou obstáculos (posições por que não se pode passar).

A primeira linha da entrada do programa consiste em 6 valores inteiros:
LT CT LE CE LS CS

Cada valor representa o seguinte:

LT	Número total de linhas da matriz.
CT	Número total de colunas da matriz.
LE	Índice da linha da posição de entrada.
CE	Índice da coluna da posição de entrada.
LS	Índice da linha da posição de saída.
CS	Índice da coluna da posição de saída.

Em seguida, será informada a matriz que representa o labirinto. Casa posição da matriz será um caractere, que pode ser 'O' quando for uma posição vazia, ou 'X' quando for uma posição com obstáculo.

Dentro do labirinto existem apenas 4 movimentos válidos:

- Para cima
- Para baixo
- Para a esquerda

- Para a direita

Movimentos na diagonal não são permitidos.

A saída do programa deve ser uma matriz com as mesmas dimensões que a matriz de entrada. Nas posições que fazem parte do caminho, deve haver o caractere '*' e, em todas as demais posições, deve haver o caractere ' ' (espaço).

Exemplo

Entrada:

```
5 5 3 1 2 0
XOOOX
OOXOO
OXOXO
XOOXO
XXOOO
```

Saída:

```
 ***
** **
*   *
 ** *
  ***
```

Dicas:

A matriz representando o labirinto é formada por caracteres, logo deve-se ler cada um deles usando `scanf("%c", &c);` Lembre-se que no final de cada linha existe o caracter `'\n'` que não faz parte da matriz. Esse caractere deve ser lido e descartado antes de ler a linha seguinte.

Os índices de entrada e saída vão de 0 até o tamanho da linha/coluna - 1.
Se a linha tem tamanho 15, o índice vai ser um valor entre 0 e 14.

Critérios específicos

- Para as turmas E e F, este laboratório tem peso 2.

- Para as turmas G e H, este laboratório tem peso 2.
- Deverá ser submetido o seguinte arquivo: **lab07.c**.
- Tempo máximo de execução: 2 segundos.

Testando

Para compilar com o Makefile fornecido e verificar se a solução está correta basta seguir o exemplo abaixo.

```
make
./lab07 < arq01.in > arq01.out
diff arq01.out arq01.res
```

onde `arq01.in` é a entrada (casos de testes disponíveis no SuSy) e `arq01.out` é a saída do seu programa. O Makefile também contém uma regra para testar todos os testes de uma vez; nesse caso, basta digitar:

```
make testar_tudo
```

Observações gerais

No SuSy, haverá 3 tipos de tarefas com siglas diferentes para cada laboratório de programação. Todas possuirão os mesmos casos de teste. As siglas são:

1. **SANDBOX:** Esta tarefa serve para testar o programa no SuSy antes de submeter a versão final. Nessa tarefa, tanto o prazo quanto o número de submissões são ilimitados, porém arquivos submetidos aqui **não serão corrigidos**.
2. **ENTREGA:** Esta tarefa tem limite de **uma única submissão** e serve para entregar a versão final dentro do prazo estabelecido para o laboratório. Não use essa tarefa para testar o seu programa: submeta aqui quando não for mais fazer alterações no seu programa.
3. **FORAPRAZO:** Esta tarefa tem limite de **uma única submissão** e serve para entregar a versão final após o prazo estabelecido para o laboratório, mas com nota reduzida (conforme a ementa). O envio nesta tarefa irá substituir a nota obtida na tarefa ENTREGA apenas se o aluno tiver realizado as correções sugeridas no feedback ou caso não tenha enviado anteriormente em ENTREGA.

Observações sobre SuSy:

- Versão do GCC: C-ANSI 4.8.2 20140120 (Red Hat 4.8.2-15).
- Flags de compilação:
  `-ansi -Wall -pedantic-errors -Werror -g -lm`
- Utilize comentários do tipo `/* comentário */`;
  comentários do tipo `//` serão tratados como erros pelo SuSy.

Além das observações acima, esse laboratório será avaliado pelos critérios gerais:

- Indentação de código e outras boas práticas, tais como:
  - uso de comentários (apenas quando forem relevantes);
  - código simples e fácil de entender;
  - sem duplicidade (partes que fazem a mesma coisa).
- Organização do código:
  - tipos de dados criados pelo usuário e funções bem definidas e tão independentes quanto possível.
- Corretude do programa:
  - programa correto e implementado conforme solicitado no enunciado;
  - inicialização de variáveis sempre que for necessário;
  - dentre outros critérios.