Exercicío 5 – Pilhas

1) Estenda a biblioteca da calculadora e inclua novos operadores raiz quadrada e exponenciação.

2) Resolução de problemas de tentativa e erro. Leia o contexto e tente resolver esse programa usando um estrutura do tipo pilha.

Para implementar uma técnica conhecida como backtracking(ou retrocesso), frequentemente usada na inteligência artificial para resolver problemas por meio de tentativa e erro. Essa técnica é útil em situações em que, a cada instante, temos várias opções possíveis e não sabemos avaliar a melhor. Então, escolhemos uma delas e , caso essa escolha não leve à solução do problema, retrocedemos e fazemos uma nova escolha.

Para ilustrar o uso dessa técnica , vamos considerar o problema em que um rato preso num labirinto precisa achar um caminho que o leve à saída, Figura 1. Para isso, ele tenta sistematicamente cada caminho. Quando um caminho o leva a um beco, ele retrocede até um ponto onde possa tentar outro caminho ainda não explorado.

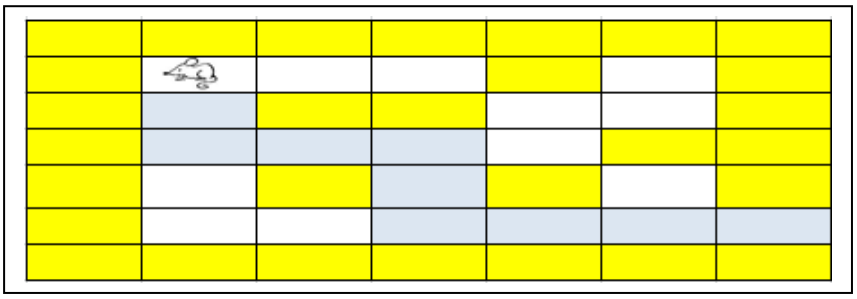

Figura 1

Para representar o labirinto, vamos usar uma matriz quadrda nXn, cuja posições armazenam uma das seguintes marcar: *livre*, *parede*, *visitada* ou *beco*,e apenas posições livres podem ser alcançadas pelo rato. Quando uma posição livre é alcançada, sua marca é alterada para *visitada* e quando fica determinado que uma posição visitada conduz a um beco, sua marca é alterada para *beco*. As definições necessárias para a criação dessa matriz são apresentadas na Tabela 1. **(TRANSFORME EM UM TAD!)**

```
#define tam 3

enum  marca   {  livre = 1,   parede=2, visitada=3, beco=4 };
typedef enum marca tipo_marca;

typedef tipo_marca tipo_matriz[tam][tam];

tipo_matriz mat;
```

Tabela 1 – representação do labirinto

A cada vez que um labirinto é criado, as bordas da matriz são marcadas como paredes e sua configuração interna é definida aleatoriamente. Além disso, a posição inicial do rato do rato *(2,2)* e a posição de saída do labirinto *(tam-1,tam)* são marcadas como livres.

Para definir a configuração interna da matriz, usamos a função random(m), que a cada chamada sorteia um valor do conjunto {0,1,2,...,m-1}. Assim, por exemplo , a probabilidade de random(3) devolver 0 como resposta é de 1/3. Então, para que tenhamos aproximadamente 2/3 das posições internas livres, marcamos uma posição como parede apenas quando random(3) devolver 0. Veja o código da Tabela 2 que

contém a operação criar o labirinto (está em portugol, passe para a linguagem C e INCLUA A OPERAÇÃO EM UM TAD!):

```
Procedimento cria(var L: tipo_matriz)
var i, j: inteiro;
inicio
    para i:=1 até tam faça
    inicio
        L[1,i] := parede;
        L[n,i] := parede;
        L[i,1] := parede;
        L[i,n] := parede;
   Fim;
Para i:=2 até n-1 Faça
    Para j:=2 até n-1 Faça
         Se random(3)=0 entao L[i,j]:=parede;
         Senão                      L[i,j]:=livre;

L[2,2]:=livre;
L[n-1,n]:=livre;
Fim;
```

Tabela 2

Para facilitar a visualização do processo de busca da saída do labirinto, **crie uma operação que exiba a matriz no vídeo**. Siga as seguinte diretrizes para imprimir a matriz: * *livre* será representado por espaço em branco; * *parede* será representado por um bloco sólido printf("%c",219); * *visitada* será representada por um ponto; * *beco* será representada por um bloco pontilhado printf("%c",176);

Para encontrar a saída do labirinto crie uma operação SAI com as seguintes regras (USE ESTRUTURA PILHA):

- Defina a posição inicial do rato como (2,2).
- Inicie uma pilha **P** vazia.
- Até que a posição corrente(i,j) se torne a posição de saída (n-1,n):
    - Marcamos a posição corrente (i,j) como visitada.
    - Se houver uma posição livre adjacente à posição corrente, empilhamos a posição corrente e nos movimentamos para essa posição livre.
    - Senão, estamos num beco e precisamos retroceder à última posição pela qual passamos para explorar um outro caminho. Para isso, desempilharmos uma posição de **P**, que passa a ser a nova posição corrente. Caso a pilha esteja vazia, o labirinto não tem saída e a busca fracassa.
    - Alcançada a posição de saída, a busca termina com sucesso.

- Para facilitar a codificação do algoritmos, use uma pilha de inteiros e transforme o par de coordenada(i,j), num inteiro correspondente i*100+j. Por exemplo, o par de coordenadas(13,12) é empilhado como 13*100+12 que é igual a 1312. Ao desempilhar esse número, pode-se restaurar o par de coordenadas original, fazer: int c = 1312/ 100 = 13; int d = 1312 = 12 % 100. Esse artifício funciona corretamente apenas quando a coordenada tem no máximo dois dígitos.

Faça um programa principal que use a operação CRIA labirinto e SAI do labirinto.