UNIVERSIDADE FEDERAL DO PARANÁ
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS
DS-130 - ESTRUTURAS DE DADOS I – 3º. PERÍODO

MÓDULO II – TRABALHO PRÁTICO II

Postagem do Trabalho no Moodle – 12/12/2021 (domingo)
Defesas do Trabalho: 13 a 17/12/2021

TRABALHO PRÁTICO II

1. **Aplicação de Pilha: escolha um dos dois problemas abaixo para a equipe implementar: (50 PONTOS)**

Problema 1: Calculadora de Expressões Numéricas

Desenvolver um programa em C que manipule expressões numéricas como segue o menu abaixo:

2. Transforma uma expressão Infixa em Pósfixa
3. Transforma uma expressão Infixa em Préfixa
4. Lê uma expressão e informa se a expressão é válida
5. Lê uma expressão e informa o resultado da expressão
6. Sair

Requisitos:

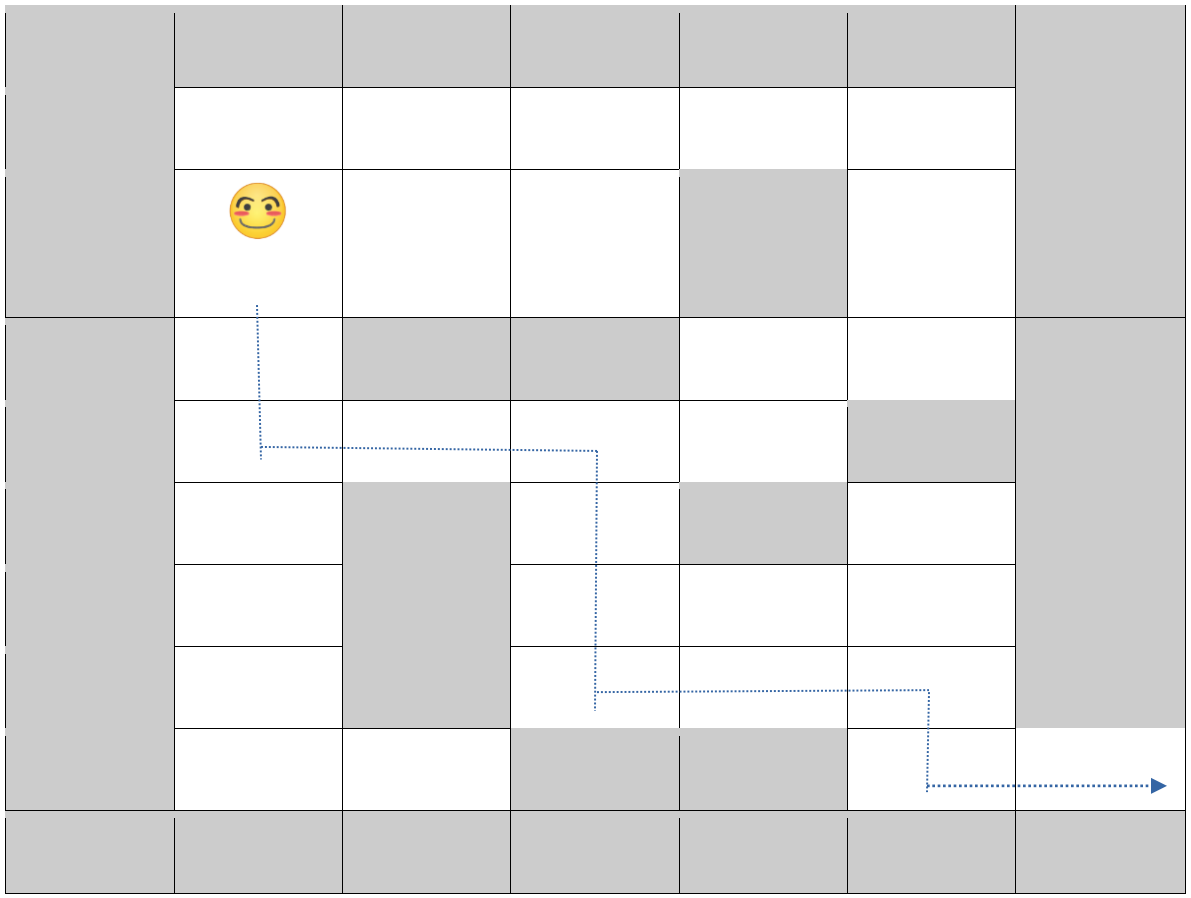
1. Deve ser aplicada a estrutura de dado pilha com alocação encadeada para calcular a expressão. Obs. pesquisar o algoritmo de cálculo de expressões com uso de pilha na internet ou livros de Estruturas de Dados.
2. O usuário poderá utilizar as seguintes operações em suas expressões: +, -, *, / e exponenciação. Além disso, as expressões podem conter parênteses para indicar prioridades de operações.
   Exemplo de expressão -**(5 + 8) * 7 – expo(2,3) /5**
3. O programa deve ser implementado de forma modular, ou seja, implementação de funções com passagens de parâmetros.

Problema 2: Aplicação de Pilha: O problema do labirinto

O objetivo deste exercício é usar uma pilha para implementar uma técnica conhecida como *backtracking* (ou retrocesso), frequentemente usada em 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 o 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, aplicaremos o problema do rato em um labirinto. O rato está preso no labirinto e precisa achar o caminho que o live a à saída, conforme ilustra a imagem abaixo



Para tanto, você precisa:

- Declarar uma matriz quadrada 30X30 para representar o labirinto.
- As posições da matriz podem armazenar uma das seguintes marcas: livre, parede, visitada ou beco (você pode definir essas marcas como constantes inteiras). Inicialmente, toda posição é marcada como livre ou parede, 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.

- Toda 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 e a posição de saída do labirinto são marcadas como livres e definidas de forma aleatória.
- Para definir a configuração interna da matriz, deve-se usar a função *random*.
- Para facilitar a visualização do processo de busca da saída do labirinto, ao exibir a matriz em vídeo, as posições devem ser marcadas como:
    - livre: será representada por um espaço em branco;
    - parede: será representada por um bloco sólido (ASCII 219);
    - visitada: será representada por um ponto;
    - beco: será representada por um bloco pontilhado (ASCII 176);
    - posição em que o rato se encontra no labirinto, no momento em que ele é exibido: será representada pelo caracter - ASCII 1 (carinha feliz)

Obervação: A equipe pode escolher outra representação para os elementos do labirinto.

- Para encontrar a saída do labirinto, aplicar o seguinte algoritmo:
    - Definir (i,j) como posição corrente do rato, que inicialmente é (2,2);
    - Iniciar uma pilha P vazia;
    - Até que a posição corrente (i,j) se torne a posição de saída (n-1, n):
        - Marcar a posição corrente (i,j) como visitada;
        - Se houver uma posição livre adjacente a posição corrente, empilhamos a posição corrente e movimentamos o rato para essa posição livre;
        - Senão, estamos num beco e precisamos retroceder à última posição pela qual passamos para explorar outro caminho. Para isso, desempilhamos 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 manipulação da pilha, deve ser aplicado uma pilha de inteiros. Para tanto, deve ser transformado o par de coordenadas (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,

podemos restaurar o par de coordenadas original, fazendo **i = 1312 div 100** cujo resultado é 13 e **j = 1312 mod 100** cujo resultado é 12. Esse artifício funciona corretamente apenas quando cada coordenada tem no máximo dois dígitos.

Serão avaliados os seguintes itens na implementação da solução desse problema:

- Manipulação de uma pilha encadeada;
- Aplicação de funções, passagem de parâmetro por valor e por referência, retorno de função;
- Implementação de todos os elementos especificados acima.

2. Aplicação de Fila: escolha um dos dois problemas abaixo para a equipe implementar (50 pontos)

Problema 1: Cadastro de pacientes que necessitam de doação de coração

Um hospital de cardiologia precisa de um sistema para efetuar o cadastro de pacientes que necessitam de doação de coração. Para cada paciente que é incluído no sistema deve ser informado o nome, telefone e o grau de urgência para transplante. O grau de urgência é definido na seguinte escala: (5) Muito urgente; (4) Urgente; (3) Médio; (2) Pouco urgente; (1) Sem urgência. Sempre que o hospital recebe um novo coração o sistema é consultado para obter o próximo paciente que deverá ser operado. O sistema informa o nome e o telefone do paciente. Também a qualquer momento é possível visualizar o tamanho da fila de espera. Observação: os dados não precisam ser persistidos em arquivos, podem ficar armazenados somente na memória.

O sistema deve ter o seguinte menu:

1. Cadastrar paciente (Nome, telefone e grau de urgência)
2. Buscar paciente (informar a posição do paciente na fila e o seu grau de urgência)
3. Próximo paciente a ser operado (retorna os dados do paciente e exclui da lista*)
4. Verificar tamanho da fila
5. Sair

* Ao excluir da lista o paciente que receberá a próxima doação de coração, seus dados devem ser gravados em um arquivo txt. Este arquivo armazenará os dados de todos os pacientes já transplantados.

Para tanto, uma Fila de Prioridade deve ser implementada. Existem diversas maneiras de se implementar uma fila de prioridade. Em qualquer implementação, o objetivo é acessar eficientemente o item de maior prioridade que se encontra na lista. As abordagens mais comuns são:

- Lista encadeada sem ordenação. Nesse caso, a inserção é feita no início da lista com custo temporal θ(1) e a remoção pode ocorrer em qualquer nó da lista com custo temporal θ(n). Ou seja, remoção requer uma busca sequencial na lista para encontrar o item com a maior prioridade.
- Lista encadeada ordenada. Supondo que uma lista encadeada seja mantida ordenada em ordem decrescente de prioridade, inserir um item requer que a posição de inserção seja encontrada para manter a lista ordenada a cada inserção e, portanto, o custo temporal dessa operação é θ(n). A remoção é sempre feita no início da lista com custo temporal θ(1).
- Lista indexada sem ordenação. Nesse caso, a inserção é feita ao final da lista com custo temporal θ(1) e a remoção pode ocorrer em qualquer posição da lista com custo temporal θ(n).
- Lista indexada ordenada. Remoção é uma operação com custo temporal θ(1) se a lista estiver em ordem crescente de prioridade, de modo que o elemento removido é sempre o último elemento da lista. Inserção requer que se encontre o local de inserção do item com custo θ(log n), se for usada busca binária. Mas, por outro lado, inserção também requer rearranjo dos elementos da lista, que é uma operação com custo temporal θ(n). Logo a combinação das duas operações tem custo θ(n).
- Árvore binária de busca balanceada. Nessa abordagem, inserção e remoção têm o mesmo custo temporal θ(log n). Por outro lado, essa é a opção com mais alto custo de implementação.
- Heap binário (ou apenas heap). Essa abordagem oferece várias vantagens, tais como simplicidade, rapidez e uso de pouco espaço de armazenamento. Além do mais, essa abordagem é muito fácil de implementar.

Serão avaliados os seguintes itens na implementação da solução desse problema:

- Manipulação de uma fila de prioridade encadeada;
- Aplicação de funções, passagem de parâmetro por valor e por referência, retorno de função;

- Implementação de todos os elementos especificados acima.

Problema 2: Fila de impressão

Em um ambiente computacional em rede existe uma impressora que é compartilhada por todos. Para gerenciar o uso dessa impressora, o sistema operacional dispõe de 4 filas, a saber:

1. Fila de Entrada: recebe as solicitações de impressão de origem diversa;
2. Fila 0: recebe todas as solicitações de impressão da fila de entrada com prioridade máxima;
3. Fila 1: recebe as solicitações de impressão da fila de entrada com prioridade normal;
4. Fila 2: recebe as solicitações de impressão da fila de entrada com prioridade baixa;

Cada nó destas filas é formado por um registro com os seguintes campos: prioridade, identificação e ponteiro com o endereço do próximo nodo. Em cada 0.5 segundo **(pesquise a função delay para isso)** o programa deve descarregar 5 solicitações da fila de entrada, distribuindo-as entre as filas com prioridades. Para isso, escreva um programa em C que carrega a fila de entrada com solicitações de impressões e suas respectivas prioridades (a fila deve ser inicializada com no mínimo 50 solicitações), distribuiu as solicitações de impressão em suas respectivas filas de prioridade e imprime na tela a situação de cada fila conforme o programa vai processando as distribuições.

Requisitos:

- Deve ser aplicada a estrutura de dado fila com alocação encadeada.
- As prioridades devem ser definidas da seguinte forma:

  Máxima = 1
  Normal = 2
  Baixa = 3

- A identificação pode ser definida pelo IP do computador que enviou a solicitação.
- As solicitações que serão carregadas na Fila de Entrada devem ser lidas de um arquivo txt com o seguinte formato:

prioridade1, identificação1; prioridade2, identificação2; prioridade3, identificação3; prioridade2, identificação4; prioridade1, identificação5; ……; prioridade3, identificação50.

- O programa deve ser implementado de forma modular, ou seja, implementação de funções com passagens de parâmetros.

ATENÇÃO:

1. A avaliação do trabalho é individual.
2. Em caso de cópias de trabalhos todas as equipes envolvidas receberão nota zero.
3. Todos os requisitos descritos devem ser implementados.

4. **A equipe deve informar no seu Fórum de Trabalho das Equipes no Módulo Trabalhos Práticos, até o dia 28/11/2021 quais os dois problemas que irão implementar (1 de pilha e 1 de fila).**