UNIVERSIDADE FEDERAL DE UBERLÂNDIA
Faculdade da Computação
2º Trabalho de Algoritmos e Estrutura de Dados 1
Prof. Luiz Gustavo Almeida Martins

- Os códigos deverão ser implementados somente em Linguagem C, sendo necessária a utilização das estruturas de dados conforme discutido nas aulas.
- A implementação do TAD deve contemplar todas as operações básicas da estrutura em questão, a saber: criação, verificação de vazia e cheia (quando pertinente), inserção e remoção de elementos, e apagar e esvaziar a estrutura.
- Deve-se aproveitar o conhecimento sobre a estrutura para buscar a maior eficiência das operações implementadas no código.
- A entrega do trabalho consiste no envio de um arquivo zip, contendo todos os arquivos necessários para a compilação e execução do programa, organizados em diretórios (uma pasta por questão). O envio é individual e deve ser feito até a data e horário especificados na tarefa no ambiente virtual. A integridade desse arquivo é de responsabilidade dos alunos.
- A apresentação do trabalho será individual em data e horário previamente agendados entre o professor e o grupo. O representante do grupo deve entrar em contato com o professor através do chat para o agendamento.

1) (2 ptos) Implemente um TAD Pilha de no máximo 15 **números inteiros** usando alocação **estática/sequencial** com todas as operações básicas.

Utilizando esse TAD, desenvolver um programa aplicativo que faça a conversão de um número inteiro positivo na base 10 (incluindo o zero), digitado pelo usuário, para outras bases de acordo com a opção do usuário (B-binário, O-octal e H-hexadecimal). O programa deve repetir esse processo até que o usuário digite um número negativo.

2) (8 ptos) Implemente um TAD Pilha **HETEROGÊNEA usando ponteiro VOID** que armazene um **caractere** ou um **número real,** usando alocação **dinâmica/encadeada** com todas as operações básicas.

Utilizando esse TAD, desenvolver um programa aplicativo para manipulação de expressões matemáticas envolvendo: variáveis literais de A a F; operadores de adição (+), subtração (-), divisão (/), multiplicação (*) e potenciação (^); e os delimitadores de escopo parênteses, colchetes e chaves. O programa inicia com a entrada da expressão, na forma infixa, pelo usuário. Em seguida, ele deve realizar as seguintes operações:

- **Validação de escopo:** percorrer a expressão verificando se os escopos estão sendo abertos e fechados corretamente **considerando a precedência entre os delimitadores**, ou seja, os escopos dos colchetes devem abranger os parênteses e os escopos das chaves abrangem os colchetes. Por exemplo:

- Expressão **[({A+D)/B}*F]** não é válida (ordem dos fechamentos divergem da ordem das aberturas).
- Expressão **[({A+D}/B)*F]** não é válida (precedência não é obedecida)
- Expressão **{[(A+D)/B]*F}** é válida

Ao final do processo, a operação deve retornar se a expressão é válida ou não e, no caso de erro, indicar qual o problema encontrado. O restante do processo só deve ser realizado se a expressão for válida. Caso contrário, voltar para o início (entrada da expressão).

- **Conversão da expressão:** realizar a conversão da expressão para a forma pós-fixa. Além de retornar a expressão pós-fixada, a operação também deve indicar se a conversão foi bem sucedida (1) ou não (0). Caso a conversão seja bem sucedida, o programa deve exibir a expressão pós-fixada resultante e iniciar sua avaliação. Caso contrário, uma mensagem de erro deve ser apresentada, antes de voltar para o início (entrada da expressão).

- **Avaliação da expressão:** esse módulo deve solicitar ao usuário os valores para as literais utilizadas na expressão (uma única entrada por literal), resolvê-la utilizando os valores digitados e mostrar o resultado obtido ou uma mensagem indicando algum erro encontrado (falta de operando ou de operador). Exemplos de falha:
  - **(A+D)*(^C)** não é válida (número de operandos não é adequado)
  - **(AD)/B^F** não é válida (número de operadores não é adequado)

OBS: O programa deve repetir todo o processo até que o usuário digite "FIM" para a expressão.

3) (8 ptos) Implemente um TAD Fila de carros usando alocação **estática/sequencial com uso do contador** e contemplando todas as operações básicas e as operações ***tamanho*** que retorna o tamanho da fila passada como entrada; e ***get_ini*** que retorna os dados do carro no início da fila, sem removê-lo. Para cada carro são guardados os seguintes dados: placa, tipo do serviço (A – avulso ou M – mensal) e hora de entrada.

Utilizando as operações desse TAD, desenvolver um programa aplicativo para controlar a entrada e saída de veículos em um estacionamento com as seguintes características:

- O estacionamento é composto por 5 boxes, sendo que em cada box pode ter no máximo 10 veículos estacionados.
- Para retirar um veículo do meio de um box, é necessário retirar os veículos que estão na frente e colocá-los novamente no final do box.

Além da opção de encerramento, a aplicação deverá apresentar um menu com as seguintes funcionalidades:

a) **Entrada de veículos**: onde o atendente (usuário) cadastrará os dados do veículo que está entrando no estacionamento. Esse veículo deve ser colocado no box mais vazio, visando minimizar o remanejamento de veículos na retirada. Quando os 5 boxes estiverem cheios, o veículo deve ser colocado em uma fila de espera para mais 10 carros. Acima desta capacidade, o estacionamento rejeita o veículo.

b) **Saída de veículos**: onde o atendente indica a placa do veículo que está saindo. A partir desta placa, o programa deve localizar o veículo e retirá-lo do estacionamento, fazendo a relocação necessária dos veículos dos boxes e da fila de espera (não pode ter carro na fila de espera se houver vaga em algum dos boxes). Se o carro for avulso (tipo de serviço), o sistema deve calcular o valor a ser pago. Atualmente, o estacionamento está cobrando R$ 5,00 para a primeira hora e R$ 1,50 para cada hora adicional, inclusive fração (com tolerância de 10 minutos). Ou seja, se passar 10 minutos da hora cheia, cobra-se uma nova hora.

c) **Visualização do cenário**: apresenta a situação do estacionamento, mostrando a disposição dos veículos estacionados em cada box e na fila de espera.

OBS: As horas de entrada e saída devem ser obtidas automaticamente pelo sistema a partir das funções disponibilizadas pela biblioteca time.h. Faz parte da questão a pesquisa sobre essa biblioteca e quais funções são mais adequadas ao exercício.

4) (4 ptos) Implemente um Fila de Prioridade Ascendente (FPA) de produtos perecíveis, usando alocação **dinâmica/encadeada simples e remoção por prioridade**. A estrutura do produto é formada pelos campos: código, descrição, valor e data de validade (campo usado na ordenação). Essa estrutura deve ser declarada de modo que possa ser acessada pelo programa aplicativo, facilitando a implementação. Utilizando as operações disponibilizadas no TAD, implemente um programa aplicativo que permita criar, esvaziar, apagar e imprimir uma FPA, bem como inserir e remover produtos (mostrando os dados do produto removido) em uma FPA criada. Essas ações devem ser executadas repetidamente até que o usuário solicite a saída do sistema, exceto pela criação da FPA que só pode ser executada uma única vez e antes de qualquer outra.

5) (4 ptos) Implementar o TAD Deque (SEM restrições de entrada e saída) de strings (de tamanho 20) usando alocação **estática/sequencial (com desperdício de posição)** com no máximo 15 elementos. O TAD deve contemplar todas as operações básicas da estrutura. O programa aplicativo deve permitir ao usuário realizar repetidamente as seguintes ações: criar, esvaziar, apagar e imprimir o conteúdo de um deque, bem como inserir e remover elementos (mostrando a *string* removida). A opção de criação só pode ser executada se o deque não existir (início do programa ou após ser apagado). As demais opções só podem ser executadas se o deque já tiver sido criado. O programa também deve ter uma opção para encerrar a execução.