UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
Bacharelado em Ciência da Computação e
Engenharia da Computação
INF 01203 – Estruturas de Dados
Profa. Viviane Moreira (viviane@inf.ufrgs.br)

PRIMEIRA AVALIAÇÃO

IDENTIFICAÇÃO	
Nome: André Schaidhauer Luckmann	**23/04/2013**

1. Quando se utiliza a notação infixada para representar expressões matemáticas, é necessário tomar cuidado com a precedência definida para os operadores e utilizar parênteses para lidar com esta precedência. A notação polonesa reversa é bastante comum em calculadoras, pois não é necessária a parentização, nem regras específicas de precedência dos operadores. O operador é colocado após os operandos. Compare o exemplo abaixo:

- Infixada: (5+9) * 2 + 6*5
- Notação polonesa reversa: 5 9 + 2 x 6 5 x +

a) **[valor 1,0]** Dentre os modelos lógicos de lista (fila, pilha ou deque) e físicos (lista sobre arranjo, lista simplesmente encadeada, lista duplamente encadeada), quais os mais adequados para o cálculo de uma expressão aritmética em notação polonesa reversa? Justifique sua resposta.

Pilha, com lista simplemente encadeada.
Pilha, pois iremos dividir a inserção/remoção respectivamente aos símbolos da notação polonesa reversa. Números são inseridos, e ao encontrar uma operação, removemos os ultimos dois numeros, realiza-se a operação e se insere o resultado de volta na pilha. Desta forma é possível utilizar a notação polonesa reversa de forma adequada.
Lista simplesmente encadeada pois permite que a expressão tenha tamanho dinâmico, pois a lista encadeada utiliza alocação dinâmica de memória.

b) **[valor 1,5]** Explique o funcionamento das operações de inclusão e exclusão na estrutura selecionada (no item a) de forma a possibilitar o cálculo de expressões como a do exemplo .

As operações são realizadas ao percorrer a expressão, da seguinte forma:
Números: são inseridos na pilha.
Operações: Retira-se os dois últimos números da pilha, realiza-se a operação, e o resultado é inserido na pilha novamente.
Usando o exemplo do item a):
1) 5 e 9 são inseridos na pilha;
2) Retira-se o 5 e 9, os soma, e o resultado (14) é inserido na pilha;
3) Insere-se 2 na pilha;
4) Retira-se 14 e 2 da pilha, multiplicam-se, e o resultado (28) é inserido na pilha;
5) Insere-se 6 e 5 na pilha; 6) Retira-se 6 e 5 da pilha e os multiplica, e o resultad (30) é inserido na pilha;
7) Retira-se 28 e 30 da pilha e os soma, o resultado (58) é inserido na pilha. 8) A expressão acabou, a pilha deve ter somente um número que é o resultado (58).

2. Considere as funções dos TADs Pilha e Fila abaixo:

Funções do TAD Pilha:

```
typedef int Info;

struct TPtPilha{
        Info dado;
        struct TPtPilha *elo;
};

typedef struct TPtPilha Pilha;

Pilha* InicializaPilha (Pilha *Topo);
int VaziaPilha (Pilha *Topo);
Pilha* PushPilha (Pilha *Topo, Info Dado);
int PopPilha (Pilha **Topo, Info *Dado);
Info ConsultaPilha (Pilha *Topo);
```

Funções do TAD Fila:

```
typedef int Info;
struct TPtFila{
        Info dado;
        struct TPtFila *elo;
};

typedef struct TPtFila TipoFila;

struct s_Fila{
      struct TPtFila *prim;
      struct TPtFila *ult;
};

typedef struct s_Fila Fila;


Fila* InicializaFila (Fila *PtDFila);
Info ConsultaFila (Fila *PtDFila);
int InserirFila(Fila **PtDFila, Info Dado);
int RemoverFila(Fila **PtDFila, Info *Dado);
int VaziaFila(Fila *PtDFila);
```

```
int OQueSera(Pilha **P)
{
    Fila *F;
    int x = 0;
    Info d;

    F = InicializaFila(F);

    while (!VaziaPilha(*P))
    {
        PopPilha(P,&d);
        if (d > x)
            x = d;
        InserirFila(&F,d);
    }

    while (!VaziaFila(F))
    {
         RemoverFila(&F,&d);
        *P=PushPilha(*P,d);
    }
    return x;
}
```

a) **[valor 1,5]** Explique sucintamente o que faz a função OQueSera. Como fica a estrutura P após a execução da função?

A função o que será procura pelo maior inteiro em uma pilha de inteiros, retornando o valor deste inteiro.

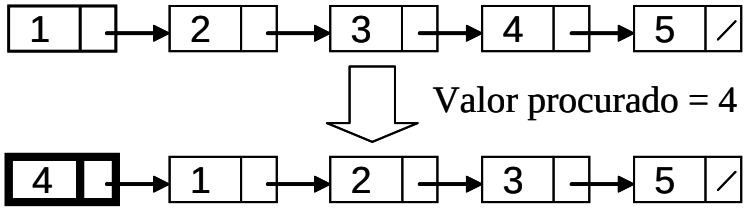
A estrutura P ficará invertida após a execução de OQueSera, pois ela tira os elementos de uma Pilha (LIFO), e as coloca em uma Fila (FIFO) temporariamente, e depois tira os elementos da fila e os coloca de volta na pilha. Dessa forma, o topo da pilha é colocado na fila, e na hora de retornar a pilha será o primeiro elemento a ser colocado, ficando na base da pilha.

b) **[valor 3,0]** Crie uma função em C que ordene uma fila usando duas pilhas como auxiliares. A função criada deve conter apenas chamadas às funções dos TADs Pilha e Fila.

Exemplos: Entrada (fila): 3 → 1 → 2 → 5 → 4
Saída (fila ordenada): 1 → 2 → 3 → 4 → 5

```c
void sortQueueWithStacks(Queue **q) {
    if (isEmptyQueue(&q)) {
        return;
    }

    Stack *s1 = initStack(), *s2 = initStack();

    Info data, removal;

    // Insert all elements from the queue to the stack 1.
    while(!isEmptyQueue(&q)) {
        dequeue(q, &data);
        s1 = push(s1, data);
    }

    // Iterate through stack 1, removing all its elements.
    while(!isEmptyStack(s1)) {
        pop(&s1, &data);

        // Enquanto a pilha s2 não for vazia e o topo de s2 for maior que o elemento de s1,
        // tira o elemento de s2 e coloca em s1 de volta, para ter a s2 ordenada.
        while(!isEmptyStack(s2) && queryStack(s2).id > data.id) {
            pop(&s2, &removal);
            push(s1, removal);
        }

        // Coloca o elemento em s2 na ordem correta.
        s2 = push(s2, data);
    }

    // Depois, é só tirar os elementos de s2 (já ordenada) e colocar na fila de volta.
    while(!isEmptyStack(s2)) {
        pop(&s2, &data);
        enqueue(q, data);
    }
}
```

3. [valor 3,0] Faça uma função em C **com comentários** que procure um valor em uma lista simplesmente encadeada. Caso o valor seja encontrado, o nodo que o contém deve ser movido para o início da lista. Veja o exemplo:




```c
bool queryNodeToHead(Node **head, int targetVal) {
    // Trata os casos em que a lista é vazia, retornando false.
    if (*head == NULL || head == NULL) {
        return false;
    }

    // Caso o primeiro elemento da lista seja o valor procurado, retorna true.
    if ((*head)->nodeInfo.id == targetVal) {
        return true;
    }

    Node *prev = *head;
    // Procura-se pelo nodo anterior ao nodo procurado.
    while (prev->nextNode != NULL && prev->nextNode->nodeInfo.id != targetVal) {
        prev = prev->nextNode;
    }

    // Caso a lista toda seja percorrida e o nodo não foi encontrado, retorna false.
    if (prev->nextNode == NULL) return false;

    // Guarda ponteiro para o nodo procurado.
    Node *targetNode = prev->nextNode;

    // Atualiza o ponteiro do nodo anterior ao procurado para o nodo próximo ao procurado.
    prev->nextNode = targetNode->nextNode;

    // Aponto o próximo nodo do nodo procurado para o começo da lista.
    targetNode->nextNode = *head;

    // Faz o primeiro elemento da lista ser o nodo procurado.
    *head = targetNode;

    // Neste caso, retorna true.
    return true;
}
```