ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

EINF MA CID

Disciplina: Estruturas de Dados e Algoritmos I-2021/2022 Prova: teste 1 (29-03-2022)

Esta prova tem a duração de **2 horas** e é **sem consulta**. Identifique TODAS as folhas de teste.

1. Considere definida em C a função `q1` :

```
int q1(int n){

    Stack S=CreateStack(n);
    for (int i=2;i<=n;i++)
        Push(i,S);
    int toReturn=Pop(S);
    while (!IsEmpty(S)) {
        toReturn *= Pop(S);
    }
    return toReturn;

}
```

(a) Qual o resultado da execução de `q1(7)`? q1(7) = 5040

(b) Qual o resultado da execução de `q1(-3)`? q1(-3) = "Stack size is too small." e exit(1). (Interrompe o programa devido ao erro)

(c) Qual a complexidade da função `q1`? O(n)

2. Considere definida em C, a função:

```
Queue q2(Queue Q){
    Stack U=CreateStack(10);
    Stack A=CreateStack(10);

    while (!IsEmptyQueue(Q)){
1       Push(Dequeue(Q),U);

        while(!IsEmptyQueue(Q)){
            if(Front(Q)<Top(U)){
                Push(Pop(U),A);
                Push(Dequeue(Q),U);
2           }
            else
                Push(Dequeue(Q),A);
        }

        while (!IsEmpty(A))
            Enqueue(Pop(A),Q);
3   }

    while(!IsEmpty(U))
        Enqueue(Pop(U),Q);
    return Q;

}
```

(a) Suponha a existência duma queue $Q = [3; 14; 1; 8; 4]$? Qual o retorno de *q2(Q)*?

(b) Qual o conteúdo da Queue Q, à saída da função *q2(Q)* e antes do retorno?

(c) Qual a complexidade da função `q2`?

1. a) q1(7):

toReturn = 7

toReturn = 7×6×5×4×3×2 = 5040

q1(7) = 5040

b) q1(-3):

Ao executar a função 'CreateStack', esta interromperá o programa e retornará a mensagem de erro "Stack size is too small", não sendo possível criar Stacks com menos de 5 elementos e, consequentemente, não será possível criar uma Stack com o valor 'MaxElements' negativo.

stackarint.c

```
...
#define MinStackSize ( 5 )
...
Stack CreateStack ( int MaxElements){
  Stack S;
if(MaxElements < MinStackSize){
  Error("Stack size is too small");
...
```

c) Apenas executa um loop 'n-1' vezes e cada operação associada a este mesmo loop terá complexidade de tempo constante em uma pilha. Logo, a complexidade total é O(n).

2. a)

0	Q=[3;14;1;8;4]	Stack U = ∅	Stack A = ∅
1	Q=[14;1;8;4;]	Stack U = [3]	
2	Q=[1;8;4; ;]		Stack A = [14]
	Q=[8;4; ; ;]	Stack U = [1|3]	Stack A = [14|3]
	Q=[4; ; ; ;]		Stack A = [14|3|8]
	Q=[; ; ; ;]		Stack A = [14|3|8|4]
3	Q=[4;8;3;14;]		Stack A = ∅
1	Q=[8;3;14; ;]	Stack U = [1|4]	
2	Q=[3;14; ; ;]		Stack A = [8]
	Q=[14; ; ; ;]	Stack U = [1|3]	Stack A = [8|4]
	Q=[; ; ; ;]		Stack A = [8|4|14]
3	Q=[14;4;8; ;]		Stack A = ∅
1	Q=[4;8; ; ;]	Stack U = [1|3|14]	
2	Q=[8; ; ; ;]	Stack U = [1|3|4]	Stack A = [14]
	Q=[; ; ; ;]		Stack A = [14|8]
3	Q=[8;14; ; ;]		Stack A = ∅
1	Q=[14; ; ; ;]	Stack U = [1|3|4|8]	
2	Q=[; ; ; ;]		Stack A = [14]
3	Q=[14; ; ; ;]		Stack A = ∅
1	Q=[; ; ; ;]	Stack U = [1|3|4|8|14]	
2			
3			
Final	Q = [14;8;4;3;1]	Stack U = ∅	Stack A = ∅

```
Queue q2(Queue Q){
    Stack U=CreateStack(10);
    Stack A=CreateStack(10);

    while (!IsEmptyQueue(Q)){
1       Push(Dequeue(Q),U);

        while(!IsEmptyQueue(Q)){
            if(Front(Q)<Top(U)){
                Push(Pop(U),A);
                Push(Dequeue(Q),U);
2           }
            else
                Push(Dequeue(Q),A);
        }

        while (!IsEmpty(A))
            Enqueue(Pop(A),Q);
3   }

    while(!IsEmpty(U))
        Enqueue(Pop(U),Q);
    return Q;

}
```

q2(Q) = [14;8;4;3;1]

b) A execução de 'q2(Q)' classifica os elementos de 'Q' e o conteúdo de 'Q' será uma versão ordenada dos elementos originais de 'Q', organizando-os por ordem decrescente. ???

c) Executa um loop dentro de outro loop.
Logo, a complexidade total é $O(n^2)$.

3. Considere definida em C a função q3 :

```
List q3(List L,int n, int m, int k){
    List X=CreateList(NULL);
    Position Px=Header(X);
    Position Pl=First(L);
    int i;
    for (i=0;i<n;i++){
        if (Pl==NULL)
            return X;
        else Pl=Advance(Pl);

    }
    while(Pl!=NULL && i<m){
       Insert(Pl->Element,X,Px);
       Px=Advance(Px);

       for (int p=0;p<k;p++){
           if(Pl!=NULL){
                Pl = Advance(Pl);
                i++;
           }
           else
               return X;
       }
   }
   return X;
}
```

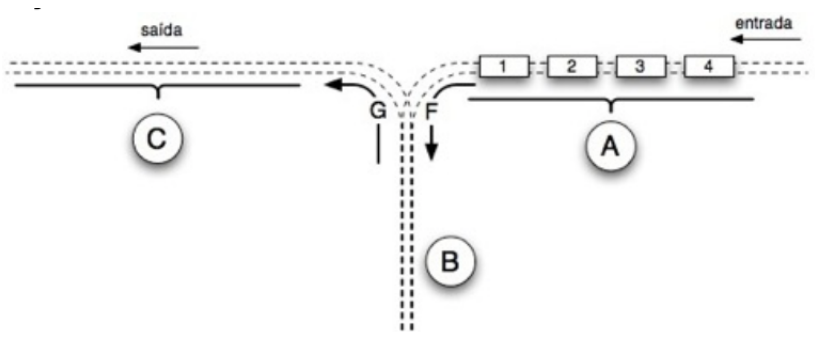
Assumindo L = [1;2;3;4;5;6;7;8;9;10] e que o tipo List é uma implementação de listas simplesmente ligadas, com "dummy Node" no início, sendo este o "header" da lista:

(a) Qual o resultado da execução de `q3(L, 0, 10, 5)`?

(b) Qual o resultado da execução de `q3(2,7,1)`?

(c) Qual a complexidade da função q3?

4. Suponha que quatro carruagens numeradas de 1 a 4 estão posicionadas à entrada duma estação ferroviária (ver figura) Suponha possíveis as seguintes acções:



- *Entrada(carruagem)*, que permite a entrada duma carruagem na estação(pela zona A)
- *Saida()*, que permite que uma carruagem saia da estação (pela zona C)
- *F()* que permite que uma carruagem passe da zona A para a zona B
- *G()* que permite que uma carruagem passe da zona B para a zona C

(a) A sequência de acções: Entrada(1), Entrada(2),Entrada(3), Entrada(4), F(), F(), F(), F(), G(), G(), G(), G(), Saida(), Saida(), Saida(), Saida(), permitirá retirar da estação as carruagens pela ordem 4,3,2,1. Assumido que estão posicionadas na estação as carruagens na zona de entrada, na sequência (3,2,1), que sequência de operações deverá realizar para retirar da estação a carruagem 2?

(b) Apresente um programa em C que modele o comportamento da estação, exibindo a codificação das operações acima descritas. Não pode usar arrays.

3. a) Lista vazia??

b)

c) $O(n^2)$??

4. a)

b)

```c
#include <stdio.h>
#include "stackarint.h"
#include "queue.h"

struct estacaoRecord{
    char *nome;
    Queue zonaA;
    Stack zonaB;
    Queue zonaC;
};
typedef struct estacaoRecord Estacao;

void printEstacao(Estacao e){
    printf("%s\n",e.nome);
    printf("Zona A=");
    PrintQueue(e.zonaA);
    printf("Zona B=");
    PrintStackInt(e.zonaB);
    printf("Zona C=");
    PrintQueue(e.zonaC);
    printf("\n");
}
void entrada(int c,Estacao e){
    Enqueue(c,e.zonaA);
}
void F(Estacao e){
    Push(Dequeue(e.zonaA), e.zonaB);
}
void G(Estacao e){
    Enqueue(Pop(e.zonaB), e.zonaC);
}
int saida(Estacao e){
    return Dequeue(e.zonaC);
}
```

```c
}

int main(int argc, const char * argv[]) {
    Estacao nova;
    nova.nome="Estação Nova";
    nova.zonaA=CreateQueue(10);
    nova.zonaB=CreateStack(10);
    nova.zonaC=CreateQueue(10);
    printEstacao(nova);
    entrada(1,nova);
    entrada(2,nova);
    entrada(3,nova);
    entrada(4,nova);
    printEstacao(nova);
    F(nova);
    printEstacao(nova);
    G(nova);
    printEstacao(nova);
    int x=saida(nova);
    printf("Sai comboio %d\n", x);
    printEstacao(nova);

    Estacao velha;
    velha.nome="Estação Velha";
    velha.zonaA=CreateQueue(10);
    velha.zonaB=CreateStack(10);
    velha.zonaC=CreateQueue(10);

    printEstacao(velha);
    entrada(1,velha);
    entrada(2,velha);
    printEstacao(velha);
    printEstacao(nova);
    return 0;
}
```

3. Considere o algoritmo de conversão infix-postfix dado nas aulas teóricas e implementado nas aulas práticas. Considere que `input` corresponde à string lida e a conversão da expressão lida é armazenada na variável `output`. Assuma também que o input é analisado token a token, sendo um token a substring obtida de input, entre espaços.
Sendo `input`=(2 + -5) / (30 / 15) + 5 * 8

   (a) Qual o 4º token lido?

   (b) Qual o output da conversão para postfix da expressão da variável input, quando o token lido for a primeira "/"

   (c) Quando o token lido for o último ")", qual o conteúdo da stack? Desenhe a stack, indicando o seu topo.

   (d) Qual a conversão para postfix da expressão lida?

   (e) Usando a expressão obtida na alínea anterior, desenhe a stack de avaliação da expressão em postfix, quando for lida a segunda divisão "/"

4. Uma *Deque* é uma estrutura de dados que consiste numa lista de itens, e permite as seguintes operações:

   - `push(x, d)`: que insere o item x, no início da *deque*
   - `pop(d)`: que retira da *deque* o elemento que está no início e devolve-o
   - `inject(x, d)`: que insere o item x, no fim da *deque*
   - `eject(d)`: que retira da *deque* o elemento que está no fim e devolve-o

   (a) Apresente o tipo composto *dequeStruct*, assumindo que tal como no caso da implementação das Stacks e das Queues se tem a seguinte definição `typedef struct dequeStruct *Deque;`

   (b) Apresente funções que realizem as operações descritas para a *deque* em tempo constante

3 a) (2 + -5) ...

4: Token

4: Token => -5

b) (2 + -5) /

2 -5 + /

c)

Stack Vazia???

d)

1	2	3	4	5	6	7				
2	-5	+	30	15	/	/	5	8	*	+

e)

4. a)

b)