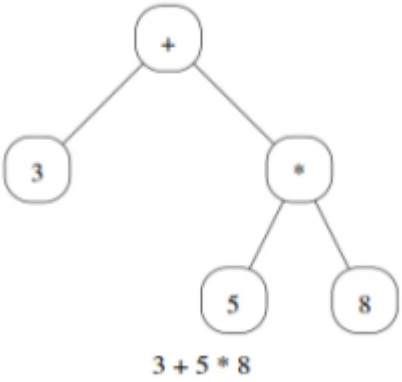
Lista de Exercicios

1. Dado o código que verifica se um dado número esta na árvore, faça o teste de mesa da função.

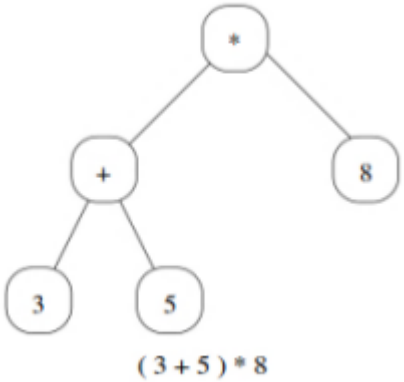
```
// verifica se a ocorrencia de um caracter na arvore
int belongTree(Tree* t, int c){
	if(isEmpty(t)){
		return 0;
	}else{
		return t->info == c || belongTree(t->left,c) || belongTree(t->right,c);
	}
}
```

*** lembrando que o conceito de teste de mesa é executa as instruções do algortimo com objetivo de validar se a lógica do código faz de fato o que é pedido.

2. Considere arvores binarias que representam expressões aritméticas (composta por operandos compostos por um unico algarismo, opera ações de +, -, * e / e parênteses) como as apresentadas abaixo.



3 + 5 * 8



(3 + 5) * 8

Escreva um algoritmo que receba um tipo de dado abstrato representando tais árvores e retorne uma string corresponde a vers~ao pré-fixa, infixa e pós-fixa da expressão.

3. A soma dos conteudos de todos os nós em uma árvore binária, considerando que cada no contém um inteiro;

4. Faça o teste de mesa do código abaixo, que verifica a quantidade de nós que existe na árvore

```
// número de nós da arvore.
int numero_nos(Tree* t){
	if(isEmpty(t->left) && isEmpty(t->right)){
		return 1;
	}else{
		return 1 + numero_nos(t->left) + numero_nos(t->right);
	}
}
```

5. O código verifica a altura de uma arvore, faça o teste de mesa deste trecho.

```
//altura da arvore = maior caminho percorrido para chegar na folha
int altura(Tree* t){
	if(isEmpty(t)){
		return 0;
	}
	if(isEmpty(t->left) && isEmpty(t->right)){
		return 0;
	}else{
		int tamR = altura(t->right);
		int tamL = altura(t->left);
		if(tamR > tamL){
			return tamR + 1;
		}else{
			return tamL + 1;
		}
	}
}
```

6. Faça o teste de mesa do código abaixo que verifica o grau de uma árvore.

```
//grau da arvore = número de filhos de um nó
int grau(Tree* t){
	if(isEmpty(t->left) && isEmpty(t->right)){
		return 0;
	}else if( (isEmpty(t->left) && !isEmpty(t->right)) || !isEmpty(t->left) && isEmpty(t->right)){
		return 1;
	}else if(!isEmpty(t->left) && !isEmpty(t->right)){
		return 2;
	}
}
```