```
function apply_operation(int a, int b, char operator)
	if operator is '+' then
		return a + b
	if operator is '-' then
		return a - b
	if operator is '*' then
		return a * b
	if operator is '/' then
		return a / b

function evaluate(char[] expression)
	value_stack = init_stack()
	ops_stack = init_stack()
	i ← 0
	while i < expression.length() do
		curr ← expression[i]
		if curr is digit then
			value_stack.push(expression[i])

		else if curr is '(' then
			ops_stack.push(expression[i])

		else if curr is ')' then
			if value_stack has less than 2 values or top of ops_stack is '(' then
				return "NotWellFormed"
			while ops_stack is non-empty and top of ops_stack is not '(' do
				v2 ← value_stack.pop()
				v1 ← value_stack.pop()
				operator = ops_stack.pop()
				result = apply_operation(v1, v2, operator)
				value_stack.push(result)
			if top of ops_stack is not '(' then
				return "NotWellFormed"
			else
				ops_stack.pop()

		else if curr is '+' or '-' or '*' or '/' then
			if ops_stack is empty then
				return "NotWellFormed"
			while ops_stack is non-empty and value_stack has 2 or more values and top of
	ops_stack has higher or equal precedence than curr, do
				v2 ← value_stack.pop()
				v1 ← value_stack.pop()
				operator = ops_stack.pop()
				result = apply_operation(v1, v2, operator)
				value_stack.push(result)
			ops_stack.push(curr)

		else
			return "NotWellFormed"

		i ← i + 1

	if only one of ops_stack and value_stack is empty do
		return "NotWellFormed"

	while ops_stack is non-empty do
		v2 ← value_stack.pop()
		v1 ← value_stack.pop()
		operator = ops_stack.pop()
		result = apply_operation(v1, v2, operator)
		value_stack.push(result)

	return value_stack.pop()
```