```
/*
CS610-851
Programming Assignment 2
Author: Haard Shah
Due date: November 4, 2018
*/

#include <string>
#include <iostream>
#include <cctype>

#define DONE 'd'

using namespace std;

// #include "ParseNode.h"
string prob = "";

char getNext(bool peek=0) {
	static int count = 0;

	while (count < prob.size() && isspace(prob.at(count))) {count++;}

	if (peek) {
		if (count < prob.size()) {
			return prob.at(count);
		}
		else {
			return DONE;
		}
	}
	if (count < prob.size()) {
		count++;
		return prob.at(count-1);
	}
	return DONE; // done
}

class Node {
public:
	Node *left;
	Node *right;
	char curr;
	Node(Node *left=0, Node *right=0, char curr=DONE) : left(left), right(right), curr(curr){}
```

```
};

Node *Expr();
Node *Term();
Node *Factor();
Node *Dig();

// expr := term | term + expr | term - expr
Node *Expr() {
	Node *term = Term();
	if (term == 0) {
		return 0;
	}
	char op = getNext(1);
	if (op != '+' && op != '-') {
		return term;
	} else {
		getNext(); //read the + or - operator
		Node *expr = Expr();
		if (expr) {
			return new Node(term, expr, op);
		}
		else {
			cout << "expression should follow <term> " << op << "..." << endl;
			return 0;
		}
	}

	cout << "unexpected behavior in Expr()" << endl;
	return 0; // shouldn't get here;
}

// term := factor | factor * term | factor / term
Node *Term() {
	Node *factor = Factor();
	if (factor == 0) {
		return 0;
	}
	char op = getNext(1);
	if (op != '*' && op != '/') {
		return factor;
	} else {
		getNext(); //read the * or / operator
		Node *term = Term();
```

```
                if (term) {
                        return new Node(factor, term, op);
                } else {
                        cout << "term should follow <factor> " << op << "..." << endl;
                        return 0;
                }
        }

        cout << "unexpected behavior in Term()" << endl;
        return 0; // shouldn't get here;
}

// factor := dig | LBR expr RBR
Node *Factor() {
        char lbr = getNext(1);
        if (lbr == '(') {
                getNext(); // read the LBR
                Node *expr = Expr();
                if (expr) {
                        char rbr = getNext(); // don't need to peek should be rbr
                        if (rbr == ')') {
                                return expr;
                        }
                        else {
                                cout << "expecting RBR" << endl;
                                return 0;
                        }
                }
                else {
                        cout << "expr should follow LBR..." << endl;
                        return 0;
                }
        }
        else {
                Node *dig = Dig();
                if (dig) {
                        return dig;
                }
                else {
                        cout << "not sure if this is normal. Inside Factor()" << endl;
                        return 0;
                }
        }
```

```
	cout << "unexpected behavior in Factor()" << endl;
	return 0; // shouldn't get here;
}

// dig := 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
Node *Dig() {
	char dig = getNext(); // don't need to peek because expecting char
	if (isdigit(dig)) {
		return new Node(0,0,dig);
	} else {
		cout << "not sure if this is expected inside Dig()" << endl;
		return 0;
	}

	cout << "unexpected behavior in Dig()" << endl;
	return 0; // shouldn't get here;
}

// void inorder(Node *head) {
//	if (!head) return;
//	if ((!head->left) && (!head->right)) {
//		cout << head->curr;
//		return;
//	}
//	cout << "(";
//	inorder(head->left);
//	cout << head->curr;
//	inorder(head->right);
//	cout << ")";
// }

void preorder(Node *head) {
	if (!head) return;
	cout << head->curr << ", ";
	preorder(head->left);
	preorder(head->right);
}

void inorder(Node *head) {
	if (!head) return;
	inorder(head->left);
	cout << head->curr << ", ";
	inorder(head->right);
}
```

```
void postorder(Node *head) {
	if (!head) return;
	postorder(head->left);
	postorder(head->right);
	cout << head->curr << ", ";
}

void printTraversals(Node *expr) {
	cout << "Inorder Traversal:" << endl;
	inorder(expr);
	cout << endl;

	cout << "Preorder Traversal:" << endl;
	preorder(expr);
	cout << endl;

	cout << "Postorder Traversal:" << endl;
	postorder(expr);
	cout << endl;
}

double evaluate(Node *head) {
	if (!head) {
		return 0;
	}
	if ((!head->left) && (!head->right)) {
		return (double)(head->curr - '0');
	}

	char op = head->curr;
	double lval = evaluate(head->left);
	double rval = evaluate(head->right);

	// cout << lval << op << rval << endl;

	if (op == '+') {
		return (double)(lval + rval);
	}
	else if(op == '-') {
		return (double)(lval - rval);
	}
	else if(op == '*') {
		return (double)(lval * rval);
```

```
	}
	else if(op == '/') {
		if (rval == 0) {
			cout << "divide by zero: " << lval << "/" << rval << "..." << endl;
			return 0;
		}
		return (double)(lval / rval);
	}

	cout << "unexpected behavior inside evaluate()." << endl;
	return 0; //shouldn't get here
}

int main(int argc, char *argv[]) {
	// prob = "((((3+1)*3)/ ((9-5)+2))-((3*(7-4))+6))";
	// prob = "4+(6*(4/1))";
	if (argc != 2) {
		printf("Usage: %s <expression>\n", argv[0]);
		return 1;
	}
	prob = argv[1];

	Node *expr = Expr();
	// inorder(expr); // prints original expression

	// print expression tree
	printTraversals(expr);

	cout << "Result: " << evaluate(expr) << endl;
}
```