```c
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "astg.h"

void getBrackets(char *in, int bpos, char *out){
        // gets the content of the brackets that open at bpos
        if (in[0] != '(' && in[0] != '[')
                out = NULL;

        char *input = malloc(strlen(in) + 1); // cpy in for mem safety
        char **Pinput = input;
        memcpy(input, in, strlen(in) + 1);

        int i = 0;
        while (i != bpos){
                input++;
                i++;
        }
        i = 0;
        int depth = 0;
        while (input[0] != '\0'){ // loop through input
                out[i] = input[0];
                if (input[0] == '(' || input[0] == '[')
                        depth++;
                if (input[0] == ')' || input[0] == ']')
                        depth--;
                if (depth == 0){
                        i++;
                        break;
                }

                input++;
                i++;
        }

        out[i] = '\0';
        free(Pinput);
}

int getContents(char *brackets){ // get all the content in brackets
        int i = 0;
        char *num = malloc(strlen(brackets));
        while (brackets[0] != '\0'){
                if (brackets[0] != '(' && brackets[0] != '['){
                        num[i] = brackets[0];
                        i++;
                }
                brackets++;
        }
        num[i] = '\0';
        int out = atoi(num);
        free(num);
        return out;
}

ast_node *genAst(char *expression){
        ast_node *out = malloc(sizeof(ast_node));

        /* take the expression
         * get the first operation from it
         * get the first number from after the expression
         * if that number is another expression(
         *      grab everything inside its braket and then call this function on it
         * )
         * get the second number from the expression
         * if that number is another expression(
         *      grab everything inside its braket and then call this function on it
         * )
         */

        int i = 0, j = 0;
        int currentTokenVal = -1;

        char *currentToken = malloc(strlen(expression)+1);
        while (expression[i] != '\0'){
                if (expression[i] == '(' || expression[i] == '['){
                        getBrackets(expression, i, currentToken);
                        currentTokenVal++;

                        int depth = 0;

                        if (i != 0){
                                while (expression[i] != '\0'){
                                        if (expression[i] == '(' || expression[i] == '[')
                                                depth++;
                                        if (expression[i] == ')' || expression[i] == ']')
                                                depth--;
                                        if (depth == 0){
                                                break;
                                        }
                                        i++;
                                }
                        }

                        if (currentTokenVal == 0){
                                switch (currentToken[1]) {
                                        case '+':
                                                out->operator= ADD;
                                                break;
                                        case '-':
                                                out->operation = SUB;
                                                break;
                                        case '*':
                                                out->operation = MUL;
                                                break;
                                        case '/':
                                                out->operation = DIV;
                                                break;
                                }
                        }

                        if (currentTokenVal == 1){
                                if (currentToken[0] == '(')
                                        out->left = genAst(currentToken);
                                else {
                                        out->realLeft = getContents(currentToken);
                                        out->left = NULL;
                                }
                        } else if (currentTokenVal == 2){
                                if (currentToken[0] == '(')
                                        out->right= genAst(currentToken);
                                else {
                                        out->realRight = getContents(currentToken);
                                        out->right = NULL;
                                }
                        }
                }
        i++;
        }
        free(currentToken);

        return out;
}

int main(int argc, char **argv){
        if (argc < 2) // die if no argument given
                exit(1);

        ast_node *head = genAst(argv[1]);

        printf("%d\n", exec(head));

        freeAst(head);
}
```