- CS

```
// CPP program to evaluate a given
// expression where tokens are
// separated by space.
#include <bits/stdc++.h>
using namespace std;

// Function to find precedence of
// operators.
int precedence(char op){
    if(op == '+'||op == '-')
    return 1;
    if(op == '*'||op == '/')
    return 2;
    return 0;
}

// Function to perform arithmetic operations.
int applyOp(int a, int b, char op){
    switch(op){
        case '+': return a + b;
        case '-': return a - b;
        case '*': return a * b;
        case '/': return a / b;
    }
}

// Function that returns value of
// expression after evaluation.
int evaluate(string tokens){
    int i;

    // stack to store integer values.
    stack <int> values;

    // stack to store operators.
    stack <char> ops;

    for(i = 0; i < tokens.length(); i++){

        // Current token is a whitespace,
        // skip it.
        if(tokens[i] == ' ')
            continue;

        // Current token is an opening
        // brace, push it to 'ops'
        else if(tokens[i] == '('){
            ops.push(tokens[i]);
        }

        // Current token is a number, push
        // it to stack for numbers.
        else if(isdigit(tokens[i])){
        /*  int val = 0;

            // There may be more than one
            // digits in number.
            while(i < tokens.length() &&
                        isdigit(tokens[i]))
            {
                val = (val*10) + (tokens[i]-'0');
                i++;
            }

            values.push(val);*/
            values.push(tokens[i]);

            // right now the i points to
            // the character next to the digit,
            // since the for loop also increases
            // the i, we would skip one
            // token position; we need to
            // decrease the value of i by 1 to
            // correct the offset.
            i--;
        }

        // Closing brace encountered, solve
        // entire brace.
        else if(tokens[i] == ')')
        {
            while(!ops.empty() && ops.top() != '(')
            {
                int val2 = values.top();
                values.pop();

                int val1 = values.top();
                values.pop();

                char op = ops.top();
                ops.pop();

                values.push(applyOp(val1, val2, op));
            }

            // pop opening brace.
            if(!ops.empty())
            ops.pop();
        }

        // Current token is an operator.
        else
        {
            // While top of 'ops' has same or greater
            // precedence to current token, which
            // is an operator. Apply operator on top
            // of 'ops' to top two elements in values stack.
            while(!ops.empty() && precedence(ops.top())
                                >= precedence(tokens[i])){
                int val2 = values.top();
                values.pop();

                int val1 = values.top();
                values.pop();

                char op = ops.top();
                ops.pop();

                values.push(applyOp(val1, val2, op));
            }

            // Push current token to 'ops'.
            ops.push(tokens[i]);
        }
    }

    // Entire expression has been parsed at this
    // point, apply remaining ops to remaining
    // values.
    while(!ops.empty()){
        int val2 = values.top();
        values.pop();

        int val1 = values.top();
        values.pop();

        char op = ops.top();
        ops.pop();

        values.push(applyOp(val1, val2, op));
    }

    // Top of 'values' contains result, return it.
    return values.top();
}

int main() {
    cout << evaluate("10 + 2 * 6") << "\n";
    cout << evaluate("100 * 2 + 12") << "\n";
    cout << evaluate("100 * ( 2 + 12 )") << "\n";
    cout << evaluate("100 * ( 2 + 12 ) / 14");
    return 0;
}

// This code is contributed by Nikhil jindal.

```