```cpp
// 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++){

```

```
40:         // Current token is a whitespace,
41:         // skip it.
42:         if(tokens[i] == ' ')
43:             continue;
44:
45:         // Current token is an opening
46:         // brace, push it to 'ops'
47:         else if(tokens[i] == '('){
48:             ops.push(tokens[i]);
49:         }
50:
51:         // Current token is a number, push
52:         // it to stack for numbers.
53:         else if(isdigit(tokens[i])){
54:         /*  int val = 0;
55:
56:             // There may be more than one
57:             // digits in number.
58:             while(i < tokens.length() &&
59:                         isdigit(tokens[i]))
60:             {
61:                 val = (val*10) + (tokens[i]-'0');
62:                 i++;
63:             }
64:
65:             values.push(val);*/
66:             values.push(tokens[i]);
67:
68:             // right now the i points to
69:             // the character next to the digit,
70:             // since the for loop also increases
71:             // the i, we would skip one
72:             // token position; we need to
73:             // decrease the value of i by 1 to
74:             // correct the offset.
75:             i--;
76:         }
77:
78:         // Closing brace encountered, solve
```

```
79:         // entire brace.
80:         else if(tokens[i] == ')')
81:         {
82:             while(!ops.empty() && ops.top() != '(')
83:             {
84:                 int val2 = values.top();
85:                 values.pop();
86:
87:                 int val1 = values.top();
88:                 values.pop();
89:
90:                 char op = ops.top();
91:                 ops.pop();
92:
93:                 values.push(applyOp(val1, val2, op));
94:             }
95:
96:             // pop opening brace.
97:             if(!ops.empty())
98:             ops.pop();
99:         }
100:
101:        // Current token is an operator.
102:        else
103:        {
104:            // While top of 'ops' has same or greater
105:            // precedence to current token, which
106:            // is an operator. Apply operator on top
107:            // of 'ops' to top two elements in values stack.
108:            while(!ops.empty() && precedence(ops.top())
109:                                  >= precedence(tokens[i])){
110:                int val2 = values.top();
111:                values.pop();
112:
113:                int val1 = values.top();
114:                values.pop();
115:
116:                char op = ops.top();
117:                ops.pop();
```

```
118:
119:                 values.push(applyOp(val1, val2, op));
120:             }
121:
122:             // Push current token to 'ops'.
123:             ops.push(tokens[i]);
124:         }
125:     }
126:
127:     // Entire expression has been parsed at this
128:     // point, apply remaining ops to remaining
129:     // values.
130:     while(!ops.empty()){
131:         int val2 = values.top();
132:         values.pop();
133:
134:         int val1 = values.top();
135:         values.pop();
136:
137:         char op = ops.top();
138:         ops.pop();
139:
140:         values.push(applyOp(val1, val2, op));
141:     }
142:
143:     // Top of 'values' contains result, return it.
144:     return values.top();
145: }
146:
147: int main() {
148:     cout << evaluate("10 + 2 * 6") << "\n";
149:     cout << evaluate("100 * 2 + 12") << "\n";
150:     cout << evaluate("100 * ( 2 + 12 )") << "\n";
151:     cout << evaluate("100 * ( 2 + 12 ) / 14");
152:     return 0;
153: }
154:
155: // This code is contributed by Nikhil jindal.
156:
```