Experiment -9

Aim: To implement syntax-directed translation.

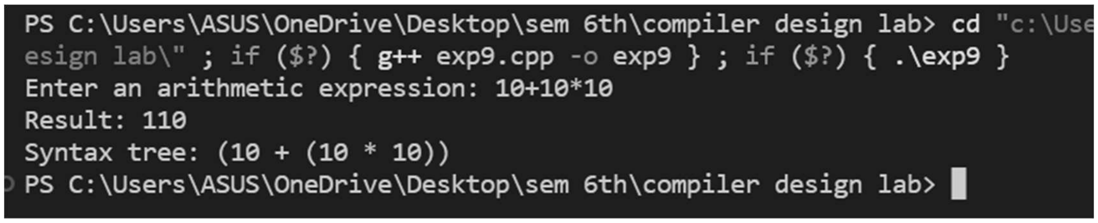
Code:

```
#include <iostream>
#include <string>
#include <sstream>
#include <vector>
using namespace std;

// A class to represent a node in the syntax tree
class TreeNode
{
public:
   virtual ~TreeNode() {}
   virtual int evaluate() const = 0;
   virtual string translate() const = 0;
};

// A class to represent a non-terminal node in the syntax tree
class NonTerminalNode : public TreeNode
{
public:
   virtual ~NonTerminalNode() {}
   virtual int evaluate() const = 0;
   virtual string translate() const = 0;
};

// A class to represent a terminal node in the syntax tree
class TerminalNode : public TreeNode
{
public:
   virtual ~TerminalNode() {}
   virtual int evaluate() const = 0;
   virtual string translate() const = 0;
};

// A class to represent an addition operation
class AddNode : public NonTerminalNode
{
public:
   AddNode(TreeNode *left, TreeNode *right) : left(left), right(right) {}
   virtual ~AddNode()
   {
      delete left;
```

```
        delete right;
    }
    virtual int evaluate() const { return left->evaluate() + right->evaluate(); }
    virtual string translate() const { return "(" + left->translate() + " + " + right-
>translate() + ")"; }

private:
    TreeNode *left;
    TreeNode *right;
};

// A class to represent a multiplication operation
class MultiplyNode : public NonTerminalNode
{
public:
    MultiplyNode(TreeNode *left, TreeNode *right) : left(left), right(right) {}
    virtual ~MultiplyNode()
    {
        delete left;
        delete right;
    }
    virtual int evaluate() const { return left->evaluate() * right->evaluate(); }
    virtual string translate() const { return "(" + left->translate() + " * " + right-
>translate() + ")"; }

private:
    TreeNode *left;
    TreeNode *right;
};

// A class to represent a number
class NumberNode : public TerminalNode
{
public:
    NumberNode(int value) : value(value) {}
    virtual int evaluate() const { return value; }
    virtual string translate() const
    {
        stringstream ss;
        ss << value;
        return ss.str();
    }

private:
    int value;
};
```

```
// A class to represent a parser for the given grammar
class Parser
{
public:
    Parser(const string &input) : input(input), pos(0) {}
    TreeNode *parseE()
    {
        TreeNode *left = parseT();
        while (pos < input.size() && input[pos] == '+')
        {
            pos++;
            TreeNode *right = parseT();
            left = new AddNode(left, right);
        }
        return left;
    }
    TreeNode *parseT()
    {
        TreeNode *left = parseF();
        while (pos < input.size() && input[pos] == '*')
        {
            pos++;
            TreeNode *right = parseF();
            left = new MultiplyNode(left, right);
        }
        return left;
    }
    TreeNode *parseF()
    {
        if (input[pos] == '(')
        {
            pos++;
            TreeNode *result = parseE();
            pos++;
            return result;
        }
        else
        {
            int num = 0;
            while (pos < input.size() && isdigit(input[pos]))
            {
                num = num * 10 + (input[pos] - '0');
                pos++;
            }
            return new NumberNode(num);
        }
    }

private:
    const string &input;
    size_t pos;
};

int main()
{
    string input;
    cout << "Enter an arithmetic expression: ";
    getline(cin, input);
    Parser parser(input);
    TreeNode *root = parser.parseE();
    // Evaluate and print the result
    cout << "Result: " << root->evaluate() << endl;

    // Translate and print the syntax tree
    cout << "Syntax tree: " << root->translate() << endl;

    // Clean up memory
    delete root;

    return 0;
}
```

Output:



```
PS C:\Users\ASUS\OneDrive\Desktop\sem 6th\compiler design lab> cd "c:\Use
esign lab\" ; if ($?) { g++ exp9.cpp -o exp9 } ; if ($?) { .\exp9 }
Enter an arithmetic expression: 10+10*10
Result: 110
Syntax tree: (10 + (10 * 10))
PS C:\Users\ASUS\OneDrive\Desktop\sem 6th\compiler design lab>
```