```
// Bee Cha
// CSCI 117
// This program is an interpreter for a mini-language. You initiate the language
// first by typing 'program'. After that, you will declare your variables by typing
// var a, b, c, d;
// or as many variable as needed. Currently, the program can only store 26 variables.
// Each variable is stored as an ID in a struc with a corresponding value to it. When
// you are done declaring your variables, type 'begin' to start the statements. The
// statements that are avaiable is the input, output, and end statement. For the input
// I've made it so it can take a value from the keyboard and store it into the corresponding
// ID. I.E., intput a 9; will store the value 9 into variable a. For now, it can only take a
// single digit from the keyboard. BUT, it can take an expression as an input.
// input b (2+3)*7+2^(1+2); will store the value 43 into the variable b.
// The output statement can output a variable or an expression.
// I.E., output 5; will print 5. output 3*(5+2); will print 21. And output a; will output
// the value stored in a. For the expressions, I had to include a semi-colon.
// By typing 'end', it will stop the program and exit.

//-----Source Code-----//

#include <iostream>
#include <fstream>
#include <math.h>
#include <string>

using namespace std;

void declarations(), declaration(), statements(), statement(string);
void input_st(), output_st(), assign_st(string);
int Exp(), Term(), Exp2(int), Term2(int), Fact(), Pwr();
ifstream inputFile;

struct symbol
{
	char id = '_'; // (string id works too but use char for now)
	int value = 0;
};

// table to hold 26 alphabet identifiers for now
symbol table[26];
int index = 0;

int main(int argc, char* argv[])
{
	/*
	string fileName = "input4.txt";
	inputFile.open(fileName);

	if (!inputFile.is_open())
	{
		cerr << "Unable to open file " << fileName << std::endl;
	}

	cout << "Result = " << Exp() << endl;

	inputFile.close();
	*/

	string p;
	cin >> p;
```

```
	// If the string input is 'program' then we can start running all the declarations
	// such as variables, functions, etc.
	// Otherwise throw an error
	if (p.compare("program") == 0)
	{
		declarations();

		//having input 'begin' will continue from here
		statements();
	}
	else
	{
		cout << "Syntax Error. Input \'" << p << "\' doesn't
match \'programn\'\nExiting...\n";
		cout << "Type \'end\' to exit\n";
	}

	cout << "End of Program\n";

	return 0;
}

// Declarations() is where we will declare all the variables for the program
// Example: var a, b, c;
// Keep a recursive call to Declarations() to loop it back until we don't
// need any more declarations
void declarations()
{
	string s;
	cin >> s;
	if (s.compare("begin") == 0)
	{
		return;
	}
	else if (s.compare("var") == 0)
	{
		declaration();
	}
	else
	{
		cout << "ERROR: Unknown Identifier\n";
	}

	declarations();
}

// Read the ID, store it into a struct table with an index and ID
// with the ID referencing to the value. It will later write to the
// value when calling the input_st() function when searching for
// that ID. Keep reading variable IDs until hitting a semi-colon (;)
// *NOTE* skip the comma (,) character if it's in the input stream
// Assume no duplicate ID is entered
void declaration()
{
	char c;
	cin >> c;
	while (c != ';')
	{
		if (c == ',') // Need to skip the comma and grab the next char
```

```
                cin >> c;
            table[index].id = c;
            index++;
            cin >> c;
        }

}

// This is the body of the program. The simplest of the program
// is if the program is empty. Or in this case, if the program
// only has 'end'. This will exit the program. The only three
// cases for statements are the input, output, and assignemnt
// functions. Anything else should throw out a semantic error.
void statements()
{
        string s;
        cin >> s;
        if (s.compare("end") == 0)
        {
            return;
        }
        else
        {
            statement(s);
        }

        statements();

}


void statement(string s)
{
        if (s == "end")
        {
            return;
        }
        else if (s == "input")
        {
            input_st();
        }
        else if (s == "output")
        {
            output_st();
        }
        else
        {
            // Assume assignment statements are syntax correct
            assign_st(s);
        }

}

// Read a character. Then check the table for the corresponding
// ID that matches the character. If the ID exists, then read
// a digit input and set that value. If it doesn't exist, then
// throw off a semantic error (variable does not exist)
void input_st()
{
        char c;
```

```
	cin >> c;

	if (c == ';')
		cin >> c;

	for (int i = 0; i < 26; i++)
	{
		if (table[i].id == c)
		{
			table[i].value = Exp();
			return;
		}

	}

}

// Read a character. Search the table for the matching ID.
// If matching ID is found, then output the corresponding value.
// If the ID is not found, then output a semantic error that
// the variable was not declared.
void output_st()
{
	char c;
	cin >> c;

	if (c == ';')
		cin >> c;

	for (int i = 0; i < 26; i++)
	{
		if (c == table[i].id)
		{
			cout << table[i].value << endl;
			return;
		}
	}

	cin.putback(c);
	cout << Exp() << endl;
}


// s[0] is the ID. We have to check if the variable ID exist first. And s[1]
// or s[2] will be the '=' operator, but if s[1]
// is a whitespace, keep skipping the index until you reach the '=' operator.
// After reaching the equal operator, we can call the expression() function
// to get the value of the expression then assign it to the corresponding ID.
// Should I assume no 'whitespace' is to be entered?
// I could throw in a syntax error when 'whitespace' is used

// Example:
// s = "b = (1+2)*3;"
// s[0] = 'b'
// s[1] = ' '
// s[2] = '='
// s[3] = ' '
// s[4+] = Exp()

// Or I can use 'whitespace' as the delimiter for input stream
```

```
// s = "b = (1+2)*3;
// In this case, the next char in the stream is '='
void assign_st(string s)
{
	char a;
	char id = s[0]; // This will grab the ID identifier from the string
	if (id == ';') // A semi-colon to end the assignment statement
		return;
	cin >> a; // This will grab the '='
	// Now just grab the expression on the right-hand side
	// and set that value to the corresponding ID in the table
	if (a == '=')
	{
		for (int i = 0; i < 26; i++)
		{
			if (table[i].id == id)
			{
				table[i].value = Exp();
			}
		}
	}

}



//----------Lab 2 defining lower part of the grammar----------//
// Changed the input from keyboard instead of from file
// I had to include the semi-colon as a delimiter for some
// of the expressions.
int Exp()
{
	return Exp2(Term());
}

int Term()
{
	return Term2(Pwr());
}

int Exp2(int inp)
{
	int result = inp;
	char a;
	if (cin >> a)
	{
		if (a == '+')
		{
			result = Exp2(result + Term());
		}
		else if (a == '-')
		{
			result = Exp2(result - Term());
		}
		else if (a == ')' || a == ';')
			cin.putback(a);
	}

	return result;
```

```
}

int Term2(int inp)
{
	int result = inp;
	char a;
	if (cin >> a)
	{
		if (a == '*')
			result = Term2(result * Pwr());
		else if (a == '/')
			result = Term2(result / Pwr());
		else if (a == '+' || a == '-' || a == ')' || a == ';')
			cin.putback(a);
	}

	return result;
}

int Pwr() {
	int p = Fact();
	char a;
	if (cin >> a) {
		if (a == '^') {
			p = pow(p, Pwr());
		}
		if (a == '+' || a == '-' || a == '*' || a == '/' || a == ')' || a == ';')
			cin.putback(a);
	}

	return p;
}

int Fact()
{
	char a;
	cin >> a;
	if (a == '(')
	{
		int exp = Exp();
		cin >> a;
		if (a == ')')
		{
			return exp;
		}
	}

	return (a - '0');
}
```