

University of the Philippines Cebu

Lahug, Cebu City

# **IDE-WOW**

**(MyHL IDE)**

Submitted by:

Cabarrubias, Aldwyn F.

Manlosa, Shairine M.

Pepito, Lindy Lou C.

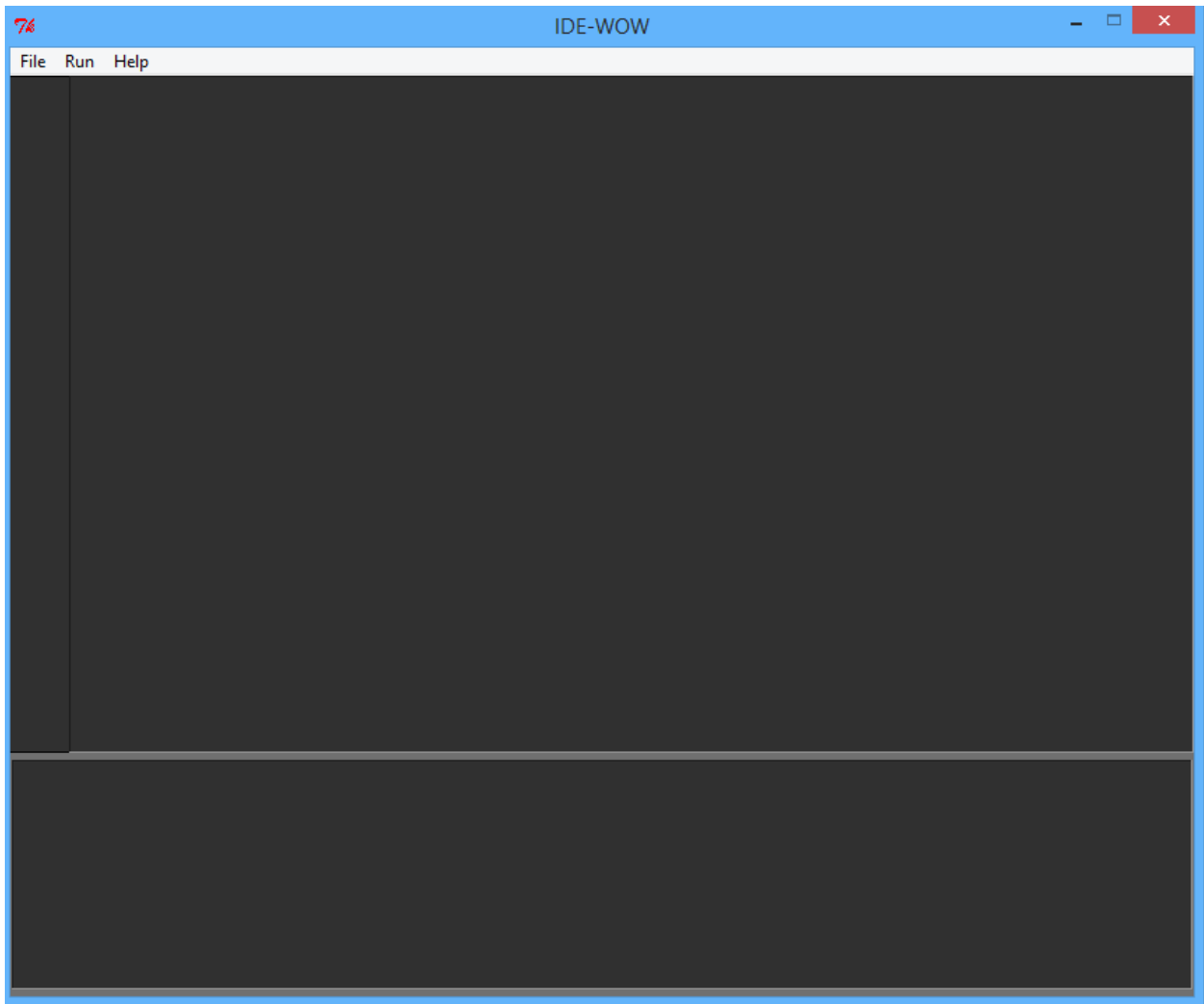
Sumanga, Elise Margaret R.

Submitted to:

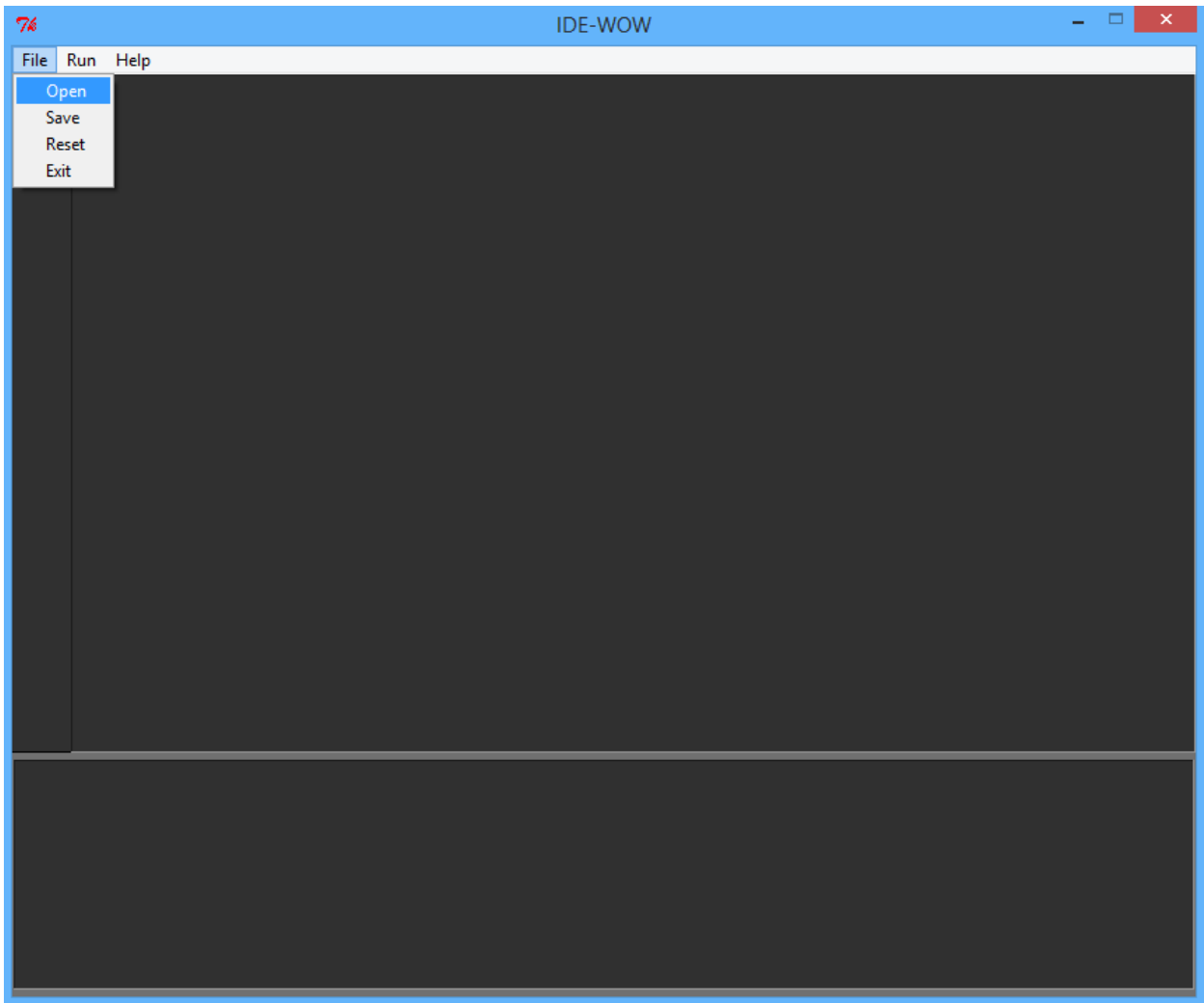
Professor Chito Patino

## HOW TO USE THE APPLICATION

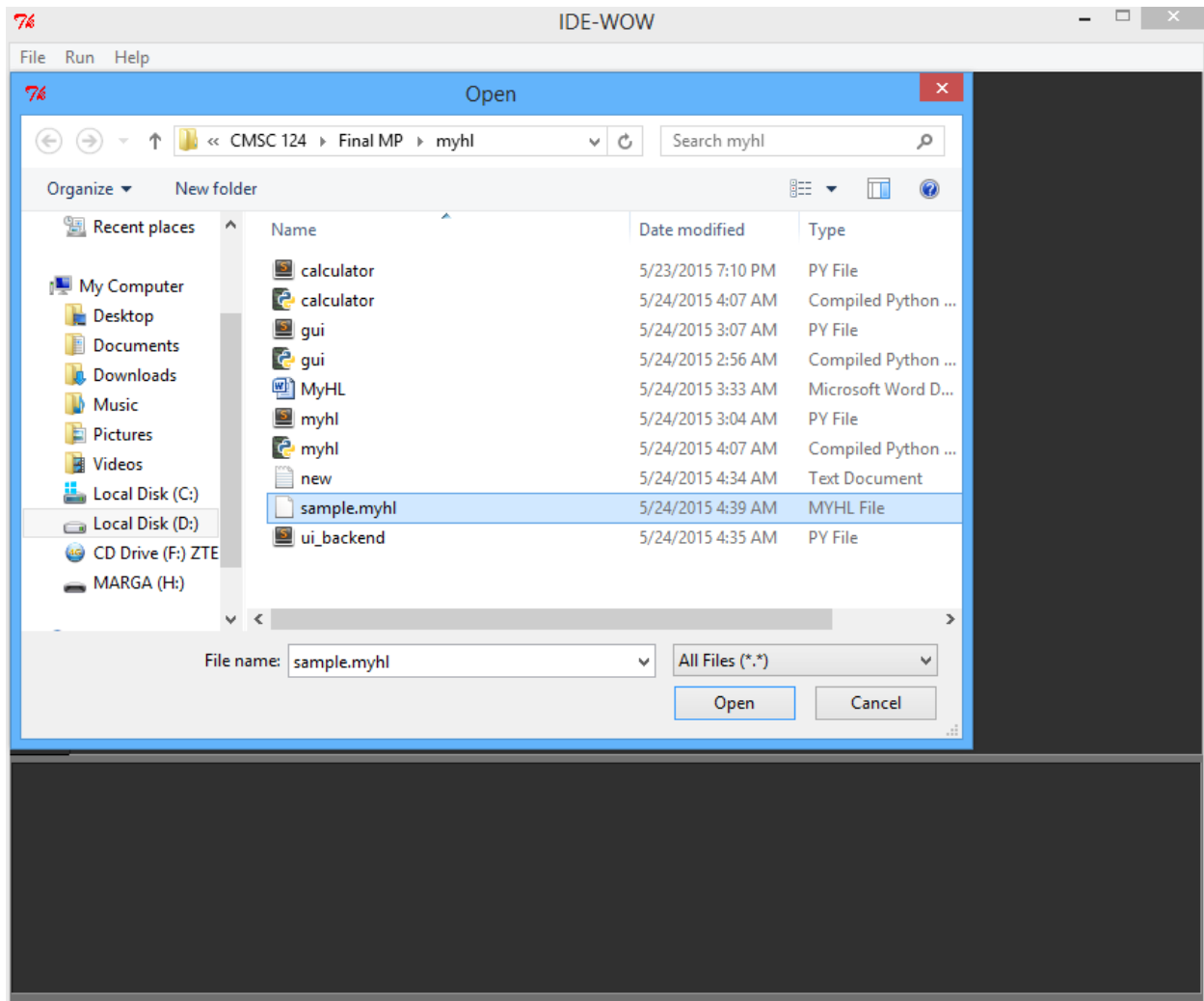
1. Open the User Interface (ui\_backend.py)



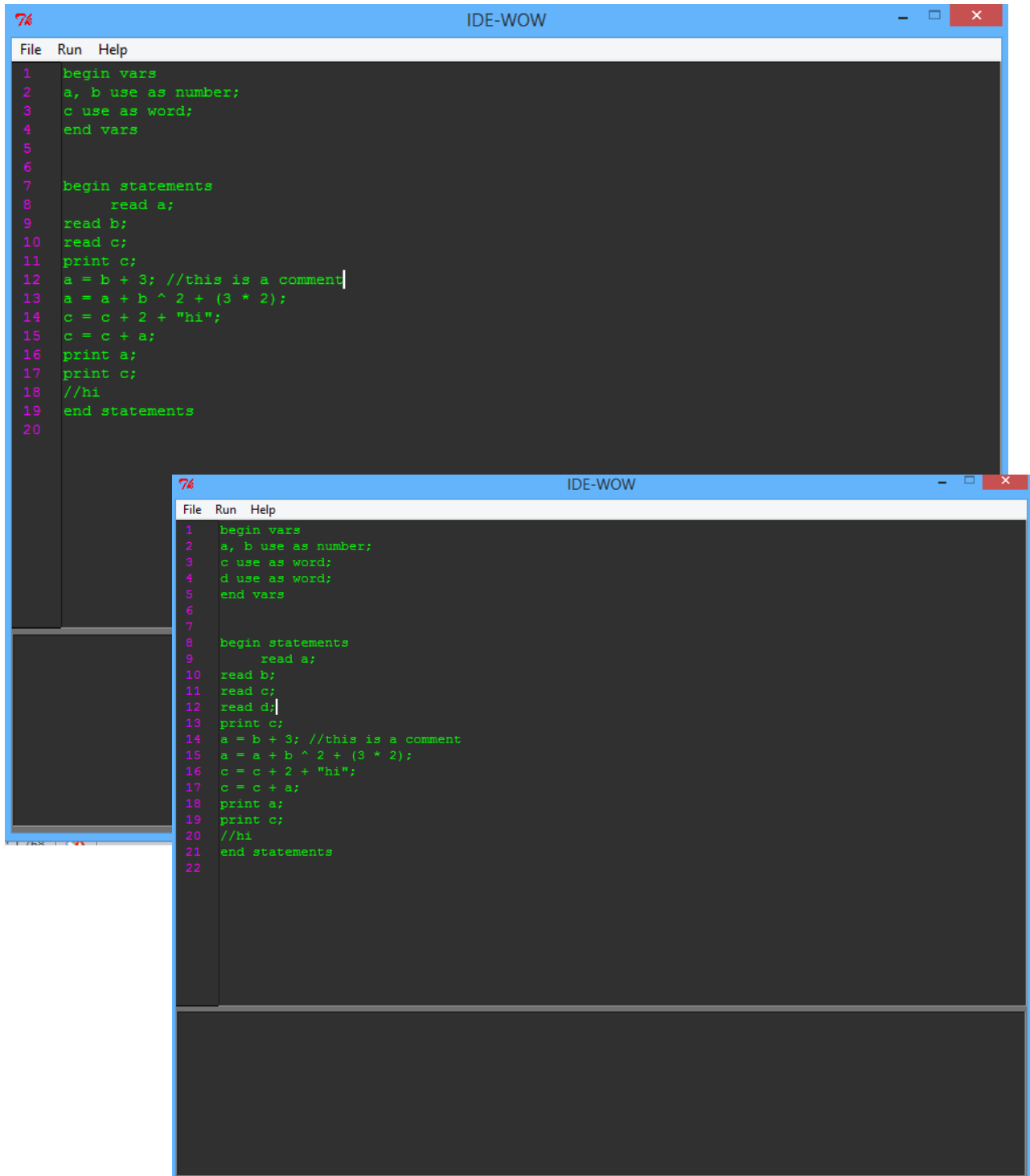
2. The user must click the File menu in the Menubar, and from the dropdown options, click Open



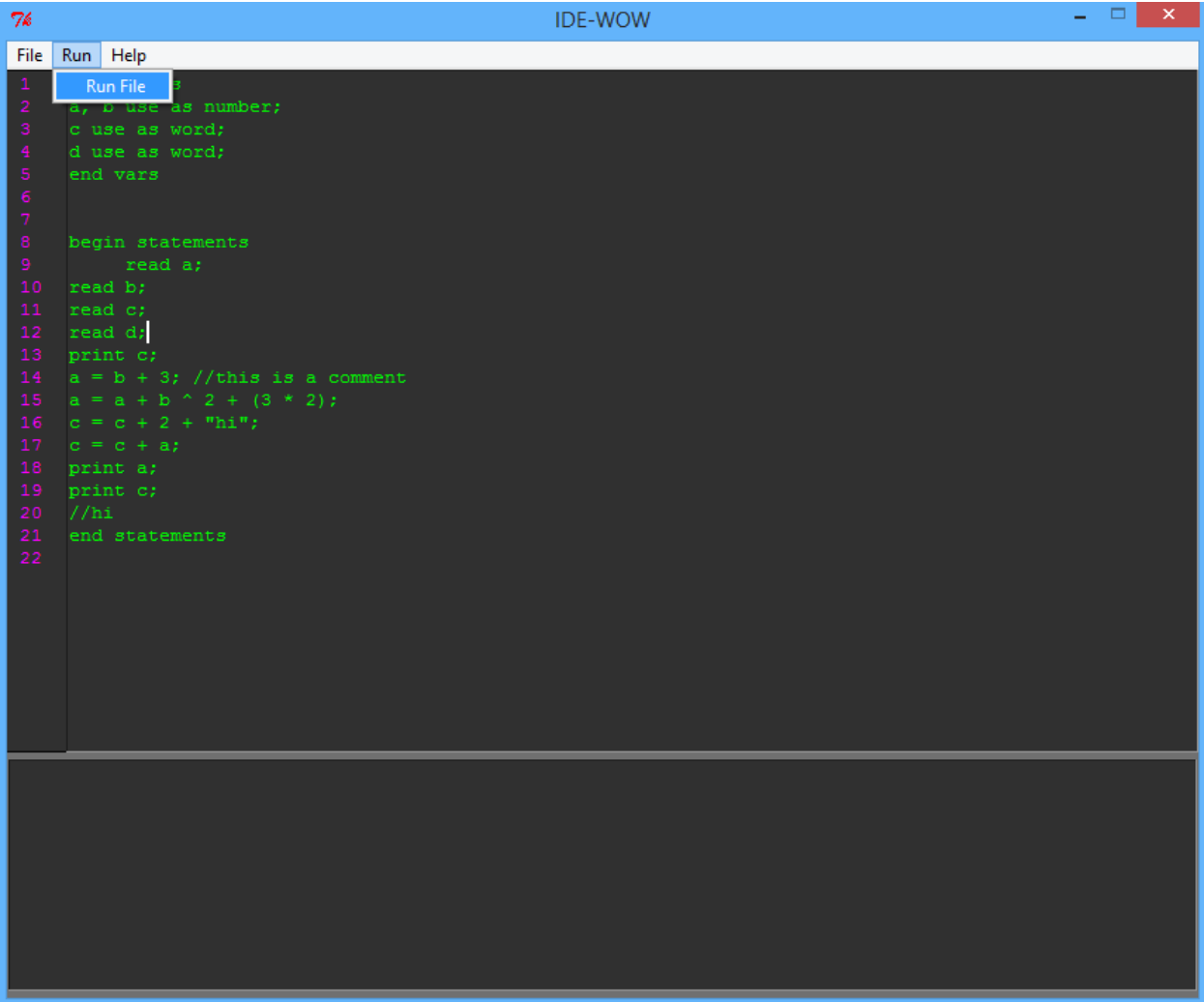
3. The user must choose the file to be executed by the system. (Ex. sample.txt)



4. The file is shown in the text area. The user has the option to edit the code.



5. The user must click the Run in the Menubar, and choose Run File option in order execute the file.



```
1  *
2  a, b use as number;
3  c use as word;
4  d use as word;
5  end vars
6
7
8  begin statements
9      read a;
10     read b;
11     read c;
12     read d;
13     print c;
14     a = b + 3; //this is a comment
15     a = a + b ^ 2 + (3 * 2);
16     c = c + 2 + "hi";
17     c = c + a;
18     print a;
19     print c;
20     //hi
21 end statements
22
```

6. The user must enter the input for each Read statement encountered. Click Go button in order to enter the value.



7. After inputting the values, the output is displayed in the console.

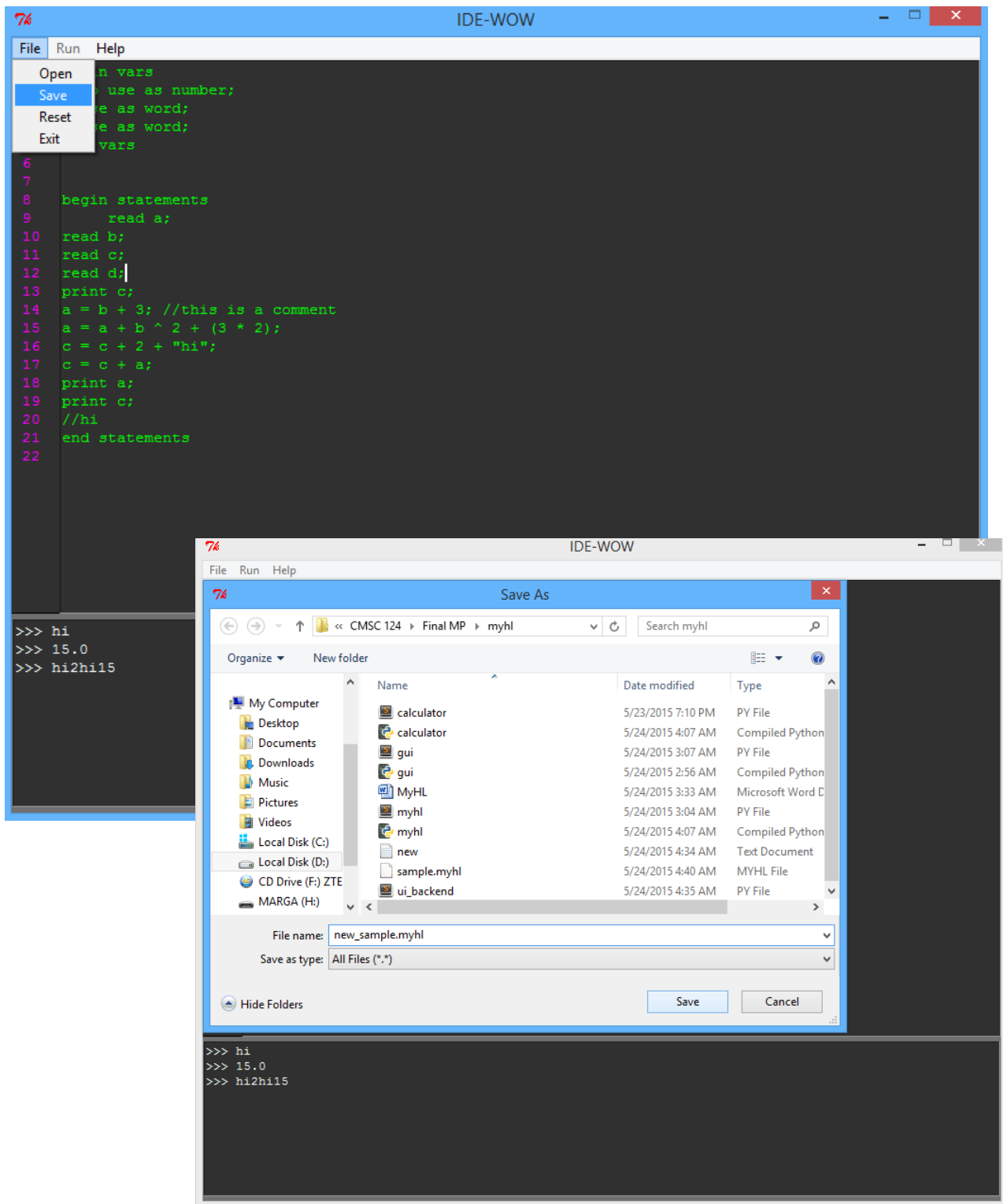
A screenshot of the IDE-WOW application window. The window has a blue title bar with the text "IDE-WOW" and standard window controls. The main area is dark gray and contains a list of code statements. The statements are numbered 1 through 22. The code is as follows:

```
1 begin vars
2 a, b use as number;
3 c use as word;
4 d use as word;
5 end vars
6
7
8 begin statements
9   read a;
10  read b;
11  read c;
12  read d;
13  print c;
14  a = b + 3; //this is a comment
15  a = a + b ^ 2 + (3 * 2);
16  c = c + 2 + "hi";
17  c = c + a;
18  print a;
19  print c;
20  //hi
21 end statements
22
```

The console area at the bottom of the window is dark gray and contains the following output:

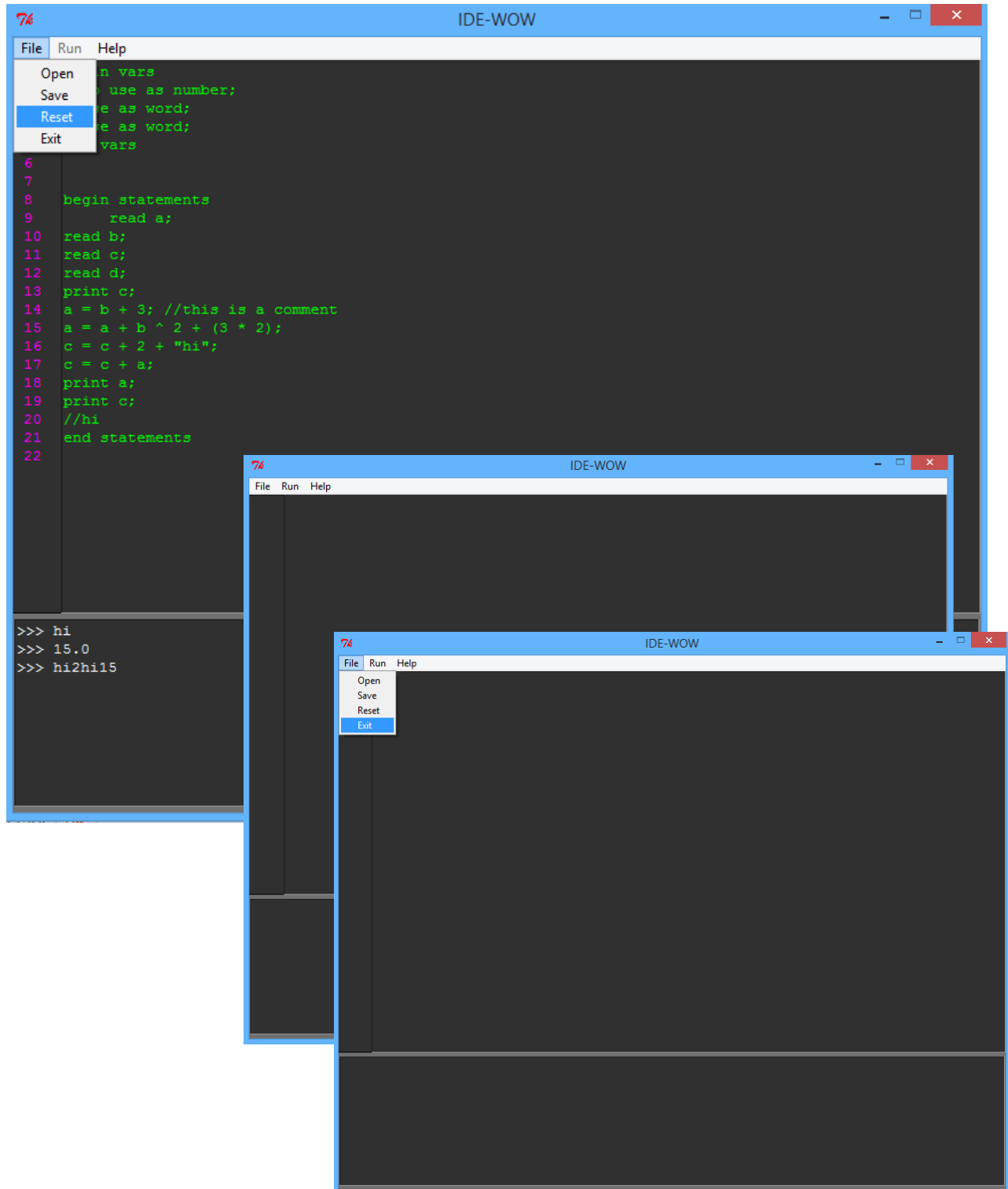
```
>>> hi
>>> 15.0
>>> hi2hi15
```

8. The user has the option to Save the edited code from the IDE.





9. The user has the option to Reset the IDE or Exit from the IDE.



## CODE SYNTAX

Below are the rules for the MyHL code:

- The first line of code specifies the start of the declaration of variables and is denoted by the word: **begin vars**.
- The last line in the declaration of variables is denoted by the word: **end vars**.
- Between **begin vars** and **end vars** are the formal declaration of variables that will be used in the MyHL program.
- Grammar for variable declaration:  
    <variable declaration> ::= <identifier list> **use as** <data type>;  
    <identifier list> ::= <identifier> | <identifier> , <identifier list>  
    <data type> ::= <number> | <word>  
    <number> follows the rules of production for non-negative int of C/Java  
    <word> follows the rules of production for Strings of Java  
    <identifier> follows the naming rules of C
- Program statements follow the **begin vars** and **end vars** block.
- The first line of the program statement block is denoted by the word **begin statements**.
- The last line of the program statement block is denoted by the word **end statements**.
- A statement in MyHL is either: read, print, or assignment.
- Grammar for program statements:  
    <program statement> ::= <read statement> | <print statement> | <assignment statement>;  
    <read statement> ::= read <identifier>  
    <print statement> ::= print <identifier>  
    <assignment statement> ::= <identifier> = <expression>  
    <expression> ::= <number-expression> | <word> | <identifier>  
    <number-expression> follows the rules of C on arithmetic expressions.
- Nothing must follow **end statements**
- **print statement** can print word or number data type.
- MyHL allows comments. Codes to right of **//** is considered a comment.

## DISCUSSION OF ALGORITHMS

### Parser

- Read each line of the input file.
- Each line is stored in a list.
- For each line containing a comment denoted by '//', the comment is removed from the line.
- Each line is stored in a dictionary as a key. The value is the index of the line to keep track the line number.

Variable (Key)	Line Number
<b>begin vars</b>	1
<b>a, b use as number</b>	2

Figure 1: "Line Number" Dictionary

- Variable declarations between the "begin vars" and "end vars" instructions are saved in a list. These lines belong to the variable declaration block.
- Statements between the "begin statements" and "end statements" instructions are saved in a list. These lines belong to the statement block.
- Reserved words are stored in a list such as the following: begin, end, vars, print, read and statements.

Reserved Words
<b>begin</b>
<b>end</b>
<b>vars</b>
<b>statements</b>
<b>print</b>
<b>read</b>

Figure 2: List of Reserved Words

- Initialize a 'memory' dictionary which stores the variables as key, and the values are stored in a list – the first value is initialized as "None", and the second value is the data type.

### Parse variable declaration block

The variable declaration block is where the user can declare variables. The block begins with "begin vars" and ends with "end vars" statements. The declarations between these statements should consist of identifier/s separated by ",", "use as", and the data type. The data

type is either word or number. The naming conventions of the identifiers should be a combination of the following:

- It should start with a letter or an underscore.
- It should be followed by digits, letters or underscores.
- It should not consist of special characters and spaces.
- It should not be one of the reserved words.
- Every line of declaration must end with “;”.

Below are the steps in parsing the variable declarations:

- The declarations are split by “use as”. The list contains the variables and data type only.
- If the variables contain “,”, it is split by “,”.
- Append the variables in the ‘memory’ dictionary together with its values.

Variable (Key)	Value 1 (None)	Value 2 (Data type)
a	None	Number
b	None	Word

Figure 3: “Memory” Dictionary with initialized “None” values

### Parse and execute statement block

The statement block starts with “begin statements” and ends with “end statements”. There are only three program statements: **read statement**, **print statements**, and **assignment statement**. The read statements can be called using the word “read” with the parameter of a variable identifier. It allows the user to accept an input, which will be assigned to the variable identifier. The print statements can be called using the word “print” with the parameter of a variable identifier. It allows the user to display the variable. The assignment statement of a variable may either consist of an arithmetic expression, an identifier, or a word/constant. It follows the rules of C on arithmetic expressions. Each statement block ends with a “;”.

There are two types of a statement declaration:

#### 1. The read and print statement

- The read statement is split by “ “. The variable to be read is checked if it is in the ‘memory’ dictionary. If it is in the dictionary, it accepts a user input. The input should either be a number or a word depending on the data type of the variable. Otherwise, an “Unknown data type” error occurs. The value of the variable in the ‘memory’ dictionary is set to the value of the user input.
- The print statement is split by “ “. The variable to be printed is checked if it is in the ‘memory’ dictionary. If it is in the dictionary, the value of the variable is displayed. Otherwise, a “Variable undefined” error occurs.

## 2. The assignment statement

- The assignment statement is split by “ ”. The left part of the statement is the assignment variable. If it is in the ‘memory’ dictionary, check the right part of the statement. The right part of the statement is the expression. If the expression starts with “=”, the expression to the right of the “=” sign is evaluated.
- There are three types of evaluating the expression: basic assignment, arithmetic expression and string concatenation.
- The basic assignment is assigning a value to the assignment variable as long as the value corresponds to the data type of the assignment variable. (e.g. a = 1;)
- The arithmetic expression consists of operands and operators (+, -, \*, /, ^) in infix form. The operands can be variables as long as they are in the ‘memory’ dictionary and the data type is a number. Otherwise, an “Incompatible type” error occurs.
- The string concatenation contains strings to be concatenated (e.g. a = “Hi sir” + “!”;) or variables as long as these are in the ‘memory’ dictionary. (e.g. a = “Hi sir” + b);

Variable (Key)	Value 1	Value 2 (Data type)
a	3	Number
b	“Hi”	Word

Figure 4: “Memory” Dictionary with set values

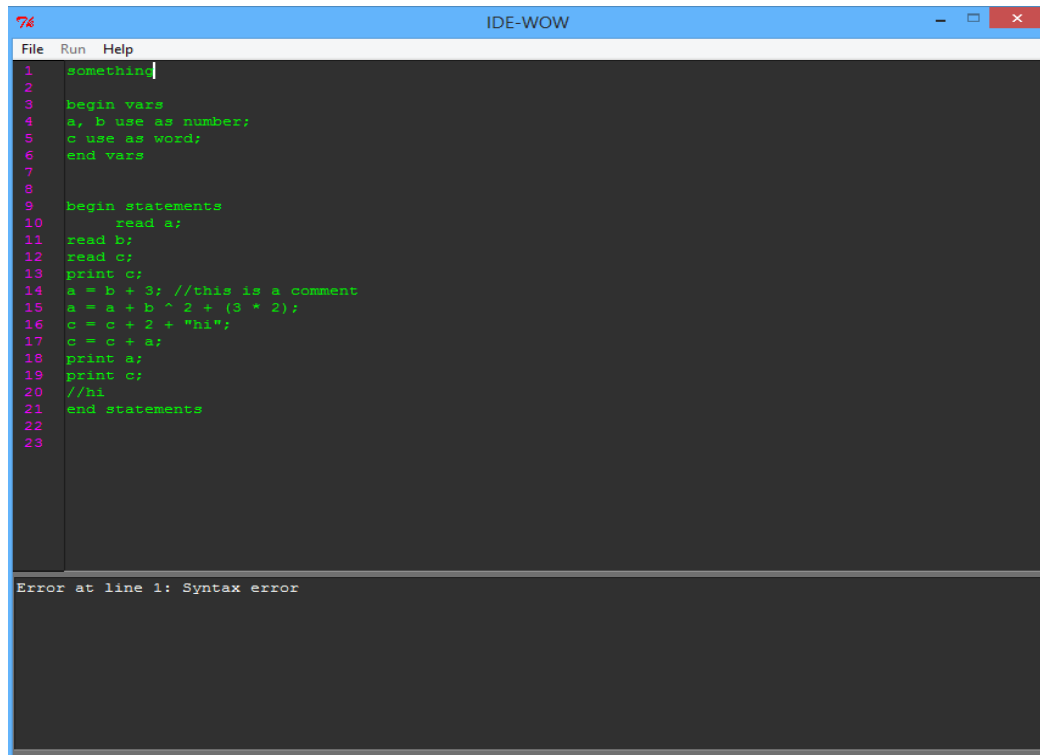
## LIST OF ERRORS

- List of Translation Errors

If an error is found, the program is automatically terminated.

Translation Error	Description	Example
1. Misplaced 'begin vars' statement	Error occurs if 'begin vars' is not found in the first statement of the MyHL code.	a use as word; begin vars
2. Syntax Error	Error occurs if code is not a variable declaration or a program statement.	begin vars weird stuff; a use as word; end vars
3. No Semicolon Found Error	Error occurs if the variable declarations and program statements do not end with semicolon.	begin vars a, b use as number; c use as word; end vars
4. Unmatched Parenthesis Error	Error occurs if arithmetic expressions in the assignment statement lack a parenthesis.	sum = 1 + (2 + 1;
5. Invalid Variable Name Error	Error occurs if variable names do not follow the specified naming convention.	1sum use as number; str*_ use as word;
6. Unknown Data Type	Error occurs if data type is not a word or a number.	sum use as string;
7. Duplicate Variable	Error occurs if variable is it is already in the 'memory' dictionary.	sum use as number; sum use as word;
8. Multiple Variables Found Error	Error occurs if multiple variables are read or printed.	read a, b, c; print a, b, c;
9. Variable Undefined Error	Error occurs if the variable is not in the 'memory' dictionary.	{a : [3, number} print b;
10. Invalid Expression Error	Error occurs if the arithmetic expression returns a non-type value.	a = 1 1 +; or b = ( 2 + 3 ;
11. String Concatenation Error	Error occurs if operator used in concatenating the string is not "+".	a = "Hi" * (1 + 1);
12. Assignment Error	Error occurs if the assignment operator is not "=".	a += "hi";

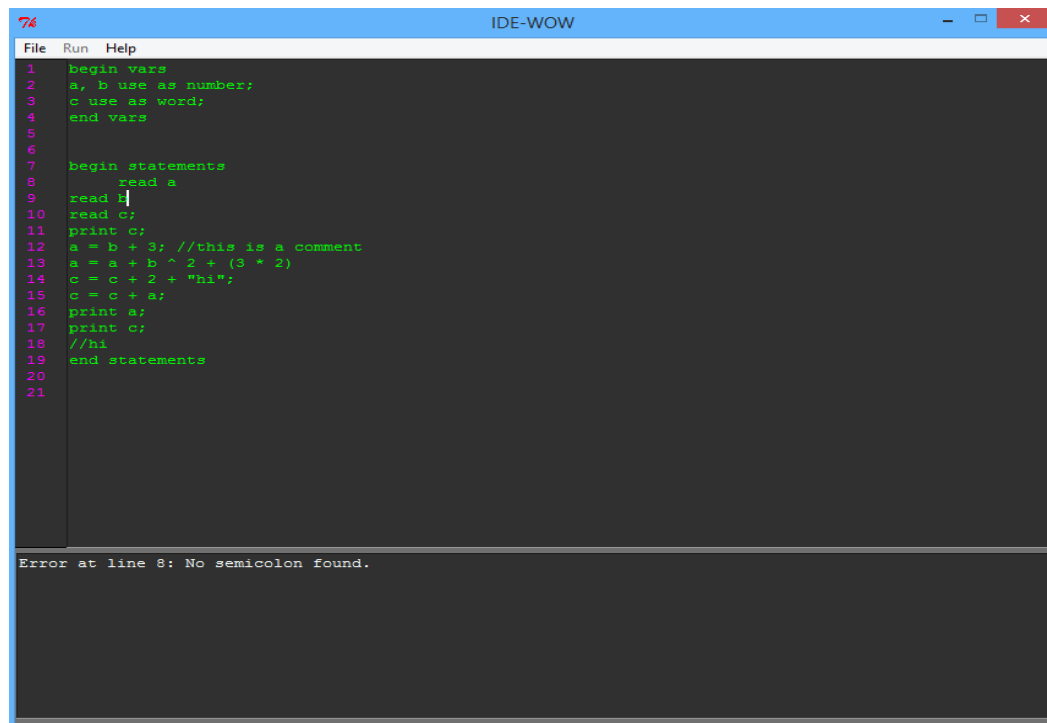
## Sample Translation Error Screenshots:



The screenshot shows the IDE-WOW window with a menu bar (File, Run, Help) and a code editor. The code is as follows:

```
1 something
2
3 begin vars
4 a, b use as number;
5 c use as word;
6 end vars
7
8
9 begin statements
10 read a;
11 read b;
12 read c;
13 print c;
14 a = b + 3; //this is a comment
15 a = a + b ^ 2 + (3 * 2);
16 c = c + 2 + "hi";
17 c = c + a;
18 print a;
19 print c;
20 //hi
21 end statements
22
23
```

The error message at the bottom reads: "Error at line 1: Syntax error".



The screenshot shows the IDE-WOW window with a menu bar (File, Run, Help) and a code editor. The code is as follows:

```
1 begin vars
2 a, b use as number;
3 c use as word;
4 end vars
5
6
7 begin statements
8 read a
9 read b
10 read c;
11 print c;
12 a = b + 3; //this is a comment
13 a = a + b ^ 2 + (3 * 2);
14 c = c + 2 + "hi";
15 c = c + a;
16 print a;
17 print c;
18 //hi
19 end statements
20
21
```

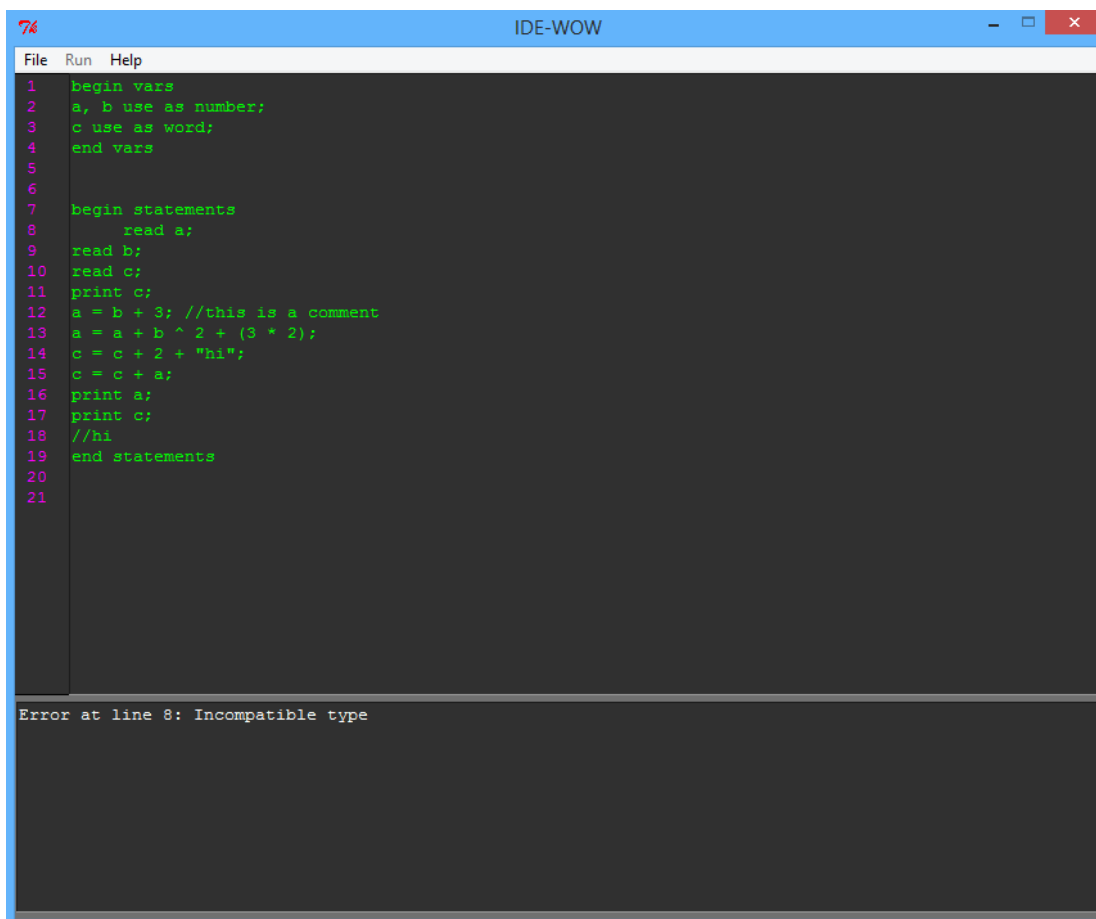
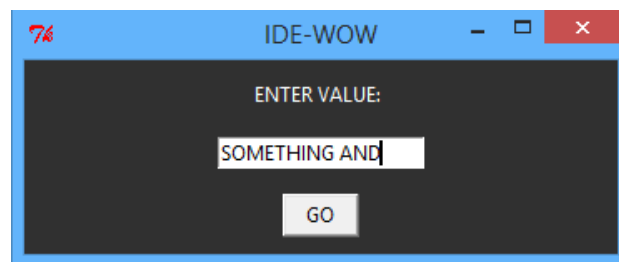
The error message at the bottom reads: "Error at line 8: No semicolon found."

- List of Run-time Errors

If an error is found, the program is automatically terminated.

Run-time Error	Description	Example
1. Incompatible Type Error	Error occurs if the user input read does not correspond to the data type of the variable	read a; (user input is "hi" but a is a number)
2. Null Pointer Exception Error	Error occurs if the variables are in the 'memory' dictionary but are not yet read, thus, returns a null pointer.	a = b + c; read a; read b; read c;

Sample Runtime Error Screenshot:





## INSIGHTS

*"In this machine problem, we learned that first and foremost, before we start coding, we must understand the problem and come up with an algorithm. We must take into account our strengths and weaknesses in order to designate the tasks properly and efficiently implement our solution to the problem. Since we know that our schedule is hectic, we must learn how to manage our time. Also, communication with one another is an important key in accomplishing the machine problem.*

*In addition, we realized how important the lab exercises, discussions and MP1 are in solving the problem. Some methods in our MyAsm machine problem served as our basis in making our MyHL compiler. "*

**- Aldwyn C., Elise S., Lindy P., Shairine M.**

## REFERENCES

- Python GUI Programming (Tkinter)  
[http://www.tutorialspoint.com/python/python\\_gui\\_programming.htm](http://www.tutorialspoint.com/python/python_gui_programming.htm)
- Python Errors  
<http://cs-people.bu.edu/dgs/courses/cs111/assignments/errors.html>
- Java Naming Conventions  
<http://www.javatpoint.com/java-naming-conventions>