Project (Part 1)

1 Team Members

- Thanh Pham - NetID: tvp200000
- Bach Nguyen - NetID: nvb180000

2 The changes

Add a new feature: FOR loop

- Syntax: Lines: (9) and (17)
- Static Semantics: Line (9)
- Dynamic Semantics: densem(<for>)

3 The Language

3.1 Syntax

```
(1)        <prog> → <stmt_list>
(2)   <stmt_list> → ε
(3)               | <stmt> “;” <stmt_list>
(4)        <stmt> → <print>
(5)               | <input>
(6)               | <assign>
(7)               | <if>
(8)               | <while>
(9)               | <for>
(10)      <print> → “print” <p-arg>
(11)      <p-arg> → STRING
(12)              | <expr>
(13)      <input> → “get” ID
(14)     <assign> → ID “=” <expr>
(15)         <if> → “if” <expr> “then” <stmt_list> “else” <stmt_list> “end”
(16)      <while> → “while” <expr> “do” <stmt_list> “end”
(17)        <for> → “for” <expr> <assign> “,” <expr> “,” <expr> <stmt_list>
(18)       <expr> → <n_expr> <b_expr>
(19)     <b_expr> → ε
```

(20) | “and” <n_expr>

(21) | “or” <n_expr>

(22) <n_expr> → <term> <t_expr>

(23) <t_expr> → ϵ

(24) | “+” <n_expr>

(25) | “-” <n_expr>

(26) <term> → <factor> <f_expr>

(27) <f_expr> → ϵ

(28) | “*” <term>

(29) | “/” <term>

(30) | “%” <term>

(31) <factor> → <value> <v_expr>

(32) <v_expr> → ϵ

(33) | “>” <value>

(34) | “>=” <value>

(35) | “<” <value>

(36) | “<=” <value>

(37) | “==” <value>

(38) | “!=” <value>

(39) <value> → “(” <expr> “)”

(40) | “not” <value>

(41) | “-” <value>

(42) | **ID**

(43) | **INT**

3.1.1 Tokens

This subsection describes the token used in the above grammar. Provided for each token is a regex and a description. The regex is for those that know regular expressions and prefer it as a description. The description says the same thing in English. Preprocessing describes how the lexeme is transformed before passing it to the parser.

STRING

As a regex: "([^"]|\")*"

Description: A quotation mark followed by zero or more characters, where quotation marks must be preceded by a backslash, followed by another quotation mark.

Preprocessing: The first and last quotation marks are removed. Scanning from left to right, “\\” is replaced with “\”, “\t” is replaced with a tab, “\n” is replaced with a newline, “\"” is replaced with “"”, and any “\” that is followed by anything else is removed.

ID

As regex: [_a-zA-Z][_a-zA-Z0-9]*

Description: A letter or underscore followed by a combination of zero or more letters, underscores or digits.

INT

As Regex: (+|-)?[0-9]$^+$

Description: an optional "+" or "-" followed by one or more digits.

3.2 Static Semantics

1 `<stmt_list>` $.ids = \{\}$

3 `<stmt_list>` $[1].ids = \{$ `<stmt>` $.id\} \cup$ `<stmt_list>` $[0].ids$
`<stmt>` $.ids =$ `<stmt_list>` $[0].ids$

4 `<print>` $.ids =$ `<stmt>` $.ids$

5 `<stmt>` $.id =$ `<input>` $.id$

6 `<stmt>` $.id =$ `<assign>` $.id$
`<assign>` $.ids =$ `<stmt>` $.ids$

7 `<if>` $.ids =$ `<stmt>` $.ids$

8 `<while>` $.ids =$ `<stmt>` $.ids$

9 `<for>` $.ids =$ `<stmt>` $.ids$

10 `<p-arg>` $.ids =$ `<print>` $.ids$

11 `<expr>` $.ids =$ `<p-arg>` $.ids$

12 `<input>` $.id =$ **ID** $.id$

13 `<assign>` $.id =$ **ID** $.id$

16 `<n_expr>` $.ids =$ `<expr>`
`<b_expr>` $.ids =$ `<expr>`

18 `<n_expr>` $.ids =$ `<b_expr>`

19 `<n_expr>` $.ids =$ `<b_expr>`

20 `<term>` $.ids =$ `<b_expr>`
`<t_expr>` $.ids =$ `<b_expr>`

22 `<n_expr>` $.ids =$ `<t_expr>` $.ids$

23 `<n_expr>` $.ids =$ `<t_expr>` $.ids$

24 `<factor>` $.ids$ = `<term>` $.ids$
`<f_expr>` $.ids$ = `<term>` $.ids$

26 `<term>` $.ids$ = `<f_expr>` $.ids$

27 `<term>` $.ids$ = `<f_expr>` $.ids$

28 `<term>` $.ids$ = `<f_expr>` $.ids$

29 `<value>` $.ids$ = `<factor>` $.ids$
`<v_expr>` $.ids$ = `<factor>` $.ids$

31 `<value>` $.ids$ = `<v_expr>` $.ids$

32 `<value>` $.ids$ = `<v_expr>` $.ids$

33 `<value>` $.ids$ = `<v_expr>` $.ids$

34 `<value>` $.ids$ = `<v_expr>` $.ids$

35 `<value>` $.ids$ = `<v_expr>` $.ids$

36 `<value>` $.ids$ = `<v_expr>` $.ids$

37 `<expr>` $.ids$ = `<value>` $.ids$

38 `<value>` $[1].ids$ = `<value>` $[0].ids$

39 `<value>` $[1].ids$ = `<value>` $[0].ids$

40 Predicate: **ID** $.id \in$ `<value>` $.ids$

3.3 Dynamic Semantics

This section gives the dynamic semantics of the language using denotational semantics. Consider the *demsem* function the denotational semantics for this language. We will use a mapping from variable name to value to represent the symbol table of the program during execution, and in code can be represented as a HashMap or similar datatype in your language of choice. We will use a sequence of characters to represent the output of a program, with ϵ representing the empty sequence. I will also assume that all strings will be represented as sequences of characters. Assume there is a function *append* that, when given two sequences, appends the second sequence to the first. Also assume, there is a function *seq* that takes an integer and gives a sequence of characters representing that integer as text. Assume there are the functions *head*, which maps a sequence to its first element, *tail*, which maps a sequence to a new one created by removing the first element, *clean*, which maps a sequence of input characters to a new sequence by removing any non-digits from the front of the sequence, and *int* that maps a sequence of digits to the corresponding integer. If the sequence is empty, *int* will give zero. A state, as well as the meaning of a program, will be a 3-tuple consisting of a variable name mapping function, a sequence of input characters and an output sequence. The initial state for any program is $(\{\}, i, \epsilon)$, where i is some sequence of characters the user will input. If a token (represented by all caps and bold font) appears as a value on the right hand side of a function definition, then replace it with its lexeme. So if a **ID** was generated by the lexer from an x, then replace **ID** with x.

$$
\begin{aligned}
\mathit{densem}(\epsilon, (\theta, i, p)) &= (\theta, i, p) \\
\mathit{densem}(\texttt{<stmt>}\ \text{``;''}\ \texttt{<stmt\_list>}, (\theta, i, p)) &= \mathit{densem}(\texttt{<stmt\_list>}, \mathit{densem}(\texttt{<stmt>}, (\theta, i, p))) \\
\mathit{densem}(\text{``print''}\ \textbf{STRING}, (\theta, i, p)) &= (\theta, i, \mathit{append}(p, \textbf{STRING})) \\
\mathit{densem}(\text{``print''}\ \texttt{<expr>}, (\theta, i, p) &= (\theta, i, \mathit{append}(p, \mathit{seq}(\mathit{out}))) \\
&\ \text{where } \mathit{out} = \mathit{exprsem}(\texttt{<expr>}) \\
\mathit{densem}(\text{``get''}\ \textbf{ID}, (\theta, i, p)) &= (\theta', i', p) \\
&\ \text{where} \\
&(x, i') = \mathit{getInt}(\mathit{clean}(i)) \\
&\theta'(n) = \text{if } n = \textbf{ID} \text{ then } x \text{ else } \theta(n) \\
\mathit{densem}(\textbf{ID}\ \text{``=''}\ \texttt{<expr>}, (\theta, i, p)) &= (\theta', i, p) \\
&\ \text{where} \\
&\theta'(n) = \text{if } n = \textbf{ID} \text{ then } \mathit{exprsem}(\texttt{<expr>}, \theta) \text{ else } \theta(n) \\
\mathit{densem}(\texttt{<if>}, (\theta, i, p)) &= \text{if } \mathit{exprsem}(\texttt{<if>}\,.\,\texttt{<expr>}, \theta) \neq 0 \\
&\ \text{then } \mathit{densem}(\texttt{<if>}\,.\,\texttt{<stmt\_list>}[0], (\theta, i, p)) \\
&\ \text{else } \mathit{densem}(\texttt{<if>}\,.\,\texttt{<stmt\_list>}[1], (\theta, i, p)) \\
\mathit{densem}(\texttt{<while>}, (\theta, i, p)) &= \text{if } \mathit{exprsem}(\texttt{<while>}\,.\,\texttt{<expr>}, \theta) = 0 \\
&\ \text{then } (\theta, i, p) \\
&\ \text{else } \mathit{densem}(\texttt{<while>}, \\
&\qquad\qquad\quad \mathit{densem}(\texttt{<while>}\,.\,\texttt{<stmt\_list>}, (\theta, i, p))) \\
\mathit{densem}(\texttt{<for>}, (\theta, i, p)) &= \text{if } \mathit{exprsem}(\texttt{<for>}\,.\,\texttt{<expr>}\,.\,\texttt{<assign>}\,.\,\text{``,''}\,. \\
&\qquad\qquad\quad \texttt{<expr>}\,.\,\text{``,''}\,.\,\texttt{<expr>}, \theta) = 0 \\
&\ \text{then } (\theta, i, p) \\
&\ \text{else } \mathit{densem}(\texttt{<for>}, \\
&\qquad\qquad\quad \mathit{densem}(\texttt{<for>}\,.\,\texttt{<stmt\_list>}, (\theta, i, p))) \\
\mathit{exprsem}(\texttt{<expr>}, \theta) &= \text{if } \texttt{<expr>}\,.\,\texttt{<b\_expr>} = \epsilon \\
&\ \text{then } \mathit{exprsem}(\texttt{<expr>}\,.\,\texttt{<n\_expr>}, \theta) \\
&\ \text{else } \mathit{bexprsem}(\texttt{<expr>}\,.\,\texttt{<b\_expr>}, \\
&\qquad\qquad\quad \mathit{exprsem}(\texttt{<expr>}\,.\,\texttt{<n\_expr>}), \theta) \\
\mathit{exprsem}(\texttt{<n\_expr>}, \theta) &= \text{if } \texttt{<n\_expr>}\,.\,\texttt{<t\_expr>} = \epsilon \\
&\ \text{then } \mathit{exprsem}(\texttt{<n\_expr>}\,.\,\texttt{<term>}, \theta) \\
&\ \text{else } \mathit{texprsem}(\texttt{<n\_expr>}\,.\,\texttt{<t\_expr>}, \\
&\qquad\qquad\quad \mathit{exprsem}(\texttt{<n\_expr>}\,.\,\texttt{<term>}), \theta) \\
\mathit{exprsem}(\texttt{<term>}, \theta) &= \text{if } \texttt{<term>}\,.\,\texttt{<f\_expr>} = \epsilon \\
&\ \text{then } \mathit{exprsem}(\texttt{<term>}\,.\,\texttt{<factor>}, \theta) \\
&\ \text{else } \mathit{fexprsem}(\texttt{<term>}\,.\,\texttt{<f\_expr>},
\end{aligned}
$$

$$
\begin{aligned}
& \qquad\qquad\qquad\qquad \mathit{exprsem}(\, \texttt{<term>} \,.\, \texttt{<factor>} \,), \theta) \\
\mathit{exprsem}(\, \texttt{<factor>} \,, \theta) &= \text{if } \ \texttt{<factor>} \,.\, \texttt{<v\_expr>} \ = \epsilon \\
& \text{then } \mathit{exprsem}(\, \texttt{<factor>} \,.\, \texttt{<value>} \,, \theta) \\
& \text{else } \mathit{vexprsem}(\, \texttt{<factor>} \,.\, \texttt{<v\_expr>} \,, \\
& \qquad\qquad\qquad \mathit{exprsem}(\, \texttt{<factor>} \,.\, \texttt{<value>} \,), \theta) \\
\mathit{exprsem}(\, \text{“(”} \ \texttt{<expr>} \ \text{“)”} \,, \theta) &= \mathit{exprsem}(\, \texttt{<expr>} \,, \theta) \\
\mathit{exprsem}(\, \text{“not”} \ \texttt{<value>} \,, \theta) &= \text{if } \mathit{exprsem}(\, \texttt{<value>} \,, \theta) = 0 \text{ then } 1 \text{ else } 0 \\
\mathit{exprsem}(\, \text{“-”} \ \texttt{<value>} \,, \theta) &= -\mathit{exprsem}(\, \texttt{<value>} \,, \theta) \\
\mathit{exprsem}(\, \mathbf{ID} \,, \theta) &= \theta(\, \mathbf{ID} \,) \\
\mathit{exprsem}(\, \mathbf{INT} \,, \theta) &= \ \mathbf{INT} \\
\mathit{bexprsem}(\, \text{“and”} \ \texttt{<n\_expr>} \,, v, \theta) &= \text{if } v \neq 0 \text{ and } \mathit{exprsem}(\, \texttt{<n\_expr>} \,, \theta) \neq 0 \text{ then } 1 \text{ else } 0 \\
\mathit{bexprsem}(\, \text{“or”} \ \texttt{<n\_expr>} \,, v, \theta) &= \text{if } v \neq 0 \text{ or } \mathit{exprsem}(\, \texttt{<n\_expr>} \,, \theta) \neq 0 \text{ then } 1 \text{ else } 0 \\
\mathit{texprsem}(\, \text{“+”} \ \texttt{<n\_expr>} \,, v, \theta) &= v + \mathit{exprsem}(\, \texttt{<n\_expr>} \,, \theta) \\
\mathit{texprsem}(\, \text{“-”} \ \texttt{<n\_expr>} \,, v, \theta) &= v - \mathit{exprsem}(\, \texttt{<n\_expr>} \,, \theta) \\
\mathit{fexprsem}(\, \text{“*”} \ \texttt{<term>} \,, v, \theta) &= v \times \mathit{exprsem}(\, \texttt{<term>} \,, \theta) \\
\mathit{fexprsem}(\, \text{“/”} \ \texttt{<term>} \,, v, \theta) &= \frac{v}{\mathit{exprsem}(\, \texttt{<term>} \,, \theta)} \\
\mathit{fexprsem}(\, \text{“\%”} \ \texttt{<term>} \,, v, \theta) &= v \mod \mathit{exprsem}(\, \texttt{<term>} \,, \theta) \\
\mathit{vexprsem}(\, \text{“>”} \ \texttt{<value>} \,, v, \theta) &= \text{if } v > \mathit{exprsem}(\, \texttt{<value>} \,, \theta) \text{ then } 1 \text{ else } 0 \\
\mathit{vexprsem}(\, \text{“>=”} \ \texttt{<value>} \,, v, \theta) &= \text{if } v \geqslant \mathit{exprsem}(\, \texttt{<value>} \,, \theta) \text{ then } 1 \text{ else } 0 \\
\mathit{vexprsem}(\, \text{“<”} \ \texttt{<value>} \,, v, \theta) &= \text{if } v < \mathit{exprsem}(\, \texttt{<value>} \,, \theta) \text{ then } 1 \text{ else } 0 \\
\mathit{vexprsem}(\, \text{“<=”} \ \texttt{<value>} \,, v, \theta) &= \text{if } v \leqslant \mathit{exprsem}(\, \texttt{<value>} \,, \theta) \text{ then } 1 \text{ else } 0 \\
\mathit{vexprsem}(\, \text{“==”} \ \texttt{<value>} \,, v, \theta) &= \text{if } v = \mathit{exprsem}(\, \texttt{<value>} \,, \theta) \text{ then } 1 \text{ else } 0 \\
\mathit{vexprsem}(\, \text{“!=”} \ \texttt{<value>} \,, v, \theta) &= \text{if } v \neq \mathit{exprsem}(\, \texttt{<value>} \,, \theta) \text{ then } 1 \text{ else } 0 \\
\mathit{getInt}(i) &= (\mathit{int}(x), i') \\
& \text{where } (x, i') = \mathit{getIntSeq}(\epsilon, i) \\
\mathit{getIntSeq}(i_1, i_2) &= \text{if } \mathit{digit}(\mathit{head}(i_2)) \\
& \text{then } \mathit{getIntSeq}(\mathit{append}(i_1, \mathit{head}(i_2)), \mathit{tail}(i_2)) \\
& \text{else } (i_1, i_2)
\end{aligned}
$$

3.4 State Transition Diagram

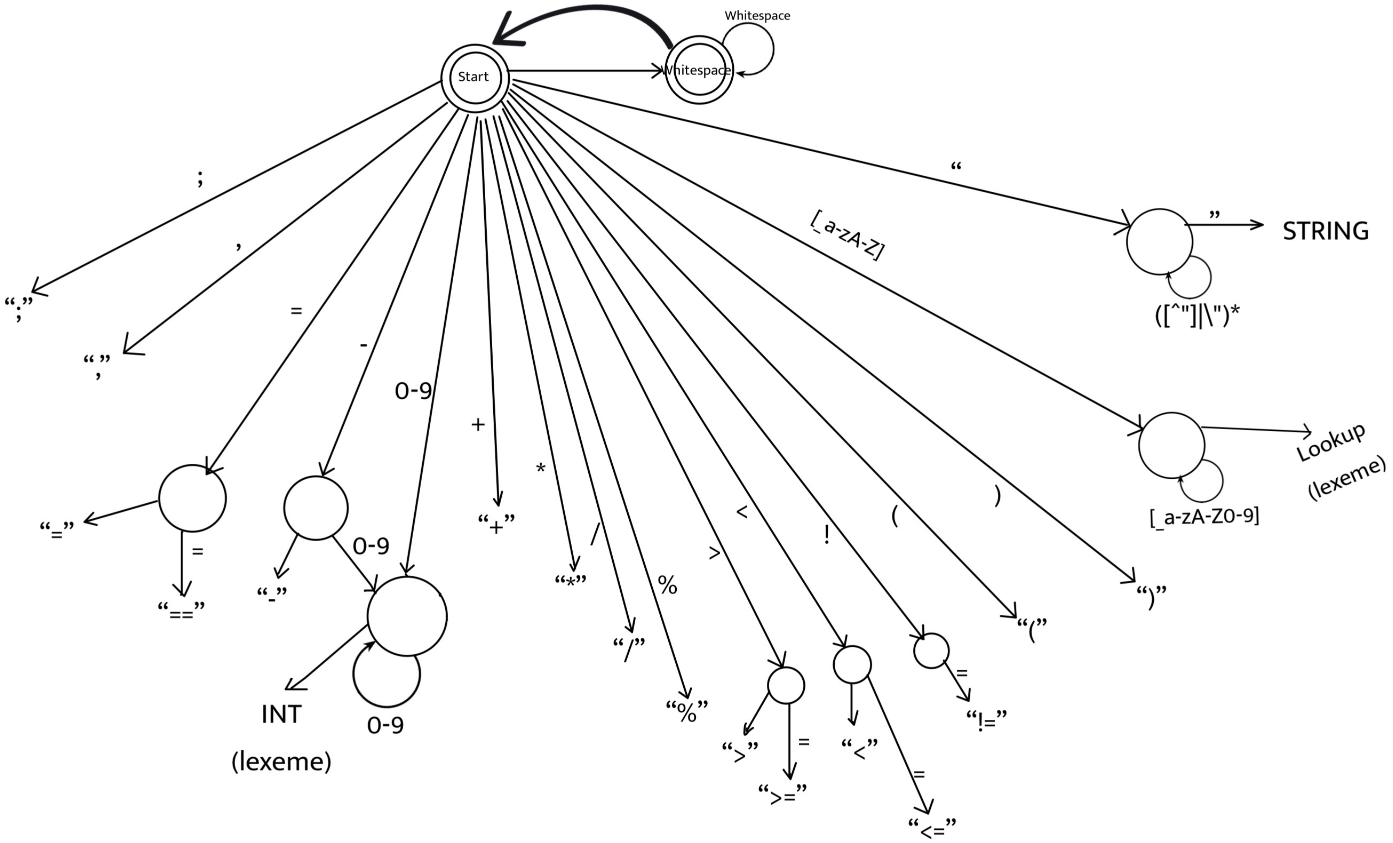
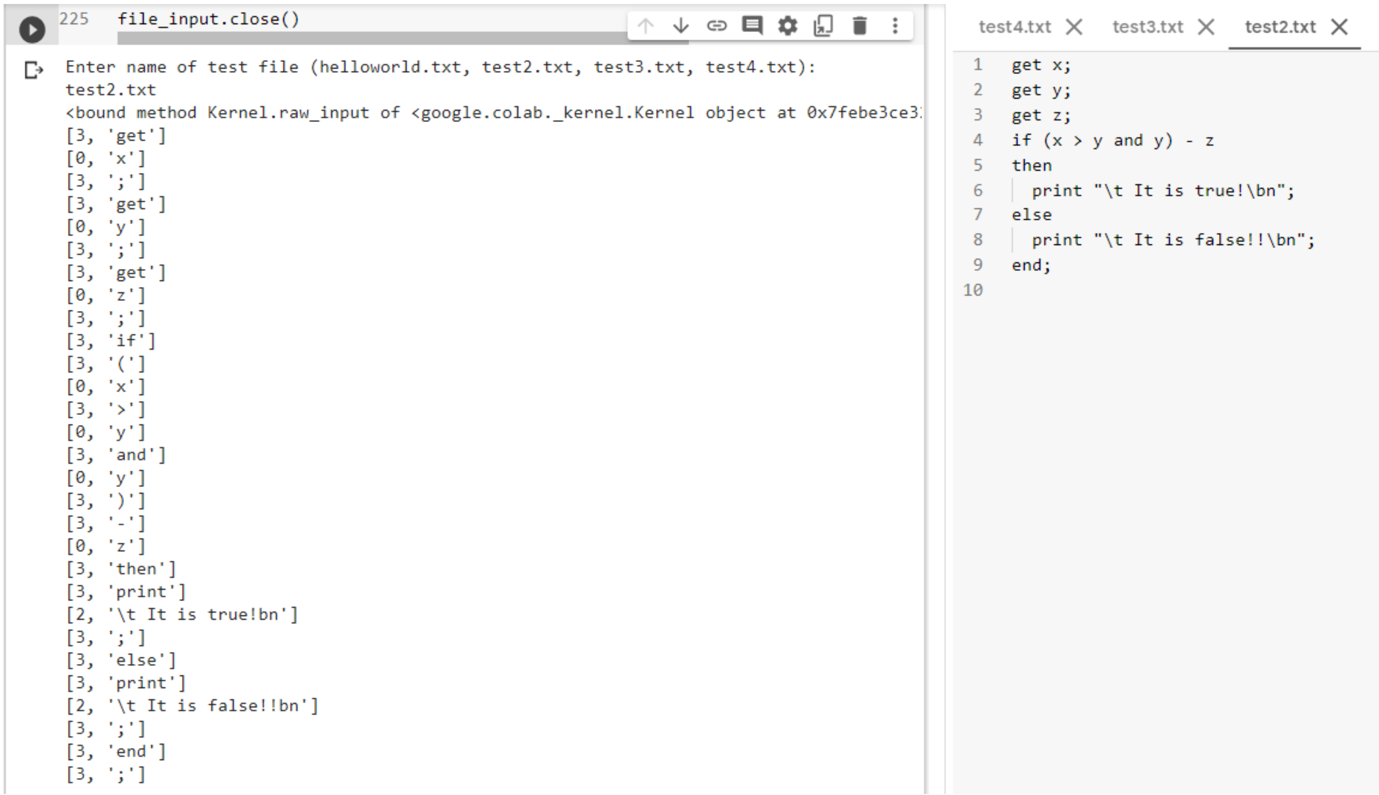


Figure 1: State Transition Diagram.

3.5 Test Programs: Output

This section contains a few test programs with output.

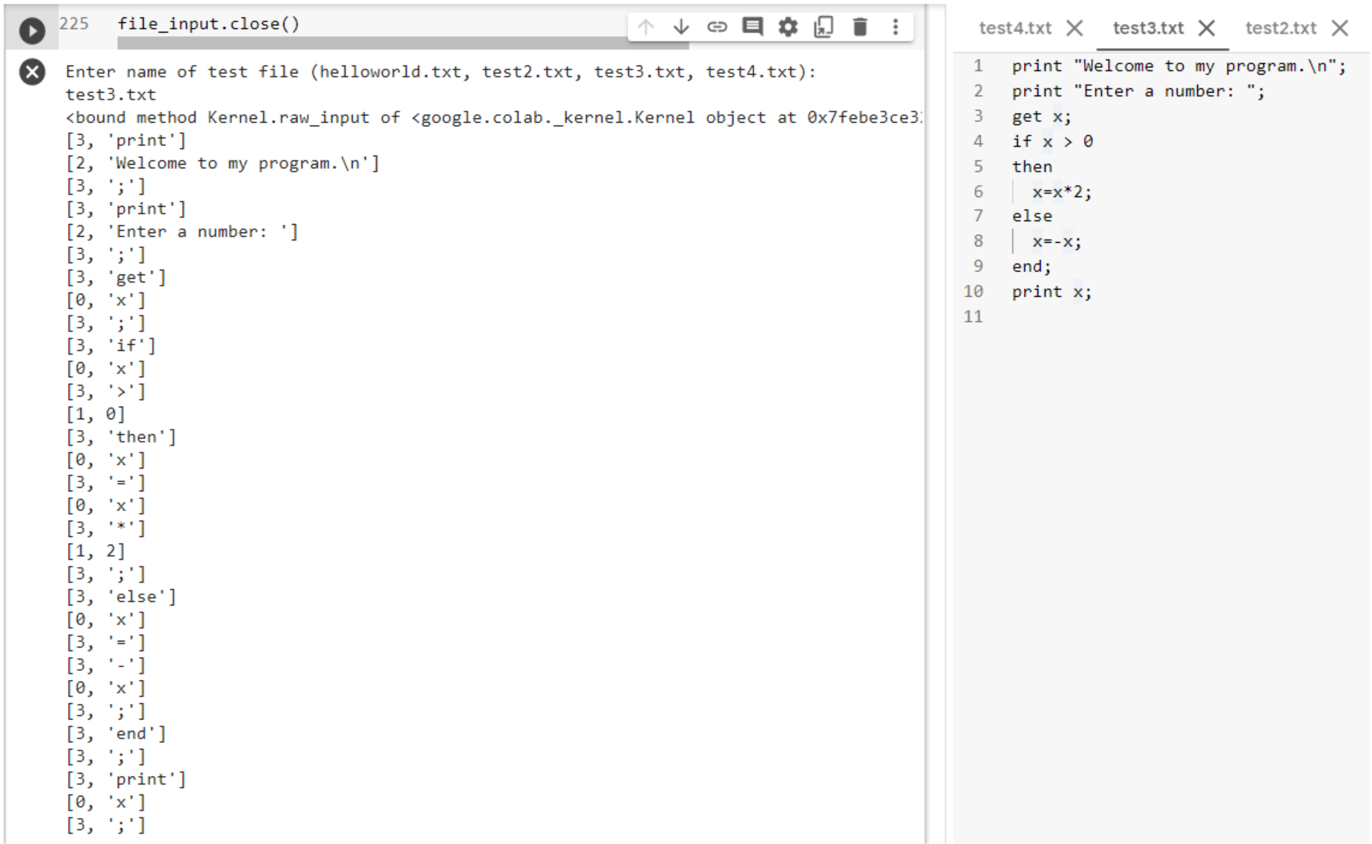


```
file_input.close()
```

```
Enter name of test file (helloworld.txt, test2.txt, test3.txt, test4.txt):
test2.txt
<bound method Kernel.raw_input of <google.colab._kernel.Kernel object at 0x7febe3ce3
[3, 'get']
[0, 'x']
[3, ';']
[3, 'get']
[0, 'y']
[3, ';']
[3, 'get']
[0, 'z']
[3, ';']
[3, 'if']
[3, '(']
[0, 'x']
[3, '>']
[0, 'y']
[3, 'and']
[0, 'y']
[3, ')']
[3, '-']
[0, 'z']
[3, 'then']
[3, 'print']
[2, '\t It is true!bn']
[3, ';']
[3, 'else']
[3, 'print']
[2, '\t It is false!!bn']
[3, ';']
[3, 'end']
[3, ';']
```

```
get x;
get y;
get z;
if (x > y and y) - z
then
  print "\t It is true!\bn";
else
  print "\t It is false!!\bn";
end;
```

Figure 2: test2.txt's output.



```
file_input.close()
```

```
Enter name of test file (helloworld.txt, test2.txt, test3.txt, test4.txt):
test3.txt
<bound method Kernel.raw_input of <google.colab._kernel.Kernel object at 0x7febe3ce3
[3, 'print']
[2, 'Welcome to my program.\n']
[3, ';']
[3, 'print']
[2, 'Enter a number: ']
[3, ';']
[3, 'get']
[0, 'x']
[3, ';']
[3, 'if']
[0, 'x']
[3, '>']
[1, 0]
[3, 'then']
[0, 'x']
[3, '=']
[0, 'x']
[3, '*']
[1, 2]
[3, ';']
[3, 'else']
[0, 'x']
[3, '=']
[3, '-']
[0, 'x']
[3, ';']
[3, 'end']
[3, ';']
[3, 'print']
[0, 'x']
[3, ';']
```

```
print "Welcome to my program.\n";
print "Enter a number: ";
get x;
if x > 0
then
  x=x*2;
else
  x=-x;
end;
print x;

```

Figure 3: test3.txt's output.