Danmarks Tekniske Universitet

02141 Computer Science Modelling

Mandatory Assignment 3

Operational Semantics and Interpreters

Submitted by:

Fariha Nazmul

(s094747)

Part 1

In this part, we consider the While language with a repeat construct.

Question 1

The repeat construct of the While language is given as:

$$repeat\ S\ until\ b$$

A natural semantics for the repeat construct is specified as below:

$[\boldsymbol{repeat}^{\boldsymbol{tt}}_{\boldsymbol{ns}}]$

$$\frac{\langle S, s\rangle \rightarrow s'}{\langle repeat\ S\ until\ b, s\rangle \rightarrow s'} \quad if\ B[\![b]\!]s = tt$$

$[\boldsymbol{repeat}^{\boldsymbol{ff}}_{\boldsymbol{ns}}]$

$$\frac{\langle S, s\rangle \rightarrow s'\ , \langle repeat\ S\ until\ b, s'\rangle \rightarrow s''}{\langle repeat\ S\ until\ b, s\rangle \rightarrow s''} \quad if\ B[\![b]\!]s = ff$$

We have to prove that,

$$repeat\ S\ until\ b\ \ldots.(i)$$

Is semantically equivalent to

$$S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b)\ \ldots.(ii)$$

First we prove that, (i) implies (ii)

The case $[\boldsymbol{repeat}^{\boldsymbol{tt}}_{\boldsymbol{ns}}]$ **:**

$$\frac{\langle S, s\rangle \rightarrow s'}{\langle repeat\ S\ until\ b, s\rangle \rightarrow s'} \quad if\ B[\![b]\!]s = tt$$

By using $[skip_{ns}]$ and $[if^{tt}_{ns}]$, we can write a a derivation tree as

$$\frac{\langle skip, s'\rangle \rightarrow s'}{\langle if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s'\rangle \rightarrow s'} \quad if\ B[\![b]\!]s' = tt$$

By applying to $[comp_{ns}]$, we can get the following

$$\frac{\langle S, s\rangle \rightarrow s',\ \langle skip, s'\rangle \rightarrow s'}{\langle S;\ if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s'\rangle \rightarrow s'} \quad if\ B[\![b]\!]s' = tt$$

$$\Rightarrow \frac{\langle S, s\rangle \rightarrow s',\ \langle if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s'\rangle \rightarrow s'}{\langle S;\ if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s'\rangle \rightarrow s'} \quad if\ B[\![b]\!]s' = tt$$

The case $[\boldsymbol{repeat}_{\boldsymbol{ns}}^{\boldsymbol{ff}}]$ **:**

We know that

$$\frac{\langle S, s\rangle \rightarrow s' , \langle repeat\ S\ until\ b, s'\rangle \rightarrow s''}{\langle repeat\ S\ until\ b, s\rangle \rightarrow s''} \quad if\ B[\![b]\!]s = ff$$

We can think of the premises as T_1 and T_2. By applying $[if_{ns}^{tt}]$, we can get the following derivation tree for T_2.

$$\frac{\langle repeat\ S\ until\ b, s'\rangle \rightarrow s''}{\langle if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s'\rangle \rightarrow s''} \quad if\ B[\![b]\!]s = ff$$

Now, by applying $[comp_{ns}]$ to T_1 and T_2 ,we get

$$\frac{\langle S, s\rangle \rightarrow s' , \langle repeat\ S\ until\ b, s'\rangle \rightarrow s''}{\langle S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s\rangle \rightarrow s''} \quad if\ B[\![b]\!]s = ff$$

$$\Rightarrow \frac{\langle S, s\rangle \rightarrow s' , \langle if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s'\rangle \rightarrow s''}{\langle S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s\rangle \rightarrow s''} \quad if\ B[\![b]\!]s = ff$$

Now we prove the second part i.e. (ii) implies (i)

The case $[\boldsymbol{if}_{\boldsymbol{ns}}^{\boldsymbol{tt}}]$ **:**

We can use the $[comp_{ns}]$ rule to get a derivation tree that can lead us to the proof

$$\frac{\langle S, s\rangle \rightarrow s' , \langle if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s'\rangle \rightarrow s''}{\langle S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s\rangle \rightarrow s''}$$

Again, we have

$$\frac{\langle skip, s'\rangle \rightarrow s'}{\langle if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s'\rangle \rightarrow s'} \quad if\ B[\![b]\!]s = tt$$

Replacing this in the derivation tree found above, we can get

$$\frac{\langle S, s\rangle \rightarrow s' , \langle skip, s'\rangle, s'\rangle \rightarrow s'}{\langle S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s\rangle \rightarrow s'} \ if\ B[\![b]\!]s = tt$$

$$\Rightarrow \frac{\langle S, s\rangle \rightarrow s'}{\langle S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s\rangle \rightarrow s'} \ if\ B[\![b]\!]s = tt$$

$$\Rightarrow \frac{\langle S, s\rangle \rightarrow s'}{\langle repeat\ S\ until\ b, s\rangle \rightarrow s'} \ if\ B[\![b]\!]s = tt \qquad using\ [repeat_{ns}^{tt}]$$

The case $[\boldsymbol{if}_{\boldsymbol{ns}}^{\boldsymbol{ff}}]$ **:**

We can use the $[comp_{ns}]$ rule to get a derivation tree that can lead us to the proof

$$\frac{\langle S, s\rangle \rightarrow s^{'}, \langle if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s^{'}\rangle \rightarrow s^{''}}{\langle S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s\rangle \rightarrow s^{''}}\ if\ B[\![b]\!]s = ff$$

Now, we also can write,

$$\frac{\langle repeat\ S\ until\ b, s^{'}\rangle \rightarrow s^{''}}{\langle if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s'\rangle \rightarrow s^{''}}\ if\ B[\![b]\!]s = ff$$

Replacing this in the above rule and using the definition of $[repeat_{ns}^{ff}]$, we get,

$$\frac{\langle S, s\rangle \rightarrow s^{'}, \langle repeat\ S\ until\ b, s^{'}\rangle \rightarrow s^{''}}{\langle repeat\ S\ until\ b, s\rangle \rightarrow s^{''}}\ if\ B[\![b]\!]s = ff$$

Question 2

A structural operational semantics for the repeat construct is specified as below:

$[\boldsymbol{repeat}_{\boldsymbol{sos}}]$

$$\langle repeat\ S\ until\ b, s\rangle \Rightarrow \langle S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s\rangle$$

Question 3

We have to prove that the repeat construct

$$repeat\ S\ until\ b$$

Is semantically equivalent to

$$S; while\ \neg b\ do\ S$$

Proof based on **natural semantics**:

Using induction on the shape of the derivation tree, we can prove this. Let us assume,

$$\langle repeat\ S\ until\ b, s\rangle \rightarrow s^{''} \ldots . (i)$$

And

$$\langle S; while\ \neg b\ do\ S, s\rangle \rightarrow s^{''} \ldots . (ii)$$

The proof is in two parts.

First we prove (i) is equivalent to (ii).

We know that the derivation tree will be:

$$\frac{\langle S, s\rangle \rightarrow s'}{\langle repeat\ S\ until\ b, s\rangle \rightarrow s'} \quad if\ B[\![b]\!]s = tt$$

Using $\langle S, s\rangle \rightarrow s'$ as a premise, we can get another derivation tree as

$$\frac{\langle S, s\rangle \rightarrow s'}{\langle S; skip, s\rangle \rightarrow s'}$$

The statement ***skip*** can be represented as the part of the derivation tree for while-construct $while\ b'\ do\ S$ where $B[\![b']\!]s = ff$. So, in other way, it can be said that when $B[\![\neg b']\!]s = tt$. Using this, we can get,

$$\frac{\langle S, s\rangle \rightarrow s'}{\langle S; while\ \neg b\ do\ S, s\rangle \rightarrow s'} \quad if\ B[\![b]\!]s = tt$$

Again, we have

$$\frac{\langle S, s\rangle \rightarrow s^{'}, \langle repeat\ S\ until\ b, s'\rangle \rightarrow s''}{\langle repeat\ S\ until\ b, s\rangle \rightarrow s''} \quad if\ B[\![b]\!]s = ff$$

The premises can be represented as T_1 and T_2 respectively. Applying induction hypothesis on T_2 , we can get $\langle S; while\ \neg b\ do\ S, s'\rangle \rightarrow s''$. Then we get the derivation tree as

$$\frac{\langle S, s\rangle \rightarrow s^{'}, \langle S, s^{'}\rangle \rightarrow s^{'''}, \langle while\ \neg b\ do\ S, s^{'''}\rangle \rightarrow s^{''}}{\langle repeat\ S\ until\ b, s\rangle \rightarrow s^{''}}$$

$$\Rightarrow \frac{\langle S, s\rangle \rightarrow s^{'''}, \langle while\ \neg b\ do\ S, s'''\rangle \rightarrow s''}{\langle repeat\ S\ until\ b, s\rangle \rightarrow s''}$$

This is the derivation tree for $\langle S; while\ \neg b\ do\ S, s\rangle \rightarrow s'$

Using the same technique i.e. induction on the shape of the derivation tree, we can prove that (ii) is equivalent to (i).

Question 4

Extended proof of Theorem 2.26 in the text book to hold for the language with the repeat-construct:

For repeat statement S of While, we have to prove that

$$S_{ns}[\![S]\!] = S_{sos}[\![S]\!]$$

For that we have to prove the Lemma 2.27 and Lemma 2.228 in the book for repeat construct.

For **Lemma 2.27**, we have to extend the proof of

$$\langle S, s\rangle \rightarrow s' \; implies \; \langle S, s\rangle \overset{*}{\Rightarrow} s'$$

for the repeat construct.

The case $[\boldsymbol{repeat}^{\boldsymbol{tt}}_{\boldsymbol{ns}}]$ **:**

$$if \; B[\![b]\!]s = tt \; , \langle repeat \; S \; until \; b, s\rangle \rightarrow s' \; implies \; \langle S, s\rangle \rightarrow \; s'$$

Applying the induction hypothesis, it can be said that

$$\langle S, s\rangle \rightarrow s' \; implies \; \langle S, s\rangle \overset{*}{\Rightarrow} s'$$

The case $[\boldsymbol{repeat}^{\boldsymbol{ff}}_{\boldsymbol{ns}}]$ **:**

Let us assume that $\langle repeat \; S \; until \; b, s\rangle \rightarrow s''$

because $if \; B[\![b]\!]s = ff, \; \langle S, s\rangle \rightarrow s' \; and \; \langle repeat \; S \; until \; b, s'\rangle \rightarrow s''$

Applying the induction hypothesis on both of the premises, we get

$$\langle S, s\rangle \overset{*}{\Rightarrow} s' \; and \; \langle repeat \; S \; until \; b, s'\rangle \overset{*}{\Rightarrow} s''$$

So, using $[comp_{ns}]$ and $[comp_{sos}]$ we can write that

$$\langle S; repeat \; S \; until \; b, s\rangle \rightarrow s'' \; implies \; \langle S; \; repeat \; S \; until \; b, s\rangle \overset{*}{\Rightarrow} s''$$

Using $[repeat_{sos}]$ and $[if^{ff}_{sos}]$ (with $if \; B[\![b]\!]s = ff$) we get

$$\langle repeat \; S \; until \; b, s\rangle$$

$$\Rightarrow \langle S; if \; b \; then \; skip \; else \; (repeat \; S \; until \; b), s\rangle$$

$$\Rightarrow \langle S; repeat \; S \; until \; b), s\rangle$$

$$\overset{*}{\Rightarrow} s''$$

For Lemma **2.28**, we have to extend the proof of

$$\langle S, s\rangle \overset{k}{\Rightarrow} s' \; implies \; \langle S, s\rangle \rightarrow s'$$

for the repeat construct.

The case $[\boldsymbol{repeat}_{\boldsymbol{sos}}]$:

We have

$$\langle repeat\ S\ until\ b, s\rangle$$

$$\Rightarrow \langle S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s\rangle$$

$$\overset{ko}{\Rightarrow} s''$$

The induction hypothesis can be applied to the k_0 last steps of the derivation sequence and we get

$$\langle S; if\ b\ then\ skip\ else\ (repeat\ S\ until\ b), s\rangle \rightarrow s''$$

And from the previous proof, we get

$$\langle\ (repeat\ S\ until\ b), s\rangle \rightarrow s''$$

So, it follows from the extended proof of Lemma 2.27 and Lemma 2.28 that if $if\ S_{ns}[\![S]\!]s = s'\ then\ S_{sos}[\![S]\!]s = s'\ and\ vice\ versa$ for the repeat-construct.

Part 2

Here we shall construct an interpreter for the While language.

Question 1

First of all, we shall develop a natural semantics for the subset of the While language provided in Appendix A.

a)

According to the specification, arithmetic expressions are evaluated according to the judgment

$$env_v, env_M \vdash_A \langle a, sto\rangle \rightarrow v$$

and Boolean expressions are evaluated according to the judgment

$$env_v, env_M \vdash_B \langle a, sto\rangle \rightarrow v$$

To evaluate methods without side effects we have to define the operational details of arithmetic expression evaluation according to the following judgments

$[numExp_{ns}]$

$$env_v, env_M \vdash_A \langle n, sto\rangle \rightarrow v$$

$[addExp_{ns}]$

$$\frac{env_v\,, env_M \;\vdash_A \langle A_1, sto\rangle \rightarrow \; v_1 \qquad env_v\,, env_M \;\vdash_A \langle A_2, sto\rangle \rightarrow v_2}{env_v\,, env_M \;\vdash_A \langle A_1 + A_2, sto\rangle \rightarrow v_1 + v_2}$$

$[subExp_{ns}]$

$$\frac{env_v\,, env_M \;\vdash_A \langle A_1, sto\rangle \rightarrow \; v_1 \qquad env_v\,, env_M \;\vdash_A \langle A_2, sto\rangle \rightarrow v_2}{env_v\,, env_M \;\vdash_A \langle A_1 - A_2, sto\rangle \rightarrow v_1 - v_2}$$

$[timesExp_{ns}]$

$$\frac{env_v\,, env_M \;\vdash_A \langle A_1, sto\rangle \rightarrow \; v_1 \qquad env_v\,, env_M \;\vdash_A \langle A_2, sto\rangle \rightarrow v_2}{env_v\,, env_M \;\vdash_A \langle A_1 * A_2, sto\rangle \rightarrow v_1 * v_2}$$

$[call_{ns}]$

$$\frac{\begin{array}{c} env_v\,, env_M \;\vdash_A \langle A,\; sto\rangle \rightarrow v \\ \vdash_D \langle S, env_v\,, sto\rangle \rightarrow \langle env'_v, sto'\rangle \\ env'_v\,[V_1 \;\mapsto\; l], env_M \;\vdash_S \langle S\,, sto'[next \mapsto l+1\,][l \;\mapsto v]\rangle \rightarrow sto'' \\ (sto''\; o\; env'_v)(V_2) = v' \end{array}}{env_v\,, env_M \;\vdash_A \langle M(A),\; sto\rangle \rightarrow v'}$$

$$where\; sto(next) = l\; and\; env_M\,(M) = (V_1\,, V_2\,, S)$$

The operational details of boolean expression evaluation are defined according to the following judgments

$$env_v\,, env_M \;\vdash_B \langle true, sto\rangle \rightarrow v$$

$$env_v\,, env_M \;\vdash_B \langle false, sto\rangle \rightarrow v$$

$[and_{ns}]$

$$\frac{env_v\,, env_M \;\vdash_B \langle B_1, sto\rangle \rightarrow \; v_1 \qquad env_v\,, env_M \;\vdash_B \langle B_2, sto\rangle \rightarrow v_2}{env_v\,, env_M \;\vdash_B \langle B_1 \wedge B_2, sto\rangle \rightarrow v_1 \wedge v_2}$$

$[or_{ns}]$

$$\frac{env_v\,, env_M \;\vdash_B \langle B_1, sto\rangle \rightarrow \; v_1 \qquad env_v\,, env_M \;\vdash_B \langle B_2, sto\rangle \rightarrow v_2}{env_v\,, env_M \;\vdash_B \langle B_1 \vee B_2, sto\rangle \rightarrow v_1 \vee v_2}$$

$[neg_{ns}]$

$$env_v\,, env_M \;\vdash_B \langle \neg B_1 \;\; , sto\rangle \rightarrow \neg v_1$$

$[equals_{ns}]$

$$\frac{env_v, env_M \vdash_B \langle A_1, sto\rangle \rightarrow v_1 \quad env_v, env_M \vdash_B \langle A_1, sto\rangle \rightarrow v_2}{env_v, env_M \vdash_B \langle A_1 = A_2, sto\rangle \rightarrow v_1 = v_2}$$

$[Larger_{ns}]$

$$\frac{env_v, env_M \vdash_B \langle A_1, sto\rangle \rightarrow v_1 \quad env_v, env_M \vdash_B \langle A_1, sto\rangle \rightarrow v_2}{env_v, env_M \vdash_B \langle A_1 > A_2, sto\rangle \rightarrow v_1 > v_2}$$

b)

Statements are evaluated according to the judgment

$$env_v, env_M \vdash_S \langle S, sto\rangle \rightarrow sto'$$

where it is seen that statements produce side-effects but not values. The operational details of statement evaluation are defined according to the following judgment

$[skip_{ns}]$

$$env_v, env_M \vdash_S \langle skip, sto\rangle \rightarrow sto$$

$[assn_{ns}]$

$$env_v, env_M \vdash_S \langle x := a, sto\rangle \rightarrow sto[l \mapsto v]$$

$$where\ l = env_v\ x \quad and \quad env_v, env_M \vdash_A \langle a, sto\rangle \rightarrow v$$

$[seq_{ns}]$

$$\frac{env_v, env_M \vdash_S \langle S_1,\ sto\rangle \rightarrow sto' \quad env_v, env_M \vdash_S \langle S_2,\ sto'\rangle \rightarrow sto''}{env_v, env_M \vdash_S \langle S_1; S_2,\ sto\rangle \rightarrow sto'}$$

$[if_{ns}^{tt}]$

$$\frac{env_v, env_M \vdash_S \langle S_1,\ sto\rangle \rightarrow sto'}{env_v, env_M \vdash_S \langle if\ b\ then\ S_1\ else\ S_2\ fi,\ sto\rangle \rightarrow sto'} \quad if\ env_v, env_M \vdash_B \langle b, sto\rangle \rightarrow true$$

$[if_{ns}^{ff}]$

$$\frac{env_v, env_M \vdash_S \langle S_2,\ sto\rangle \rightarrow sto'}{env_v, env_M \vdash_S \langle if\ b\ then\ S_1\ else\ S_2\ fi,\ sto\rangle \rightarrow sto'} \quad if\ env_v, env_M \vdash_B \langle b, sto\rangle \rightarrow false$$

$[while_{ns}^{tt}]$

$$\frac{env_v\,, env_M \vdash_S \langle S_1,\ sto\rangle \rightarrow sto'\,,\quad env_v\,, env_M \vdash_S \langle while\ b\ do\ S_1\ od,\ sto'\rangle \rightarrow sto''}{env_v\,, env_M \vdash_S \langle while\ b\ do\ S_1 od,\ sto\rangle \rightarrow sto''}$$
$$if\ env_v\,, env_M \vdash_B \langle b, sto\rangle \rightarrow true$$

$[while_{ns}^{ff}]$

$$env_v\,, env_M \vdash_S \langle while\ b\ do\ S_1 od, sto\rangle \rightarrow sto$$
$$if\ env_v\,, env_M \vdash_B \langle b, sto\rangle \rightarrow false$$

c)

Declarations do not cause any evaluation but they update/extend the appropriate environments and stores with new values. We update the variable environments and stores to account for the assigned variables in accordance with the following judgment

$$\vdash_D \langle S, env_v\,, sto\rangle \rightarrow \langle env'_v, sto'\rangle$$

A variable assignment causes the variable environment and store to be updated in one of the two ways. In the first case, if the variable is declared for the first time in the environment of context, then a new location is bound to the variable and the environment and store is updated accordingly.

$$\frac{env_v\,, env_M \vdash_D \langle x := a,\ env_v,\ sto\rangle \rightarrow\ env'_v\,, env_M \vdash_S \langle S,\ ,sto[next \mapsto l+1\ ][l\ \mapsto v]\rangle \rightarrow sto'}{\vdash_D \langle S, env_v\,, sto\rangle \rightarrow \langle env'_v, sto'\rangle}$$
$$where\ l = sto(next),\ env'_v\ =\ env_v[x\ \mapsto\ l]\ and\ v =\ env_v\,, env_M \vdash_A \langle a, sto\rangle \rightarrow v$$

On the other hand, if the variable is already declared and placed in the environment then only the store has to be updated according to the new value. This can be done using the following judgment

$$\frac{env_v\,, env_M \vdash_D \langle x := a,\ env_v,\ sto\rangle \rightarrow\ env_v\,, env_M \vdash_S \langle S,\ ,sto[l\ \mapsto v]\rangle \rightarrow sto'}{\vdash_D \langle S, env_v\,, sto\rangle \rightarrow \langle env'_v, sto'\rangle}$$
$$where\ l =\ env_v\ x,\ \ and\ v =\ env_v\,, env_M \vdash_A \langle a, sto\rangle \rightarrow v$$

d)

For method declarations the update is performed by the function

$$upd_M : MDec \rightarrow Env_M$$

This function can be expressed as following

$$upd_M\ ([\,]) \rightarrow \emptyset$$

$$upd_M\ (M = \{\ input(V_1);\ S; output(V_2)\}; D_M) \rightarrow\ upd_M\ (D_M)[M\ \mapsto (V_1, V_2, S)]$$

e)

For program we follow the judgments of the form

$$\vdash_P\ P\ \rightarrow\ sto$$

defined by the single rule

$$\frac{\vdash_D\ \langle S, [\ \ ],\ [\,]\rangle \rightarrow \langle env_v, sto\rangle \qquad env_v\ ,\ env_M\ \vdash_S\ \langle S, sto\rangle\ \rightarrow\ sto'}{\vdash_P\ D_M\, S\ \rightarrow\ sto'}$$

$$where\ env_M =\ upd_M(D_M)(env_v)\ and\ occurences\ of\ [\,]denote\ empty\ environments/stores$$

f)

The format of the method environments is defined as

$$env_M\ \in Env_{M=}\ MetID\ \rightarrow Var \times\ Var\ \times Stm$$

and the update for method declarations is performed by the function

$$upd_M : MDec\ \rightarrow Env_M$$

The syntax of the program is given as

$$P ::=\ D_M\, S$$

Since all the methods are declared before any method call evaluations, the environment method will be always updated with the definition of the method before method call. Again, the methods are evaluated without side effects in the evaluation judgments of the semantics. So, this will always enable the definition of recursive and mutually recursive methods.

Question 2

In this part we have implemented the interpreter for the While language given in Appendix A. The implementation of the interpreter is done in a Java class **Interpreter** in a sub-directory **Interpreter** of the working directory. The class encapsulates the four semantic functions and exposes a main method that, given the file name of a valid While program, evaluates the program and displays the final state to standard out.

Here, the input of the interpreter is the internal representations of While programs created by the parser of assignment 2 with some extension. The provided Java files **Symbol.java** and **Table.java** are used for the implementation of the domains that are required for the interpretation process.

The interpreter checks for valid variable, valid input/output, valid method names and provides some error messages according to arisen error.

The implemented interpreter has been tested for some test cases.

Test case 1:

Here the input file is **test.w** and it contains the implementation of factorial function using a while loop. The interpreter can generate the final state correctly.

```
fact = {
            input(n);
            (
               m:=1;
               while (n>1)
               do
                        (m:=(m*n);
                        n:= (n-1))
               od
            );
            output(m)
         };

(i:=5;
f:= fact(i))
```

Output:

```
i -> 5
f -> 120
```

Test case 2:

The input file **test1.w** contains the mutually recursive definition of factorial and the interpreter can handle the mutually recursive function definition.

```
factorial1 = {
                  input(n);
                  if (n=1) then
                        result:=n
                  else
                        result:= (n * factorial2((n-1)))
                  fi;
                  output(result)
            };

factorial2 = {
                  input(n);
                  if (n=1) then
                        result:=n
                  else
                        result:= (n * factorial1((n-1)))
                  fi;
                  output(result)
            };

(n := 7;
x:=factorial1(n))
```

Output:

```
n -> 7
x -> 5040
```

Test Case 3:

The input file **test2.w** contains a dummy function. The output of the function is not defined and the input to the function call is invalid. The interpreter can show meaning error messages for these errors.

```
M2 = {
            input(in);
            skip;
            output(out)
     };

m:=M2(i);
```

Output:

```
The variable i is not initialized.
The ouput variable out is not defined.
m -> 0
```

Collaborated with:

Sameer K.C. (s094746)