Work Time: 2hours and 30 minutes

If a problem implementation does not compile or does not run you will get 0 points for that problem (that means no default points)!!!

1. **(0.5p by default) Problem 1: Implement Repeat...Until statement in Toy Language.**

a. (2.75p). Define the new statement:

repeat stmt1 until exp2

The statement stmt1 is executed as long as the expression exp2 is not true. Its execution on the ExeStack is the following:

- pop the statement
- create the following statement: stmt1;(while(!exp2) stmt1)
- push the new statement on the stack

The typecheck method of repeat statement verifies if exp2 has the type bool and also typecheck the statement stmt1.

b. (with a working GUI 1.75p, with a working text UI 0.25p). Show the step-by-step execution of the following program. At each step display the content of each program state (all the structures of the program state). The step-by-step execution must be displayed on the screen and also must be saved into a readable log text file. The following program must be hard coded in your implementation:

int v; int x; int y; v=0;
(repeat (fork(print(v);v=v-1);v=v+1) until v==3);
x=1;nop;y=3;nop;
print(v*10)

The final Out should be {0,1,2,30}

2. **(0.5p by default). Problem 2: Implement a CyclicBarrier mechanism in ToyLanguage.**

a. (0.5p). Inside PrgState, define a new global table (global means it is similar to Heap, FileTable and Out tables and it is shared among different threads), BarrierTable that maps an integer to a pair: an integer and a list of integers. BarrierTable must be supported by all of the previous statements. It must be implemented in the same manner as Heap, namely an interface and a class which implements the interface. *Note that the lookup and the update of the BarrierTable must be atomic operations, that means they cannot be interrupted by the execution of the other PrgStates. Therefore you must use the lock mechanisms of the host language Java over the BarrierTable in order to read and write the values of the BarrierTable entrances .*

b. (0.75p). Define a new statement

newBarrier(var,exp)

which creates a new barrier into the BarrierTable. The statement execution rule is as follows:

Stack1={newBarrier(var, exp)| Stmt2|...}
SymTable1
Out1
Heap1
FileTable1
BarrierTable1

==>

Stack2={Stmt2|...}
Out2=Out1
Heap2=Heap1
FileTable2=FileTable1

- evaluate the expression exp using SymTable1 and Heap1 and the evaluation result must be an integer . If it is not an integer then print an error and stop the execution. Let be Nr the result of this evaluation.

BarrierTable2 = BarrierTable1 synchronizedUnion
{newfreelocation ->(Nr,empty list)}
if var exists in SymTable1 and var has the type int *then*
SymTable2 = update(SymTable1,var, newfreelocation)
else print an error and stop the execution

Note that you must use the lock mechanisms of the host language Java over the BarrierTable in order to add a new barrier to the table.

c. (0.75p). Define the new statement

await(var)

where var represents a variable from SymTable which is the key for an entry into the BarrierTable. Its execution on the ExeStack is the following:

- pop the statement
- foundIndex=lookup(SymTable,var). If var is not in SymTable or has not the int type print an error message and terminate the execution.

- *if* foundIndex is not an index in the BarrierTable *then*
print an error message and terminate the execution
else
- retrieve the entry for that foundIndex, as BarrierTable[foundIndex]==
(N1,List1)
- compute the length of that list List1 as NL=length(L1)
- *if* (N1>NL) *then*
if(the identifier of the current PrgState is in L1) *then*
- push back await(var) on the ExeStack
else
- add the id of the current PrgState to L1
- push back await(var) on the ExeStack
else
do nothing

d.(0.75). Implement the method typecheck for the statement newBarrier(var, exp) to verify if both var and exp have the type int. Implement the method typecheck for the statement await(var) to verify if var has the type int.
e. (1p). Extend your GUI to suport step-by-step execution of the new added features. To represent the BarrierTable please use a TableView with three columns: an index, a value and a list of values.
f. (with a working GUI 0.75p, with a working text UI 0.25p). Show the step-by-step execution of the following program. At each step display the content of each program state (all the structures of the program state). The step-by-step execution must be displayed on the screen and also must be saved into a readable log text file. The following program must be hard coded in your implementation.
Ref int v1; Ref int v2; Ref int v3; int cnt;
new(v1,2);new(v2,3);new(v3,4);newBarrier(cnt,rH(v2));
fork(await(cnt);wh(v1,rh(v1)*10);print(rh(v1)));
fork(await(cnt);wh(v2,rh(v2)*10);wh(v2,rh(v2)*10);print(rh(v2)));
await(cnt);
print(rH(v3))
The final Out should be {4,20,300}.