Traveller's Transpiler (Shaurya's Group I)

Presented by :-
Avni Maheshwari (210241)
Srushti Srivastava (211060)
Khushi Gupta (220531)
Manvi Bengani (220629)

General Variables to be used:

1) String s : to store the code in super stack
2) int cond : to maintain compass(initial value =0)
3) int cond_f : for implementing if and fi commands (loops)

General state of books:

1) mem_3 always points to the second last location where input is made
2) mem_2 always points to the last location where input is made
3) mem_1 always points to the second last location where input is made
4) Before the initialization of the stack, an EOS character is maintained.

The commands of SuperStack esolang are transpiled as follows:

- 123:
  For this we have defined function num so as to push the absolute value of the number declared in the super stack code. In function num we defined a vector so as to store the commands of IITK Traveller to be implemented so as to reach the required IITK Traveller code.
  Function num is used as a recursive function.
  For a number x,
  Base cases:
  x=0: nothing pushed in the vector
  x=1,2,3: pushed oat_stairs_2 x times
  Otherwise x's square root is calculated and type casted as an integer. We have defined the variable dev which is the deviation of x from the nearest perfect square. We use eshop_2 to square the number and oat_stairs_2 dev number of times to reach the required number.
  We use this recursively on the square root of x till we reach the base cases. The vector is then iterated and printed backwards.
  If we have a negative number then first we push the absolute value of the number by the above method. Then we shift mem_1 to the location after the stack top and mem_3 on the stack top. Then we use hall_5 to store the difference of mem_1 and mem_2 in mem_3. Since value in mem_1 is zero, we have the negative of the number. In the end the position of the books are adjusted.
- Add:
  mem_3 should contain the addition of the two numbers having pointers mem_2 and mem_1. mem_2 value is then changed to 0. And then the positions of the three books

mem_1, mem_2, mem_3 are accordingly adjusted as the general state we assumed above.

- Sub:
  mem_3 then contains the subtraction of the two numbers by using hall_5 and then the process is the same as one used above.
- Mul:
  Similarly , we used hall_3 command to multiply the two numbers and repeated the process.
- Div:
  And here, we used hall_12 for division.
- Mod:
  Here we used rm_1 to shift the position of mem_1 forward by 1 position. Then we increased the cond value so as to use rm_1 again. Then we equated the values of mem_1 and mem_3 and then divided the two numbers, then multiplied the two values at 1,2 then made the mem_2 integer = 0 and adjusted the positions of the books as per our above assumptions.
- Quit:
  Just output “finish”.
- And:
  Firstly, mem_1 is shifted to a temporary location(just after the top of stack in the infinite tape) the top of the stack is compared with 0 using lecture_hall_eq and then popped.
  If the top of the stack was 0, the new top is made zero and the position of the books is accordingly adjusted.
  If top of the stack was not zero, the new top is again compared with 0 and
  - ❖ If its 0, the positions of the books are adjusted
  - ❖ If it's not 0, it's made 1 and then positions of the books are adjusted
- Pop:
  We first make the mem_2 value = 0 and adjust the positions of the book as per our assumptions (we shift them back by 1).
- Dup:
  We first shift the second and third book by 1 at the back, and equate the values at mem_2 and mem_3. Then adjust the books’ position as per our assumption.
- Not:
  firstly mem_1 is shifted to a temporary location, then value at mem_1 is equated to 0. Then we compare the 0 (mem[mem_1]) by the value at mem_2 by lecture_hall, if it's true then we convert the value to 1, if it’s false we convert it to 0 and then the positions of the books are adjusted.
- Or:
  Here also firstly mem_1 is shifted to a temporary location(just after the top of stack in the infinite tape) the top of the stack is compared with 0 using lecture_hall_eq and then popped.
  If the top of the stack was 1, the new top is made 1 and the position of the books is accordingly adjusted.
  If top of the stack was zero, the new top is again compared with 0 and
  - ❖ If its 0, the positions of the books are adjusted

- If it's not 0, it's made 1 and then positions of the books are adjusted

- Xor:
  Here firstly mem_1 is shifted to a temporary location(just after the top of stack in the infinite tape) the top of the stack is compared with 0 using lecture_hall_eq and
  - If its 0, then no change is made in its value
  - If it's not 0, then its value is made 1

  Similarly, the second value from the top is manipulated.
  Then the positions of mem_1 and mem_2 are set so as to compare these.
  - If these values are equal, top is popped and the new top is made 0
  - If these values are not equal , top is popped and the new top is made 1

  At last the positions of the books are adjusted.
- Nand:
  Suitable changes in And are made.
- Cycle:
  Here we used the infinite property of the tape.We moved mem_1 to the location before the first value in the stack and mem_3 to the top of the stack and copied mem_3 in mem_1 and popped mem_3. In the end the positions of the books are adjusted.
- Rcycle:
  Here we used the infinite property of the tape. We moved mem_1 to the first value in the stack and mem_3 to the location just after the top position of the stack and copied mem_1 in mem_3 and popped mem_1. In the end, the positions of the books are adjusted.
- Output:
  First we use iit_gate_out_2 to output the value on mem_2. Then we use hall_13_2 to equate the value on mem_2 to 0. In the end the position of the books are adjusted.
- Input:
  First we use iit_gate_in_2 to take integer input and store the value on mem[mem_2]. Then the positions of the books are adjusted.
- Inputascii:
  Firstly we input the values of the stack by using airstrip_land_2 and then the tape is reversed. The values are copied in desired order from 2 locations after the top of the stack and then the initial values are swapped leaving behind the desired stack. Then extra values are popped. And the position of the books are adjusted.
- Outputascii:
  First we use nankari_gate_out_2 to print the ascii character corresponding to the ascii value stored at mem[mem_2]. Then we use hall_13_2 to equate the value on mem_2 to 0. In the end the position of the books are adjusted.
- Swap:
  First we shift mem_3 to its next position. Then mem_2 is also shifted to its next position. Then we use mt_2_3 to store the value in mem_3 in mem_2. Then we use mt_3_1 to store the value in mem_1 in mem_3. Then mem_3 is shifted 1 forward and mem_2 1 backward. Then we use mt_1_3 to store the value in mem_3 in mem_1. Then we equate the value in mem_3 to 0. In the end the position of the books are adjusted.
- Rev:

Here the infinite property of the tape is used. The values are copied in reverse order from 2 locations after the top of the stack and then the initials values are popped leaving behind the desired stack.In the end, the position of the books are adjusted

- If and fi:
  Here firstly mem_1 is shifted to a temporary location (just after the top of stack in the infinite tape) and then the top of the stack is compared with 0 using the Lecture_hall_eq landmark in IITK Traveller and the loop is set. For the body of the loop at first the positions of the books are adjusted according to general state and then according to the code of SuperStack the commands for IITK Traveller are printed. On the encounter of "Fi" the positions of the books and the condition value is set as suitable for loop formation. In the end, after loop is set, the position of the books is adjusted as stated in the general state of books.
- Debug:
  Assumption: the entire stack is printed with each value separated by space.
  First mem_1 is shifted to initial value and then the loop is set using events_1 to print the stack values.In the end, the position of the books is adjusted.