Autocomplete (100 คะแนน)

1 second, 256 megabytes

ภาษา TUMSO (The Untyped Microlanguage without Strings and Objects) เป็นภาษาที่ใช้สำหรับการคำนวณจำนวนเต็ม

ภาษา TUMSO

ไวยากรณ์ของภาษา TUMSO มีดังนี้ (ให้อ่าน `::=` ว่า "คือ" และอ่าน `|` ว่า "หรือ")

```
<program> ::= <expression>

<expression> ::= <number>
               | TRUE
               | FALSE
               | <identifier>
               | (CALL <expression> <expression>)
               | (FUNCTION (<identifier>) <expression>)
               | (LET <identifier> = <expression> IN <expression>)
               | (IF <expression> THEN <expression> ELSE <expression>)
               | (PLUS <expression> <expression>)
               | (MINUS <expression> <expression>)
               | (IS_EQUAL <expression> <expression>)
```

- `<number>` เป็นจำนวนเต็ม ตัวอย่างเช่น `-1234`
- `<identifier>` เป็นชื่อตัวแปร ประกอบไปด้วยอักขระในช่วง `a` ถึง `z` และเครื่องหมาย `_` เช่น `the_sum_of_two_numbers`

โทเคนในภาษานี้ได้แก่ ตัวเลข `TRUE` `FALSE` `CALL` `FUNCTION` `LET` `=` `IN` `IF` `THEN` `ELSE` `PLUS` `MINUS` `IS_EQUAL` และตัวแปร โดยโทเคนจะต้องถูกแยกออกจากกันด้วยช่องว่าง วงเล็บ หรืออักขระขึ้นบรรทัด

ความหมายของโครงสร้างทางภาษา

- สำหรับตัวเลข และ `TRUE` กับ `FALSE` จะได้ค่าผลลัพธ์เป็นเป็นค่าตัวเลข ค่า `TRUE` หรือค่า `FALSE` นั้น ๆ
- สำหรับตัวแปร จะได้ค่าผลลัพธ์เป็นค่าที่ตัวแปรนั้นเก็บอยู่
- สำหรับ `(CALL <f> <a>)` ค่าผลลัพธ์ของ `<f>` ควรจะเป็นค่าฟังก์ชัน และจะได้ค่าผลลัพธ์เป็นค่าของการเรียกใช้ฟังก์ชันนั้นด้วยพารามิเตอร์ซึ่งก็คือค่าผลลัพธ์ของ `<a>`
- สำหรับ `(FUNCTION (<a>) <e>)` จะได้ค่าผลลัพธ์เป็นค่าฟังก์ชันนั้น ๆ ที่มีพารามิเตอร์คือ `<a>` และเมื่อมีการเรียกฟังก์ชันด้วยพารามิเตอร์ `<v>` จะสร้างตัวแปร `<a>` ขึ้นมาเก็บค่า `<v>` และได้ค่าผลลัพธ์เป็น `<e>` (โดยที่ `<a>` สามารถถูกอ้างถึงได้ใน `<e>`)
- สำหรับ `(LET <x> = <v> IN <e>)` จะสร้างตัวแปร `<x>` ขึ้นมาเก็บค่าของ `<v>` และได้ค่าผลลัพธ์เป็น `<e>` (โดยที่ `<x>` สามารถถูกอ้างถึงได้ในทั้ง `<v>` และ `<e>`)
- สำหรับ `(IF <cond> THEN <then> ELSE <else>)` หาก `<cond>` มีค่าคือผลลัพธ์คือ `TRUE` จะได้ค่าผลลัพธ์เป็น `<then>` แต่หาก `<cond>` มีค่าคือผลลัพธ์คือ `FALSE` จะได้ค่าผลลัพธ์เป็น `<else>`
- สำหรับ `(PLUS <a> <b>)` ค่าผลลัพธ์ของ `<a>` และ `<b>` ควรจะเป็นตัวเลข และจะได้ค่าผลลัพธ์เป็น `<a>` $+$ `<b>`
- สำหรับ `(MINUS <a> <b>)` ค่าผลลัพธ์ของ `<a>` และ `<b>` ควรจะเป็นตัวเลข และจะได้ค่าผลลัพธ์เป็น `<a>` $-$ `<b>`
- สำหรับ `(IS_EQUAL <a> <b>)` ค่าผลลัพธ์ของ `<a>` และ `<b>` ควรจะเป็นตัวเลข และจะได้ค่าผลลัพธ์เป็น `TRUE` หาก `<a>` $=$ `<b>` และจะได้ค่าผลลัพธ์เป็น `FALSE` หาก `<a>` $\neq$ `<b>`

ตัวอย่างโปรแกรมในภาษา TUMSO

โปรแกรม	ผลลัพธ์
`FALSE`	`FALSE`

`(LET a = 1 IN`
`  (PLUS`
`  (LET a = 2 IN a)`
`  a))`
	`3`

`(LET mult = (FUNCTION (a)`
`  (FUNCTION (b)`
`  (IF (IS_EQUAL 0 a)`
`  THEN 0`
`  ELSE (PLUS b (CALL (CALL mult b)`
`  (MINUS a 1)))))`
`  ) IN`
`  (LET fact = (FUNCTION (n)`
`  (IF (IS_EQUAL 0 n)`
`  THEN 1`
`  ELSE (CALL (CALL mult n)`
`  (CALL fact`
`  (MINUS n 1)))))`
`  IN`
`  (CALL fact 5)))`
	`120`

`a`	`Error: `a' is undefined`
`(CALL 1 2)`	`Error: type mismatch`

Task

ปัญหาการเรียกใช้ตัวแปรที่ไม่ได้ถูกนิยามไว้เป็นปัญหาที่พบได้ในภาษาโปรแกรมส่วนใหญ่ รวมถึงภาษา TUMSO ด้วย (ดูตัวอย่างโปรแกรมที่ 4 เป็นต้น) บาง IDE (Integrated development environment) เช่น Eclipse ของภาษา Java มีเครื่องมือแนะนำ*ตัวแปรที่สามารถใช้ได้* ในตำแหน่งที่เคอร์เซอร์กำลังอยู่ เพื่อที่คุณจะได้ไม่เขียนโปรแกรมผิดตั้งแต่แรก

รูปที่ 1: ตัวอย่างโปรแกรม Eclipse ที่แนะนำตัวแปรที่สามารถใช้งานได้ในตำแหน่งที่เคอร์เซอร์อยู่สำหรับภาษา Java

หน้าที่ของคุณคือให้เขียนโปรแกรมรับโค้ดที่ไม่สมบูรณ์ในภาษา TUMSO โดยมีสัญลักษณ์ `#` อยู่หนึ่งที่ในตำแหน่ง `<expression>` (แสดงถึงเคอร์เซอร์ในโปรแกรมที่กำลังเขียนอยู่) และแสดงผลตัวแปรทั้งหมดที่สามารถใช้ได้ที่ตำแหน่ง `#`

ข้อมูลนำเข้า

ประกอบไปด้วยโค้ดที่ไม่สมบูรณ์ในภาษา TUMSO ซึ่งมีอักขระ `#` อยู่หนึ่งที่ในตำแหน่ง `<expression>` และหากแทนที่อักขระ `#` ในโค้ดนี้ด้วย `<expression>` ใด ๆ (เช่น `0`) จะทำให้กลายเป็นโปรแกรมที่มีไวยากรณ์ถูกต้องในภาษา TUMSO โค้ดนี้จะมีความยาวกี่บรรทัดก็ได้

ข้อมูลส่งออก

มี n บรรทัด โดย n คือจำนวนตัวแปรที่สามารถใช้ได้ในตำแหน่ง `#` และแต่ละบรรทัดมีชื่อของตัวแปรที่สามารถใช้ได้ดังกล่าวในลำดับพจนานุกรม (lexicographic order)

Constraints

โค้ดที่ให้จะมีขนาดไม่เกิน 10^6 ไบต์

ตัวอย่างข้อมูลนำเข้าและข้อมูลส่งออก

Input	Output

`(LET mult = (FUNCTION (a)`
`(FUNCTION (b)`
`(IF (IS_EQUAL 0 a)`
`THEN 0`
`ELSE (PLUS b (CALL (CALL mult b)`
`(MINUS a 1)))))`
`) IN`
`(LET fact = (FUNCTION (n)`
`(IF (IS_EQUAL 0 n)`
`THEN 1`
`ELSE (CALL (CALL mult n)`
`(CALL fact`
`(MINUS n 1)))))`
`IN`
`(CALL fact #)))`
	`factorial`
	`mult`

`(IF TRUE THEN a ELSE (LET x = 1 IN #))`	`x`