วาดฝันจินตนาการ

ในตอนที่ฉันมองขึ้นไปบนท้องฟ้า ฉันเห็นดวงดาวอันสวยงาม ในตอนที่ฉันมองกลับมา ฉันเห็นตนเองอยู่บนพื้นดิน

วันหนึ่ง ฉันได้มีโอกาสไปเที่ยวที่ชายหาด ฉันเล่นทราย ฉันก่อประสาททราย ฉันลองหยิบหินขึ้นมา โยนลงไปบนพื้นทราย ฉันหยิบหินขึ้นมาอีกหลายก้อน ก่อกันจนเป็นฐานที่ยิ่งใหญ่ ให้กับประสาททราย ถึงแม้ฉันจะสามารถสร้างประสาททรายได้ แต่เมื่อคลื่นทะเลพัดมา ประสาททรายก็ค่อย ๆ สลายหายไป

ฉันเห็นกิ่งไม้ และเชือก อยู่บนชายหาด ฉันปักกิ่งไม้ลงไปบนพื้นทราย จนเป็นหลุม ฉันเอาเชือกผูกกับกิ่งไม้ และใช้นิ้วของฉัน ลากเส้นรอบกิ่งไม้ กลายเป็นวงกลม

ฉันเห็น จุด สองจุด ที่อยู่ห่างกัน ฉันจึงลากเส้นเชื่อมทั้งสองจุดเข้าด้วยกัน

ฉันเห็น เส้น ฉันเห็น วงกลม ฉันจึงรู้ว่าเส้นกับวงกลมนั้นตัดกันตรงไหนบ้าง

งานของคุณ

ในระบบพิกัดฉาก หรือระบบพิกัดคาร์ทีเซียน (Cartesian coordinate system) จุดแต่ละจุดจะมีพิกัด (x, y) ซึ่งเป็น**จำนวนจริง** (อาจไม่ใช่จำนวนเต็ม) คุณรู้ว่าพิกัดของจุด `p0` อยู่ที่ตำแหน่ง $(0, 0)$ และรู้ว่าพิกัดของจุด `p1` อยู่ที่ตำแหน่ง $(0, 1)$ อย่างไรก็ตาม คุณอยากสร้างจุดที่พิกัด (x_t, y_t) โดยใช้เพียงวงเวียน (compass) และสันตรง (straightedge) เท่านั้น งานของคุณคือให้หาวิธีการสร้างจุด (x_t, y_t) โดยใช้จำนวนคำสั่งไม่เกิน $5\,000$ คำสั่ง

หมายเหตุ หากทำการส่งไปแล้วตัวตรวจไม่สามารถตรวจสำเร็จภายใน 2 นาที กรุณาทำการแจ้งผู้คุมสอบ

รายละเอียดการเขียนโปรแกรม

ชนิดข้อมูลสำหรับจุด เส้น และ วงกลม

ในข้อนี้จะมีการให้ `class` นิยามดังนี้

สำหรับ จุด:

```
class point {
public:
  int index;
  long double x, y;
  point() : index(-1), x(0), y(0) {}
  point(int idx, long double x, long double y) : index(idx), x(x), y(
y) {}
  point(const point &other) : index(other.index), x(other.x), y(other
.y) {}
};
```

คุณสามารถเรียกใช้งาน `index`, `x`, `y` ได้ โดยจะแสดงถึง ดัชนีตำแหน่ง พิกัดตามแกน `x` และพิกัดตามแกน `y` ตามลำดับ โดยค่าของ `x` และ `y` จะเป็นเพียง **ค่าประมาณ** เท่านั้น

สำหรับ เส้นตรง:

```
class line {
public:
  int index;
  point p1, p2;
  line() : index(-1), p1(), p2() {}
  line(int idx, point p1, point p2) : index(idx), p1(p1), p2(p2) {}
};
```

คุณสามารถเรียกใช้งาน `index`, `p1`, `p2` ได้ โดยจะแสดงถึง ดัชนีตำแหน่ง จุด `p1` และจุด `p2` แล้วเส้นตรงเส้นนี้จะเป็นเส้นตรง**ยาวไม่จำกัด**

สำหรับ วงกลม:

```
class circle {
public:
  int index;
  point p;
  long double r;
  circle() : index(-1), p(), r(0) {}
  circle(int idx, point p, long double r) : index(idx), p(p), r(r) {}
};
```

คุณสามารถเรียกใช้งาน `index`, `p`, `r` ได้ โดยจะแสดงถึง ดัชนีตำแหน่ง จุดศูนย์กลางของวงกลม และรัศมีวงกลม

สตริงนิพจน์

สำหรับการเขียนฟังก์ชันในขั้นตอนถัดไป สตริงที่ให้มาจะเป็นสตริงนิพจน์ เราจะกล่าวว่า `s` เป็นสตริงนิพจน์ก็ต่อ

เมื่อเงื่อนไขอย่างใดอย่างหนึ่งต่อไปนี้เป็นจริง:

- `s` เป็นเลขโดดของจำนวนเต็มระหว่าง 0 ถึง 9
- `s` อยู่ในรูป `SQRT(t)` สำหรับบางสตริงนิพจน์ `t`
- `s` อยู่ในรูป `ADD(t1,t2)` สำหรับบางสตริงนิพจน์ `t1` และ `t2`
- `s` อยู่ในรูป `SUB(t1,t2)` สำหรับบางสตริงนิพจน์ `t1` และ `t2`
- `s` อยู่ในรูป `MUL(t1,t2)` สำหรับบางสตริงนิพจน์ `t1` และ `t2`
- `s` อยู่ในรูป `DIV(t1,t2)` สำหรับบางสตริงนิพจน์ `t1` และ `t2`

โดยเมื่อคำนวณตามวิธีการทางคณิตศาสตร์แล้วจะได้ผลลัพธ์ออกมาเป็นจำนวนจริง ฟังก์ชันภายในนิพจน์ นิยามดังนี้:

- `SQRT` รับจำนวนจริง $x \geq 0$ แล้วคืนค่า $\sqrt{x}$
- `ADD` รับจำนวนจริง x และ y แล้วคืนค่า $x + y$
- `SUB` รับจำนวนจริง x และ y แล้วคืนค่า $x - y$
- `MUL` รับจำนวนจริง x และ y แล้วคืนค่า xy
- `DIV` รับจำนวนจริง x และ y โดยที่ $y \neq 0$ แล้วคืนค่า $\frac{x}{y}$

การเขียนฟังก์ชัน

คุณจะต้องเขียนฟังก์ชันดังต่อไปนี้

```
point draw(string x, string y)
```

- ฟังก์ชันนี้จะถูกเรียกเพียงครั้งเดียว
- สตริง `x` จะเป็นสตริงนิพจน์ ที่เมื่อคำนวณออกมาแล้วจะมีค่าเท่ากับ x_t
- สตริง `y` จะเป็นสตริงนิพจน์ ที่เมื่อคำนวณออกมาแล้วจะมีค่าเท่ากับ y_t
- สตริงนิพจน์ `x` และ `y` จะประกอบด้วยคำสั่งทางคณิตศาสตร์ไม่เกิน 7 ครั้งเท่านั้น
- ฟังก์ชันนี้จะต้องคืนค่าจุดที่เกิดจากการสร้างด้วยวงเวียนและสันตรง จากจุดเริ่มต้น `p0` กับ `p1` โดยจุดที่คืนออกมา เมื่อใช้วงเวียนและสันตรงที่มีความแม่นยำ (accuracy) สูงมากจะต้องได้เป็นจุด (x_t, y_t) อย่างเที่ยงตรง (precise)
- ฟังก์ชันนี้จะสามารถเรียกฟังก์ชันทั้งหกดังต่อไปนี้ได้

```
circle compass(point p1, point p2)
```

- ฟังก์ชันนี้จะคืนค่าวงกลมที่เกิดจากการปักวงเวียนไปที่จุดศูนย์กลาง `p1` แล้วลากจากจุด `p2` ไปรอบวงจนกลับมาที่เดิม

```
line straightedge(point p1, point p2)
```

- ฟังก์ชันนี้จะคืนค่าเส้นตรงยาวไม่จำกัด ที่เกิดจากการลากเส้นเชื่อมระหว่างจุด `p1` กับ `p2`

```
point intersection(line l1, line l2)
```

- ฟังก์ชันนี้จะคืนค่าจุดที่เป็นจุดตัดระหว่างเส้นตรง `l1` และเส้นตรง `l2`

```
point intersection(line l, circle c, int idx)
```

- ฟังก์ชันนี้จะคืนค่าจุดที่เป็นจุดตัดระหว่างเส้นตรง `l` และวงกลม `c`
- เนื่องจากจุดตัดอาจมีหลายค่า จึงคืนค่าเฉพาะจุดตัดที่ `idx` มา (สามารถใส่ค่า `idx` เป็น 0 หรือ 1 ก็ได้ แต่หากตัดจุดเดียวจะไม่สามารถกำหนดค่าเป็น 1 ได้)

```
point intersection(circle c, line l, int idx)
```

- ฟังก์ชันนี้จะคืนค่าจุดที่เป็นจุดตัดระหว่างเส้นตรง `l` และวงกลม `c`
- เนื่องจากจุดตัดอาจมีหลายค่า จึงคืนค่าเฉพาะจุดตัดที่ `idx` มา (สามารถใส่ค่า `idx` เป็น 0 หรือ 1 ก็ได้ แต่หากตัดจุดเดียวจะไม่สามารถกำหนดค่าเป็น 1 ได้)

```
point intersection(circle c1, circle c2, int idx)
```

- ฟังก์ชันนี้จะคืนค่าจุดที่เป็นจุดตัดระหว่างวงกลม `c1` และวงกลม `c2`
- เนื่องจากจุดตัดอาจมีหลายค่า จึงคืนค่าเฉพาะจุดตัดที่ `idx` มา (สามารถใส่ค่า `idx` เป็น 0 หรือ 1 ก็ได้ แต่หากตัดจุดเดียวจะไม่สามารถกำหนดค่าเป็น 1 ได้)

หมายเหตุ

- การเรียกใช้ฟังก์ชันทั้งหกนี้ สามารถเรียกใช้งานรวมกันได้ไม่เกิน $5\,000$ ครั้งเท่านั้น
- นอกจากนี้ สำหรับการเรียกใช้ฟังก์ชันเหล่านี้ ข้อมูลทั้งประเภท `point`, `line` และ `circle` จะต้องเป็นข้อมูลที่ตรงตามที่ได้รับจากครั้งก่อน (กล่าวคือ ไม่สามารถแก้ `index` เป็นค่าอื่น หรือแก้ค่าพิกัดให้ไม่ตรงตามเดิมได้)

การช่วยเหลือด้านการอ่านนิพจน์

นอกจากฟังก์ชันทั้งหกที่สามารถเรียกใช้งานได้แล้ว จะสามารถเรียกใช้งานฟังก์ชันช่วยเหลือเพิ่มเติมได้อีกหนึ่งฟังก์ชัน (หรือไม่เรียกก็ได้) นั่นคือ

```
pair<vector<node>, int> parse(string expression)
```

โดยฟังก์ชันนี้จะทำการอ่านสตริงนิพจน์ แล้วคืนค่าเป็นต้นไม้ทวิภาค (binary tree) ที่แต่ละปมจะมีโครงสร้างดังนี้

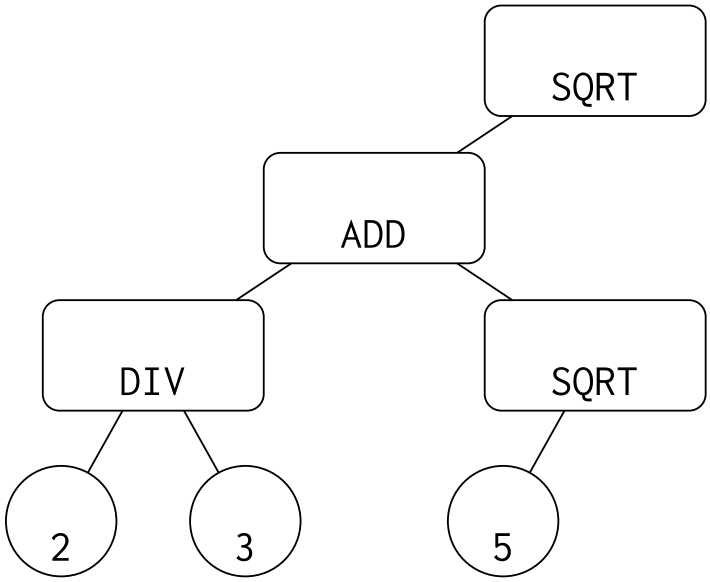
```
struct node {
  string ops;
  int value = -999;
  int left = -1;
  int right = -1;
};
```

หากเรียกผลลัพธ์ของฟังก์ชันว่า `result` แล้ว `result.first` จะเก็บปมทั้งหมดของต้นไม้ไว้ ส่วน `result.second` จะเก็บดัชนีของปมรากของต้นไม้ใน `result.first`

สำหรับแต่ละปมในต้นไม้ หากเรียกปมนั้นว่า `current_node` แล้ว จะมีสมบัติดังต่อไปนี้:

- หาก `current_node.value` มีค่าเท่ากับ `-999` ปมปัจจุบันจะเป็นปมการดำเนินการทางคณิตศาสตร์ โดยค่าของปมจะขึ้นอยู่กับสตริง `current_node.ops`
- หาก `current_node.value` มีค่าไม่เท่ากับ `-999` แล้วปมปัจจุบันจะเป็นปมตัวเลข และจะไม่มีลูกทางซ้ายหรือลูกทางขวาเลย นอกจากนี้ `current_node.ops` จะเป็นสตริงว่าง
- หาก `current_node.left` มีค่าเท่ากับ `-1` จะกล่าวว่าปมปัจจุบันไม่มีลูกทางซ้าย แต่หากไม่เท่ากับ `-1` จะกล่าวว่าลูกปัจจุบันจะเป็นปมที่มีดัชนี `current_node.left` (กล่าวคือ `result.first[current_node.left]` เป็นปมลูกทางซ้ายของปมปัจจุบัน) ในทำนองเดียวกัน เงื่อนไขนี้ก็เป็นจริงสำหรับ `current_node.right`
- หาก `current_node.ops` มีค่า `SQRT` จะมีเพียงลูกทางซ้ายอย่างเดียวเท่านั้น แต่หาก `current_node.ops` มีค่าอื่น และไม่เป็นสตริงว่าง ปมปัจจุบันจะมีทั้งลูกทางซ้ายและลูกทางขวาอยู่

ยกตัวอย่างการอ่านนิพจน์ `SQRT(ADD(DIV(2,3),SQRT(5)))` จะได้ผลลัพธ์เป็นต้นไม้ดังนี้



ซึ่งเมื่อคำนวณออกมาแล้วจะมีค่าเท่ากับ $\sqrt{\frac{2}{3}+\sqrt{5}}$

ปัญหาย่อย

1. (3 คะแนน) `x` เป็นสตริง `1` และ `y` เป็นสตริง `0`
2. (6 คะแนน) `x` เป็นสตริง `0` และ `y` ประกอบด้วยจำนวนเต็มบวกตัวเดียว
3. (7 คะแนน) ทั้ง `x` และ `y` ประกอบด้วยจำนวนเต็มบวกตัวเดียว หรืออยู่ในรูป `SUB(0,t)` เมื่อ `t` เป็นสตริงของจำนวนเต็มบวก
4. (10 คะแนน) `x` เป็นสตริง `0` และ `y` จะเป็นสตริงหารสมบูรณ์ โดยเราจะกล่าวว่าสตริง `t` เป็นสตริงหารสมบูรณ์ก็ต่อเมื่อ `t` เป็น `1` หรือ `t` อยู่ในรูป `DIV(u,2)` สำหรับบางสตริง `u` ที่เป็นสตริงหารสมบูรณ์เช่นกัน
5. (6 คะแนน) ทั้ง `x` และ `y` จะเป็นสตริงหารสมบูรณ์
6. (9 คะแนน) `x` เป็นสตริง `0` และไม่มี `SQRT` ใน `y` เลย
7. (6 คะแนน) ไม่มี `SQRT` ทั้งใน `x` และใน `y` เลย
8. (7 คะแนน) `x` เป็นสตริง `0` และ `y` อยู่ในรูปแบบ `SQRT(t)` สำหรับบางจำนวนเต็ม $0 \leq t \leq 9$
9. (13 คะแนน) เมื่อคำนวณค่าของ `x` และ `y` เป็นจำนวนจริงแล้ว (เขียนแทนด้วย x และ y ตามลำดับ) จะได้ว่าทั้ง x และ y เป็นจำนวนเต็ม (เช่น `MUL(SQRT(2),SQRT(2))` เป็นต้น)
10. (10 คะแนน) ไม่มีการซ้อนกันของ `SQRT` กล่าวคือ ภายในสตริง `x` และ `y` หากมีสัญลักษณ์ `SQRT(t)` สำหรับบางสตริง `t` แล้วจะรับประกันว่า `t` ไม่มี `SQRT` เป็นสตริงย่อย
11. (23 คะแนน) ไม่มีเงื่อนไขเพิ่มเติม

ตัวอย่าง

```
draw("0", "SUB(MUL(3,SQRT(2)),1)")
```

เมื่อมีการเรียกฟังก์ชัน `draw` แล้ว ต่อมาฟังก์ชัน `draw` เรียกใช้ฟังก์ชัน ดังนี้

```
circle c1 = compass(p0, p1);
line l0 = straightedge(p0, p1);
point p2 = intersection(l0, c1, 0);
circle c2 = compass(p1, p2);
circle c3 = compass(p2, p1);
point p3 = intersection(c2, c3, 0);
point p4 = intersection(c2, c3, 1);
line l1 = straightedge(p3, p4);
point p5 = intersection(l1, c1, 0);
line l2 = straightedge(p2, p5);
circle c4 = compass(p1, p0);
point p6 = intersection(l0, c4, 1);
circle c5 = compass(p6, p2);
point p7 = intersection(l2, c5, 0);
circle c6 = compass(p2, p7);
point p8 = intersection(l0, c6, 1);
```

ซึ่งจะสอดคล้องกับไฟล์ตัวอย่าง `canvas.cpp` และจะสอดคล้องกับรูปภาพอธิบายตัวอย่างที่ออกมาจากโปรแกรมตัวช่วย (อ่านต่อได้ในส่วนถัดไปจากนี้) เพื่อเป็นการตรวจสอบความเรียบร้อย และเป็นการให้รูปภาพ

อธิบายตัวอย่าง จะมีการแจกไฟล์รูปภาพอธิบายตัวอย่างเป็น zip ให้ด้วย ชื่อว่า `canvas_figures.zip`

เกรดเดอร์ตัวอย่าง

เกรดเดอร์ตัวอย่างอ่านข้อมูลนำเข้าดังต่อไปนี้:

- บรรทัดที่ 1: x_t ในรูปสตริงนิพจน์ `x`
- บรรทัดที่ 2: y_t ในรูปสตริงนิพจน์ `y`

หมายเหตุ: ข้อมูลส่งออกนี้อาจไม่ใช่สิ่งจำเป็นที่จะต้องสนใจ ผู้เข้าแข่งขันทำการคัดลอกข้อมูลส่งออกนี้ไปแปะลงในไฟล์ แล้วเรียกใช้งานโปรแกรมตัวช่วยเพื่อตรวจสอบความถูกต้องได้ จะเห็นผลลัพธ์ชัดเจนกว่า อย่างไรก็ตาม การอ่านข้อมูลส่งออกนี้อาจเป็นการช่วยให้เข้าใจว่าฟังก์ชัน `draw` ทำงานอย่างไรบ้าง

เกรดเดอร์ตัวอย่างจะส่งออกข้อมูลส่งออกในลักษณะเป็นชุดคำสั่งหลายบรรทัดโดยจะสอดคล้องกับวิธีการดังนี้สำหรับแต่ละบรรทัด:

- หากบรรทัดขึ้นต้นด้วย `#` จะถือว่าบรรทัดนั้นเป็นเพียงคอมเมนต์ (comment) ไม่มีความเกี่ยวข้องกับการทำงานอื่นใด
- หากบรรทัดเป็นบรรทัดว่าง จะถือว่าบรรทัดนั้นไม่มีความเกี่ยวข้องใด
- หากบรรทัดขึ้นต้นด้วยสัญลักษณ์ `P`, `L`, `C` ติดกับตัวเลข `x` ตามด้วยชุดคำสั่ง `y` จะแปลว่า กำหนดให้ค่าของ "จุด", "เส้น", "วงกลม" ดัชนี `x` มีค่าเท่ากับผลของชุดคำสั่ง `y`
  - หาก `y` ขึ้นต้นด้วย `COM` จะสอดคล้องกับการเรียกใช้ฟังก์ชัน `compass`
  - หาก `y` ขึ้นต้นด้วย `LIN` จะสอดคล้องกับการเรียกใช้ฟังก์ชัน `straightedge`
  - หาก `y` ขึ้นต้นด้วย `INT` จะสอดคล้องกับการเรียกใช้ฟังก์ชัน `intersection`
- หากบรรทัดขึ้นต้นด้วยสัญลักษณ์ `!!` จะถือว่าทำการส่งออกจุดนั้นเป็นคำตอบ
- หากบรรทัดขึ้นต้นด้วยสัญลักษณ์ `!x` แสดงว่าเกรดเดอร์พบเจอปัญหาบางอย่าง โดยจะมีข้อความบอกปัญหาตามหลังมา

โปรแกรมตัวช่วย

สำหรับข้อนี้จะมีการแจกโปรแกรมตัวช่วย `vis.py` โดยจะต้องทำการเรียกใช้โปรแกรมด้วย Python 3.6+ ที่มี Matplotlib อยู่ด้วย วิธีการใช้งานคือเรียก `python3 vis.py example.txt` เพื่อแสดงผลการทำงาน เมื่อ `example.txt` เป็นข้อมูลที่ส่งออกมาจากเกรดเดอร์ตัวอย่าง (สามารถเปลี่ยนชื่อไฟล์นี้ได้)

ขอบเขต

- Time limit: 1.0 second
- Memory limit: 512 MB