Data-flow analysis exercises

Draw the control-flow graph for the following program, then perform range analysis.

All integer range [-128…127]

```
prog1:
var a = input()
var b = 50-a
var c = -1
if(a > b && a > c) {
  c = a
} else if(b > c) {
  c = b
}
var c = 100/c
```

```
prog2:
var a = input()
var c = 1
val r = 1
while(c < a) {
  if( a % c == 0 ) r = c
  c = c + 1
}
r = 64/(64-r)
```

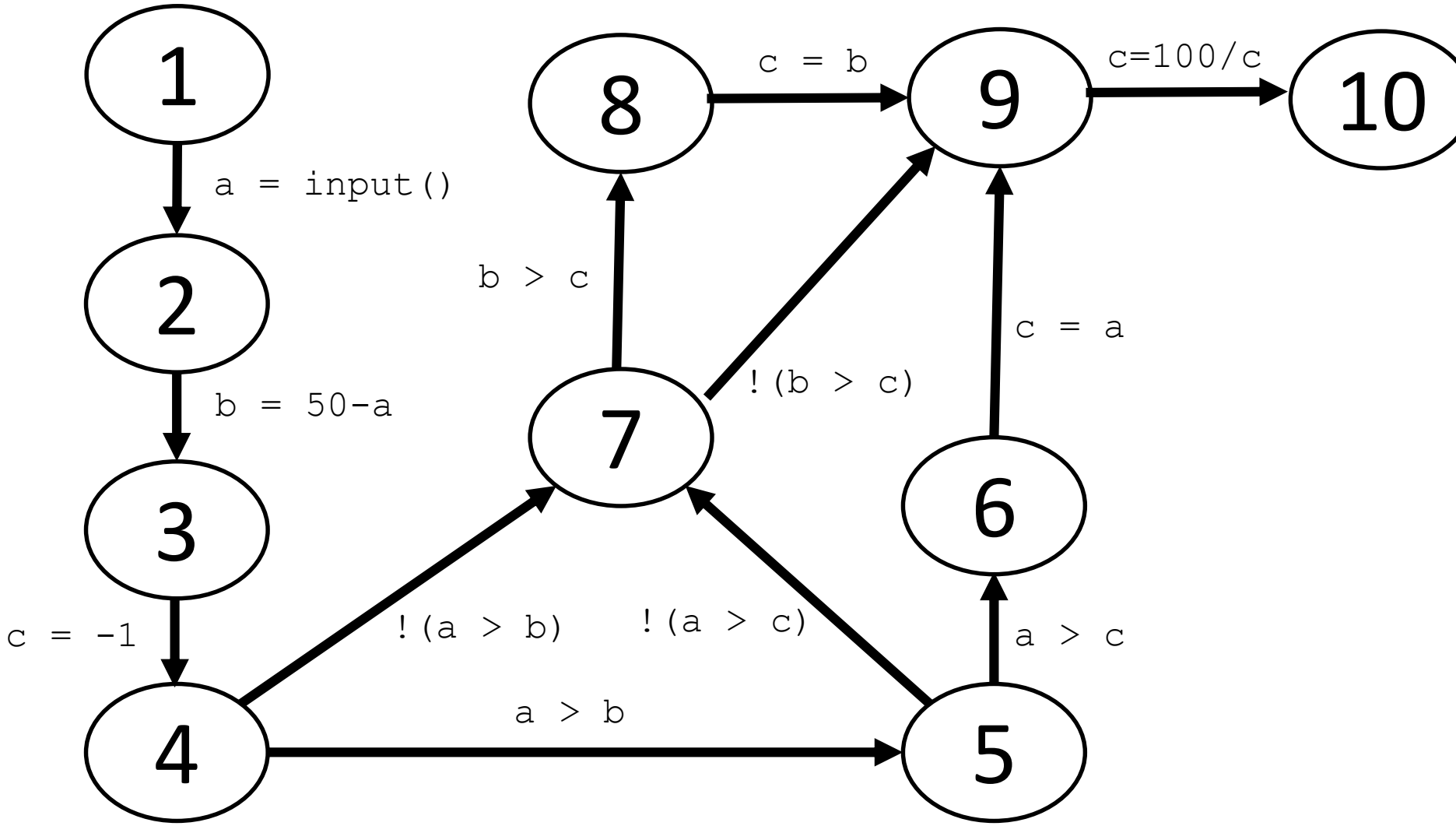
```
prog3:
var x = 2
var y = input()
if (x == y) {
  do {
    y = y + 1
    x = x + y + 3
  } while (y < 4)
} else if (y <= -1 && y >=
-7) {
  y = y * y
} else {
  y = x - 3
}
val z = x/y
```

1. Draw the control-flow graph for the following program, then perform range analysis.

All integer range [-128…127]

```
var a = input()
var b = 50-a
var c = -1
if(a > b && a > c) {
  c = a
} else if(b > c) {
  c = b
}
var c = 100/c
```

1. Solution



1. Range analysis

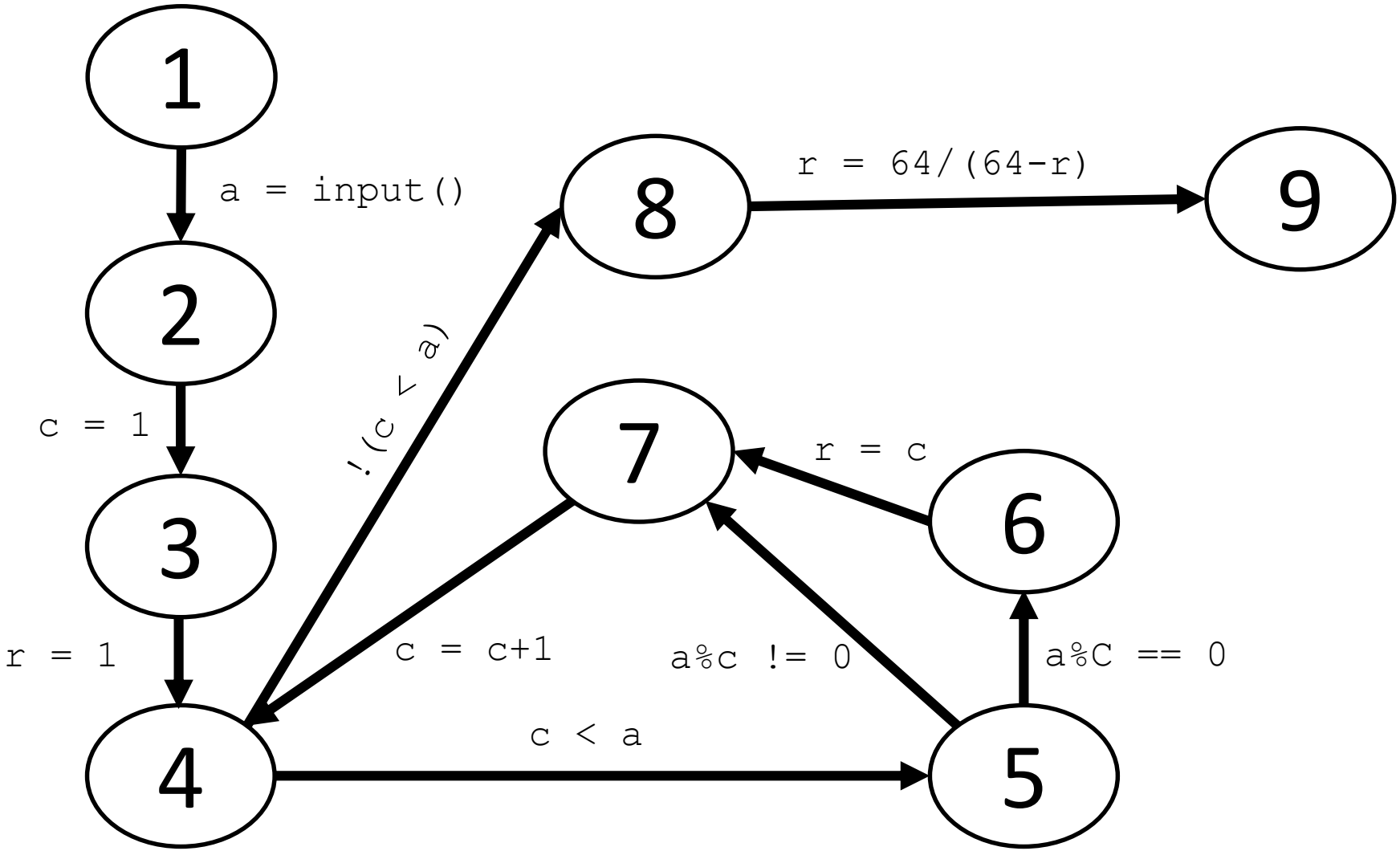
	a	b	c
1:	bottom	bottom	bottom
2:	[-128, 127]	bottom	bottom
3:	[-128, 127]	[-77;127]	bottom
4:	[-128, 127]	[-77;127]	[-1]
5:	[-76, 127]	[-77;126]	[-1]
6:	[0, 127]	[-77;126]	[-1]
7:	[-128, 127]	[-77;127]	[-1]
8:	[-128, 127]	[0;127]	[-1]
9:	[0, 127]	[-77;127]	[0,127]
10:	[0, 127]	[-77;127]	**[0,100]**

2. Draw the control-flow graph for the following program, then perform range analysis.

All integer range [-128,127]

```
var a = input()
var c = 1
val r = 1
while(c < a) {
  if( a % c == 0 ) r = c
  c = c + 1
}
r = 64/(64-r)
```

2. Solution



2. Range analysis

	a	c	r
1:	bottom	bottom	bottom
2:	[-128, 127]	bottom	bottom
3:	[-128, 127]	[1]	bottom
4:	[-128, 127]	[1]	[1]
5:	[2, 127]	[1]	[1]
6:	[2, 127]	[1]	[1]
7:	[2, 127]	[1]	[1]
8:	[-128, 1]	[1]	[1]
9:	[-128, 1]	[1]	[1]

2. Range analysis

	a	c	r
1:	bottom	bottom	bottom
2:	[-128, 127]	bottom	bottom
3:	[-128, 127]	[1]	bottom
4:	[-128, 127]	[1,**2**]	[1]
5:	[2, 127]	[1,**2**]	[1]
6:	[2, 127]	[1,**2**]	[1]
7:	[2, 127]	[1,**2**]	[1,**2**]
8:	[-128, **2**]	[1,**2**]	[1,**2**]
9:	[-128, **2**]	[1,**2**]	[1]

2. Range analysis

	a	c	r
1:	bottom	bottom	bottom
2:	[-128, 127]	bottom	bottom
3:	[-128, 127]	[1]	bottom
4:	[-128, 127]	[1,**3**]	[1,**2**]
5:	[2, 127]	[1,**3**]	[1,**2**]
6:	[2, 127]	[1,**3**]	[1,**2**]
7:	[2, 127]	[1,**3**]	[1,**3**]
8:	[-128, **3**]	[1,**3**]	[1,**3**]
9:	[-128, **3**]	[1,**3**]	[1]

2. Range analysis

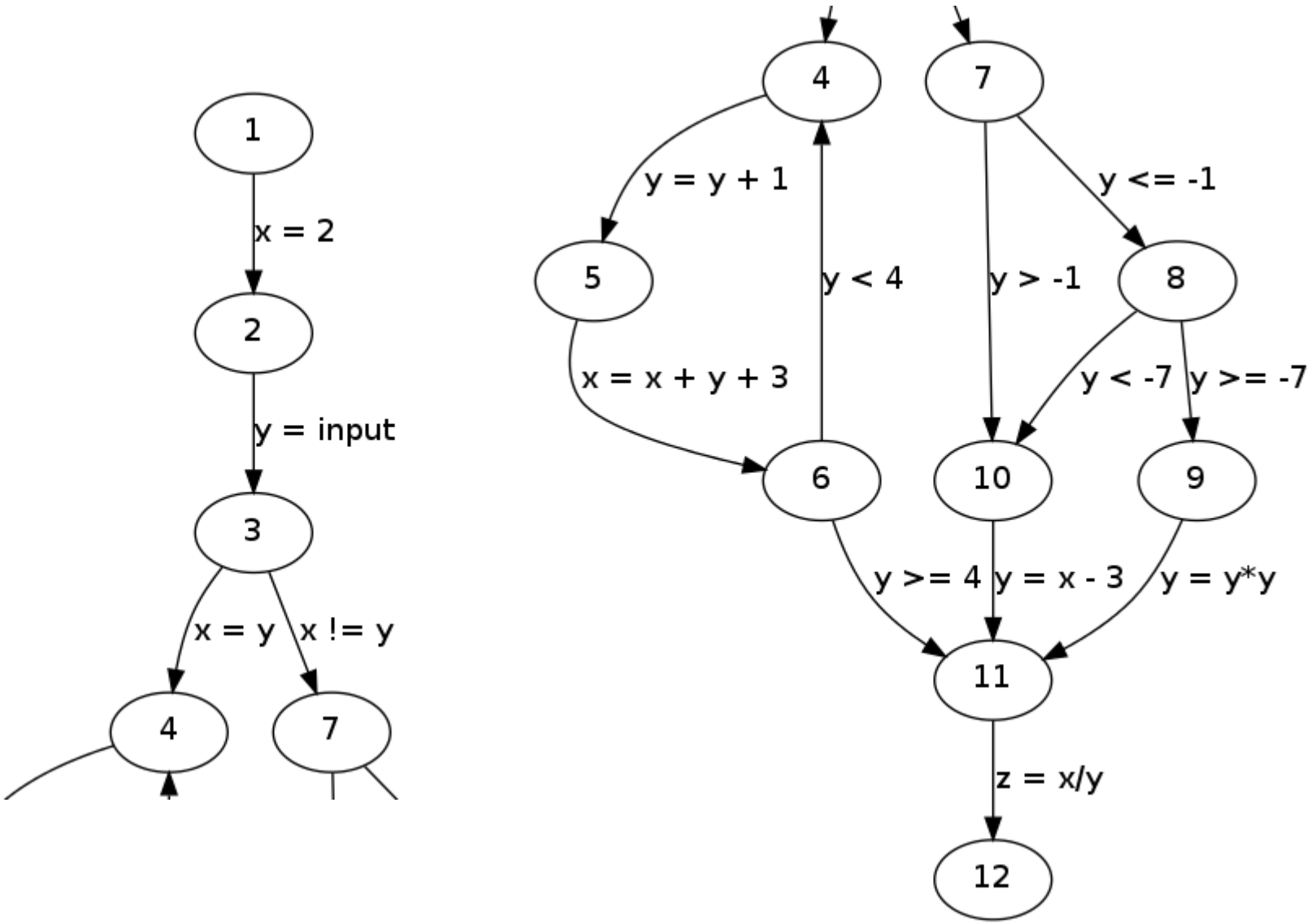
	a	c	r
1:	bottom	bottom	bottom
2:	[-128, 127]	bottom	bottom
3:	[-128, 127]	[1]	bottom
4:	[-128, 127]	[1,**127**]	[1,**126**]
5:	[2, 127]	[1,**126**]	[1,**126**]
6:	[2, 127]	[1,**63**]	[1,**63**]
7:	[2, 127]	[1,**126**]	[1,**126**]
8:	[-128, **127**]	[1,**126**]	[1,**126**]
9:	[-128, **127**]	[1,**127**]	**[0,64]**

3. Draw the control-flow graph for the following program, then perform range analysis.

All integer range [-128,127]

```
var x = 2
var y = input()
if (x == y) {
  do {
    y = y + 1
    x = x + y + 3
  } while (y < 4)
} else if (y <= -1 && y >= -7) {
  y = y * y
} else {
  y = x - 3
}
val z = x/y
```

3. Solution


1
x = 2
2
y = input
3
x = y
x != y
4
7
4
7
y = y + 1
y <= -1
5
y < 4
y > -1
8
x = x + y + 3
y < -7
y >= -7
6
10
9
y >= 4
y = x - 3
y = y*y
11
z = x/y
12

3. Range analysis

```
 x              y
 1:  bottom     bottom
 2: [2, 2]     [-128, 127]
 3: [2, 2]     [-128, 127]
 4: [2, 127]   [2, 3]
 5: [2, 127]   [3, 4]
 6: [8, 127]   [3, 4]
 7: [2, 2]     [-128, 127]
 8: [2, 2]     [-128, -1]
 9: [2, 2]     [-7, -1]
10: [2, 2]     [-128, 127]
11: [2, 127]   [-1, 49]
12: [2, 127]   [-1, 49]     z: T
```