A mysterious function approaches

Mystery function

```
int f(int x, int y) {
  int r = 1;
  while (y > 1) {
    if (y % 2 == 1) {
    r = x * r;
    }
    x = x * x;
    y = y / 2;
  }
  return r * x;
}
```

Is this good code?

What does it compute?

Mystery function

```
int f(int x, int y) {
 int r = 1;
 while (y > 1) {
  if (y % 2 == 1) {
  r = x * r;
  }
  x = x * x;
  y = y / 2;
 }
 return r * x;
}
```

What does it compute?

$x^0 = 1$

$x^y = x^{(y-1)} * x$ if $y > 0$

Mystery function

$x^0 = 1$

$x^y = x^{(y-1)} * x$ if $y > 0$

```
int f(int x, int y)
//@requires e >= 0;
{
  int r = 1;
  while (y > 1) {
    if (y % 2 == 1) {
    r = x * r;
    }
    x = x * x;
    y = y / 2;
  }
  return r * x;
}
```

Specification function

$x^0 = 1$

$x^y = x^{(y-1)} * x$ if $y > 0$

```
int POW(int b, int e)
//@requires e >= 0;
{
  if (e == 0) return 1;
  return b * POW(b, e-1);
}
```

With a contract

```
int POW(int b, int e)
//@requires e >= 0;
{
  if (e == 0) return 1;
  return b * POW(b, e-1);
}

int f(int x, int y)
//@requires y >= 0;
//@ensures POW(x,y) == \result;
{
  int r = 1;
  while (y > 1) {
    if (y % 2 == 1) {
    r = x * r;
    }
    x = x * x;
    y = y / 2;
  }
  return r * x;
}
```

It won't compile!

```
int POW(int b, int e)
//@requires e >= 0;
{
  if (e == 0) return 1;
  return b * POW(b, e-1);
}

int f(int x, int y)
//@requires y >= 0;
//@ensures POW(x,y) == \result;
{ int b = x;
  int e = y;
  int r = 1;
  while (e > 1) {
    if (e % 2 == 1) {
    r = b * r;
    }
    b = b * b;
    e = e / 2;
  }
  return r * b;
}
```

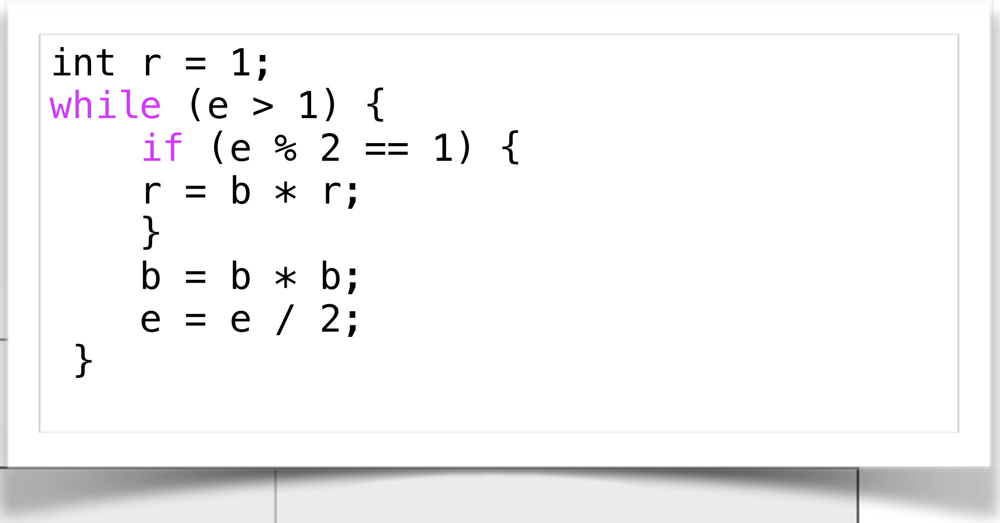
call to f is NOT safe

f(3,-1); // @requires fails

f(3,0); // @ensures fails

f is NOT correct

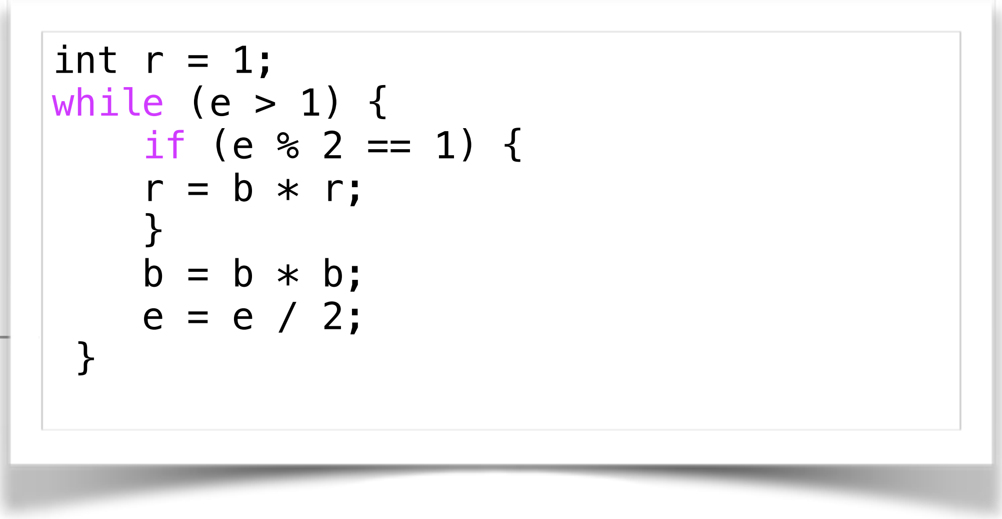
Tracing the loop



```
int r = 1;
while (e > 1) {
    if (e % 2 == 1) {
    r = b * r;
    }
    b = b * b;
    e = e / 2;
 }
```

b	e	r	
2	8	1	
4	4	1	
16	2	1	
256	1	1	

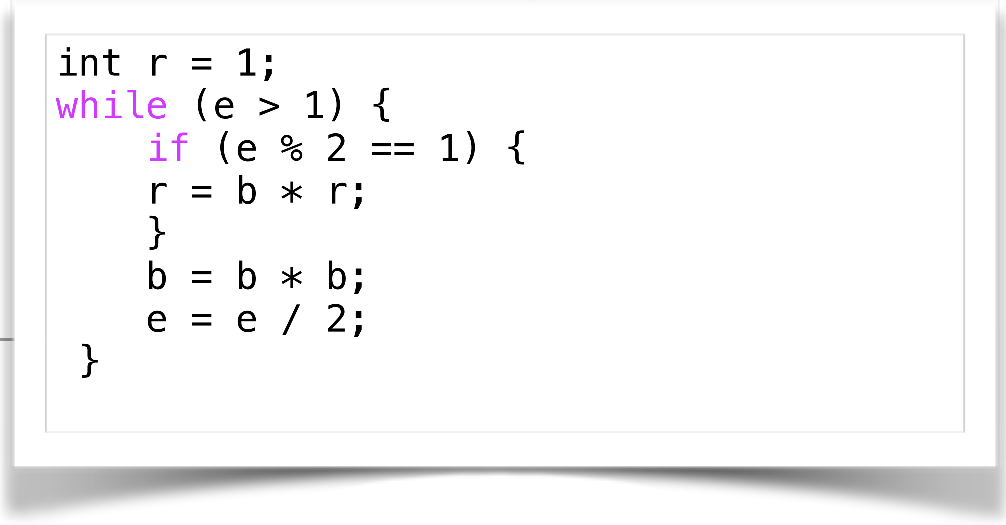
An invariant?



```
int r = 1;
while (e > 1) {
    if (e % 2 == 1) {
    r = b * r;
    }
    b = b * b;
    e = e / 2;
 }
```

b	**e**	**r**	**b^e**
2	8	1	256
4	4	1	256
16	2	1	256
256	1	1	256

What if e is not even?



```
int r = 1;
while (e > 1) {
    if (e % 2 == 1) {
    r = b * r;
    }
    b = b * b;
    e = e / 2;
 }
```

b	**e**	**r**	**b^e**
2	7	1	128
4	3	2	64
16	1	8	16

What remains constant is **b^e * r**

It is always 128

```
int POW(int b, int e)
//@requires e >= 0;
{
  if (e == 0) return 1;
  return b * POW(b, e-1);
}

int f(int x, int y)
//@requires y >= 0;
//@ensures POW(x,y) == \result;
{ int b = x;
  int e = y;
  int r = 1;
  while (e > 1)
  //@loop_invariant e >=0;
  //@loop_invariant POW(b,e) * r = POW(x,y);
  {
    if (e % 2 == 1) {
    r = b * r;
    }
    b = b * b;
    e = e / 2;
  }
  return r * b;
}

```

call to POW(b,e) is safe

Toward proving f correct …

Consider the correctness of g below:

```
int g(int x, int y)
//@requires y >= 0;
//@ensures x == \result;
{ int b = x;


  int r = 1;


  return r * b;
}
```

Proving loop invariants

next time