```
method Euclid(m : int, n : int) returns (q : int, r : int)
{
  q := 0;
  r := m;
  while (r > n)
  {
    r := r - n;
    q := q + 1;
  }
}
```

```
method Euclid(m : int, n : int) returns (q : int, r : int)
ensures m == q * n + r; // ADDED
ensures r < n;  // ADDED
{
  q := 0;
  r := m;
  while (r > n)
  {
    r := r - n;
    q := q + 1;
  }
}
```

```
method Euclid(m : int, n : int) returns (q : int, r : int)
requires n > 0; // ADDED
requires m >= 0; // ADDED
ensures m == q * n + r;
ensures r < n;
{
  q := 0;
  r := m;
  while (r > n)
  {
    r := r - n;
    q := q + 1;
  }
}
```

```
method Euclid(m : int, n : int) returns (q : int, r : int)
requires n > 0;
requires m >= 0;
ensures m == q * n + r;
ensures r < n;
{
  q := 0;
  r := m;
  while (r > n)
  invariant m == q * n + r; // ADDED
  {
    r := r - n;
    q := q + 1;
  }
}
```

```
method Euclid(m : int, n : int) returns (q : int, r : int)
requires n > 0;
requires m >= 0;
ensures m == q * n + r;
ensures r < n;
{
  q := 0;
  r := m;
  while (r >= n) // CHANGED > to >=
  invariant m == q * n + r;
  {
    r := r - n;
    q := q + 1;
  }
}
```

```
method Euclid(m : int, n : int) returns (q : int, r : int)
requires n > 0;
requires m >= 0;
ensures m == q * n + r;
ensures r < n;
{
  q := 0;
  r := m;
  while (r >= n)
  invariant m == q * n + r;
  decreases r; // ADDED
  {
    r := r - n;
    q := q + 1;
  }
}
```

```
method Midpoint(m : int, n : int) returns (r : int)
{
  var delta := (n - m) / 2;
  return m + delta;
}

method Main()
{
  var r := Midpoint(1, 5);
  print r;
  print "\n";
}
```

```
method Midpoint(m : int, n : int) returns (r : int)
ensures r > m && r < n; // ADDED
{
  var delta := (n - m) / 2;
  return m + delta;
}
```

```
method Midpoint(m : int, n : int) returns (r : int)
requires m < n;  // ADDED
ensures r > m && r < n;
{
  var delta := (n - m) / 2;
  return m + delta;
}
```

```
method Midpoint(m : int, n : int) returns (r : int)
requires m < n;
ensures r > m; // CHANGED
ensures r < n; // CHANGED
{
  var delta := (n - m) / 2;
  return m + delta;
}
```

```
method Midpoint(m : int, n : int) returns (r : int)
requires m < n;
ensures r >= m; // CHANGED
ensures r < n;
{
  var delta := (n - m) / 2;
  return m + delta;
}
```