Operator overloading benchmarks

This is a benchmark of operator overloading in boo. Benchmarks are shown for boo without operator overloading, boo with function calls in place of arithmetic operators, boo with operator overloading using a simple struct called "myNum", boo with operator overloading using a class instead of a struct, boo "duck-typed" operators, and Python (without operator overloading). The benchmark computes a piece of the Mandelbrot set. The times to complete the benchmark on a 1.7GHz Pentium M IBM T41p running Windows XP Professional SP1:

Benchmark	Normalized time	Actual time (secs)
boo - no overloaded operators	1	0.35
boo - function calls	3	1.07
boo - overloaded operators (struct version)	16	5.45
boo - overloaded operators (class version)	17-20	5.8 - 7
ActivePython 2.3.2	74	25.75
boo - duck operators	189	66

At its best the class version is only marginally slower than the struct version. However, the variability of the class-based version was much higher, sometimes it took up to 7 seconds to run, presumably because garbage collection kicked in. 190,917,418 myNum objects were created while running the class-based version. Only 129,586 myNums were created while running the struct-based version. Its surprising how much overhead the struct "copy semantics" incurs - the struct version is 5 times slower than the function calls version.

The benchmark code is reproduced below for normal operators, function calls, overloaded operators - struct version, overloaded operators - class version, duck operators, and Python. The benchmark computes a square of the Mandelbrot set 250 by 250 pixels wide.

Normal version (no overloading)

```
/*
  boo unoverloaded version - Mandelbrot Benchmark by Bill Wood
*/

def mb_d(xl as double, xs as double, yl as double, ys as double, ix as int,
iyl as int, iyh as int, r as double, iterations as int, buf as (short)):

    bp = 0
    for iy in range(iyl, iyh):
        cx = xl + ix * xs
        ci = yl + iy * ys
        rx2 = ri2 = rx = ri = 0.0
        count = 0
        while ((rx2 + ri2) <= r) and (count < iterations):
            ri = (rx + rx) * ri + ci
            rx = rx2 - ri2 + cx
            rx2 = rx * rx
            ri2 = ri * ri
            count += 1
        if (rx2 + ri2 > r) and (count <= iterations):
            buf[bp] = count
            bp += 1
        else:
            buf[bp] = 0
            bp += 1

def main(argv as (string)):

    xl = -0.74526593488600
    yl = 0.11303858131900
    xh = -0.74525997120900
    yh = 0.11304454499600
    r = 4.0

    size = int.Parse(argv[0])
    iterations = int.Parse(argv[1])
    print ("size = $size, iterations = $iterations")

    buf = array(short, size)
    xs = (xh - xl) / (size - 1)
    ys = (yh - yl) / (size - 1)

    start = date.Now
    for ix in range(0, size):
        mb_d(xl, xs, yl, ys, ix, 0, size, r, iterations, buf)
    elapsed = date.Now.Subtract(start)
    print ("Boo elapsed time = $elapsed")
    for i in buf:
        System.Console.Write(i + " ")

main(("250", "1000"))
```

Function calls version

```
/*
  boo myNum function calls version - Mandelbrot Benchmark by Bill Wood
*/

def op_Multiply(x as double, j as int):
 return x*j
def op_Multiply(x as double, y as double):
 return x*y
def op_Division(x as double, y as double):
 return x/y
def op_Addition(x as double, j as int):
 return x + j
def op_Addition(x as double, y as double):
 return x + y
def op_Subtraction(x as double, j as int):
 return x - j
def op_Subtraction(x as double, y as double):
 return x - y
def op_GreaterThan(x as double, y as double):
 return x > y
def op_LessThan(x as double, y as double):
 return x < y
def op_LessThanOrEqual(x as double, y as double):
 return x <= y

def mb_d(xl as double, xs as double, yl as double, ys as double, ix as
int,\
  iyl as int, iyh as int, r as double, iterations as int, buf as (short)):
 bp = 0
 iy = iyl
 while op_LessThan(iy, iyh):
  cx = op_Addition(xl, op_Multiply(ix, xs))
  ci = op_Addition(yl, op_Multiply(iy, ys))
  rx2 = ri2 = rx = ri = 0.0
  count = 0
  while (op_LessThanOrEqual(op_Addition(rx2, ri2), r) and
op_LessThan(count, iterations)):
   ri = op_Addition(op_Multiply(op_Addition(rx, rx), ri), ci)
   rx = op_Addition(op_Subtraction(rx2, ri2), cx)
   rx2 = op_Multiply(rx, rx)
   ri2 = op_Multiply(ri, ri)
   count = op_Addition(count, 1)
  if (op_GreaterThan(op_Addition(rx2, ri2), r) and
op_LessThanOrEqual(count, iterations)):
   buf[bp] = count
   bp = op_Addition(bp, 1)
  else:
   buf[bp] = 0
   bp = op_Addition(bp, 1)
  iy = op_Addition(iy, 1)
```

```
def main(argv as (System.String)):
 xl = -0.74526593489
 yl = 0.11303858132
 xh = -0.74525997121
 yh = 0.113044545
 r = 4.0
 size = int.Parse(argv[0])
 iterations = int.Parse(argv[1])

 print("size = $size, iterations = $iterations")
 buf = array(short, size)
 xs = op_Division(op_Subtraction(xh, xl), op_Subtraction(size, 1))
 ys = op_Division(op_Subtraction(yh, yl), op_Subtraction(size, 1))

 start = date.Now
 ix = 0
 while op_LessThan(ix, size):
  mb_d(xl, xs, yl, ys, ix, 0, size, r, iterations, buf)
  ix = op_Addition(ix, 1)
 elapsed = date.Now - start
 print ("Boo elapsed time = $elapsed")
 for i in buf:
  System.Console.Write(i + " ")
```

```
main(("250", "1000"))
```

Overloaded operators struct version

```
/*
    boo myNum struct version - Mandelbrot Benchmark by Bill Wood
*/

struct myNum:
 public static n as int
 public i as double
 def constructor(j as int):
  i = j
  ++n
 def constructor(y as double):
  i = y
  ++n
 def constructor(x as myNum):
  i = x.i
  ++n
 static def op_Multiply(x as myNum, j as int):
  x.i = x.i * j
  return x
 static def op_Multiply(x as myNum, y as myNum):
  x.i = x.i * y.i
  return x
 static def op_Division(x as myNum, y as myNum):
  x.i = x.i / y.i
  return x
 static def op_Addition(x as myNum, j as int):
  x.i = x.i + j
  return x
 static def op_Addition(x as myNum, y as myNum):
  x.i = x.i + y.i
  return x
 static def op_Subtraction(x as myNum, j as int):
  x.i = x.i - j
  return x
 static def op_Subtraction(x as myNum, y as myNum):
  x.i = x.i - y.i
  return x
 static def op_GreaterThan(x as myNum, y as myNum):
  return x.i > y.i
 static def op_LessThan(x as myNum, y as myNum):
  return x.i < y.i
 static def op_LessThanOrEqual(x as myNum, y as myNum):
  return x.i <= y.i
 def ToString():
  return i.ToString()
```

```
def mb_d(xl as myNum, xs as myNum, yl as myNum, ys as myNum, ix as myNum,\
 iyl as myNum, iyh as myNum, r as myNum, iterations as myNum, buf as
(myNum)):

    bp = myNum(0)
    iy = iyl
    while iy < iyh:
        cx = xl + ix * xs
        ci = yl + iy * ys
        rx2 = ri2 = rx = ri = myNum(0)
        count = myNum(0)
        while ((rx2 + ri2) <= r) and (count < iterations):
            ri = (rx + rx) * ri + ci
            rx = rx2 - ri2 + cx
            rx2 = rx * rx
            ri2 = ri * ri
            count += 1
        if (rx2 + ri2 > r) and (count <= iterations):
            buf[bp.i] = count
            bp += 1
        else:
            buf[bp.i] = myNum(0)
            bp += 1
        iy += 1

def main(argv as (string)):

    xl = myNum(-0.74526593488600)
    yl = myNum(0.11303858131900)
    xh = myNum(-0.74525997120900)
    yh = myNum(0.11304454499600)
    r = myNum(4.0)

    size = myNum(int.Parse(argv[0]))
    iterations = myNum(int.Parse(argv[1]))
    print ("size = $(size.i), iterations = $(iterations.i)")

    buf = array(myNum, size.i)
    xs = (xh - xl) / (size - 1)
    ys = (yh - yl) / (size - 1)

    start = date.Now
    ix = myNum(0)
    while ix < size:
        mb_d(xl, xs, yl, ys, ix, myNum(0), size, r, iterations, buf)
        ix += 1
    elapsed = date.Now.Subtract(start)
    print ("Boo elapsed time = $elapsed")
    for i in buf:
        System.Console.Write(i + " ")
    print
    print
    print myNum.n, "myNums created"
```

```
main(("250", "1000"))
```

Overloaded operators class version

```
/*
  boo myNum version - Mandelbrot Benchmark by Bill Wood
*/
class myNum:
 public static n as int
 public i as double
 def constructor(j as int):
  i = j
  ++n
 def constructor(y as double):
  i = y
  ++n
 def constructor(x as myNum):
  i = x.i
  ++n
 static def op_Multiply(x as myNum, j as int):
  return myNum(x.i * j)
 static def op_Multiply(x as myNum, y as myNum):
  return myNum(x.i * y.i)
 static def op_Division(x as myNum, y as myNum):
  return myNum(x.i / y.i)
 static def op_Addition(x as myNum, j as int):
  return myNum(x.i + j)
 static def op_Addition(x as myNum, y as myNum):
  return myNum(x.i + y.i)
 static def op_Subtraction(x as myNum, j as int):
  return myNum(x.i - j)
 static def op_Subtraction(x as myNum, y as myNum):
  return myNum(x.i - y.i)
 static def op_GreaterThan(x as myNum, y as myNum):
  return x.i > y.i
 static def op_LessThan(x as myNum, y as myNum):
  return x.i < y.i
 static def op_LessThanOrEqual(x as myNum, y as myNum):
  return x.i <= y.i
 def ToString():
  return i.ToString()

def mb_d(xl as myNum, xs as myNum, yl as myNum, ys as myNum, ix as myNum,\
 iyl as myNum, iyh as myNum, r as myNum, iterations as myNum, buf as
(myNum)):

    bp = myNum(0)
    iy = iyl
    while iy < iyh:
        cx = xl + ix * xs
```

```
        ci = yl + iy * ys
        rx2 = ri2 = rx = ri = myNum(0)
        count = myNum(0)
        while ((rx2 + ri2) <= r) and (count < iterations):
            ri = (rx + rx) * ri + ci
            rx = rx2 - ri2 + cx
            rx2 = rx * rx
            ri2 = ri * ri
            count += 1
        if (rx2 + ri2 > r) and (count <= iterations):
            buf[bp.i] = count
            bp += 1
        else:
            buf[bp.i] = myNum(0)
            bp += 1
        iy += 1

def main(argv as (string)):

    xl = myNum(-0.74526593488600)
    yl = myNum(0.11303858131900)
    xh = myNum(-0.74525997120900)
    yh = myNum(0.11304454499600)
    r = myNum(4.0)

    size = myNum(int.Parse(argv[0]))
    iterations = myNum(int.Parse(argv[1]))
    print ("size = $(size.i), iterations = $(iterations.i)")

    buf = array(myNum, size.i)
    xs = (xh - xl) / (size - 1)
    ys = (yh - yl) / (size - 1)

    start = date.Now
    ix = myNum(0)
    while ix < size:
        mb_d(xl, xs, yl, ys, ix, myNum(0), size, r, iterations, buf)
        ix += 1
    elapsed = date.Now.Subtract(start)
    print ("Boo elapsed time = $elapsed")
    for i in buf:
        System.Console.Write(i + " ")
    print
    print
    print myNum.n, "myNums created"
```

```
main(("250", "1000"))
```

Duck operators version

```
/*
  boo ducktyped version - Mandelbrot Benchmark by Bill Wood
*/

def mb_d(xl as duck, xs as duck, yl as duck, ys as duck, ix as duck, iyl as
duck, iyh as duck, r as duck, iterations as duck, buf as (duck)):

    bp as duck = 0
    for iy as duck in range(iyl, iyh):
        cx as duck = xl + ix * xs
        ci as duck = yl + iy * ys
#        rx2 = ri2 = rx = ri = 0.0
        rx2 as duck = 0.0; ri2 as duck = 0.0; rx as duck = 0.0; ri as duck
= 0.0
        count as duck = 0
        while ((rx2 + ri2) <= r) and (count < iterations):
            ri = (rx + rx) * ri + ci
            rx = rx2 - ri2 + cx
            rx2 = rx * rx
            ri2 = ri * ri
            count += 1
        if (rx2 + ri2 > r) and (count <= iterations):
            buf[bp] = count
            bp += 1
        else:
            buf[bp] = 0
            bp += 1

def main(argv as (string)):

    xl as duck = -0.74526593488600
    yl as duck = 0.11303858131900
    xh as duck = -0.74525997120900
    yh as duck = 0.11304454499600
    r as duck = 4.0

    size as duck = int.Parse(argv[0])
    iterations as duck = int.Parse(argv[1])
    print ("size = $size, iterations = $iterations")

    buf = array(duck, cast(int, size))
    xs as duck = (xh - xl) / (size - 1)
    ys as duck = (yh - yl) / (size - 1)

    start = date.Now
    for ix as duck in range(0, size):
```

```
    mb_d(xl, xs, yl, ys, ix, 0, size, r, iterations, buf)
elapsed = date.Now.Subtract(start)
print ("Boo elapsed time = $elapsed")
for i as duck in buf:
    System.Console.Write(i + " ")
```

```
main(("250", "1000"))
```

Python version

```
"""
  Python version - Mandelbrot benchmark by Bill Wood

"""


def mb_d(xl, xs, yl, ys, ix, iyl, iyh, r, iterations, buf):

    bp = 0
    for iy in xrange(iyl, iyh):
        cx = xl + ix * xs
        ci = yl + iy * ys
        rx2 = ri2 = rx = ri = 0
        count = 0
        while ((rx2 + ri2) <= r) and (count < iterations):
            ri = (rx + rx) * ri + ci
            rx = rx2 - ri2 + cx
            rx2 = rx * rx
            ri2 = ri * ri
            count += 1
        if (rx2 + ri2 > r) and (count <= iterations):
            buf[bp] = count
            bp += 1
        else:
            buf[bp] = 0
            bp += 1

import sys, time

def Main():

    xl = -0.74526593488600
    yl = 0.11303858131900
    xh = -0.74525997120900
    yh = 0.11304454499600
    r = 4.0

    size = 250
    iterations = 1000
    print "size = ", size, ", iterations = ", iterations

    buf = range(0, size)
    xs = (xh - xl) / (size - 1)
    ys = (yh - yl) / (size - 1)

    starttime = time.clock()
    for ix in xrange(0, size):
        mb_d(xl, xs, yl, ys, ix, 0, size, r, iterations, buf)
    print "Total time:    ", time.clock() - starttime
    print buf

Main()
```