**CSE 330 Homework 2 Report**

Daniel Meyer

Data Structures

Fall 2017

**Status:** 100%

**Time Complexity:** O(1)

**Storage Complexity:** O(1)

**Source Code:**

Page 2: Modulo.h

Page 3-6: Modulo.cpp

**Sample Run:** Page 7

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Kay Zemoudeh

Oct 10, 2017

Modulo.cpp

Implementation of Modulo class given in Modulo.h.

This file implements most of the usual operators for modulo arithmatic such

as =, +, -, ++, etc. along with the constructors.

... (Dear students, add a little more comment here!)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Daniel Meyer

\*10/11/17

\*Modulo.cpp

\*A simple modulo n arithmetic class.

\*An instance of this class is a modulo n object where n > 0.

\*Value of a modulo n object is always >= 0 and < n.

\*

\*CSE 330

\*Fall 2017

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef MODULO\_H

#define MODULO\_H

#include <iostream>

using namespace std;

class Modulo {

int n;

int value;

public:

Modulo(); // default constructor, mod 2 (binary)

Modulo(int m); // mod m object

Modulo(const Modulo & op); // copy constructor

Modulo(int m, int op); // mod m, init to op

Modulo& operator=(int);

Modulo& operator=(const Modulo &);

Modulo operator+(int);

Modulo operator+(const Modulo &);

Modulo& operator++(); // pre-increment

Modulo operator++(int); // post-increment

Modulo operator-(int);

Modulo operator-(const Modulo &);

Modulo& operator--(); // pre-decrement

Modulo operator--(int); // post-decrement

Modulo& operator+=(int);

Modulo& operator+=(const Modulo &);

Modulo& operator-=(int);

Modulo& operator-=(const Modulo &);

friend ostream& operator<<(ostream&, const Modulo &);

}; // Modulo

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Kay Zemoudeh

Oct 10, 2017

Modulo.h

A simple modulo n arithmetic class.

An instance of this class is a modulo n object where n > 0.

Value of a modulo n object is always >= 0 and < n.

... (Dear students, add a little more comment here!)

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*Daniel Meyer

\*10/11/17

\*Modulo.cpp

\*Implementation of Modulo class given in Modulo.h.

\*This file implements most of the usual operators for modulo arithmatic such

\*as =, +, -, ++, etc. along with the constructors.

\*

\*CSE 330

\*Fall 2017

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <cassert>

#include "Modulo.h"

using namespace std;

//Default constructor for Modulo class

Modulo::Modulo() : n(2), value(0) { }

//Constructor for creating a Modulo class from a givin divisor

Modulo::Modulo(int m) : n(m), value(0) { assert(n > 0); }

//Constructor for creating a Modulo class from an existing Modulo class

Modulo::Modulo(const Modulo & op) : n(op.n), value(op.value) { }

//Constructor for creating a Modulo class from a givin divisor and remainder

Modulo::Modulo(int m, int op)

{

assert(m > 0);

n = m;

value = op;

if (value >= 0)

value = value % n;

else

while (value < 0)

value += n;

}

//Overloaded operator for setting current divisor to new divisor and adjusting remainder

Modulo& Modulo::operator=(int op)

{

value = op;

value %= n;

return \*this;

}

//Overloaded operator for setting divisor to existing divisor and adjusting remainder

Modulo& Modulo::operator=(const Modulo & op)

{

value = op.value;

value %= n;

return \*this;

}

//Overloaded operator for adding current and new divisor and adjusting remainder

Modulo Modulo::operator+(int op)

{

Modulo temp = \*this;

temp.value += op;

temp.value %= temp.n;

return temp;

}

//Overloaded operator for adding current and existing divisor and adjusting remainder

Modulo Modulo::operator+(const Modulo & op)

{

Modulo temp = \*this;

temp.value += op.value;

temp.value %= temp.n;

return temp;

}

//Overlaoded pre-increment operator to account for new type

Modulo& Modulo::operator++()

{

value++;

value %= n;

return \*this;

}

//Overlaoded post-increment operator to account for new type

Modulo Modulo::operator++(int op)

{

Modulo temp = \*this;

++(\*this); // calls pre-increment above

return temp;

}

//Overloaded operator to subtract divisor from current divisor and adjust the remainder

Modulo Modulo::operator-(int op)

{

Modulo temp = \*this;

temp.value -= op;

temp.value %= temp.n;

return temp;

}

//Overloaded operator to subtract existing divisor from current divisor and adjust the remainder

Modulo Modulo::operator-(const Modulo &op)

{

Modulo temp = \*this;

temp.value -= op.value;

temp.value %= temp.n;

return temp;

}

//Overlaoded pre-decrement operator to account for new type

Modulo& Modulo::operator--()

{

value--;

value %= n;

return \*this;

}

//Overlaoded post-decrement operator to account for new type

Modulo Modulo::operator--(int op)

{

Modulo temp = \*this;

--(\*this); // calls pre-decrement above

return temp;

}

//Overloaded operator to add new and current divisors and set the result to existing operator and adjust the remainder

Modulo& Modulo::operator+=(int op)

{

Modulo temp = \*this;

temp.value += op;

temp.value %= temp.n;

\*this = temp;

return \*this;

}

//Overloaded operator to add existing and current divisors and set the result to existing operator and adjust the remainder

Modulo& Modulo::operator+=(const Modulo &op)

{

Modulo temp = \*this;

temp.value += op.value;

temp.value %= temp.n;

\*this = temp;

return \*this;

}

//Overloaded operator to subtract new and current divisors and set the result to existing operator and adjust the remainder

Modulo& Modulo::operator-=(int op)

{

Modulo temp = \*this;

temp.value -= op;

temp.value %= temp.n;

\*this = temp;

return \*this;

}

//Overloaded operator to subtract existing and current divisors and set the result to existing operator and adjust the remainder

Modulo& Modulo::operator-=(const Modulo &op)

{

Modulo temp = \*this;

temp.value -= op.value;

temp.value %= temp.n;

\*this = temp;

return \*this;

}

//Overloaded output operator to account for new Modulo class

ostream& operator<<(ostream&os, const Modulo &op)

{

os << op.n << " remainder " << op.value;

return os;

}

**Sample Run:**

Script started on Sun 15 Oct 2017 11:36:34 PM UTC

To run a command as administrator (user "root"), use "sudo <command>".

See "man sudo\_root" for details.

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ g++ Modulo.h Modulo.cpp Modulo\_Test.cpp

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ /a.out

bash: /a.out: No such file or directory

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ ./a.out

o = 5 remainder 3 p = 5 remainder 1

p = 5 remainder 3

p = 5 remainder 3

p = 5 remainder 4

p = 5 remainder 0

q = 5 remainder 0

r = 4 remainder 1 s = 2 remainder 1

r = 4 remainder 2 s = 2 remainder 1

t = 3 remainder 0

t = 3 remainder -1

u = 2 remainder 1

v = 5 remainder -3

v = 5 remainder -4

]0;ubuntu@ubuntu: ~/Desktop[01;32mubuntu@ubuntu[00m:[01;34m~/Desktop[00m$ exit

Script done on Sun 15 Oct 2017 11:37:10 PM UTC