**Lab 06**

**Object Oriented Programming Lab**

**Individual Solution**

**Challenge-1:** *Bounded Integral Type*

**BoundedInteger.h**

#ifndef BOUNDED\_INTEGER\_H

#define BOUNDED\_INTEGER\_H

#include<iostream>

class BoundedInteger

{

int lowerBound;

int upperBound;

int value;

bool isValidBound(int lb, int ub)const;

public:

BoundedInteger();

BoundedInteger(int lb,int ub);

BoundedInteger(int lb,int ub,int val);

void setValue(int val);

int getValue()const;

int getLowerBound()const;

int getUpperBound()const;

void increment(int inc = 1);

void decrement(int dec = 1);

bool isValidValue(int val)const;

};

#endif

**BoundedInteger.cpp**

#include "BoundedInteger.h"

bool BoundedInteger::isValidBound(int lb, int ub)

{

return lb <= ub;

}

BoundedInteger::BoundedInteger() : lowerBound(-2147483647), upperBound(2147483647), value(lowerBound)

{}

BoundedInteger::BoundedInteger(int lb, int ub) : BoundedInteger()

{

if (!isValidBound(lb, ub))

{

return;

}

lowerBound = lb;

upperBound = ub;

value = lowerBound;

}

BoundedInteger::BoundedInteger(int lb, int ub, int val) : BoundedInteger()

{

if (!isValidBound(lb, ub))

{

return;

}

lowerBound = lb;

upperBound = ub;

value = isValidValue(val) ? val : lowerBound;

}

void BoundedInteger::setValue(int val)

{

if (isValidValue(val))

{

value = val;

}

}

int BoundedInteger::getValue() const

{

return value;

}

int BoundedInteger::getLowerBound() const

{

return lowerBound;

}

int BoundedInteger::getUpperBound() const

{

return upperBound;

}

void BoundedInteger::increment(int inc)

{

value = (value + inc - lowerBound) % (upperBound - lowerBound + 1) + lowerBound;

}

void BoundedInteger::decrement(int dec)

{

value = (value - dec - lowerBound + upperBound - lowerBound + 1) % (upperBound - lowerBound + 1) + lowerBound;

}

bool BoundedInteger::isValidValue(int val) const

{

return val >= lowerBound && val <= upperBound;

}

**Challenge-2: *Time***

**Time.h**

#ifndef TIME\_H

#define TIME\_H

#include <iostream>

using namespace std;

#include"BoundedInteger.h"

class Time

{

BoundedInteger hour;

BoundedInteger minute;

BoundedInteger second;

public:

Time(int h=0, int m=0, int s=0);

void setHour(int h);

void setMinute(int m);

void setSecond(int s);

void setTime(int h, int m, int s);

int getHour()const;

int getMinute()const;

int getSecond()const;

void printTwentyFourHourFormat()const;

void printTwelveFourHourFormat()const;

void incSec(int inc = 1);

void incMin(int inc = 1);

void incHour(int inc = 1);

void decSec(int inc = 1);

void decMin(int inc = 1);

void decHour(int inc = 1);

};

#endif // !TIME\_H

**Time.cpp**

#include "Time.h"

Time::Time(int h, int m, int s) : hour(0, 23, h), minute(0, 59, m), second(0, 59, s)

{}

void Time::setHour(int h)

{

hour.setValue(h);

}

void Time::setMinute(int m)

{

minute.setValue(m);

}

void Time::setSecond(int s)

{

second.setValue(s);

}

void Time::setTime(int h, int m, int s)

{

hour.setValue(h);

minute.setValue(m);

second.setValue(s);

}

int Time::getHour() const

{

return hour.getValue();

}

int Time::getMinute() const

{

return minute.getValue();

}

int Time::getSecond() const

{

return second.getValue();

}

void Time::printTwentyFourHourFormat() const

{

cout << "Time: " << hour.getValue() << ":" << minute.getValue() << ":" << second.getValue() << endl;

}

void Time::printTwelveFourHourFormat() const

{

int h = hour.getValue();

char period[3] = { 'A', 'M', '\0' };

if (h >= 12)

{

period[0] = 'P';

h = (h == 12) ? 12 : h % 12;

}

else

{

h = (h == 0) ? 12 : h;

}

cout << "Time: " << h << ":" << minute.getValue() << ":" << second.getValue() << " " << period << endl;

}

void Time::incSec(int inc)

{

second.increment(inc);

if (second.getValue() >= 60)

{

second.setValue(second.getValue() % 60);

incMin();

}

}

void Time::incMin(int inc)

{

incHour(inc / 60);

if (minute.getValue() + inc % 60 > 59)

{

incHour(1);

}

minute.increment(inc % 60);

}

void Time::incHour(int inc)

{

hour.increment(inc);

hour.setValue(hour.getValue() % 24);

}

void Time::decSec(int dec)

{

second.decrement(dec);

if (second.getValue() < 0)

{

minute.decrement(1);

second.increment(60);

}

}

void Time::decMin(int dec)

{

minute.decrement(dec);

if (minute.getValue() < 0)

{

hour.decrement(1);

minute.increment(60);

}

}

void Time::decHour(int dec)

{

hour.decrement(dec);

if (hour.getValue() < 0)

{

hour.increment(24);

}

}