|  |
| --- |
| **Course**: Programming Fundamental – ENSF 337  **Lab #**: Lab 7  **Instructor**: M. Moussavi  **Student Name**: Aarushi Roy Choudhury  **Lab Section**: B01  **Date submitted**: Nov, 18 2021 |

**Exercise A**

Diagram, engineering drawing

Description automatically generated

**Exercise C**

clock.h

class Clock{

        private:

                int hour,minute,second;

                int hms\_to\_sec();

                void sec\_to\_hms(int n);

        public:

                Clock();

                Clock(int s);

                Clock(int h,int m,int s);

                void set\_hour(int h);

                void set\_minute(int m);

                void set\_second(int s);

                int get\_hour()const;

                int get\_minute()const;

                int get\_second()const;

                void increment();

                void decrement();

                void add\_seconds(int s);

};

clock.cpp

#include "clock.h"

Clock::Clock(){

        hour=0;

        minute=0;

        second=0;

}

Clock::Clock(int s){

                hour=0;

        minute=0;

        second=0;

        sec\_to\_hms(s);

}

Clock::Clock(int h,int m,int s){

        hour=0;

        minute=0;

        second=0;

        if(h>=0&&m<=23&&m>=0&&m<=59&&s>=0&&s<=59){

                        set\_hour(h);

                set\_minute(m);

                set\_second(s);

        }

}

void Clock::set\_hour(int h){

        if(h>=0&&h<=23){

                hour=h;

        }

}

void Clock::set\_minute(int m){

        if(m>=0&&m<=59){

                minute=m;

        }

}

void Clock::set\_second(int s){

        if(s>=0&&s<=59){

                second=s;

        }

}

int Clock::get\_hour()const{

        return hour;

}

int Clock::get\_minute()const{

        return minute;

}

int Clock::get\_second()const{

        return second;

}

void Clock::increment(){

        if(second==59){

                second=0;

                if(minute==59)

                {

                        minute=0;

                        if(hour==23){

                                hour=0;

                        }else

                                hour++;

                }else

                        minute++;

        }else

                second++;

}

void Clock::decrement(){

        if(second==0){

                second=59;

                if(minute==0)

                {

                        minute=59;

                        if(hour==0){

                                hour=23;

                        }else

                                hour--;

                }else

                        minute--;

        }else

                second--;

}

void Clock::add\_seconds(int s){

        for(int i=1;i<=s;i++){

                increment();

        }

}

int Clock::hms\_to\_sec(){

        return second+minute\*60+hour\*60\*60;

}

void Clock::sec\_to\_hms(int sec){

        if(sec>0){

        int m=0,s=0,h=0;

        h=sec/(3600);

        sec=sec-h\*(3600);

        if(sec>=60){

                m=sec/60;

                s=sec%60;

        }else

                s=sec;

                if(h==24)

                        set\_hour(0);

                else

        set\_hour(h);

        set\_minute(m);

        set\_second(s);

        }

}

Text

Description automatically generated

**Exercise D**

#include "simpleVector.h"

#include <cassert>

using namespace std;

SimpleVector::SimpleVector(const TYPE \*arr, int n) {

    storageM = new TYPE[n];

    sizeM = n;

    capacityM = n;

    for(int i =0; i < sizeM; i++)

        storageM[i] = arr[i];

}

TYPE& SimpleVector::at(int i) {

    assert(i >= 0 && i < sizeM);

    return storageM[i];

}

const TYPE& SimpleVector::at(int i)const {

    assert(i >= 0 && i < sizeM);

    return storageM[i];

}

// The followng member function should follow the above-mentioned memory

// management policy to resize the vector, if necessary. More specifically:

//   - If sizeM < capacityM it doesn't need to make any changes to the size of

//     allocated memory for vector

//   - Otherwise it follows the above-mentioned memory policy to create additionl

//     memory space and adds the new value, val, to the end of the current vector

//     and increments the value of sizeM by 1

void SimpleVector::push\_back(TYPE val) {

//  THIS FUNCTION MUST BE COMPLETED BY THE STUDENTS

if(sizeM == capacityM)

{

    if(capacityM == 0)

    {

        capacityM = 2;

        storageM = new TYPE[capacityM];

    }

    else

    {

        TYPE\* arr = new TYPE[capacityM];

        for(int i=0; i<capacityM; i++)

            arr[i] = storageM[i];

        capacityM \*= 2;

        storageM = new TYPE[capacityM];

        for(int i=0; i<capacityM/2; i++)

            storageM[i] = arr[i];

        delete[] arr;

    }

}

storageM[sizeM++] = val;

}

SimpleVector::SimpleVector(const SimpleVector& source) {

//  THIS FUNCTION MUST BE COMPLETED BY THE STUDENTS

sizeM = source.sizeM;

    capacityM = source.capacityM;

    storageM = new TYPE[capacityM];

    for(int i = 0; i < sizeM; i++)

        storageM[i] = source.storageM[i];

}

SimpleVector& SimpleVector::operator= (const SimpleVector& rhs ){

//  THIS FUNCTION MUST BE COMPLETED BY THE STUDENTS

     sizeM = rhs.sizeM;

    capacityM = rhs.capacityM;

    storageM = new TYPE[capacityM];

    for(int i = 0; i < sizeM; i++)

        storageM[i] = rhs.storageM[i];

    return \*this;

}

Text

Description automatically generated

**Exercise E**

Text

Description automatically generated

Text

Description automatically generated