AM3611: C++ for Scientific Computing

Assignment3: Basics of Object-Oriented Programming

Due: 19 October

Yizhou Tang

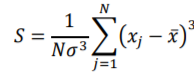
250888541

**Chapter 5:**

**Question 5.4:**

**Written description:**

Four major parts of the program:

1. Function: generate random numbers
   1. This function’s inputs require an array and the size of the array. It loops through the given array to assign a random number between 0 and 1.0 to each element of the array. The random number generation is done by using std::rand() to generate a random integer first, then cast integer to a double and divide by RAND\_MAX in order to scale it to 0 and 1.0.
2. Function: mean calculation
   1. This function also takes an array and the size of the array. It starts off by initialize a variable *sum* as 0. It then loops over every element of the array and add to the sum variable each loop. Once the sum of the array is ready, the mean is calculated by dividing sum by the size of the array.
3. Function: standard deviation and skew calculation (optional)
   1. This function requires: an array, size of the array, mean of the array, reference to variable for standard deviation, and an optional pointer argument for skew calculation. The function first loops through the elements of the array to calculate the sum of (xj-mean)^2，where x represents the elements from the array. It then divides the sum by the size of the array. Lastly, square roots the result to get standard deviation.
   2. The function also has a default value for skew, nullptr. If a correct input was given, then the function will proceed to calculate skew for the array. This is done by following the formula: ****. It calculates the sum of (xj – mean)^3 through a for loop over all the elements of the array, then proceeds to compute skew by dividing the sum by the product of size and (standard deviation) ^3.
4. Main function
   1. The main function takes a command line argument, integer N, then constructs an dynamically allocated array with size N. It then proceeds to call the random number generator function first to fill in the empty array, then the mean and standard deviation & skew functions to compute the statistical results of the data.

**Code:**

#include <iostream>

#include <cmath>

void randNumbers(double array[],int size);

double mean(double array[], int size);

void stdev\_skew(double array[], int size, double mean, double& stdev,double\* p\_Skew = nullptr);

int main (int argc, char\* argv[])

{

//Convert to integer

int N = atoi(argv[1]);

//Pointer for the array

double \*dynamic\_array;

//Dynamically allocate array based off of user's input

dynamic\_array = new double[N];

//Call the randNumbers function to fill in random numbers to the array

randNumbers(dynamic\_array, N);

//display the array

std::cout << "The Array:"<< std::endl;

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

{

std::cout <<dynamic\_array[i] << "\n";

}

//Calculate and display the mean of the array

double average;

average = mean(dynamic\_array,N);

std::cout <<"Mean: " << average<< "\n";

//Calculate and display the standard deviation and the skew of the array

double stdev, skew;

std::cout<<"stdev\_skew(dynamic\_array, N, average, stdev), output:\n";

stdev\_skew(dynamic\_array, N, average, stdev);

std::cout << "\n";

std::cout<<"stdev\_skew(dynamic\_array, N, average, stdev,skew), output:\n";

stdev\_skew(dynamic\_array, N, average, stdev,&skew);

//de-allocate at the end of the main function

delete [] dynamic\_array;

return 0;

}

void randNumbers(double \*array, int size)

{

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

{

int randomint;

randomint = std::rand();

double randomdoub;

randomdoub = static\_cast<double>(randomint) / RAND\_MAX;

array[i] = randomdoub;

}

}

double mean(double \*array, int size)

{

//Mean calculation

double sum = 0;

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

{

sum += array[i];

}

double average;

average = sum / size;

return average;

}

void stdev\_skew(double \*array, int size, double mean, double& stdev,double\* p\_Skew){

//Standard deviation calculation

stdev = 0;

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

{

//double a = array[i];

stdev += pow(array[i] - mean,2.0);

}

stdev = sqrt(stdev / double(size));

std::cout <<"Standard Deviation: " << stdev<< "\n";

//Only calculate skew if the variable is given in the input

if (p\_Skew){

\*p\_Skew =0;

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

{

\*p\_Skew += pow(array[i] - mean,3.0);

}

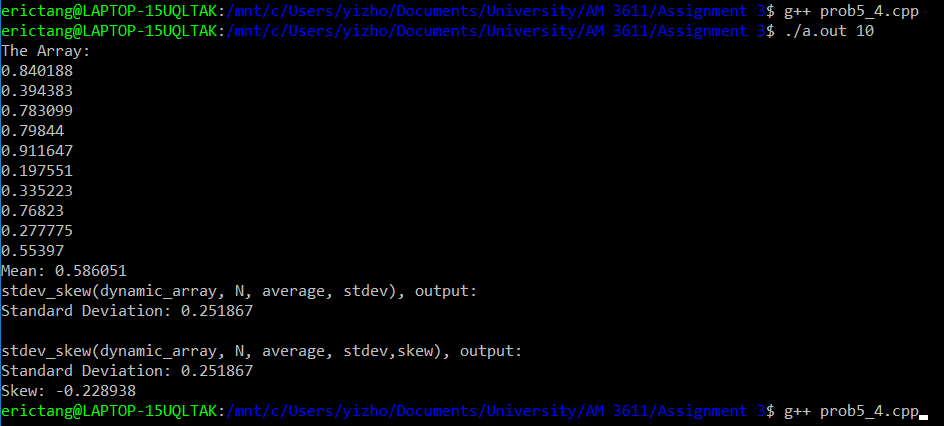
\*p\_Skew = \*p\_Skew/(size\*pow(stdev,3.0));

std::cout <<"Skew: " << \*p\_Skew << "\n";

}

}

**Screenshot:**

****

**Comments:**

As the screenshot above shows, the program successfully meets all the objectives required.

Check:

Mean:

Sum of the 10 numbers generated above = sumA = 5.860506,

Mean = sumA/10 = 5.860506 / 10 = 0.586051,

which yielded the same result as the program.

Standard deviation:

Sum of (xi - mean) ^ 2 = sumB = 0.6343714,

Stdev = sqrt (sumB / 10) =0.251867,

which yielded the same result as the program.

Skew:

Sum of (xi - mean) ^ 3 = sumC = -0.03658,

Skew = sumC / (N \* stdev ^ 3) = -0.03658 / (10 \* 0.251867^3) = -0.22894,

which yielded the same result as the program.

**Question 5.5:**

**Written description:**

Two major parts of the program:

1. Main function
   1. The main function builds two matrices for testing purposes (dynamic allocation): A = {{0, 1}, {1, 0}} and B = {{0, 0, 2}, {0, 0, 3}}. It then calls the multiply function twice. Correct input in the first time and incorrect input for the second time.
   2. First time it calls the function to compute AB, which should successfully run since A is a 2x2 matrix and B is a 2x3 matrix.
   3. The second time it calls the function to compute BA, which shouldn’t successfully run since it is against the matrix multiplication law.
2. Function: Multiply
   1. This function takes in six inputs, 2 matrices that are to be multiplied, and the number of rows and columns for both matrices.
   2. Before anything, it uses an assert statement to check if the number of columns of the first matrix is equal to the number of rows of the second matrix.
   3. It constructs a dynamically allocated matrix variable to hold the answer.
   4. It then proceeds to compute matrix multiplication by using three nested for loops.

**Code:**

#include <iostream>

#include <cassert>

//Two matrices of given sizes

void multiply(double\*\* matrixA, double\*\* matrixB, int rowA, int colA, int rowB, int colB);

int main (int argc, char\* argv[])

{

//Parameters for matrices

int rowA,colA, rowB, colB, rowC, colC;

rowA = 2;

colA = 2;

rowB = 2;

colB = 3;

//Allocate memories dynamically for the matrices

double\*\* A = new double\*[rowA];

double\*\* B = new double\*[rowB];

A[0] = new double [rowA \* colA];

B[0] = new double [rowB \* colB];

for (int i = 1; i < rowA; i ++)

{

A[i] = A[i-1] +colA;

}

for (int i = 1; i < rowB; i ++)

{

B[i] = B[i-1] +colB;

}

//Assign values to the matices

A[0][1] = 1.0;

A[1][0] = 1.0;

B[0][2] = 2.0;

B[1][2] = 3.0;

//Display the matrices

std::cout << "Matrices:\n";

std::cout<< "A: {{" << A[0][0] << "," << A[0][1] << "},{"

<< A[1][0] << "," << A[1][1] << "}}\n";

std::cout<< "B: {{" << B[0][0] << "," << B[0][1] << "," << B[0][2]<< "},{"

<< B[1][0] << "," << B[1][1] << "," << B[1][2]<< "}}\n";

std::cout<< "\n";

//Show the correct and incorrect scenarios

std::cout << "Correct Scenario: \nResult of AB(2x2 and 2x3):\n";

multiply (A,B,rowA,colA,rowB,colB);

std::cout<<"\n";

std::cout << "Incorrect Scenario: \nResult of BA(2x3 and 2x2):\n";

multiply (B,A,rowB,colB, rowA,colA);

delete A[0];

delete B[0];

delete A;

delete B;

}

//Matrix multiplication

void multiply(double\*\* matrixA, double\*\* matrixB, int rowA, int colA, int rowB, int colB)

{

//Check if the matrices have suitable sizes

assert (colA == rowB);

//Construct a new matrix for answer

double\*\* answer = new double\*[rowA];

answer[0] = new double [rowA \* colB];

for (int i = 1; i < rowA; i ++)

{

answer[i] = answer[i-1] + colB;

}

//Compute matrix multiplication

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

{

for (int j = 0; j < colB; j++)

{

for (int k = 0; k < colA; k++)

{

answer[i][j] += matrixA[i][k] \* matrixB[k][j];

}

}

}

//Display the result to the screen

//This is done so that it can print matrices with any sizes correctly

std::cout<<"{";

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

{

if(i != 0){

std::cout<<",";

}

std::cout<<"{";

for (int j = 0; j <colB; j++)

{

std::cout<<answer[i][j];

if (j != colB -1){

std::cout<< ", ";

}

}

std::cout<<"}";

}

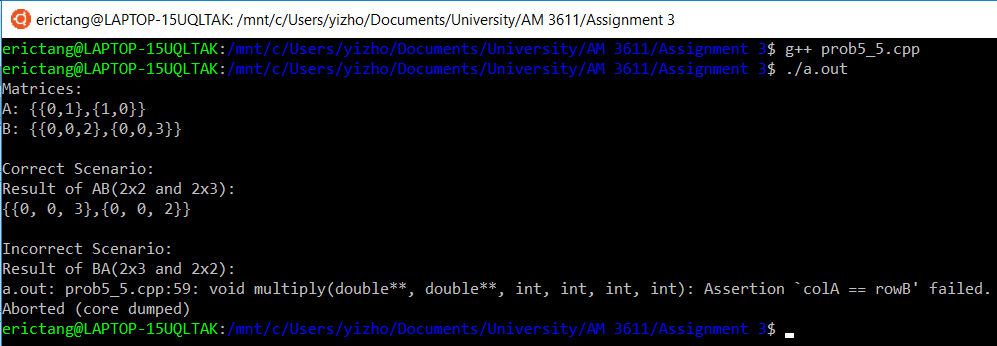
std::cout<<"}\n";

delete answer[0];

delete answer;

}

**Screenshot:**



**Comments:**

As the above screenshot shows, the program is able to successfully compute the product of two matrices and identify when the inputs are incorrect.

A = {{0, 1}, {1, 0}},

B = {{0, 0, 2}, {0, 0, 3}},

AB = {{(0 \* 0 + 1 \* 0), (0 \* 0 + 1 \* 0), (0 \* 2 + 1\*3)}, {(1 \* 0 + 0 \* 0), (1 \* 0 + 0 \* 0), (1 \* 2 + 0 \* 3)}.

AB = {{0, 0, 3}, {0, 0, 2}}.

The above calculation shows the same result as the program.

In the incorrect scenario, the assert statement of the program successfully picked up the error: “Assertion ‘colA == rowB’ failed.”

**Question 5.6:**

**Written description:**

This question is very similar to the previous question; however, it asks us to write 4 additional functions (overload).

1. a vector and a matrix of given sizes;
   1. void multiply(double\* vector, double\*\* matrix, int size, int rowB, int colB);
2. a matrix and a vector of given sizes;
   1. void multiply(double\*\* matrix, double\* vector, int rowA, int colA, int size);
3. a scalar and a matrix of a given size; and
   1. void multiply(double scalar, double\*\* matrix, int rowB, int colB);
4. a matrix and a given size and a scalar.
   1. void multiply(double\*\* matrix, double scalar, int rowA, int colA);

For testing purposes, a matrix and a vector were constructed:

A = {{0, 2}, {2, 0}},

B = {0,10}.

Since we assume the vector is a row vector in the first question and the vector is a column vector in the second question. The result should be one row vector and another column vector as well.

In addition, since the order does not matter in scalar and matrix multiplication, we should expect the result for question 3 and 4 to be the same.

**Code:**

#include <iostream>

#include <cassert>

//A vector and a matrix of given sizes (assume row vector)

void multiply(double\* vector, double\*\* matrix, int size, int rowB, int colB);

//A matrix and a vector of given sizes （assume column vector)

void multiply(double\*\* matrix, double\* vector, int rowA, int colA, int size);

//A scalar and a matrix of a given size

void multiply(double scalar, double\*\* matrix, int rowB, int colB);

//A matrix of a given size and a scalar

void multiply(double\*\* matrix,double scalar, int rowA, int colA);

int main (int argc, char\* argv[])

{

int rowA,colA, size;

rowA = 2;

colA = 2;

size = 2;

//Allocate memories dynamically for the matrix

double\*\* A = new double\*[rowA];

A[0] = new double [rowA \* colA];

for (int i = 1; i < rowA; i ++)

{

A[i] = A[i-1] +colA;

}

//Assign values to the matrix

A[0][1] = 2.0;

A[1][0] = 2.0;

double\* B = new double [size];

B[1] = 10;

B[0] = 0;

//Display the matrix and vectors

std::cout << "Matrix:\n";

std::cout<< "A: {{" << A[0][0] << "," << A[0][1] << "},{"

<< A[1][0] << "," << A[1][1] << "}}\n";

std::cout << "Vector:\n";

std::cout <<" B: {" << B[0] << ", "<< B[1] << "}\n";

//Question 1: Vector x matrix, assuming the vector is a row vector (1 x n)

std::cout << "Question 1, Vector & Matrix Multiplication, BA (Assume row vector):\n";

multiply(B,A, size,rowA,colA);

//Question 2: Matrix x vector, assuming the vector is a column vector

std::cout << "Question 2, Matrix & Vector Multiplication, AB (Assume column vector):\n";

multiply(A,B,rowA,colA,size);

std::cout << "Question 3, Scalar & Matrix Multiplication, 2A:\n";

multiply(2,A, rowA,colA);

std::cout << "Question 4, Matrix & Scalar Multiplication, A\*2:\n";

multiply(A,2, rowA,colA);

delete A[0];

delete A;

delete B;

}

//Vector x matrix (assume vector is row vector)

void multiply(double\* vector, double\*\* matrix, int size, int rowB, int colB)

{

assert(size == rowB);

double\* answer = new double[size];

//Compute matrix multiplication

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

{

for (int j = 0; j < size;j++)

{

answer[i] += vector[j] \* matrix[i][j];

}

}

std::cout << "{";

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

{

std::cout << answer[i];

if (i != size-1)

{

std::cout << ", ";

}

}

std::cout << "}\n";

}

//Matrix x vector (assume vector is column vector)

void multiply(double\*\* matrix, double\* vector, int rowA, int colA, int size)

{

assert(colA == size);

double\* answer = new double[size];

//Compute matrix multiplication

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

{

for (int j = 0; j < size;j++)

{

answer[i] += matrix[i][j] \* vector[j] ;

}

}

std::cout << "{";

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

{

std::cout <<"{"<< answer[i] << "}";

if (i != size-1)

{

std::cout << ", ";

}

}

std::cout << "}\n";

delete answer;

}

//Scalar x matrix

void multiply(double scalar, double\*\* matrix, int rowB, int colB)

{

//Construct a new matrix for answer

double\*\* answer = new double\*[rowB];

answer[0] = new double [rowB \* colB];

for (int i = 1; i < rowB; i ++)

{

answer[i] = answer[i-1] + colB;

}

//Compute matrix multiplication

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

{

for (int j = 0; j < colB; j++)

{

answer[i][j] = scalar \* matrix[i][j];

}

}

//Display the result to the screen

//This is done so that it can print matrices with any sizes correctly

std::cout<<"{";

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

{

if(i != 0){

std::cout<<",";

}

std::cout<<"{";

for (int j = 0; j <colB; j++)

{

std::cout<<answer[i][j];

if (j != colB -1){

std::cout<< ", ";

}

}

std::cout<<"}";

}

std::cout<<"}\n";

}

//Matrix x scalar

void multiply(double\*\* matrix,double scalar, int rowA, int colA)

{

//Construct a new matrix for answer

double\*\* answer = new double\*[rowA];

answer[0] = new double [rowA \* colA];

for (int i = 1; i < rowA; i ++)

{

answer[i] = answer[i-1] + colA;

}

//Compute matrix multiplication

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

{

for (int j = 0; j < colA; j++)

{

answer[i][j] = scalar \* matrix[i][j];

}

}

//Display the result to the screen

//This is done so that it can print matrices with any sizes correctly

std::cout<<"{";

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

{

if(i != 0){

std::cout<<",";

}

std::cout<<"{";

for (int j = 0; j <colA; j++)

{

std::cout<<answer[i][j];

if (j != colA -1){

std::cout<< ", ";

}

}

std::cout<<"}";

}

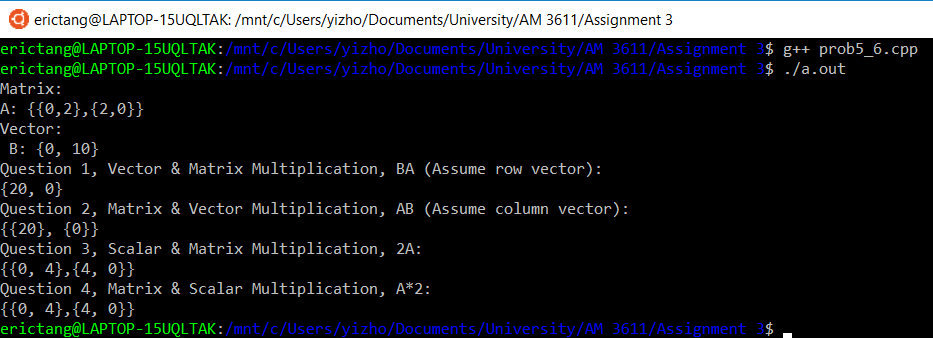
std::cout<<"}\n";

delete answer[0];

delete answer;

}

**Screenshot:**

****

**Comments:**

As the screenshot shows above, the program is able to perform all four tasks correctly.

BA if B is a row vector:

BA = {(0 \* 0 + 10 \* 2), (0 \* 2 + 10 \* 0)},

BA = {20, 0}.

AB if B is a column vector:

AB = {{0 \* 0 + 2 \* 10}, {2 \* 0 + 0 \* 10}},

AB = {{20}, {0}}.

Both calculations above confirms the results from the program. This means that the program is able to compute multiplications between a vector and a matrix correctly, despite the order of multiplication (as long as if the sizes are suitable).

For the scalar multiplications, it is very straight forward to see that the program is performing correctly. Multiplying any matrix by 2 simply doubles all the numbers in the matrix, which is demonstrated in the screenshot above.

**Question 5.11:**

**Written description:**

This program attempts to compute a simple approximation to a Julia set that has a fractal form. Two major parts of the program:

1. Main function
   1. The main function first takes a command line argument that represents the number of “pixels” in the y-direction, Ny. It then calculates Nx, which is twice the Ny. Once both Ny and Nx are ready, the function dynamically allocates an array of integers that represent pixels on the output.
   2. Once all the preparations are finished, a nested loop is used to iterate through all the all the elements of the matrix. The function InJuliaSet is called on every single iteration to identify whether or not the point is in the Julia set (or at least whether or not it remained bounded for a large number of iterations).
   3. Once every single point is identified, the program prints out the results to the terminal screen. “X ” if the function returned 1, “ “ if the function returned 0, for each element.
2. Function: InJuliaSet
   1. The InJuliaSet function takes four integer inputs: ix, iy, Nx, and Ny. The first two arguments represent the specific spots of a matrix, and the later two represents the pixels required by the user.
   2. The first step is to calculate zx and zy, which are scaled x and y coordinates of pixcel ix and iy, in (-2, 2) and (-1, 1).
   3. The major step of the function is the while loop. The loop will continue iterating until one of the two conditions are met: zx \* zx + zy \* zy <4, or iteration reaches a predefined number (which is 1000 in the code). In each iteration, zx \* zx – zy \* zy is hold in a temporary variable and both zy and zx are updated. The record of the number of iterations also increases by one at the end of each iteration.
   4. After the while loop finishes, an if statement is used to check whether the total number of iterations has reached the predefined max iteration number. The function will return 1 if it is true, 0 if false.

**Code:**

#include <iostream>

#include <cmath>

int InJuliaSet(int ix,int iy, int Nx, int Ny);

int main (int argc, char\* argv[])

{

//Convert to integer

//Number of elements in the y direction

int Ny = atoi(argv[1]);

//Number of elements in the x direction

int Nx = 2 \* Ny;

int\*\* pixcelArray = new int\*[Ny];

pixcelArray[0] = new int [Ny \* Nx];

for (int i = 1; i < Nx; i ++)

{

pixcelArray[i] = pixcelArray[i-1] +Nx;

}

//Use the InJuliaSet function on every single element of the matrix

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

{

for (int j = 0; j <Nx; j++)

{

pixcelArray[i][j] =InJuliaSet(j,i,Nx,Ny);

}

}

//Print out the results

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

{

for (int j = 0; j <Nx; j++)

{

//"X " if in the set, " " if not in the set

if (pixcelArray[i][j] == 1)

{

std::cout << "X ";

}

else

{

std::cout<<" ";

}

if (j == Nx-1)

{

std::cout<< "\n";

}

}

}

}

int InJuliaSet(int ix,int iy, int Nx, int Ny)

{

double zx;

double zy;

double cx;

double cy;

cx = 0.7885\*cos(M\_PI);

cy = 0.7885\*sin(M\_PI);

//zx = scaled x coordinate of pixel ix (scale to lie in (-2, 2))

//zx represents the real part of z

zx = -2.0 + static\_cast<double>(ix)/static\_cast<double>(Nx)\*(4.0);

//zy = scaled y coordinate of pixel iy (scale to lie in (-1, 1))

//zy represents the imaginary part of z

zy = -1.0 + static\_cast<double>(iy)/static\_cast<double>(Ny)\*(2.0);

int iteration = 0;

int max\_iteration = 1000;

while (((zx \* zx + zy \* zy) < 4) && (iteration < max\_iteration))

{

double xtemp;

xtemp = zx\*zx -zy\*zy;

zy = 2.0\*zx\*zy + cy;

zx = xtemp + cx;

iteration ++;

}

if (iteration == max\_iteration)

{

return 1;

}

else

{

return 0;

}

}

**Screenshot:**

****

**Comment:**

The above screenshot is the result when cx = 0.7885\*cos(π) and cy = 0.7885\*sin(π). This shows that the program is capable of taking changing the resolutions based on the command line input.

**Chapter 6:**

**Question 6.0:**

**Written description:**

This question starts with a given file, DateClass.cpp, where we need to perform modifications according to the instructions.

1. Modify the setDate function to use a switch statement that checks if the day is appropriate for the month given. One particular case if February, decision between 29 days and 28 days is based on the result of year mod 4. If the remainder is 0 then 29 days, otherwise 28 days. In addition, it also uses an if statement to check whether the year is in a reasonable range (-5000 to 2500). The function also returns a boolean value true if the date is successfully set and false if not successful.
2. Add a constructor to the class that asks for month, day, and year as inputs, but also has a default date of January 1st, 2018. It also calls the function setDate and uses an assert statement to check if the date was successfully set. Lastly, the program attempts to demonstrate that the class can be initialized using direct initialization, uniform initialization, and by using the implicit copy constructor.

**Code**:

#include <iostream>

#include <cassert>

class DateClass // members are private by default

{

private:

int m\_month;

int m\_day;

int m\_year;

public:

//Constructor with parameters

DateClass(int month = 1, int day = 1, int year = 2018)

{

assert(setDate(month,day,year));

setDate(month,day,year);

}

bool setDate(int month, int day, int year) // public, accessible to anyone

{

//Check if the day is appropriate for the month given

bool succesful = true;

m\_month = month;

m\_day = day;

m\_year = year;

if ((month < 1) || (month >12))

{

std::cout << "Month out of range.\n";

succesful = false;

}

switch(m\_month)

{

//Months with 31 days

case 1:

case 3:

case 5:

case 7:

case 8:

case 10:

case 12:

if ((m\_month < 1) || (m\_month >31))

{

std::cout << "Date out of range. \n";

succesful = false;

}

break;

//months with 30 days

case 4:

case 6:

case 9:

case 11:

if ((m\_month < 1) || (m\_month >30))

{

std::cout << "Date out of range. \n";

succesful = false;

}

break;

//Feburary special case

case 2:

if (m\_year % 4 == 0)

{

if ((m\_month < 1) || (m\_month >29))

{

std::cout << "Date out of range. \n";

succesful = false;

}

}

else

{

if ((m\_month < 1) || (m\_month >28))

{

std::cout << "Date out of range. \n";

succesful = false;

}

}

break;

}

if ((m\_year < -5000) || (m\_year > 2500))

{

std::cout << "Year out of range.\n";

succesful = false;

}

//Return if the method has succesfully set the appropriate date

return (succesful);

}

void print() // public, can be accessed by anyone

{

std::cout << m\_month << "/" << m\_day << "/" << m\_year;

}

};

int main()

{

DateClass date;

if (date.setDate(10, 14, 2020))

{

std::cout << "The date was set succesfully.\n";

}

else

{

std::cout << "The date was not set succesfully.\n";

}

date.print(); // okay, because print() is public

std::cout << std::endl;

std::cout<<"Direct Initialization Result:\n";

DateClass directDate(10,14,2020);

directDate.print();

std::cout << std::endl;

std::cout << "Uniform Initialization Result:\n";

DateClass uniformDate{10,14,2020};

uniformDate.print();

std::cout << std::endl;

std::cout <<"Implicit Copy Constructor:\n";

DateClass copyDate(date);

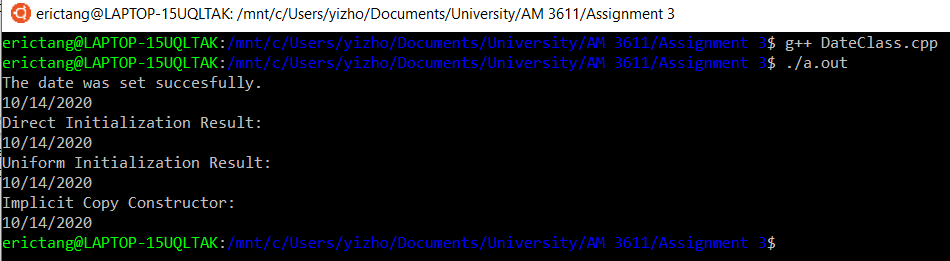
copyDate.print();

std::cout << std::endl;

return 0;

}

**Screenshot:**

****

**Comments:**

As the screenshot clearly shows, all three different methods: direct initialization, uniform initialization and the implicit copy constructor are able to initialize the given class, correctly.