# 1DV533 STEP 5 Assignment report

## Jesper Malmberg [em222vs@student.lnu.se](mailto:em222vs@student.lnu.se)

## Task 1

I created a dynamic array call arr where the integers will be stored. I then check the input in a while loop and if it’s non numeric the loop will break. I then created a dynamic tempo array where I the original array is copied to and the last int is inserted at the end of the array. I then delete the first array and reset the point to point to tempo.

//-----------------------------------------------------------------------

//-----------------------------------------------------------------------

// File: Task\_1.cpp

// Summary: This program takes a random number of integers to a dynamic

// array.

// Version: 1.1

// Owner: Jesper Malmberg

//-----------------------------------------------------------------------

// Log: 2022-01-01 Created file

//-----------------------------------------------------------------------

// Preprocessor directives

#include <iostream>

#include<cstring>

#include <algorithm>

#include <iterator>

using namespace std;

// Prototypes

int main()

{

char answer;

do {

system("CLS"); // Clear the console

int counter = 0;

int\* arr = new int[0]; // The initial array

int i; // Stores the input

cout << "Enter integer numbers to store (Stop by entering any non-numeric value):";

while(cin >> i) {

counter++;

int \*tempo = new int[counter]; // Temporary array = arr + 1 in lenght

copy(arr, arr + counter, tempo); // Copy the contents of arr to tempo

tempo[counter - 1] = i; // Insert the input at last index of tempo

delete []arr; // Delete the first array

arr = tempo; // Reset the pointer of arr to point to the tempo array

}

// Print the results

cout << "You entered " << counter << " numbers: ";

for (int i = 0; i < counter; i++) {

cout << arr[i] << " ";

}

cout << endl;

// Clear the input buffer of errors and old data in preparation for the next run of the program

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Try Again (Y/N)? ";

cin.get(answer);

// Clear the input buffer of errors and old data

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

} while (toupper(answer) == 'Y');

return 0;

}

## Task 2

I simply moved the call to srand(time(0)) to outside the loop in order to preserve the initial seed value. If the rand function keeps getting the same value, as was pretty much the case before the same random numbers will be generated. Time(0) was being used over and over again in short succession which generated pretty much the same seed value.

//-----------------------------------------------------------------------

//-----------------------------------------------------------------------

// File: Task\_2.cpp

// Summary: THIS IS AN IMPROVEMENT OF ANOTER PROJECT

// Version: 1.1

// Owner: Jesper Malmberg

//-----------------------------------------------------------------------

// Log: 2022-02-02 Created file

//-----------------------------------------------------------------------

// Preprocessor directives

#include <iostream>

#include <ctime>

using namespace std;

enum Coin { Tails, Heads };

int main()

{

int frequency0 = 0, frequency1 = 0;

Coin face;

// Move the call to srand() outside the scope of the for loop in order to get the same seed value for all calls to rand()

// The psuedo random generator should only be seeded once, seeding it 10 times within short time period is likely the reason

// for the same random numbers beeing generated.

srand(time(0));

for (int counter = 1; counter <= 10; ++counter)

{

face = Coin(rand() % 2);

cout << rand() << " ";

switch (face)

{

case Tails: ++frequency0;

break;

case Heads: ++frequency1;

break;

}

}

cout << "Tails were rolled " << frequency0 << " times" << endl;

cout << "Heads were rolled " << frequency1 << " times" << endl;

cin.get();

return 0;

}

## Task 3

Pretty straight forward, just created a structure called TimeType with two int members to store hours and minutes.

//-----------------------------------------------------------------------

//-----------------------------------------------------------------------

// File: Task\_3.cpp

// Summary: This program demonstrates the use of structure TimeType

// Version: 1.1

// Owner: Jesper Malmberg

//-----------------------------------------------------------------------

// Log: 2022-02-04 Created file

//-----------------------------------------------------------------------

// Preprocessor directives

#include <iostream>

#include<cstring>

#include <algorithm>

#include <iterator>

#include <ctime>

using namespace std;

struct TimeType

{

int hour;

int min;

};

// Prototypes

int toMinutes(TimeType);

TimeType toTime(int);

TimeType timeDifference(TimeType, TimeType);

void print(TimeType);

TimeType dynamicTimes();

int main()

{

// Demonstration of subtaskt a)

int minutes;

TimeType timeA = { 10, 15 };

minutes = toMinutes(timeA);

cout << minutes << " minutes" << endl; // Should write: 615 minutes

// Demonstration of subtask b)

int min = 124;

TimeType time = toTime(min);

print(time);

cout << endl;

// Demonstration of subtask c)

TimeType timeB = { 10, 30 }, timeC = { 13, 20 }, difference;

difference = timeDifference(timeB, timeC);

print(difference);

cout << endl;

cout << endl;

// Demonstration of subtask d)

TimeType maxDiff = dynamicTimes();

print(maxDiff);

cout << endl;

return 0;

}

//----------------------------------------------------------------------

// toMinutes(TimeType)

// This function returns the total minutes from TimeType struct

// Returns: int

//----------------------------------------------------------------------

int toMinutes(TimeType tp) {

return tp.hour \* 60 + tp.min;

}

//----------------------------------------------------------------------

// toTime(int)

// This function returns the hours and minutes in the form of TimeType

// Returns: TimeType

//----------------------------------------------------------------------

TimeType toTime(int min) {

TimeType tp;

tp.hour = min / 60;

tp.min = min % 60;

return tp;

}

//----------------------------------------------------------------------

// timeDifference(TimeType, TimeType)

// This function returns the difference of two times

// Returns: TimeType

//----------------------------------------------------------------------

TimeType timeDifference(TimeType t1, TimeType t2) {

int difference = toMinutes(t1) - toMinutes(t2);

if (difference < 0) {

difference = -difference;

}

return toTime(difference);

}

//----------------------------------------------------------------------

// dynamicTimes()

// This function returns the difference of two times, the largest and

// smallest of 200 random times

// Returns: TimeType

//----------------------------------------------------------------------

TimeType dynamicTimes() {

// Seed the randomizer

srand(time(0));

TimeType\* times = new TimeType[200]; // Dynamic array with 200 timeTypes

int\* tempo = new int[200]; // Dynamic array with the timetype as minutes for sorting

// Randomize time in minutes

for (int i = 0; i < 200; i++) {

tempo[i] = rand();

}

// Sort the minutes

sort(tempo, tempo + 200);

// Create TimeTypes

for (int i = 0; i < 200; i++) {

times[i] = toTime(tempo[i]);

}

// Calculate difference between first and last element in times

TimeType tt = timeDifference(times[0], times[199]);

// Delete dynamic arrays

delete[]tempo;

delete[]times;

return tt;

}

//----------------------------------------------------------------------

// print(TimeType)

// This helper function prints a TimeType in the hh:mm format

//----------------------------------------------------------------------

void print(TimeType tp) {

if (tp.hour < 10) {

cout << "time 0" << tp.hour << ":";

}

else {

cout << "time " << tp.hour << ":";

}

if (tp.min < 10) {

cout << "0" << tp.min;

}

else {

cout << tp.min;

}

}

## Task 4

This game is made up of two functions, gameOver and compareHands plus an enum named hand. I seed the srand() function with current time, and while the game is not over a loop keeps asking for selection and randomizes the computers selection. A series of switches in compareHands checks who scored and updates the scores and when either reaches 10 the gameOver function returns true and the game ends.

//-----------------------------------------------------------------------

//-----------------------------------------------------------------------

// File: Task\_4.cpp

// Summary: This program simulates a Rock, Paper, Scissors game.

// Version: 1.1

// Owner: Jesper Malmberg

//-----------------------------------------------------------------------

// Log: 2022-02-10 Created file

//-----------------------------------------------------------------------

// Preprocessor directives

#include <iostream>

#include<cstring>

#include <algorithm>

#include <iterator>

#include <ctime>

using namespace std;

// Enum Hand

enum Hand { Rock, Paper, Scissor };

// Prototype

bool gameOver(int, int);

void compareHands(Hand, Hand, int&, int&);

int main()

{

char answer; // Play again or not

do {

system("CLS"); // Clear the console

cout << "STONE SCISSORS PAPPER" << endl;

cout << "=====================" << endl;

Hand playerSelection; // The player selection

Hand computerSelection; // The computer selection

int playerScore = 0; // Players running score

int computerScore = 0; // Computers running score

int inpt;

srand(time(0)); // Seed the rand with current time

// While the game is not over solicit input and compare

while (!gameOver(playerScore, computerScore)) {

cout << "Your choice(Rock = 0 / Paper = 1 / Scissors = 2) : ";

cin >> inpt;

playerSelection = Hand(inpt);

computerSelection = Hand(rand() % 3 + 0);

// Comparer hanads

compareHands(playerSelection, computerSelection, playerScore, computerScore);

cout << "Score (you - computer): " << playerScore << " - " << computerScore << endl;

}

// Clear the input buffer of errors and old data

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cout << "Play again? (Y/N)";

cin.get(answer);

} while (toupper(answer) == 'Y');

return 0;

}

//----------------------------------------------------------------------

// gameOver(int, int)

// This function returns a boolean indication if the game is over

// Returns: bool

//----------------------------------------------------------------------

bool gameOver(int ps, int cs) {

if (ps == 10) {

cout << "You won with " << ps << "-" << cs << endl;

return true;

}

if (cs == 10) {

cout << "You lost with " << cs << "-" << ps << endl;

return true;

}

return false;

}

//----------------------------------------------------------------------

// compareHands(Hand, Hand, pScore, cScore)

// This function compares the two input enums Hand and adds to the score

// of the winning hand

// Returns: void

//----------------------------------------------------------------------

void compareHands(Hand player, Hand computer, int &pScore, int &cScore) {

// Print computer's selection

switch (computer) {

case Rock: cout << "Computer chose Rock" << endl;

break;

case Paper: cout << "Computer chose Paper" << endl;

break;

case Scissor: cout << "Computer chose Scissor" << endl;

break;

}

// Check player's hand. The nested switch then checks the computers hand with the nested switches and prints who won and adds to the running score

switch (player) {

case Rock:

switch (computer) {

case Rock: cout << "Equal!" << endl;

break;

case Paper: cout << "Computer won!" << endl;

cScore++;

break;

case Scissor: cout << "You won!" << endl;

pScore++;

break;

}

break;

case Paper:

switch (computer) {

case Rock: cout << "You won!" << endl;

pScore++;

break;

case Paper: cout << "Equal!" << endl;

break;

case Scissor: cout << "Computer won!" << endl;

cScore++;

break;

}

break;

case Scissor:

switch (computer) {

case Rock: cout << "Computer won!" << endl;

cScore++;

break;

case Paper: cout << "You won!" << endl;

pScore++;

break;

case Scissor: cout << "Equal!" << endl;

break;

}

break;

}

}

## Task 5

I store the friends’ names in dynamic arrays within friends array. Once inputted the sort function uses strcmp to sort in alphanumerical order. Capital letters comes before lowercase as is the value in ASCII. For example, Jesper comes before adam. Once the sorted list is printed to the console the memory from the dynamic arrays is freed up with delete. Finally the pointer to friends is removed and the memory freed up.

//-----------------------------------------------------------------------

//-----------------------------------------------------------------------

// File: Task\_5.cpp

// Summary:

// Version: 1.0

// Owner: Jesper Malmberg

//-----------------------------------------------------------------------

// Log: 2022-02-12 Created file

//-----------------------------------------------------------------------

// Preprocessor directives

#include <iostream>

#include <cstring>

using namespace std;

const int BUFLEN = 100; // Max length of reading buffer

void sort(char\* friendList[], int n); // n is the number of elements

void print(char\* friendList[], int n); // n is the number of elements

void terminate(char\* friendList[], int n); // n is the number of elements

const int AMOUNT = 5;

int main() {

char\* friends[AMOUNT]; // Dynamic array with AMOUNT pcs of string pointers

char buff[BUFLEN] = { "" }; // Creates a string buffer (null terminated)

int count = 0;

while (count < AMOUNT) // enter AMOUNT number of friends

{

cout << "Name a friend: ";

cin.getline(buff, BUFLEN); // Temporary reading into string buffer

friends[count] = new char[strlen(buff) + 1]; // Add dynamic char array to friends

// Add the characters

for (int i = 0; i < strlen(buff); i++) {

friends[count][i] = buff[i];

}

// Add null terminator

friends[count][strlen(buff)] = '\0';

++count;

}

sort(friends, count); // Sorts the ‘count’ strings

print(friends, count); // Prints the ‘count’ first names

terminate(friends, count); // Releases all allocated memory space

return 0;

}

//----------------------------------------------------------------------

// sort(char\*, int)

// This function sort the strings in accordance with strcmp logic

// Returns: void

//----------------------------------------------------------------------

void sort(char\* friendList[], int n) {

char\* tempo;

for (int i = 1; i < n; i++) {

for (int j = 1; j < n; j++) {

if (strcmp(friendList[j - 1], friendList[j]) > 0) {

tempo = friendList[j - 1];

friendList[j - 1] = friendList[j];

friendList[j] = tempo;

}

}

}

}

//----------------------------------------------------------------------

// print(char\*, int)

// This function prints the sorted names

// Returns: void

//----------------------------------------------------------------------

void print(char\* friendList[], int n) {

// Print each char

for (int i = 0; i < n; i++) {

for (int j = 0; j < strlen(friendList[i]); j++) {

cout << friendList[i][j];

}

cout << " ";

}

}

//----------------------------------------------------------------------

// terminate(char\*, int)

// This function frees up the allocated memory for friendList

// Returns: void

//----------------------------------------------------------------------

void terminate(char\* friendList[], int n) {

// Delete the number of dynamic char arrays

for (int i = 0; i < n; i++) {

delete [] friendList[i];

}

// Remove reference to friendList

friendList = nullptr;

delete[] friendList;

}

## Task 6

The throw arrow uses the random\_device class to generate random doubles to generate two doubles that’s stored as coordinates in the Coord struct. I used the proided formula to check if below the arc in below() function. In throwSeries() function I basically call the throwArrow function n number of times and check how many ended up inside the arc. This is then used to calculate Pi and the relative fault.

//-----------------------------------------------------------------------

//-----------------------------------------------------------------------

// File: Task\_6.cpp

// Summary: This program calculates Pi using randomness

// Version: 1.0

// Owner: Jesper Malmberg

//-----------------------------------------------------------------------

// Log: 2022-02-12 Created file

//-----------------------------------------------------------------------

// Preprocessor directives

#include <iostream>

#include <random>

#include <iomanip>

using namespace std;

const double R = 100; // Radius of the circle (=square side) is determined here

struct Coord // Data representation of a point where the arrow hits

{

double x, y;

};

// Prototype

Coord throwArrow();

bool below(Coord);

void throwSeries(double);

int main() {

// Print header

cout << " n pi Rel. fault" << endl;

cout << "------|----------|-------" << endl;

// Throw series

for (int i = 100; i < 10100; i = i + 100) {

throwSeries(i);

}

return 0;

}

//----------------------------------------------------------------------

// throwArrow()

// This function simulates the throw of an arrow and returns an x and y

// coordinate

// Returns: Coord

//----------------------------------------------------------------------

Coord throwArrow() {

std::random\_device rd; // Will be used to obtain a seed for the random number engine

std::mt19937 gen(rd()); // Standard mersenne\_twister\_engine seeded with rd()

std::uniform\_real\_distribution<> dis(0.0, 100.0); // Double between 0 - 100

return Coord{dis(gen),dis(gen)};

}

//----------------------------------------------------------------------

// below(Coord)

// This function checks if a resulting arrow throw ends up below the arc

// with radius R

// Returns: boolean

//----------------------------------------------------------------------

bool below(Coord cord) {

// Check if below the arc

if (pow(cord.x, 2) + pow(cord.y, 2) < pow(R, 2)) {

return true;

}

return false;

}

//----------------------------------------------------------------------

// throwSeries(double)

// This function simulates a series of throws n, then calculates pi and

// compares it with a constant pi and prints the results to the console

// Returns: void

//----------------------------------------------------------------------

void throwSeries(double n) {

Coord c;

const double Pi = 4 \* atan(1.0); // PI Constant

double belowArc = 0; // Number of arrows inside the arc

double piCalculated; // The calculated pi

double relFault; // Relative fault in %

// Throw the arrow n times and record how many below the arc

for (int i = 0; i < n; i++) {

c = throwArrow();

if (below(c)) {

belowArc++;

}

}

// Calculate pi and relative fault

piCalculated = (belowArc \* 4) / n;

relFault = (piCalculated - Pi) / Pi \* 100;

// Print the result to the console

cout << fixed << setw(6) << right << setprecision(0) << n;

cout << fixed << right << setw(11) << setprecision(5) << piCalculated;

cout << fixed << right << setw(8) << setprecision(1) << relFault << endl;

}

## Task 7

I declare and setup the two matrixes in the main function with the help of setupMatrix() function which basically just solicits input into the Matrixes. In the readMatrix() function I basically solicit values to input into the two different matrixes. I then multiply the two matrixes according to the formula provided, which is done in a triple nested for loop, the new values are stored in a third Matrix and then printed to the console.

//-----------------------------------------------------------------------

//-----------------------------------------------------------------------

// File: Task\_7.cpp

// Summary: This program populates two matrixes and caluclates the product

// after multiplication

// Version: 1.0

// Owner: Jesper Malmberg

//-----------------------------------------------------------------------

// Log: 2022-02-22 Created file

//-----------------------------------------------------------------------

// Preprocessor directives

#include <iostream>

#include <iomanip>

using namespace std;

const int MAXDIM = 5; // max number of rows and columns supported by program

typedef double Matrix[MAXDIM][MAXDIM];

// Prototype

void setUpMatrix(Matrix M, int &row, int &col);

void readMatrix(Matrix M, int row, int col);

void multMatrix(const Matrix A, const Matrix B, Matrix C, int m, int n, int p);

void printMatrix(const Matrix M, int row, int col);

int main() {

char answer;

do {

system("CLS"); // Clear the console

cout << "Matrix manipulation" << endl;

cout << "===================" << endl;

cout << endl;

// The three matrixes

Matrix A;

Matrix B;

Matrix C;

// The three variables representing rows and columns for the three matrixes

int row\_m = 0;

int col\_n= 0;

int colRow\_p = 0;

// Enter dimensions and values of Matrix A

cout << "Enter dimension of matrix A (row/col) with space between: ";

setUpMatrix(A, row\_m, colRow\_p);

cout << "Enter matrix A in free format:" << endl;

readMatrix(A, row\_m, colRow\_p);

// Enter dimensions and values of Matrix B

cout << "Enter dimension of matrix B (row/col) with space between: ";

setUpMatrix(B, colRow\_p, col\_n);

cout << "Enter matrix B in free format:" << endl;

readMatrix(B, colRow\_p, col\_n);

// Multiply the two matrices

multMatrix(A, B, C, row\_m, col\_n, colRow\_p);

// Print the result

printMatrix(C, row\_m, col\_n);

cout << "One more time (Y/N)?: ";

// Clear the input buffer of errors and old data

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

cin.get(answer);

// Clear the input buffer of errors and old data

cin.clear();

cin.ignore(numeric\_limits<streamsize>::max(), '\n');

} while (toupper(answer) == 'Y');

return 0;

}

//----------------------------------------------------------------------

// setupMatrix(Matrix, int, int)

// This function takes user inputs from the console to setup a matrix

// Returns: void

//----------------------------------------------------------------------

void setUpMatrix(Matrix M, int& row, int& col) {

while (true) {

cin >> col;

if (cin.peek() == '\n') {

break;

}

row = col;

}

M[row][col];

}

//----------------------------------------------------------------------

// readMatrix(Matrix, int, int)

// This function takes user inputs from the console to populate a matrix

// Returns: void

//----------------------------------------------------------------------

void readMatrix(Matrix M, int row, int col) {

for (int i = 0; i < row; i++) {

for (int j = 0; j < col; j++) {

cin >> M[i][j];

}

}

}

//----------------------------------------------------------------------

// multMatrix(Matrix, Matrix, Matrix, int, int, int)

// This function multiplies two matrixes and puts the result in a third

// Returns: void

//----------------------------------------------------------------------

void multMatrix(const Matrix A, const Matrix B, Matrix C, int m, int n, int p) {

// Initialize the new C matrix with zeroes

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

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

{

C[i][j] = 0;

}

// Triple nested loop to calculate the product of each row and col

for (int h = 0; h < m; h++) {

for (int i = 0; i < p; i++) {

for (int j = 0; j < n; j++) {

C[h][i] += A[h][j] \* B[j][i];

}

}

}

}

//----------------------------------------------------------------------

// printMatrix(Matrix, int, int)

// This function prints the matrix in a tabular form to the console

// Returns: void

//----------------------------------------------------------------------

void printMatrix(const Matrix M, int row, int col) {

cout << "----------------------------------------" << endl;

cout << "ANSWER" << endl;

cout << endl;

for (int i = 0; i < row; i++) {

for (int j = 0; j < col; j++) {

cout << fixed << setw(10) << right << setprecision(1) << M[i][j];

}

cout << endl;

}

cout << endl;

}