**C++ Part I (INFO1-CE9264) Fall 2014 – Midterm**

Clement Chan

**Question 1 – Mean and Standard Deviation**

#include <iostream>

#include <math.h>

using namespace std;

//Define the two arrays to be tested

const int MaxArr = 100;

const int MaxFib = 20;

float Array[MaxArr];

int Fib[MaxFib];

//Transferring float to int array

int arr[MaxArr];

inline void initarrays(){

//Define Array with 100

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

Array[i] = i + 1;

}

//Converting Array from float to int

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

arr[i] = int(Array[i]);

}

//Define Array with Fibonacci sequence

Fib[0] = 0;

Fib[1] = 1;

for (int i=2; i < MaxFib; i++){

Fib[i] = Fib[i-1] + Fib[i-2];

}

}

float Mean(int arr[], int Size){

float sum;

float average;

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

sum += arr[i];

}

average = sum / Size;

return average;

}

float Mean(float arr[], int Size = 10){

float sum;

float average;

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

sum += arr[i];

}

average = sum / Size;

return average;

}

float SD(int arr[], int Size){

float sumstd[Size];

float sumstdfinal;

float std;

double average = Mean(arr,Size);

double average\_1 = double(average);

cout<<"Calling Mean Function from SD: " << average\_1<<endl;

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

sumstd[i] = (arr[i] - average\_1)\*(arr[i] - average\_1);

}

sumstdfinal = 0;

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

sumstdfinal += sumstd[i];

}

std = sqrt(sumstdfinal / Size);

return std;

}

float SD(float arr[], int Size = 10){

//delete previous result

float ave;

float sumstd[Size];

float sumstdfinal;

float std;

ave = Mean(arr);

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

sumstd[i] = (arr[i] - ave)\*(arr[i] - ave);

}

sumstdfinal = 0;

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

sumstdfinal += sumstd[i];

}

std = sqrt(sumstdfinal / Size);

return std;

}

int main(){

//First initiate all value

initarrays();

//Deliver the results after array initialization

cout << "The mean for int Array[100] is: " << Mean(arr, MaxArr) <<endl;

cout << "The Standard Deviation for int Array[100] is: " << SD(arr, MaxArr)<<endl;

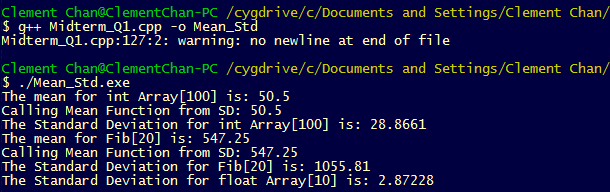
cout << "The mean for Fib[20] is: " << Mean(Fib, MaxFib) <<endl;

cout << "The Standard Deviation for Fib[20] is: " << SD(Fib, MaxFib) <<endl;

cout << "The Standard Deviation for float Array[10] is: " << SD(Array) <<endl;

}

**Output**

****

**Question 2 –Recursive Function**

#include <iostream>

using namespace std;

//Define the units needed when numbers go out of scope in terms of data types

int units;

//Define the Recursive method

inline int Pow\_Recursive(int X, int N){

units = 0;

int X1;

int \*X2;

if (N == 0){

X = 1;

return X;

}

else if (N == 1){

return X;

}

else{

X \*= Pow\_Recursive(X,N-1);

X2 = &X;

cout<<"The X's address is: " << X2 << " The X value is: " << X << endl;

if(X > 1000000000){

X1 = X/1000;

X = X1;

units = units + 3;

}

return X;

}

delete X2;

}

inline int Pow\_Alt(int X, int N){

int i=1;

double X1=X;

double Y=X;

double Final;

double m = N % 2;

units=0;

if( m == 0){

while(i<N/2){

X1 = X1\*X;

Y = Y\*X;

if(X1 > 1000000000 && Y > 1000000000){

X1 = X1 / 1000000000;

Y = Y / 1000000000;

units = units + 18;

}

i++;

}

if(X1\*Y > 1000000000){

Final = (X1\*Y) / 1000000000;

units = units + 9;

return Final;

}

else{

return X1\*Y;

}

}

else{

while(i<N/2){

X1 = X1\*X;

Y = Y\*X;

i++;

}

return X1\*Y\*X;

}

}

int getUnits(int a){

return a;

}

int main(){

//Calling Recursive Power Function

int Result;

//Recursive Power Results

Result = Pow\_Recursive(10,3);

cout<<"Result using Recursive for Pow(10,3) is: " << Result<<endl;

Result = Pow\_Recursive(2,10);

cout<<"Result using Recursive for Pow(2,10) is: " << Result<<endl;

Result = Pow\_Recursive(2,24);

cout<<"Result using Recursive for Pow(2,24) is: " << Result<<endl;

Result = Pow\_Recursive(1024,10);

cout<<"Result using Recursive for Pow(1024,10) is: " << Result<<"e+"<<units<<endl;

//Alternative Power Results

Result = Pow\_Alt(10,3);

cout<<"Result using Alternative for Pow(10,3) is: " << Result<<endl;

Result = Pow\_Alt(2,10);

cout<<"Result using Alternative for Pow(2,10) is: " << Result<<endl;

Result = Pow\_Alt(2,24);

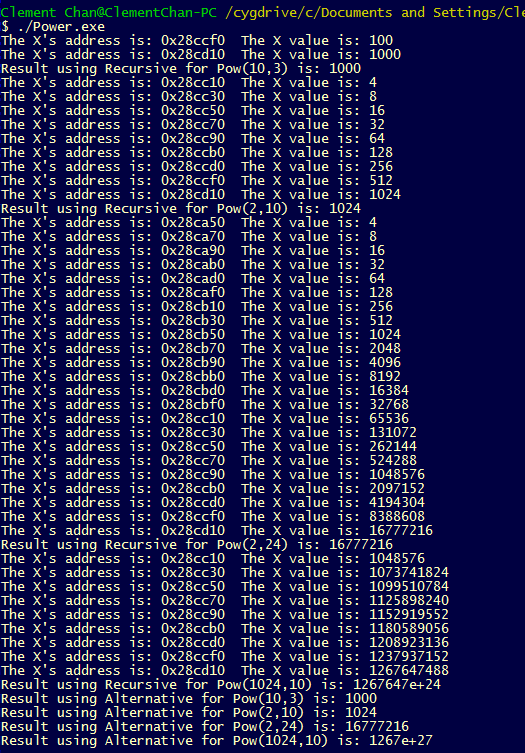
cout<<"Result using Alternative for Pow(2,24) is: " << Result<<endl;

Result = Pow\_Alt(1024,10);

cout<<"Result using Alternative for Pow(1024,10) is: " << Result<<"e+"<<units<<endl;

}

**Output**



**Question 3 – Points**

**Points.h**

class Point{

private:

int \*x;

int \*y;

public:

Point(); //Default Constructor

Point(int, int); //Parametric Constructor

Point(const Point &p); //Copy Constructor

~Point(); //Destructor

void reset\_location(int, int);

void move(int, int);

void set(int, int);

int getX();

int getX() const;

int getY();

int getY() const;

void pass\_x(); //For void Reverse use later on

};

**Points.cpp**

#include <iostream>

#include "Midterm\_Q3.h"

using namespace std;

//Method definitions in Header files

Point::Point(){

cout<<"This is Default Constructor"<<endl;

x = new int;

y = new int;

\*x=0;

\*y=0;

}

Point::Point(int a, int b){

cout<<"This is Parametarized Constructor"<<endl;

x = new int;

y = new int;

\*x=a;

\*y=b;

}

Point::Point(const Point &p){

cout<<"This is the Copy Constructor"<<endl;

x = new int;

y = new int;

\*x=\*p.x;

\*y=\*p.y;

}

void Point::reset\_location(int value\_x, int value\_y){

x = new int;

y = new int;

\*x=value\_x;

\*y=value\_y;

}

void Point::move(int delta\_x, int delta\_y){

\*x += delta\_x;

\*y += delta\_y;

}

void Point::set(int a, int b){

\*x = a;

\*y = b;

}

int Point::getX(){

return \*x;

}

int Point::getX() const{

return \*x;

}

int Point::getY(){

return \*y;

}

int Point::getY() const{

return \*y;

}

void Point::pass\_x(){

cout << \*x << endl;

}

void Reverse(Point &p1, Point p2){

p1.pass\_x();

cout<<p2.getY()<<endl;

}

Point::~Point(){

cout<<"Here is the destructor." << endl;

delete[] x;

delete[] y;

}

int main(){

const int Size = 5;

Point P1,P2(3,4);

const Point P3(10,4);

P2.move(5,10);

cout<<"After moving 5 units, P2's x value is: " << P2.getX()<<endl;

cout<<"P3's y value is: "<< P3.getY() <<endl;

//Getting the array of straight Line

Point PointArray[5];

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

PointArray[i].set(1,i);

cout<<"The x value is: " << PointArray[i].getX()<<" The y value is: " <<PointArray[i].getY() << endl;

}

//Reverse Function

Reverse(P1,P2);

}

**Output**