**Họ và Tên: Trần Thị Ngọc Diệp**

**MSSV: 1827005**

**Lớp: B2HK182 – Cấu Trúc Dữ Liệu và Giải Thuật**

------------------------

**BÀI KIỂM TRA THỰC HÀNH 1**

#include <iostream>

#include <sstream>

#include <math.h>

#include <cmath>

#include <stdbool.h>

using namespace std;

#define Eps 10e-6

//Class Point to store one point

class Point {

public:

float x;

float y;

float z;

char name;

Point \*next;

Point();

Point(float x, float y, float z, char name);

};

Point::Point(){

this->x = 0;

this->y = 0;

this->z = 0;

this->name = NULL;

this->next = NULL;

}

Point::Point(float x, float y, float z, char name){

this->x = x;

this->y = y;

this->z = z;

this->name = name;

this->next = NULL;

}

//Class List to manage a list of points

class List{

public:

int counter;

Point \*head;

List();

void addList(float x, float y, float z, char name);

};

List::List(){

counter = 0;

this->head = NULL;

}

//add a new point at the end of the List

void List::addList(float x, float y, float z, char name){

Point \*pTemp = this->head;

if (pTemp == NULL){

pTemp = new Point;

pTemp->x = x;

pTemp->y = y;

pTemp->z = z;

pTemp->name = name;

this->head = pTemp;

pTemp->next = NULL;

}

else {

Point \*pTemp2;

while (pTemp->next != NULL){

pTemp = pTemp->next;

}

pTemp2 = new Point;

pTemp2->x = x;

pTemp2->y = y;

pTemp2->z = z;

pTemp2->name = name;

pTemp2->next = NULL;

pTemp->next = pTemp2;

}

this->counter++;

}

//Check the position of a point P comparing to a circle S (I, R)

int checkP(Point P, Point I, float R){

float result;

//calculate the distance between I and P

result = sqrt(pow((P.x - I.x),2) + pow((P.y - I.y),2) + pow((P.z - I.z),2));

if (result > R){

return 1;

}

else if (result < R){

return -1;

}

else {

return 0;

}

}

//check if 3 points are collinear

char checkCollinear (Point A, Point B, Point C){

float AB, AC, BC;

//find the distance between each pair of 3 Points

AB = sqrt(pow((A.x - B.x),2) + pow((A.y - B.y),2) + pow((A.z - B.z),2));

AC = sqrt(pow((A.x - C.x),2) + pow((A.y - C.y),2) + pow((A.z - C.z),2));

BC = sqrt(pow((B.x - C.x),2) + pow((B.y - C.y),2) + pow((B.z - C.z),2));

//check if they are collinear

if (abs(AB + AC - BC) < Eps){

return A.name;

}

else if (abs(AB + BC - AC) < Eps){

return B.name;

}

else if (abs(BC + AC - AB) < Eps){

return C.name;

}

else {

return NULL;

}

}

//check for collinear points but return true or false

bool checkCollinear (Point \*A, Point \*B, Point \*C){

float AB, AC, BC;

bool flag = false;

//find the distance between each pair of 3 Points

AB = sqrt(pow((A->x - B->x),2) + pow((A->y - B->y),2) + pow((A->z - B->z),2));

AC = sqrt(pow((A->x - C->x),2) + pow((A->y - C->y),2) + pow((A->z - C->z),2));

BC = sqrt(pow((B->x - C->x),2) + pow((B->y - C->y),2) + pow((B->z - C->z),2));

//check if they are collinear

if (abs(AB + AC - BC) < Eps){

flag = true;

return flag;

}

else if (abs(AB + BC - AC) < Eps){

flag = true;

return flag;

}

else if (abs(BC + AC - AB) < Eps){

flag = true;

return flag;

}

else {

return flag;

}

}

//Check if there are collinear points in a List of points

void printCollinear(List L){

bool flag = false;

if (L.counter == 0 || L.counter == 1 || L.counter == 2){

return;

}

else if (L.counter == 3){

Point \*pTemp1 = L.head;

Point \*pTemp2 = pTemp1->next;

Point \*pTemp3 = pTemp2->next;

flag = checkCollinear(pTemp1, pTemp2, pTemp3);

if (flag == true){

cout<<pTemp1->name<<", "<<pTemp2->name<<", "<<pTemp3->name<<"\n";

}

}

else{

Point \*pTemp1 = L.head;

Point \*pTemp2 = pTemp1->next;

Point \*pTemp3 = pTemp2->next;

while (pTemp1->next->next != NULL){

while (pTemp2->next != NULL){

while (pTemp3 != NULL){

flag = checkCollinear(pTemp1, pTemp2, pTemp3);

if (flag == true){

cout<<pTemp1->name<<", "<<pTemp2->name<<", "<<pTemp3->name<<"\n";

}

flag = false;

pTemp3 = pTemp3->next;

}

pTemp2 = pTemp2->next;

pTemp3 = pTemp2->next;

}

pTemp1 = pTemp1->next;

pTemp2 = pTemp1->next;

pTemp3 = pTemp2->next;

}

}

}

//get the distance between 2 points

float findDistance(Point \*A, Point \*B){

float AB = sqrt(pow((A->x - B->x),2) + pow((A->y - B->y),2) + pow((A->z - B->z),2));

return AB;

}

//swap 2 points and their data

void switchPoint(Point \*A, Point \*B){

Point \*temp;

temp->x = A->x;

temp->y = A->y;

temp->z = A->z;

temp->name = A->name;

A->x = B->x;

A->y = B->y;

A->z = B->z;

A->name = B->name;

B->x = temp->x;

B->y = temp->y;

B->z = temp->z;

B->name = temp->name;

}

//sort a List depending on the distance between each point and I

void sortList(List &L, Point \*I){

int n = L.counter;

Point \*pTemp1 = L.head;

Point \*pTemp2 = pTemp1->next;

while (pTemp1 != NULL){

while (pTemp2 != NULL){

if (findDistance(pTemp1, I)<findDistance(pTemp2, I)){

switchPoint(pTemp1, pTemp2);

}

pTemp2 = pTemp2->next;

}

pTemp1 = pTemp1->next;

}

}

//recursive function returns the sum of distance between 2 next Points, input is the head pointer of the list

float sumDistance(Point \*head){

Point \*pTemp1 = head;

if (pTemp1 == NULL){

return 0;

}

else if (pTemp1->next == NULL){

return 0;

}

else if (pTemp1->next->next == NULL){

Point \*pTemp = head;

return findDistance(pTemp, pTemp->next);

}

else{

Point \*pTemp1 = head;

Point \*pTemp2 = pTemp1->next;

return findDistance(pTemp1, pTemp2) + sumDistance(pTemp1->next);

}

}

//Check if 3 points make a Isosceles Right Triangle

bool CheckTriagle(Point \*A, Point \*B, Point \*C){

float AB, AC, BC;

AB = findDistance(A, B);

AC = findDistance(A, C);

BC = findDistance(B, C);

if (abs(AB\*AB+AC\*AC-BC\*BC)<Eps && abs(AB-AC)<Eps){

return true;

}

else if (abs(AB\*AB+BC\*BC-AC\*AC)<Eps && abs(AB-BC)<Eps){

return true;

}

else if (abs(AC\*AC+BC\*BC-AB\*AB)<Eps && abs(AC-BC)<Eps){

return true;

}

else {

return false;

}

}

List \*CheckTriagle(List \*L){

bool flag = false;

if (L->counter == 0 || L->counter == 1 || L->counter == 2){

List \*result = new List();

return result;

}

else if (L->counter == 3){

Point \*pTemp1 = L->head;

Point \*pTemp2 = pTemp1->next;

Point \*pTemp3 = pTemp2->next;

flag = CheckTriagle(pTemp1, pTemp2, pTemp3);

if (flag == true){

return L;

}

}

else{

Point \*pTemp1 = L->head;

Point \*pTemp2 = pTemp1->next;

Point \*pTemp3 = pTemp2->next;

List \*result = new List();

while (pTemp1->next->next != NULL){

while (pTemp2->next != NULL){

while (pTemp3 != NULL){

flag = CheckTriagle(pTemp1, pTemp2, pTemp3);

if (flag == true){

result->addList(pTemp1->x, pTemp1->y, pTemp1->z, pTemp1->name);

result->addList(pTemp2->x, pTemp2->y, pTemp2->z, pTemp2->name);

result->addList(pTemp3->x, pTemp3->y, pTemp3->z, pTemp3->name);

goto TheEnd;

}

pTemp3 = pTemp3->next;

}

pTemp2 = pTemp2->next;

pTemp3 = pTemp2->next;

}

pTemp1 = pTemp1->next;

pTemp2 = pTemp1->next;

pTemp3 = pTemp2->next;

}

TheEnd:

return result;

}

}

void printList(List L){

Point \*pTemp = L.head;

while (pTemp != NULL){

cout<<pTemp->name<<" ("<<pTemp->x<<", "<<pTemp->y<<", "<<pTemp->z<<")\n";

pTemp = pTemp->next;

}

}

int main(){

/\*

//Check for question 1

List G1;

G1.addList(3,4,5, 'I');

G1.addList(3,0,5, 'A');

G1.addList(3,4,0, 'B');

printList(G1);

\*/

/\*

//Check for question 2

Point I(3,4,5, 'I');

Point A(3,0,5, 'A');

Point B(3,4,0, 'B');

Point C(0,0,0, 'C');

cout<<checkP(A, I, 5)<<"\n";

cout<<checkP(B, I, 5)<<"\n";

cout<<checkP(C, I, 5)<<"\n";

\*/

/\*

//Check for question 3a

Point A(1,1,1, 'A');

Point B(2,2,2, 'B');

Point C(-1,-1,-1, 'C');

cout<<checkCollinear(B, A, C)<<"\n";

\*/

/\*

//Check for question 3b

List G;

G.addList(1,1,1, 'A');

G.addList(2,2,2, 'B');

G.addList(-1,-1,-1, 'C');

G.addList(2,3,3, 'D');

G.addList(-2,-2,-2, 'E');

printCollinear(G);

\*/}

**Câu 1:**

int main(){

//Check for question 1

List G1;

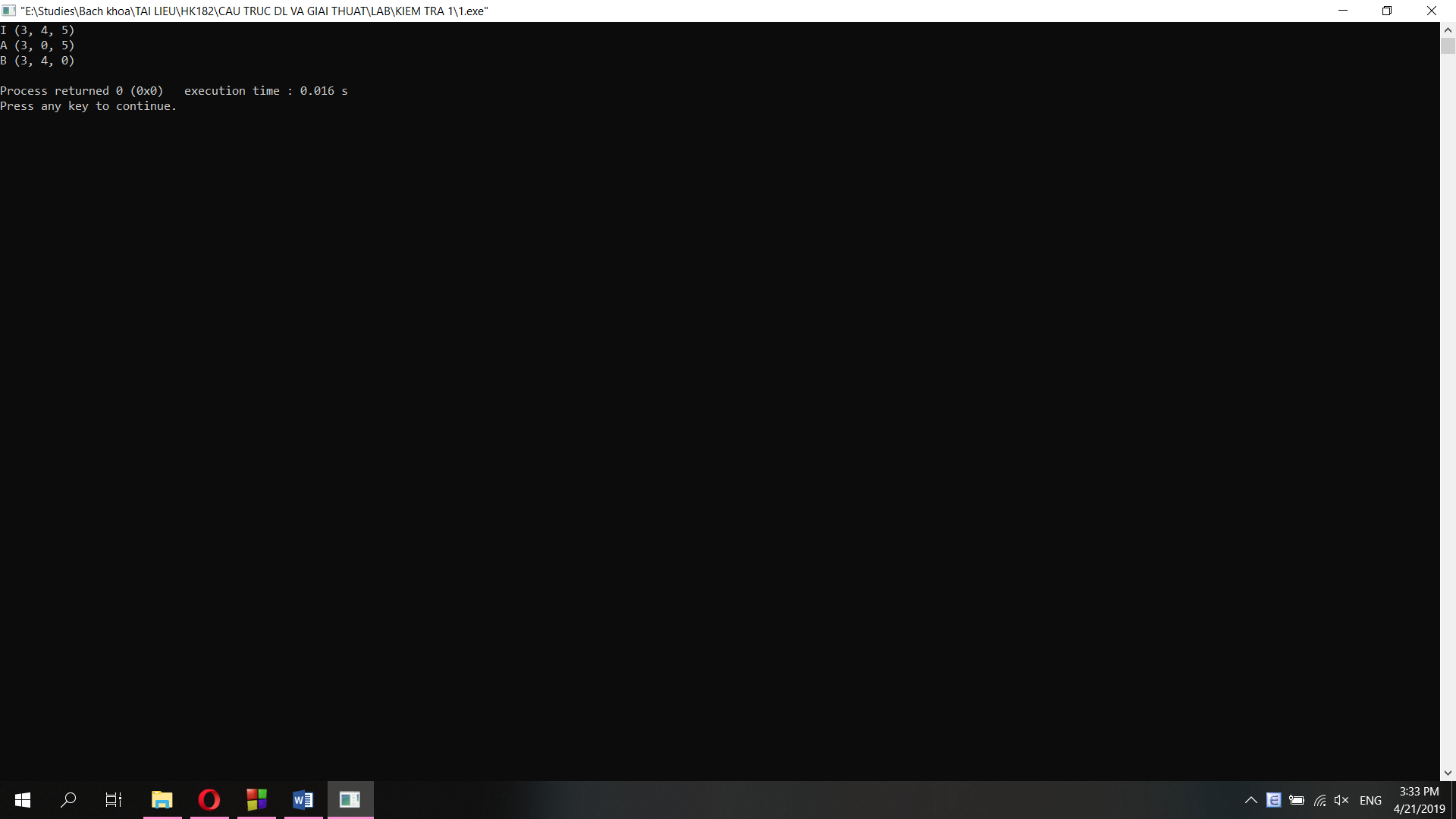
G1.addList(3,4,5, 'I');

G1.addList(3,0,5, 'A');

G1.addList(3,4,0, 'B');

printList(G1);

}



**Câu 2:**

int main(){

Point I(3,4,5, 'I');

Point A(3,0,5, 'A');

Point B(3,4,0, 'B');

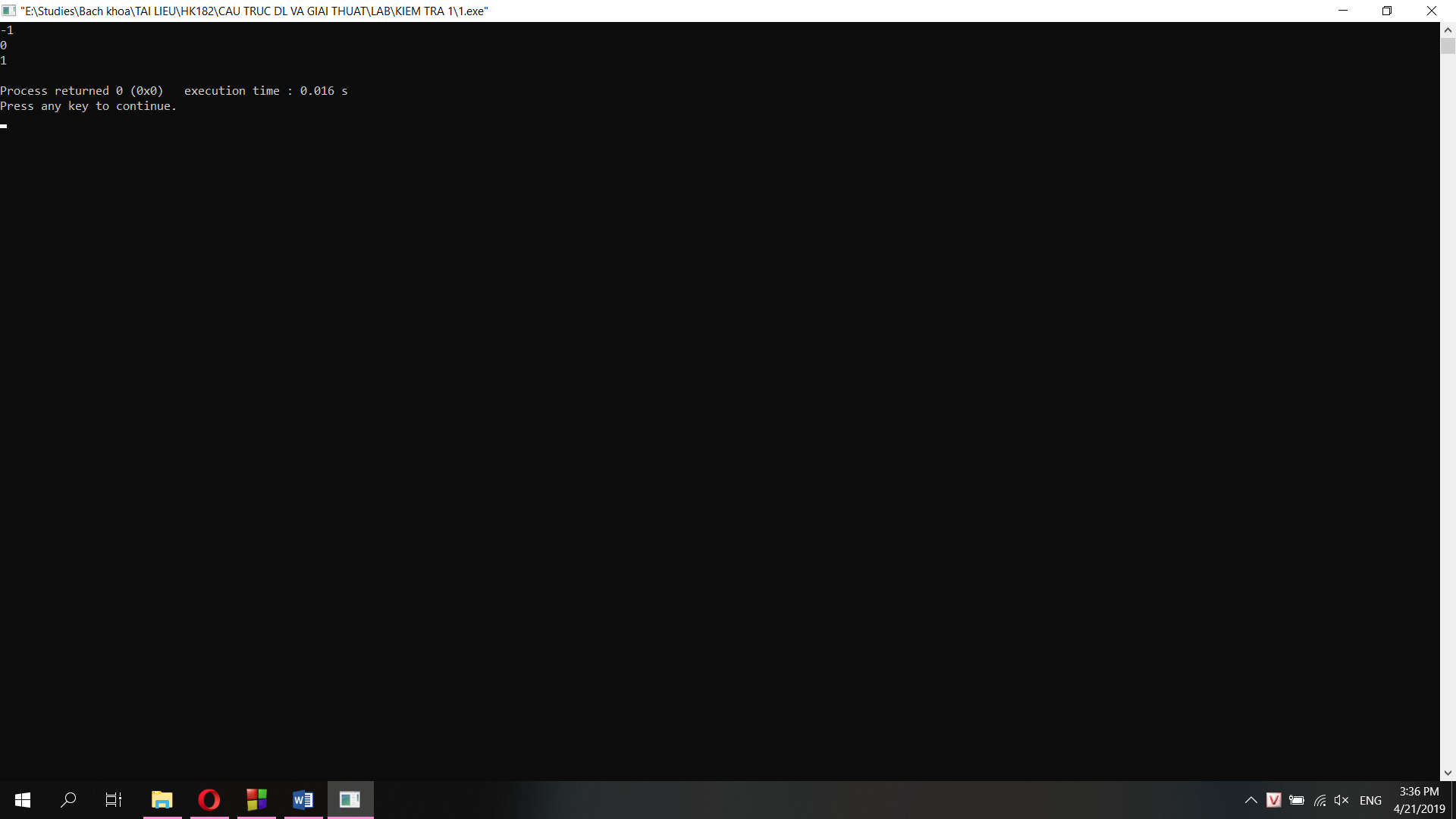
Point C(0,0,0, 'C');

cout<<checkP(A, I, 5)<<"\n";

cout<<checkP(B, I, 5)<<"\n";

cout<<checkP(C, I, 5)<<"\n";

}



**Câu 3:**

int main(){

List G;

G.addList(1,1,1, 'A');

G.addList(2,2,2, 'B');

G.addList(-1,-1,-1, 'C');

G.addList(2,3,3, 'D');

G.addList(-2,-2,-2, 'E');

printCollinear(G);

}

