Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 3 з дисципліни
«Основи програмування 2. Модульне програмування»
«Класи і об’єкти»
Варіант 7

Виконав студент ІП-15, Гуменюк Олександр Володимирович

(шифр, прізвище, ім'я, по батькові)

Перевірила Вєчерковська Анастасія Сергіївна

(прізвище, ім'я, по батькові)

Київ 2022

Лабораторна робота 3

Класи і об’єкти

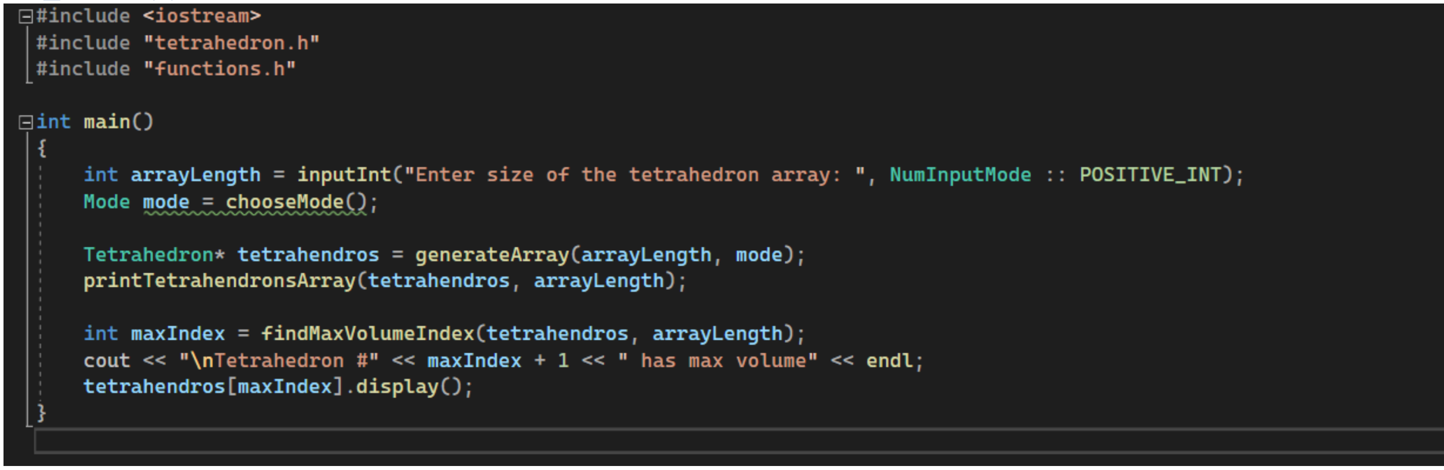
Індивідуальне завдання

Варіант 7

7. Розробити клас ”тетраедр”, який заданий координатами своїх вершин в просторі. Створити масив об’єктів даного класу. Визначити тетраедр з найбільшим об’ємом.

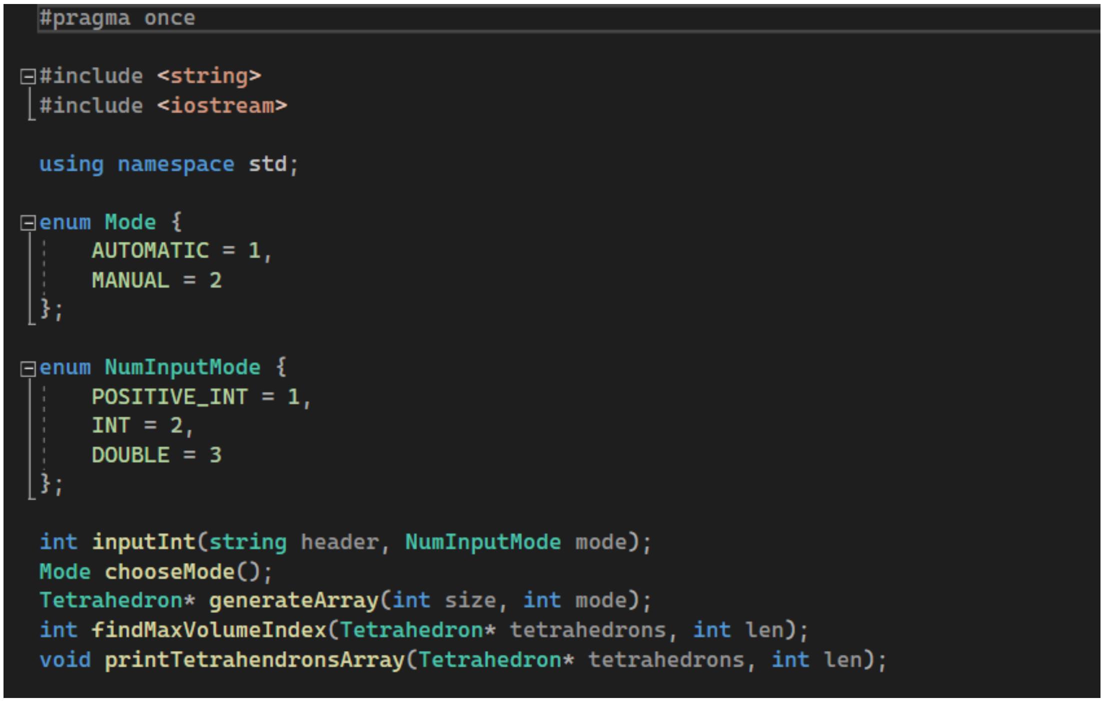
Код C++

OP_Lab1.cpp



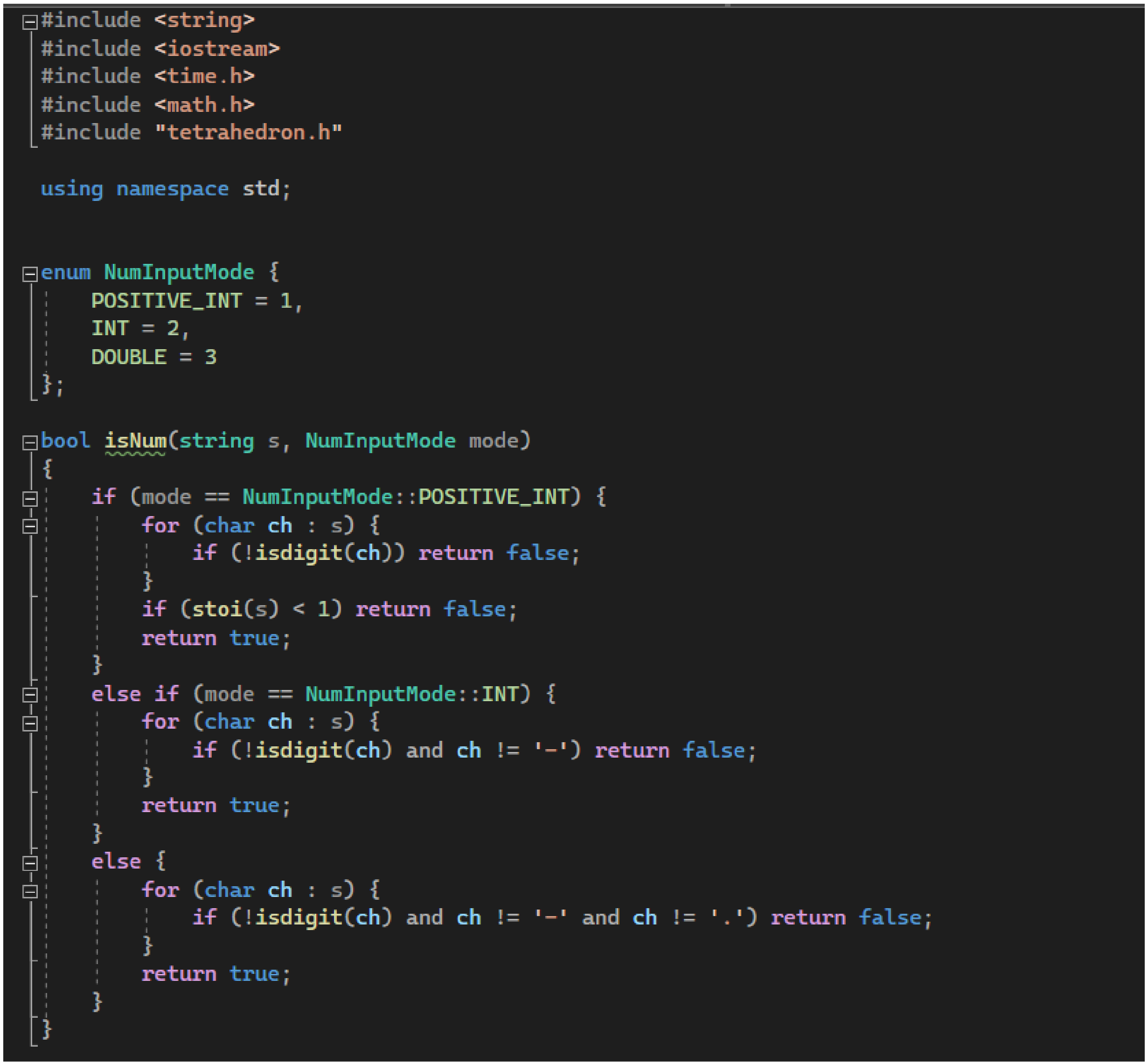
```
#include <iostream>
#include "tetrahedron.h"
#include "functions.h"

int main()
{
    int arrayLength = inputInt("Enter size of the tetrahedron array: ", NumInputMode :: POSITIVE_INT);
    Mode mode = chooseMode();

    Tetrahedron* tetrahendros = generateArray(arrayLength, mode);
    printTetrahendronsArray(tetrahendros, arrayLength);

    int maxIndex = findMaxVolumeIndex(tetrahendros, arrayLength);
    cout << "\nTetrahedron #" << maxIndex + 1 << " has max volume" << endl;
    tetrahendros[maxIndex].display();
}
```

functions.h

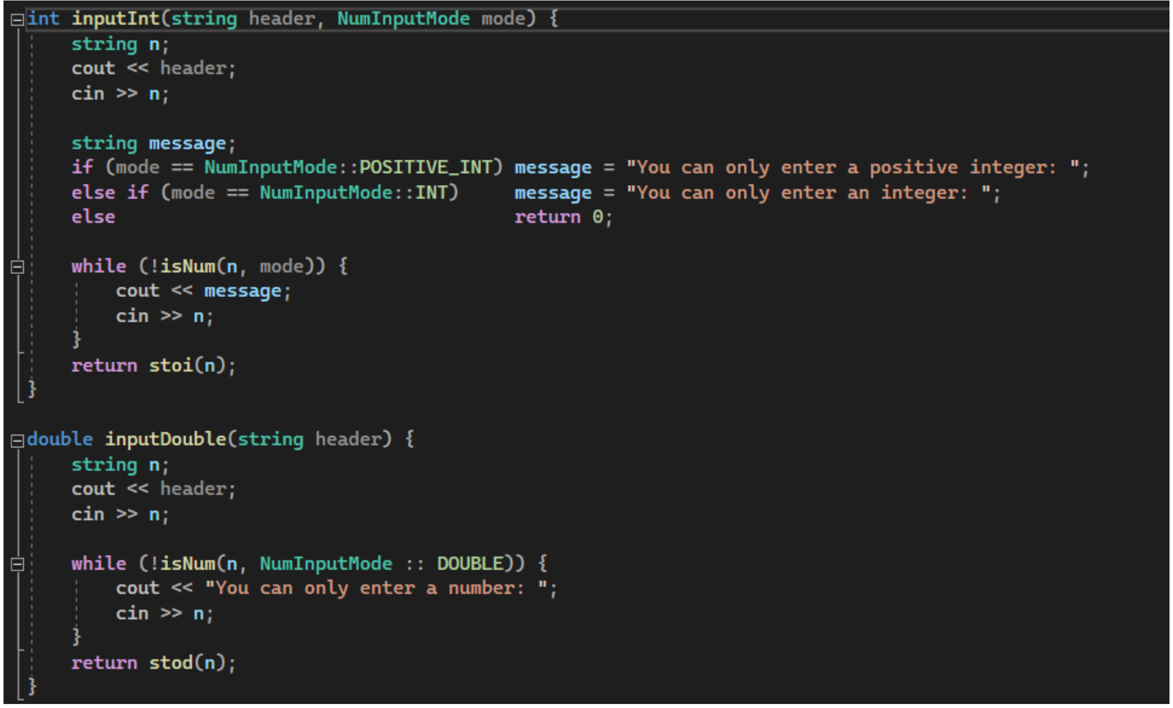


```
#pragma once

#include <string>
#include <iostream>

using namespace std;

enum Mode {
    AUTOMATIC = 1,
    MANUAL = 2
};

enum NumInputMode {
    POSITIVE_INT = 1,
    INT = 2,
    DOUBLE = 3
};

int inputInt(string header, NumInputMode mode);
Mode chooseMode();
Tetrahedron* generateArray(int size, int mode);
int findMaxVolumeIndex(Tetrahedron* tetrahedrons, int len);
void printTetrahendronsArray(Tetrahedron* tetrahedrons, int len);
```

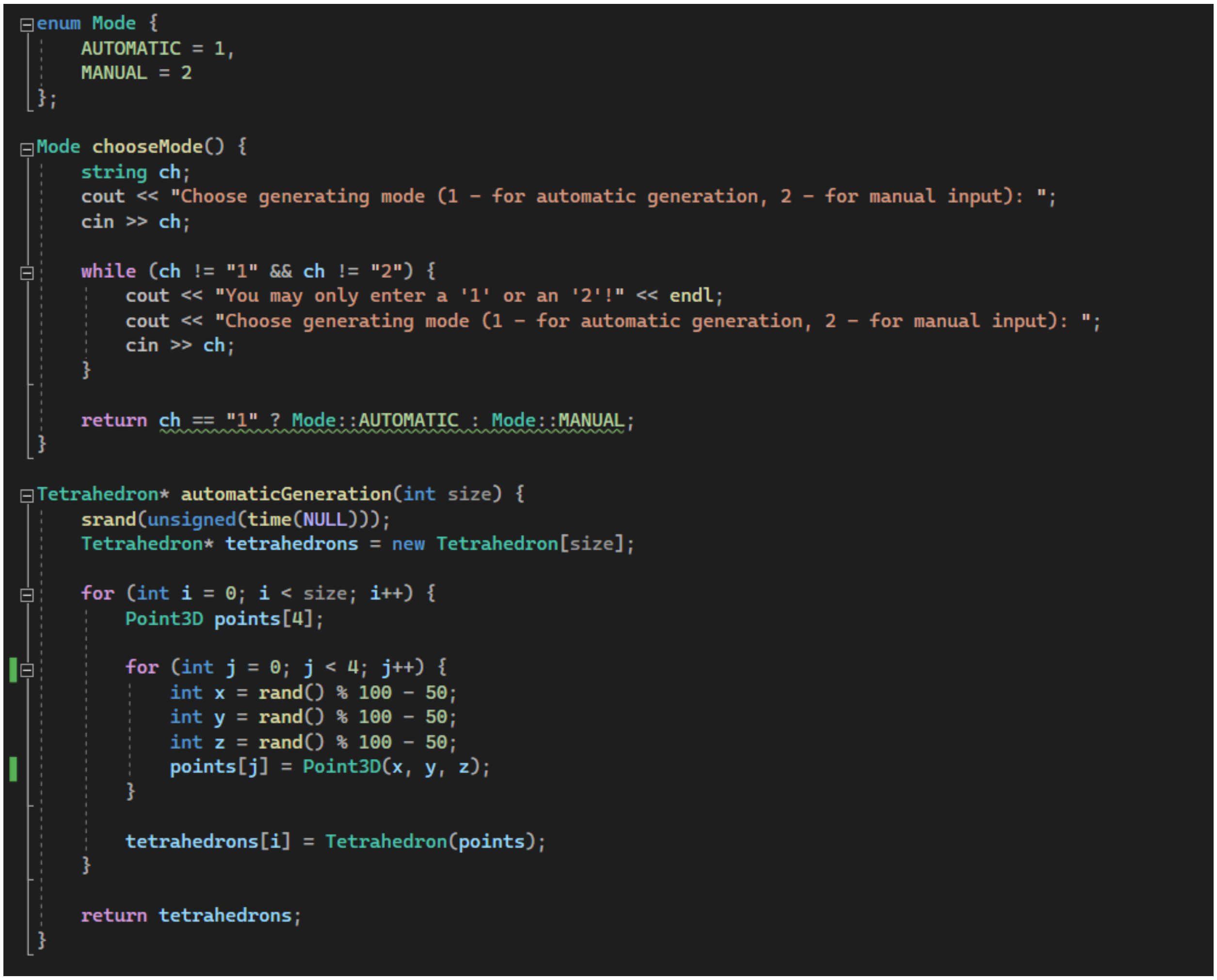
functions.cpp



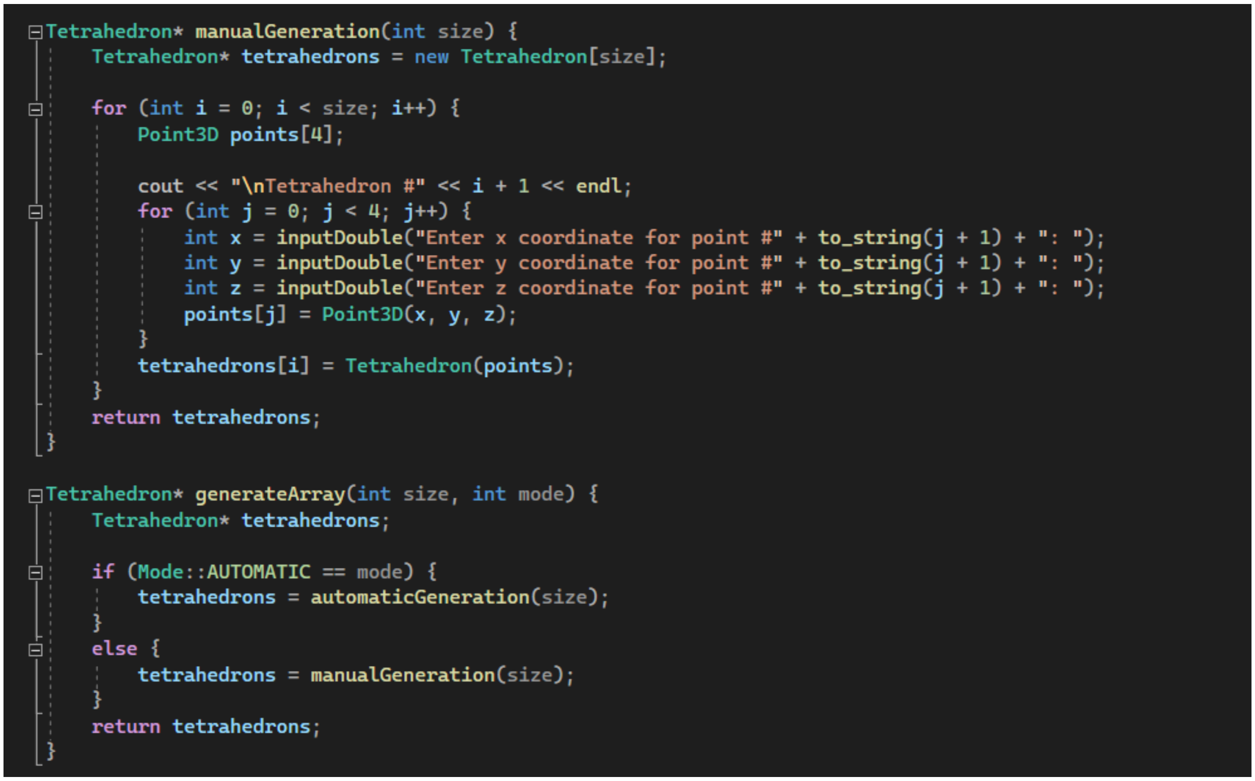
```cpp
#include <string>
#include <iostream>
#include <time.h>
#include <math.h>
#include "tetrahedron.h"

using namespace std;


enum NumInputMode {
    POSITIVE_INT = 1,
    INT = 2,
    DOUBLE = 3
};

bool isNum(string s, NumInputMode mode)
{
    if (mode == NumInputMode::POSITIVE_INT) {
        for (char ch : s) {
            if (!isdigit(ch)) return false;
        }
        if (stoi(s) < 1) return false;
        return true;
    }
    else if (mode == NumInputMode::INT) {
        for (char ch : s) {
            if (!isdigit(ch) and ch != '-') return false;
        }
        return true;
    }
    else {
        for (char ch : s) {
            if (!isdigit(ch) and ch != '-' and ch != '.') return false;
        }
        return true;
    }
}
```



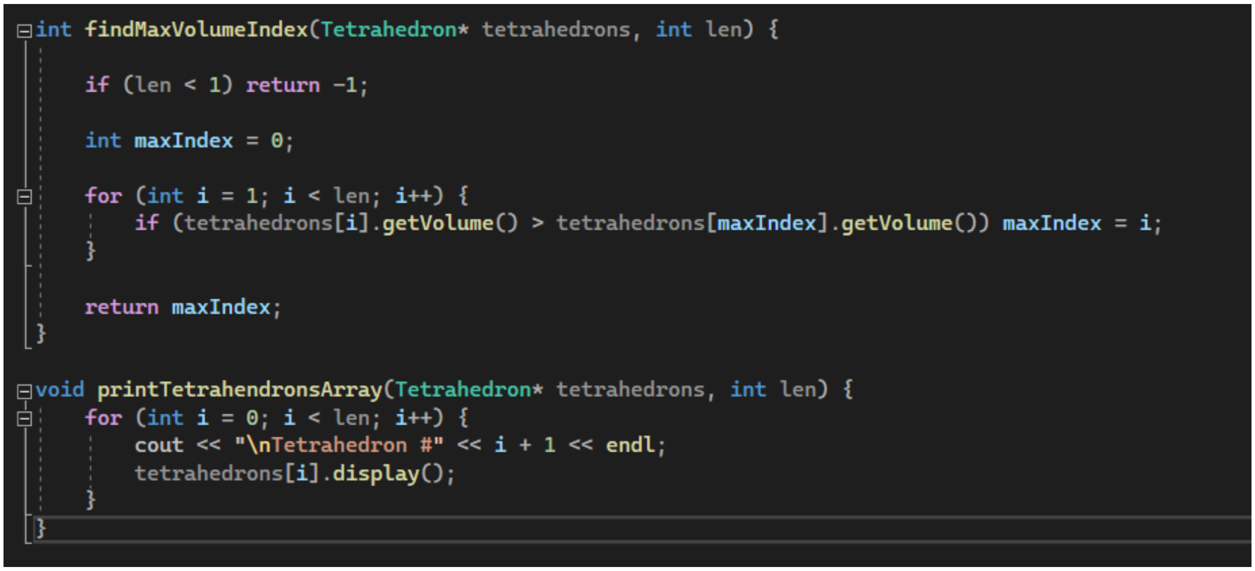
```cpp
int inputInt(string header, NumInputMode mode) {
    string n;
    cout << header;
    cin >> n;

    string message;
    if (mode == NumInputMode::POSITIVE_INT) message = "You can only enter a positive integer: ";
    else if (mode == NumInputMode::INT)     message = "You can only enter an integer: ";
    else                                    return 0;

    while (!isNum(n, mode)) {
        cout << message;
        cin >> n;
    }
    return stoi(n);
}

double inputDouble(string header) {
    string n;
    cout << header;
    cin >> n;

    while (!isNum(n, NumInputMode :: DOUBLE)) {
        cout << "You can only enter a number: ";
        cin >> n;
    }
    return stod(n);
}
```



```
enum Mode {
    AUTOMATIC = 1,
    MANUAL = 2
};

Mode chooseMode() {
    string ch;
    cout << "Choose generating mode (1 - for automatic generation, 2 - for manual input): ";
    cin >> ch;

    while (ch != "1" && ch != "2") {
        cout << "You may only enter a '1' or an '2'!" << endl;
        cout << "Choose generating mode (1 - for automatic generation, 2 - for manual input): ";
        cin >> ch;
    }

    return ch == "1" ? Mode::AUTOMATIC : Mode::MANUAL;
}

Tetrahedron* automaticGeneration(int size) {
    srand(unsigned(time(NULL)));
    Tetrahedron* tetrahedrons = new Tetrahedron[size];

    for (int i = 0; i < size; i++) {
        Point3D points[4];

        for (int j = 0; j < 4; j++) {
            int x = rand() % 100 - 50;
            int y = rand() % 100 - 50;
            int z = rand() % 100 - 50;
            points[j] = Point3D(x, y, z);
        }

        tetrahedrons[i] = Tetrahedron(points);
    }

    return tetrahedrons;
}
```

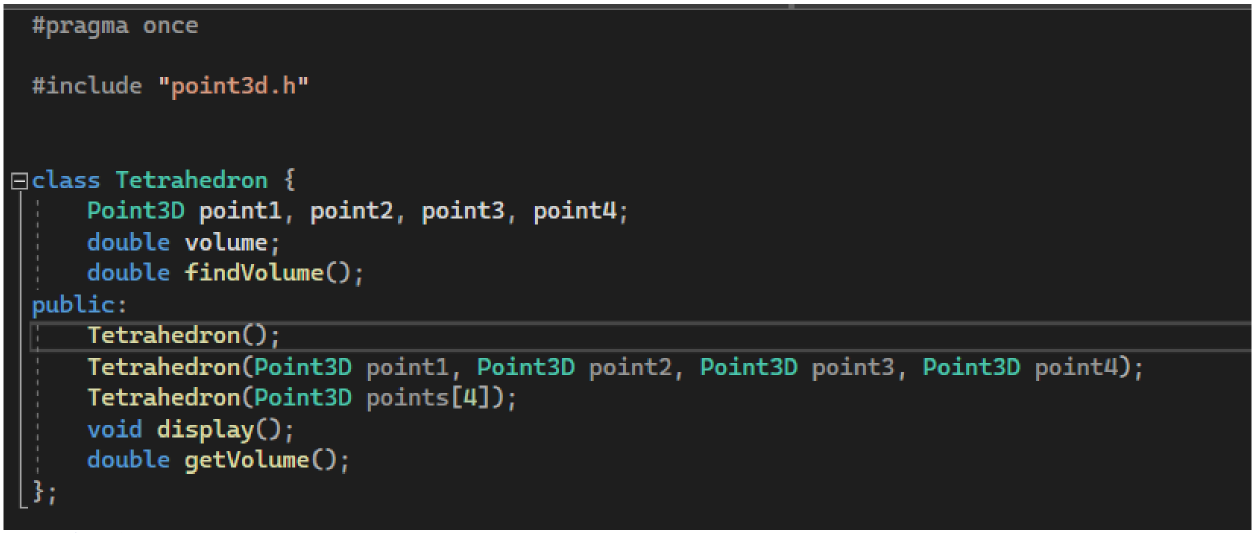


```
Tetrahedron* manualGeneration(int size) {
    Tetrahedron* tetrahedrons = new Tetrahedron[size];

    for (int i = 0; i < size; i++) {
        Point3D points[4];

        cout << "\nTetrahedron #" << i + 1 << endl;
        for (int j = 0; j < 4; j++) {
            int x = inputDouble("Enter x coordinate for point #" + to_string(j + 1) + ": ");
            int y = inputDouble("Enter y coordinate for point #" + to_string(j + 1) + ": ");
            int z = inputDouble("Enter z coordinate for point #" + to_string(j + 1) + ": ");
            points[j] = Point3D(x, y, z);
        }
        tetrahedrons[i] = Tetrahedron(points);
    }
    return tetrahedrons;
}

Tetrahedron* generateArray(int size, int mode) {
    Tetrahedron* tetrahedrons;

    if (Mode::AUTOMATIC == mode) {
        tetrahedrons = automaticGeneration(size);
    }
    else {
        tetrahedrons = manualGeneration(size);
    }
    return tetrahedrons;
}
```



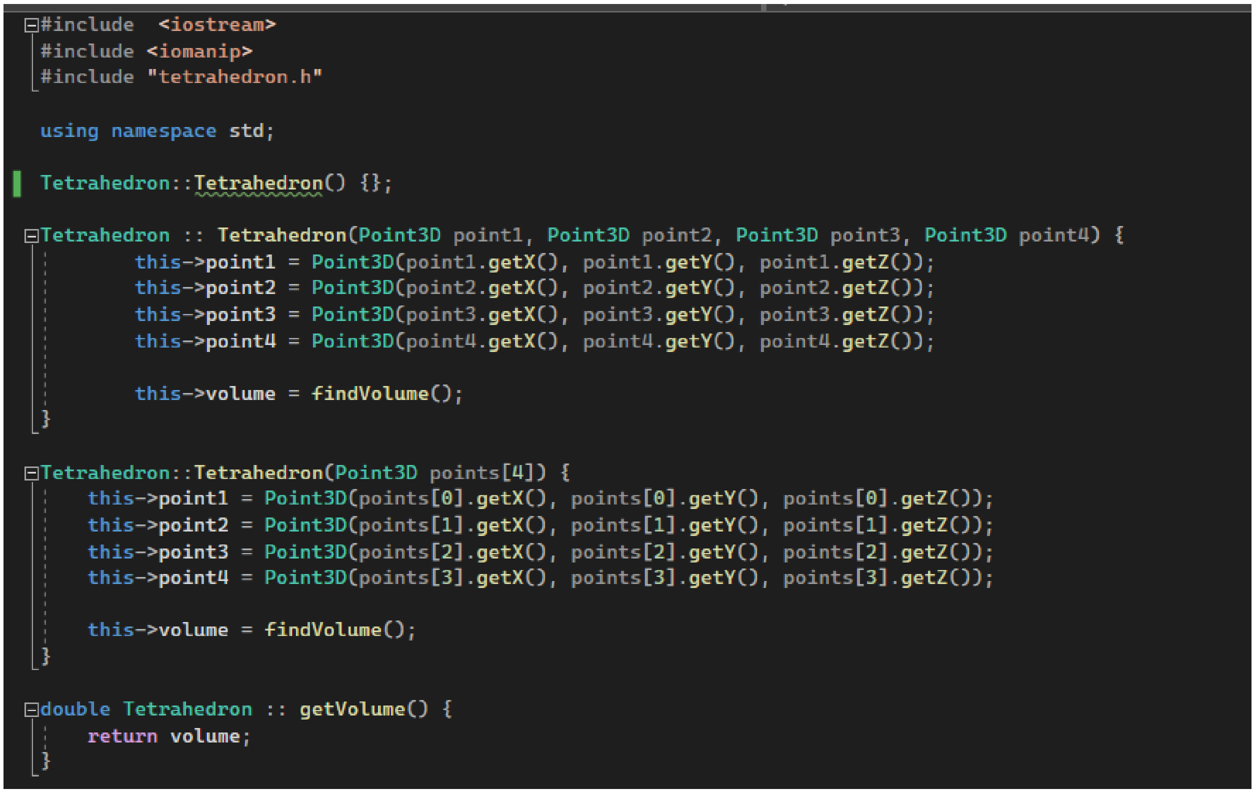
```
int findMaxVolumeIndex(Tetrahedron* tetrahedrons, int len) {

    if (len < 1) return -1;

    int maxIndex = 0;

    for (int i = 1; i < len; i++) {
        if (tetrahedrons[i].getVolume() > tetrahedrons[maxIndex].getVolume()) maxIndex = i;
    }

    return maxIndex;
}

void printTetrahendronsArray(Tetrahedron* tetrahedrons, int len) {
    for (int i = 0; i < len; i++) {
        cout << "\nTetrahedron #" << i + 1 << endl;
        tetrahedrons[i].display();
    }
}
```

tetrahedron.h

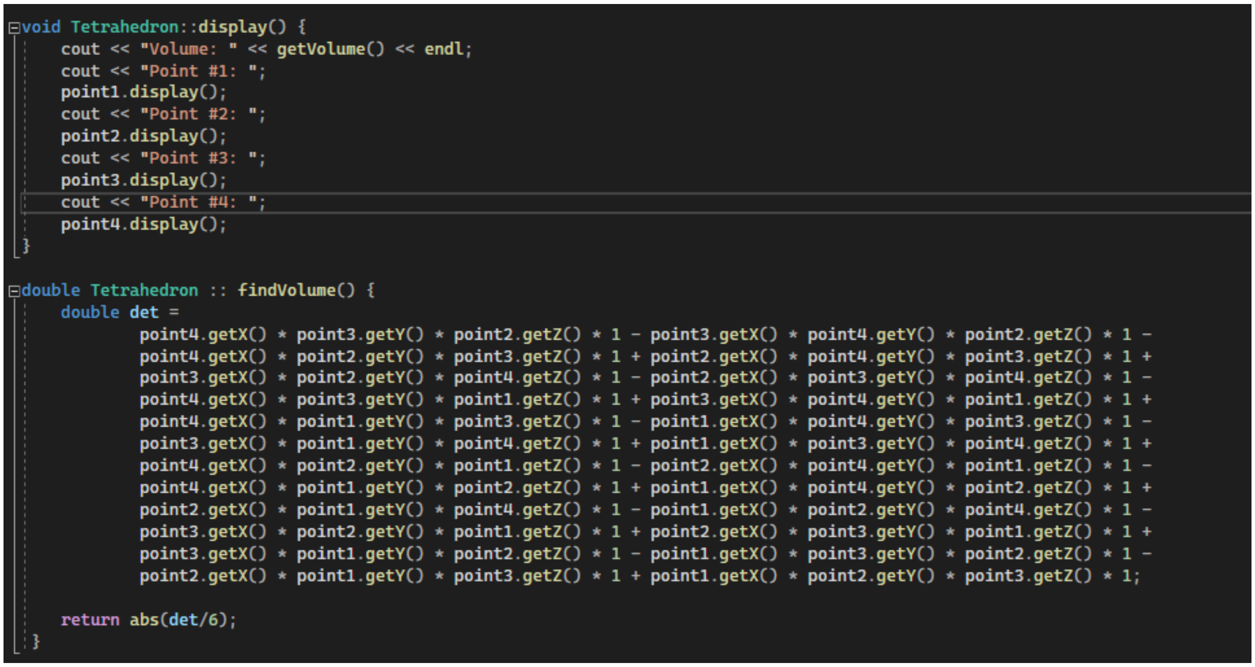


```
#pragma once

#include "point3d.h"


class Tetrahedron {
    Point3D point1, point2, point3, point4;
    double volume;
    double findVolume();
public:
    Tetrahedron();
    Tetrahedron(Point3D point1, Point3D point2, Point3D point3, Point3D point4);
    Tetrahedron(Point3D points[4]);
    void display();
    double getVolume();
};
```

tetrahedron.cpp

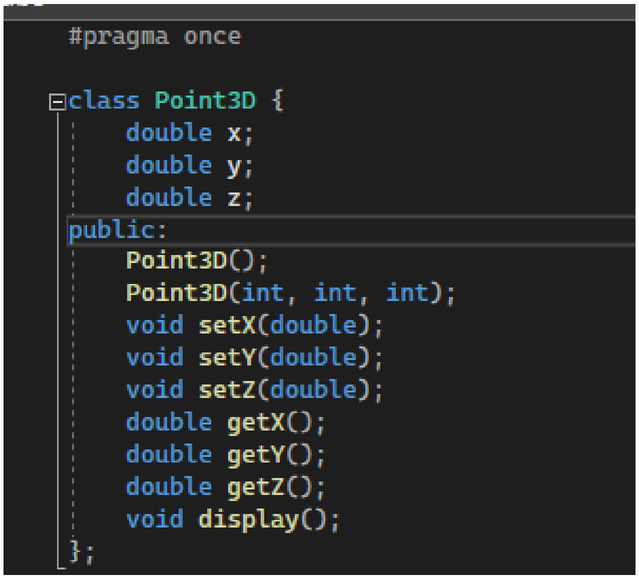


```
#include  <iostream>
#include <iomanip>
#include "tetrahedron.h"

using namespace std;

Tetrahedron::Tetrahedron() {};

Tetrahedron :: Tetrahedron(Point3D point1, Point3D point2, Point3D point3, Point3D point4) {
        this->point1 = Point3D(point1.getX(), point1.getY(), point1.getZ());
        this->point2 = Point3D(point2.getX(), point2.getY(), point2.getZ());
        this->point3 = Point3D(point3.getX(), point3.getY(), point3.getZ());
        this->point4 = Point3D(point4.getX(), point4.getY(), point4.getZ());

        this->volume = findVolume();
}

Tetrahedron::Tetrahedron(Point3D points[4]) {
    this->point1 = Point3D(points[0].getX(), points[0].getY(), points[0].getZ());
    this->point2 = Point3D(points[1].getX(), points[1].getY(), points[1].getZ());
    this->point3 = Point3D(points[2].getX(), points[2].getY(), points[2].getZ());
    this->point4 = Point3D(points[3].getX(), points[3].getY(), points[3].getZ());

    this->volume = findVolume();
}

double Tetrahedron :: getVolume() {
    return volume;
}
```



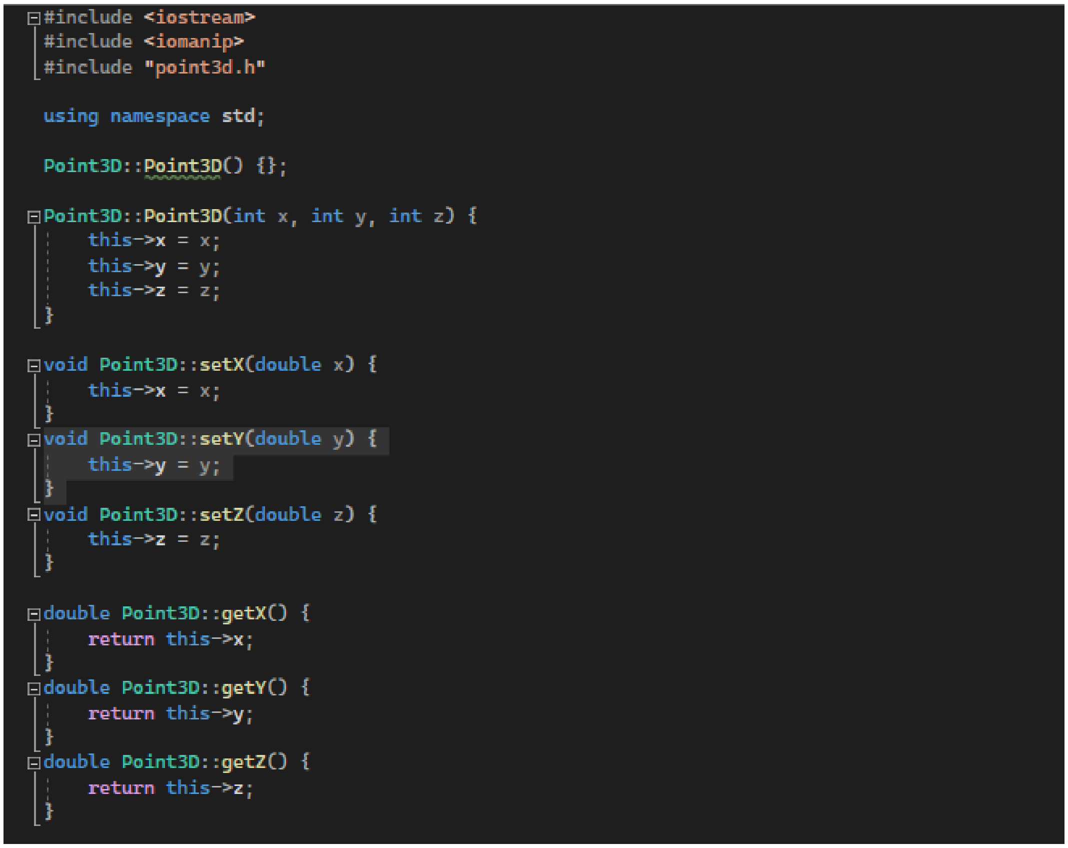
```cpp
void Tetrahedron::display() {
    cout << "Volume: " << getVolume() << endl;
    cout << "Point #1: ";
    point1.display();
    cout << "Point #2: ";
    point2.display();
    cout << "Point #3: ";
    point3.display();
    cout << "Point #4: ";
    point4.display();
}

double Tetrahedron :: findVolume() {
    double det =
            point4.getX() * point3.getY() * point2.getZ() * 1 - point3.getX() * point4.getY() * point2.getZ() * 1 -
            point4.getX() * point2.getY() * point3.getZ() * 1 + point2.getX() * point4.getY() * point3.getZ() * 1 +
            point3.getX() * point2.getY() * point4.getZ() * 1 - point2.getX() * point3.getY() * point4.getZ() * 1 -
            point4.getX() * point3.getY() * point1.getZ() * 1 + point3.getX() * point4.getY() * point1.getZ() * 1 +
            point4.getX() * point1.getY() * point3.getZ() * 1 - point1.getX() * point4.getY() * point3.getZ() * 1 -
            point3.getX() * point1.getY() * point4.getZ() * 1 + point1.getX() * point3.getY() * point4.getZ() * 1 +
            point4.getX() * point2.getY() * point1.getZ() * 1 - point2.getX() * point4.getY() * point1.getZ() * 1 -
            point4.getX() * point1.getY() * point2.getZ() * 1 + point1.getX() * point4.getY() * point2.getZ() * 1 +
            point2.getX() * point1.getY() * point4.getZ() * 1 - point1.getX() * point2.getY() * point4.getZ() * 1 -
            point3.getX() * point2.getY() * point1.getZ() * 1 + point2.getX() * point3.getY() * point1.getZ() * 1 +
            point3.getX() * point1.getY() * point2.getZ() * 1 - point1.getX() * point3.getY() * point2.getZ() * 1 -
            point2.getX() * point1.getY() * point3.getZ() * 1 + point1.getX() * point2.getY() * point3.getZ() * 1;

    return abs(det/6);
}
```

point3d.h

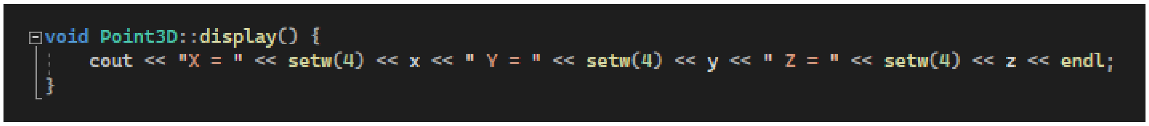


```cpp
#pragma once

class Point3D {
    double x;
    double y;
    double z;
public:
    Point3D();
    Point3D(int, int, int);
    void setX(double);
    void setY(double);
    void setZ(double);
    double getX();
    double getY();
    double getZ();
    void display();
};
```

point3d.cpp

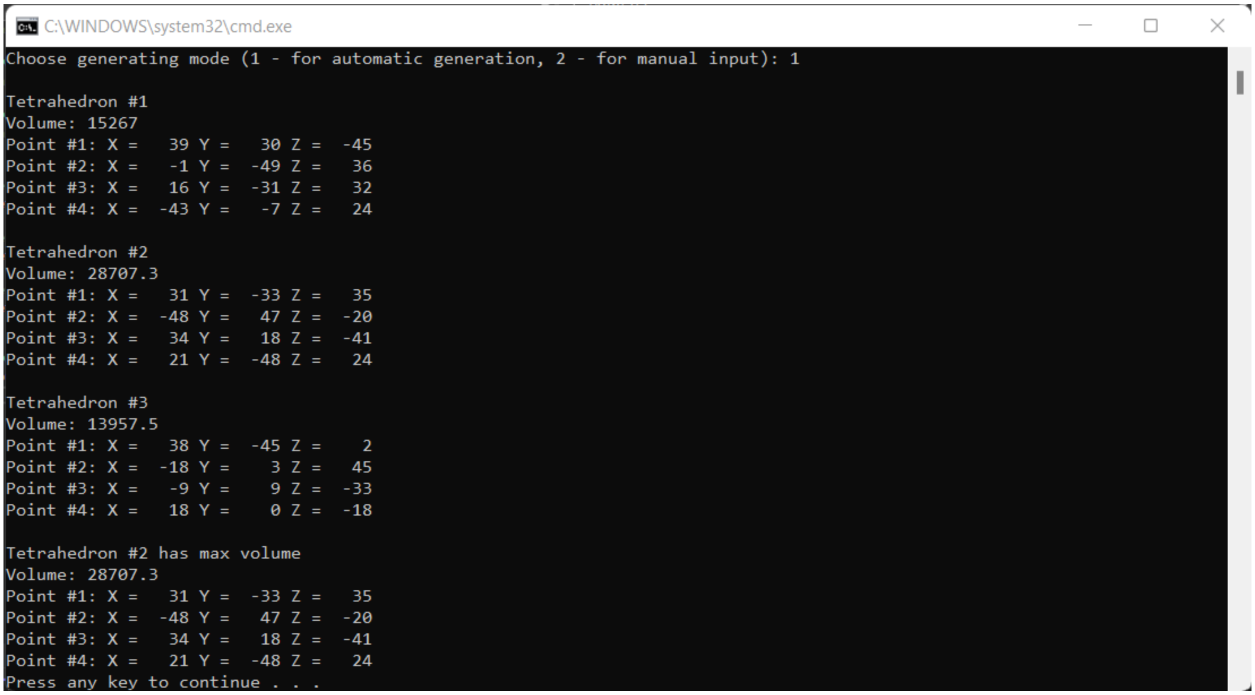


```cpp
#include <iostream>
#include <iomanip>
#include "point3d.h"

using namespace std;

Point3D::Point3D() {};

Point3D::Point3D(int x, int y, int z) {
    this->x = x;
    this->y = y;
    this->z = z;
}

void Point3D::setX(double x) {
    this->x = x;
}
void Point3D::setY(double y) {
    this->y = y;
}
void Point3D::setZ(double z) {
    this->z = z;
}

double Point3D::getX() {
    return this->x;
}
double Point3D::getY() {
    return this->y;
}
double Point3D::getZ() {
    return this->z;
}
```

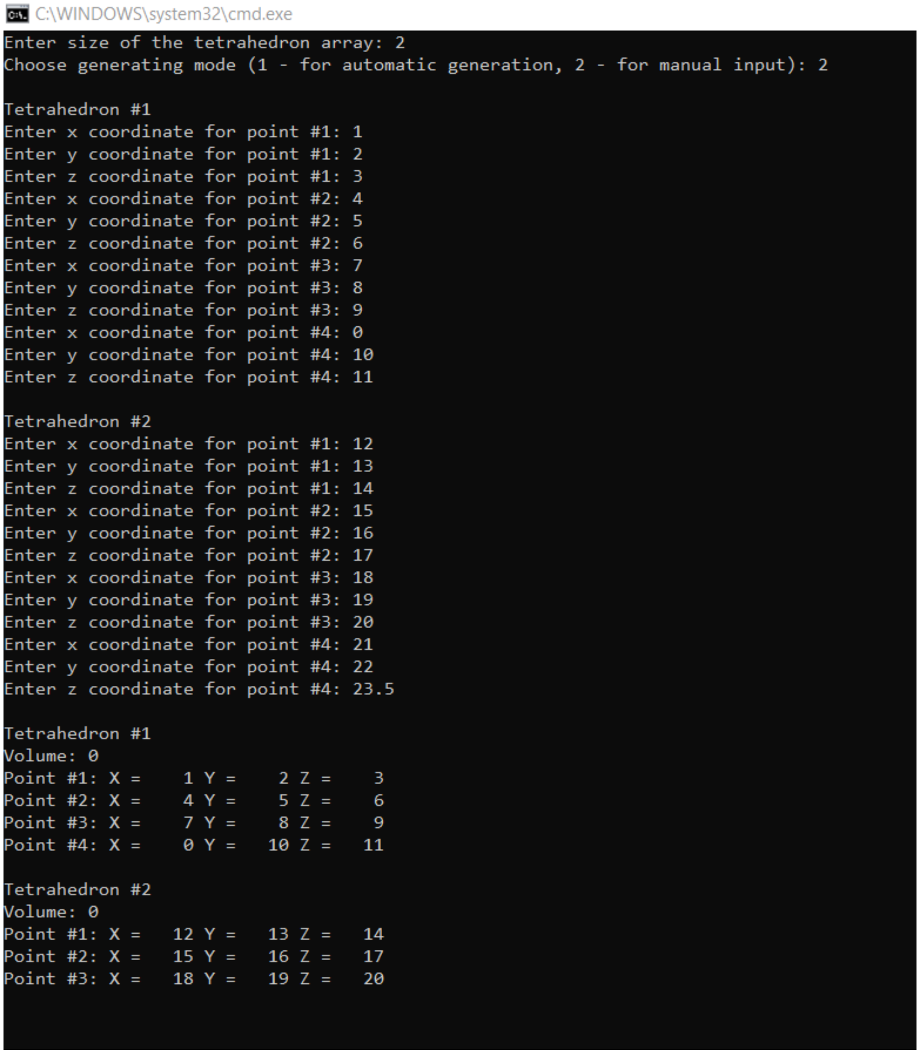


```cpp
void Point3D::display() {
    cout << "X = " << setw(4) << x << " Y = " << setw(4) << y << " Z = " << setw(4) << z << endl;
}
```

Тестування C++



```
Choose generating mode (1 - for automatic generation, 2 - for manual input): 1

Tetrahedron #1
Volume: 15267
Point #1: X =   39 Y =   30 Z =  -45
Point #2: X =   -1 Y =  -49 Z =   36
Point #3: X =   16 Y =  -31 Z =   32
Point #4: X =  -43 Y =   -7 Z =   24

Tetrahedron #2
Volume: 28707.3
Point #1: X =   31 Y =  -33 Z =   35
Point #2: X =  -48 Y =   47 Z =  -20
Point #3: X =   34 Y =   18 Z =  -41
Point #4: X =   21 Y =  -48 Z =   24

Tetrahedron #3
Volume: 13957.5
Point #1: X =   38 Y =  -45 Z =    2
Point #2: X =  -18 Y =    3 Z =   45
Point #3: X =   -9 Y =    9 Z =  -33
Point #4: X =   18 Y =    0 Z =  -18

Tetrahedron #2 has max volume
Volume: 28707.3
Point #1: X =   31 Y =  -33 Z =   35
Point #2: X =  -48 Y =   47 Z =  -20
Point #3: X =   34 Y =   18 Z =  -41
Point #4: X =   21 Y =  -48 Z =   24
Press any key to continue . . .
```



```
Enter size of the tetrahedron array: 2
Choose generating mode (1 - for automatic generation, 2 - for manual input): 2

Tetrahedron #1
Enter x coordinate for point #1: 1
Enter y coordinate for point #1: 2
Enter z coordinate for point #1: 3
Enter x coordinate for point #2: 4
Enter y coordinate for point #2: 5
Enter z coordinate for point #2: 6
Enter x coordinate for point #3: 7
Enter y coordinate for point #3: 8
Enter z coordinate for point #3: 9
Enter x coordinate for point #4: 0
Enter y coordinate for point #4: 10
Enter z coordinate for point #4: 11

Tetrahedron #2
Enter x coordinate for point #1: 12
Enter y coordinate for point #1: 13
Enter z coordinate for point #1: 14
Enter x coordinate for point #2: 15
Enter y coordinate for point #2: 16
Enter z coordinate for point #2: 17
Enter x coordinate for point #3: 18
Enter y coordinate for point #3: 19
Enter z coordinate for point #3: 20
Enter x coordinate for point #4: 21
Enter y coordinate for point #4: 22
Enter z coordinate for point #4: 23.5

Tetrahedron #1
Volume: 0
Point #1: X =    1 Y =    2 Z =    3
Point #2: X =    4 Y =    5 Z =    6
Point #3: X =    7 Y =    8 Z =    9
Point #4: X =    0 Y =   10 Z =   11

Tetrahedron #2
Volume: 0
Point #1: X =   12 Y =   13 Z =   14
Point #2: X =   15 Y =   16 Z =   17
Point #3: X =   18 Y =   19 Z =   20
```