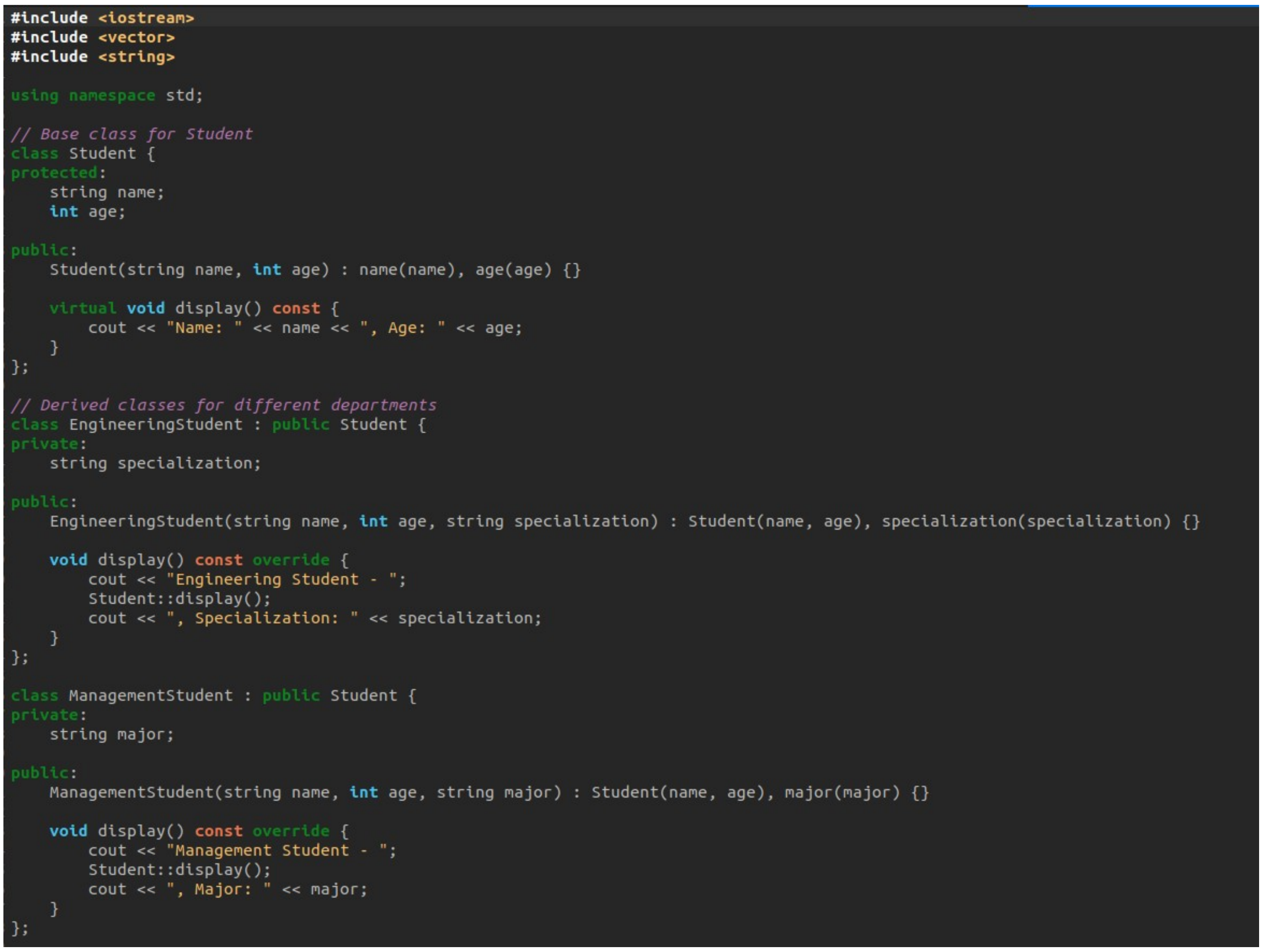
Nithin S
221IT085

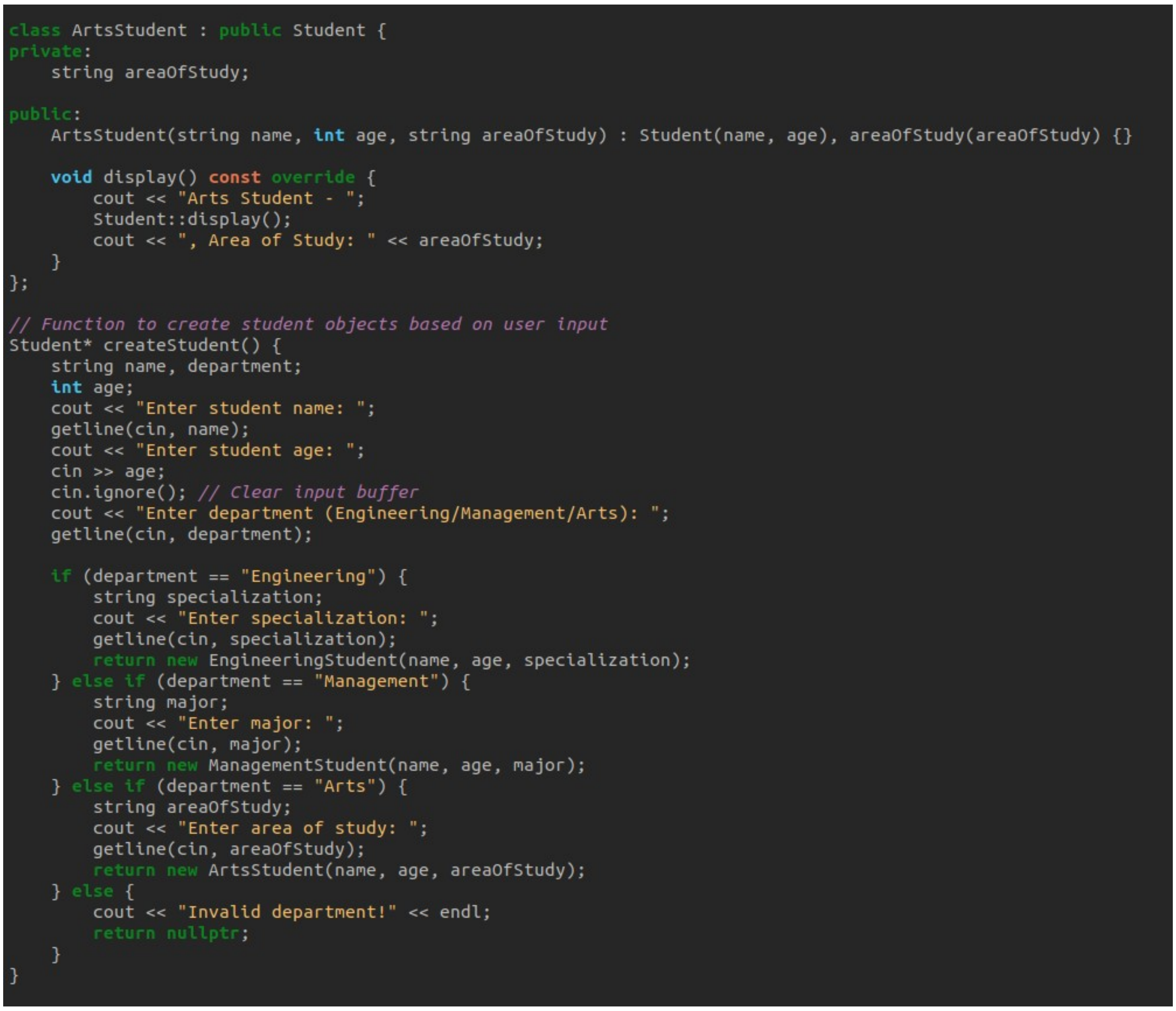
IT150 Class Assignment

1. Implement a system to manage students in an organization, categorizing them into different departments
like Engineering, Management, Arts, Medical each with specific qualifications and common attributes.
Use all the key concepts of OOPs which you have and implement using C++. Minimum 5 Concepts to be used.
Note: (i) Use as many concepts as possible
(ii) Mention the features/ concept used along with the coding

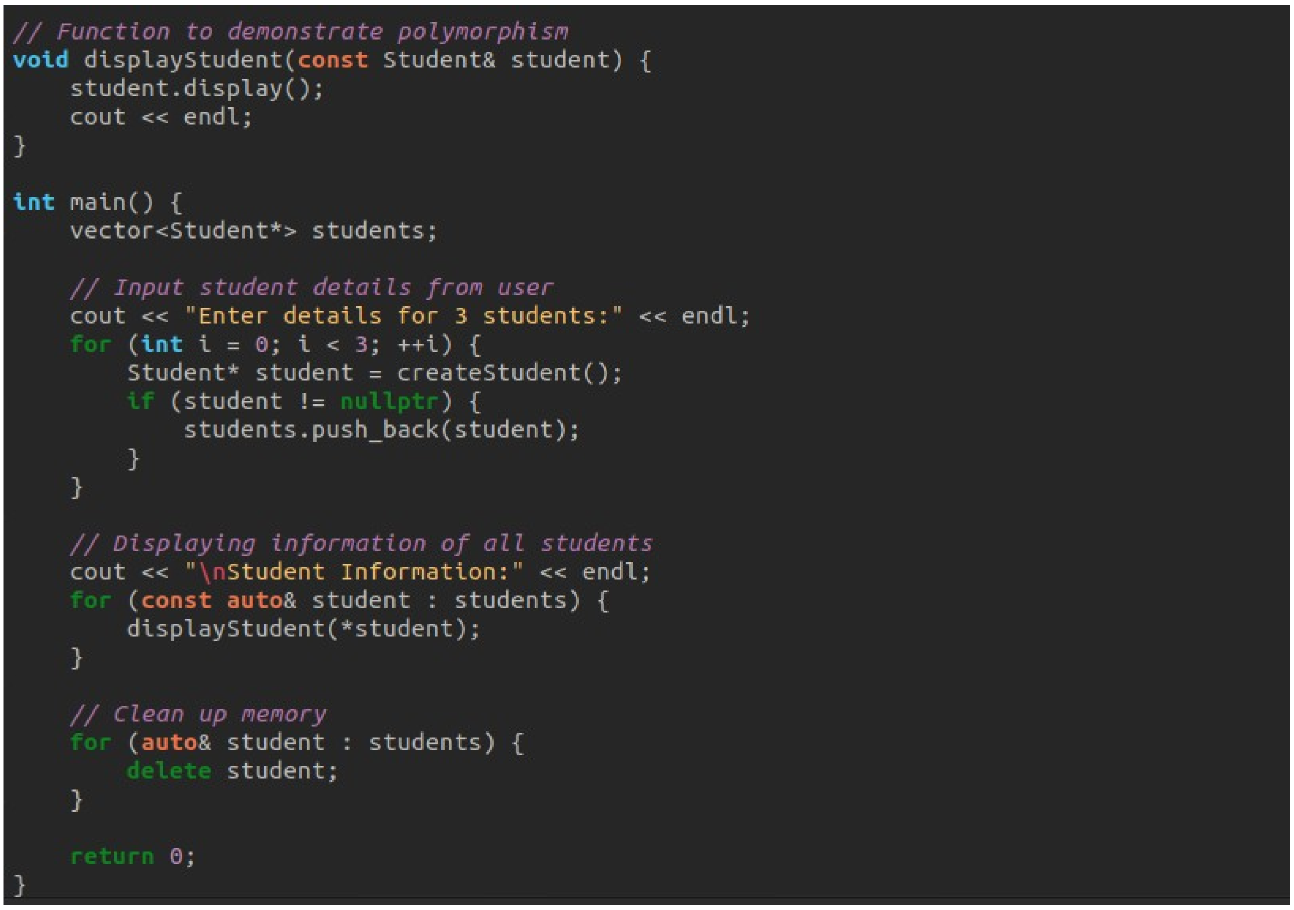
CODE



```cpp
#include <iostream>
#include <vector>
#include <string>

using namespace std;

// Base class for Student
class Student {
protected:
    string name;
    int age;

public:
    Student(string name, int age) : name(name), age(age) {}

    virtual void display() const {
        cout << "Name: " << name << ", Age: " << age;
    }
};

// Derived classes for different departments
class EngineeringStudent : public Student {
private:
    string specialization;

public:
    EngineeringStudent(string name, int age, string specialization) : Student(name, age), specialization(specialization) {}

    void display() const override {
        cout << "Engineering Student - ";
        Student::display();
        cout << ", Specialization: " << specialization;
    }
};

class ManagementStudent : public Student {
private:
    string major;

public:
    ManagementStudent(string name, int age, string major) : Student(name, age), major(major) {}

    void display() const override {
        cout << "Management Student - ";
        Student::display();
        cout << ", Major: " << major;
    }
};
```



```cpp
class ArtsStudent : public Student {
private:
    string areaOfStudy;

public:
    ArtsStudent(string name, int age, string areaOfStudy) : Student(name, age), areaOfStudy(areaOfStudy) {}

    void display() const override {
        cout << "Arts Student - ";
        Student::display();
        cout << ", Area of Study: " << areaOfStudy;
    }
};

// Function to create student objects based on user input
Student* createStudent() {
    string name, department;
    int age;
    cout << "Enter student name: ";
    getline(cin, name);
    cout << "Enter student age: ";
    cin >> age;
    cin.ignore(); // Clear input buffer
    cout << "Enter department (Engineering/Management/Arts): ";
    getline(cin, department);

    if (department == "Engineering") {
        string specialization;
        cout << "Enter specialization: ";
        getline(cin, specialization);
        return new EngineeringStudent(name, age, specialization);
    } else if (department == "Management") {
        string major;
        cout << "Enter major: ";
        getline(cin, major);
        return new ManagementStudent(name, age, major);
    } else if (department == "Arts") {
        string areaOfStudy;
        cout << "Enter area of study: ";
        getline(cin, areaOfStudy);
        return new ArtsStudent(name, age, areaOfStudy);
    } else {
        cout << "Invalid department!" << endl;
        return nullptr;
    }
}
```



```cpp
// Function to demonstrate polymorphism
void displayStudent(const Student& student) {
    student.display();
    cout << endl;
}

int main() {
    vector<Student*> students;

    // Input student details from user
    cout << "Enter details for 3 students:" << endl;
    for (int i = 0; i < 3; ++i) {
        Student* student = createStudent();
        if (student != nullptr) {
            students.push_back(student);
        }
    }

    // Displaying information of all students
    cout << "\nStudent Information:" << endl;
    for (const auto& student : students) {
        displayStudent(*student);
    }

    // Clean up memory
    for (auto& student : students) {
        delete student;
    }

    return 0;
}
```

In this implementation:

Classes: Various classes are defined such as Student, EngineeringStudent, ManagementStudent, and ArtsStudent.

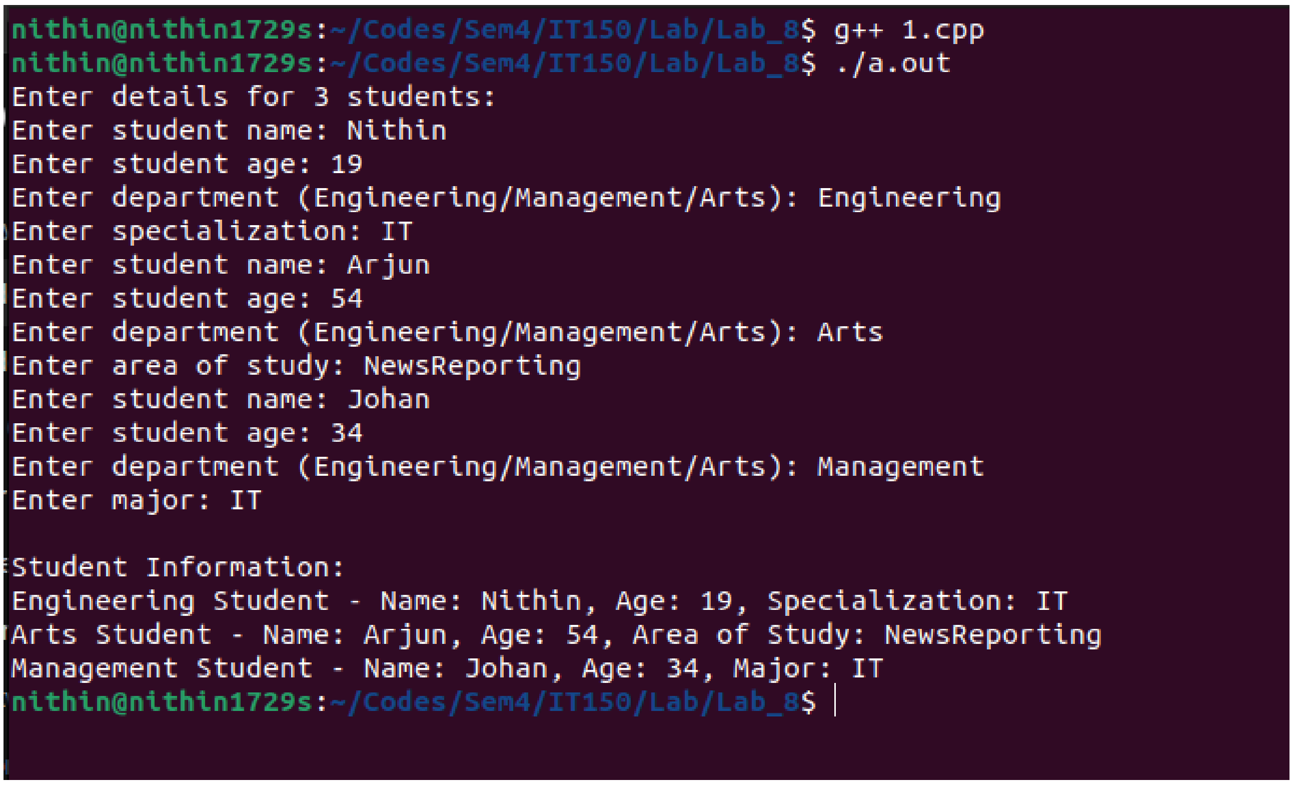
Inheritance: Derived classes EngineeringStudent, ManagementStudent, and ArtsStudent inherit from the base class Student.

Polymorphism: The display() function is marked as virtual in the base class Student and overridden in derived classes, allowing polymorphic behavior. Also, the displayStudent() function demonstrates polymorphism by accepting a reference to a Student object but can display information of any derived class.

Encapsulation: Data members of the Student class are encapsulated by making them protected, accessible to derived classes but not directly accessible from outside the class.

Abstraction: The implementation hides the internal details of each department's student class, providing a simple interface for displaying student information.

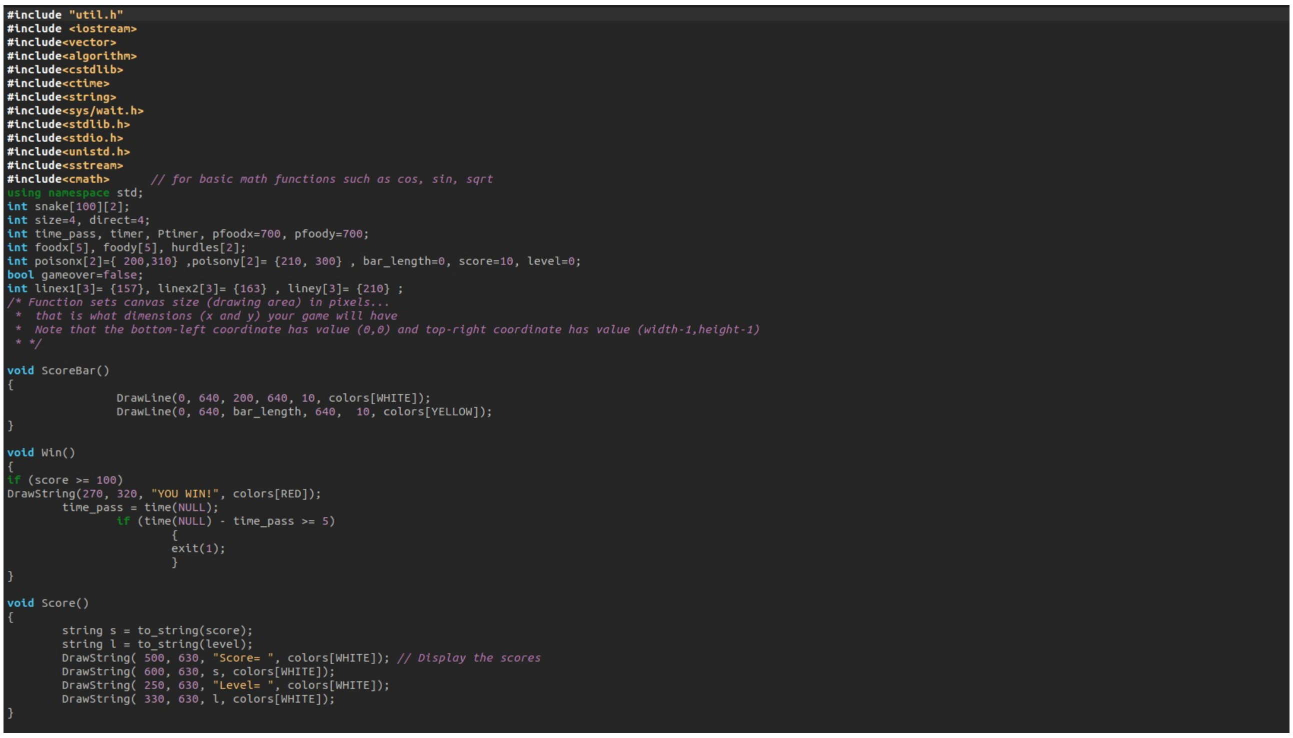
Output



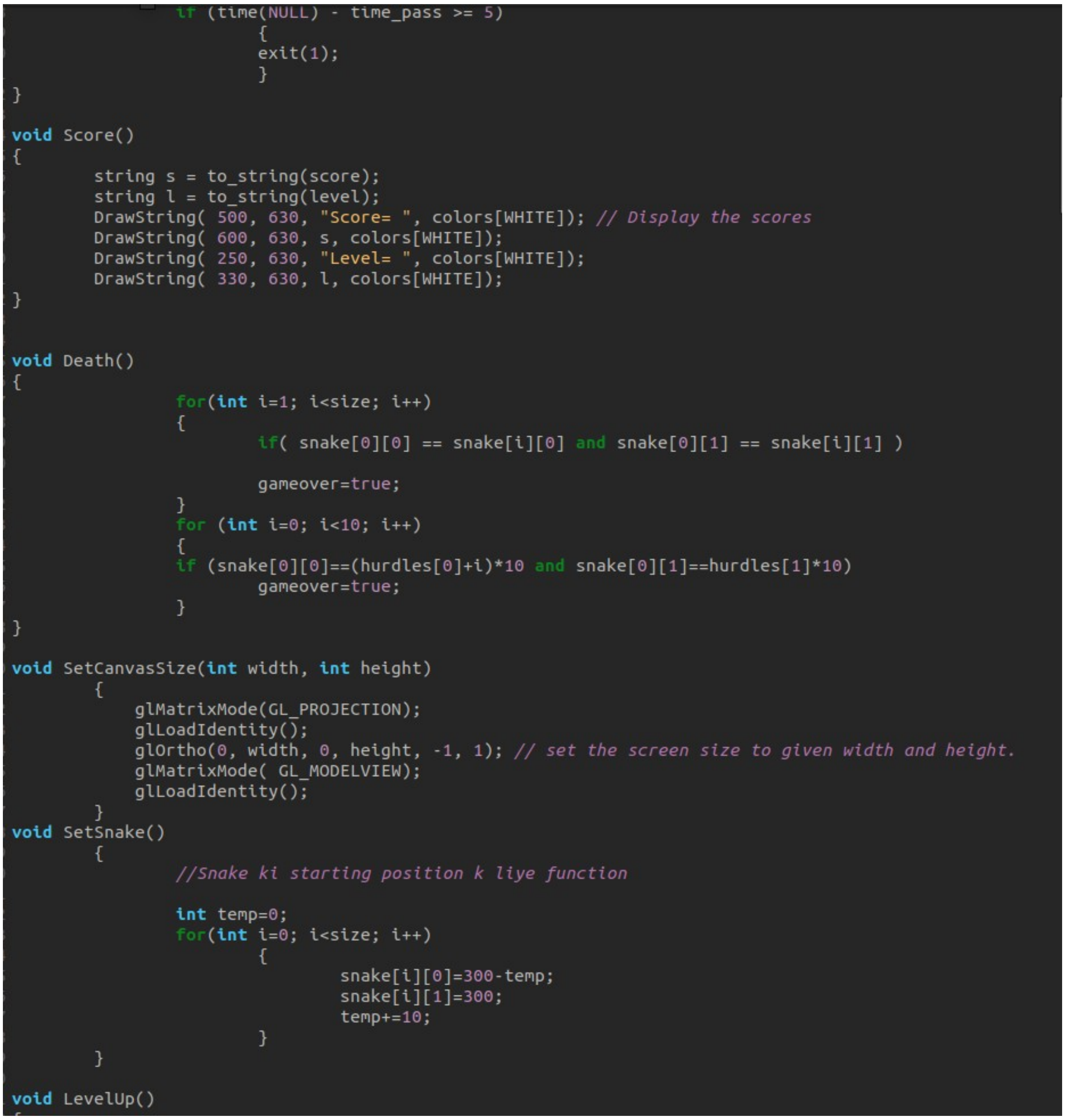
```
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_8$ g++ 1.cpp
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_8$ ./a.out
Enter details for 3 students:
Enter student name: Nithin
Enter student age: 19
Enter department (Engineering/Management/Arts): Engineering
Enter specialization: IT
Enter student name: Arjun
Enter student age: 54
Enter department (Engineering/Management/Arts): Arts
Enter area of study: NewsReporting
Enter student name: Johan
Enter student age: 34
Enter department (Engineering/Management/Arts): Management
Enter major: IT

Student Information:
Engineering Student - Name: Nithin, Age: 19, Specialization: IT
Arts Student - Name: Arjun, Age: 54, Area of Study: NewsReporting
Management Student - Name: Johan, Age: 34, Major: IT
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_8$
```

2. Write a C++ code to design a simple snake game with necessary functions and parameters. The following
to be considered during coding:
a. handle user input,
b. update the game logic,
c. display the game screen.
Follow the below Rules to Play Snake Game:
🌍 Don’t hit a wall and don’t bite your own tail.
🌍 Crashing into a wall or your tail will end the game immediately.
🌍 1 point will be added to the player’s score for eating the fruit (#).
🌍 The player’s total score is calculated based on the number of fruits the snake consumed.
🌍 The length of the snake will be increased after eating the fruits.
🌍 Use w, a, s, d to move the snake.

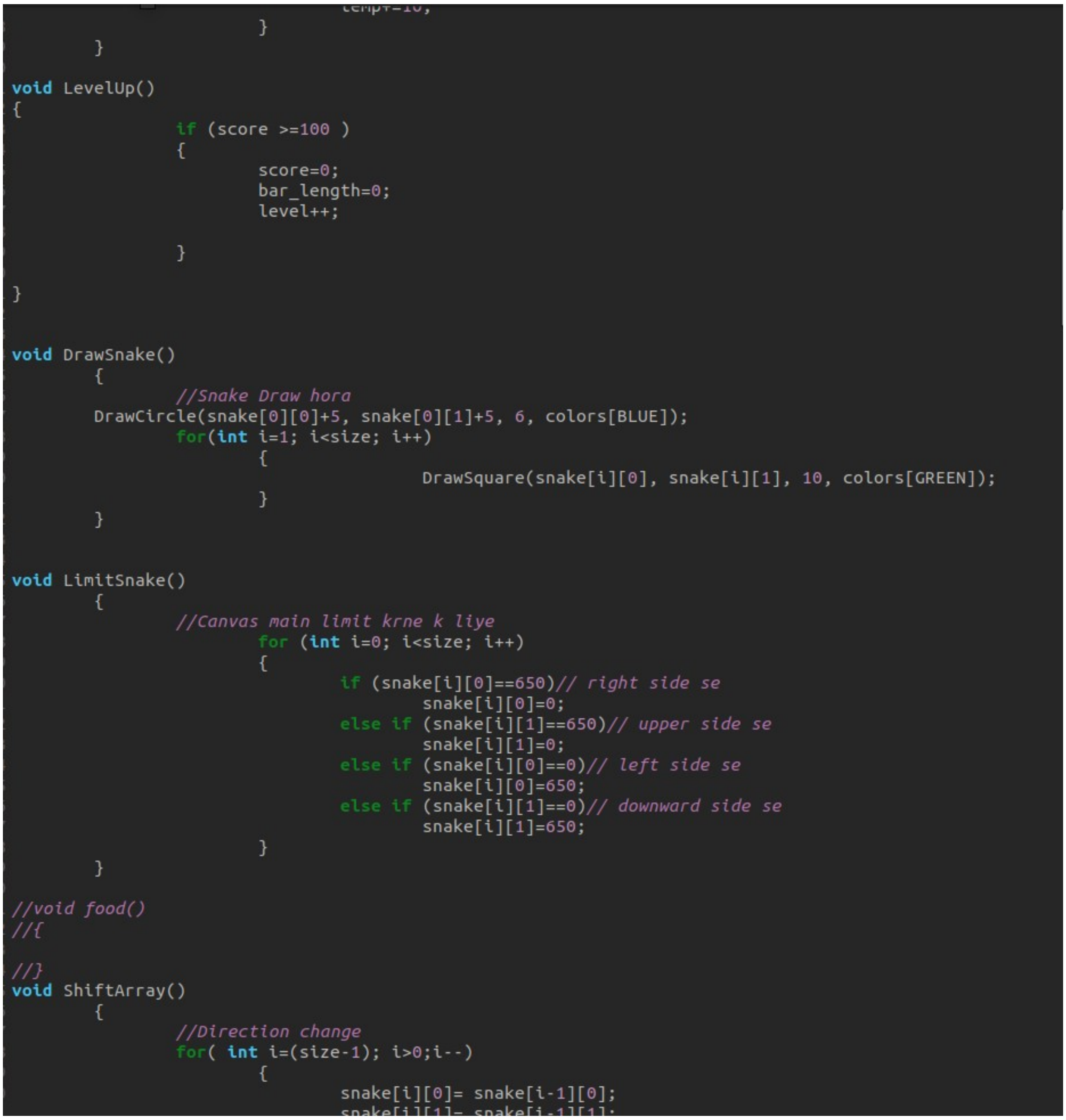
CODE



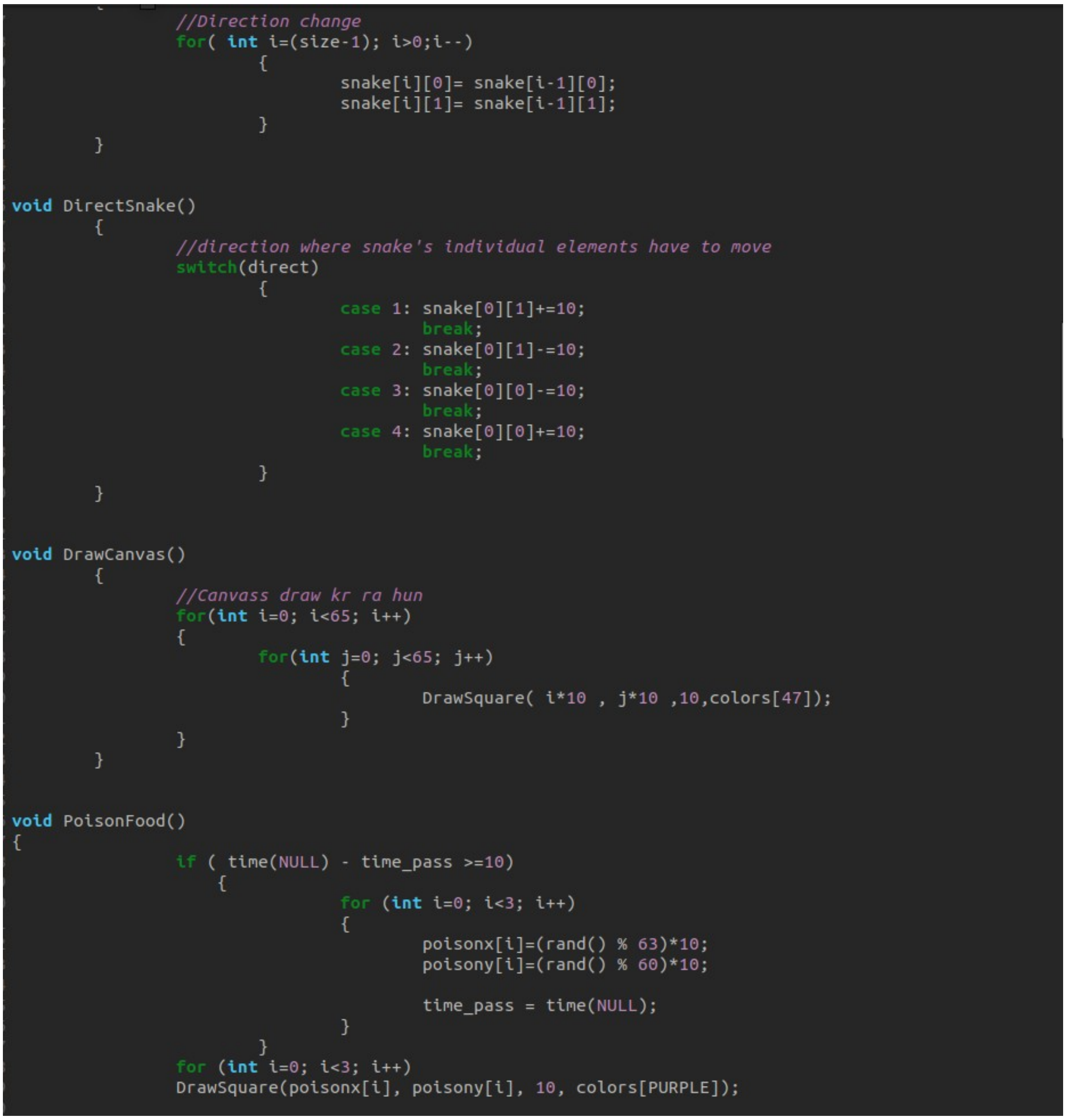
```cpp
#include "util.h"
#include <iostream>
#include<vector>
#include<algorithm>
#include<cstdlib>
#include<ctime>
#include<string>
#include<sys/wait.h>
#include<stdlib.h>
#include<stdio.h>
#include<unistd.h>
#include<sstream>
#include<cmath>      // for basic math functions such as cos, sin, sqrt
using namespace std;
int snake[100][2];
int size=4, direct=4;
int time_pass, timer, Ptimer, pfoodx=700, pfoody=700;
int foodx[5], foody[5], hurdles[2];
int poisonx[2]={ 200,310} ,poisony[2]= {210, 300} , bar_length=0, score=10, level=0;
bool gameover=false;
int linex1[3]= {157}, linex2[3]= {163} , liney[3]= {210} ;
/* Function sets canvas size (drawing area) in pixels...
 *  that is what dimensions (x and y) your game will have
 *  Note that the bottom-left coordinate has value (0,0) and top-right coordinate has value (width-1,height-1)
 * */

void ScoreBar()
{
                DrawLine(0, 640, 200, 640, 10, colors[WHITE]);
                DrawLine(0, 640, bar_length, 640,  10, colors[YELLOW]);
}

void Win()
{
if (score >= 100)
DrawString(270, 320, "YOU WIN!", colors[RED]);
        time_pass = time(NULL);
                if (time(NULL) - time_pass >= 5)
                        {
                        exit(1);
                        }
}

void Score()
{
        string s = to_string(score);
        string l = to_string(level);
        DrawString( 500, 630, "Score= ", colors[WHITE]); // Display the scores
        DrawString( 600, 630, s, colors[WHITE]);
        DrawString( 250, 630, "Level= ", colors[WHITE]);
        DrawString( 330, 630, l, colors[WHITE]);
}
```



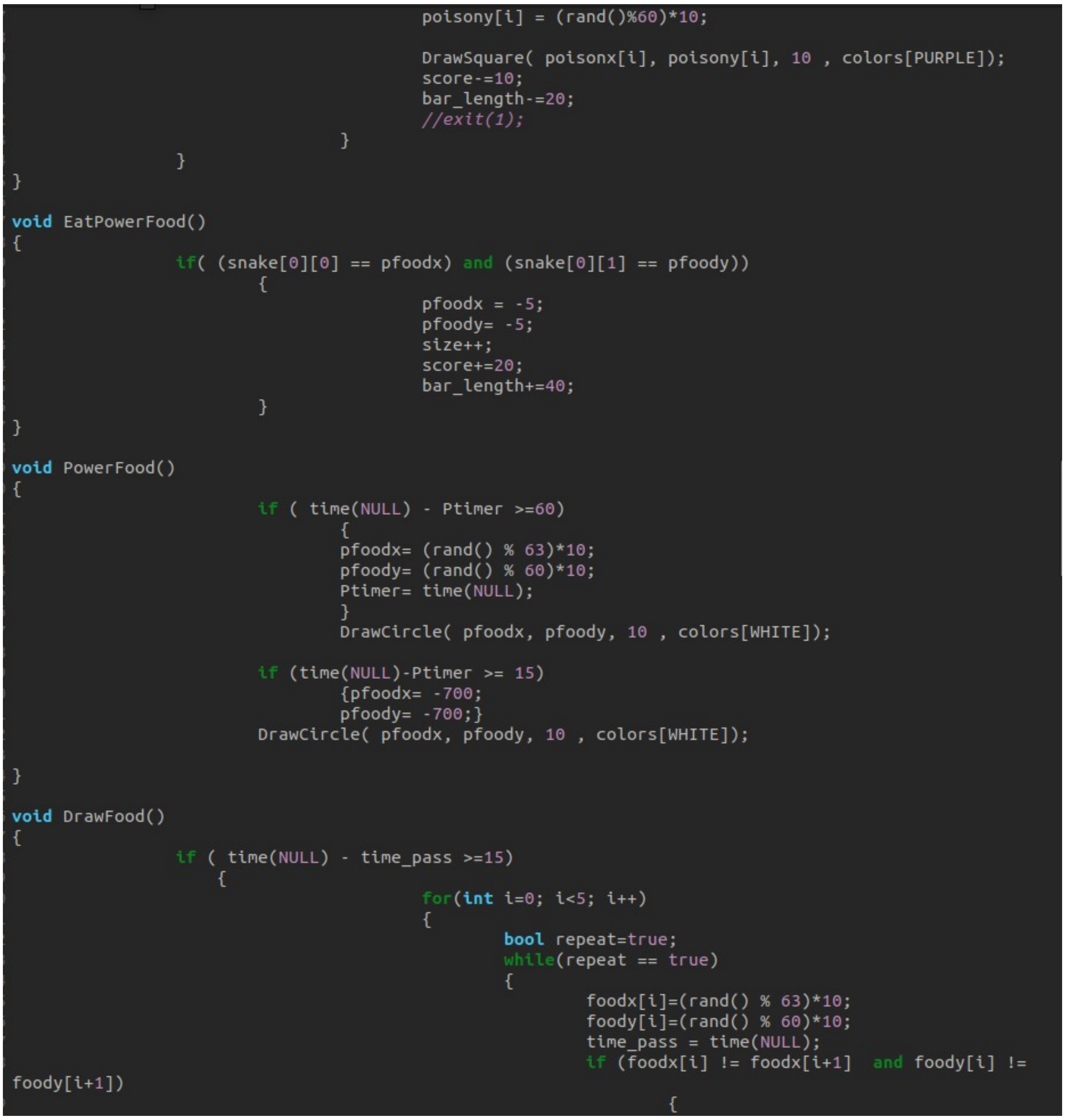
```
                if (time(NULL) - time_pass >= 5)
                        {
                        exit(1);
                        }
}

void Score()
{
        string s = to_string(score);
        string l = to_string(level);
        DrawString( 500, 630, "Score= ", colors[WHITE]); // Display the scores
        DrawString( 600, 630, s, colors[WHITE]);
        DrawString( 250, 630, "Level= ", colors[WHITE]);
        DrawString( 330, 630, l, colors[WHITE]);
}


void Death()
{
                for(int i=1; i<size; i++)
                {
                        if( snake[0][0] == snake[i][0] and snake[0][1] == snake[i][1] )

                        gameover=true;
                }
                for (int i=0; i<10; i++)
                {
                if (snake[0][0]==(hurdles[0]+i)*10 and snake[0][1]==hurdles[1]*10)
                        gameover=true;
                }
}

void SetCanvasSize(int width, int height)
        {
            glMatrixMode(GL_PROJECTION);
            glLoadIdentity();
            glOrtho(0, width, 0, height, -1, 1); // set the screen size to given width and height.
            glMatrixMode( GL_MODELVIEW);
            glLoadIdentity();
        }
void SetSnake()
        {
                //Snake ki starting position k liye function

                int temp=0;
                for(int i=0; i<size; i++)
                        {
                                snake[i][0]=300-temp;
                                snake[i][1]=300;
                                temp+=10;
                        }
        }

void LevelUp()
```



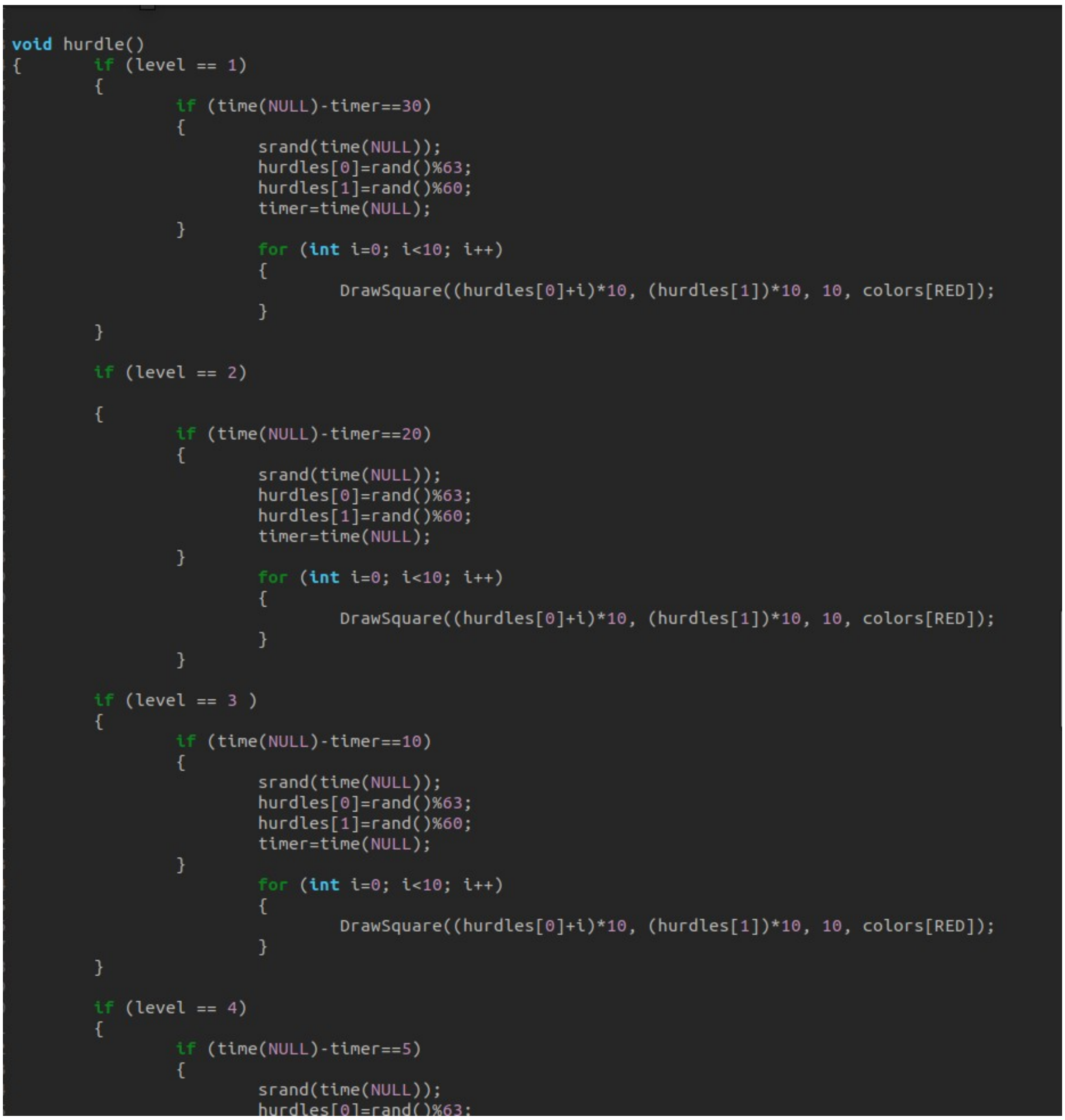
```
                                        temp+=10;
                                }
                }

void LevelUp()
{
                        if (score >=100 )
                        {
                                score=0;
                                bar_length=0;
                                level++;

                        }

}


void DrawSnake()
        {
                //Snake Draw hora
        DrawCircle(snake[0][0]+5, snake[0][1]+5, 6, colors[BLUE]);
                for(int i=1; i<size; i++)
                        {
                                        DrawSquare(snake[i][0], snake[i][1], 10, colors[GREEN]);
                        }
        }


void LimitSnake()
        {
                //Canvas main limit krne k liye
                        for (int i=0; i<size; i++)
                        {
                                if (snake[i][0]==650)// right side se
                                        snake[i][0]=0;
                                else if (snake[i][1]==650)// upper side se
                                        snake[i][1]=0;
                                else if (snake[i][0]==0)// left side se
                                        snake[i][0]=650;
                                else if (snake[i][1]==0)// downward side se
                                        snake[i][1]=650;
                        }
        }

//void food()
//{

//}
void ShiftArray()
        {
                //Direction change
                for( int i=(size-1); i>0;i--)
                        {
                                snake[i][0]= snake[i-1][0];
                                snake[i][1]= snake[i-1][1];
```



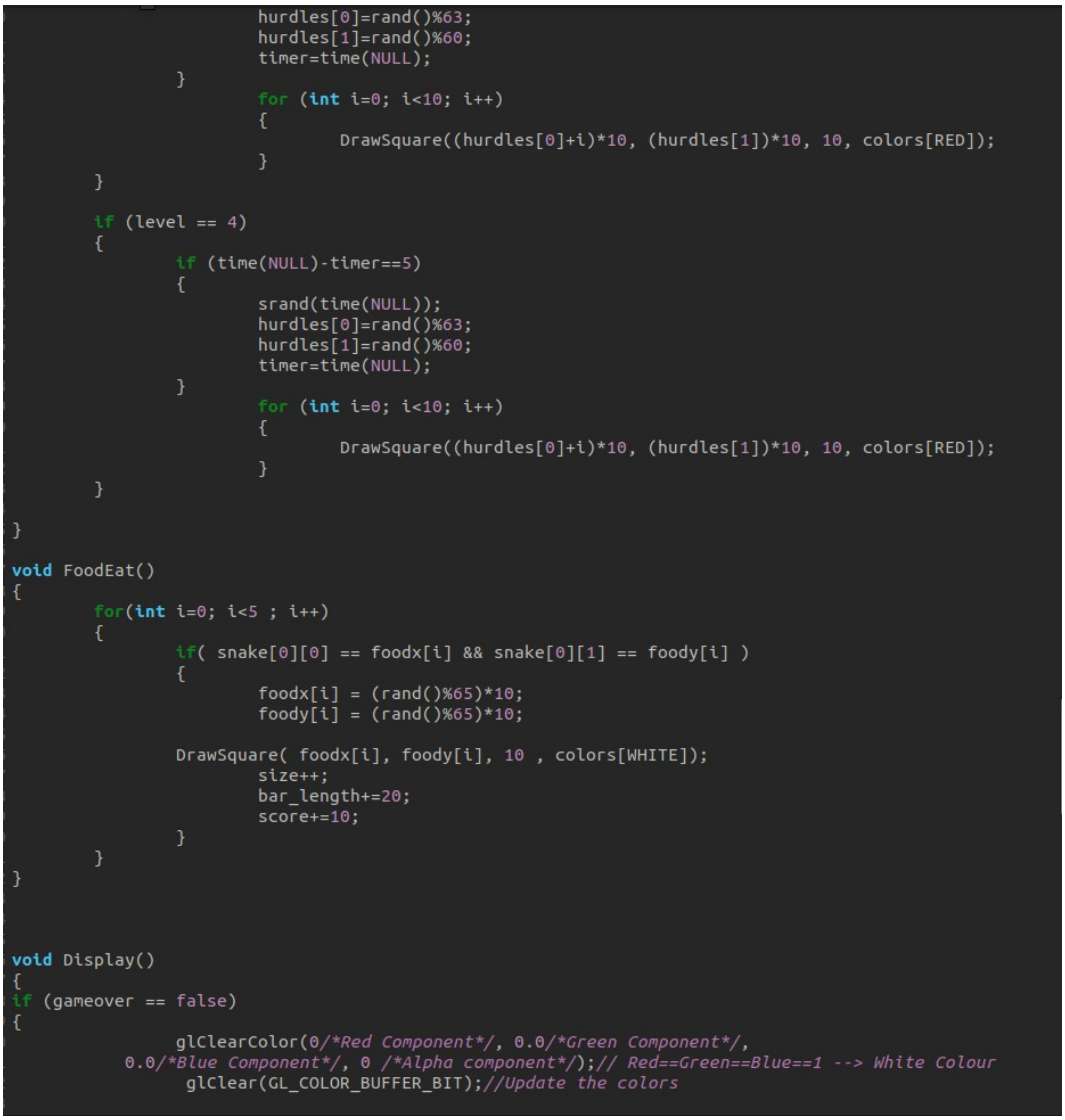
```
                //Direction change
                for( int i=(size-1); i>0;i--)
                        {
                                snake[i][0]= snake[i-1][0];
                                snake[i][1]= snake[i-1][1];
                        }
        }


void DirectSnake()
        {
                //direction where snake's individual elements have to move
                switch(direct)
                        {
                                case 1: snake[0][1]+=10;
                                        break;
                                case 2: snake[0][1]-=10;
                                        break;
                                case 3: snake[0][0]-=10;
                                        break;
                                case 4: snake[0][0]+=10;
                                        break;
                        }
        }


void DrawCanvas()
        {
                //Canvass draw kr ra hun
                for(int i=0; i<65; i++)
                {
                        for(int j=0; j<65; j++)
                                {
                                        DrawSquare( i*10 , j*10 ,10,colors[47]);
                                }
                }
        }


void PoisonFood()
{
                if ( time(NULL) - time_pass >=10)
                    {
                                for (int i=0; i<3; i++)
                                {
                                        poisonx[i]=(rand() % 63)*10;
                                        poisony[i]=(rand() % 60)*10;

                                        time_pass = time(NULL);
                                }
                        }
                for (int i=0; i<3; i++)
                DrawSquare(poisonx[i], poisony[i], 10, colors[PURPLE]);
```



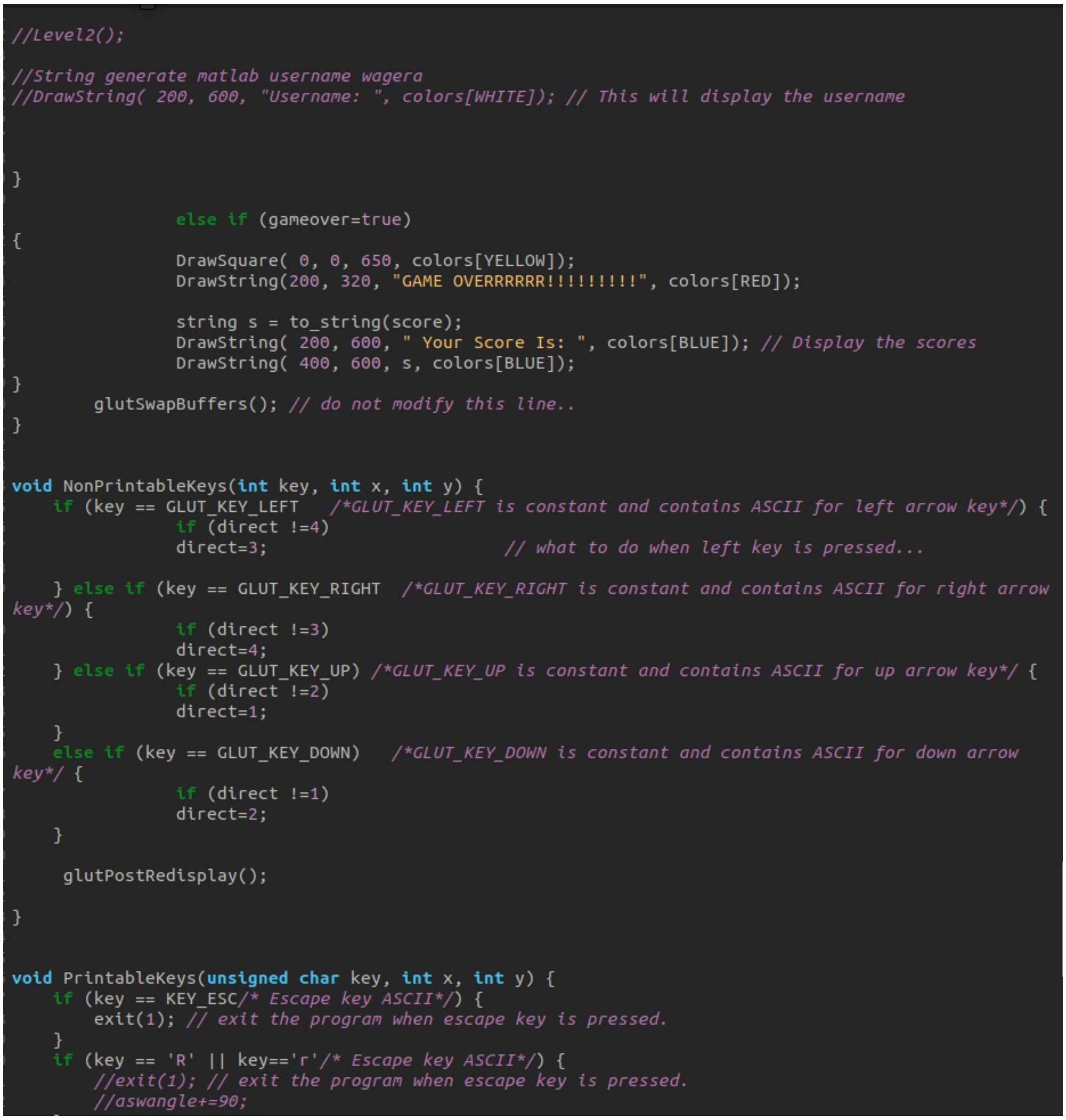
```
                                        poisony[i] = (rand()%60)*10;

                                        DrawSquare( poisonx[i], poisony[i], 10 , colors[PURPLE]);
                                        score-=10;
                                        bar_length-=20;
                                        //exit(1);
                                }
                }
}

void EatPowerFood()
{
                if( (snake[0][0] == pfoodx) and (snake[0][1] == pfoody))
                        {
                                        pfoodx = -5;
                                        pfoody= -5;
                                        size++;
                                        score+=20;
                                        bar_length+=40;
                        }
}

void PowerFood()
{
                        if ( time(NULL) - Ptimer >=60)
                                {
                                pfoodx= (rand() % 63)*10;
                                pfoody= (rand() % 60)*10;
                                Ptimer= time(NULL);
                                }
                                DrawCircle( pfoodx, pfoody, 10 , colors[WHITE]);

                        if (time(NULL)-Ptimer >= 15)
                                {pfoodx= -700;
                                pfoody= -700;}
                        DrawCircle( pfoodx, pfoody, 10 , colors[WHITE]);

}

void DrawFood()
{
                if ( time(NULL) - time_pass >=15)
                    {
                                        for(int i=0; i<5; i++)
                                        {
                                                bool repeat=true;
                                                while(repeat == true)
                                                {
                                                        foodx[i]=(rand() % 63)*10;
                                                        foody[i]=(rand() % 60)*10;
                                                        time_pass = time(NULL);
                                                        if (foodx[i] != foodx[i+1]  and foody[i] !=
foody[i+1])
                                                                {
```



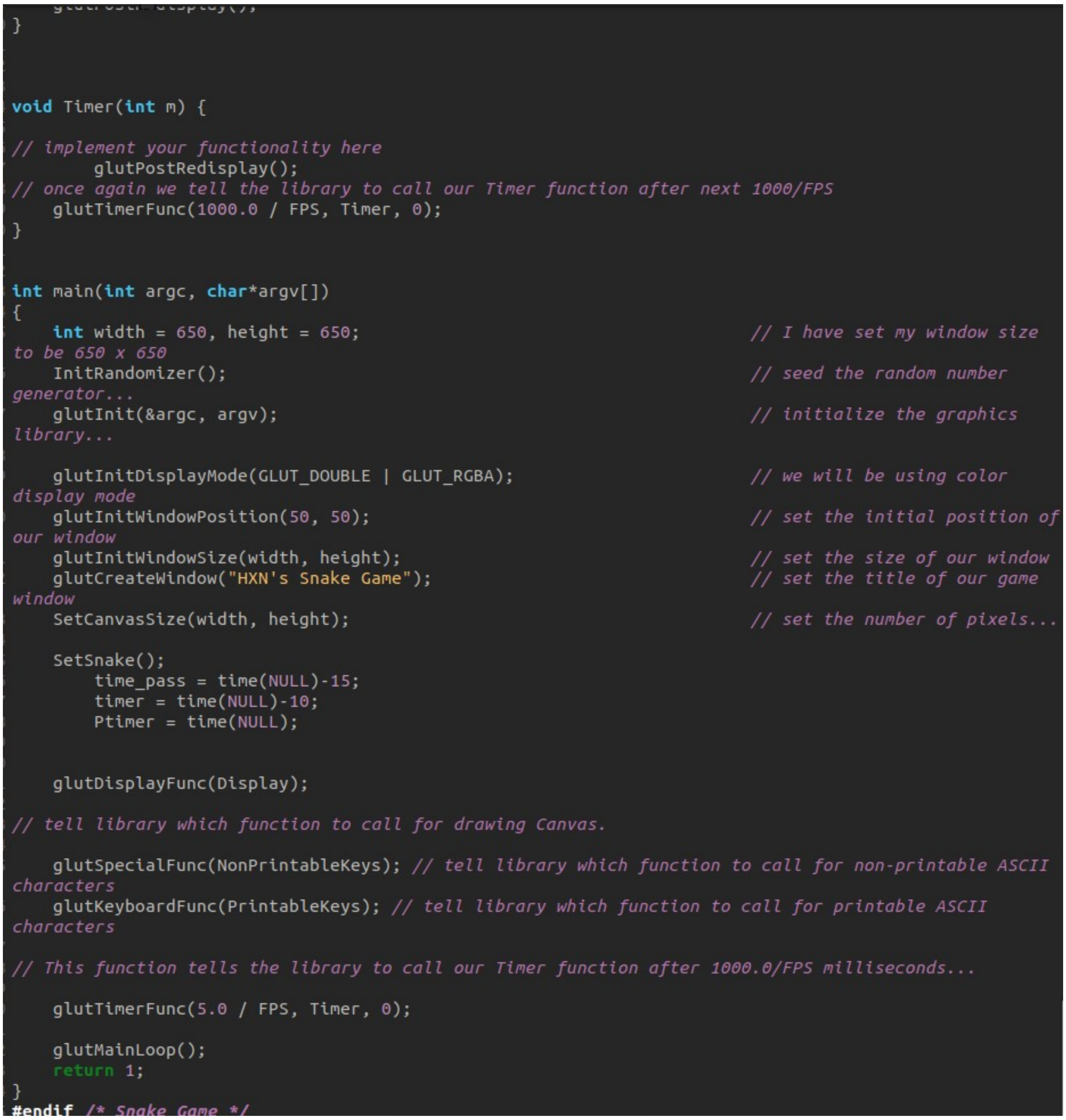
```c
void hurdle()
{       if (level == 1)
        {
                if (time(NULL)-timer==30)
                {
                        srand(time(NULL));
                        hurdles[0]=rand()%63;
                        hurdles[1]=rand()%60;
                        timer=time(NULL);
                }
                        for (int i=0; i<10; i++)
                        {
                                DrawSquare((hurdles[0]+i)*10, (hurdles[1])*10, 10, colors[RED]);
                        }
        }

        if (level == 2)

        {
                if (time(NULL)-timer==20)
                {
                        srand(time(NULL));
                        hurdles[0]=rand()%63;
                        hurdles[1]=rand()%60;
                        timer=time(NULL);
                }
                        for (int i=0; i<10; i++)
                        {
                                DrawSquare((hurdles[0]+i)*10, (hurdles[1])*10, 10, colors[RED]);
                        }
                }

        if (level == 3 )
        {
                if (time(NULL)-timer==10)
                {
                        srand(time(NULL));
                        hurdles[0]=rand()%63;
                        hurdles[1]=rand()%60;
                        timer=time(NULL);
                }
                        for (int i=0; i<10; i++)
                        {
                                DrawSquare((hurdles[0]+i)*10, (hurdles[1])*10, 10, colors[RED]);
                        }
        }

        if (level == 4)
        {
                if (time(NULL)-timer==5)
                {
                        srand(time(NULL));
                        hurdles[0]=rand()%63;
```



```
                        hurdles[0]=rand()%63;
                        hurdles[1]=rand()%60;
                        timer=time(NULL);
                }
                        for (int i=0; i<10; i++)
                        {
                                DrawSquare((hurdles[0]+i)*10, (hurdles[1])*10, 10, colors[RED]);
                        }
        }

        if (level == 4)
        {
                if (time(NULL)-timer==5)
                {
                        srand(time(NULL));
                        hurdles[0]=rand()%63;
                        hurdles[1]=rand()%60;
                        timer=time(NULL);
                }
                        for (int i=0; i<10; i++)
                        {
                                DrawSquare((hurdles[0]+i)*10, (hurdles[1])*10, 10, colors[RED]);
                        }
        }

}

void FoodEat()
{
        for(int i=0; i<5 ; i++)
        {
                if( snake[0][0] == foodx[i] && snake[0][1] == foody[i] )
                {
                        foodx[i] = (rand()%65)*10;
                        foody[i] = (rand()%65)*10;

                DrawSquare( foodx[i], foody[i], 10 , colors[WHITE]);
                        size++;
                        bar_length+=20;
                        score+=10;
                }
        }
}



void Display()
{
if (gameover == false)
{
                glClearColor(0/*Red Component*/, 0.0/*Green Component*/,
            0.0/*Blue Component*/, 0 /*Alpha component*/);// Red==Green==Blue==1 --> White Colour
                 glClear(GL_COLOR_BUFFER_BIT);//Update the colors
```

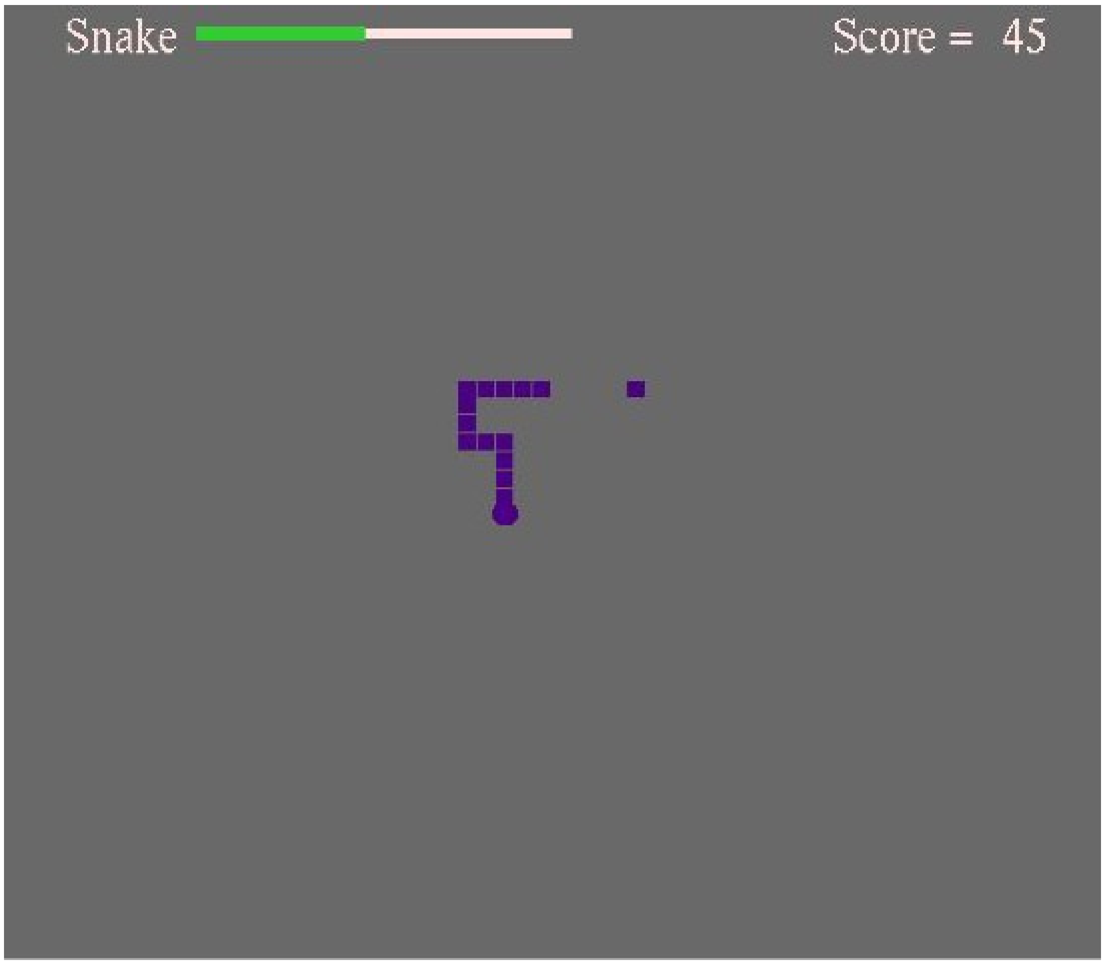


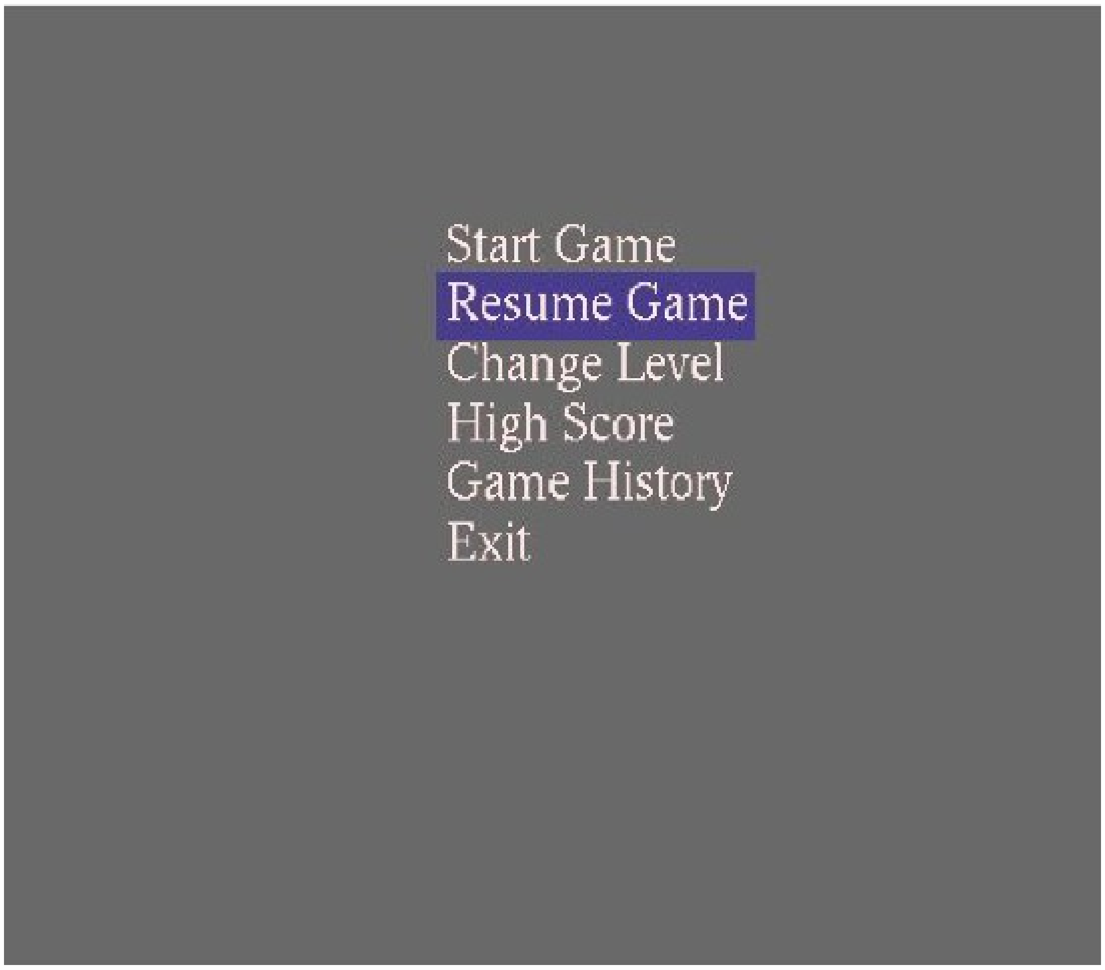
```
//Level2();

//String generate matlab username wagera
//DrawString( 200, 600, "Username: ", colors[WHITE]); // This will display the username



}
                else if (gameover=true)
{
                DrawSquare( 0, 0, 650, colors[YELLOW]);
                DrawString(200, 320, "GAME OVERRRRRR!!!!!!!!!", colors[RED]);

                string s = to_string(score);
                DrawString( 200, 600, " Your Score Is: ", colors[BLUE]); // Display the scores
                DrawString( 400, 600, s, colors[BLUE]);
}
        glutSwapBuffers(); // do not modify this line..
}


void NonPrintableKeys(int key, int x, int y) {
    if (key == GLUT_KEY_LEFT   /*GLUT_KEY_LEFT is constant and contains ASCII for left arrow key*/) {
                if (direct !=4)
                direct=3;                    // what to do when left key is pressed...

    } else if (key == GLUT_KEY_RIGHT  /*GLUT_KEY_RIGHT is constant and contains ASCII for right arrow
key*/) {
                if (direct !=3)
                direct=4;
    } else if (key == GLUT_KEY_UP) /*GLUT_KEY_UP is constant and contains ASCII for up arrow key*/ {
                if (direct !=2)
                direct=1;
    }
    else if (key == GLUT_KEY_DOWN)   /*GLUT_KEY_DOWN is constant and contains ASCII for down arrow
key*/ {
                if (direct !=1)
                direct=2;
    }

     glutPostRedisplay();

}


void PrintableKeys(unsigned char key, int x, int y) {
    if (key == KEY_ESC/* Escape key ASCII*/) {
        exit(1); // exit the program when escape key is pressed.
    }
    if (key == 'R' || key=='r'/* Escape key ASCII*/) {
        //exit(1); // exit the program when escape key is pressed.
        //aswangle+=90;
```



```
}



void Timer(int m) {

// implement your functionality here
        glutPostRedisplay();
// once again we tell the library to call our Timer function after next 1000/FPS
    glutTimerFunc(1000.0 / FPS, Timer, 0);
}


int main(int argc, char*argv[])
{
    int width = 650, height = 650;                                      // I have set my window size
to be 650 x 650
    InitRandomizer();                                                   // seed the random number
generator...
    glutInit(&argc, argv);                                              // initialize the graphics
library...

    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);                       // we will be using color
display mode
    glutInitWindowPosition(50, 50);                                     // set the initial position of
our window
    glutInitWindowSize(width, height);                                  // set the size of our window
    glutCreateWindow("HXN's Snake Game");                               // set the title of our game
window
    SetCanvasSize(width, height);                                       // set the number of pixels...

    SetSnake();
        time_pass = time(NULL)-15;
        timer = time(NULL)-10;
        Ptimer = time(NULL);


    glutDisplayFunc(Display);

// tell library which function to call for drawing Canvas.

    glutSpecialFunc(NonPrintableKeys); // tell library which function to call for non-printable ASCII
characters
    glutKeyboardFunc(PrintableKeys); // tell library which function to call for printable ASCII
characters

// This function tells the library to call our Timer function after 1000.0/FPS milliseconds...

    glutTimerFunc(5.0 / FPS, Timer, 0);

    glutMainLoop();
    return 1;
}
#endif /* Snake Game */
```

OUTPUT




Start Game
Resume Game
Change Level
High Score
Game History
Exit