



RIPHAH

INTERNATIONAL

UNIVERSITY

LAHORE

PROJECT:      SNAKE GAME

SUBJECT:

PROGRAMMING FUNDAMENTALS

SUBMITTED TO:

MS. KHOLA NASEEM

SUBMITTED BY:

ASAD AHMAD BAJWA

SAP ID:

28987

DEPARTMENT:

BSCS SPRING 2021

SEMESTER:

1<sup>ST</sup>

SECTION:

A1

DATE:

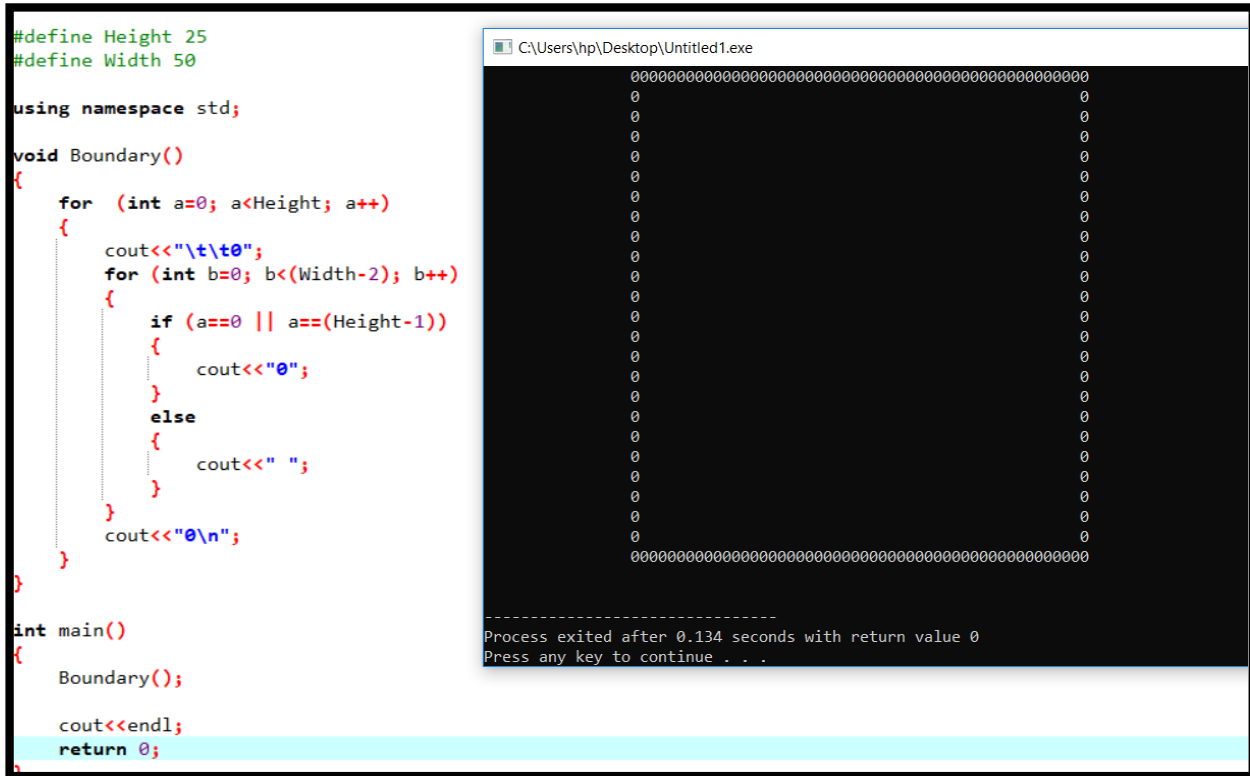
08 July, 2021

# DETAILED REPORT

## 1. Making Boundary:

Build a user-defined function and calling it in the main function.

Drawing symbols using Loops to define the boundary.



```
#define Height 25
#define Width 50

using namespace std;

void Boundary()
{
    for (int a=0; a<Height; a++)
    {
        cout<<"\t\t0";
        for (int b=0; b<(Width-2); b++)
        {
            if (a==0 || a==(Height-1))
            {
                cout<<"0";
            }
            else
            {
                cout<<" ";
            }
        }
        cout<<"0\n";
    }
}

int main()
{
    Boundary();

    cout<<endl;
    return 0;
}
```

The output window shows a 25x50 grid of characters. The top and bottom rows are filled with '0's. The first and last columns are also filled with '0's, forming a rectangular boundary. The interior of the grid is filled with spaces.

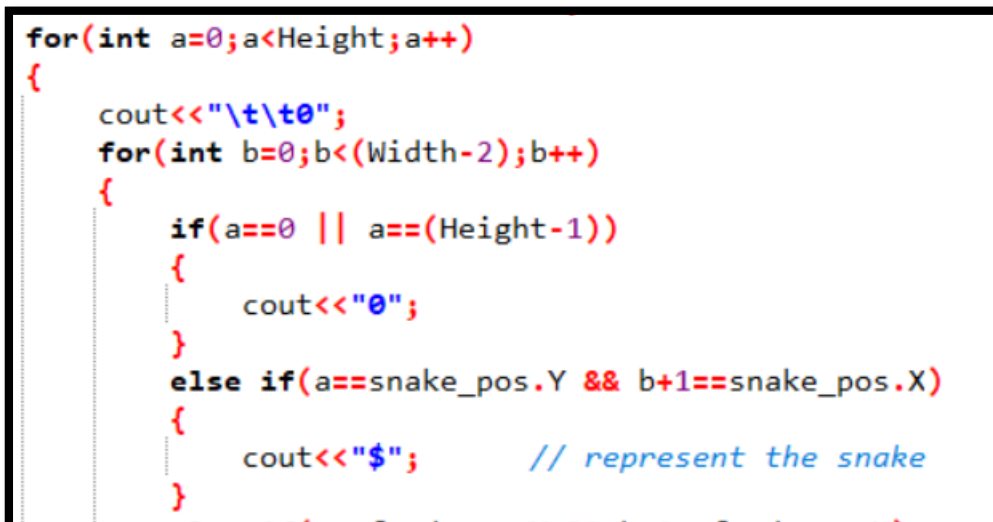
Process exited after 0.134 seconds with return value 0  
Press any key to continue . . .

## 2. Defining Snake on the Screen:

We use COORD function to define the position of snake on the screen.

We make Snake class in our project which contains and control all the function of our snake.

Point No.: 6 explain the Snake class.



```
for(int a=0;a<Height;a++)
{
    cout<<"\t\t0";
    for(int b=0;b<(Width-2);b++)
    {
        if(a==0 || a==(Height-1))
        {
            cout<<"0";
        }
        else if(a==snake_pos.Y && b+1==snake_pos.X)
        {
            cout<<"$";    // represent the snake
        }
    }
}
```

### 3. Repetition of the Board:

As we need to run our game and perform multiple operations again and again, so we use while loop in the main function and call the Boundary(); function again and again.

This repeats the program until our condition becomes false.

```
void Boundary()
{
    for (int a=0; a<Height; a++)
    {
        cout<<"\t\t0";
        for (int b=0; b<(Width-2); b++)
        {
            if (a==0 || a==(Height-1))
            {
                cout<<"0";
            }
            else if (a==y && b==x)
            {
                cout<<"$";
            }
            else
            {
                cout<<" ";
            }
        }
        cout<<"0\n";
    }
}

int main()
{
    while (true)
    {
        Boundary();
```

### 4. Clearing Random Functions in Repetition:

In console, output screen goes down and down with the repetitions and we cannot understand that what's going on. To solve this problem, C++ provide a Library function "cls". It used to clear the screen. We add a header file "cstdlib" for it.

As we run our program again and again, it clears the previous screen and write the output again on it.

```
        cout<<"$";
    }
    else
    {
        cout<<" ";
    }
}
cout<<"0\n";
}
}

int main()
{
    while (true)
    {
        Boundary();
        x++;
        system("cls");
    }

    cout<<endl;
    return 0;
}
```

## 5. To get a stable output screen:

As “cls” clear the screen and add output on it again but during this we saw that some kind of flickering happened on the screen.

So instead of using "cls", we use another function code provided by C++ by adding a header file "windows.h". By using the co-ordinate system of the C++, it over-write the outputs again and again. It does not clear the screen but it can over-write the next output on the previous output. To use it efficiently, we set the position every time at point {0,0}. Now we do not get any flickering on the screen instead we get a stable output screen. The code we use for it is:

```
SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), {0,0});
```

This Code is especially related with the console output. It sets the Cursor position on our desired position on the output screen. This code is only work for Windows. If you are using any other System-software, then you have to find a relative code for it.

```

for (int b=0; b<(Width-2); b++)
{
    if (a==0 || a==(Height-1))
    {
        cout<<"0";
    }
    else if (a==y && b==x)
    {
        cout<<"$";
    }
    else
    {
        cout<<" ";
    }
}
cout<<"0\n";
}

int main()
{
    while (true)
    {
        Boundary();
        x++;
        SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), {0,0});
    }
}

```

# CLASS

Class is an object-oriented programming, in which we define an object, its different behaviors or functions in our program. The class relate to a specific object and it controls all its behaviors within the program.

We can add this in our main program by simply calling it as:

```
#include "class_name.h"
```

## 6. Snake Class:

For defining the special tasks of our snake, we make a class which contains all its function. The benefit of this is that now we treat our snake as an element.

For defining our Snake position on the screen, we use “windows.h”. Now we can treat our snake with respect to the co-ordinates.

```
[*] main.cpp [*] Snake.h [*] Snake.cpp
1  #ifndef SNAKE_H
2  #define SNAKE_H
3
4  #include <windows.h>
5
6  class Snake
7  {
8      private:
9          COORD pos;
10         int len;
11         int vel;
12         char direction;
13
14     public:
15         Snake(COORD pos, int vel);
16         void change_dir(char dir);
17         void move_snake();
18     };
19
20 #endif
```

## 7. Directions of Snake:

Now we define that in which direction our snake moves.

After defining all the functions of our snake in “snake.h”, we write the code that defining its function in “snake.cpp”.

```
void Snake::change_dir(char dir)
{
    direction=dir;
}

void Snake::move_snake()
{
    switch(direction)
    {
        case "u" :
        {
            pos.Y -= vel;
            break;
        }
        case "d" :
        {
            pos.Y += vel;
            break;
        }
        case "l" :
        {
            pos.X -= vel;
            break;
        }
        case "r" :
        {
            pos.X += vel;
            break;
        }
    }
}
```

## 8. Adding Snake class in main program:

In main program, we first add snake class by #include "Snake.h".

As we want to represent our snake on the screen until the game over, we add it in the Boundary(); function. At start, we set the location of our snake at the center of the screen. Later when the program is running and we move our snake, so for capturing its current position with respect to co-ordinates we simply call its position function with COORD.

```
#include "Snake.h"           // to add 'Snake' class

#define Height 25            // one line functions called Macros
#define Width 50
using namespace std;

Snake snake ({Width/2, Height/2}, 1) ;

void Boundary()              //for making the boundary of the game
{
    COORD snake_pos=snake.get_pos();           //to capture/get the position of snake on screen

    for(int a=0;a<Height;a++)
    {
        cout<<"\t\t0";
        for(int b=0;b<(Width-2);b++)
        {
            if(a==0 || a==(Height-1))
            {
                cout<<"0";
            }
            else if(a==snake_pos.Y && b+1==snake_pos.X)
            {
                cout<<"$";           // represent the snake
            }
        }
    }
}
```

## 9. Controlling Movement of Snake:

As previous, we already define the directions of the snake. Now for moving the snake to a specific direction we use Arrow-keys. We use a switch statement and wrote the ASCII values of Arrow-keys in each case and define in which direction our snake moves when we press the relative key against it case.

C++ provide a library function "kbhit()", which is returns true if we press any key and for capturing that key value we use "getch()". We add a header file "conio.h" for using these functions. It relates with the output screen and control the input and ouput on the console output.

After taking the key, and giving the direction we call the move function of the snake to move it in that direction.

```

main.cpp Snake.h Snake.cpp
36 while (true)
37 {
38     board();
39     if(kbhit()) // true if any key hit by keyboard
40     {
41         switch (getch()) // to capture which key hit by the keyboard
42         {
43             case 72:
44             {
45                 snake.change_dir('u');
46                 break;
47             }
48             case 80:
49             {
50                 snake.change_dir('d');
51                 break;
52             }
53             case 75:
54             {
55                 snake.change_dir('l');
56                 break;
57             }
58             case 77:
59             {
60                 snake.change_dir('r');
61                 break;
62             }
63         }
64     }
65     snake.move_snake();
66 }

```

## 10. Food Class:

Now we add the food of the snake as a class also because we want to treat it as an object.

The food class contains all the functions related to food.

```

main.cpp Snake.h Snake.cpp Food.h Food.cpp
#ifndef FOOD_H
#define FOOD_H

#include <windows.h>
#include <cstdlib> //to generate random values

#define Height 25
#define Width 50

class Food
{
private:
    COORD pos;

public:
    void gen_food();
    COORD get_pos();
};

#endif

```

## 11. Adding Food on the Screen:

We add the food class in main function same as we add snake class.

After adding, we use a symbol which represent our food and we add it in the Boundary(); function.

```
for(int a=0;a<Height;a++)
{
    cout<<"\t\t0";
    for(int b=0;b<(Width-2);b++)
    {
        if(a==0 || a==(Height-1))
        {
            cout<<"0";
        }
        else if(a==snake_pos.Y && b+1==snake_pos.X)
        {
            cout<<"$";    // represent the snake
        }
        else if(a==food_pos.Y && b+1==food_pos.X)
        {
            cout<<"*";
        }
    }
}
```

## 12. Generating Food at Random Positions:

For generating our food at random positions, we use COORD system of C++ and the rand() library function.

Formula for generating random positions is:

$$= (\text{rand}() \% (\text{U}-\text{L}+1))+\text{L}$$

U= Upper limit

L= Lower limit

Instead of 1 we use 2 as Lower limit because it generates the food on the wall if lower limit is 1.

Similarly, we use (Width-1) as Upper limit because we don't want to generate the food on the wall.

```
void Food::gen_food()    // to generate food
{
    pos.X= (rand() % (Width-4))+2;    //to generate food at random positions
    pos.Y= (rand() % (Height-4))+2;
}
```



### 13. Eating, Generating Food and Growing Snake:

We write a code that whenever co-ordinate of snake is equals to the co-ordinate of food than we generate a new food and also we increase the length of the snake and also increment our score.

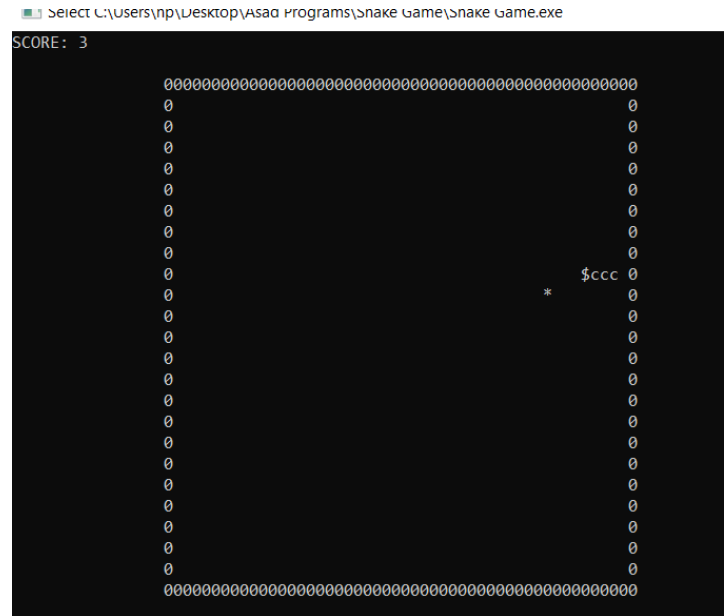
Code:

```
if(snake.eaten(food.get_pos()))
{
    food.gen_food();
    snake.grow();
    score++;
}
```

Increasing Snake Length:

```
void Snake::grow()
{
    length++;
}
```

Output:

A screenshot of a Windows command prompt window titled "C:\Users\hp\Desktop\Asad Programs\Snake Game\Snake Game.exe". The window shows a text-based snake game. At the top left, it says "SCORE: 3". The game board is a grid of 30x20 characters. The snake is represented by 'o' characters, forming a vertical line on the left side. The food is represented by '\$' and 'c' characters, located at approximately row 15, column 25. The snake's head is at the bottom of the vertical line, and it is positioned to eat the food. The text "Process exited after 163.3 seconds with return value 000000000000" is visible at the bottom of the window.

### 14. Collision of Snake with Wall:

Now we want that whenever the snake hit the wall, the game is over.

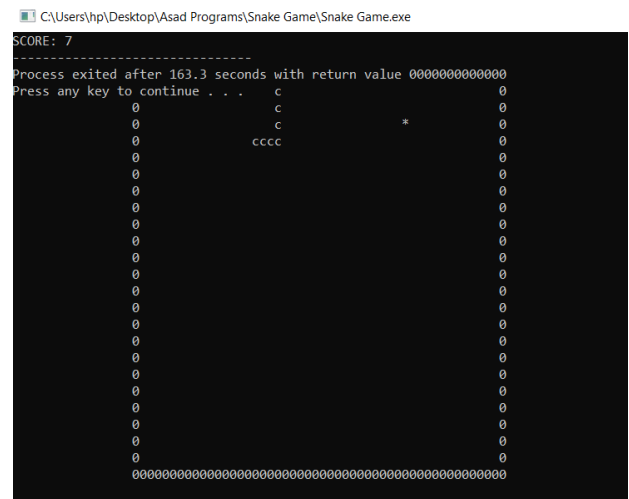
In main function, after moving the snake in specific direction we call the collision function of snake as:

```
if(snake.collide())
{
    game_over=true;
}
```

Whenever the snake collided, then game\_over is true and it exit the while loop.

We use Collision function in snake.cpp and we use bool function for collision because it returns true or false value and it is very efficient in that case.

```
bool Snake::collide()
{
    if(pos.X<1 || pos.X>Width-2 || pos.Y<1 || pos.Y>Height-2)
    {
        return true;
    }
    for(int i=0; i<length-1; i++)
    {
        if(pos.X==body[i].X && pos.Y==body[i].Y)
        {
            return true;
        }
    }
    return false;
}
```

A screenshot of a Windows command prompt window titled "C:\Users\hp\Desktop\Asad Programs\Snake Game\Snake Game.exe". The window shows a text-based snake game. At the top left, it says "SCORE: 7". The game board is a grid of 30x20 characters. The snake is represented by 'o' characters, forming a vertical line on the left side. The food is represented by '\$' and 'c' characters, located at approximately row 15, column 25. The snake's head is at the bottom of the vertical line, and it is positioned to eat the food. The text "Process exited after 163.3 seconds with return value 000000000000" is visible at the bottom of the window.

## 15. Snake Collision with itself:

Now we want that if the snake collides with itself, it returns true for `game_over` and end the game.

For this, we have to make the body of the snake a vector because two vectors never cross each other. So whenever it hit its own body it returns true value for game\_over.

SNAKE.CPP	FOOD.H	FOOD.CPP
<pre>         cout&lt;&lt;"*";     } else {     bool Body_collision = false;     for(int k=0; k&lt;snake_body.size()-1; k++)     {         if (a==snake_body[k].Y &amp;&amp; b+1== snake_body[k].X)         {             cout&lt;&lt;"c";             Body_collision=true;             break;         }     }     if (!Body_collision)     {         cout&lt;&lt;" ";     } } cout&lt;&lt;"0"&lt;&lt;endl;</pre>		

C:\Users\hp\Desktop\Asad Programs\SNAKE GAME\SNAKE GAME.EXE

SCORE: 7

---

Process exited after 255.9 seconds with return value 000000000000  
Press any key to continue . . .

## 16. Hit the Wall and Appearing from Opposite Wall:

Now we want that whenever the snake hit the wall it appears from its opposite wall.

So we remove the wall collision function but not remove the itself collision function.

We use position function of snake for this purpose.

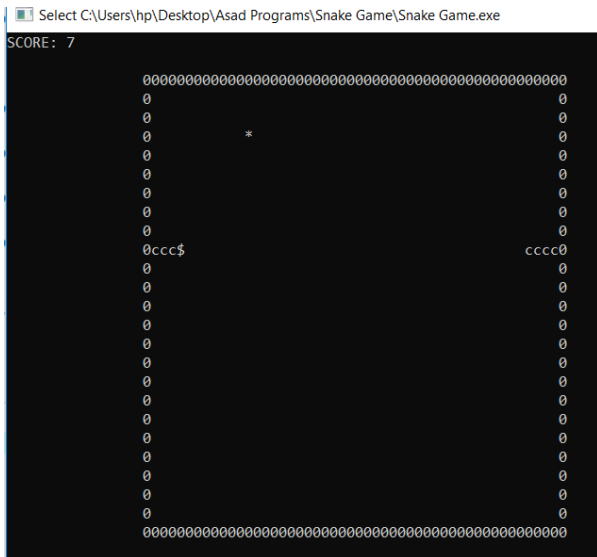
We write our code in the snake.cpp within its snake.move(); function because whenever snake moves then it change its direction and may be it strikes the wall at that instant so we its very effective to define it within the move function of the snake.

```

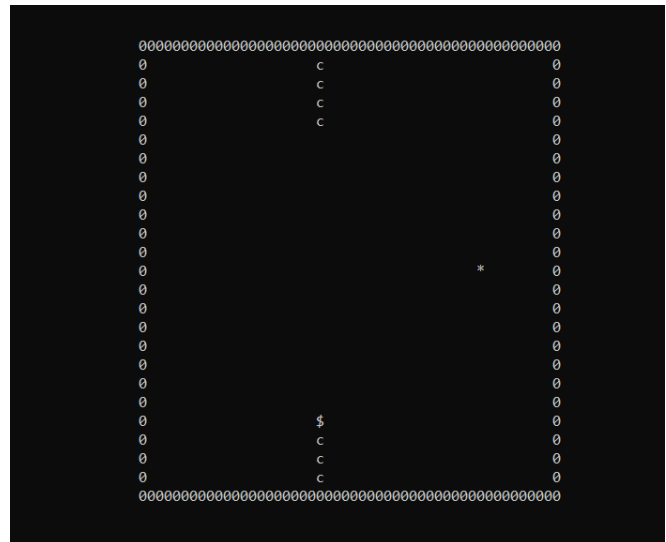
    }
    if(pos.X>Width-2)    // if snake strike the right side it appear from left side
    {
        pos.X=1;
    }
    if(pos.X<1)          // if snake strike left side it appear from left side
    {
        pos.X=Width-2;
    }
    if(pos.Y>Height-2)    // if snake strike the upper side it appear from down side
    {
        pos.Y=1;
    }
    if(pos.Y<1)           // if snake strike down side it appear from upper side
    {
        pos.Y=Height-2;
    }
}

```

Left and Right side:



Up and Down side:



## 17. Storing, Displaying and Updating High Score:

For this purpose, we use file handling in C++.

We include "fstream" header file to handle the file.

We can open the file in three modes:

1. Input (ios::in)      2. Output (ios::out)      3. Append (ios::app)

ifstream is used to read data from the file.

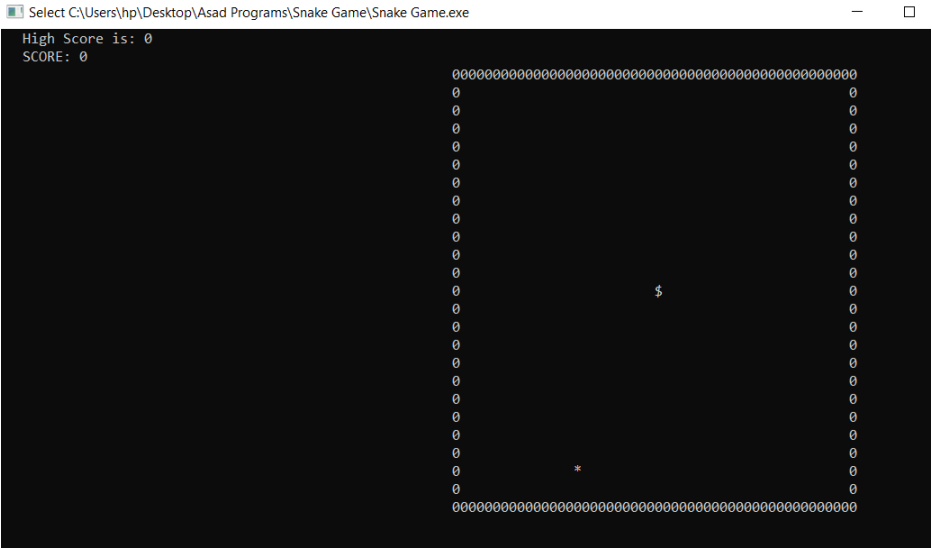
ofstream is used to write data in the file and overwrite the previous data.

Append mode is used for adding data in the file but not removing the previous data.

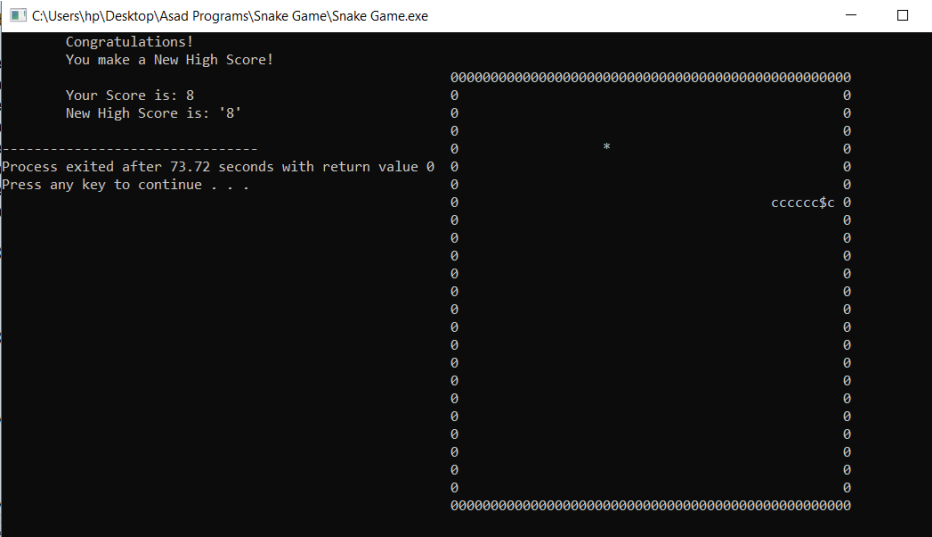
```
int High=0;
ifstream file("SnakeGame_HighScore.txt");
file>>High;
file.close();

ofstream HighScore("SnakeGame_HighScore.txt");
if (score>High)
{
    HighScore<<score;
}
else
{
    HighScore<<High;
}
HighScore.close();
if (score<High)
{
    cout<<"\tYour Score: "<<score;
    cout<<"\n\tGame Over!";
}
else
{
    cout<<"\tCongratulations!\n\tYou make a New High Score!";
    cout<<"\n\n\tYour Score: "<<score;
    cout<<"\n\tNew High Score is: '"<<score<<"'\n";
}
```

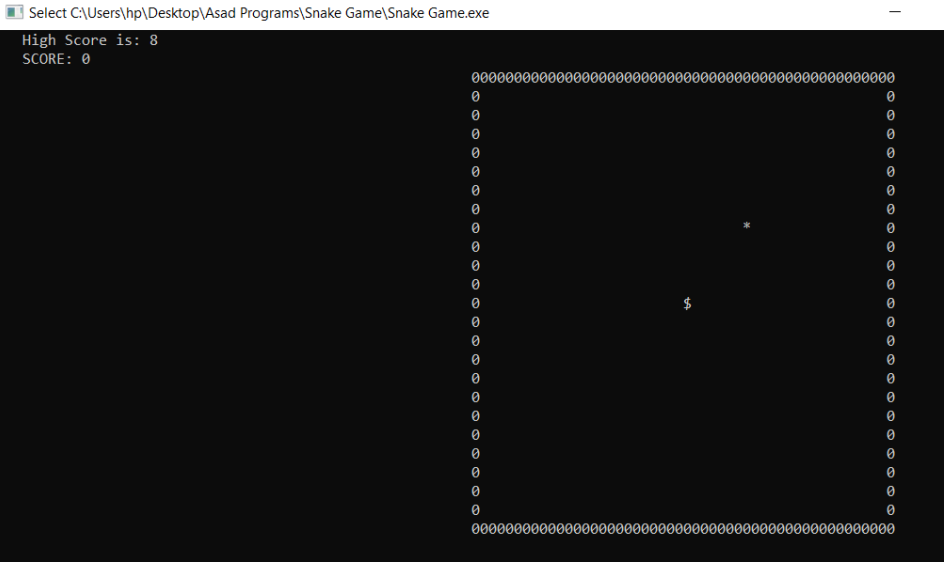
At Initial:



After Achieving a  
New High Score:

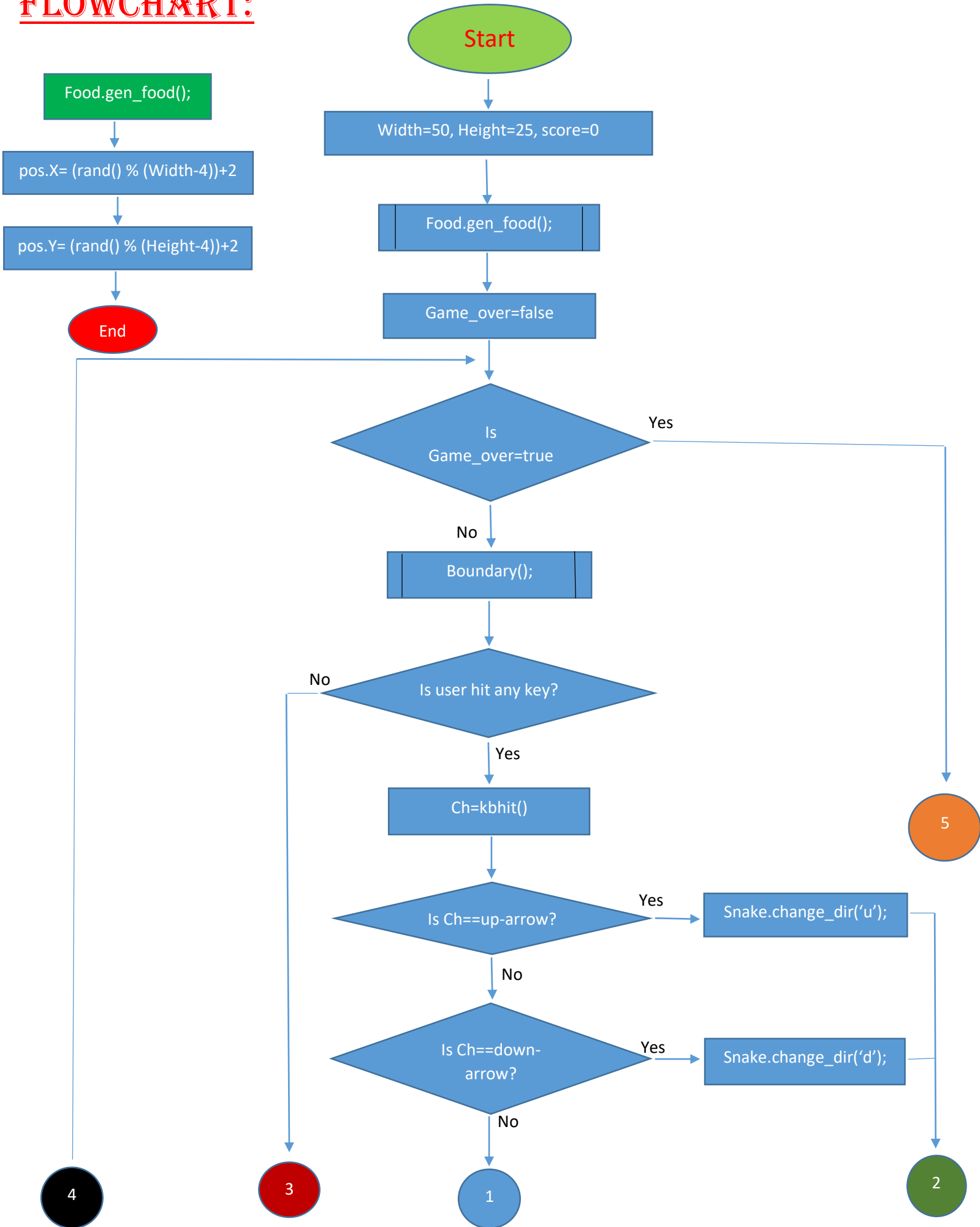


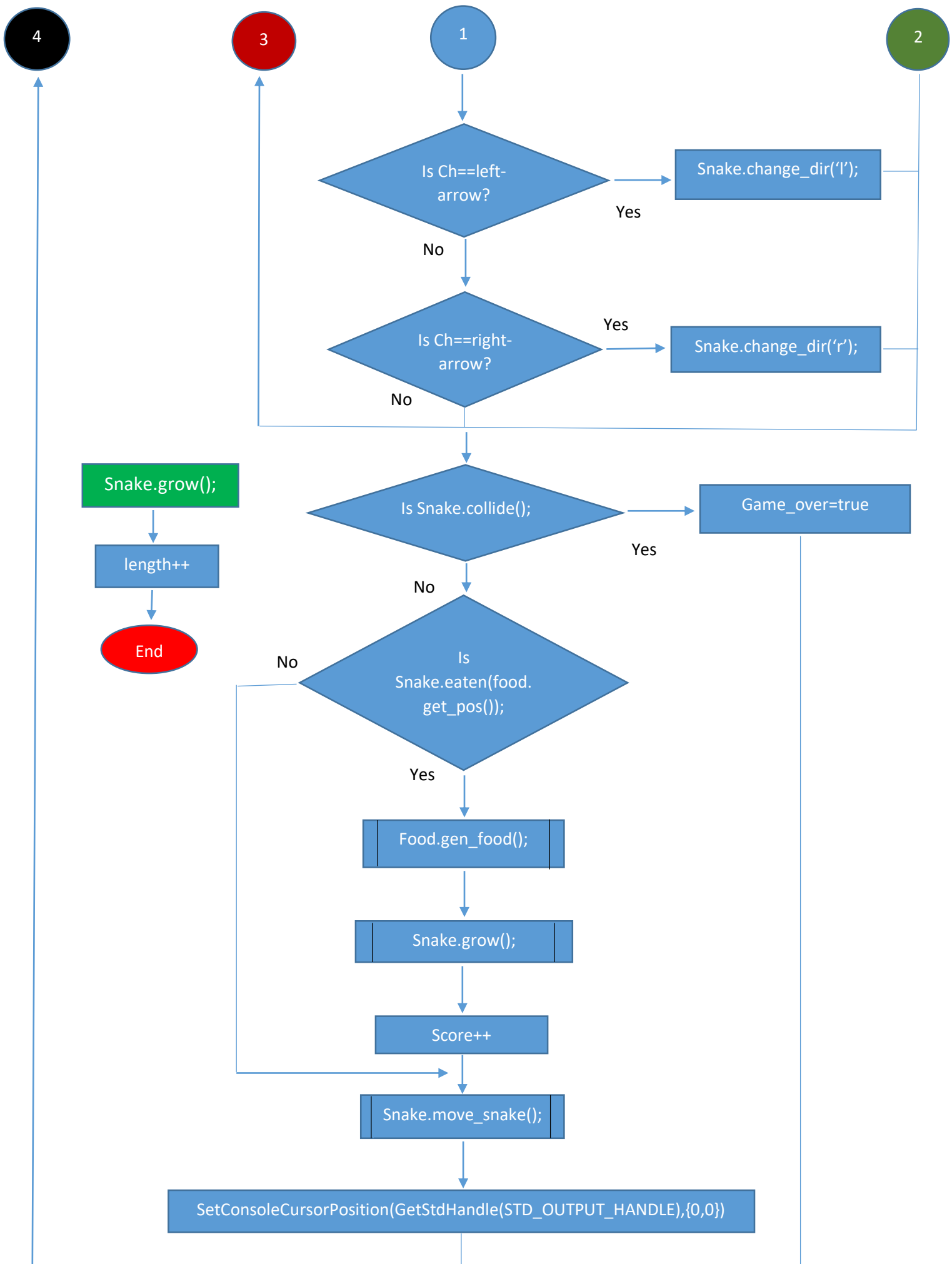
Starting a  
New Game:

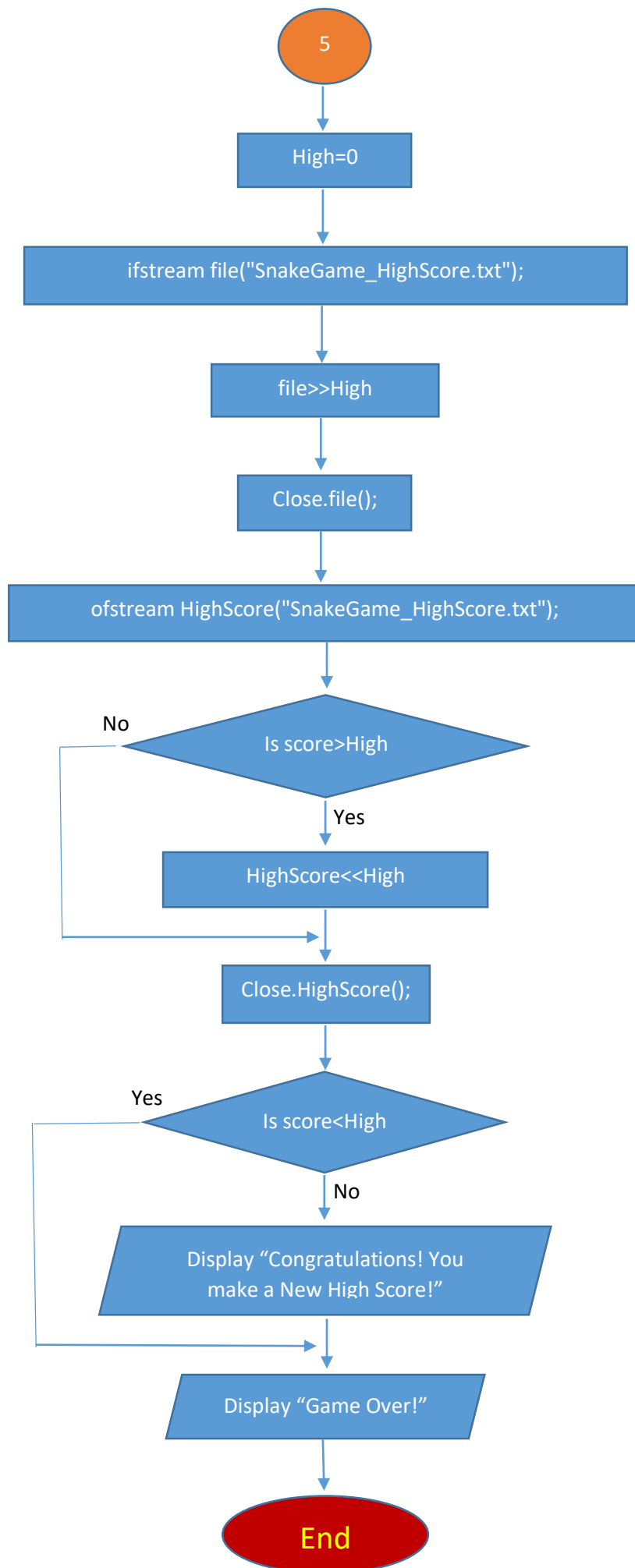


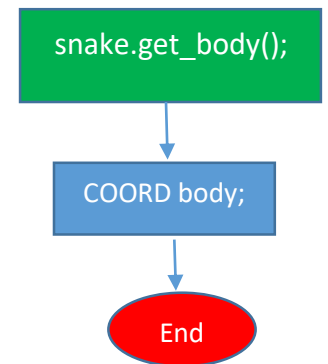
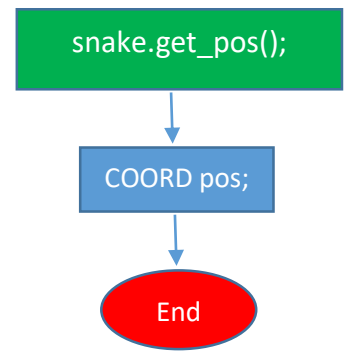
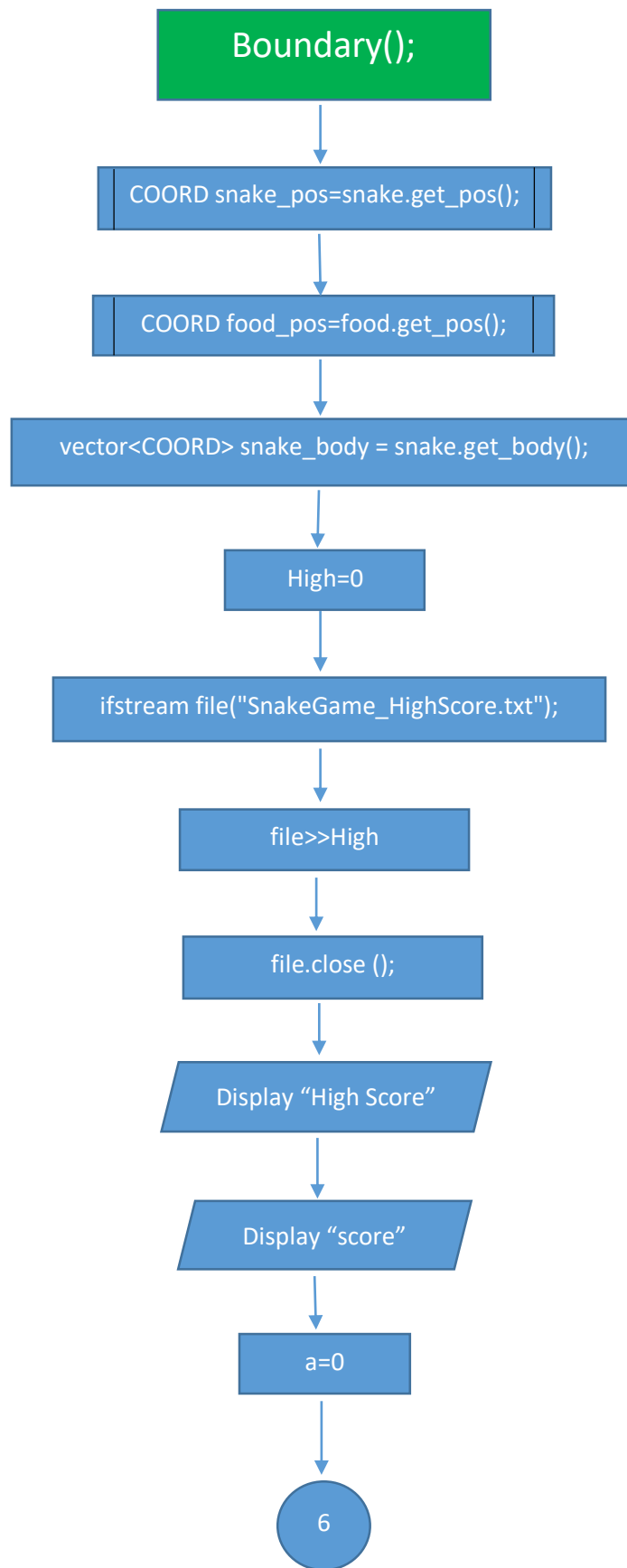
{COMPLETED}

FLOWCHART:

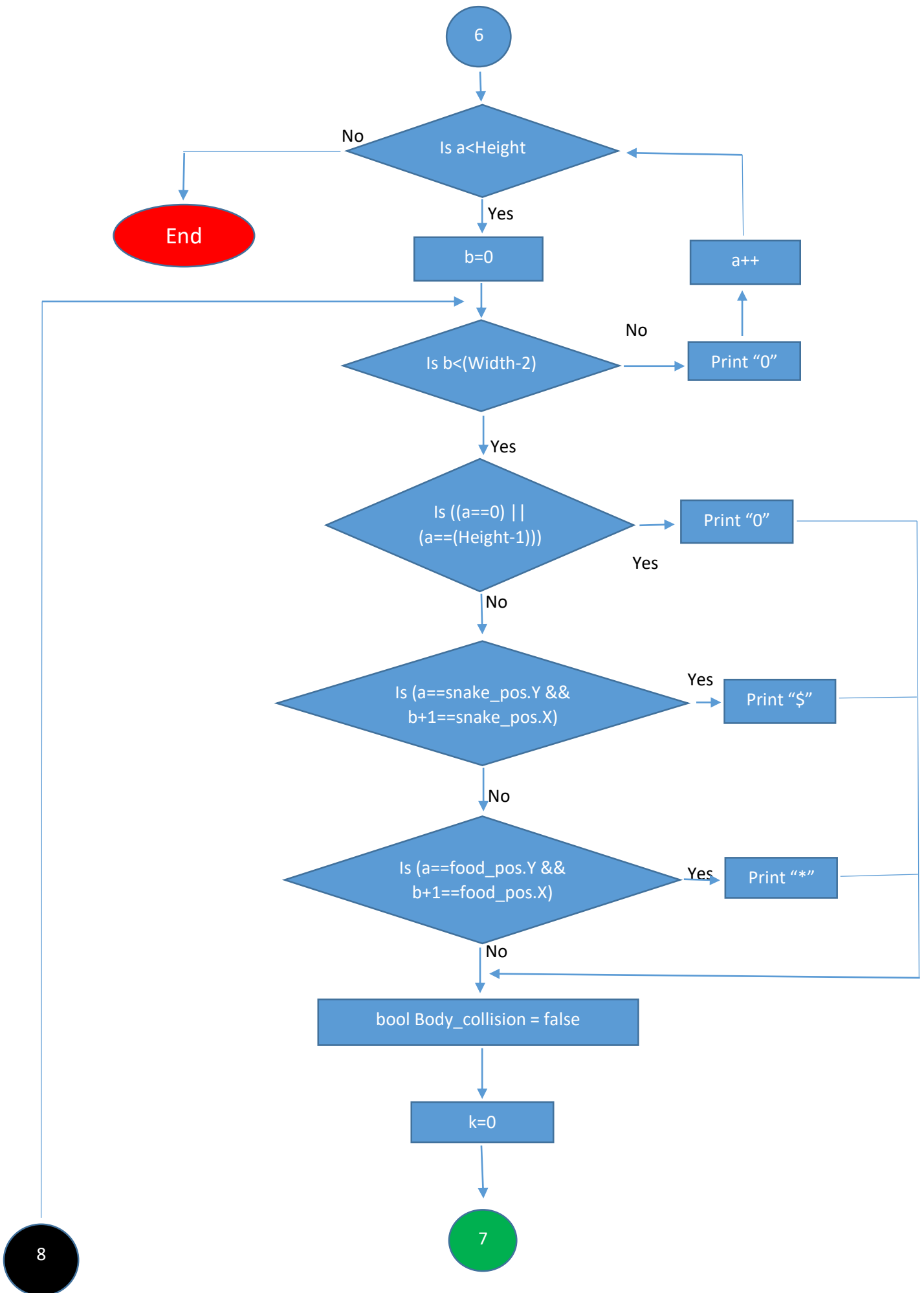






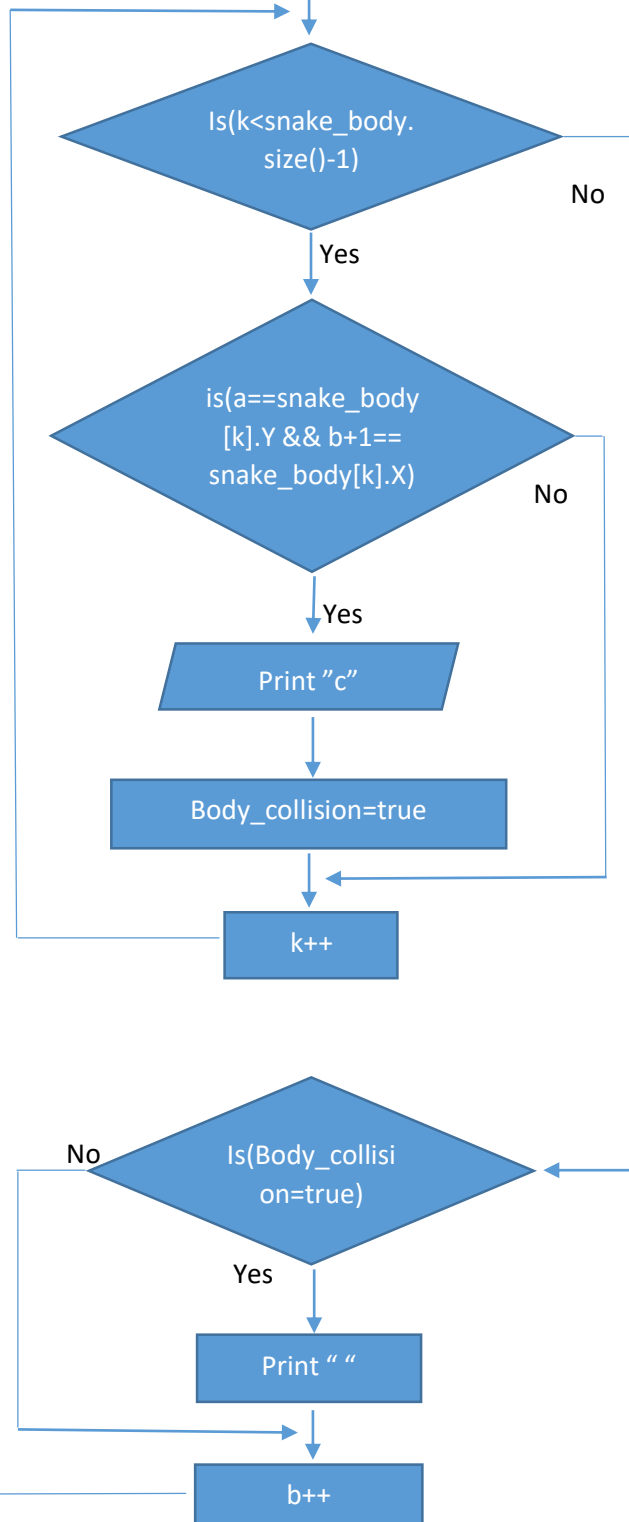


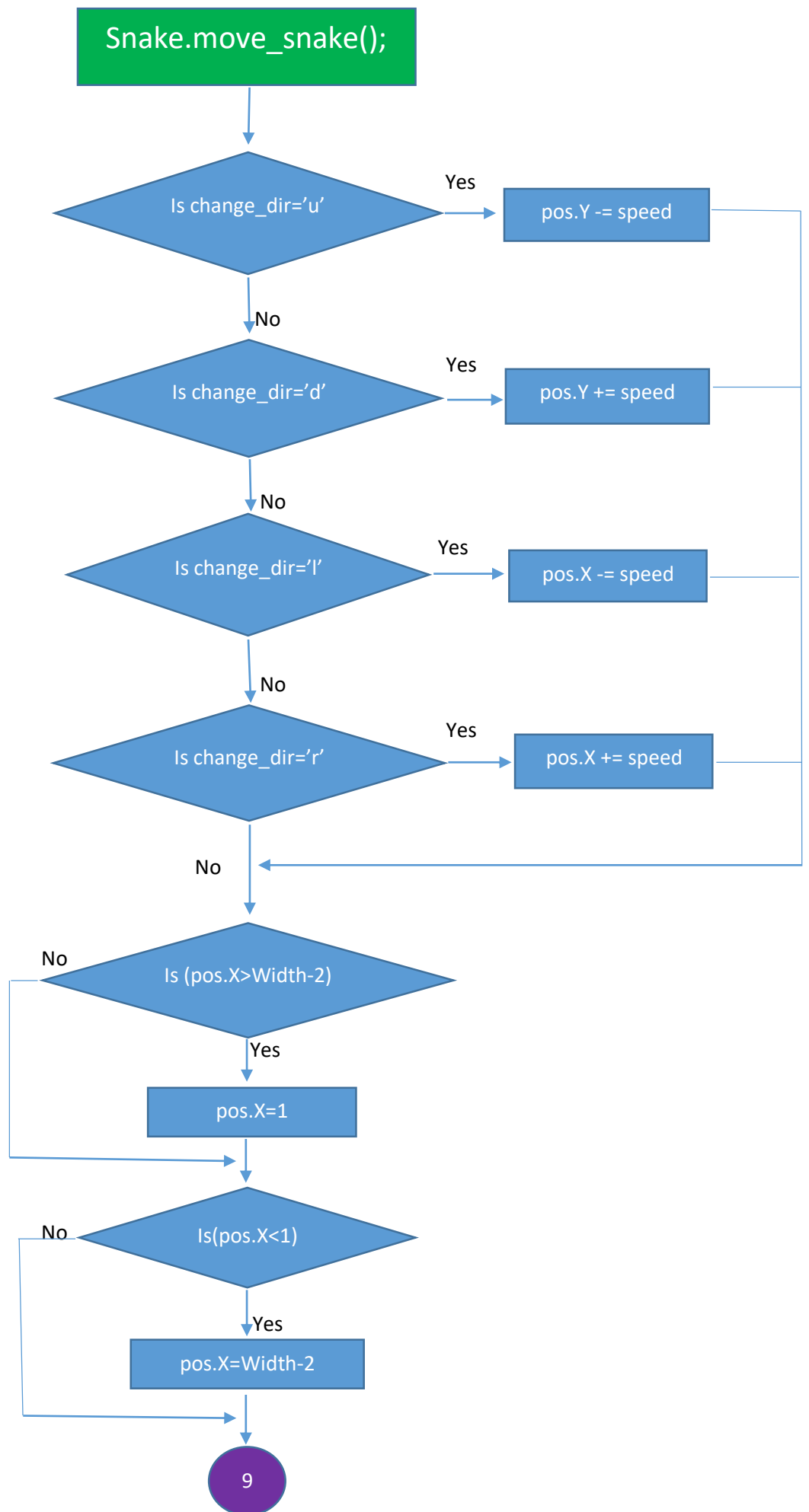


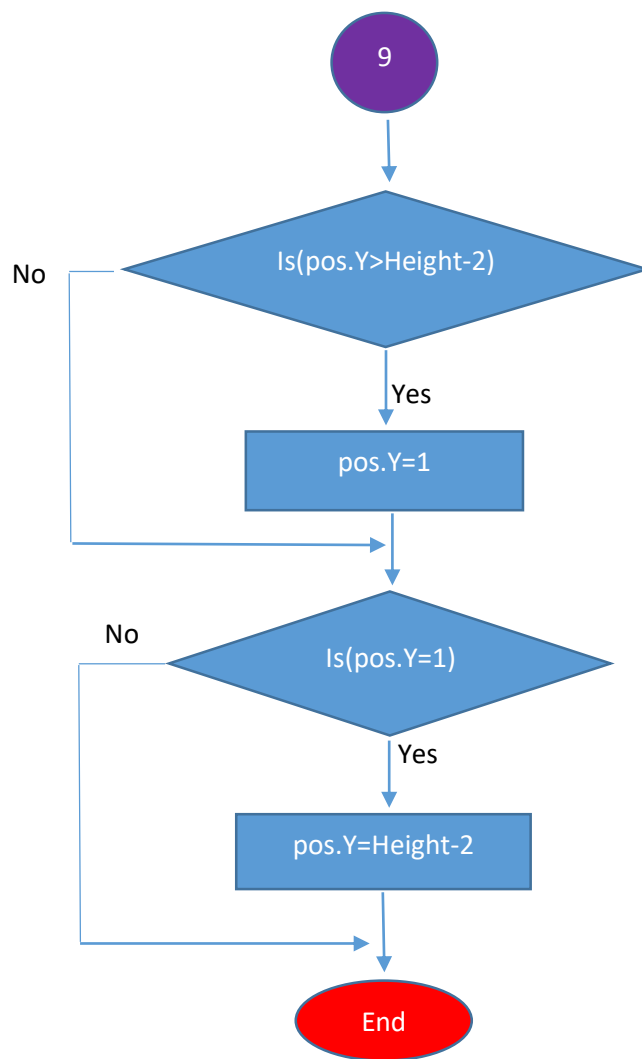


8

7







**{COMPLETED}**