PS # 11

Deadline: 14.06.2020 at 23:50

Hand-in Policy:

Create a cpp file for each part of the assignment and save the screenshot while these parts are running.

Your assignment should consist of 3 cpp files and 3 screenshots.

Zip all files of the assignment. Your zip files should look like this

Ps11_studentnumber_name_surname.zip

1)

The goal for this programming project is to create a simple 2D predator–prey simulation. In this simulation, the prey are ants and the predators are doodlebugs. These critters live in a 20 * 20 grid of cells. Only one critter may occupy a cell at a time. The grid is enclosed, so a critter is not allowed to move off the edges of the world. Time is simulated in steps. Each critter performs some action every time step.
The ants behave according to the following model:
■ *Move* . For every time step, the ants randomly try to move up, down, left, or right. If the neighboring cell in the selected direction is occupied or would move the ant off the grid, then the ant stays in the current cell.
■ *Breed* . If an ant survives for three time steps, at the end of the time step (i.e., after moving) the ant will breed. This is simulated by creating a new ant in an adjacent (up, down, left, or right) cell that is empty. If there is no empty cell available, no breeding occurs. Once an offspring is produced, an ant cannot produce an offspring again until it has survived three more time steps.
The doodlebugs behave according to the following model:
■ *Move* . For every time step, the doodlebug will move to an adjacent cell containing an ant and eat the ant. If there are no ants in adjoining cells, the doodlebug moves according to the same rules as the ant. Note that a doodlebug cannot eat other doodlebugs.
■ *Breed* . If a doodlebug survives for eight time steps, at the end of the time step it will spawn off a new doodlebug in the same manner as the ant.
■ *Starve* . If a doodlebug has not eaten an ant within three time steps, at the end of the third time step it will starve and die. The doodlebug should then be removed from the grid of cells.
During one turn, all the doodlebugs should move before the ants.
Write a program to implement this simulation and draw the world using ASCII characters of “O” for an ant and “X” for a doodlebug. Create a class named Organism that encapsulates basic data common to ants and doodlebugs. This class should have a virtual function named move that is defined in the derived classes of Ant and Doodlebug . You may need additional data structures to keep track of which critters have moved.
Initialize the world with 5 doodlebugs and 100 ants. After each time step, prompt
the user to press Enter to move to the next time step. You should see a cyclical
pattern between the population of predators and prey, although random perturbations
may lead to the elimination of one or both species.

2)

The following shows code to play a guessing game in which two players attempt to guess a number. Your task is to extend the program with objects that represent either a human player or a computer player.

```
bool checkForWin(int guess, int answer)
{
	if (answer == guess)
	{
		cout << "You're right! You win!" << endl;
		return true;
	}
	else if (answer < guess)
		cout << "Your guess is too high." << endl;
	else
		cout << "Your guess is too low." << endl;
	return false;
}
void play(Player &player1, Player &player2)
{
	int answer = 0, guess = 0;
	answer = rand( ) % 100;
	bool win = false;
	while (!win)
	{
		cout << "Player 1's turn to guess." << endl;
		guess = player1.getGuess( );
		win = checkForWin(guess, answer);
		if (win) return;

		cout << "Player 2's turn to guess." << endl;
		guess = player2.getGuess( );
		win = checkForWin(guess, answer);
	}
}
```

The play function takes as input two Player objects. Define the Player class with a virtual function named getGuess( ) . The implementation of Player :: getGuess( ) can simply return 0. Next, define a class named HumanPlayer derived from Player . The implementation of HumanPlayer :: getGuess( ) should prompt the user to enter a number and return the value entered from the keyboard. Next, define a class named ComputerPlayer derived from Player . The implementation of ComputerPlayer :: getGuess( ) should randomly select a number from 0 to 100. Finally, construct a main function that invokes play(Player &player1, Player &player2) with two instances of a HumanPlayer (human versus human), an instance of a HumanPlayer and ComputerPlayer (human versus computer), and two instances of ComputerPlayer (computer versus computer).

3)
Write a void function that takes a linked list of integers and reverses the order of its nodes. The function will have one call-by-reference parameter that is a pointer to the head of the list. After the function is called, this pointer will point to the head of a linked list that has the same nodes as the original list but in the reverse of the order they had in the original list. Note that your

function will neither create nor destroy any nodes. It will simply rearrange nodes. Place your function in a suitable test program.