Chermaine Cheang
Assignment 1 Design

Implement a simulation of Langton's Ant that will move around a grid based on the rules that

1. If the ant is currently standing on a white (' ') cell, the ant will turn 90° to the right and move forward by a distance of 1. The cell where the ant was previously standing will now become black ('#')
2. If the ant is currently standing on a black ('#') cell, the ant will turn 90° to the left and move forward by a distance of 1. The cell where the ant was previously standing on will then change to white (' ')

In order to implement this program, I will need to keep track of the direction the ant is facing. Since there are only 4 directions that the ant could be facing, namely up, down, left and right, I will label these directions with integers 1, 2, 3, and 4 (UP = 1, DOWN = 2, LEFT = 3, RIGHT = 4), and have an integer variable called curFace in Ant Class to store current direction of Ant.

I also need to keep track of what color is the cell the ant is currently standing on. To do these, I will have a string variable called curColor to store current cell color. Then a get function for curColor will take the x and y coordinates of the cell the ant is currently standing on and return the that cell color.

I will need to create an Ant Class with data members as xCoor and yCoor to hold current x and y coordinates of the ant, a pointer to a Grid object for the ant to move on, curColor to hold current cell color and curFace to hold current direction the ant is facing. Functions that I need in my Ant class include:

- void setXCoor(int x): to set x-coordinate of the ant
- void setYCoor(int y): to set y-coordinate of the ant
- int getXCoor(): return x-coordinate of the ant
- int getYCoor(): return y-coordinate of the ant
- void setCurColor(string): set current cell color with parameters passed in
- string getCurColor(): return current color of cell ant currently standing on
- void setCurFace(int dir): set current direction the ant is facing
- int getCurFace(): return current direction the ant is facing
- void randFace(): to generate random starting direction the ant is facing
- void userFace(int dir): set current ant's facing direction to user's specified direction
- void randStart(): to generate a random starting position of the ant if the user wishes
- void userStart(int x, int y): to generate the starting position of the ant based on user's input
- void moveAnt(string curColor): generate ant movement based on current cell color and call updataColor() to change cell color accordingly
- void whiteMove(int face, int x, int y): called by moveAnt if current cell color is white and move ant according to the rule for white cell, call Grid class updateLocation() to update ant's location and display the movement on screen

- void blackMove(int face, int x, int y): called by moveAnt if current cell color is black and move ant according to the rule for black cell, call Grid class updateLocation() to update ant's location and display the movement on screen

As for the array or matrix of cells, I will re-use the Grid class from module A and update the fillGrid() and updateLocation() functions, and add an updateColor() to update cell's color.

```
void Grid::updateColor(std::string curColor, int x, int y){
        if current color == white
                myGrid[y][x] = black
        else
                myGrid[y][x] = white;
}
```

In my main(), I will prompt the user for number of rows and columns for the 2D array, the number of steps the ant should take, and ask the user for starting location of the Ant or randomly start. I will validate user inputs to make sure that all values inputted by user are greater than 0, as the grid has to be at least 1 x 1 in size. I will have a menu function to ask user whether they want to specify starting location or randomly generate a start location.

My menu will look this:
Do you wish to specify the Ant's starting location yourselves or you wish for the system to randomly generate a starting location?
1 – Specify my own
2 – Randomly generate
wait for user's input

Which direction the Ant is facing?
1 – UP (NORTH)
2 – DOWN (SOUTH)
3 – LEFT (WEST)
4 – RIGHT (EAST)
5 – Randomly generate a direction
wait for user's input

I will have my input validation function in a separate .cpp file, so that I can re-use them in the future by just #include the file.

Algorithm for ant's movement:
If the ant is currently standing on a white cell:

- Change the cell color to black.
- If the ant is facing UP, move the ant to the right by a distance of 1, that is increment x-coordinate by 1, and change the ant direction to facing RIGHT

- If the ant is facing DOWN, move the ant to the left, that is decrement x-coordinate by 1, and change the ant direction to facing LEFT
- If the ant is facing LEFT, move the ant upwards, that is decrement y-coordinate by 1, and change the ant direction to facing UP
- If the ant is facing RIGHT, move the ant downwards, that is increment y-coordinate by 1, and change the ant direction to facing DOWN
- Update ant's location

If the ant is currently standing on a black cell:

- Change cell color to white
- If ant is facing UP, move ant to the left, that is decrement x-coordinate by 1, and change the ant direction to facing LEFT
- If the ant is facing DOWN, move the ant to the right, that is increment x-coordinate by 1, and change the ant direction to facing RIGHT
- If the ant is facing LEFT, move the ant downwards, that is increment y-coordinate by 1, and change the ant direction to facing DOWN
- If the ant is facing RIGHT, move the ant upwards, that is decrement y-coordinate by 1, and change the ant direction to facing UP
- Update ant's location

Test Plan

Test Case	Input Values	Driver functions	Expected Outcomes
1	Row = 0 Col = 0	intValidate(input)	Invalid input. Prompt user to re-enter number of rows and columns
2	Row = 0 Col = 10	intValidate(input)	Invalid row input. Prompt user to re-enter number of rows and columns
3	Row = 10 Col = 0	intValidate(input)	Invalid column input. Prompt user to re-enter number of rows and columns
4	Row = 10 Col = 10 User's choice for starting location of ant = 9	intValidate(input)	Invalid choice input. Prompt user to re-enter choices. 1 for specifying starting location 2 for randomly generate a starting location
5	Row = 10 Col = 10 User's choice = 1 User's x and y coordinates	intValidate(input) updateLocation(x,y)	Screen will display current location of ant at position (x,y)
6	User's choice = 0 (randomly generate a	randStart()	Screen will display location of ant

	starting a position)		
7	User's choice for direction of ant facing (1, 2, 3, or 4)	intValidate(input) setCurFace()	If user entered a valid choices, then ant should be facing the selected direction
8	Number of steps entered by user	intValidate(input) moveAnt() whiteMove() blackMove()	If user entered a valid number, the ant should move the set amount times according to the rules, and the movement will be display on screen.
9	Current cell color is white	whiteMove()	The ant should turn right 90 degree and the cell should be now black.
10	Current cell color is black	blackMove()	The ant should turn left 90 degree and the cell should be now white

Pseudocode

//Ant Class specification file (Ant.hpp)

```
include guard
include Grid.hpp
class Ant {
        private:
                variable for current direction ant is facing
                variable for x-coordinate
                variable for y-coordinate
                variable for current cell's color
                pointer variable to Grid object

        public:
                Ant(Grid*, int x, int y, int face); //constructor for when user specified starting
                                                    //location and direction
                Ant(Grid*, int x, int y); //constructor for when user specified starting location but
                                          //not direction
                Ant(Grid*); //constructor for when user doesn't not specified starting location or
                            //direction
                Ant(); //default constructor
                void setXCoor(int x);
```

```
        void setYCoor(int y);
        int getXCoor();
        int getYCoor();
        void setColor(std::string);
        string getColor();
        void setFace(int dir);
        int getFace();
        void randFace();
        void userFace(int dir);
        void randStart();
        void userStart(int x, int y);
        void moveAnt(string curColor);
        void whiteMove(int face, int x, int y);
        void blackMove(int face, int x, int y);
};
```

//Ant class implementation file (Ant.cpp)

```
#include “Ant.hpp”
Ant::Ant(){
        Set pointer variable to NULL;
}

Ant::Ant(Grid* g){
        Set pointer variable to g;
        Call randStart() to randomly generate a start location for ant
        Call randFace() to randomly generate a start direction for ant
        Set color cell color to white;
}

Ant::Ant(Grid* g, int x, int y){
        Set pointer variable to g;
        Set x-coordinate to x;
        Set y-coordinate to y;
        Call randFace() to randomly generate the direction the ant is facing;
        Set current cell color to white;
}

Ant::Ant(Grid* g, int x, int y, int dir){
        Set pointer variable to g;
        Set x-coordinate to x;
        Set y-coordinate to y;
        Set current direction to dir;
        Set current cell color to white;
}
```

```
void Ant::setXCoor(int x){
        Set x-coordinate to x;
}

void Ant::setYCoor(int y){
        Set y-coordinate to y;
}

int Ant:: getXCoor(){
        return x-coordinate;
}

int Ant:: getYCoor(){
        return y-coordinate;
}

void Ant::setCurColor(std::string color){
        set current color to color;
}

std::string Ant::getCurColor(){
        return current color;
}

void Ant::setCurFace(int dir){
        set current direction the ant is facing to dir;
}

int Ant::getCurFace(){
        return current direction the ant is facing;
}

void Ant::randFace(){
        unsigned seed;
        seed = time(0);
        srand(seed);

        int antRanDir = rand() % 4 + 1;
        call setCurFace with random number generate to set current direction ant is facing;
}

void Ant::userFace(int dir){
        set current direction the ant is facing to dir;
}
```

```
void Ant::randStart(){
	unsigned seed;
	seed = time(0);
	srand(seed);

	int randX, randY;
	randX = rand();
	randY = rand();
	while (randX and randY greater than grid size)
		randX = rand();
		randY = rand();
	grid pointer ->updateLocation(randX, randY);
}

void Ant::userStart(int x, int y){
	grid pointer ->updateLocation(x , y);
}

void Ant::moveAnt(){
	std::string curCol;
	int curX, curY, curF;
	curX = getXCoor();
	curY = getYCoor();
	curF = getCurFace();
	curCol = getCurColor();
	if curColor is white
		grid pointer ->updateColor(curColor, curX, curY);
		whiteMove(curF, curX, curY);
	else
		grid pointer ->updateColor(curColor, curX, curY);
		blackMove(curF, curX, curY);
}

void Ant::whiteMove(int face, int x, int y){
	if face == UP {
		x += 1;
		face = RIGHT;
	}
	else if face == DOWN {
		x -= 1;
		face = LEFT;
	}
	else if face == LEFT {
```

```
                y -= 1;
                face = UP;
        }
        else {
                y += 1;
                face = DOWN;
        }
        pointer variable->updateLocation(x, y);
}

void Ant::blackMove(int face, int x, int y){
        if face == UP{
                x -= 1;
                face = LEFT;
        }
        else if face == DOWN {
                x += 1;
                face = RIGHT;
        }
        else if face == LEFT {
                y += 1;
                face = DOWN;
        }
        else {
                y -= 1;
                face = UP;
        }
        pointer variable->updateLocation(x, y);
}
```

//validate.cpp

```
#include <iostream>

bool intValidate(int input){
        bool valid;
        if (input <= 0){
                cout << “Invalid input. Number must be greater than 0.” << endl;
                valid = false;
        }
        else
                valid = true;
        return valid;
}
```

//menu.cpp

```
#include <iostream>

int startMenu(){
	int choice;
	bool valid;
	cout << “Do you wish to specify the Ant’s starting location yourselves or you wish for the
	system to randomly generate a starting location?” << endl;
	cout << “1 – Specify my own” << endl;
	cout << “2 – Randomly generate” << endl;
	cout << “Enter 1 or 2 as your choice.” << endl;
	cin >> choice;
	valid = intValidate(choice);
	while (!valid) {
		ask user to reenter choices
		update valid
	}
	return choice;
}

int dirMenu(){
	int choice;
	bool valid;
	cout << “Which direction the Ant is facing?” << endl;
	cout << “1 – UP (NORTH)” << endl;
	cout << “2 – DOWN (SOUTH)” << endl;
	cout << “3 – LEFT (WEST)” << endl;
	cout << “4 – RIGHT (EAST)” << endl;
	cin >> choice;
	valid = intValidate(choice);
	while not valid
		ask user to reenter choices
		update valid
	}
	return choice;
}
```

//main.cpp

```
#include “Grid.hpp”
#include “Ant.hpp”
#include “validate.cpp”
#include “startMenu.cpp”
#include <iostream>
```

```
using std::cout;
using std::cin;
using std::endl;

int main() {
	int row, column, choice, step;
	bool validRow, validCol, validStep;
	cout << “Enter number of rows and columns” << endl;
	cin >> row >> column;
	validRow = intValidate(row);
	validCol = intValidate(column);
	while (!validRow || !validCol){
		cout<< “Enter number of rows and columns” << endl;
		cin >> row >> column;
		validRow = intValidate(row);
		validCol = intValidate(col);
	}
	Grid* grid1(row, column);

	choice = startMenu();

	if choice == 1
		ask user for x and y coordinates
		validate x and y coordinates entered
		if valid
			call userStart(x, y) to set user starting location
		if not valid
			ask user for x and y coordinates again
	if choice == 2
		call randStart()

	choice = dirMenu();

	if choice == 1
		call setCurFace(1);
	if choice == 2
		setCurFace(2)
	if choice == 3
		setCurFace(3)
	if choice == 4
		setCurFace(4)

	ask user for number of steps
	cin >> step;
```

```
        validStep = intValidate(step)
        while validStep is false
                prompt user to re-enter number of steps
                update validStep
        }

        for (int i = 0; i < step ; i++) {
                moveAnt();
                printGrid(row, column);
        }

        return 0
}
```