**C++ Part II (INFO1-CE9265) Spring 2015 – Homework 2**

Clement Chan

**Question 1a:**

Figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

#include <string>

#include <iostream>

class figure{

public:

const char\* erase();

const char\* draw();

void center();

};

#endif

Figure.cpp

#include <string>

#include <cstdlib>

#include <iostream>

#include "Figure.h"

const char\* figure :: erase(){

return "This is erase from class figure!";

}

const char\* figure :: draw(){

return "This is draw from class figure!";

}

void figure :: center(){

std::cout<<"Center is called from figure..."<<std::endl;

std::cout<<erase()<<std::endl;

std::cout<<draw()<<std::endl;

}

Triangle.h

#ifndef TRIANGLE\_H

#define TRIANGLE\_H

#include <string>

#include <iostream>

#include "figure.h"

class triangle : public figure{

public:

const char\* erase();

const char\* draw();

void center();

};

#endif

Triangle.cpp

#include <string>

#include <cstdlib>

#include <iostream>

#include "Triangle.h"

const char\* triangle :: erase(){

return "This is erase from class triangle!";

}

const char\* triangle :: draw(){

return "This is draw from class triangle!";

std::cout<<" "<<std::endl;

}

void triangle :: center(){

std::cout<<"Center is called from triangle..."<<std::endl;

std::cout<<erase()<<std::endl;

std::cout<<draw()<<std::endl;

}

Rectangle.h

#ifndef RECTANGLE\_H

#define RECTANGLE\_H

#include <string>

#include <iostream>

#include "figure.h"

class rectangle : public figure{

public:

const char\* erase();

const char\* draw();

void center();

};

#endif

Rectangle.cpp

#include <string>

#include <cstdlib>

#include <iostream>

#include "Rectangle.h"

const char\* rectangle :: erase(){

return "This is erase from class rectangle!";

}

const char\* rectangle :: draw(){

return "This is draw from class rectangle!";

std::cout<<" "<<std::endl;

}

void rectangle :: center(){

std::cout<<"Center is called from rectangle..."<<std::endl;

std::cout<<erase()<<std::endl;

std::cout<<draw()<<std::endl;

}

Main.cpp

#include <iostream>

#include "figure.h"

#include "Rectangle.h"

#include "Triangle.h"

int main(){

triangle tri;

figure \*fig = &tri;

fig->draw();

std::cout <<

"\nDerived class Triangle object calling center()\n";

fig->center();

rectangle rect;

figure \*fig1 = &rect;

fig1->draw();

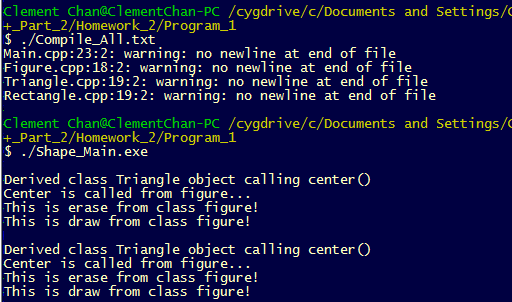
std::cout <<

"\nDerived class Triangle object calling center()\n";

fig1->center();

}

**Output**



**Question 1b: Changing the functions from const to virtual**

Figure.h

#ifndef FIGURE\_H

#define FIGURE\_H

#include <string>

#include <iostream>

class figure{

public:

virtual const char\* erase();

virtual const char\* draw();

void center();

};

#endif

Figure.cpp

#include <string>

#include <cstdlib>

#include <iostream>

#include "Figure.h"

const char\* figure :: erase(){

return "This is erase from class figure!";

}

const char\* figure :: draw(){

return "This is draw from class figure!";

}

void figure :: center(){

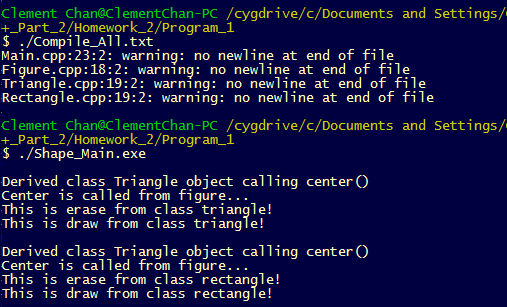
std::cout<<"Center is called from figure..."<<std::endl;

std::cout<<erase()<<std::endl;

std::cout<<draw()<<std::endl;

}

**Output**



Compile\_All.txt:

g++ -c Main.cpp && g++ -c Figure.cpp && g++ -c Triangle.cpp && g++ -c Rectangle.cpp && g++ -o Shape\_Main Main.o Figure.o Triangle.o Rectangle.o

**Question 1c:**

Because the function erase() and draw() in 1a was pointed towards the base class since both functions are not virtual, this makes pointers in main() to display base class messages. For 1b, since erase() and draw() are virtual, the initiation in main() tells the display to go look for functions at a derived class. Therefore, 1b displays the functions in derived class.

**Question 3:**

Organism (Predator).h

#ifndef Predator\_H

#define Predator\_H

#include <string>

#include <iostream>

class Predator{

protected:

char \*\*Grid;

int \*\*DoodleBug\_Coord;

int \*\*DoodleBug\_Coord\_1;

int \*\*Ants\_Coord;

int size;

int counter;

int \*counter\_DB;

int \*counter\_starve;

public:

Predator(); //Default Constructor

void setgrid(int grid\_size);

void DisplayGrid();

char get\_grid(int x, int y);

int get\_size();

~Predator(); // Destructor to destroy grid

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Define Pure Virtual function to be extended to Ants and Bugs\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

virtual void move() = 0;

virtual void Breed() = 0;

};

#endif

Organism (Predator).cpp

#include "Predator.h"

Predator :: Predator(){

counter = 0;

}

void Predator:: setgrid(int grid\_size){

size = grid\_size;

Grid = new char\*[size];

for(int i=0; i<size; i++){

Grid[i] = new char[size];

}

for(int i=0; i<size; i++){

for(int j=0; j<size; j++){

Grid[i][j] = '.';

}

}

}

void Predator :: DisplayGrid(){

for(int i = 0; i < size; i++){

for(int j=0; j < size; j++){

if(j < size-1){

std::cout << Grid[i][j] << " ";

}else if (j = size-1){

std::cout << Grid[i][j] << std::endl;

}

}

}

}

char Predator :: get\_grid(int x, int y){

return Grid[x][y];

}

int Predator :: get\_size(){

return size;

}

Predator :: ~Predator(){

std::cout << "Initiating Predator (Base) Destructor ... " << std::endl;

delete[] \*Grid;

}

Ants.h

#ifndef Ants\_H

#define Ants\_H

#include <string>

#include <iostream>

#include <cmath>

#include <cstdlib>

#include <time.h>

#include "Predator.h"

class Ants : public Predator{

protected:

int number\_of\_ants;

int doodle\_counter;

int number\_DB;

public:

Ants(); //Default Constructor

void setcoord(int numants, int numDB);

void intakegrid(char antgrid, int x, int y);

void MarkAntsOnGrid(int x, int y); //Mark where the ants is

void MarkMoveGrid(int x, int y); //Mark where the ants moved to

~Ants(); // Destructor to destroy grid

//Virtual Functions overloading

void move();

void Breed();

};

#endif

Ants.cpp

#include "Ants.h"

Ants :: Ants(){

//Deliberately left empty

}

void Ants:: setcoord(int numants, int numDB){

srand(time(NULL));

number\_of\_ants = numants;

counter = 0;

doodle\_counter = 0;

Ants\_Coord = new int\*[number\_of\_ants];

for(int i=0; i<number\_of\_ants; i++){

Ants\_Coord[i] = new int[2];

}

number\_DB = numDB;

DoodleBug\_Coord = new int\*[numDB];

for(int i=0; i < numDB; i++){

DoodleBug\_Coord[i] = new int[2];

}

for(int i=0; i<number\_of\_ants; i++){

Ants\_Coord[i][0] = 0 + ( std::rand() % size); //Generate random numbers from 0 to 20

Ants\_Coord[i][1] = 0 + ( std::rand() % size);

int x\_coord = Ants\_Coord[i][0];

int y\_coord = Ants\_Coord[i][1];

for(int k=0; k<i; k++){

if(Ants\_Coord[k][0] == x\_coord && Ants\_Coord[k][1] == y\_coord){

int search = 0;

if((x\_coord+1)>0 && (x\_coord+1) < size && Grid[x\_coord+1][y\_coord] =='.' &&

search == 0){

Grid[x\_coord+1][y\_coord] = 'O';

search++;}

else if((x\_coord-1)>0 && (x\_coord-1) < size && Grid[x\_coord-1][y\_coord] =='.' && search == 0){

Grid[x\_coord-1][y\_coord] = 'O';

search++;}

else if((y\_coord+1)>0 && (y\_coord+1) < size && Grid[x\_coord][y\_coord + 1] =='.' && search == 0){

Grid[x\_coord][y\_coord + 1] = 'O';

search++;}

else if((y\_coord-1)>0 && (y\_coord-1) < size && Grid[x\_coord][y\_coord - 1] =='.' && search == 0){

Grid[x\_coord][y\_coord - 1] = 'O';

search++;}

else if((x\_coord+1)>0 && (x\_coord+1) < size && (y\_coord + 1)>0 && (y\_coord + 1) < size && Grid[x\_coord + 1][y\_coord + 1] =='.' && search == 0){

Grid[x\_coord + 1][y\_coord + 1] = 'O';

search++;}

else if((x\_coord - 1)>0 && (x\_coord - 1) < size && (y\_coord + 1)>0 && (y\_coord + 1) < size &&Grid[x\_coord - 1][y\_coord + 1] =='.' && search == 0){

Grid[x\_coord - 1][y\_coord + 1] = 'O';

search++;}

else if((x\_coord+1)>0 && (x\_coord+1) < size && (y\_coord - 1)>0 && (y\_coord - 1) < size &&Grid[x\_coord + 1][y\_coord - 1] =='.' && search == 0){

Grid[x\_coord + 1][y\_coord - 1] = 'O';

search++;}

else if((x\_coord - 1)>0 && (x\_coord - 1) < size && (y\_coord - 1)>0 && (y\_coord - 1) < size && Grid[x\_coord - 1][y\_coord - 1] =='.' && search == 0){

Grid[x\_coord - 1][y\_coord - 1] = 'O';

search++;}

}

};

MarkAntsOnGrid(x\_coord, y\_coord);

}

}

void Ants::MarkAntsOnGrid(int x, int y){

Grid[x][y] = 'O';

}

void Ants::MarkMoveGrid(int x, int y){

Grid[x][y] = '.';

}

void Ants::move(){

srand(time(NULL));

int x\_coord;

int y\_coord;

int temp;

int k;

int k1;

counter = (counter+1)%3;

//Updating all the DoodleBug's Location

k = 0;

for(int i=0; i<size; i++){

for(int j=0; j<size;j++){

if(Grid[i][j] == 'X'){

k++;

}

}

}

delete[] \*DoodleBug\_Coord;

DoodleBug\_Coord = new int\*[k];

for(int i=0; i<k; i++){

DoodleBug\_Coord[i] = new int[2];

}

k1 = 0;

for(int i=0; i<size; i++){

for(int j=0; j<size;j++){

if(Grid[i][j] == 'O'){

k1++;

}

}

}

delete[] \*Ants\_Coord;

Ants\_Coord = new int\*[k1];

for(int i=0; i<k1; i++){

Ants\_Coord[i] = new int[2];

}

k = 0; //reset k to 0

for(int i=0; i<size; i++){

for(int j=0; j<size;j++){

if(Grid[i][j] == 'X'){

DoodleBug\_Coord[k][0] =0;

DoodleBug\_Coord[k][0] =i;

DoodleBug\_Coord[k][1] =0;

DoodleBug\_Coord[k][1] =j;

k++;

}

}

}

k1 = 0; //reset k1 to 0

for(int i=0; i<size; i++){

for(int j=0; j<size;j++){

if(Grid[i][j] == 'O'){

Ants\_Coord[k1][0] =0;

Ants\_Coord[k1][0] =i;

Ants\_Coord[k1][1] =0;

Ants\_Coord[k1][1] =j;

k1++;

}

}

}

number\_of\_ants = k1;

for(int i=0; i<number\_of\_ants; i++){

x\_coord = Ants\_Coord[i][0];

y\_coord = Ants\_Coord[i][1];

int coord\_move = 0 + (std::rand() % 2);

int random\_move = -1 + (std::rand() % 2);

if(random\_move == 0){

random\_move = 1;

}

for(int j=0; j<k;j++){

if((x\_coord + random\_move == DoodleBug\_Coord[j][0] && y\_coord ==

DoodleBug\_Coord[j][1]) ||(y\_coord + random\_move == DoodleBug\_Coord[j][1]

&& x\_coord == DoodleBug\_Coord[j][0]) || x\_coord + random\_move >= size-1 ||

x\_coord + random\_move <= 0 || y\_coord + random\_move >=size -1 ||

y\_coord+random\_move <=0){

MarkAntsOnGrid(x\_coord,y\_coord);

}

else{

if(coord\_move == 0 && Grid[x\_coord+random\_move][y\_coord] != 'X'){

MarkAntsOnGrid(x\_coord+random\_move, y\_coord);

if(Grid[x\_coord][y\_coord] != 'X'){

MarkMoveGrid(x\_coord, y\_coord);

}

}else if(coord\_move == 1 && Grid[x\_coord][y\_coord+random\_move] != 'X'){

MarkAntsOnGrid(x\_coord, y\_coord + random\_move);

if(Grid[x\_coord][y\_coord] != 'X'){

MarkMoveGrid(x\_coord, y\_coord);

}

}

}

}

}

}

void Ants::Breed(){

if(counter == 2){

int ants\_add = 0;

int ants\_add\_1 = ants\_add;

for(int i=0; i<number\_of\_ants; i++){

int x\_coord = Ants\_Coord[i][0];

int y\_coord = Ants\_Coord[i][1];

int search=0;

if(x\_coord == 0){x\_coord =1;};

if(x\_coord == size - 1){x\_coord = size-2;};

if(y\_coord == 0){y\_coord = 1;};

if(y\_coord == size - 1){y\_coord = size-2;};

if((x\_coord+1)>0 && (x\_coord+1) < size-1 && Grid[x\_coord+1][y\_coord] =='.' && Grid[x\_coord+1][y\_coord] !='X' && search == 0){

Grid[x\_coord+1][y\_coord] = 'O';

ants\_add++;

search++;}

else if((x\_coord-1)>0 && (x\_coord-1) < size-1 && Grid[x\_coord-1][y\_coord] =='.' && Grid[x\_coord-1][y\_coord] !='X' && search == 0){

Grid[x\_coord-1][y\_coord] = 'O';

ants\_add++;

search++;}

else if((y\_coord+1)>0 && (y\_coord+1) < size-1 && Grid[x\_coord][y\_coord + 1] =='.' && Grid[x\_coord][y\_coord+1] !='X' && search == 0){

Grid[x\_coord][y\_coord + 1] = 'O';

ants\_add++;

search++;}

else if((y\_coord-1)>0 && (y\_coord-1) < size-1 && Grid[x\_coord][y\_coord - 1] =='.' && Grid[x\_coord][y\_coord-1] !='X' && search == 0){

Grid[x\_coord][y\_coord - 1] = 'O';

ants\_add++;

search++;}

}//for statement

} //if statement

}

void Ants::intakegrid(char grid, int x, int y){

Grid[x][y] = grid;

doodle\_counter = 0;

if(Grid[x][y] == 'X'){

DoodleBug\_Coord[doodle\_counter][0] = x;

DoodleBug\_Coord[doodle\_counter][1] = y;

//std::cout<< DoodleBug\_Coord[doodle\_counter][0] << "," <<DoodleBug\_Coord[doodle\_counter][1] << std::endl;

doodle\_counter+=1;

}

}

Ants :: ~Ants(){

std::cout << "Initiating Ants (Derived) Destructor ... " << std::endl;

delete[] \*Ants\_Coord;

}

DoodleBug.h

#ifndef DoodleBug\_H

#define DoodleBug\_H

#include <string>

#include <iostream>

#include <cmath>

#include <cstdlib>

#include <time.h>

#include "Predator.h"

class DoodleBug : public Predator{

protected:

int number\_of\_DB;

int \*ant\_ate;

int \*DB\_Coord;

public:

DoodleBug(); //Default Constructor

void setcoord(int numdb);

void intakegrid(char antgrid, int x, int y);

void MarkDBOnGrid(int x, int y); //Mark where the ants is

void MarkDBGrid(int x, int y); //Mark where the ants moved to

void starve(); //Only Doodlebug has it

void make\_new\_coord();

~DoodleBug(); // Destructor to destroy grid

//Virtual Functions overloading

void move();

void Breed();

};

#endif

DoodleBug.cpp

#include "DoodleBug.h"

DoodleBug :: DoodleBug(){

//Deliberately left empty

}

void DoodleBug:: setcoord(int numdb){

srand(time(NULL));

number\_of\_DB = numdb;

DB\_Coord = new int[number\_of\_DB];

counter\_starve = new int[number\_of\_DB];

counter\_DB = new int[number\_of\_DB];

for(int i=0; i<number\_of\_DB; i++){

DB\_Coord[i] = 0;

}

for(int i=0; i<number\_of\_DB; i++){

counter\_starve[i] = 0;

}

for(int i=0; i<number\_of\_DB; i++){

counter\_DB[i] = 0;

}

DoodleBug\_Coord = new int\*[number\_of\_DB];

for(int i=0; i<number\_of\_DB; i++){

DoodleBug\_Coord[i] = new int[2];

}

for(int i=0; i<number\_of\_DB; i++){

DoodleBug\_Coord[i][0] = 0 + ( std::rand() % size); //Generate random numbers from 0 to 20

DoodleBug\_Coord[i][1] = 0 + ( std::rand() % size);

int x\_coord = DoodleBug\_Coord[i][0];

int y\_coord = DoodleBug\_Coord[i][1];

for(int k=0; k<i; k++){

if(DoodleBug\_Coord[k][0] == x\_coord && DoodleBug\_Coord[k][1] == y\_coord){

int search = 0;

if((x\_coord+1)>0 && (x\_coord+1) < size && Grid[x\_coord+1][y\_coord] =='.' &&

Grid[x\_coord+1][y\_coord] !='O' && search == 0){

Grid[x\_coord+1][y\_coord] = 'X';

search++;}

else if((x\_coord-1)>0 && (x\_coord-1) < size && Grid[x\_coord-1][y\_coord] =='.' &&

Grid[x\_coord-1][y\_coord] !='O' && search == 0){

Grid[x\_coord-1][y\_coord] = 'X';

search++;}

else if((y\_coord+1)>0 && (y\_coord+1) < size && Grid[x\_coord][y\_coord + 1] =='.'

&& Grid[x\_coord][y\_coord + 1] !='O' && search == 0){

Grid[x\_coord][y\_coord + 1] = 'X';

search++;}

else if((y\_coord-1)>0 && (y\_coord-1) < size && Grid[x\_coord][y\_coord - 1] =='.'

&& Grid[x\_coord][y\_coord - 1] !='O' && search == 0){

Grid[x\_coord][y\_coord - 1] = 'X';

search++;}

else if((x\_coord+1)>0 && (x\_coord+1) < size && (y\_coord + 1)>0 && (y\_coord + 1)

< size && Grid[x\_coord + 1][y\_coord + 1] =='.' && Grid[x\_coord + 1][y\_coord +

1] !='O' && search == 0){

Grid[x\_coord + 1][y\_coord + 1] = 'X';

search++;}

else if((x\_coord - 1)>0 && (x\_coord - 1) < size && (y\_coord + 1)>0 && (y\_coord +

1) < size && Grid[x\_coord - 1][y\_coord + 1] =='.' && Grid[x\_coord - 1][y\_coord +

1] !='O' && search == 0){

Grid[x\_coord - 1][y\_coord + 1] = 'X';

search++;}

else if((x\_coord+1)>0 && (x\_coord+1) < size && (y\_coord - 1)>0 && (y\_coord - 1)

< size && Grid[x\_coord + 1][y\_coord - 1] =='.' && Grid[x\_coord + 1][y\_coord –

1] !='O' && search == 0){

Grid[x\_coord + 1][y\_coord - 1] = 'X';

search++;}

else if((x\_coord - 1)>0 && (x\_coord - 1) < size && (y\_coord - 1)>0 && (y\_coord -

1) < size && Grid[x\_coord - 1][y\_coord - 1] =='.' && Grid[x\_coord - 1][y\_coord –

1] !='O' && search == 0){

Grid[x\_coord - 1][y\_coord - 1] = 'X';

search++;}

}

};

MarkDBOnGrid(x\_coord, y\_coord);

}

}

void DoodleBug::MarkDBOnGrid(int x, int y){

Grid[x][y] = 'X';

}

void DoodleBug::MarkDBGrid(int x, int y){

Grid[x][y] = '.';

}

void DoodleBug::intakegrid(char antgrid, int x, int y){

Grid[x][y] = antgrid;

}

void DoodleBug::move(){

srand(time(NULL));

int x\_coord;

int y\_coord;

int temp;

int k;

k = 0;

for(int i=0; i<size; i++){

for(int j=0; j<size;j++){

if(Grid[i][j] == 'X'){

k++;

}

}

}

delete[] \*DoodleBug\_Coord;

DoodleBug\_Coord = new int\*[k];

for(int i=0; i<k; i++){

DoodleBug\_Coord[i] = new int[2];

}

k = 0;

for(int i=0; i<size; i++){

for(int j=0; j<size;j++){

if(Grid[i][j] == 'X'){

DoodleBug\_Coord[k][0] =0;

DoodleBug\_Coord[k][0] =i;

DoodleBug\_Coord[k][1] =0;

DoodleBug\_Coord[k][1] =j;

k++;

}

}

}

number\_of\_DB = k;

ant\_ate = new int[number\_of\_DB];

for (int i=0; i < number\_of\_DB; i++){

counter\_DB[i] = (counter\_DB[i]+1)%8;

int search = 0;

int coord\_move = 0 + (std::rand() % 2);

int random\_move = -1 + (std::rand() % 2);

if(random\_move == 0){

random\_move = 1;

}

//std::cout<<DoodleBug\_Coord[i][0]<<","<<DoodleBug\_Coord[i][1]<<std::endl;

//std::cout<<coord\_move<<","<<random\_move<<std::endl;

x\_coord = DoodleBug\_Coord[i][0];

y\_coord = DoodleBug\_Coord[i][1];

//std::cout<<x\_coord<<","<<y\_coord<<std::endl;

if(x\_coord < size-1 && Grid[x\_coord+1][y\_coord]=='O' && search == 0){ //Adjacent Left

MarkDBGrid(x\_coord,y\_coord); //x\_coord

if(x\_coord== size -1 ){

x\_coord = x\_coord - 2;

}

DoodleBug\_Coord[i][0] = x\_coord + 1;

x\_coord = DoodleBug\_Coord[i][0];

MarkDBOnGrid(x\_coord, y\_coord);

search++;

ant\_ate[i] += 1;

counter\_starve[i] = 0; // reset counter starve

//std::cout<<counter\_starve[i] << std::endl;

}

else if(x\_coord > 1 && Grid[x\_coord-1][y\_coord]=='O' && search == 0){ //Adjacent Right

MarkDBGrid(x\_coord,y\_coord);

if(x\_coord == 0){

x\_coord = x\_coord + 2;

}

DoodleBug\_Coord[i][0] = x\_coord - 1;

x\_coord = DoodleBug\_Coord[i][0];

MarkDBOnGrid(x\_coord, y\_coord);

search++;

ant\_ate[i] += 1;

counter\_starve[i] = 0; // reset counter starve

//std::cout<<counter\_starve[i] << std::endl;

}

else if(y\_coord < size-1 && Grid[x\_coord][y\_coord+1]=='O' && search == 0){ //Adjacent Up

MarkDBGrid(x\_coord,y\_coord); //y\_coord

if(y\_coord == size -1){

y\_coord = y\_coord - 2;

}

DoodleBug\_Coord[i][1] = y\_coord+1;

y\_coord = DoodleBug\_Coord[i][1];

MarkDBOnGrid(x\_coord, y\_coord);

search++;

ant\_ate[i] += 1;

counter\_starve[i] = 0; // reset counter starve

//std::cout<<counter\_starve[i] << std::endl;

}

else if(y\_coord > 1 && Grid[x\_coord][y\_coord-1]=='O' && search == 0){ //Adjacent Down

MarkDBGrid(x\_coord,y\_coord); //y\_coord

if(y\_coord == 0){

y\_coord = y\_coord + 2;

}

DoodleBug\_Coord[i][1] = y\_coord-1;

y\_coord = DoodleBug\_Coord[i][1];

MarkDBOnGrid(x\_coord, y\_coord);

search++;

ant\_ate[i] += 1;

counter\_starve[i] = 0; // reset counter starve

//std::cout<<counter\_starve[i] << std::endl;

}

else if(coord\_move == 0){

temp = DoodleBug\_Coord[i][0];

//std::cout<<"This is temp: " << temp << std::endl;

if(temp == 0){

temp = 1;

}else if(temp == size - 1){

temp = size - 2;

}

temp += random\_move;

y\_coord = DoodleBug\_Coord[i][1]; //Move y-axis location

if(temp < 0 || temp > size - 1 || Grid[temp][y\_coord] == 'X'){

x\_coord = DoodleBug\_Coord[i][0]; //If the temp moved out of scope, nothing changed

y\_coord = DoodleBug\_Coord[i][1];

MarkDBOnGrid(x\_coord, y\_coord);

ant\_ate[i] -= 1;

counter\_starve[i]+=1;

counter\_starve[i] = counter\_starve[i] %3;

//std::cout<<counter\_starve[i] << std::endl;

}

else if (Grid[temp][y\_coord] == '.'){

x\_coord = DoodleBug\_Coord[i][0]; //If the temp is not out of scope, replace 'X' with '.' and write 'X' in new grid

y\_coord = DoodleBug\_Coord[i][1];

MarkDBGrid(x\_coord, y\_coord);

DoodleBug\_Coord[i][0] = temp;

x\_coord = DoodleBug\_Coord[i][0];

MarkDBOnGrid(x\_coord, y\_coord);

ant\_ate[i] -= 1;

counter\_starve[i]+=1;

counter\_starve[i] = counter\_starve[i] %3;

//std::cout<<counter\_starve[i] << std::endl;

}

}

else if(coord\_move == 1){

temp = DoodleBug\_Coord[i][1];

//std::cout<<"This is temp: " << temp << std::endl;

if(temp == 0){

temp = 1;

}else if(temp == size-1){

temp = size - 2;

}

temp += random\_move;

x\_coord = DoodleBug\_Coord[i][0]; //Move x-axis location

if(temp < 0 || temp > size - 1 || Grid[x\_coord][temp] == 'X'){

x\_coord = DoodleBug\_Coord[i][0]; //If the temp moved out of scope and if any adjacent is occupied, nothing changed

y\_coord = DoodleBug\_Coord[i][1];

MarkDBOnGrid(x\_coord, y\_coord);

ant\_ate[i] -= 1;

counter\_starve[i]+=1;

counter\_starve[i] = counter\_starve[i] %3;

//std::cout<<counter\_starve[i] << std::endl;

}

else if (Grid[temp][y\_coord] == '.'){

x\_coord = DoodleBug\_Coord[i][0]; //If the temp is not out of scope, replace 'X' with '.' and write 'X' in new grid

y\_coord = DoodleBug\_Coord[i][1];

MarkDBGrid(x\_coord, y\_coord);

DoodleBug\_Coord[i][1] = temp;

y\_coord = DoodleBug\_Coord[i][1];

MarkDBOnGrid(x\_coord, y\_coord);

ant\_ate[i] -= 1;

counter\_starve[i]+=1;

counter\_starve[i] = counter\_starve[i] %3;

//std::cout<<counter\_starve[i] << std::endl;

}

}

//std::cout<<DoodleBug\_Coord[i][0]<<","<<DoodleBug\_Coord[i][1]<<std::endl;

}

}

void DoodleBug::Breed(){

int db\_add = 0;

int db\_add\_1 = db\_add;

int k;

for(int i=0; i<number\_of\_DB; i++){

if(counter\_DB[i] == 0){

int x\_coord = DoodleBug\_Coord[i][0];

int y\_coord = DoodleBug\_Coord[i][1];

int search=0;

if((x\_coord+1)>0 && (x\_coord+1) < size && Grid[x\_coord+1][y\_coord] =='.' && search == 0){

Grid[x\_coord+1][y\_coord] = 'X';

db\_add++;

search++;}

else if((x\_coord-1)>0 && (x\_coord-1) < size && Grid[x\_coord-1][y\_coord] =='.' && search == 0){

Grid[x\_coord-1][y\_coord] = 'X';

db\_add++;

search++;}

else if((y\_coord+1)>0 && (y\_coord+1) < size && Grid[x\_coord][y\_coord + 1] =='.' && search == 0){

Grid[x\_coord][y\_coord + 1] = 'X';

db\_add++;

search++;}

else if((y\_coord-1)>0 && (y\_coord-1) < size && Grid[x\_coord][y\_coord - 1] =='.' && search == 0){

Grid[x\_coord][y\_coord - 1] = 'X';

db\_add++;

search++;}

}//if statement

}//for statement

k = 0;

for(int i=0; i<size; i++){

for(int j=0; j<size;j++){

if(Grid[i][j] == 'X'){

k++;

}

}

}

number\_of\_DB = k;

delete[] counter\_starve;

counter\_starve = new int[number\_of\_DB ];

for(int i=0; i<number\_of\_DB ; i++){

counter\_starve[i] = 0;

}

}

void DoodleBug::starve(){

for(int i=0; i<number\_of\_DB; i++){

if(counter\_starve[i] == 2){

int x\_coord = DoodleBug\_Coord[i][0];

int y\_coord = DoodleBug\_Coord[i][1];

MarkDBGrid(x\_coord, y\_coord);

DB\_Coord[i] = 1;

} //if statement

} //for statement

}

void DoodleBug :: make\_new\_coord(){

int count = 0;

for(int i = 0; i<number\_of\_DB; i++){

if(counter\_starve[i] == 2){

std::cout<<"The dead bug is in location: "<< i <<std::endl;

count+=1;

}

}

int counter[count];

int k = 0;

for(int i = 0; i<number\_of\_DB; i++){

if(counter\_starve[i] == 2){

counter[k] = i;

k++;

}

}

for(int m=0; m<count; m++){

if(counter[m] > 0 && counter[m] < number\_of\_DB){ //if the number is in the middle

DoodleBug\_Coord\_1 = new int\*[number\_of\_DB-1];

for(int k=0; k<number\_of\_DB-1; k++){

DoodleBug\_Coord\_1[k] = new int[2];

}

for(int k=0; k<counter[m]; k++){

DoodleBug\_Coord\_1[k] = DoodleBug\_Coord[k];

}

for(int k=counter[m]+1; k<number\_of\_DB; k++){

DoodleBug\_Coord\_1[k-1] = DoodleBug\_Coord[k];

}

DoodleBug\_Coord = new int\*[number\_of\_DB-1];

for(int k=0; k<number\_of\_DB-1; k++){

DoodleBug\_Coord[k] = new int[2];

}

for(int k=0; k<number\_of\_DB-1; k++){

DoodleBug\_Coord[k] = DoodleBug\_Coord\_1[k];

}

delete[] DoodleBug\_Coord\_1;

number\_of\_DB -= 1;

//std::cout<<number\_of\_DB<<std::endl;

}

else if (counter[m] ==0){ //When the location is at the front

DoodleBug\_Coord\_1 = new int\*[number\_of\_DB-1];

for(int k=0; k<number\_of\_DB-1; k++){

DoodleBug\_Coord\_1[k] = new int[2];

}

for(int k=1; k<counter[m]; k++){

DoodleBug\_Coord\_1[k-1] = DoodleBug\_Coord[k];

}

DoodleBug\_Coord = new int\*[number\_of\_DB-1];

for(int k=0; k<number\_of\_DB-1; k++){

DoodleBug\_Coord[k] = new int[2];

}

for(int k=0; k<number\_of\_DB-1; k++){

DoodleBug\_Coord[k] = DoodleBug\_Coord\_1[k];

}

//delete[] DoodleBug\_Coord\_1;

number\_of\_DB -= 1;

}

else if(counter[m] == number\_of\_DB){ // if the number is at the end

DoodleBug\_Coord\_1 = new int\*[number\_of\_DB-1];

for(int k=0; k<number\_of\_DB-1; k++){

DoodleBug\_Coord\_1[k] = new int[2];

}

for(int k=0; k<counter[m] ; k++){

DoodleBug\_Coord\_1[k] = DoodleBug\_Coord[k];

}

DoodleBug\_Coord = new int\*[number\_of\_DB-1];

for(int k=0; k<number\_of\_DB-1; k++){

DoodleBug\_Coord[k] = new int[2];

}

for(int k=0; k<number\_of\_DB-1; k++){

DoodleBug\_Coord[k] = DoodleBug\_Coord\_1[k];

}

//delete[] DoodleBug\_Coord\_1;

number\_of\_DB -= 1;

}

}

}

DoodleBug :: ~DoodleBug(){

std::cout << "Initiating Ants (Derived) Destructor ... " << std::endl;

delete[] \*DoodleBug\_Coord;

delete[] ant\_ate;

delete[] counter\_DB;

delete[] counter\_starve;

}

Main.cpp

#include "Predator.h"

#include "Ants.h"

#include "DoodleBug.h"

int main(){

char ans;

//Define the Grid

Ants A;

DoodleBug DB;

//Create Predator Pointer

Predator \*P\_1;

Predator \*P\_2;

P\_1 = &A;

P\_2 = &DB;

P\_1->setgrid(20); //Initialize the grid

P\_2->setgrid(20); //Initialize the grid

A.setcoord(100,5); //Set the number of Ants coordinates

int size = A.get\_size();

char Grid[size][size]; //Set a pass on grid

//Pass Grid from Ants to DoodleBug

for(int i=0; i<size; i++){

for(int j=0; j<size; j++){

Grid[i][j] = A.get\_grid(i,j);

DB.intakegrid(Grid[i][j],i,j);

}

}

//Set DoodleBug coordinates

DB.setcoord(5);

//Displate the combined grid

std::cout<< "The original grid with 100 ants and 5 doodlebugs is: " << std::endl;

P\_2->DisplayGrid();

do{

P\_2 -> move();

DB.starve();

DB.make\_new\_coord();

P\_2 -> Breed();

//Transfer everything to A

for(int i=0; i<size; i++){

for(int j=0; j<size; j++){

Grid[i][j] = DB.get\_grid(i,j);

A.intakegrid(Grid[i][j],i,j);

}

}

P\_1 -> move();

P\_1 -> Breed();

std::cout<< "After the move, the grid becomes: " << std::endl;

P\_1->DisplayGrid();

//Transfer everything to DB

for(int i=0; i<size; i++){

for(int j=0; j<size; j++){

Grid[i][j] = P\_1->get\_grid(i,j);

DB.intakegrid(Grid[i][j],i,j);

}

}

std::cout<< "Do you want to continue? (y/n) " << std::endl;

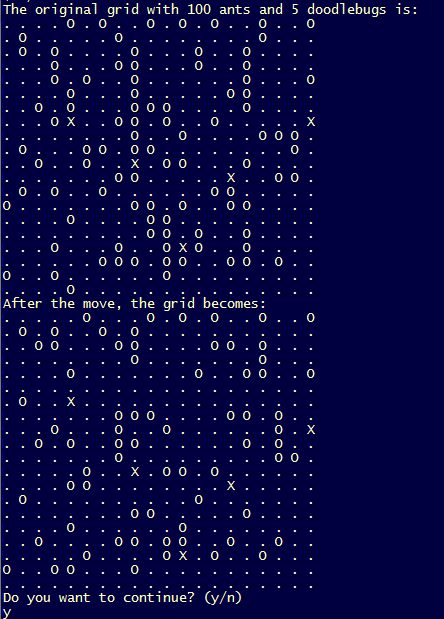
std::cin >> ans;

}while(ans=='Y' || ans =='y');

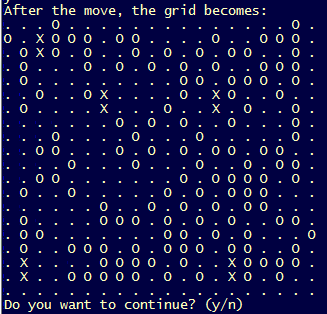
}

**Output**

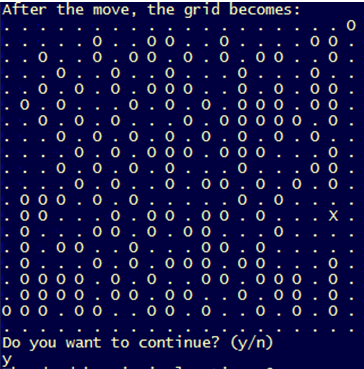
First Step and Second Step:



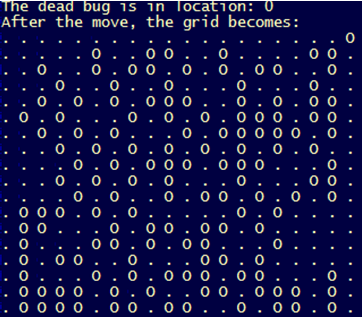
After 8 rounds



Second Last Step



Last Step:

****

**Exstinction of DoodleBug**

Compile\_All.txt:

g++ -c Main.cpp && g++ -c Predator.cpp && g++ -c Ants.cpp && g++ -c DoodleBug.cpp && g++ -o Predator Predator.o Main.o Ants.o DoodleBug.o

**Shape Question:**

Shape.h

#include <string>

#include <iostream>

class Shape{

protected:

double height;

double width;

double base;

double radius;

double side;

public:

virtual double Area() = 0;

};

class Square : public Shape{

public:

void set\_side(double a){

side = a;

};

double Area(){

return side \* side;

};

};

class Circle : public Shape{

public:

void set\_radius(double r){

radius = r;

};

double Area(){

return 3.14159265\*radius\*radius;

};

};

class Triangle : public Shape{

public:

void set\_bh(double b, double h){

base = b;

height = h;

};

double Area(){

return (base\*height/2);

};

};

class Rectangle : public Shape{

public:

void set\_side(double h, double w){

height = h;

width = w;

};

double Area(){

return height \* width;

};

};

Shape.cpp

#include "Shape.h"

int main(){

Square Sq;

Sq.set\_side(5);

Rectangle Rect;

Rect.set\_side(4.5,4);

Triangle Tri;

Tri.set\_bh(5.5,3);

Circle Circ;

Circ.set\_radius(2.5);

Shape \*S1 = &Sq;

std::cout<<"The area of Square is: " << S1 -> Area() << std::endl;

Shape \*S2 = &Rect;

std::cout<<"The area of Rectangle is: " << S2 -> Area() << std::endl;

Shape \*S3 = &Tri;

std::cout<<"The area of Triangle is: " << S3 -> Area() << std::endl;

Shape \*S4 = &Circ;

std::cout<<"The area of Circle is: " << S4 -> Area() << std::endl;

}

**Output**

