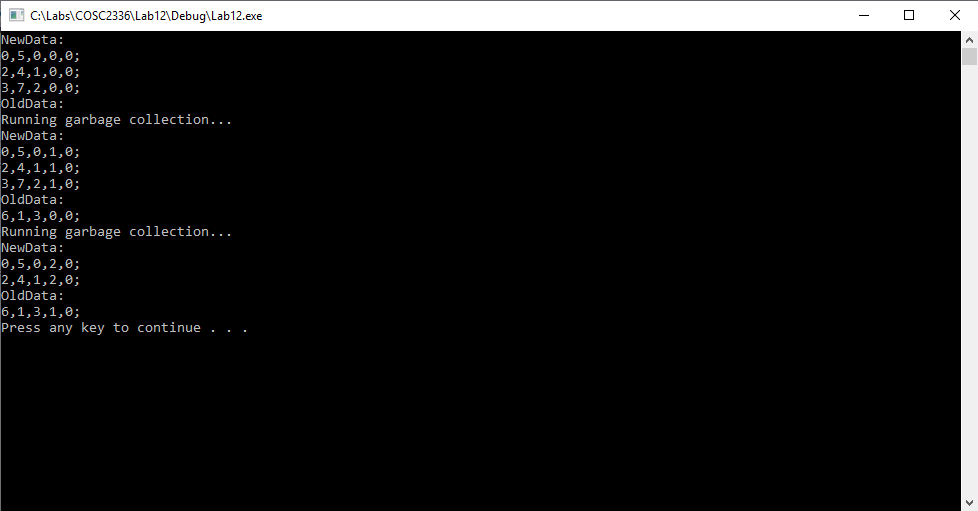
Name: William A. Brannon

Assignment: Lab Assignment Report #12

Date Due: Monday, April, 29, 2019

Class: Fundamentals of Programming III Section #1

**Program Output**



**Lab12.cpp**

// Lab 12: Memory Management

// By: William Brannon on 04/24/2019

#include "stdafx.h"

#include <iostream>

#include <vector>

using namespace std;

class CoordinatePair {

public:

int x;

int y;

int identifier;

int life; //number of GC cycles survived

bool mark;

CoordinatePair(int x1, int y1, int id) {

x = x1;

y = y1;

identifier = id;

life = 0;

mark = false;

}

void display() {

cout << x << "," << y << "," << identifier << "," << life << "," << mark << ";" << endl;

}

};

class GCmanager {

public:

vector<CoordinatePair> newData;

vector<CoordinatePair> oldData;

int counter = 0; //identifier for objects

GCmanager(int n) {

//n is the number of elements you want to allocate in each region

newData.reserve(n);

oldData.reserve(n);

}

bool hasRoom(int a) {

//a indicates which data section is being checked

//0 indicates new, 1 indicates old

switch (a)

{

case 0: // code to be executed if a = 0;

if (newData.size() < newData.capacity()) {

return true;

}

else {

return false;

}

case 1: // code to be executed if a = 1;

if (oldData.size() < oldData.capacity()) {

return true;

}

else {

return false;

}

default: // code to be executed if a doesn't match any cases

return false;

}

}

void markObj(int id) {

//marks the object for removal by its identifier

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

if (newData.at(i).identifier == id) {

newData.at(i).mark = true;

}

}

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

if (oldData.at(i).identifier == id) {

oldData.at(i).mark = true;

}

}

}

void addObj(int x, int y) {

CoordinatePair temp(x, y, counter);

if (hasRoom(0)) {

newData.push\_back(temp);

counter++;

return;

}

GC();

if (hasRoom(0)) {

newData.push\_back(temp);

counter++;

return;

}

else if (hasRoom(1)) {

oldData.push\_back(temp);

counter++;

return;

}

cerr << "Memory Overload";

}

void GC() {

cout << "Running garbage collection..." << endl;

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

if (newData.at(i).mark == true) {

newData.erase(newData.begin() + i);

i--;

}

if (newData.at(i).life > 2 && hasRoom(1)) {

CoordinatePair temp = newData.at(i);

newData.erase(newData.begin() + i);

i--;

oldData.push\_back(temp);

}

}

if (!hasRoom(1)) {

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

if (oldData.at(i).mark == true) {

oldData.erase(newData.begin() + i);

i--;

}

}

}

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

newData.at(i).life++;

}

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

oldData.at(i).life++;

}

}

void display() {

cout << "NewData: " << endl;

for (CoordinatePair temp : newData) {

temp.display();

}

cout << "OldData: " << endl;

for (CoordinatePair temp : oldData) {

temp.display();

}

}

};

int main()

{

//example events to show functionality hard-coded merely to show one potential example of a real-world runtime scenario

GCmanager tester(3);

tester.addObj(0, 5); //example of initializing a new obj

tester.addObj(2, 4);

tester.addObj(3, 7);

tester.display(); //display function for demonstrating what is going on

tester.addObj(6, 1);

tester.display();

tester.markObj(2); //marks an object for cleanup, similar to delete

tester.GC(); //runs GC (not necessary for it to be done manually, the code is automatic but it helps to show what is going on after the mark)

tester.display();

system("pause");

return 0;

}