***Headers.h***

#include <iostream>

#include <vector>

#include <fstream>

#include <istream>

#include <sstream>

#include<iterator>

#include <algorithm>

#include <map>

#include <set>

#include "FlowerBed.h"

#include "GreenHouse.h"

***Main.cpp***

#include "headers.h"  
  
int main() {  
 GreenHouse green\_house;  
 vector<FlowerBed> data;  
 *//1* data=green\_house.ReadData("input.txt");  
 *//2* green\_house.SortByShape(data);  
 string filename="output.txt";  
 green\_house.PrintData(data,filename);  
 *//3* multimap<Shape,FlowerBed> data\_map=green\_house.MoveToContainer(data);  
 *//4.2* green\_house.PrintFlowersByNumber(data,filename,2);  
 *//4.3* green\_house.PrintVarietyOfFlowers(data,filename);  
 *//5.1* vector<FlowerBed> max\_flowers=green\_house.FbedWithMaxFlowers(data);   
 green\_house.PrintData(max\_flowers,filename);  
 *//5.3* vector<FlowerBed> the\_same\_flowers=green\_house.TheSameFlowers(data);  
 green\_house.PrintData(the\_same\_flowers,filename);  
 *//6.1* int flowers\_in\_circle\_fbed=green\_house.FlowersInCircleFbed(data\_map);  
 ofstream out (filename,ios::app);  
 out<<"\nAmount of flowers in circle flower bed: "<<flowers\_in\_circle\_fbed<<"\n";  
 *//6.3* vector<FlowerBed> fbeds\_with\_amount=green\_house.FbedsWithAmountOfFlowers(data,2);  
 out<<"\nFlower beds with 2 flowers:"<< "\n";  
 green\_house.PrintData(fbeds\_with\_amount,filename);  
 out<<"\n";  
 *//6.4* vector<FlowerBed> by\_shape=green\_house.SearchByShape(data,*CIRCLE*);  
 out<<"Flower beds with circle shape:\n";  
 green\_house.PrintData(by\_shape,filename);  
 return 0;  
}

***FlowerBed.h***

#include "headers.h"  
using namespace std;  
enum Shape  
{  
 *CIRCLE*,  
 *SQUARE*,  
 *RHOMBUS*};  
class FlowerBed{  
 private:  
 int bed\_number\_;  
 Shape shape\_;  
 vector<string> flowers\_;  
 public:  
 FlowerBed()=default;  
 ~FlowerBed()=default;  
 void SetNumber(int number){bed\_number\_=number;};  
 void SetShape(Shape shape){shape\_=shape;};  
 void SetFlowers(const std::vector<std::string>& flowers){flowers\_=flowers;};  
 int GetNumber() const {return bed\_number\_;}  
 Shape GetShape() const {return shape\_;}  
 vector<string> GetFlowers() const {return flowers\_;}  
 static Shape GetShapeByString(const std::string& shapeString);  
  
};  
std::istream& operator>>(istream& in, FlowerBed&);  
std::ostream& operator<<(std::ostream&, const FlowerBed&);

***FlowerBed.cpp***

#include "headers.h"  
std::istream& operator>>(istream& in, FlowerBed& flower\_bed)  
{  
 int number;  
 in >> number;  
 flower\_bed.SetNumber(number);  
 std::string shapeString;  
 in >> shapeString;  
 flower\_bed.SetShape(FlowerBed::GetShapeByString(shapeString));  
 std::string flower;  
 std::vector<std::string> flowers;  
 while (in >> flower)  
 {  
 flowers.push\_back(flower);  
 }  
 flower\_bed.SetFlowers(flowers);  
 return in;  
}  
  
std::ostream& operator<<(std::ostream& out, const FlowerBed& flower\_bed)  
{  
 out << "Flower bed number: "<< flower\_bed.GetNumber()  
 << " Shape: " << flower\_bed.GetShape()  
 << " Flowers:";  
 vector<string> flowers=flower\_bed.GetFlowers();  
 for(const string &flower : flowers){  
 out<<" "<<flower;  
 }  
 return out;  
}  
  
Shape FlowerBed::GetShapeByString(const std::string &shapeString)  
{  
 if (shapeString == "круг")  
 {  
 return *CIRCLE*;  
 }  
 else if (shapeString == "квадрат")  
 {  
 return *SQUARE*;  
 }  
 else if (shapeString == "ромб")  
 {  
 return *RHOMBUS*;  
 }  
 return *CIRCLE*;  
}

***GreenHouse.h***

#include "headers.h"  
  
class GreenHouse{  
 public:  
 vector<FlowerBed> ReadData(string filename);*//Разместить данные в последовательном контейнере (1).* void PrintData(vector<FlowerBed>,string filename);  
  
 struct CompareByShape {  
 bool operator()(const FlowerBed& a, const FlowerBed& b) {  
 return a.GetShape() < b.GetShape();  
 }  
 };  
 void SortByShape(vector<FlowerBed>); *//Выполнить сортировку по геометрической форме клумбы (2).* multimap<Shape,FlowerBed> MoveToContainer(vector<FlowerBed>); *//Далее данные из (1) размещаются в ассоциативный контейнер. Использовать как ключ (3)(MULTIMAP).* void PrintFlowersByNumber(vector<FlowerBed>,string filename,int bed\_number);*//4.2) вывести список всех цветов указанной клумбы* void PrintVarietyOfFlowers(vector<FlowerBed>,string filename); *//4.3) вывести список всех видов цветов;* vector<FlowerBed> FbedWithMaxFlowers(vector<FlowerBed>); *//5.1) найти клумбу с наибольшим количеством цветов; (я вывожу КЛУМБЫ (а не клумбу), если несколько клумб имеют одинаковое(макс) кол-во цветов));* vector<FlowerBed> TheSameFlowers(vector<FlowerBed>); *//5.3) найти клумбу с только одним видом цветов; (как и в 5.1 вывожу КЛУМБЫ)* static bool CompareWithShape(FlowerBed flower\_bed){  
 Shape shape=*CIRCLE*;  
 return(flower\_bed.GetShape()==shape);  
 }  
 int FlowersInCircleFbed(const multimap<Shape,FlowerBed>&); *//6.1) подсчет числа цветков на всех круглых клумбах (с функцией);* vector<FlowerBed> FbedsWithAmountOfFlowers(vector<FlowerBed>,int amount); *//6.3) поиск клумб с заданным числом цветов (функц объект);* class CompareWithAmount{  
 private:  
 int amount\_;  
 public:  
 CompareWithAmount(int amount):amount\_(amount){};  
 bool operator() (const FlowerBed& element){  
 return(element.GetFlowers().size()==amount\_);  
 }  
 };  
  
 vector<FlowerBed> SearchByShape(vector<FlowerBed>,Shape shape); *//6.4) поиск всех клумб указанной формы*};

***GreenHouse.cpp***

#include "headers.h"  
vector<FlowerBed> GreenHouse::ReadData(string filename) {  
 vector<FlowerBed> data;  
 std::ifstream fin(filename);  
 if (!fin.is\_open()) {  
 std::cout << "Error!\n";  
 return data;  
 }  
 string string\_data;  
 FlowerBed flower\_bed;  
 while (!fin.eof()) {  
 getline(fin,string\_data);  
 istringstream string\_stream(string\_data);  
 string\_stream >> flower\_bed;  
 data.push\_back(flower\_bed);  
 }  
 return data;  
}  
  
void GreenHouse::PrintData(vector<FlowerBed> data,string filename) {  
 std::ofstream out (filename);  
 std::copy(data.begin(),data.end(),std::ostream\_iterator<FlowerBed>(out,"\n"));  
}  
  
void GreenHouse::SortByShape(vector<FlowerBed> data) {  
 sort(data.begin(),data.end(),CompareByShape());  
}  
  
  
multimap<Shape,FlowerBed> GreenHouse::MoveToContainer(vector<FlowerBed> data) {  
 multimap<Shape, FlowerBed> container;  
 for (const FlowerBed& flower\_bed : data) {  
 container.insert(make\_pair(flower\_bed.GetShape(),flower\_bed));  
 }  
 return container;  
}  
  
void GreenHouse::PrintVarietyOfFlowers(vector<FlowerBed> data,string filename) {  
 ofstream out(filename,ios::app);  
 set<string> flowers;  
 for(int i=0;i<data.size();++i){  
 vector<string> one\_fbed\_flowers=data[i].GetFlowers();  
 for(int j=0;j<one\_fbed\_flowers.size();++j){  
 flowers.insert(one\_fbed\_flowers[j]);  
 }  
 }  
 out<<"Variety of flowers: ";  
 std::copy(flowers.begin(),flowers.end(),std::ostream\_iterator<string>(out," "));  
 out<<"\n";  
}  
  
void GreenHouse::PrintFlowersByNumber(vector<FlowerBed> data,string filename,int bed\_number) {  
 ofstream out(filename,ios::app);  
 auto iterator=std::find\_if(data.begin(),data.end(),  
 [bed\_number](const FlowerBed& flower\_bed){return bed\_number==flower\_bed.GetNumber();});  
 out<<"Flowers in the flower bed number "<<bed\_number<<": ";  
 vector<string> flowers=iterator->GetFlowers();  
 std::copy(flowers.begin(),flowers.end(),std::ostream\_iterator<string>(out," "));  
 out<<"\n";  
}  
  
vector<FlowerBed> GreenHouse::FbedWithMaxFlowers(vector<FlowerBed> data) {  
 vector<FlowerBed> result;  
 multimap<int, FlowerBed> amount\_of\_flowers;  
 for (int i = 0; i < data.size(); i++) {  
 amount\_of\_flowers.insert(make\_pair(data[i].GetFlowers().size(), data[i]));  
 }  
 int max\_size = amount\_of\_flowers.rbegin()->first;  
 pair range = amount\_of\_flowers.equal\_range(max\_size);  
 for (auto i = range.first; i != range.second; ++i) {  
 result.push\_back(i->second);  
 }  
 return result;  
}  
  
vector<FlowerBed> GreenHouse::TheSameFlowers(vector<FlowerBed> data) {  
 vector<FlowerBed> the\_same\_flowers;  
 for (int i = 0; i < data.size(); ++i) {  
 vector<string> flowers = data[i].GetFlowers();  
 if (count(flowers.begin(), flowers.end(), flowers[0]) == flowers.size()) {  
 the\_same\_flowers.push\_back(data[i]);  
 }  
 }  
 return the\_same\_flowers;  
}  
  
int CalculateFlowers(int sum, const std::pair<Shape,FlowerBed>& flowerbed\_pair){  
 if(flowerbed\_pair.first==*CIRCLE*){  
 return sum+flowerbed\_pair.second.GetFlowers().size();  
 }  
 return sum;  
}  
  
int GreenHouse::FlowersInCircleFbed(const multimap<Shape,FlowerBed>& flower\_bed\_map) {  
 return std::accumulate(flower\_bed\_map.begin(), flower\_bed\_map.end(), 0, CalculateFlowers);}  
  
vector<FlowerBed> GreenHouse::FbedsWithAmountOfFlowers(vector<FlowerBed> data,int amount) {  
 vector<FlowerBed> fbeds\_with\_amount;  
 CompareWithAmount compare(amount);  
 auto iterator=find\_if(data.begin(),data.end(),compare);  
 if(iterator==data.end()){  
 return fbeds\_with\_amount;  
 }  
 while(iterator!=data.end()){  
 fbeds\_with\_amount.push\_back(\*iterator);  
 iterator=find\_if(iterator+1,data.end(),compare);  
 }  
 return fbeds\_with\_amount;  
}  
  
vector<FlowerBed> GreenHouse::SearchByShape(vector<FlowerBed> data,Shape shape) {  
 vector<FlowerBed> by\_shape;  
 auto iterator=find\_if(data.begin(),data.end(),[shape](FlowerBed flower\_bed){return flower\_bed.GetShape()==shape;});  
 if(iterator==data.end()){  
 return by\_shape;  
 }  
 while(iterator!=data.end()){  
 by\_shape.push\_back(\*iterator);  
 iterator=find\_if(iterator+1,data.end(),[shape](FlowerBed flower\_bed){return flower\_bed.GetShape()==shape;});  
 }  
 return by\_shape;  
}

***Result:***

