Problem Statement: We want to create a vector to add to our original smart home system. This will allow us to add more than the set number of lights, blinds, and rooms. We want to create a house to put these attributes in. The house will have different rooms with unique number of blinds and lights.

|  |  |
| --- | --- |
| **Class Design:** | **Method Description: Comments in some of code** |
| **Vec:** |  |
| \_init(); void | //resets all attributes |
| \_clear(); void | //clears the vector |
| \_create(); void | //creates an empty or null vector |
| \_resize(int); int | //resizes the vector when adding or removing |
| \_addValue(); int | //adds value to vector |
| removeValue(); template A | //removes value in vector |
| getSize(); int | //returns size of vector |
| isEmpty(); bool | //checks to see if vector is empty |
| isFull(); bool | //checks to see if vector is full and needs to resize |
|  |  |
| **House:** |  |
| addRoom(); void | //adds a room to the vector |
| menu\_home(); void | //menu method |
| getHouseID(); int | //returns the id of a house |
| getHouseName(); string | //returns the name of a house |
| getHouseAdd1(); string | //returns the address 1 of house |
| getHouseAdd2(); string | //returns the address 2 of house |
| getNextRoomID(); int | //returns the id of the next room in vector |
| getNextLightID(); int | //returns the id of the next light in vector |
| getNextBlindID(); int | //returns the id of the next blind in vector |
|  |  |
| **Room:** |  |
| room\_menu(); void | //room menu method |
| addLight(); void | //adds light to vector |
| addBlind(); void | //adds blind to vector |
| setRoomID(): void | //sets the room ID value |
| setRoomName(); void | //sets the name of the room |
| setRoomLocation(); void | //sets the location of a room in the house |
| setLightState(); void | //sets the state of a light |
| setNumLights(); void | //sets the number of lights |
| setNumBlinds(); void | //sets the number of blinds |
| setBlind(); void | //sets the status of a blind |
| getRoomID(); int | //returns the room id |
| getRoomName(); string | //returns the name of a room |
| getRoomLocation(); string | //returns the location of a room |
| getBlindID(); int | //returns the id of a blind |
| getBlindLocation(); string | //returns the location of a blind |
| getOCState(); string | //returns the open close state of blind |
| getRLState(); string | //returns the raise lower state of blind |
| getNumBlinds(); int | //returns the number of blinds in vector |
| getLightID(); int | //returns the id of a light |
| getLigthName(); string | //returns the name of a light |
| getLightState(); string | //returns the state of a light(ON OFF) |
| getNumLights(); int | //returns the number of lights in a vector |
|  |  |
| **Blind:** |  |
| setID(); void | //sets the id of blind |
| setName(); void | //sets the name of blind |
| setBlind(); void | //sets the state of the blind |
| getID(); int | //returns the if of blind |
| getName(); string | //returns the name of blind |
| blindOpenState(); bool | //sets the OC state of blind |
| blindRaiseState(); bool | //sets the RL state of blind |
|  |  |
| **Light:** |  |
| setID(); void | //sets the id of light |
| setName(); void | //sets the name of light |
| setState(); void | //sets the state of light |
| getID(); int | //returns the id of light |
| getState(); bool | //returns the state of light |
| getName(); string | //returns the name of light |
|  |  |
| ostream& | //all print overloading methods |
| vec(); house(); light(); blind(); | //all constructors that initialize data |

Graphical user interface, text, application

Description automatically generated

Vec.h:

#include "hw.h"

using namespace std;

template<class A>

class Vec {

private:

A \*\_data;

int \_size;

int \_max;

void \_init();

void \_clear();

void \_create();

int \_resize(int);

public:

Vec(); // default constructor

A& at(int);

int addValue(A);

A removeValue();

int getSize() { return \_size; }

bool isEmpty();

bool isFull();

ostream& print(ostream&);

};

template<class A>

A& Vec<A>::at(int i){ // .at() implementation

return \_data[i];

}

template<class A>

Vec<A>::Vec() {

\_init();

}

template<class A>

void Vec<A>::\_init() {

\_data = NULL;

\_size = 0;

\_max = \_size;

}

template<class A>

void Vec<A>::\_clear() {

if (!isEmpty()) {

delete[] \_data;

}

\_init();

}

template<class A>

void Vec<A>::\_create() {

\_clear();

\_max = 1;

\_data = new A[\_max];

}

template<class A>

int Vec<A>::\_resize(int inc) {

if (isEmpty()) {

\_create();

} else {

\_max = \_max + inc;

A \*newData = new A[\_max];

int numVals = \_size;

if (inc<0) numVals = numVals + inc;

\_size = numVals;

for(int i=0;i<numVals;i++) newData[i] = \_data[i]; // \*(newData+i) = \*(\_data+i

delete[] \_data;

\_data = newData;

}

return \_max;

}

template<class A>

bool Vec<A>::isEmpty() {

if (\_size==0) return true;

else return false;

}

template<class A>

bool Vec<A>::isFull() {

if (\_size==\_max) return true;

else return false;

}

template<class A>

int Vec<A>::addValue(A val) {

if (\_data==NULL) {

\_create();

\_data[\_size] = val;

\_size++;

} else if (isFull()) {

\_max = \_resize(1);

\_data[\_size] = val;

\_size++;

} else {

\_data[\_size] = val;

\_size++;

}

return \_max;

}

template<class A>

A Vec<A>::removeValue() {

A val;

if (!isEmpty()) {

val = \_data[\_size-1];

\_max = \_resize(-1);

}

return val;

}

template<class A>

ostream& Vec<A>::print(ostream &out) {

if (\_data==NULL) out << "Vector not created\n";

else if (isEmpty()) out << "Vector is empty\n";

else {

for(int i=0;i<\_size;i++) {

out << "data[" << i << "] = " << \_data[i] << endl;

}

}

return out;

}

Light.h:

#ifndef LIGHT\_H\_

#define LIGHT\_H\_

#include "vec.h"

using namespace std;

class light {

private:

int id;

string name;

bool state;

public:

//constructor

light();

light(int, string);

//setters

void setId(int);

void setName(string);

void setState(bool);

//getters

int getId();

bool getState();

string getName();

};

#endif

Light.cpp:

#include "light.h"

//constructor

light::light() {

state = false;

}

light::light(int iD, string n){

id=iD;

name=n;

state=false;

}

//setter methods for light

void light::setId(int iD){

id=iD;

}

void light::setName(string n){

name=n;

}

void light::setState(bool s){

state=s;

}

//getters

int light::getId(){

return id;

}

string light::getName(){

return name;

}

bool light::getState(){

if (state==true){return true;}

else return false;

}

Blinds.h:

#ifndef BLIND\_H

#define BLIND\_H

#include "light.h"

using namespace std;

class blind {

private:

int id;

string name;

bool ocState;

bool rlState;

public:

int open;

int close;

int raise;

int lower;

//construct

blind();

blind(int,string);

//setters

void setId(int);

void setName(string);

void setBlind(char);

//getters

int getId();

string getName();

bool blindOpenState();

bool blindraiseState();

};

#endif

Blinds.cpp:

#include "blinds.h"

//contruct

blind::blind() {

id=0;

name = " ";

open = 0;

close = 1;

raise = 0;

lower = 1;

ocState = false;

rlState = false;

}

blind::blind(int i, string n){

id=i;

name=n;

open = 0;

close = 1;

raise = 0;

lower =1;

ocState = false;

rlState = false;

}

//setters

void blind::setId(int i){

id=i;

}

void blind::setName(string l){

name=l;

}

void blind::setBlind(char x) {

if (x=='o'){

open=1;

ocState=true;

close =0;

}

else if (x=='c'){

close=1;

open=0;

ocState=false;

raise=0;

lower=1;

rlState=false;

}

else if (u=='r'){

ocState=true;

raise=1;

lower=0;

rlState=true;

open=1;

close=0;

}

else if (x=='l'){

lower=1;

rlState=false;

raise=0;

}

}

//getters

int blind::getId() {

return id;

}

string blind::getName() {

return name;

}

bool blind::blindOpenState() {

return ocState;

}

bool blind::blindraiseState() {

return rlState;

}

Room.h:

#ifndef ROOM\_H\_

#define ROOM\_H\_

#include "blinds.h"

using namespace std;

class room {

private:

int id; //room id

string name; //room name

string location; //room location

int Numlights;

Vec<light> lights; //vector of lights

int NumBlinds; //number of blinds

Vec<blind> blinds; //vector of blinds

public:

room();//contructor

room(int&, string, string);//constructor with parameters

void room\_menu(int&,int&);//menu

void addLight(int&);//light

void addBlind(int&);//blind

//setters

//rooms

void setRoomID(int);

void setRoomName(string);

void setRoomLocation(string);

//lights

void setLightState(int,bool);

void setNumLights(int);

//blinds

void setNumBlinds(int);

void setBlind(int,char);

//getters

//room

int getRoomID();

string getRoomname();

string getRoomlocation();

//blinds

int getBlindID(int);

string getBlindLocation(int);

string getOCState(int);

string getRLState(int);

int getNumBlinds();

//lights

int getLightId(int);

string getLightName(int);

string getLightState(int);

int getNumLights();

ostream& lightoverload(ostream&);//print

ostream& blindoverload(ostream&);//print

ostream& printRoom(ostream&);//print

};

#endif

Room.cpp:

#include "room.h"

room::room() {

id = -1;

name="";

location="";

Numlights=-1;

NumBlinds=-1;

}

room::room(int& nextRoom\_ID, string n, string l){

id=nextRoom\_ID;

name=n;

location=l;

Numlights=0;

NumBlinds=0;

nextRoom\_ID++;

}

void room::room\_menu(int& next\_light\_id, int& next\_blind\_id){ // room menu implementation

int topmenu=0;

while(topmenu!=98){

cout<<"\nRoom Menu:\n"<<"\n";

cout<<"Room: "<<id<<", "<<name<<", "<<location<<"\n\n";

cout<<"1. Show status all\n";

cout<<"2. On.Off Light\n";

cout<<"3. Open/Close Blind\n";

cout<<"4. Raise/Lower Blind\n";

cout<<"5. Add Light\n";

cout<<"6. Add Blind\n";

cout<<"98. Return to Top Menu\n";

cout<<"99. Exit\n\n";

cout<<"Enter a number: ";

int input;

cin >> input;

if(cin.fail()){

cin.clear();

cin.ignore(99,'\n');

cout<<"\nINVALID SELECTION\n";

}

else if(input==99){

abort();

}

else if(input==98){

topmenu=98;

}

else if(input==97){

if(!blinds.isEmpty()){

blinds.removeValue();

NumBlinds--;

next\_blind\_id--;

}

}

else if(input==96){

if(!lights.isEmpty()){

lights.removeValue();

Numlights--;

next\_light\_id--;

}

}

else if(input==1){

cout<<"\n";

printRoom(cout);

}

else if(input==2){

if(lights.isEmpty()){

cout<<"There are no lights yet.";

}

else{

cout<<"\n List of lights: ";

lightoverload(cout);

cout<<"Which light would you like to toggle? ";

cin>>input;

if(cin.fail()){

cin.clear();

cin.ignore(99,'\n');

cout<<"\nINVALID SELECTION\n";

}

else{

cout<<"\n";

lights.at(input-1).setState(!lights.at(input-1).getState());

}

}

}

else if(input==3){

if(blinds.isEmpty()){

cout<<"There are no blinds yet.";

}

else{

cout<<"\n List of blinds: ";

blindoverload(cout);

cout<<"\n Which blind would you like to toggle?";

cin>>input;

if(cin.fail()){

cin.clear();

cin.ignore(99,'\n');

cout<<"\nINVALID SELECTION\n";

}

else{

cout<<"\n";

if(!blinds.at(input-1).blindOpenState()){

blinds.at(input-1).setBlind('o');

}

else if(blinds.at(input-1).blindOpenState()){

blinds.at(input-1).setBlind('c');

}

}

}

}

else if(input==4){

if(blinds.isEmpty()){

cout<<"There are no blinds yet.";

}

else{

cout<<"List of blinds: ";

blindoverload(cout);

cout<<"Which blind would you like to toggle? ";

cin>>input;

if(cin.fail()){

cin.clear();

cin.ignore(99,'\n');

cout<<"INVALID SELECTION";

}

else{

cout<<"\n";

if(!blinds.at(input-1).blindraiseState()){

blinds.at(input-1).setBlind('r');

}

else if(blinds.at(input-1).blindraiseState()){

blinds.at(input-1).setBlind('l');

}

}

}

}

else if(input==5){

addLight(next\_light\_id);

} else if(input==6){addBlind(next\_blind\_id);}

}

}

void room::addLight(int& next\_light\_id){

Numlights++;

string ln;

cout<<"Enter light name "<<next\_light\_id<<": ";

getline(cin >> ws, ln);

light temp;

temp=light(next\_light\_id,ln);

lights.addValue(temp);

next\_light\_id++;

}

void room::addBlind(int& next\_blind\_id){

NumBlinds++;

string bn;

cout<<"Enter name of blind number "<<next\_blind\_id<<": ";

getline(cin >> ws, bn);

blind temp;

temp =blind(next\_blind\_id,bn);

blinds.addValue(temp);

next\_blind\_id++;}

void room::setRoomID(int i){id = i;}

void room::setRoomName(string n){name=n;}

void room::setRoomLocation(string l) {location=l;}

void room::setLightState(int i, bool s){

if (i<1||i>Numlights){cout<<"Light Doesnt exist"<<endl;}

else lights.at(i).setState(s);}

void room::setNumLights(int i){

Numlights=i;}

void room::setNumBlinds (int i){

NumBlinds=i;}

void room::setBlind (int i, char s){

if (i<1||i>NumBlinds){cout<<"This blind does not exist"<<endl;}

else blinds.at(i).setBlind(s);}

// all getter methods

//methods for room

int room::getRoomID(){

return id;

}

string room::getRoomname(){

return name;

}

string room::getRoomlocation(){

return location;

}

// all functions for lights

int room::getLightId(int i){ //light id

if (i<1||i>Numlights){cout<<"This light does not exist"<<endl;return -1;}

else return lights.at(i).getId();}

string room::getLightName(int i){ //name of light

if (i<1||i>Numlights){return "This light does not exist";}

else return lights.at(i).getName();}

string room::getLightState(int i){ //state of light

if (i<1||i>Numlights){return "This light does not exist";}

else if (lights.at(i).getState()==false){return "OFF";}

else return "ON";}

int room::getNumLights(){

return Numlights;}

int room::getBlindID(int i){

if (i<1||i>NumBlinds){

cout<<"This blind does not exist\n";

return -1;}

else return blinds.at(i).getId();

}

string room::getBlindLocation(int i){ // checks blind location

if (i<1||i>NumBlinds){

return "This blind does not exist\n";

}

else

return blinds.at(i).getName();

}

string room::getOCState(int i){ // checks blind open/ closed

if (i<1||i>NumBlinds){

return "This light does not exist";

}

else if (blinds.at(i).blindOpenState()==false){

return "Closed";

}

else

return "Open";

}

string room::getRLState(int i){ // checks blind raised/ lowered

if (i<1||i>NumBlinds){

return "This light does not exist";

}

else

if (blinds.at(i).blindraiseState()==false){

return "Lowered";

}

else

return "Raised";

}

int room::getNumBlinds(){

return NumBlinds;

}

//print

ostream& room::lightoverload(ostream &out){

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

out<<i+1<<". "<<lights.at(i).getName()<<" State: "<<lights.at(i).getState();

}

out<<"\n";

return out;

}

ostream& room::blindoverload(ostream &out){

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

out<<i+1<<". "<<blinds.at(i).getName()<<" Open state: "<<blinds.at(i).blindOpenState()<<" Raised state: "<<blinds.at(i).blindraiseState();

}

out<<"\n";

return out;

}

ostream& room::printRoom(ostream &out){

out<<""<< name <<" Id: "<<id<<" Location: "<<location;

if(lights.isEmpty()){

out<<"THERE ARE NO LIGHTS IN THIS ROOM.";

// return out;

}

else if(!lights.isEmpty()){

out<<"Lights: ";

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

out<<lights.at(i).getName()<<" Id: "<<lights.at(i).getId()<<" State: "<<lights.at(i).getState();

}

out<<"\n";

// return out;

}

if(blinds.isEmpty()){

out<<"THERE ARE NO BLINDS IN THIS ROOM.\n";

// return out;

}

else if(!blinds.isEmpty()){

out<<"Blinds: ";

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

out<<"\n "<<blinds.at(i).getName()<<" Id: "<<blinds.at(i).getId()<<" Open State: "<<blinds.at(i).blindOpenState()<<" Raised State: "<<blinds.at(i).blindraiseState();

}

out<<"\n";

// return out;

}

return out;

}

House.h:

#ifndef HOUSE\_H\_

#define HOUSE\_H\_

#include "room.h"

using namespace std;

class house {

private:

int id; //house id

string name; //house name

string address1; //house address line 1

string address2; //house address line 2

string loginname; //loginname

string password; //password

Vec<room> rooms; // room vector

int next\_room\_id;

int next\_light\_id;

int next\_blind\_id;

public:

house();//constructor

house(int, string, string, string);//constructor with parameters

void addRoom();//add a room to house function

void menu\_home();//menu method

//getters

int getHouseID();

string getHouseName();

string getHouseAdd1();

string getHouseAdd2();

int getNextRoomID();

int getNextLightID();

int getNextBlindID();

ostream& room\_menu(ostream&);//print function

ostream& print\_house(ostream&);//print function

};

#endif

House.cpp:

#include "house.h"

//constructor: initiallizing the values for the attributes of house

house::house(){

id = -1;

name = "";

address1 = "";

address2="";

loginname="username";

password="password";

next\_room\_id =-1;

next\_light\_id =-1;

next\_blind\_id =-1;

}

//setting the inputted values for the attributes of house

house::house(int i, string n, string a1, string a2){

id = i;

name = n;

address1 = a1;

address2 = a2;

loginname="username";

password="password";

next\_room\_id = 100;

next\_light\_id = 200;

next\_blind\_id = 300;

}

//implementation of our house menu

void house::menu\_home(){

do {

cout << "Main Menu:"<<endl;

cout<<"House: "<<id<<endl;

cout<<address1<<endl;

cout<<address2<<endl;

cout<<"1. Show status all"<<endl;

cout<<"2. Room menu"<<endl;

cout<<"3. Add Room"<<endl;

cout<<"99. Exit"<<endl<<endl;

cout<<"Enter a number: "<<endl;

int input;

cin >> input;

if(cin.fail()){

cin.clear();

cout<<"INVALID SELECTION"<<endl;

}

else if(input==99){break;}

else if(input==97){

if(!rooms.isEmpty()){

rooms.removeValue();

next\_room\_id --;}

}

else if(input==1){print\_house(cout);

}

else if(input==2){

if(!rooms.isEmpty()){

cout<<"List of rooms: "<<endl;

room\_menu(cout);

cout<<"Which room would you like? ";

cin>>input;

rooms.at(input-1).room\_menu(next\_light\_id, next\_blind\_id);

}

else{

cout<<endl<<"There are no rooms created";

}

}

else if(input==3){

addRoom();

}

else{

input=0;

}

}while(true);

}

void house::addRoom(){

string n;

string l;

int room\_index = next\_room\_id -99;

//room menu add

id=next\_room\_id ;

cout<<"Enter room name: ";

getline(cin >> ws, n);

cout<<"Enter the location of "<<n<<": ";

getline(cin >> ws, l);

room temp;

temp = room(next\_room\_id ,n,l);

rooms.addValue(temp);

}

//getters

int house::getHouseID(){

return id;

}

string house::getHouseName(){

return name;

}

string house::getHouseAdd1(){

return address1;

}

string house::getHouseAdd2(){

return address2;

}

int house::getNextRoomID (){

return next\_room\_id ;

}

int house::getNextLightID(){

return next\_light\_id;

}

int house::getNextBlindID(){

return next\_blind\_id ;

}

//printer

ostream& house::room\_menu(ostream &out){

int room\_number = next\_room\_id -100;

for(int i=0;i< room\_number ;i++){

out<<i+1<<". "<<rooms.at(i).getRoomname();

}out<<"\n"; return out;

}

ostream& house::print\_house(ostream &out){

if(rooms.isEmpty()){

cout<<"\nNo more rooms created. ";

}

else{

int room\_number = next\_room\_id -100;

for(int i=0;i<room\_number;i++){

out<<"\n";

rooms.at(i).printRoom(cout);

}

}

return out;

}

ostream& operator<<(ostream &out, house &h){

h.print\_house(out);

return out;

}