**CS 31 Winter 2012 Project 7 Test Data**

There were 18 test cases, each worth 5.556 points. (The 18th case, making sure that your Valley destructor was coded correctly, was by visual inspection of your code.) To run the test cases:

1. Remove the main routine from your robots.cpp file.
2. Append the following text to the end of your robots.cpp file, and build the resulting program.
3. For any test case you wish to try, run the program, providing as input the test number.

#include <iostream>

#include <string>

#include <algorithm>

#include <cstdlib>

#include <cassert>

using namespace std;

int main()

{

cout << "Enter test case number (1-17): ";

int n;

cin >> n;

switch (n)

{

default: {

cout << "Bad argument" << endl;

} break; case 1: { // Robot ctor, accessor

Valley v(10, 10);

Robot r("Fred", &v, 3, 4, WEST);

assert(r.row() == 3 && r.col() == 4);

} break; case 2: { // Robot ctor, accessor

Valley v(10, 10);

Robot r("Fred", &v, 3, 4, WEST);

assert(r.dir() == WEST);

} break; case 3: { // Robot ctor, accessor

Valley v(10, 10);

Robot r("Fred", &v, 3, 4, WEST);

assert(r.energy() == 70);

} break; case 4: { // add too many robots

Valley v(10, 10);

string name = "@";

for (int r = 0; r < 15; r++)

{

name[0] = 'A' + r;

assert(v.addRobot(name, r/NCOLS, r%NCOLS, SOUTH));

}

name[0] = 'Z';

assert(!v.addRobot(name, 9, 9, NORTH));

} break; case 5: { // getRobotByName finds one

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

v.addRobot("Betty", 4, 4, NORTH);

v.addRobot("Chris", 5, 4, NORTH);

Robot\* rp = v.getRobotByName("Betty");

assert(rp != NULL && rp->name() == "Betty");

assert(v.getRobotByName("Betty") == rp); // doesn't create copy

} break; case 6: { // getRobotByName doesn't find mismatch

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

v.addRobot("Betty", 4, 4, NORTH);

v.addRobot("Chris", 5, 4, NORTH);

Robot\* rp = v.getRobotByName("Agnes");

assert(rp == NULL && v.getRobotByName("Abner") != NULL);

} break; case 7: { // otherRobotAt finds one

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

v.addRobot("Agnes", 3, 4, NORTH);

Robot\* rp = v.getRobotByName("Agnes");

Robot\* rp2 = v.otherRobotAt(rp);

assert(rp2 != NULL && v.otherRobotAt(rp) == rp2);

} break; case 8: { // otherRobotAt doesn't find mismatch

// Try 100 times; there's 3/4 probability to succeed

int k;

for (k = 0; k < 100; k++)

{

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

Robot\* rp = v.getRobotByName("Abner");

assert(rp != NULL);

for (int n = 0; n < 72; n++)

rp->step();

int r = rp->row()+1;

int d = NORTH;

if (rp->row() == 9)

{

r = 8;

d = SOUTH;

}

v.addRobot("Betty", r, rp->col(), d);

Robot\* rp2 = v.getRobotByName("Betty");

assert(rp2 != NULL);

if (v.otherRobotAt(rp2) == NULL)

break;

rp2->step();

if (rp2->row() != rp->row() || rp2->col() != rp->col())

continue;

if (v.otherRobotAt(rp2) != NULL)

break;

}

assert(k < 100);

} break; case 9: { // otherRobotAt doesn't find self

// Try 100 times; there's 3/4 probability to succeed

int k;

for (int k = 0; k < 100; k++)

{

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

Robot\* rp = v.getRobotByName("Abner");

assert(rp != NULL);

for (int n = 0; n < 72; n++)

rp->step();

if (v.otherRobotAt(rp) == NULL)

break;

int r = rp->row()+1;

int d = NORTH;

if (rp->row() == 9)

{

r = 8;

d = SOUTH;

}

v.addRobot("Betty", r, rp->col(), d);

Robot\* rp2 = v.getRobotByName("Betty");

v.addRobot("Chris", 5, 5, d);

Robot\* rp3 = v.getRobotByName("Chris");

assert(rp2 != NULL && rp3 != NULL);

rp2->step();

rp3->step();

if (rp2->row() != rp->row() || rp2->col() != rp->col())

continue;

if (rp3->row() == rp->row() && rp3->col() == rp->col())

continue;

if (v.otherRobotAt(rp) == rp2)

break;

}

assert(k < 100);

} break; case 10: { // step moves robot

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

Robot\* rp = v.getRobotByName("Abner");

assert(rp != NULL);

rp->step();

assert((rp->dir() == NORTH && rp->row() == 2 && rp->col() == 4) ||

(rp->dir() == SOUTH && rp->row() == 4 && rp->col() == 4) ||

(rp->dir() == EAST && rp->row() == 3 && rp->col() == 5) ||

(rp->dir() == WEST && rp->row() == 3 && rp->col() == 3));

} break; case 11: { // step costs robot

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

Robot\* rp = v.getRobotByName("Abner");

assert(rp != NULL);

rp->step();

assert(rp->energy() == 69);

} break; case 12: { // step sometimes turns robot

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

Robot\* rp = v.getRobotByName("Abner");

assert(rp != NULL);

int k;

for (k = 0; k < 70; k++)

{

rp->step();

if (rp->dir() != NORTH)

break;

}

assert(k < 70);

} break; case 13: { // step returns false if robot dead

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

Robot\* rp = v.getRobotByName("Abner");

assert(rp != NULL);

for (int k = 0; k < 69; k++)

assert(rp->step());

rp->step();

assert(!rp->step());

} break; case 14: { // step recharges robot at energy source

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

Robot\* rp = v.getRobotByName("Abner");

assert(rp != NULL);

rp->step();

v.addEnergySource(rp->row()-1, rp->col() );

v.addEnergySource(rp->row()+1, rp->col() );

v.addEnergySource(rp->row() , rp->col()-1);

v.addEnergySource(rp->row() , rp->col()+1);

rp->step();

assert(rp->energy() == 70);

} break; case 15: { // Brother, can you spare a dime?

// Try 100 times; there's 3/4 probability to succeed

int k;

for (k = 0; k < 100; k++)

{

Valley v(10, 10);

v.addRobot("Abner", 3, 4, NORTH);

Robot\* rp = v.getRobotByName("Abner");

assert(rp != NULL);

for (int n = 0; n < 72; n++)

rp->step();

assert(rp->energy() == 0);

int r = rp->row()+1;

int d = NORTH;

if (rp->row() == 9)

{

r = 8;

d = SOUTH;

}

v.addRobot("Betty", r, rp->col(), d);

Robot\* rp2 = v.getRobotByName("Betty");

int oldEnergy = rp2->energy();

rp2->step();

if (rp2->row() == rp->row() && rp2->col() == rp->col())

{

assert(rp->energy() == 10);

assert(rp2->energy() == oldEnergy - 11);

break;

}

}

assert(k < 100);

} break; case 16: { // otherRobotAt finds minimum energy

// Try 100 times; there's (3/4)^5 probability to succeed

int k;

for (k = 0; k < 100; k++)

{

Valley v(10, 10);

v.addRobot("Abner", 3, 5, EAST);

Robot\* rpa = v.getRobotByName("Abner");

v.addRobot("Betty", 3, 5, EAST);

Robot\* rpb = v.getRobotByName("Betty");

v.addRobot("Chris", 3, 8, WEST);

Robot\* rpc = v.getRobotByName("Chris");

assert(rpa != NULL && rpb != NULL && rpc != NULL);

rpc->step();

if (rpc->row() != 3 || rpc->col() != 7)

continue;

v.addRobot("Duke", 3, 7, WEST);

Robot\* rpd = v.getRobotByName("Duke");

assert(rpd != NULL);

v.step();

if (rpa->row() != 3 || rpa->col() != 6 || rpb->row() != 3 ||

rpb->col() != 6 || rpc->row() != 3 || rpc->col() != 6 ||

rpd->row() != 3 || rpd->col() != 6)

continue;

assert(v.otherRobotAt(rpa) == rpc);

break;

}

assert(k < 100);

} break; case 17: { // Robot never steps out of bounds

Valley v(5, 15);

v.addRobot("Abner", 4, 14, SOUTH);

Robot\* rp = v.getRobotByName("Abner");

assert(rp != NULL);

for (int r = 0; r < 5; r++)

for (int c = 0; c < 15; c++)

v.addEnergySource(r, c);

bool moved = false;

bool movedpast10 = false;

for (int n = 0; n < 1000; n++)

{

rp->step();

if (rp->row() != 4 || rp->col() != 14)

moved = true;

if (rp->col() >= 10)

movedpast10 = true;

assert(rp->row() >= 0 && rp->row() < 5 &&

rp->col() >= 0 && rp->col() < 15);

}

assert(moved && movedpast10);

} break;

}

cout << "Passed" << endl;

}

Winter 2013 CS 32

## Programming Assignment 1 Beneath the Valley of the Robots

**Time due: 9:00 PM Tuesday, January 15**

The appendix to this document is the specification of the last CS 31 project from last quarter. We will provide you with a correct[1](http://www.cs.ucla.edu/classes/winter13/cs32/Projects/1/spec.html#endnote1) solution to that project. Your assignment is to (1) organize the code for the solution in appropriate header and implementation files, and (2) implement a couple of new features.

You should read [the appendix](http://www.cs.ucla.edu/classes/winter13/cs32/Projects/1/cs31p7spec.html) now. It describes robots exploring a valley. You will be working with [this code that implements the solution](http://www.cs.ucla.edu/classes/winter13/cs32/Projects/1/cs31p7soln.cpp.txt). Notice that it is a single file. (Just so you know, [the way we tested its correctness](http://www.cs.ucla.edu/classes/winter13/cs32/Projects/1/cs31p7testing.html) is similar to how we'll test the correctness of the programs you write in CS 32.)

### Organize the code

Take the single source file, and divide it into appropriate header files and implementation files, one pair of files for each class. Place the main routine in its own file named main.cpp. Make sure each file #includes the headers it needs. Each header file must have include guards.

Now what about the manifest constants? Place them in their own header file named globals.h. And what about the auxiliary function clearScreen? Place it in its own implementation file namedutilities.cpp, and place its prototype declaration in globals.h.

The [Visual C++ 2010](http://www.cs.ucla.edu/classes/winter13/cs32/visualcpp.html) writeup demonstrates how to create a multi-file project. From the collection of the nine files produced as a result of this part of the project, make sure you can build an executable file that behaves exactly the same way as the original single-file program.

### Add two features

Under the current movement rules, robots tend to get stuck along walls and in corners. For example, a robot at the north end of the valley that is facing north will try (and fail) to move north, using up one energy unit anyway.

One of your tasks is to implement a new rule. As before, a robot with nonzero energy will, with probability 1/3, pick a random direction to face. Then the following ensues: As before, if the robot can move one step in the direction it is now facing, it will do so. However, if the robot cannot, it will reverse direction and then, if it can, move one step. (The only reason it wouldn't be able to move after reversing would be if the valley is only one unit across.) In either case, the attempt to move costs one energy unit. For example, suppose (after possibly turning) a robot is facing north in the middle of the valley; it will take one step north. But if a north-facing robot were at the north edge of the valley, it would turn to face south and move one step (unless the valley has only one row).

You should be able to have the robots bouncing off the walls in this manner by making changes to only one source file.

Your second task is to modify the robots so that each has a non-rechargeable battery. When a robot is created, its battery is charged with 12 energy units. When a robot is asked to step:

* if its energy level (not counting the battery) is greater than 0, then it follows the usual rules.
* if its energy level (not counting the battery) is 0 and the battery level is greater than 0, then one unit of energy is transferred from the battery to the robot (so the battery level decreases by 1), and the robot follows the usual rules.
* if its energy level is 0 and the battery level is 0, the robot will not move.

If a robot moves onto an energy source, it is recharged to 70 energy units (not counting the battery) and the battery level is unchanged. If a robot with at least 35 energy units (not counting the battery) moves onto a spot occupied by a robot with 0 units (not counting the battery), it will transfer 10 units to that other robot; neither robots' battery level is changed.

For example, suppose a robot has energy level 1 and battery level 5. After stepping, it will have energy level 0 and battery level 5. When asked to step again, it will transfer one unit from the battery to itself and use it to move, resulting in energy level of 0 and battery level of 4. If another robot donates energy to it, it will have energy level 10 and battery level of 4. if it then steps onto an energy source, it will have energy level 70 and battery level of 4.

To implement this functionality, you will need to do the following:

* Define a class named Battery with the following public interface:
* class Battery
* {
* public:
* Battery(int initialLevel);
* int level() const;
* void decrease();
* };

The constructor initializes a Battery object to have the number of energy units to begin with specified by the parameter. You may assume (i.e., you do not have to check) that initialLevel will not be negative. The decrease function does nothing if the battery's energy level is 0; otherwise, it decreases it by 1. The level function returns the battery's current level.

The class declaration (with any private members you choose to add to support your implementation) must be in a file named Battery.h, and the implementation of the Battery class's member functions must be in Battery.cpp. You must not add any other public members to the class. (This implies, for example, that you must not add a public default constructor.) No member function of the Battery class may write to cout.

* Add a data member of type Battery (not of type pointer-to-Battery) to the Robot class, and provide this public function to return the energy level of a robot's battery:
* class Robot
* {
* …
* int batteryLevel() const;
* …
* };

Note that if a robot has energy level 1 and battery level 4, the batteryLevel function returns 4 and the energy function returns 1 (not 5).

* Make it so that a message that used to appear in the form
* Robbie has 23 energy units left

now appear in the form

Robbie has 23 energy units left; battery level is 8

* Implement the new step rules that take the battery into account.

### Turn it in

By Monday, January 14, there will be a link on the class webpage that will enable you to turn in your source files. You do not have to turn in a report or other documentation for this project. What you will turn in for this project will be one zip file containing only the eleven files you produced, no more and no less. The files must have these names exactly:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Battery.h | Robot.h | EnergySource.h | Valley.h | globals.h |  |
| Battery.cpp | Robot.cpp | EnergySource.cpp | Valley.cpp | utilities.cpp | main.cpp |

The zip file itself may be named whatever you like.

If we take these eleven source files, we must be able to successfully build an executable using Visual C++ 2010 and one using clang++ or g++ — you must not introduce compilation or link errors.

If you do not follow the requirements in the above paragraphs, your score on this project will be zero. "Do you mean that if I do everything right except misspell a file name or include an extra file or leave off one semicolon, I'll get no points whatsoever?" Yes. That seems harsh, but attention to detail is an important skill in this field. A draconian grading policy certainly encourages you to develop this skill.

The only exception to the requirement that the zip file contain exactly eleven files of the indicated names is that if you create the zip file under Mac OS X, it is acceptable if it contains the additional files that the Mac OS X zip utility sometimes introduces: \_\_MACOSX, .DS\_Store, and names starting with .\_ that contain your file names.

To not get a zero on this project, your program must meet these requirements as well:

* Except to add the member function Robot::batteryLevel, you must not make any additions or changes to the public interface of any of the classes. (You are free to make changes to the private members and to the implementations of the member functions.)
* If we replace your main.cpp file with the following, the program must build successfully under both Visual C++ and either clang++ or g++:
* #include "Valley.h"
* #include "Valley.h"
* #include "Robot.h"
* #include "Robot.h"
* #include "EnergySource.h"
* #include "EnergySource.h"
* #include "Battery.h"
* #include "Battery.h"
* #include "globals.h"
* #include "globals.h"
* int main()
* {}
* If we replace your main.cpp file with the following, the program must build successfully under both Visual C++ and either clang++ or g++:
* #include "Battery.h"
* int main()
* {
* Battery b(20);
* if (b.level() > 15)
* b.decrease();
* }

Battery.h must not contain any #include line that, if removed, still allows the above replacement main.cpp to compile successfully under both Visual C++ and either clang++ or g++.

* If we replace your main.cpp file with the following, the program must build successfully under both Visual C++ and either clang++ or g++:
* #include "Robot.h"
* int main()
* {}

Robot.h must not contain any #include line that, if removed, still allows the above replacement main.cpp to compile successfully under both Visual C++ and either clang++ or g++.

* If we replace your main.cpp file with the following, the program must build successfully under both Visual C++ and either clang++ or g++:
* #include "EnergySource.h"
* int main()
* {
* EnergySource es(1, 1);
* }

EnergySource.h must not contain any #include line that, if removed, still allows the above replacement main.cpp to compile successfully under both Visual C++ and either clang++ or g++.

* If we replace your main.cpp file with the following, the program must build successfully under both Visual C++ and either clang++ or g++:
* #include "globals.h"
* #include "Valley.h"
* int main()
* {
* Valley v(10, 10);
* v.addRobot("Abner", 3, 4, WEST);
* }

Valley.h must not contain any #include line that, if removed, still allows the above replacement main.cpp to compile successfully under both Visual C++ and either clang++ or g++, except that Valley.h should include globals.h.

* If we replace your main.cpp file with the following, the program must build successfully under both Visual C++ and either clang++ or g++:
* #include "Robot.h"
* #include "Valley.h"
* #include "globals.h"
* int main()
* {
* Valley v(10, 10);
* Robot r("Abner", &v, 3, 4, WEST);
* }
* If we replace your main.cpp file with the following, the program must build successfully under both Visual C++ and either clang++ or g++:
* #include "Valley.h"
* #include "Robot.h"
* #include "globals.h"
* int main()
* {
* Valley v(10, 10);
* Robot r("Abner", &v, 3, 4, WEST);
* }
* If we replace your main.cpp file with the following, the program must not build successfully; attempting to compile it should produce an error message like 'es' uses undefined class 'EnergySource' or Variable has incomplete type 'EnergySource' or variable 'EnergySource es' has initializer but incomplete type (and perhaps other error messages):
* #include "Valley.h"
* #include "Robot.h"
* #include "globals.h"
* int main()
* {
* Valley v(10, 10);
* Robot r("Abner", &v, 3, 4, WEST);
* EnergySource es(1, 1);
* }
* If we replace your main.cpp file with the following, the program must not build successfully; attempting to compile it should produce an error message like 'v' uses undefined class 'Valley'or Variable has incomplete type 'Valley' or variable 'Valley v' has initializer but incomplete type (and perhaps other error messages):
* #include "globals.h"
* #include "EnergySource.h"
* #include "Robot.h"
* int main()
* {
* Valley v(10, 10);
* }
* If we replace your main.cpp file with the following, the program must not build successfully; attempting to compile it should produce an error message like 'Battery' : no appropriate default constructor available or no matching constructor for initialization of 'Blah' or no matching function for call to 'Battery::Battery()' (and perhaps other error messages):
* #include "Battery.h"
* int main()
* {
* Battery b;
* }
* If a .cpp file uses a class or function declared in a particular header file, then it should #include that header. The idea is that someone writing a .cpp file should not worry about which header files include other header files. For example, a .cpp file using an A object and a B object should include both A.h (where presumably the class A is declared) and B.h (where B is declared), without considering whether or not A.h includes B.h or vice versa.

To create a zip file on a SEASnet machine, you can select the eleven files you want to turn in, right click, and select "Send To / Compressed (zipped) Folder". Under Mac OS X, copy the files into a new folder, select the folder in Finder, and select File / Compress "folderName"; make sure you copied the files into the folder instead of creating aliases to the files.

### Advice

Developing your solution incrementally will make your work easier. Start by making sure you can build and run the original program successfully with the one source file having the name main.cpp. Then, starting with EnergySource, say, produce EnergySource.h, removing the code declaring the EnergySource class from main.cpp, but leaving in main.cpp the implementation of the EnergySource member functions. Get that two-file solution to work. Also, make sure you meet those of the requirements above that involve only the EnergySource.h header.

Next, split off Robot.h, testing the now three-file solution and also making sure you meet those of the requirements above that involve only the EnergySource.h and Robot.h headers. Continue in this manner until you've produced all the required headers except Battery.h, the whole program still works, and you meet all the applicable requirements.

Now split off the member function implementations of, say, EnergySource, putting them in EnergySource.cpp. Test everything again. You see where this is going. The basic principle is to not try to produce all the files at once, because many misconceptions you have will affect many files. This will make it difficult to fix all those problems, since many of them will interfere with each other. By tackling one file at a time, and importantly, not proceeding to another until you've got everything so far working, you'll keep the job manageable, increasing the likelihood of completing the project successfully and, as a nice bonus, reducing the amount of time you spend on it.

### Help

While we will provide you assistance in clarifying what this assignment is asking for and in using Visual C++ and either clang++ or g++, we will otherwise offer minimal help with this assignment. This is to give you a chance to honestly evaluate your own current programming ability. If you find that you're having trouble with the C++ program itself (not simply the VC++, Xcode, or g++ environment, which may be new to you), then you may want to reconsider your decision to take this class this quarter. Perhaps you've let your C++ programming skills get rusty, or maybe you didn't learn the material in CS 31 or its equivalent very well. If you decide to take the course later, what you should do between now and then is program, program, program! Solve some old or current CS 31 or PIC 10A or early PIC 10B projects, and read and do the exercises in a good introductory programming text using C++. You'll have to be self-motivated to make time for that, but the payoff will be a greater likelihood for success in CS 32.

#### Endnote

[1](http://www.cs.ucla.edu/classes/winter13/cs32/Projects/1/spec.html#ref1) "Correct" in terms of what a CS 31 student would know. For example, a CS 31 student wouldn't know that sometimes you need to write a copy constructor, so the posted solution ignores that issue. (You don't have to worry about that issue for this project, either.)