Darwin

Due: Thu, 7 Nov 2013, 10pm
70 pts, 7% of total grade.

Specification

Write a program to simulate **Darwin's World**.

Darwin's World contains a two-dimensional grid. Each square in the world can contain at most one creature.

Each **creature** has a **species**, a **direction**, and a **program counter**.

Each **species** has a **program** (a set of instructions).

A creature executes the instruction indicated by the program counter. Each creature is given a turn.

Here's an example of a creature program:

0. if_wall **3**
1. hop
2. go **0**
3. left
4. go **0**

Creatures begin executing at line **0**. There are **two** types of instructions: those which cause an **action** and those which affect the flow of **control**. Above, the only action instructions are **hop** and **left**. The rest are control instructions.

Darwin gives each Creature a turn in a **left-to-right** and **top-down** ordering. During a turn a Creature can execute only one **action** instruction.

Here are the descriptions of the **9** instructions:

Type	Instruction	Description
Action	hop	If the space ahead is empty, move forward, otherwise, do nothing.
	left	Turn to face left.
	right	Turn to face right.
	infect	If the space ahead contains a creature of a different species, change that creature to be of your species, reset the program counter, but leave the direction unchanged, otherwise, do nothing.
Control	if_empty **n**	If the space ahead is empty, go to line **n**, otherwise, go to the next line.
	if_wall **n**	If the space ahead is a wall, go to line **n**, otherwise, go to the next line.
	if_random **n**	Randomly choose between going to line **n** or the next line. If **rand()** from **<cstdlib>** returns an **odd** number, go to line **n**. Call **srand(0)** at the start of every test case that uses **rand()**.
	if_enemy **n**	If the space ahead contains a creature of a different species, go to line **n**, otherwise, go to the next line.
	go **n**	Go to line **n**.

Create a good **OO** design by writing well-defined classes that are responsible for a specific and modular part of the solution. Avoid **getters**, **setters**, and **friends**, which are often signs of a bad design:

Getters and Setters

You **must** design an **API** for the various classes.

The **API** should be rich enough to be able to create the **best** creature without writing any code.

I mean something like this:

```
Species best;
best.addInstruction(...);
best.addInstruction(...);
...
Creature b1(best, ...);
Creature b2(best, ...);
...
Darwin x(72, 72);
x.addCreature(b1, ...);
x.addCreature(b2, ...);
...
```

Create a **UML** diagram to represent the design. Use any **UML** editor that you like. The diagram **only** needs to show the associations and multiplicity between the classes.

For **all** projects, the **minimum** requirement for getting a **non-zero** grade is to write **standard-compliant C++ (-std=c++0x)** and to satisfy **all** of the **requirements** in the **table** below, including the precise **naming** of all the **files**.

For this project, yet another additional **minimum** requirement for getting a **non-zero** grade is that your code **must** successfully pass **five** other students' acceptance tests.

You can earn **5 bonus pts**, if you work with a **partner** using **pair programming** and vouch for the fact that you worked on the project **together** for more than **75%** of the time.

Only **one** solution **must** be turned in for the **pair**. If **two** solutions are turned in, there will be a **10%** penalty, and the **later** one will be graded.

You can earn an additional **5 bonus pts**, if your **best** creature outnumbers **all** other species.

Bonus pts will **not** increase the **total score** beyond the **max** score.

You **may not** use **new**, **delete**, **malloc()**, **free()**. You **may** use the **STL**, except for **allocator**.

Analysis

These are additional descriptions of the underlying math:

- Dickinson
- Duke
- Stanford

Tools

- Doxygen
- Git
- GitHub
- Gliffy

- Google Test (1.6.0)
- Valgrind
- yUML

Guides

- Git Cheat Sheet
- Git Guide
- Git Immersion
- Git Reference
- Google C++ Style Guide
- Try GitHub

Requirements

	Points	Description	Files	Submission
1	5 pts	**Git Repository** Set up a **private Git repository** at **GitHub**, named **cs371p-darwin**. Invite the grader to your repository. Commit **at least 5 times**. Commit once for each **bug** or **feature**. If you cannot describe your changes in a sentence, you are not committing often enough. Write meaningful commit messages and identify the corresponding **issue** in the **issue tracker** (below). Create a **tag** for important milestones (e.g. without a cache, with a lazy cache, etc.). Create a **log** of the commits. Push frequently. It is **your** responsibility to protect your code from the rest of the students in the class. If your code gets out, **you** are as guilty as the recipient of **academic dishonesty**.	Darwin.log	GitHub Turnin
2	5 pts	**Issue Tracker** The **GitHub** repository comes with an **issue tracker**. Create an issue for each of the **requirements** in this table. Create an issue for each **bug** or **feature**, both open and closed. Describe and label each issue adequately. Create **at least 10** more issues in addition to the **requirements** in this table.		GitHub
3	15 pts	**Unit Tests** The grader's **GitHub** account will have a public **Git repository** for **unit tests** and **acceptance tests**. It is **critical** that you clone the grader's public repo into a **different** directory than the one you're using for your private repo. Write unit tests **before** you write the code. When you encounter a bug, write a unit test that **fails**, fix the bug, and confirm that the unit test passes. Write **at least an average of 3** unit tests for **each** function. Tests corner cases and failure cases. Name tests logically. Push and pull the unit tests to and from the grader's repository. Prepend **<cs-username>**- to the file names at **GitHub** (i.e. **foo-TestDarwin.c++** and **foo-TestDarwin.out**). Reach consensus on the unit tests. You **must** use **Valgrind**.	TestDarwin.c++ TestDarwin.out	GitHub Turnin
4	15 pts	**Acceptance Tests** The grader's **GitHub** account will have a public **Git repository** for **unit tests** and **acceptance tests**. It is **critical** that you clone the grader's public repo into a **different** directory than the one you're using for your private repo. Write acceptance tests **before** your write the code. When you encounter a bug, write an acceptance test that **fails**, fix the bug, and confirm that the acceptance test passes. Create **at least 200 lines** of acceptance tests. Tests corner cases and failure cases. In your acceptance tests include **five** other students' acceptance tests (another **200 lines** for all **five**, at least). Push and pull **only** your acceptance tests to and from the grader's repository. Prepend **<cs-username>**- to the file names at **GitHub** (i.e. **foo-RunDarwin.c++** and **foo-RunDarwin.out**). Reach consensus on the acceptance tests. You **must** use **Valgrind**.	RunDarwin.c++ RunDarwin.out	GitHub Turnin
5	15 pts	**Implementation** Use **assert** to check **pre-conditions**, **post-conditions**, **argument validity**, **return-value validity**, and **invariants**. Worry about this **last**, but your program should run as **fast** as possible and use as **little** memory as possible.	Darwin.c++ Darwin.h	GitHub Turnin
6	5 pts	**Documentation** Use **Doxygen** to document the **interfaces**. The above documentation only needs to be generated for **Darwin.h**. Comment each function meaningfully. Use comments **only** if you need to explain the **why** of a particular	html/*	Turnin

		implementation. Choose a coding convention and be consistent. Use good variable names. Write readable code with good indentation, blank lines, and blank spaces.		
7	5 pts	**Design** The **UML** diagram.	Darwin.pdf	GitHub Turnin
8	5 pts	**Submission** Rename "**makefile.c++**" to "**makefile**". Fill out the **Google Form** and submit the **ZIP file** to **Turnin**.	makefile.c++ Darwin.zip	Google Turnin

Grader

Name	GitHub ID	GitHub Test Repository	Turnin ID	Turnin Project Folder	Google Form
Reza Mahjourian	rezama	cs371p-darwin-tests	reza	cs371ppj4	Google Form

Submission

Submit a single **ZIP** file, named **Darwin.zip**, to the grader's **Turnin** account, with the following files:

- html/*
- makefile
- Darwin.c++
- Darwin.h
- Darwin.log
- Darwin.pdf
- RunDarwin.c++
- RunDarwin.out
- TestDarwin.c++
- TestDarwin.out