Functionality Outline

**GeneticsSim**

Programming Assignment 1

February 18, 2016

**Prepared by:**

Marcus Posey

[mlp0027@uah.edu](mailto:mlp0027@uah.edu)

**Prepared for:**

Dr. Rick Coleman

CS 307, Object Oriented Programming

Computer Science Department

University of Alabama in Huntsville

**Table of Contents**

1. System Overview ……………………….. 1
2. Relevant Terms and Acronyms …………. 1
3. Object Functionality …..………………… 1

Simulation.cpp ....…………………… 1

Allele.cpp …………………………… 2

Gene.cpp ……………………………. 2

Organism.cpp ……………………….. 3

ColemanXMLParser.cpp …………… 4

Observable.cpp ……………………... 4

IObserver.cpp ………………………. 4

LoveChamber.cpp ………………….. 4

StatCounter.cpp …………………….. 5

1. **System Overview**

The main objective of the project is to produce a program capable of simulating Mendelian laws of inheritance. These laws of inheritance govern what composes the genetic makeup of the offspring of two organisms. This task will be achieved by creating software models of basic components of a living organism. These include, but may not be limited to, alleles, genes, genotypes, and the organism itself. By crossing the genotypes of two parent organisms that are to be read from an initial xml file, new genotypes that belong to offspring will be created. This data will then be recorded to acquire statistical representations of total gene and genotype counts among all children. All genetic data (i.e. that of parent organisms and offspring) will be presented on the screen in some manner—initially through a command-line interface and, in the future, possibly through a GUI.

1. **Relevant Terms and Acronyms**

* **Allele** – These are represented in the model by a single character that is either uppercase or lowercase. Uppercase letters dominant traits, while lowercase represent recessive traits. Pairs of alleles make up genes.
* **Gene –** Genes are pairs of alleles that influence the phenotype (visible characteristics) of an organism.
* **Genotype –** The collection of genes an organism possesses that contribute to its traits and characteristics. Within this program, the genotype can be thought of as the essence of an organism.
* **unordered\_map –** This is the STL implementation of a dictionary using a hash table. It assigns certain values to unique keys.
* **unique\_ptr** – This is the STL implementation of a managed pointer. It implements RAII to acquire possession of memory on the heap and free said memory when the container goes out of scope.

1. **Object Functionality**

Genetics.cpp

main()

Call run() on an anonymously instantiated Simulation object

Simulation.cpp

Simulation()

Initialize\_offspringCount to 0

Call Simulation::init

init()

Get valid file name from command line

Use ColemanXMLParser to extract organism data from file

Create unique\_ptr to LoveChamber, passing newly created organisms as arguments

Add \_statCounter to the LoveChamber object's list of observers

Get number of offspring to generate from command line

run()

Call LoveChamber::mate with \_offspringCount supplied as the argument

Call StatCounter::printStats

Allele.cpp

Allele(char symbol, const string &description)

Set \_symbol equal to symbol

Set \_description equal to description

isDominant()

Return true if \_symbol is an uppercase character

getSymbol()

Return \_symbol

getDescription()

Return \_description

Gene.cpp

Gene(const Allele &a1, const Allele &a2, const string &desc)

Push a1 and a2 to the back of \_alleles

Sort \_alleles by the symbols in each Allele (uppercase has priority)

Set \_allelesString equal to the concatenation of the symbols in each Allele in \_alleles

getRandomAllele()

Randomly pick either \_alleles[0] or \_alleles[1] and return it

first()

Return \_alleles[0]

second()

Return \_alleles[1]

getDescription()

Return \_description

getPhenotype()

Compare \_alleles[0] and \_alleles[1]

If \_alleles[0] and \_alleles[1] have the same dominance

Return the either of their descriptions

If \_alleles[0] is dominant

Return \_alleles[0] description

If \_alleles[1] is dominant

Return \_alleles[1] description

getZygosity()

If \_alleles[0] has the same dominance as \_alleles[1]

If \_alleles[0] is not dominant

Return "homozygous recessive"

Else

Return "homozygous dominant"

Else

Return "heterozygous dominant"

getAllelesAsString()

Return \_allelesAsString

toString()

Return the string: Gene::getZygosity() + " (" + Gene::getPhenotype() + " " + getAllelesAsString() + ")"

Organism.cpp

Organism(const string &genus, const string &species, const string &name)

Initialize \_serveCounter to 0

Initialize \_genus to genus

Initialize \_species to species

Initialize \_name to name

addGene(const Gene &gene)

Push gene to the back of the \_genotype vector

serveGene()

If \_serveCounter >= \_genotype.size()

Set \_serveCounter back to 0

Return \_genotype[\_serveCounter++]

printDescription()

Print the genus, species, and name to stdout

For each gene in the genotype

Print the gene description to stdout

Print the alleles of the gene to stdout

getGeneCount()

Return the size of \_genotype

getGenotype()

Return a const reference to \_genotype

getGenus()

Return \_genus

getSpecies()

Return \_species

getName()

Return \_name

ColemanXMLParser.cpp

ColemanXMLParser(const string &filename)

Pass filename.c\_str() to the constructor of \_parser for instantiation

parseFile(vector<Organism> &organisms)

Emplace two organisms into organisms with \_parser.getGenus(), \_parser.getSpecies(), and \_parser.getCommonName() data

Create chars for dominance and recessive symbols

Create char arrays for traits, dominant allele descriptions, and recessive allele descriptions

Create an unordered\_map 'possibleAlleles' with a char as the key and a pair of two strings as the value to store all alleles found in the file

For every allele in the file

Add each Allele symbol (as a key) and the associated description and trait info (as a value) to possibleAllele

Create a char array of char arrays to store the parent's genotypes in

Create a vector of Allele objects called 'alleles'

For each parent

Read pairs of 2 alleles at a time into 'alleles'

If 'alleles' contains 2 alleles

Create a gene and add it to the parent

Clear 'alleles'

Observable.cpp

addObserver(IObserver<T> &o)

Push 'o' to the back of the \_observers vector

removeObserver(const IObserver<T> &o)

Use erase-remove idiom to remove 'o' from \_observers vector

notifyAll(const T &arg)

For each IObserver<T>\* in \_observers

Call IObserver::notify(arg)

IObserver.cpp

notify(const T &arg)

Set as pure virtual function

LoveChamber.cpp

LoveChamber(Organism o1, Organism o2)

Set \_o1 to o1

Set \_o2 to o2

Seed a random number using the time

Call LoveChamber::printParentData

printParentData()

Call \_o1.printDescription()

Call \_o2.printDescription()

mate(int count)

Store the number of genes a parent has in an int 'size'

For i=0 to count

Create an organism using the genus, species, and name of one of the parents

For j=0 to size

Store the results of \_o1.serveGene() and \_o2.serveGene()

Take a random allele from each gene and create a new gene

Add the newly created gene to the organism

Call LoveChamber::notifyAll using the newly created and populated organism

StatCounter.cpp

notify(const Organism &arg)

For each gene in arg's genotype

Increment the count of the gene in the \_geneCounts hash table by one

Increment the count of the genotype in the \_genotypeCounts hash table by one

printStats()

For each pair in the hash table \_geneCounts

Print the gene description to stdout

Print the count of the gene to stdout

Print the gene as a string to stdout

For each pair in the hash table \_genotypeCounts

Print the genotype to stdout

Print the number of offspring who have that genotype