**Functionality Outline**

**Mendelian Simulation**

**Programming Assignment 1**

**February 27, 2018**

**Prepared By**

**Atchima Klomkaew**

**ak0061@uah.edu**

**Prepared for**

**Dr. Rick Coleman**

**CS 307, Object Oriented Programming**

**Computer Science Department**

**University of Alabama in Huntsville**

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

This project shall stimulate the processes of inheritance incorporating the Mendelian laws of genetics. The simulation shall read from the input file to identify the types of gene, genes’ definitions, and parents’ genotypes. It shall request the user to determine the total amount of offspring to generate. The parents’ genes will cross with one another in order to form an offspring, and the result will be similar to Mendel’s pea plants simulation (second generation). The phenotype ratio of the result is 3:1 for dominant gene and recessive gene respectively. The genotype ratio is 1:2:1 for homozygous dominant, heterozygous dominant, and homozygous recessive respectively. The output of this simulation shall include the amount of offsprings’ genotype generated for each gene, which should be in the ratio mentioned.

1. **Relevant Terms and Acronyms**

Mendelian laws of genetics include law of segregation, law of independent assortment, and law of dominance. The upper-case letter signifies dominant allele while lower-case letter signifies recessive allele. Genotype is the genetic makeup of an organism, i.e. TT Ww Ss cc. On the other hand, phenotype is the organism’s physical appearance, e.g. tall, wrinkled, green, and white.

1. **Object Functionality**

MendelianSimulation.cpp

Main()

Instantiate Simulation object

Call Simulation::initializeSimulation

Call Simulation::runSimulation

Simulation.cpp

initializeSimulation()

Query user for name of the data file

Create instance of GeneticsSimDataParser (*parser*)

Get data to define each object from data parser

Create instance of GeneMaster

Call GeneMaster::initializeGeneMaster

Parameter - parser

Define and generate parents

Call GeneticsSimDataParser::getParentGenotype

For two parents

Create instance of Organism

Call Organism::initializeOrganism()

Parameters - *parGenotype, genusName,*

*speciesName, commonName, numGenes* or

GeneticsSimDataParser::getGeneCount

Call Simulation::runSimulation

runSimulation()

Query user for amount of offsprings and set it to *numOffsprings*

Create a new instance of Breed

Call Breed::generateOffsprings

Parameters - numOffsprings, parent1, parent2, numGene

Create a new instance of OffspringReader

Call OffspringReader::getTotalOffsprings

Parameters - Breed::getVector

Organism.cpp

initializeOrganism(genotype : String, genusName : String, speciesName : String,

commonName : String, geneCount : int)

Call Organism::setGenotype

Call Organism::setGenusName

Call Organism::setSpeciesName

Call Organism::setCommonName

Split the *genotype* string by whitespaces into x substrings (where x is the

number of genes)

For number of geneCount

Call Organism::generateGene

Parameter: parAlleles

Store each Gene object created in vector<Gene> *geneStorage*

generateGene(parAlleles : String)

Initialize a Gene object and pass parAlleles to Gene object

(Gene(parAlleles))

Return the Gene object

setGenotype()

Set private member *genotype* to the value received from calling

GeneticsSimDataParser::getParentGenotype

setGenusName()

Set private member *genusName* to the value received from calling

GeneticsSimDataParser::getGenus

setSpeciesName()

Set private member *speciesName* to the value received from calling

GeneticsSimDataParser::getSpecies

setCommonName()

Set private member *commonName* to the value received from calling

GeneticsSimDataParser::getCommonName

getGenotype()

Return private member *genotype*

getGenusName()

Return private member *genusName*

getSpeciesName()

Return private member *speciesName*

getCommonName()

Return private member *commonName*

createGene(token : String)

Create new instance of Gene object

Call Gene::setGeneGenotype and set it to the passed in parameter

getRandAllele

Call Gene::createRandAllele with given gene

GeneMaster.cpp

initializeGeneMaster(*parser* : GeneticsSimDataParser)

Read from file:

Declare structure with geneTrait, domAllele, domSym, recAllele,

recSym

Declare vector<struct>

For each gene (GeneticsSimDataParser::getGeneCount)

Call GeneticsSimDataParser::getGeneData

Parameters - trait, domDesc, domSymbol, recDesc,

recSymbol

Assign to struct of gene definition and store each struct into

vector

Gene.cpp

Gene(pairAlleles String)

Set pairAlleles to private member variable

Call Gene::splitAlleles

splitAlleles()

Declare char *allele1*, *allele2*

Split the pairAlleles and store each letter in characters declared

createRandAllele()

Generate a random number to pick between allele1 and allele2

Breed.cpp

generateOffspring(numOffsprings : int, parent1 : Organism, parent2 : Organism,

geneNum : int)

Declare struct that contains : homoDom, heteroDom, homoRec

Declare a vector of struct

For geneNum

Declare a gene of the above struct

Call Organism::getRandAllele on both parents

Parameters - *allele*, geneType : Gene

Use ASCII values to indicate which letter is capitalized. Make sure

that the capitalized letter is up first.

Check the gene trait (should be in order)

Concatenate the random allele from each parent, creating offspring

gene.

Compare this pair of alleles to MasterGene

If str compare pairAlleles.toUpper to

MasterGene::domSym is 0 (switch case)

Add this newly generated pair of alleles to the struct

Add this struct to the vector for storage

OffspringReader.cpp

Declare integer values for homoDom, heteroDom, and homoRec

getTotalOffsprings(totalOffsprings : vector<struct>)

For iterator to go through totalOffsprings vector

Count by checking letters (Upper-Upper, Upper-Lower, or

Lower-lower) and increment appropriate counter

Print the results

There will be setters and getters for each private member variables.