Project 3 (Final)

CSCI 136. SEC 51 (0722)
Due Date: May 15, 2013

GENOMIC DATABASE

A genomic database stores sequenced DNA data from different organisms. Such databases provide functionalities to search and retrieve DNA data, organism information, and much more. In addition, the databases provides analytical information that help bioinformatics researchers.

One use of such databases is to find common DNA sequences (genes) among multiple organisms. Finding common genes is useful in relating different species (phylogenetics) as well as understanding gene functionalities. It is easier to study behavior and functionality of a particular DNA sequence in bacteria or simpler organisms than it is to study the behavior and functionality of the same DNA sequence in a more complex organism such as a human being.

The National Center for Biotechnology Information (NCBI) hosts one of the largest genomic databases in the world. This database is publicly available and a website is provided to interact with it at http://www.ncbi.nlm.nih.gov/genome/browse/. This is one of the most important resources for bioinformatics researchers world-wide.

PROJECT DESCRIPTION

For this last and final project you will implement a *simple* DNA Database using OOP methodology. You will be provided with detailed *interface* descriptions, and your objective is to *implement* the interfaces and verify your program with a provided test-function. The interface description appears at the end of this document.

First, an **Organism** class will be defined. This class will allow the instantiation of organism objects. Organisms can have many properties and attributes; for the purpose of our project we are only interested in the Common name (e.g. bobcat) and the Scientific name (e.g. Lynx Rufus) of an organism.

Second, a **DnaSequence** class will be defined. This class will allow the creation of DNA sequence objects. A DNA sequence is essentially a string (of characters). Given an existing sequence object, we should also have a mechanism to append more DNA data to it.

Finally, a **DnaDatabase** class will be defined. This class will allow the creation of a simple genomic database. In reality, databases are separate software systems where the information is stored efficiently in large storage media. For this project, we will create a single database object of the defined class to simulate an in-memory database. Our database should allow the **creation** of records of DNA sequences of organisms. Since our database design is very simple, only one DNA sequence is allowed per organism. A second call to add a DNA sequence for an *existing* organism should simply **update** the record by appending the requested sequence to the existing sequence of that organism. An important functionality that the database will provide is **search**: given a partial DNA sequence (in the form of a search string), the database should be able to find all organisms that have the search sequence within their sequences. The search should not be case-sensitive. A database

traditionally allows the **deletion** of records and you have to implement such a functionality. A few additional functionalities have been defined in the interface. You can safely assume that each database object will store no more than 100 records (i.e. 100 pairs of organisms and their respective DNA sequences). Any further requests to add can be ignored along with a message informing the user that the maximum limited has been exceeded.

LEARNING OBJECTIVES

- Programming: OOP; interface implementation, constructors, over-ridden member functions & constructors; strings in C++ and C-Str
- Bioinformatics: genome database concepts; NCBI genome database

GRADING

- 50 points for appropriate coding of interface and implementation. I expect 3 header files, 3 implementation files and 1 main application file that calls the provided test-function. When you submit, please comment out any other test/debug code you might have or be calling from your main() function.
- 30 points for passing all the tests in the provided test-code.
- 20 points for passing some additional tests.

VERY IMPORTANT: <u>It is extremely important to keep the original interface intact.</u> You can add more public member functions, if you need, and you are free to add any number of private member variables/ functions -- but do not change the definitions of the functions in the provided interface. If I cannot test your program with the interfaces we are agreeing upon, you will lose points.

SUBMISSION

Once completed, you will have a number of files. Assuming you have the following files: Organism.h, Organism.cpp, DnaSequence.h, DnaSequence.cpp, DnaDatabase.h, DnaDatabase.cpp, mainApp.cpp, you can aggregate all these files into a single **tar** file using the UNIX **tar** utility. Assuming, while on the UNIX terminal, your current working directory (pwd) is where all your source files are (headers, sources, main app), you can use the following command:

```
tar -cvf project3_<username>.tar Organism.h Organism.cpp DnaSequence.h DnaSequence.cpp
DnaDatabase.h DnaDatabase.cpp mainApp.cpp
```

This creates a single tar file called project2_<username>.tar, and you can submit this tar file to me.

Copy your tar file and provide adequate permissions with:

```
cp <tar_file> /data/biocs/b/student.accounts/sadatc/submissions/cs135/project3/

chmod 644 /data/biocs/b/student.accounts/sadatc/submissions/cs135/project3/<tar_file>
```

Note: You can also submit a zip file, if you have worked on this at home.

INTERFACE

Organism.h:

```
/*
 * Class defines an organism type
 * Organisms have common names and scientific names
 * The default constructor should set "unknown" for both
 *
 */
class Organism {
public:
  Organism();
  Organism(string commonName, string scientificName);

  void setName(string commonName, string scientificName);

  void setCommonName(string commonName);
  string getCommonName();

  void setScientificName(string scientificName);
  string getScientificName();
};
```

DnaSequence.h:

```
/*
 * Class defines a DNA Sequence of nucleotides (G,A,T,C)
 * When sequences are added, they can be added in any case
 * However, when asked to return the sequence, it should
 * be all-caps
 */
class DnaSequence {
public:
  DnaSequence();
  // the following constructor can take another DNA sequence
  // object and copy over its sequence into its own
  DnaSequence(const DnaSequence& dnaSequence);

  // for both of these functions, if there was
  // an existing sequence, the older sequence
  // is overwritten
  void setSequence(string sequence);
  void setSequence(const char *sequence);

  // for all these functions, appends to the tail end
  // of the existing sequence. If none exists,
  // the sequence becomes what was being requested to be
  // added (even if it is a single nucleotide)
  void append(string sequence);
  void append(char nucleotide);


  // return should be all caps
  string getSequence();

  // return the total size of the sequence
  int getSize();

};
```

DnaDatabase.h:

```
/*
 * Class defines a database of organisms and their DNA sequence
 * This Database (DB) allows:
 * - basic DB operations: add/update/delete
 * - counting records/searching records
 *
 * While adding a sequence record: if the organism already exists,
 * simply the new requested sequence is appended to the existing
 * sequence for this organism; only a single "record" should exist
 * per organism.
 *
 */
class DnaDatabase {
public:
  DnaDatabase();
  // add record by objects of appropriate class
  void addRecord(Organism organism, DnaSequence sequence);

  // add record by string values
  void addRecord(string organismName, string sequence);

  // given an organism, return the sequence
  DnaSequence getSequence(Organism organism);

   // delete the entry for a given organism
  void deleteRecord(Organism organism);

  // delete the entry for a given organism name
  void deleteRecord(string organismName);

  // print all existing organisms in the DB
  // one, per line.
  void printAllOrganisms();

  // returns the number of organisms
  int countAllOrganisms();

  // returns the number of organisms that have a given
  // partial sequence within their entire sequence
  int countOrganismsWithSequence(string partialSequence);

  // prints all the organisms that have
  // a given partial sequence within their entire sequence
  void searchAllOrganismsWithSequence(string partialSequence);

};
```

TEST CODE

This testing function has asserts that should pass. In your final submission, please place a call to this function from main() and comment out any additional output/testing/debugging you might have from your main() function and the application in general.

```
// function tests the implementation of project 3
// it creates a database, 3 organisms, multiple sequences
// and tests add/update/delete on the DB

void testCode() {
  // create a new DB
  DnaDatabase myGeneBank;

  // DB should be blank...
  assert(myGeneBank.countAllOrganisms() == 0);

  // create the first organism
  Organism frog("western clawed frog", "xenopus tropicalis");

  // test sequences for first organism
  DnaSequence frogSeq1;
  frogSeq1.setSequence("GATA");
  assert(frogSeq1.getSize()==4);

  frogSeq1.setSequence("CTCT");
  assert(frogSeq1.getSize()==4);
  string testStr1 = "CTCT";
  assert(frogSeq1.getSequence() == testStr1);
  frogSeq1.append("TTCCTCCTCAGAGTCATCTGAGCTCTCACTGTCATCGGAGCTCTCA");
  assert(frogSeq1.getSize()==51);

  // add organism1 & sequence1 to DB
  myGeneBank.addRecord(frog,frogSeq1);
  assert(myGeneBank.countAllOrganisms() == 1);


  // test second organism
  Organism human;
  human.setCommonName("human");
  human.setScientificName("homo sapiens");

  char arr1[] ={"taaagcagctccagaaacgtttcttgtctagcaggaaagccctggctcttagagccag"};
  DnaSequence tmpSequence;
  tmpSequence.setSequence(arr1);

  string testStr2;
  testStr2 = "TAAAGCAGCTCCAGAAACGTTTCTTGTCTAGCAGGAAAGCCCTGGCTCTTAGAGCCAGT";
  // note: capped and extra T added

  DnaSequence humanSeq1(tmpSequence);
  assert(humanSeq1.getSize()==59);
  humanSeq1.append('T');
  assert(humanSeq1.getSequence() == testStr2);

  // add organism2, sequence1
  myGeneBank.addRecord(human,humanSeq1);
  assert(myGeneBank.countAllOrganisms() == 2);

  DnaSequence humanSeq2;
  string testStr3 = "aataggataatagcagatgt";
  humanSeq2.append(testStr3);
  assert(humanSeq2.getSize()==22);
```

```
  // add organism2, sequence2
  myGeneBank.addRecord(human,humanSeq2);
  assert(myGeneBank.countAllOrganisms() == 2);

  DnaSequence humSeqFromBank = myGeneBank.getSequence(human);
  assert(humSeqFromBank.getSize()==81);

  // made-up organism
  Organism bigFoot("sasquatch","gigantopithecus blacki");
  // copy over human gene; most likely, some guy putting on
  // a gorilla suite
  DnaSequence bigFootSeq(humanSeq1);

  // add organism3, sequence1
  myGeneBank.addRecord(bigFoot, bigFootSeq);
  assert(myGeneBank.countAllOrganisms() == 2);

  // run some searches
  myGeneBank.searchAllOrganismsWithSequence("tct");
  assert(myGeneBank.countOrganismsWithSequence("tct")==3);

  myGeneBank.searchAllOrganismsWithSequence("cac");
  assert(myGeneBank.countOrganismsWithSequence("cac")==1);

  myGeneBank.searchAllOrganismsWithSequence("GGAA");
  assert(myGeneBank.countOrganismsWithSequence("GGAA")==2);

  // test delete
  myGeneBank.deleteRecord(bigFoot);

  // run some more searches
  myGeneBank.searchAllOrganismsWithSequence("tct");
  assert(myGeneBank.countOrganismsWithSequence("tct")==2);

  myGeneBank.searchAllOrganismsWithSequence("cac");
  assert(myGeneBank.countOrganismsWithSequence("cac")==1);

  myGeneBank.searchAllOrganismsWithSequence("GGAA");
  assert(myGeneBank.countOrganismsWithSequence("GGAA")==1);

  myGeneBank.printAllOrganisms();

  // finish off the tests...
  myGeneBank.deleteRecord(human);
  myGeneBank.deleteRecord(frog);
  assert(myGeneBank.countAllOrganisms() == 0);


}
```