CS 5010: Problem Set 00

Out: Friday, 1 September 2017
Due: Monday, 11 September 2017 at 6pm local time

This is an individual assignment. You are not allowed to discuss this problem set with any other person. You are also not allowed to search for or to view any solutions to similar problems that may be available on the World-Wide Web or in other resources that might otherwise have been available to you.

The main purpose of this problem set is to give us some idea of your programming skills at the beginning of the semester, when most of you are just starting the MS program. This problem set will also allow us to identify students whose programming skills are strong enough for us to invite them to take a programming test that could result in this course being waived for those students. Students who do not do well on Problem Set 00 will not be allowed to take that placement test.

This Problem Set 00 may be too hard for you to finish before its due date, in which case you should still submit a plain text (Latin-1 or UTF-8) file named `README` (not `README.txt`) that says `Problem Set 00 is too hard for me.` Students who submit a `README` file that begins with that sentence will receive full credit for Problem Set 00, but will not be allowed to take the placement test.

Students who are able to complete the problem set should create a `README` file that tells us how to compile and run your program on a Linux, Macintosh, or Windows machine (your choice, but please specify your choice in the `README` file). You may use an integrated development environment (IDE) to write your program, but you must submit your program as a set of plain text source files. Do not submit your program using any formats that would make sense only when viewed or interpreted by an IDE. If we cannot read, compile, and run your program without having to use an IDE, we will assume Problem Set 00 was too hard for you.

Submitting Your Solution

A third purpose of this problem set is to make sure all MS students have obtained a CCIS account, know how to use `sftp` to transfer files between machines, and can use Linux well enough to package files and to submit them by following these instructions:

0. If you're using a personal computer, make sure the `ssh` and `sftp` utilities are installed on your machine. If not, install them. (If you're using Linux, they should already be installed. If you're using a Macintosh, you can install those utilities by installing the Xcode Command Line Tools. If you're using a Windows machine, you may have to install a freeware version of those utilities such as PuTTY or OpenSSH.)
1. When you have obtained a CCIS account, you will have a home directory on the CCIS shared file system. Log into a CCIS Linux machine (in WVH 102, or you can use the `ssh` utility to log in remotely), and create a `~/classes/problem00` directory as follows:

```
% mkdir classes
% cd classes
% mkdir problem00
```

2. Use the `sftp` utility to copy your `README` and program files into the directory you created in step 1. Suppose, for example, that your CCIS ID is `sidewinder`, you have written your program in Java, and your `README` file along with all of your Java files reside in some directory on your own personal computer. Then you can open a terminal window (called a Command Prompt window on Windows machines) on your computer, `cd` to the directory that contains your files, and copy them to the CCIS shared file system as follows:

```
% sftp sidewinder@login.ccs.neu.edu
sidewinder@login.ccs.neu.edu's password:
Connected to login.ccs.neu.edu
sftp> cd classes/problem00
sftp> put README
sftp> mput *.java
sftp> bye
```

3. Log into a CCIS Linux machine, use the `tar` utility to create a `problem00.tgz` file that packages your entire `classes/problem00` directory into a single compressed file, and submit that file using the `/course/cs5010f17/submit` program. When you submit that file, you will also tell us your NU and CCIS Ids. Suppose, for example, your NU ID is 001234567 and your CCIS ID is `sidewinder`. Then you would submit your program like this:

```
% ssh sidewinder@login.ccs.neu.edu
% cd classes
% tar -czf problem00.tgz problem00
% /course/cs5010f17/submit 001234567 sidewinder problem00.tgz
```

   (Use your own CCIS ID instead of `sidewinder`, and use your own NU ID instead of 001234567.)
4. If all goes well, step 3 will end by giving you a confirmation number. Please write down that confirmation number so you can prove you have submitted Problem Set 00.

Problem Specification

The program you are to write solves a problem that arises when some parts of a large program are changed: Which source files need to be re-compiled, and in what order should they be re-compiled?

For this problem, we assume the large program is written in a programming language that supports modular programming, with a one-to-one correspondence between modules and source files. Each module is named, and each module begins with declarations that list the names of modules that are *used* by the module. For each module *M*, we define the set of modules on which *M depends* as the smallest set *D* such that

1. *D* contains every module used by *M*
2. if *N* is a module in *D*, then *D* contains every module used by *N*

We also assume:

- the operating system can tell us the time at which each module was last modified

- if a module has been compiled, its compiled form is a file whose name can be computed from the name of the module
- when a module is compiled, it is compiled in one of the following four modes: LP64, ILP64, LP32, or ILP32
- that mode can be determined by examining the compiled file
- before a module *M* is compiled in one of those modes, all of the modules on which *M* depends must have been compiled in that same mode, and must have been compiled after they were last modified and after all of the modules on which *M* depends were last compiled.

You may write your program using any language that is currently installed and available on `login.ccs.neu.edu` (which is a CCIS Linux machine).

Your program must define three public operations named `isCircular`, `bestMode`, and `bestPlan`. Those operations are specified below.

Your implementation of those operations must use an immutable abstract data type `Module`, specified below, together with a `ListOfModule` data type whose values are homogeneous lists of values drawn from the `Module` ADT; the `ListOfModule` data type should use lists that are idiomatic in your chosen programming language. The `Module` ADT also assumes a `ListOfString` data type whose values are homogeneous lists of strings.

To make this problem set easier for you, we are giving you sample implementations of the `Module` ADT and `ListOfModule` data type, written in several different programming languages. If you choose to use one of the programming languages we used to write those sample implementations, you should use the sample implementation we give you for that language. If your chosen programming language is not among those we used for the sample implementations, then you will need to translate one of the sample implementations into your language of choice, taking care to preserve the `Module` ADT's immutability, operations, types, and specifications.

Your implementation may also define its own data types, and may use any data types that are provided by the programming language you use, including its standard libraries.

To qualify for the placement test, your implementation must be correct and run in polynomial time (as a function of the number of modules in the large program).

Specifications of `isCircular`, `bestMode`, and `bestPlan`:

In examples shown below, `[x, y, z]` means a list of *x*, *y*, and *z*.

```
;;; isCircular : ListOfModule -> Boolean
;;; GIVEN: a list of module descriptions
;;; WHERE: no two descriptions are for the same module name   => descriptions are unique
;;; RETURNS: true if and only if one or more of the modules
;;;     depends upon itself
;;; EXAMPLES:
;;;
;;; isCircular ([])  =>  false
;;;
;;; isCircular
;;;  ([makeModule ("Main",  ["List", "AList"], 1504188920, -1,          "LP64"),
```

```
;;;    makeModule ("List",  ["Obj"],           1472652920, 1472658760, "LP64"),
;;;    makeModule ("AList", ["List", "Obj"],   1472654764, 1472659242, "LP64"),
;;;    makeModule ("Obj",   [],                1472630256, 1472638841, "LP64")])
;;;     =>  false
;;;
;;; isCircular
;;;  ([makeModule ("M1", ["M2", "M3"], 1500, -1, "ILP32"),
;;;    makeModule ("M2", ["M3"],       2000, -1, "ILP32"),
;;;    makeModule ("M3", ["M2"],       2500, -1, "ILP32"))
;;;     =>  true
;;;
;;; bestMode : String ListOfModule -> String
;;; GIVEN: a module name M and a list of module descriptions
;;; WHERE: no two descriptions are for the same module name,
;;;     M is among the modules described,
;;;     and none of the described modules depend upon themselves => Not circular
;;; RETURNS: the name of a mode (LP64, ILP64, LP32, or ILP32)
;;;     that, when used to compile all of the modules that need to be
;;;     compiled before module M can be used, would result in
;;;     compiling the fewest modules
;;; NOTE: this function may have more than one correct result
;;; EXAMPLES:
;;;
;;; bestMode
;;;  ("Main",
;;;   [makeModule ("Main",  ["List", "AList"], 1504188920, -1,         "LP64"),
;;;    makeModule ("List",  ["Obj"],           1472652920, 1472658760, "LP64"),
;;;    makeModule ("AList", ["List", "Obj"],   1472654764, 1472659242, "LP64"),
;;;    makeModule ("Obj",   [],                1472630256, 1472638841, "LP32")])
;;;     =>  "LP32"
;;;
;;; bestMode
;;;  ("Main",
;;;   [makeModule ("Main",  ["List", "AList"], 1504188920, -1,         "LP64"),
;;;    makeModule ("List",  ["Obj"],           1472652920, 1472658760, "LP64"),
;;;    makeModule ("AList", ["List", "Obj"],   1472654764, 1472659242, "LP64"),
;;;    makeModule ("Obj",   [],                1472630256, 1472638841, "ILP32")])
;;;     =>  "ILP32"
;;;
;;; bestPlan : String ListOfModule -> ListOfString
;;; GIVEN: a module name M and a list of module descriptions
;;; WHERE: no two descriptions are for the same module name,
;;;     M is among the modules described,
;;;     and none of the described modules depend upon themselves => Not circular
;;; RETURNS: a list of names for the modules that need to be compiled
;;;     using the best mode, in the order they should be compiled,
;;;     before module M can be used
;;; NOTE: this function may have more than one correct result
;;; EXAMPLE:
;;;
;;; bestPlan
;;;  ("Main",
;;;   [makeModule ("Main",  ["List", "AList"], 1504188920, -1,         "LP64"),
;;;    makeModule ("List",  ["Obj"],           1472652920, 1472658760, "LP64"),
;;;    makeModule ("AList", ["List", "Obj"],   1472654764, 1472659242, "LP64"),
;;;    makeModule ("Obj",   [],                1472630256, 1472638841, "ILP32")])
;;;     =>  ["List", "AList", "Main"]
```

Specification of the `Module` ADT:

```
;;; makeModule : String ListOfString Int Int String -> Module
;;; GIVEN: the name of a module,
;;;     a list of the names of all modules it uses
;;;     a non-negative integer giving the time at which the module's
;;;     source code was last modified (as the number of seconds that
;;;     have elapsed since midnight on 1 January 1970),
;;;     an integer giving the time at which the module was last compiled
;;;     (or -1 if the module has not been compiled),
;;;     and a string naming the mode in which the module was last compiled
;;;     (which is meaningless if the module has not been compiled)
;;; RETURNS: a description encapsulating the given information
```

```
;;;
;;; moduleName : Module -> String
;;; moduleUses : Module -> ListOfString
;;; modificationTime : Module -> Int
;;; compilationTime : Module -> Int
;;; compilationMode : Module -> String
;;;
;;; GIVEN: the description of a module
;;; RETURNS: the appropriate part of that description
;;; EXAMPLES:
;;; moduleName (makeModule ("Foo", ["Baz"], 1505109465, 1504097449, "LP64"))
;;;     =>  "Foo"
;;; moduleUses (makeModule ("Foo", ["Baz"], 1505109465, 1504097449, "LP64"))
;;;     =>  ["Baz"]
;;; modificationTime (makeModule ("Foo", ["Baz"], 1505109465, 1504097449, "LP64")
;;;     =>  1505109465
;;; compilationTime (makeModule ("Foo", ["Baz"], 1505109465, 1504097449, "LP64"))
;;;     =>  1504097449
;;; compilationMode (makeModule ("Foo", ["Baz"], 1505109465, 1504097449, "LP64"))
;;;     =>  "LP64"
```

Sample Implementations

We are giving you sample implementations, in three programming languages, of the `Module` ADT and the `ListOfModule` and `ListOfString` data types. You should translate those implementations into the language you choose to use for this problem set.

- C++
- Java
- Scheme

Last modified: Mon Sep 4 08:06:18 Eastern Daylight Time 2017