Programming Assignment 5:

Context-Sensitive Pointer Analysis

Course “Static Program Analysis” @Nanjing University
Assignments Designed by Tian Tan and Yue Li
Due: 23:00, Friday, June 19, 2020

1 Goal

In this programming assignment, you will implement whole-program context-sensitive pointer analysis for Java based on Bamboo. Specifically, we have provided a context-sensitive pointer analysis framework in Bamboo, and you need to implement three kinds of common context sensitivity variants introduced in Lecture 12. To show the usefulness of context sensitivity, we provide an interprocedural constant propagation, which uses the call graph constructed by your context-sensitive pointer analysis. If your implementation is correct, you can observe that context-sensitive pointer analysis can build a more precise call graph than class hierarchy analysis (CHA) and context-insensitive pointer analysis (CIPTA). Consequently, interprocedural constant propagation based on pointer analysis achieves better precision than CHA and CIPTA (please see Section 3.6 for more details). Again, you only need to consider a small subset of Java features.

2 Introduction to Bamboo

Bamboo is a static program analysis framework developed by the two instructors of this course, and it supports multiple static analyses (e.g., data-flow analysis, pointer analysis, etc.) for Java. Bamboo leverages Soot as front-end to parse Java programs and construct IRs (Jimple). In this assignment, we include a context-sensitive pointer analysis framework. In addition, we also include an interprocedural data-flow analysis framework and an interprocedural constant propagation to demonstrate the usefulness of context sensitivity.

2.1 Content of Assignment

The content resides in folder `bamboo/`, which includes:

- `analyzed/`: The folder containing test input files.
- `libs/`: The folder containing Soot classes with its dependencies.
- `src/`: The folder containing the source code of Bamboo. **You will need to modify six files in this folder to finish this assignment**.
- `test/`: The folder containing test classes.
- `build.gradle`: The Gradle build script for Bamboo.
- `copyright.txt`: The copyright of Bamboo.

2.2 Setup Instructions (Same as Assignment 1)

Bamboo is written in Java, so it is cross-platform. To build and run Bamboo, you need to have Java **8** installed on your system (other Java versions are currently not supported). You could download the Java Development Kit 8 from the following link: https://www.oracle.com/java/technologies/javase-jdk8-downloads.html

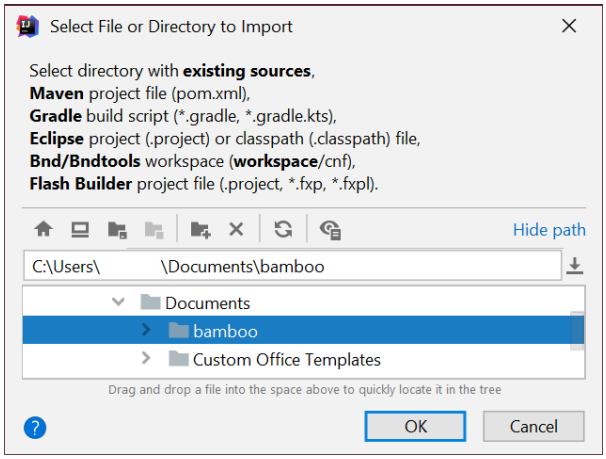
We highly recommend you to finish this (and the following) assignment(s) with IntelliJ IDEA. Given the Gradle build script, it is very easy to import Bamboo to IntelliJ IDEA, as follows.

Step 1
Download IntelliJ IDEA from JetBrains (http://www.jetbrains.com/idea/download/)

Step 2
Start to import a project

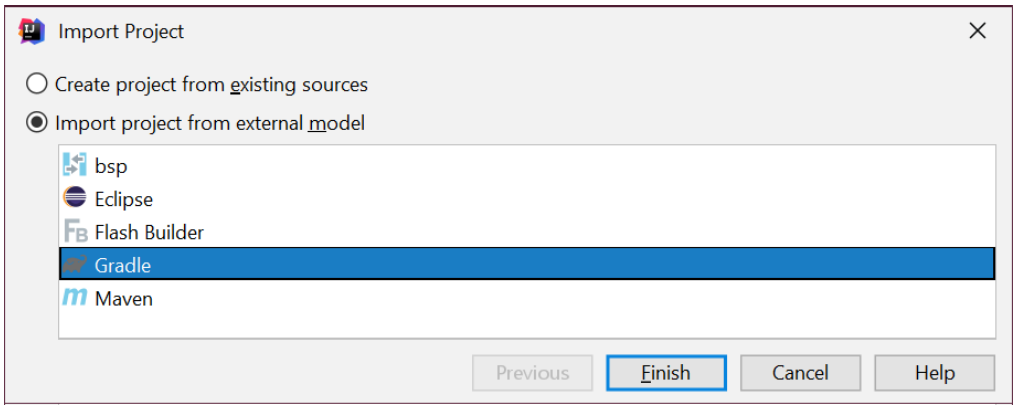
(Note: if you have already used IntelliJ IDEA, and opened some projects, then you could choose File > New > Project from Existing Sources… to open the same dialog for the next step.)

Step 3
Select the `bamboo/` directory, then click "OK".



Step 4

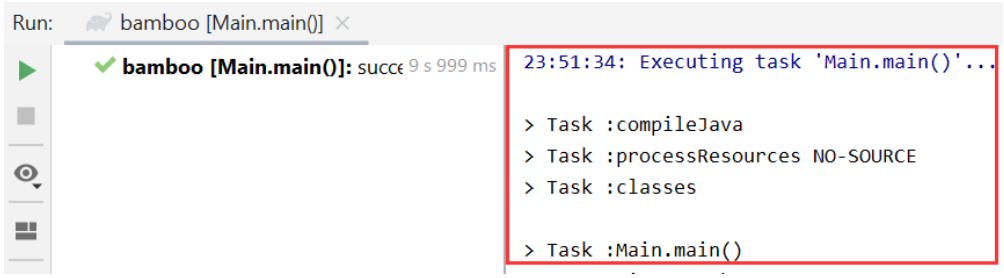
Import project from external model Gradle, then click “Finish”.



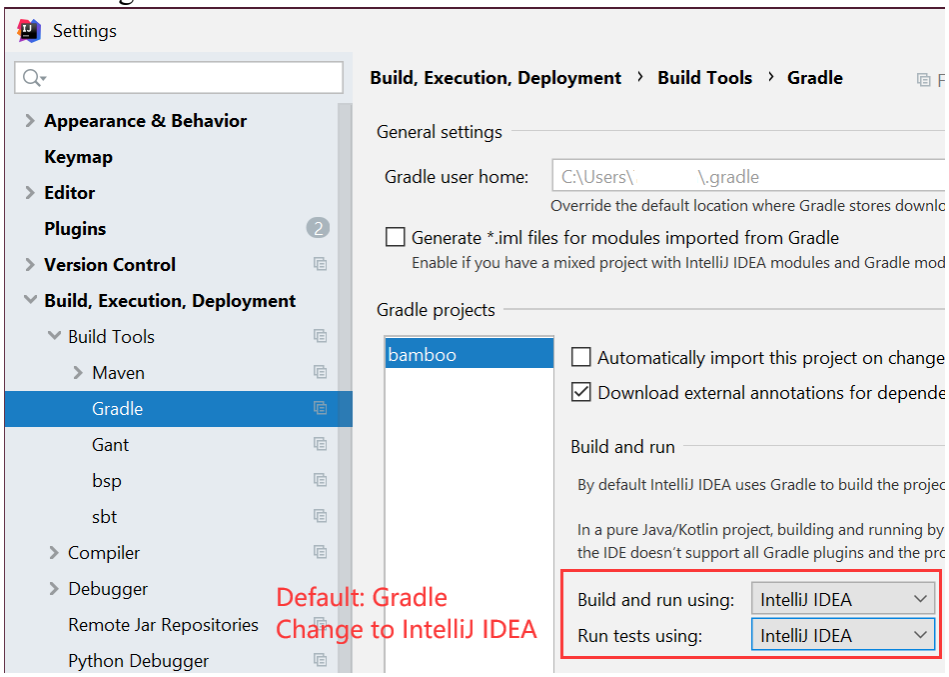
That’s it! You may wait a moment for importing Bamboo. After that, some Gradle-related files/folders will be generated in Bamboo directory, and you can ignore them.

Step 5

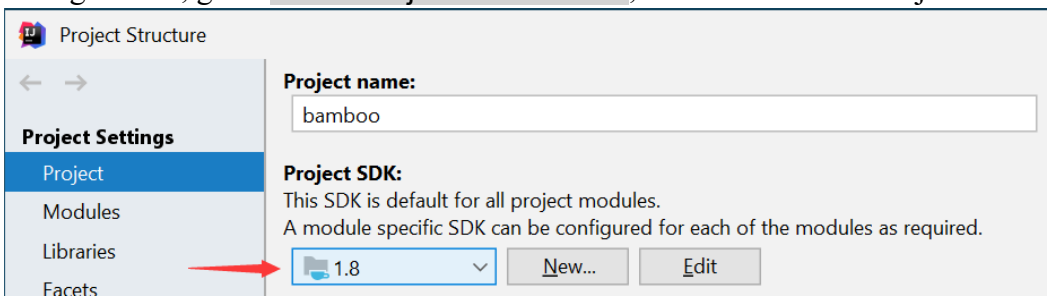
Since Bamboo is imported from Gradle model, IntelliJ IDEA always build and run it with Gradle, which makes it a bit slower and always output some annoying Gradle-related messages:



Thus, we suggest you disable the Gradle in IntelliJ IDEA. Just go to File > Settings, and change the *build and run* tool from Gradle to IntelliJ IDEA as shown:



Notice: If your system has multiple JDKs, make sure that IntelliJ IDEA uses Java 8 (otherwise you may experience `NullPointerException` thrown by Soot). To configure this, go to File > Project Sturcture…, and select **1.8** for Project SDK:



Alternatively, if you (really :-)) want to build Bamboo from command line, you could change working directory to Bamboo folder, and build it with Gradle:

```
$ gradle compileJava
```

3 Implementation of Context-Sensitive Pointer Analysis

This Section introduces the necessary knowledge about Bamboo and your task for this assignment. Note that Soot's Jimple IR is sophisticated and contains rich information, however, many of them are irrelevant to pointer analysis, and it is not that convenient to extract pointer-relevant information. To ease the implementation of pointer analysis, we have designed and implemented a new pointer analysis IR in Bamboo, which provides convenient APIs to obtain pointer-relevant information and excludes unnecessary details about the program statements. Our pointer analysis IR provides all information you need to implement pointer analysis, so in this assignment, you do not need to touch any Soot classes.

3.1 Scope

In this assignment, you will select contexts for method invocations and heap allocations, i.e., method contexts and heap contexts. You need to select contexts for both instance and static methods. We will introduce how to select contexts for static methods later.

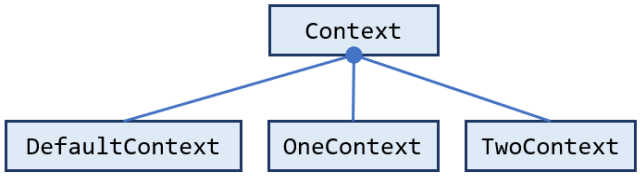
3.2 Bamboo Classes You Need to Know

To implement context sensitivity variants in Bamboo, you need to know the following classes.

- `bamboo.pta.analysis.context.Context`
  This interface represents contexts in context-sensitive pointer analysis.
  - `int depth()`: returns the depth, i.e., the number of elements of this context.
  - `Object element(int i)`: returns the i-th element of this context. The

indexes start from 1 (not 0).

This interface has three implementations, corresponding to the contexts with depth of 0, 1, and 2, as shown below.



- `bamboo.pta.analysis.context.DefaultContext`

  This class represents the default or empty context. This context is used for the entry methods. We implement it as an Enumeration, so that it can support singleton pattern. You could obtain the context instance via accessing the constant:

  ```
  DefaultContext.INSTANCE
  ```

- `bamboo.pta.analysis.context.OneContext<T>`

  This class represents contexts with 1 element, where `T` is the type of the element.

  - `OneContext(T)`: the constructor of this class, and its parameter is the context element.

  For example, you could construct a context for 1-call-site sensitivity and access its element in this way:

  ```
  CallSite cs = …
  OneContext<CallSite> ctx = new OneContext<>(cs);
  …
  CallSite e = ctx.element(1);
  ```

- `bamboo.pta.analysis.context.TwoContext<T>`

  This class represents contexts with 2 elements, where `T` is the type of the elements.

  - `TwoContext(T,T)`: the constructor of this class, and its parameter is the context elements.

  For example, you could construct a context for 2-object sensitivity and access its second element in this way:

  ```
  Obj o1 = …
  Obj o2 = …
  TwoContext<Obj> ctx = new TwoContext<>(o1, o2);
  …
  Obj e2 = ctx.element(2);
  ```

- `bamboo.pta.element.CallSite`

  This class represents call sites, which are also the context elements for call-site sensitivity.

- `bamboo.pta.element.Obj`
  This class represents abstract objects in pointer analysis, and they are also the context elements for object sensitivity.
  - `Method getContainerMethod()`: returns the method containing the allocation site of this method.

- `bamboo.pta.element.Type`
  This class represents types, which are also the context elements for type sensitivity.

- `bamboo.pta.element.Method`
  This class represents methods.
  - `Type getClassType()`: returns the class type which contains the declaration of this method.

- `bamboo.pta.analysis.data.CSCallSite`
  This class represents context-sensitive (C.S.) call sites in pointer analysis. Each C.S. call site consists of a context and a call site.
  - `Context getContext()`: returns the context.
  - `CallSite getCallSite()`: returns the call site.

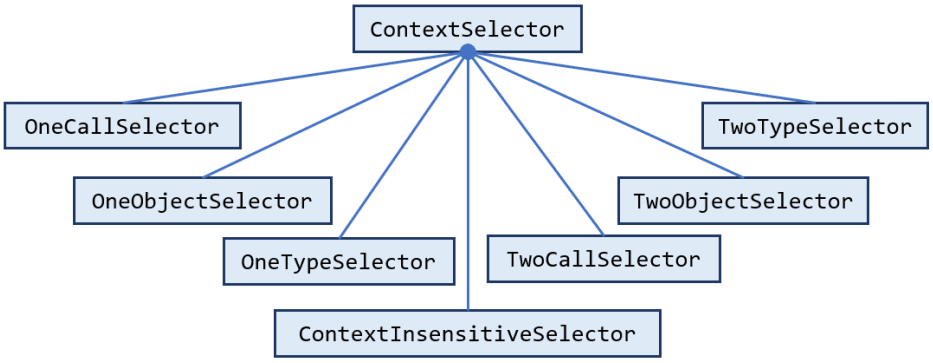
- `bamboo.pta.analysis.data.CSMethod`
  This class represents context-sensitive methods in pointer analysis. Each C.S. method consists of a context and a method.
  - `Context getContext()`: returns the context.
  - `Method getMethod()`: returns the method.

- `bamboo.pta.analysis.data.CSObj`
  This class represents context-sensitive (C.S.) objects in pointer analysis. Each C.S. object consists of a (heap) context and an object.
  - `Context getContext()`: returns the heap context.
  - `Obj getObject()`: returns the object.

- `bamboo.pta.analysis.context.ContextSelector`
  This is the interface between concrete context sensitivity variants (e.g., call-site sensitivity and object sensitivity) and context-sensitive pointer analysis framework. It has 4 APIs, which select contexts for entry methods (e.g., main method), instance methods, static methods, and heap objects, respectively. You could implement context sensitivity variants by implementing this interface.
  - `Context getDefaultContext()`: returns the context for entry methods. This API has default implementation, so you do not need to implement it.

- `Context selectContext(CSCallSite,Method)`: selects contexts for static methods. This API is called when pointer analysis handles static call. The first parameter is the context-sensitive call site, which contains the caller context and the call site. The second parameter is the callee.
- `Context selectContext(CSCallSite,CSObj,Method)`: selects contexts for instance methods. This API is called when pointer analysis handles instance call. The first parameter is the context-sensitive call site, which contains the caller context and the call site. The second parameter is the receiver object with its heap context. The third parameter is the callee.
- `Context selectHeapContext(CSMethod,Object)`: selects the heap contexts for abstract objects. This API is called when pointer analysis handles object allocation. The first parameter is the context-sensitive method, which contains the allocation site of the object, and the second parameter is the allocation site.

Currently, Bamboo provides 7 context sensitivity variants, at shown below:



`ContextInsensitiveSelector` is complete and the other six selectors are not. You will need to finish the six context selectors in this assignment, as explained in Section 3.3.

➢ `bamboo.pta.Main`
This is the main class of context-sensitive pointer analysis, which performs the analysis for input Java program. We introduce how to run this class in Section 3.4.

3.3 Your Task [Important!]

In this assignment, you need to finish three APIs (two `selectContext(…)` and one `selectHeapContext(…)`) of the six context selectors. Specifically, you will implement three common context sensitivity variants introduced in Lecture 12, i.e., call-site sensitivity, object sensitivity, and type sensitivity. For each variant, you need to finish two selectors with context limits of 1 and 2, respectively, (i.e., k-limiting context abstraction with k=1 and k=2). You also need to select heap contexts. For each k-limiting context selector, the limit of the heap contexts is k-1, e.g., for 1-call-site sensitivity, the limit of heap context is 0 (essentially no heap context), and for 2-call-

site sensitivity, the limit of heap context is 1.

To select method contexts, you need to implement

- `Context selectContext(CSCallSite,Method)`
- `Context selectContext(CSCallSite,CSObj,Method)`

of each selector, and to select heap contexts, you need to implement

- `Context selectHeapContext(CSMethod,Object)`

Below are the six context selectors you need to finish.

➢ `bamboo.pta.analysis.context.OneCallSelector`
This class implements 1-call-site sensitivity

➢ `bamboo.pta.analysis.context.OneObjectSelector`
This class implements 1-object sensitivity

➢ `bamboo.pta.analysis.context.OneTypeSelector`
This class implements 1-type sensitivity

➢ `bamboo.pta.analysis.context.TwoCallSelector`
This class implements 2-call-site sensitivity

➢ `bamboo.pta.analysis.context.TwoObjectSelector`
This class implements 2-object sensitivity

➢ `bamboo.pta.analysis.context.TwoTypeSelector`
This class implements 2-type sensitivity

❖ Hint 1: For how to implement call-site, object and type sensitivity, please refer to pages 113, 142, and 169 of the slides of Lecture 12 (on the course website).

❖ Hint 2: In call-site sensitivity, the context selection for static methods are the same as instance methods, i.e., at a static call, we add the call site to the caller context to compose the callee context. In object and type sensitivity, the convention of static methods is to *directly use the caller context* as the callee context. You should select contexts for static methods as described above.

❖ Hint 3: Note that k (for k-limiting context abstraction) is *the upper bound* of length of each context, so when k=2, you still need `DefaultContext` and `OneContext` for the contexts whose length is less than 2.

We have provided code skeletons for the above six context selectors, and your task is to fill the part with comment "`TODO - finish me`".

3.4 Run Context-Sensitive Pointer Analysis as an Application

As mentioned in Section 3.2, the main class of pointer analysis is

`bamboo.pta.Main`

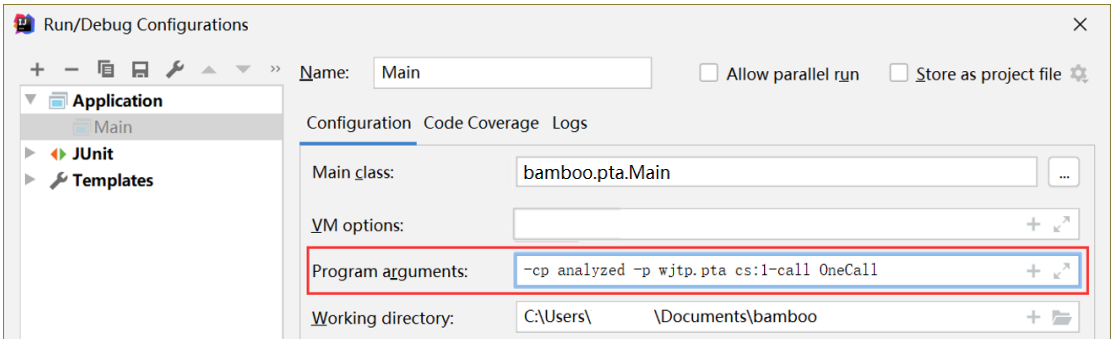
The format of its arguments is:

```
-cp <CLASS_PATH> -p wjtp.pta cs:<CONTEXT_SELECTOR> <CLASS_NAME>
```

`<CLASS_PATH>` is the class path, `<CONTEXT_SELECTOR>` is the name of the context selector. Bamboo supports 7 context sensitivity variants (selectors):

- `ci`: context insensitivity (complete and ready to use)
- `1-call`: 1-call-site sensitivity
- `1-obj`: 1-object sensitivity
- `1-type`: 1-type sensitivity
- `2-call`: 2-call sensitivity
- `2-obj`: 2-object sensitivity
- `2-type`: 2-type sensitivity

`<CLASS_NAME>` is the name of the input class to be analyzed. Bamboo locates input class from given class path. For example, to analyze the `OneCall.java` in class path `analyzed/` by 1-call-site sensitive pointer analysis, just configure program arguments in IntelliJ IDEA as follows:



For each input program, Bamboo performs pointer analysis, and outputs the process of points-to set propagation and the analysis results, including (1) reachable methods; (2) call graph edges; (3) points-to sets of all variables; (4) points-to sets of all instance fields. Note that in context-sensitive pointer analysis, the program elements are associated with their contexts. For example, a context-sensitive variable in 1-call-site sensitive pointer analysis is of format:

`[<class>(L<#line>):<callsite>]:<method>/<name>`

where `[…]` is the context, of which `<class>` is the class containing the call site, `L<#line>` is the line number of the call site, and `<callsite>` is the string representation of the call site. Besides, `<method>` is the method which the variable is declared in, and `<name>` is the variable name. You can use this information to help develop and debug. We encourage you to write some Java classes and analyze them.

Note that in this assignment package, the initial implementations of the six selectors all return `null`, so if you directly analyze some input classes, `NullPointerException` will be thrown. The exception will disappear after you implement them correctly.

Of course, you could also run the analysis using Gradle, with the following command:

```
$ gradle run --args="-cp <CLASS_PATH> -p wjtp.pta
cs:<CONTEXT_SELECTOR> <CLASS_NAME>"
```

3.5 Test Context-Sensitive Pointer Analysis with JUnit

To make testing convenient, we have prepared some Java classes as test inputs in folder `analyzed/`. Every class has an associated file named `*-expected.txt`, which contains the expected results of context-sensitive pointer analysis, i.e., the points-to sets of all variables and instance fields. You could analyze these test inputs by running test class (powered by JUnit):

```
bamboo.pta.PTATest
```

This test class analyzes all provided Java classes in `analyzed/`, and compares the given analysis results to the expected results. If your implementation of pointer analysis is correct, the tests will pass, otherwise it fails and outputs the differences between expected and given results.

Note that in this assignment, we have put all test cases in the package. So if your implementation passes all test cases, you could get full marks for this assignment.

Again, you could run tests with Gradle, just type:

```
$ gradle clean test
```

This command will delete the build directory, rebuild Bamboo, and run tests.

3.6 Run Interprocedural Constant Propagation

As mentioned in Sections 1 and 2, to demonstrate the usefulness of context-sensitive pointer analysis (it can build more precise call graph than CHA and context-insensitive pointer analysis), this assignment contains an interprocedural constant propagation, which leverages your implementation of context sensitivity variants (context selectors) to build call graph, interprocedural control-flow graph (ICFG), and performs constant propagation on the ICFG. The main class of the analysis is:

```
bamboo.dataflow.analysis.constprop.CSPTAMain
```

The format of its arguments is:

```
-cp <CLASS_PATH> -p wjtp.pta cs:<CONTEXT_SELECTOR> <CLASS_NAME>
```

We also provide a test case `OneCall.java` in `analyzed/`, which comes from Lecture 11. After you finish the context selectors, we recommend you run both pointer analyses based and CHA based constant propagation for the test case, to observe their analysis results and precision differences.

Note that before you run intra- and interprocedural constant propagation, please replace

`bamboo.dataflow.analysis.constprop.ConstantPropagation.java` in this Assignment package by your implementation for Assignment 2.

4 General Requirements

- In this assignment, your only goal is correctness. Efficiency is not your concern.
- DO NOT distribute the assignment package to any others.
- Last but not least, do not plagiarize. The work must be all your own!

5 Submission of Assignment

Your submission should be a zip file, which contains your implementation of

- `OneCallSelector.java`
- `OneObjectSelector.java`
- `OneTypeSelector.java`
- `TwoCallSelector.java`
- `TwoObjectSelector.java`
- `TwoTypeSelector.java`

The naming convention is of the zip file is:

`<STUDENT_ID>-<NAME>-A5.zip`

Please submit your assignment through 教学立方.

6 Grading

The points will be allocated for correctness. We will use your submission to analyze the given test files from the `analyzed/` directory, as well as other tests of our own, and compare your output to that of our solution.

Good luck!