CSCU9A3 Practical 1

Colin's Cycle Calculator: Part 1

Intro

In this practical class, we are going to learn how to import a project into Eclipse IDE and recap some Java concepts.

The Problem

Colin owns a small bicycle shop in Stirling, and has worked there for over 20 years. Over the years he has grown tired of the countless number of people who have asked him how long their particular cycle ride would take. He has written down a set of secret rules that he uses to estimate an answer and has come to you to turn his rules into a computer algorithm, as he'd like to offer this as a service on his shop website. Colin's rules are:

1. Firstly, gauge the cyclist's competency to work out their base average speed.
    - **Beginner: 10mph**
    - **Intermediate: 15mph**
    - **Advanced: 20mph**
2. Cycling alone or with someone else?
    - **Cycling together increases the cyclist's base average speed by 20%.**
3. The number of years of experience is crucial to the average speed, as cyclists build up appropriate muscles over the years.
    - **Increase the average speed by 0.2mph for every year experience.**
4. Finally, the weather has an impact on the average speed during the cycle.
    - **For every degree Celsius lower than 10, decrease the average speed by 0.1mph.**
    - **For every degree Celsius over 20, decrease the average speed by 0.1mph.**
    - **For every set of 15mph winds, decrease the average speed by 1mph.**
    - **If it is raining, decrease the average speed by 2mph.**

Knowing Colin's top-secret information, it is over to you to devise a way to calculate the estimated number of hours that a particular cycle would take.

Getting Started

A skeleton code, related to this practical, has been created for you. It can be downloaded on Canvas.

Copy these files to a new project directory and open them in your desired IDE. If you want to use Eclipse (we strongly recommend that you use it), check the section below to learn how to import projects into your workspace. Here, we are using Eclipse Version 2019-06 (4.12.0). You may use other version of Eclipse, however note that some configurations may change slightly.

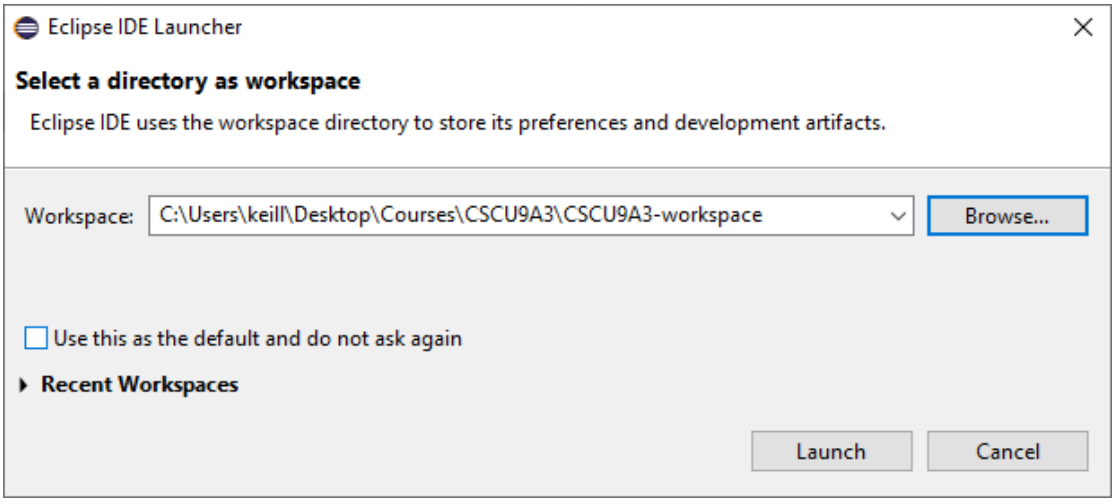
Installation

If you have already downloaded and installed Java and Eclipse, please go to the next section. Otherwise, we are using Java 11 SDK, that can be downloaded here: https://www.oracle.com/java/technologies/javase-jdk11-downloads.html. You'll need to register for a (free) Oracle account to make the download, if you don't have one already.

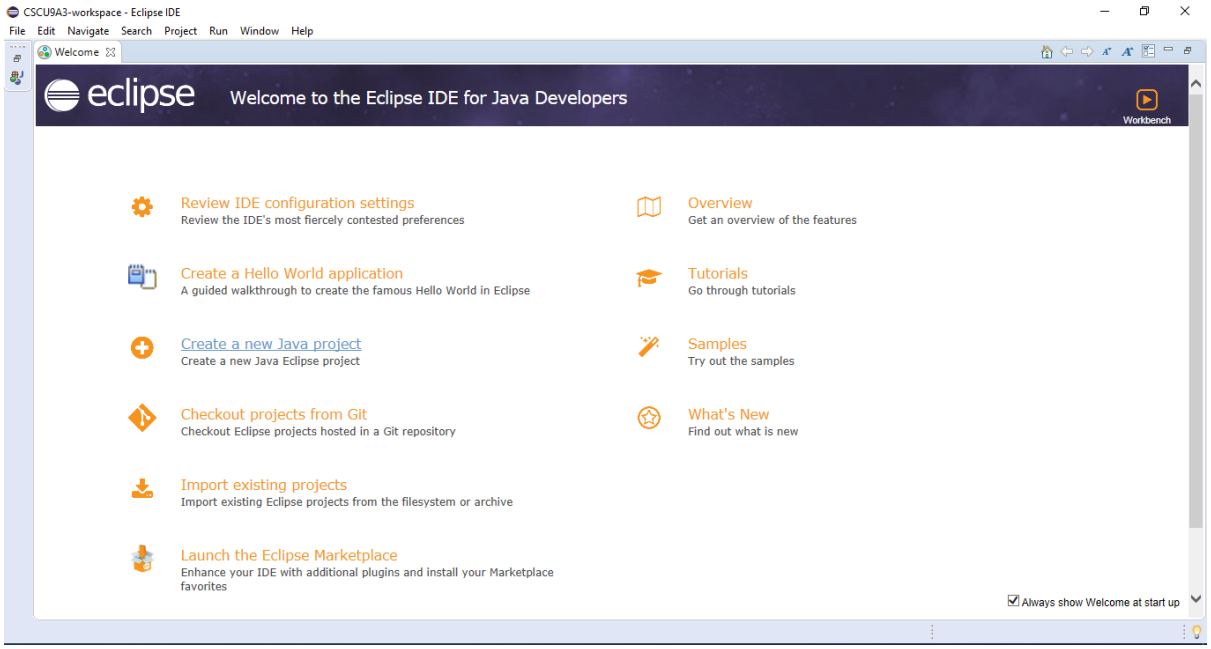
To download Eclipse, use this link: https://www.eclipse.org/downloads/. When you run the installer, select "Eclipse IDE for Java Developers".

Eclipse

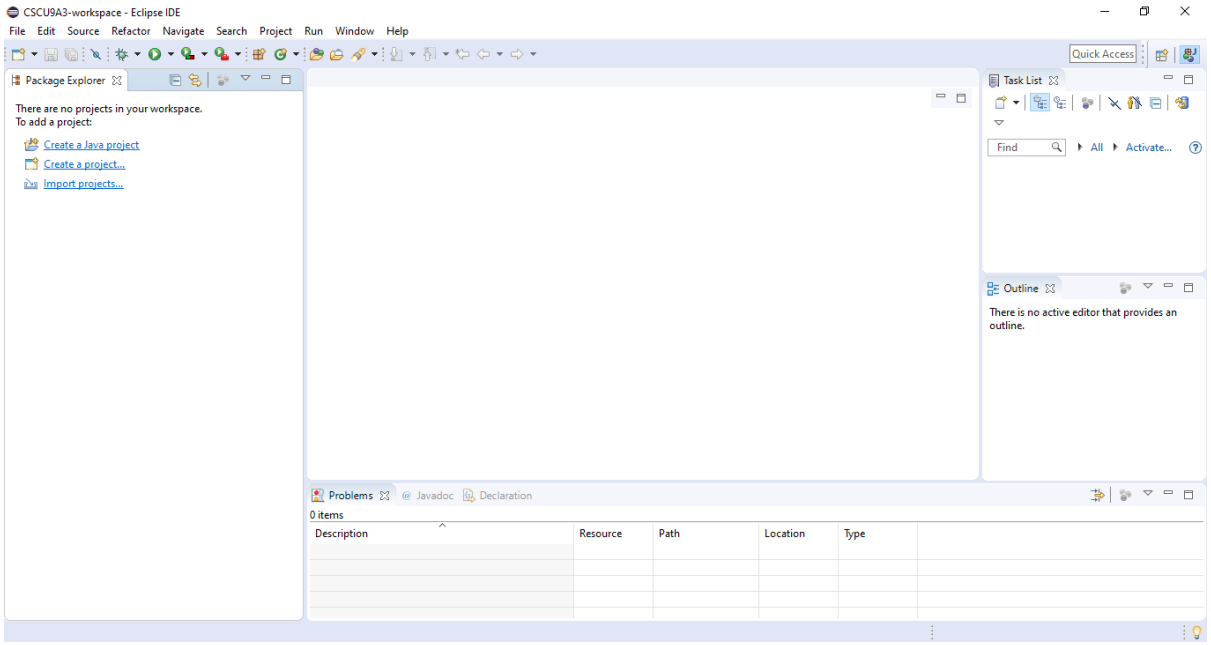
Let's import your code so we can start. When you first open the Eclipse IDE, it will ask you about your workspace, which is a directory that will be used to store all your codes produced in Eclipse. You may choose any directory you want, but in your case, a good practice would be to create a folder for each course. For example, you can create a folder C:\Users\YourName\CSCU9A3\ to store all code produced in our course. **Remember this folder, because every time you open Eclipse, you will need to inform it.**



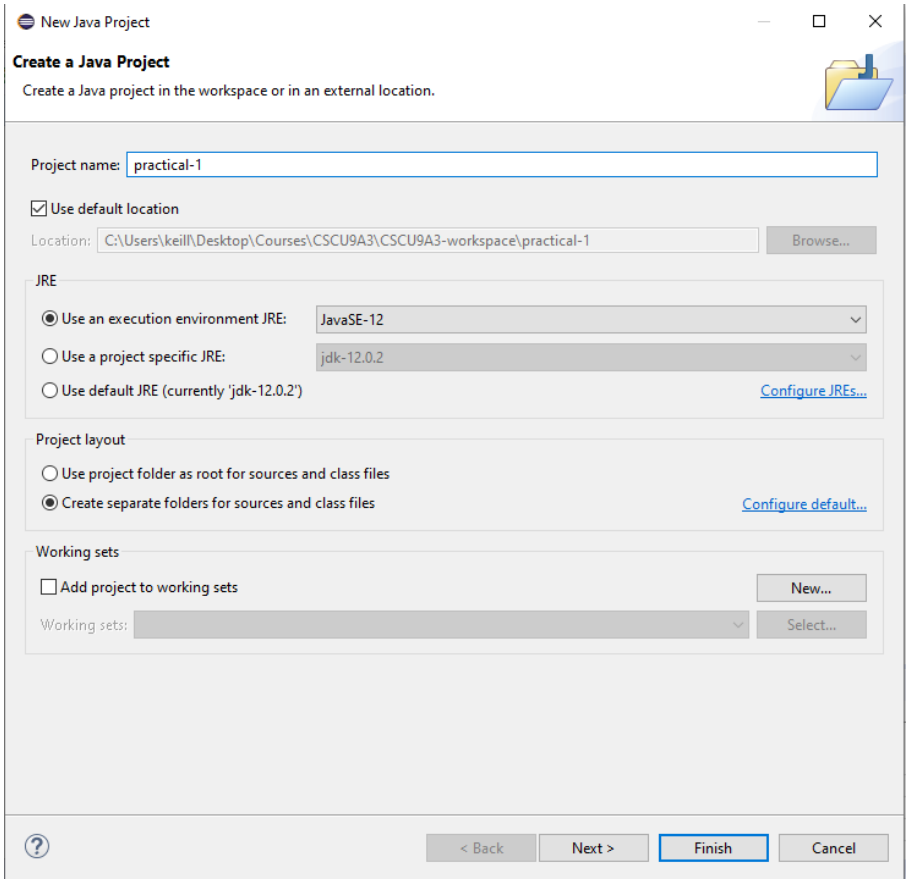
When you select the directory and launch the IDE, you may see something like this:



This is just a welcome window. You can close it and finally access the IDE, which should have no projects and will probably look like this:

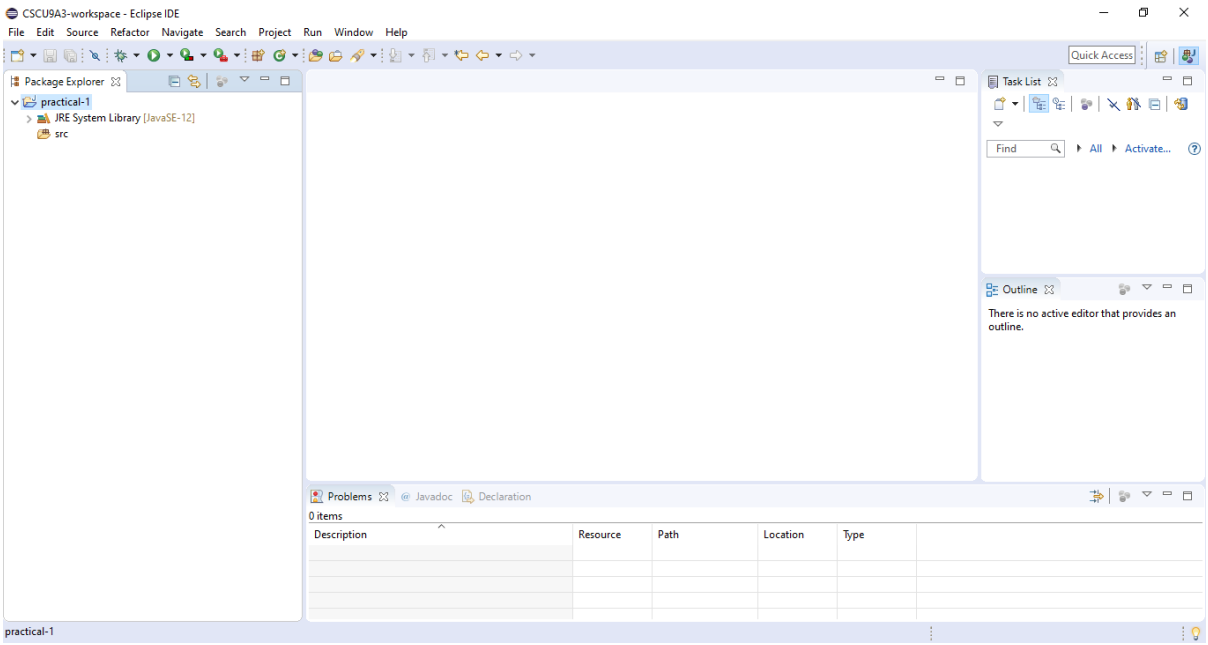


You can create or import JAVA projects now. In this case, we will create a new project. So, click on “Create a Java project”, named it practical-1, and click on “Finish”:

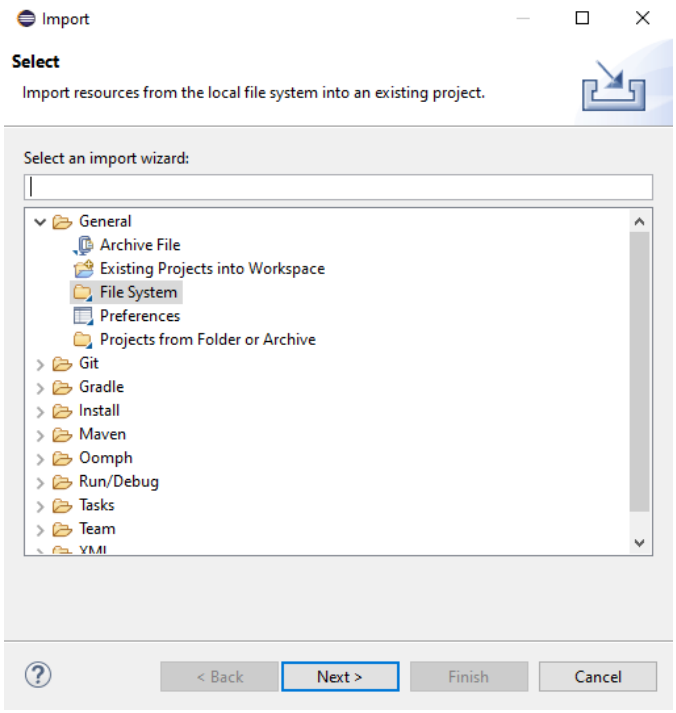


Sometimes, you may see a window asking for the Module Name. In this case, you can just click on "Don't Create".

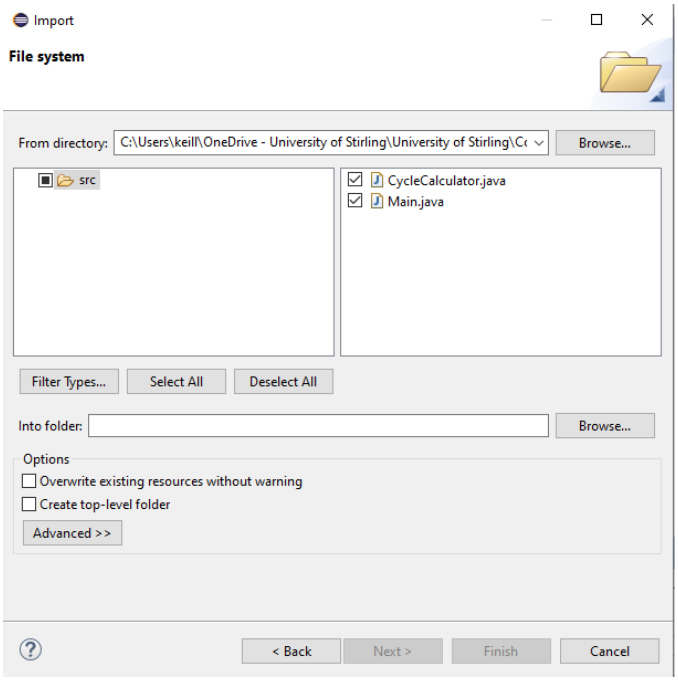
Your project should be created by now and your workspace should look like this:



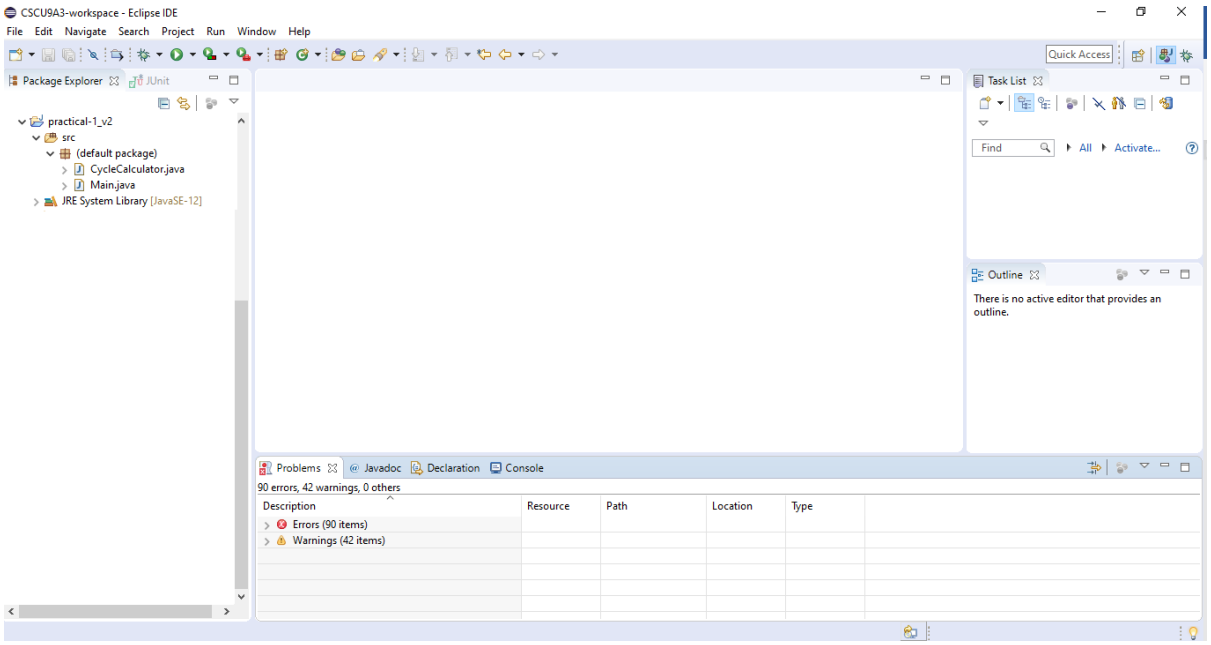
Let's now import the Java codes of this practical to our project. Right click over the src directory and then click on "Import…". On the import window, select "File System":



Browse to the aforementioned skeleton directory and select the classes to import:



Now, your workspace should be like this:



Now that we have set our workspace, we can start working on the practical.

First Version

This code has two classes: `Main` and `CycleCalcultor`. The former has the main method and can be used to "test" your code. Each print within this main method is related to a "test". If the print outputs `false`, then your code is not working properly. If it displays `true`, then your code is fine. In the next week, we'll see a better way of doing this test process. The later has only method headings and you are responsible for implementing them. The first one you need to implement is the method with the following heading:

```
public double getDuration(int numMiles, String competency, int
numYearsExperience, boolean cyclingAlone, int temp, int windSpeed,
boolean isRaining)
```

Fill in the logic within this method and return Colin's estimated number of hours that the cycle will take.

Notes

- Duration (hours) = distance (miles) / speed (mph)
- To increase x by 20%: x = x*1.2

Do not be tempted to jump straight onto the computer and start coding. Use the techniques taught to you in lectures to break the problem down on paper and produce pseudo-code to solve the problems. Only then, once you are happy with your solution on paper, should you move onto coding the cycle calculator.

Constants

Can you imagine if Colin decided to tweak some of his values in his rules? It would require you, as the programmer, to dive into the code and work out which numbers to change. This isn't particularly nice is it? When programming you can have **constants**, which are properties whose values do not

change. These are typically declared, and values assigned, at the top of a class. In Java we would write:

```
private final int BEGINNER_SPEED = 10;
```

`final` is used to denote that the property value will never change, showing that the property is a constant. You'll notice that the property name was written in uppercase; this is common practice in many programming languages where constants are written in full uppercase with words separated by an underscore.

Go through your code and refactor any constants that exist and appreciate exactly why this makes your code a lot better (constants for low and high temperature thresholds are quite useful). Ask if you don't know.

[Checkpoint]

When you finish programming and all the "tests" pass you can be happy that your solution is correct. But before you call over a demonstrator, read through your code, tidy up any dead unused code, add comments where necessary and ensure that your indentations are correct. The checkpoint will be granted only with well-formatted code that works.

Calculating total time: arrays

If a person wants to calculate the total time of several bike rides he has made in the last years, he will not be able to do so using your code. So now you will program your code to perform this type of calculation.

The ride times will be store in arrays. A method has been provided for you to fill in the above method, which receives an array of durations (as `double` values) and must return the total of all these durations added together (as a `double`):

```
public double getTotalDuration(double[] durations)
```

In order to check this method, uncomment the last lines of the `main` method. These lines are responsible for checking the `getTotalDuration` method.

Time Format

Calling our `getDuration` method returns the time in a horrible format. For example 1.75 hours would be far nicer written as "1 hour and 45 mins". Create an appropriate method within CycleCalculator class and get it to return a nicely formatted time as a String. The method signature would look something like:

```
public String getFormattedDuration(double time)
```

There is **no** "test" provided for this, so add 3 additional tests into the `main` in order to "test" this method. You should aim to try cases where the duration is exactly on an hour boundary (1.99, 2.01, 2.00, …) and ones where it is not. Hint: You may find the Math.floor method and the modulus operator (%) useful here.

[Checkpoint]

When you finish programming and all the "tests" pass you can be happy that your solution is correct. Call over a demonstrator to check your code.

Java Reminders

Variables

```
int myInteger;
int myInitialisedInteger = 25;
double bigFloatingPointNumVar = 6.21;
String niceString = "Awesome";
```

Conditionals

```
if (<condition>) {
      //do things!
}
```

<condition> examples:

- `speed < 20`
- `cyclingAlone == false` (or `!cyclingAlone`)
- `isRaining == true` (or `isRaining`)
- `competence.equals("Beginner")`

(notice for string comparisons we use '`.equals`' instead of '`==`')

Loops

For

```
for (initialise; condition; step) {
        // do stuff
}
```

Example: a loop that will run 10 times

```
for (int index = 0; index < 10; index++) {
        //do stuff
}
```

While

```
Initialise;
while (condition) {
        //do stuff
        Increment;
}
```

Example: a loop that will run 10 times

```
int index = 0;
while (index < 10) {
        //do stuff
```

```
        Index++;
}
```