Background

We advocate our learners to evolve their functions with the help of test cases. It will be interesting if they develop a code generator for themselves, that helps them generate the skeleton code for their work.

Let us try to understand this with an example.

Manuj is a software developer. Vikram (A researcher of mathematics) approached Manuj to develop a JAVA library that provides a set of mathematical functions.

Vikram helped Manuj to understand the algorithm for different mathematical functions, and provided a set of acceptance criteria for different functions.

Vikram told Manuj, that if your function gives the desired result on given inputs, I will consider this an acceptable implementation, and I will pay you the amount due for your work.

Let's take a look at the factorial function, that Vikram asked Manuj to implement

Acceptance Criterion

Function Name: factorial

Input (number)	Output (Value of Factorial)
9	362880
6	720
3	6
1	1
0	1

Manuj has an amazing way of working, he works in the backward direction. He will implement the skeleton working code first, and then he will keep on improving his code, till all the examples that he has an acceptance criterion works as expected.

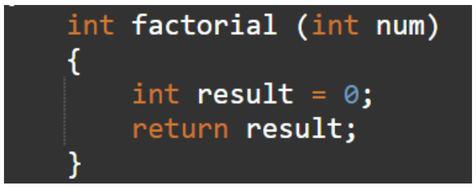
So Manuj will make a note of these things first

- What is the function name?
- What all inputs does this function accept and what are their types?
- What type of value does this function return?

To understand this better let's take the example of factorial function

- What is the function name?
    - “**factorial**”
- What all inputs does this function accept and what are their types?
    - Function receives 1 input, it is named “**num**” and it returns a value of type “**int**”
- What type of value does this function return?
    - “**int**”

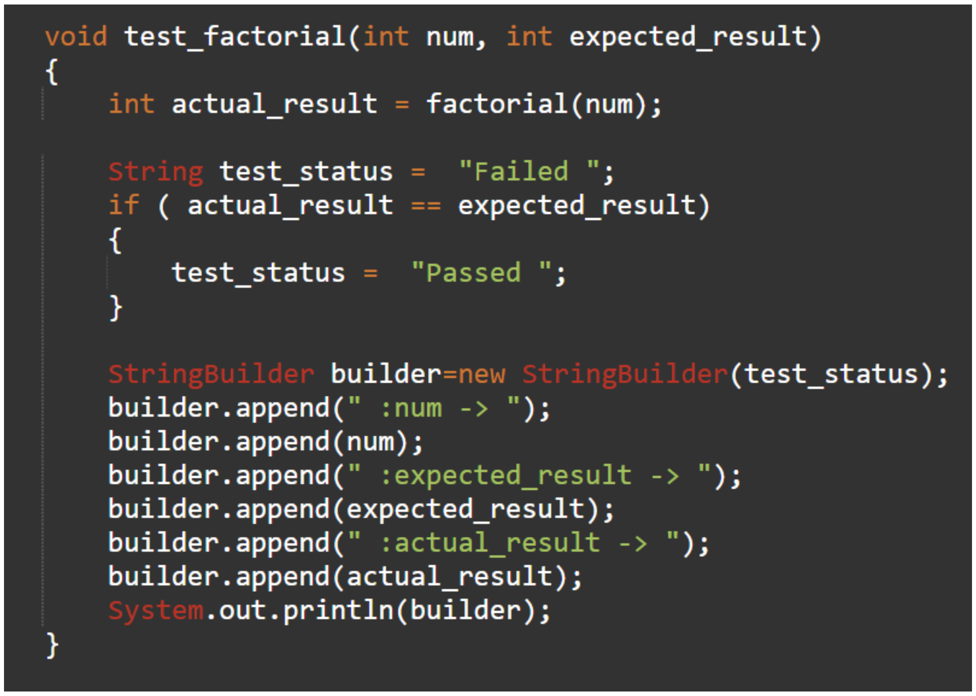
Now he would generate the skeleton code like this for the function to be implemented.



```
int factorial (int num)
{
    int result = 0;
    return result;
}
```

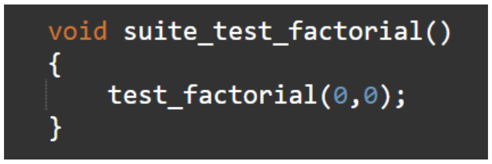
Please note that this function has no compilation error, although it does not generate correct output, as it always returns 0, irrespective of the input

Then he would implement a function to test this function, like this



```
void test_factorial(int num, int expected_result)
{
    int actual_result = factorial(num);

    String test_status =  "Failed ";
    if ( actual_result == expected_result)
    {
        test_status =  "Passed ";
    }

    StringBuilder builder=new StringBuilder(test_status);
    builder.append(" :num -> ");
    builder.append(num);
    builder.append(" :expected_result -> ");
    builder.append(expected_result);
    builder.append(" :actual_result -> ");
    builder.append(actual_result);
    System.out.println(builder);
}
```

And finally he would create a skeleton function to call the test_factorial function with default values for different data types.



```
void suite_test_factorial()
{
    test_factorial(0,0);
}
```

Objective

Help Manuj by giving him a code generation tool, that generates the skeleton functions, which manuh can enhance to do his work faster

Acceptance Criterion

Your tool must accept following inputs from the user

Function Name

Name of the function that our tool user wants to implement (**factorial** in the example you saw in background)

Function Return Type

Type of value the function needs to return (**int** in the the example you saw in background)

Argument Details

Type and name of the argument the function accepts (**int** is the type and **num** is the name you saw in background)

Constraints

Permissible types

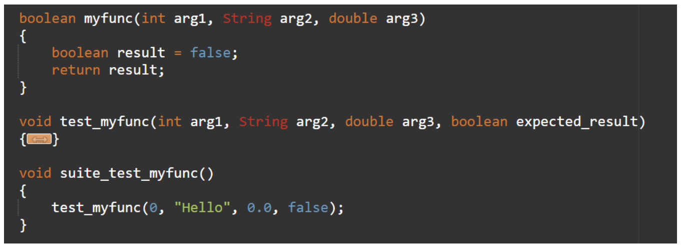
Types: “int”, “double”, “boolean”, “String”

Default values for different types

Values: “int -> 0”,   “double -> 0.0”, “boolean -> false”, “String -> “hello””

NOTE: When we create the function, we need to set the result to the default value for the function return type, also when we are calling the test function from suite_test we need to make a call with default values of the function arguments and function return type.

For example, if a function myfunc accepts an integer, a string, and a double as input and returns a boolean as output, here is how the skeleton function will look like

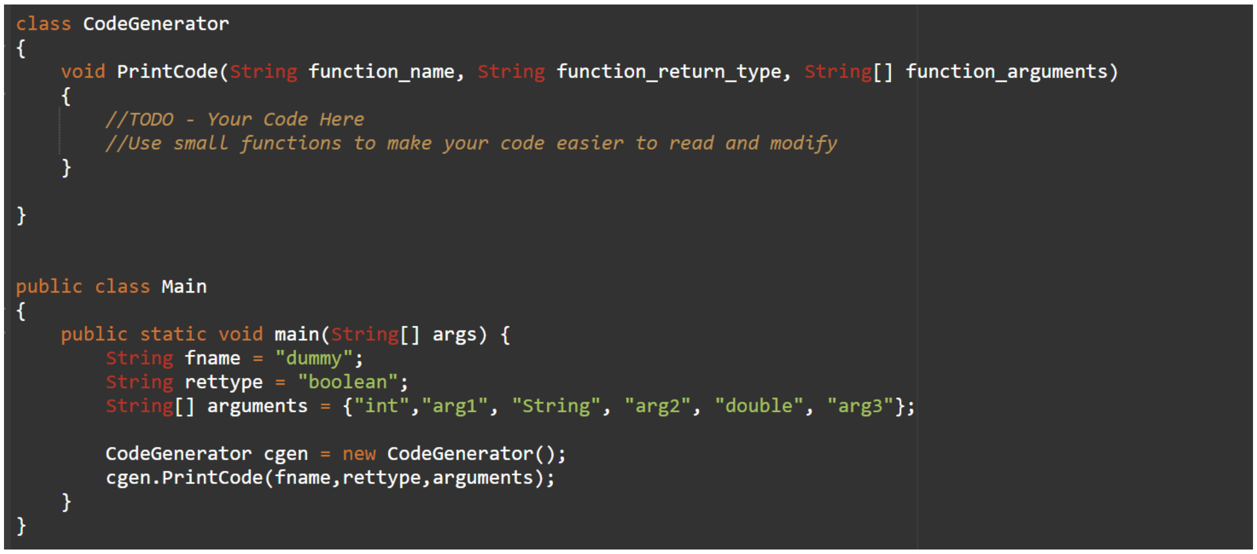


```
boolean myfunc(int arg1, String arg2, double arg3)
{
    boolean result = false;
    return result;
}

void test_myfunc(int arg1, String arg2, double arg3, boolean expected_result)
{⇔}

void suite_test_myfunc()
{
    test_myfunc(0, "Hello", 0.0, false);
}
```

Implementation Steps

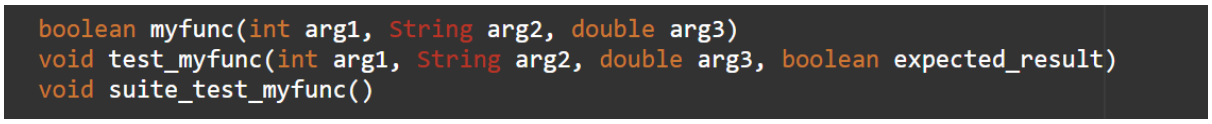
Step 1

Just write the code to print function signatures



```java
class CodeGenerator
{
    void PrintCode(String function_name, String function_return_type, String[] function_arguments)
    {
        //TODO - Your Code Here
        //Use small functions to make your code easier to read and modify
    }

}


public class Main
{
    public static void main(String[] args) {
        String fname = "dummy";
        String rettype = "boolean";
        String[] arguments = {"int","arg1", "String", "arg2", "double", "arg3"};

        CodeGenerator cgen = new CodeGenerator();
        cgen.PrintCode(fname,rettype,arguments);
    }
}
```

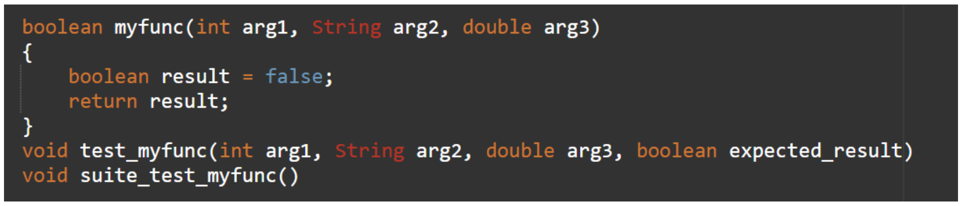
Your code must print



```java
boolean myfunc(int arg1, String arg2, double arg3)
void test_myfunc(int arg1, String arg2, double arg3, boolean expected_result)
void suite_test_myfunc()
```

Step 2

It's time to print the body of the function myfunc.
Your code should now print

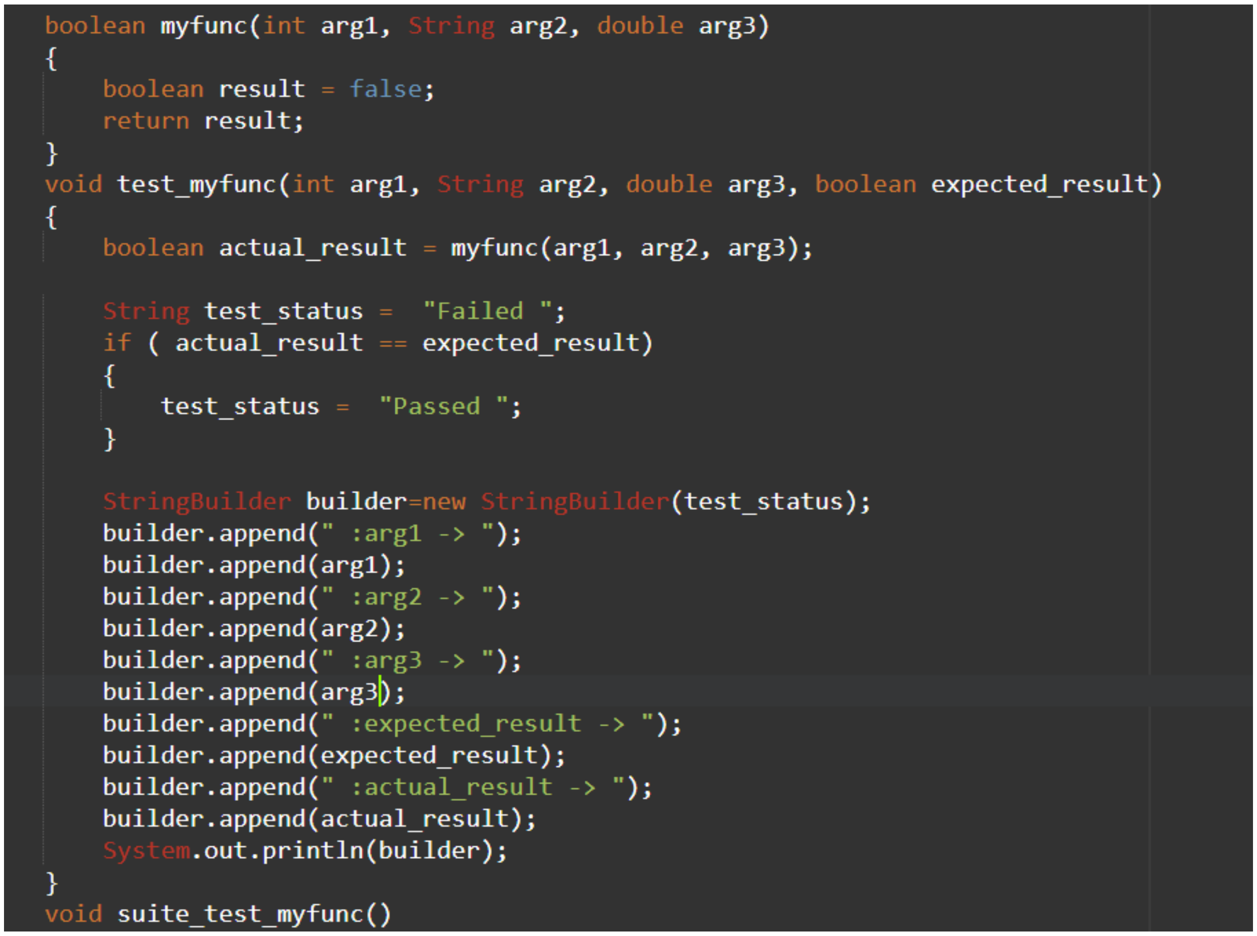


```java
boolean myfunc(int arg1, String arg2, double arg3)
{
    boolean result = false;
    return result;
}
void test_myfunc(int arg1, String arg2, double arg3, boolean expected_result)
void suite_test_myfunc()
```

Note that we are setting the result to the default value of the type that this function returns. In this case **boolean**

Step 3

We must now change our code to print the body of the test_myfunc.

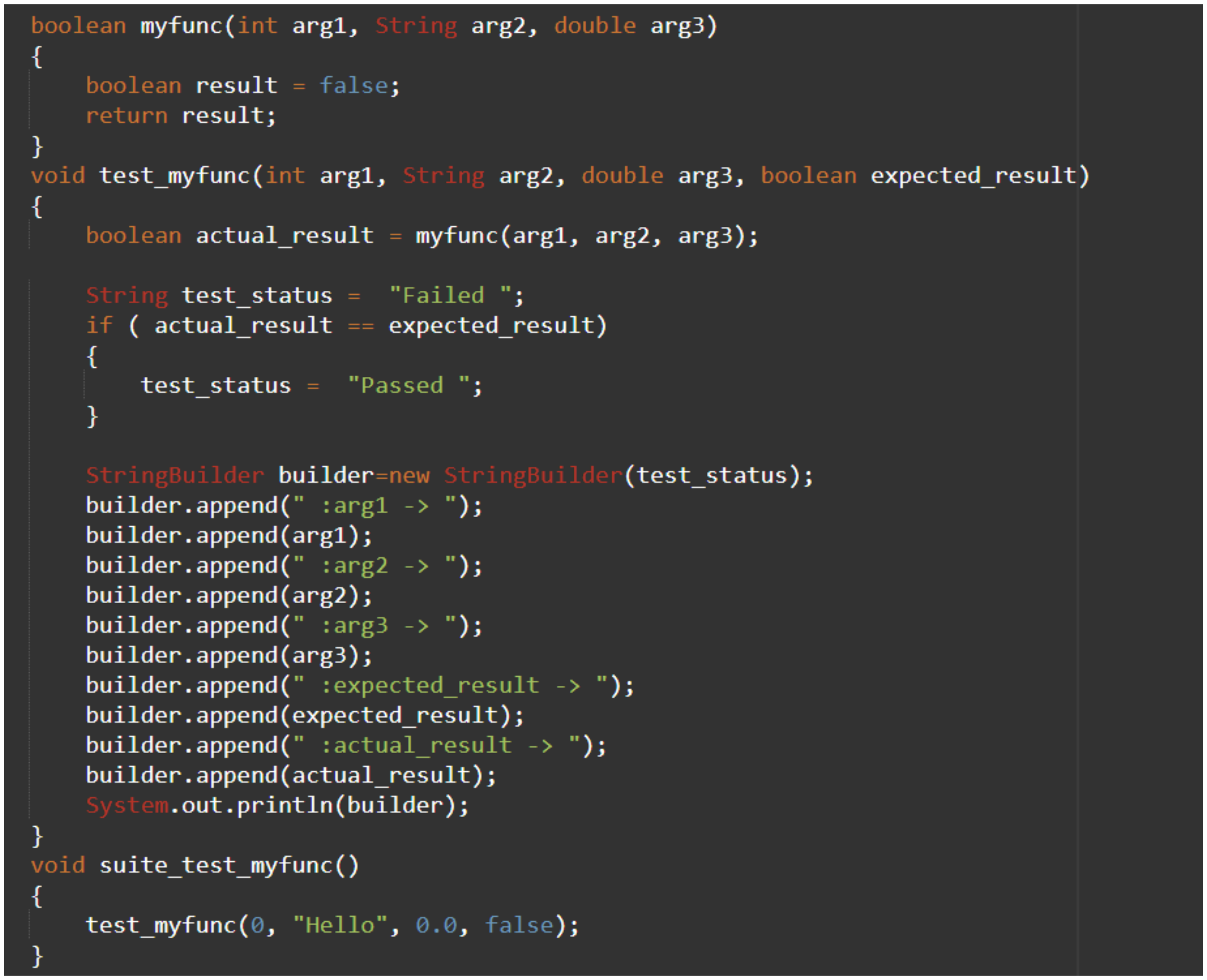


```
boolean myfunc(int arg1, String arg2, double arg3)
{
    boolean result = false;
    return result;
}
void test_myfunc(int arg1, String arg2, double arg3, boolean expected_result)
{
    boolean actual_result = myfunc(arg1, arg2, arg3);

    String test_status =  "Failed ";
    if ( actual_result == expected_result)
    {
        test_status =  "Passed ";
    }

    StringBuilder builder=new StringBuilder(test_status);
    builder.append(" :arg1 -> ");
    builder.append(arg1);
    builder.append(" :arg2 -> ");
    builder.append(arg2);
    builder.append(" :arg3 -> ");
    builder.append(arg3);
    builder.append(" :expected_result -> ");
    builder.append(expected_result);
    builder.append(" :actual_result -> ");
    builder.append(actual_result);
    System.out.println(builder);
}
void suite_test_myfunc()
```

- Note that **test_myfunc** has one extra argument **expected_result** that is of same type, that the function **myfunc** returns (**boolean**)
- We create a variable named **actual_result** of type that the function **myfunc** returns (**boolean**)
- We call the function **myfunc** with the required arguments and store its return value in **actual_result**
- We append all the argument names to our string builder

Step 4

In this final step we will also print the body of **suite_test_myfunc**



```
boolean myfunc(int arg1, String arg2, double arg3)
{
    boolean result = false;
    return result;
}
void test_myfunc(int arg1, String arg2, double arg3, boolean expected_result)
{
    boolean actual_result = myfunc(arg1, arg2, arg3);

    String test_status =  "Failed ";
    if ( actual_result == expected_result)
    {
        test_status =  "Passed ";
    }

    StringBuilder builder=new StringBuilder(test_status);
    builder.append(" :arg1 -> ");
    builder.append(arg1);
    builder.append(" :arg2 -> ");
    builder.append(arg2);
    builder.append(" :arg3 -> ");
    builder.append(arg3);
    builder.append(" :expected_result -> ");
    builder.append(expected_result);
    builder.append(" :actual_result -> ");
    builder.append(actual_result);
    System.out.println(builder);
}
void suite_test_myfunc()
{
    test_myfunc(0, "Hello", 0.0, false);
}
```

Note that we are calling the function test_myfunc with default values for the argument type it expects

Congratulations! You have your first working code generator ready. Do not forget to share what all you have learnt in the process with your fellow learners.