CMPSC461 Fall-2024
Programming Language Concepts

Instructor: Dr. Suman Saha

Assignment-4 (Function and Parameter Passing)

General Instructions:
You need to submit your homework to **Gradescope**. Do **every problem on a separate page and mark** them before submitting. **If the questions are not marked** or are submitted with incorrect page-to-question mapping, the question will be **deducted partial points**. Make sure your name and PSU ID are legible on the first page of your assignment.

You are required to submit your assignments in typed format, please follow the latex/doc template (which can be found on Canvas) for the homework submission. Furthermore, please note that no handwritten submissions (of any form on paper or digital) will be accepted.

**(Kindly refer to the syllabus for late submission and academic integration policies.)

Assignment Specific Instructions:

1. The sample examples provided in the questions below are just for your reference and do not cover every possible scenario your solution should cover. Therefore, it is advised you think through all the corner cases before finalizing your answer.

2. Students are expected to answer the questions in a way that shows their understanding of the concepts rather than just mentioning the answers. The rubric does contain partial points to encourage brief conceptual explanations.

Problem 1: Parameter Passing Exercise [20 pts]

Consider the following pseudocode:

```
x=1, y=3, z=5
function foo(a, b):
  x = x + b;
  z = x + y;
  a = a - z;
  y = a - x + b;
```

1 [16pts] For each of the cases below, write down the values of x, y, and z after the following calls to foo(). If necessary, assume that output arguments are copied back to parameters in the left-to-right order.

(a) foo(x,y) where all parameters are passed by value.

(b) foo(x,z) where all parameters are passed by reference.

(c) foo(y,z) where all parameters are passed by value-result.

(d) foo(z,z) where all parameters are passed by reference.

2 [4pts] Passed-by-result parameter passing scheme is not applicable for the above program – justify this statement with proper example.

Problem 2: Parameter Passing II [4 + 4 = 8 pts]

Like Call by Name parameter-passing mechanism, *Call by Macro Expansion* technique also uses a lazy evaluation technique. It is widely used in C, C++, etc. Unlike function calls, macros do not have runtime overhead because they are expanded at compile-time (before actual code execution). Macro expansion works in the following way:

- **No evaluation:** The literal text of each macro argument is substituted for the corresponding parameter in the macro's body.
- **No evaluation:** The resulting macro body is then textually inserted into the program where the macro was called.
- **Evaluation:** The expanded macro code is executed in the caller's environment, meaning the variables and expressions are evaluated as if they were part of the original calling code.

Below are two functions that implement the Call by Macro scheme. Each function accepts a single parameter x, which can either be a numerical value (-1, 2, 0, etc.) or an arithmetic expression consisting of numbers, and the arithmetic operators from set $\{+, -, *, /\}$.

```
\\ Function 1: Print the Square of a Number
function square(x) {
    printf(x * x);
}
\\ Function 2: Print the Value of x
function print(x) {
    printf(x);
}
```

For each function, indicate whether there will be any issues when using the Call by Macro parameter-passing scheme. If issues exist, explain why they occur and suggest how to fix them. Otherwise, explain why there is no issue.

Note 1: You should not consider changing the parameter-passing scheme as a potential fix.
Note 2: Check related exercises from this link to learn more about macro expansion (collected from the optional reading material section of Module Week 6).

Problem 3: Recursion [3 + 6 = 9 pts]

Consider the following implementation of function sum:

```
def sum(n):
    if n == 0:
        return 0
    else:
        return n + sum(n - 1)
```

1. Briefly explain why tail-recursive functions are useful.
2. Give a tail-recursive implementation for the function **sum**. You might need to define a helper function.

Problem 4: Exception Handling [3 + 10 = 13 pts]

Consider the following code snippet with exceptions. Note that the program execution starts with `main`:

```java
public class TryCatch {

    static class CustomException extends Exception {
        public CustomException(String message) {
            super(message);
        }
    }

    public static void main(String[] args) {
        try {
            methodA();
            System.out.println("D");
        } catch (CustomException e) {
            System.out.println("Outer catch: " + e.getMessage());
        }

        try {
            methodB();
            System.out.println("E");
        } catch (CustomException e) {
            System.out.println("Catch in main for methodB: " + e.getMessage());
        }
    }

    public static void methodA() throws CustomException {
        try {
            methodB();
            System.out.println("C");
        } catch (CustomException e) {
            System.out.println("Catch in methodA for methodB: " + e.getMessage());
            throw new CustomException("Exception propagated from methodA");
        }
    }

    public static void methodB() throws CustomException {
        throw new CustomException("Exception from methodB");
    }

    public static void methodC() throws CustomException {
        throw new CustomException("Exception from methodC");
    }
}
```

1. Write down what will be the output by the given program and briefly justify your answer.

2. Modify your code in a way that it prints the following messags in order (propagate the exception raised by methodC through methodA):

   (a) Catch in methodA for methodB: Exception from methodC via B

(b) Catch in main: Exception propagated from methodA