Soyeon Kim, 106292803

Homework 6

CS 131, Fall 2024
Carey Nachenberg

DATA9: Binding Semantics, Parameter Passing, Pass By Need

Part A

baz
bar
1

Part B

baz
bar
4

Part C

baz
bar
1

Part D

bar
1

FUNC1: Default Parameters

1. We can allow passing underscores to the arguments.
   For example, `result := addNumbers(_, 20, _)`
2. Pass positional arguments first, and then pass optional arguments with names after a separator.
   For example, `result := addNumbers(20 \ x: 5)` (\ is used for separator)

FUNC2: Lambdas, Closures

It seems that the `callLambda` function has access to the `counter` scope. Closures (`counter` in this case) retain access to their lexical scope, where the function is called, and it's passed by reference.

FUNC3: Error Handling, Optionals, Exceptions

Using the `Optional` struct is more suitable for AP because it's a no-throw guarantee. Since it's not throwing exceptions, it could have better performance. However, the user has to manually

check whether the return value is `nullptr` or not, and handle it separately. Using C++'s native exception handling could be beneficial if a failure to find the element genuinely represents an unexpected behavior and exceptional result, but using `Optional` is generally preferable.

FUNC4: Results, Optionals, Errors, Exceptions

Part A

Result Object because the reason for malformation is not important.

Part B

Assertion to ensure any invalid configuration which could potentially lead to system failure.

Part C

Assertion to ensure input sizes are within the range.

Part D

Exceptions to allow retries and provide the reason for the failure if it eventually failed.

FUNC5: Exceptions

- `bar(0);`
  catch 2
  I'm done
  that's what I say
  Really done!
- `bar(1);`
  catch 1
  hurray!
  I'm done!
  that's what I say
  Really done!
- `bar(2);`
  catch 1
  hurray!
  I'm done!
  that's what I say
  Really done!
- `bar(3);`
  catch3
- `bar(4);`
  hurray!
  I'm done!
  that's what I say
  Really done!