Prof. Jingke Li (lij@pdx.edu), Classes: TR 1400-1550 @OND 218, Office Hours: TR 1300-1400 (in-person @FAB 120-06). TAs: Nicholas Coltharp (`coltharp`), M 2-4pm @Fishbowl; Laura Israel (`laisrael`), W 10-11 @Fishbowl, F 1-2pm (Zoom: 761-457-3865)

Exercise 6: Functional Programming

Due: Monday 11/18/24

Learning Objectives

Upon successful completion, students will be able to:

- Write functional-style programs using higher-order functions
- Understand the performance advantages of tail-recursive functions

Grading

This exercise carries a total of 5 points. The grading will be based on effort — if you show a good-faith attempt at solving a problem, you'll get the credit.

Problems

1. Write a Python function `currying(f)` that takes an arbitrary binary function `f` as input and returns a curried version of `f`. For example:

   ```
   def add(x,y): return x + y  # a normal function
   add(2,3)                    # => 5
   currying(add)               # turning add into a curried func
   currying(add)(2)(3)         # => 5
   ```

2. Write a Python function `ktimes(f,k)` that takes a function `f` and an integer `k`, and returns a function that applies the function `f` to its argument `k` times. For example:

   ```
   ktimes(f,0)(x) = x
   ktimes(f,2)(x) = f(f(x))           # this case is the same as twice(f)
   ktimes(f,5)(x) = f(f(f(f(f(x)))))
   ```

   If `f` is the `incr` function: `def incr(x): return x+1`, and `x=5`, then the above three calls would produce `5,7,10`, respectively.

3. Python has a map function, `map(f,itr)`, which takes a function `f` and an iterable object `itr`. It applies `f` to each item of `itr` and returns the result as a map object:

   ```
   >>> map(lambda x:x+1, [1,2,3])
   <map object at 0x6fffffdab460>
   ```

   The map object is also an iterator, but its content is not directly visible. To see its content we can convert it into a visible object, such as a list:

   ```
   >>> x = map(lambda x:x+1, [1,2,3])
   >>> [i for i in x]
   [2,3,4]
   ```

   Your task is to write a new version of the map function, `mymap(f,itr)`. It performs the same operations as the built-in version, but its result is explicitly visible. Specifically, it returns an object that matches the type of the parameter object `itr`. With this version, mapping `f` to a list returns a list, mapping `f` to a tuple returns a tuple, etc. Your function only need to cover four types of iterable objects, list, tuple, set, and string. You may use the built-in map function to implement this new version, but it is not necessary.

   Here are some sample runs of this function:

```
>>> mymap(lambda x:x+1, [1,2,3])
[2,3,4]
>>> mymap(lambda x:x+1, (1,2,3))
(2,3,4)
>>> mymap(lambda x:x+1, {1,2,3})
{2,3,4}
>>> mymap(str.upper, 'abc')
'ABC'
```

4. Advanced compilers can perform optimizations on tail-recursive functions. The two C program files `fib.c` and `fibT.c` contain two implementations of the Fibonacci function, one (regular) recursive, one tail-recursive.

   (a) Compile the two programs using `gcc` with `-O2` optimization setting, and save the assembly code:

   ```
   linux> gcc -O2 -S fib.c -o fib.s
   linux> gcc -O2 -S fibT.c -o fibT.s
   ```

   Inspect the two assembly programs. Are the function `fib/fibT` still being called recursively in them?

   (b) Now compile both programs to executable code, and do a simple timing comparison:

   ```
   linux> gcc -O2 -o fib fib.c
   linux> gcc -O2 -o fibT fibT.c
   linux> time ./fib 40
   linux> time ./fibT 40
   ```

   Do you see an execution-time difference between these two versions?

Submission

Assemble your answers in `ex6sol.py` and submit it through the "File Upload" tab in the "Exercise 6" folder (which is in the "Assignments" category). (You need to press the "Start Assignment" button to see the submission options.) The submission deadline is the end of Monday (11/18).