Quiz 4

Name: ID:

1) Write a function called `create_filter` in Python that will take a filter predicate and a sequence as parameters, and returns a function. The filter predicate parameter is a function that takes an item as a parameter and returns true if the item satisfies the filter predicate, and false otherwise. The sequence parameter is any iterable object, such as a list. Finally, the return value of `create_filter` is a **generator** that returns, at each iteration, the next item in the sequence that satisfies the filter predicate. Here is an example illustrating the use of `create_filter`:

```
a = [1, 2, 3, 4]
filter = create_filter(lambda x: (x * x) % 3 == 1, a)
for v in filter():
  print v
```

This would print: `1 ¶ 2 ¶ 4 ¶`

a) (2.5pts) Write down the body of the `create_filter` function.

```
def create_filter(lam, a):
    for i in a:
        if(lam(i)):
            yield i
```

Assume that you want to create multiple filters on the same sequence, yet avoid passing in the same sequence as a parameter to the `create_filter` function for each one. For this purpose, you write a function called `bind_sequence_to_filter_creator`, which takes a filter creator function (like the one from above) and a sequence as parameters, and creates a new filter creator that does not take a sequence parameter and instead uses the one you just bound to it. For example:

```
create_filter2 = bind_sequence_to_filter_creator(create_filter, a)
filterEven = create_filter2(lambda x: x % 2 == 0)
filterOdd = create_filter2(lambda x: x % 2 == 1)

for v in filterEven():
  print v
for v in filterOdd():
  print v
```

This will print: `2 ¶ 4 ¶ 1 ¶ 3 ¶`

b) (2.5pts) Write down the body of the `bind_sequence_to_filter_creator` function.

```
def bind_sequence_to_filter_creator(cf, a):
    def filterer(l):
        return cf(l, a)
    return filterer
```

2) (5pts) What are the values of *y* and *z* at the end of the following program considering static scoping and sub-expression evaluation order of left-to-right.

```
int y;
int z;

int foo(int x)
{
  x = x + 4;
  y = x;
  x = x + 4;
  return y;
}

int bar(int x)
{
  y = foo(x) * x;
  return x;
}

void main()
{
  y = 3;
  z = bar(y);
}
```

a) (1 pts.) With pass-by-value

y = 21

z = 3

b) (2 pts.) With pass-by-value-result

y = 11 ?

z = 11 ?

c) (2 pts.) With pass-by-reference

y = 121

z = 121