# Sample exam 1

The INFDEV team

## 1 Lists, functions, and iteration

### 1.1 Question 1

Complete the missing pieces of the `filterTooLarge` function to remove all elements greater than 100 from the input list (dened as usual with `Empty` and `Node`).

**Guide to answering**

- Carefully read the question
- The only input mentioned is the list, so the function cannot take any other parameters
- To remove elements we can either return a new list, or modify the given list in place; returning a new list is usually simpler
- The function is clearly very similar to `filter`: go, recursively, through all elements
- – If we reach the empty node, then we return `Empty()` as we have nothing to remove
  – Otherwise, we check the condition (in this case if the current element is greater than 100)
  – ∗ If it is so, then we simply return the removal of all further elements
  ∗ Otherwise, we return a new node that includes the current value (we do not wish to discard it) and recurse on the next elements

```
1 def filterTooLarge(l):
2   if l.IsEmpty():
3     return Empty()
4   else:
5     if (l.Value > 100):
6       return filterTooLarge(l.Next)
7     else:
8       return Node(l.Value,filterTooLarge(l.Next))
```

## 1.2 Question 2

Complete the missing pieces of the `filterTooSmall` function to remove all elements smaller than 5 from the input list (dened as usual with `Empty` and `Node`).

**Guide to answering**

- Carefully read the question
- The only input mentioned is the list, so the function cannot take any other parameters
- To remove elements we can either return a new list, or modify the given list in place; returning a new list is usually simpler
- The function is clearly very similar to `filter`: go, recursively, through all elements
- 
  - If we reach the empty node, then we return `Empty()` as we have nothing to remove
  - Otherwise, we check the condition (in this case if the current element is smaller than 5)
  - 
    - If it is so, then we simply return the removal of all further elements
    - Otherwise, we return a new node that includes the current value (we do not wish to discard it) and recurse on the next elements

```
1 def filterTooSmall(l):
2   if l.IsEmpty():
3     return Empty()
4   else:
5     if (5 > l.Value):
6       return filterTooSmall(l.Next)
7     else:
8       return Node(l.Value,filterTooSmall(l.Next))
```

## 1.3 Question 3

Complete the missing pieces of the `multiplyBy` function to multiply all elements of the input list (dened as usual with `Empty` and `Node`) by the input number.

**Guide to answering**

- Carefully read the question
- The two inputs mentioned are the list and a number, so the function cannot take any other parameters

- To transform the elements we can either return a new list, or modify the given list in place; we show how to return a new list
- The function is clearly very similar to `map`: go, recursively, through all elements
- 
  - If we reach the empty node, then we return `Empty()` as we have no element to transform
  - Otherwise, we return a new node that includes the current value multiplied and recurse on the next elements

```
1 def multiplyBy(l,k):
2   if l.IsEmpty():
3     return Empty()
4   else:
5     return Node((l.Value * k),multiplyBy(l.Next,k))
```

## 1.4 Question 4

Write a loop that multiplies all elements of a list `l` which are greater than zero.

**Guide to answering**

- Carefully read the question
- The only variable mentioned is a list `l`, so assume it is declared and initialized
- To multiply elements we must store the product so far; we need a variable for this, which is initialized to 1
- We loop through all nodes of the list:
- 
  - If we reach the empty node, then we are done and we stop the loop
  - Otherwise, we check the condition (in this case if the current element is greater than zero)
  - If it is so, then we simply multiply it by the `product` variable
  - After the check, we move to the next element

```
1 while (l.IsEmpty() == False):
2   if (l.Value > 0):
3     product = (product * l.Value)
4   l = l.Next
```

# 2 Stack and heap

## 2.1 Question 1

Show the stack and the heap at all steps of the execution of the following function:

**Guide to answering**

- Carefully read the code; take five minutes to get an idea of what the function does
- Begin following the code, changing the variables and the PC as needed
- Remember that whenever you encounter a function call (so also for recursion) you need to put:
- 
  - another PC
  - all parameters of the function
  - the place to put the return value of the fuction on the stack (in the following we call it `ret`, but use whatever name you wish: also the name of the called function might do it)
- Only the last (rightmost) locations in the stack change, so you might choose to not rewrite those that stay the same
- Show all PC's though, as those identify the path that code has taken
- Do not get stuck on notation; as long as you show all the relevant values, your answer will be accepted

```
1 def f(n):
2   if (n > 1):
3     return (1 +   f((n // 2)))
4   else:
5     return 0
6 print(f(3))
```

Stack:

| PC |
|---|
| 1 |

Stack:

| PC | ... | | PC | ret | n |
|---|---|---|---|---|---|
| 6 | ... | | 2 | None | 3 |

Stack:

| PC | ... | | PC | ret | n |
|---|---|---|---|---|---|
| 6 | ... | | 3 | None | 3 |

Stack:

| PC | ... | | PC | ... | | PC | ret | n |
|---|---|---|---|---|---|---|---|---|
| 6 | ... | | 3 | ... | | 2 | None | 1 |

Stack:

| PC | ... | | PC | ... | | PC | ret | n |
|---|---|---|---|---|---|---|---|---|
| 6 | ... | | 3 | ... | | 5 | None | 1 |

Stack:

| PC | ... | | PC | ... | | PC | ret |
|---|---|---|---|---|---|---|---|
| 6 | ... | | 3 | ... | | 5 | 0 |

Stack:

| PC | ... | | PC | ret |
|---|---|---|---|---|
| 6 | ... | | 4 | 1 |

Stack:

| PC |
|---|
| 10 |

Output: 1

## 2.2 Question 2

Show the stack and the heap at all steps of the execution of the following function:

**Guide to answering** See above

```
1 def f(n):
2   if (n > 1):
3     return (n *   f((n - 1)))
4   else:
5     return 1
6 print(f(3))
```

Stack:

| PC |
|---|
| 1 |

Stack:

| PC | ... | | PC | ret | n |
|---|---|---|---|---|---|
| 6 | ... | | 2 | None | 3 |

Stack:

| PC | ... | | PC | ret | n |
|---|---|---|---|---|---|
| 6 | ... | | 3 | None | 3 |

Stack:

| PC | ... | | PC | ... | | PC | ret | n |
|---|---|---|---|---|---|---|---|---|
| 6 | ... | | 3 | ... | | 2 | None | 2 |

Stack:

| PC | ... | | PC | ... | | PC | ret | n |
|---|---|---|---|---|---|---|---|---|
| 6 | ... | | 3 | ... | | 3 | None | 2 |

Stack:

| PC | ... | | PC | ... | | PC | ... | | PC | ret | n |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | ... | | 3 | ... | | 3 | ... | | 2 | None | 1 |

Stack:

| PC | ... | | PC | ... | | PC | ... | | PC | ret | n |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 6 | ... | | 3 | ... | | 3 | ... | | 5 | None | 1 |

Stack:

| PC | ... | | PC | ... | | PC | ... | | PC | ret |
|---|---|---|---|---|---|---|---|---|---|---|
| 6 | ... | | 3 | ... | | 3 | ... | | 5 | 1 |

Stack:

| PC | ... | | PC | ... | | PC | ret |
|---|---|---|---|---|---|---|---|
| 6 | ... | | 3 | ... | | 4 | 2 |

Stack:

| PC | ... | | PC | ret |
|---|---|---|---|---|
| 6 | ... | | 6 | 6 |

Stack:

| PC |
|---|
| 10 |

Output: 6