Part 2 - Q/A

1. What is the meaning of cardinality of a type?

The cardinality of a type is the number of possible legal values that can be of that type.

With function types, we usually want to consider two functions that return the same value for every input to be "the same function", for cardinality purposes at least (this is known as "extensional equality")

Expressing the cardinality of types which can have a finite number of possible values, is fairly easy in principle, because you can just give a number as the cardinality. However, with infinite cardinalities, technically there is a distinction between different kinds of infinities. For example, an uncountable infinity is "larger than" a countable infinity.

2. What is the difference in between a dangling pointer and null pointer?

Pointer terminology:

* Dangling (or wild) pointer: a pointer that points somewhere, but not to a valid object.

* Null pointer: a pointer that points to a specially designated out-of-bounds location that programs will never legally store data in. Special class of dangling pointer.

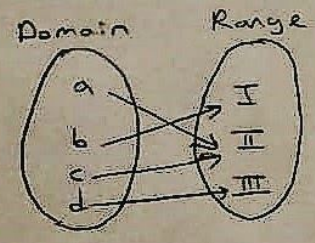
* A null pointer just means the pointer isn't pointing to anything, or in some languages means it is unknown what it is pointing at. But because it is a null pointer, you know this, the code knows this, so no problem. A dangling pointer is one that you think is pointing at something but in reality is no longer there, hence the pointer is actually inaccurate but doesn't know it.

3-) Explain the difference between C, C++ and Perl arrays.

Simply stated: there are static arrays, where the capacity is fixed and doesn't change; there are dynamic arrays that can change their capacity. In C++ and C, the arrays are static, fixed capacity. In Perl, the arrays are dynamic.

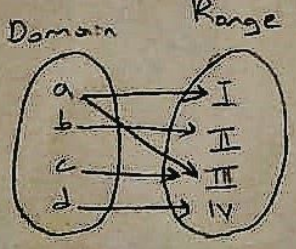
4-) Is function a mapping? If yes, how?

A function is a special type of relation in which each element of the domain is paired with exactly one element in the range. A mapping shows how the elements are paired. Its like a flow chart for a function, showing the input and output values.



✓

In this mapping, the second element of the range associates with more than one element in the domain. If the element (s) in range that have mapped more than one elment in the domain is called many-to-one mapping.



✗

In this mapping, the first element in the domain have mapped with more than one element in the range. If one element in the domain mapped with more than one element in the range, the mapping is called one-to-many relations are not functions

5.) What is deep copy? Explain it with an example

Deep Copy: it is a process of creating a new object and the copying the fields of the current object to the newly created object to make a complete copy of the internal reference types. If the specified field is a value type, then a bit-by-bit copy of the field will be performed.

```
// Copying a list using deepcopy()

import copy
old_list = [[1,1,1], [2,2,2], [3,3,3]]
new_list = copy.deepcopy(old_list)

print("Old list:", old_list)
print("New list:", new_list)
```

6. Reverse the first k elements of a queue.

```
// C++ program to reverse first k elements of a queue.
using namespace std;

void reverseQueueFirstKElements (int k, queue<int>& Queue) {
  if Queue.empty()== true || k> Queue.size())
    return;
  if (k<=0)
    return;
  stack<int> Stack;

  /*Push the first K elements into a Stack */
  for (int i=0; i<k; i++) {
      Stack.push (Queue.front());
      Queue.pop();
  }

  /*Enqueue the contents of stack at the back of the queue */
  while (!Stack.empty()) {
      Queue.push (Stack.top());
      Stack.pop();
  }

  /*Remove the remaining elements and enqueue them at the end */
  for (int i = 0; i< Queue.size() - k; i++) {
      Queue.push (Queue.front());
      Queue.pop();
  }
}

/* Utility function to print the Queue */
void Print(queue<int>& Queue)
{
  while (!Queue.empty()) {
      cout<< Queue.front() << " ";
      Queue.pop();
  }
}
```

```
//Driver code
int main()
{
queue<int> Queue;
Queue.push(10);
Queue.push(20);
Queue.push(30);
Queue.push(40);
Queue.push(50);
Queue.push(60);
Queue.push(70);
Queue.push(80);
Queue.push(90);
Queue.push(100);

int k = 5;
reverseQueueFirstKElements(k, Queue);
Print(Queue);
}
```