Heap-Allocated Memory and Box-and-Arrow Diagrams

Question 1

Consider the difference between statically and dynamically allocated memory.

- How can you tell in code that a value is being stored on the stack?
  Statically declared code is stored on the stack, this is code that is declared explicitly and has a declared size, and memory is reserved by the compiler.
- When does the memory for a variable on the stack get deleted?
  Automatically when the variable is popped from the stack after its function exits.
- Similarly, how can you tell in code that a value is being stored on the heap?

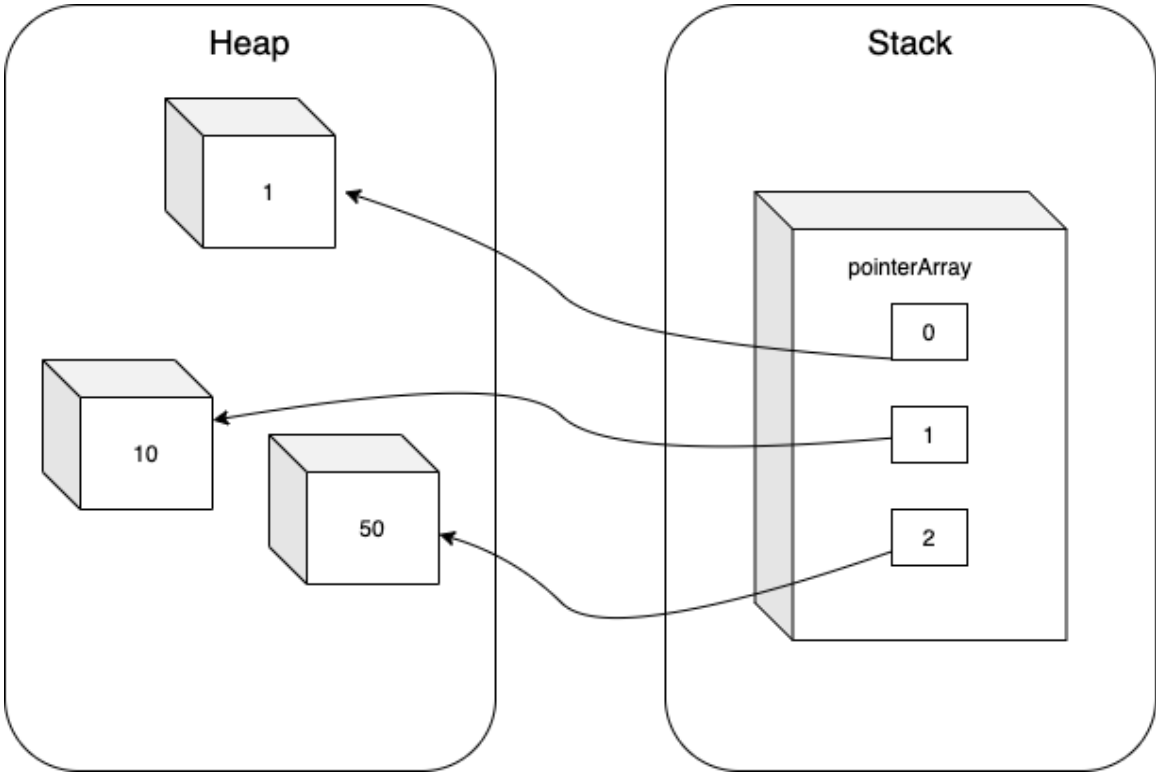
- When does memory for a value on the heap get deleted?
  Only when manually deleted by using the `delete` operator.

Question 2

In C++, you can create an array on the stack just like usual, but use it to store pointers to data created on the heap. For example, suppose we want to store integers. Instead of creating a contiguous row of boxes in memory that will each store an integer, the boxes will store a pointer to integers on the heap.

Draw a box-and-arrow diagram that illustrates how the array will look after running the following code. Keep in mind that while the array itself will be stored contiguously, boxes allocated for the integers on the heap will be placed wherever there is room (so don't draw them contiguously).

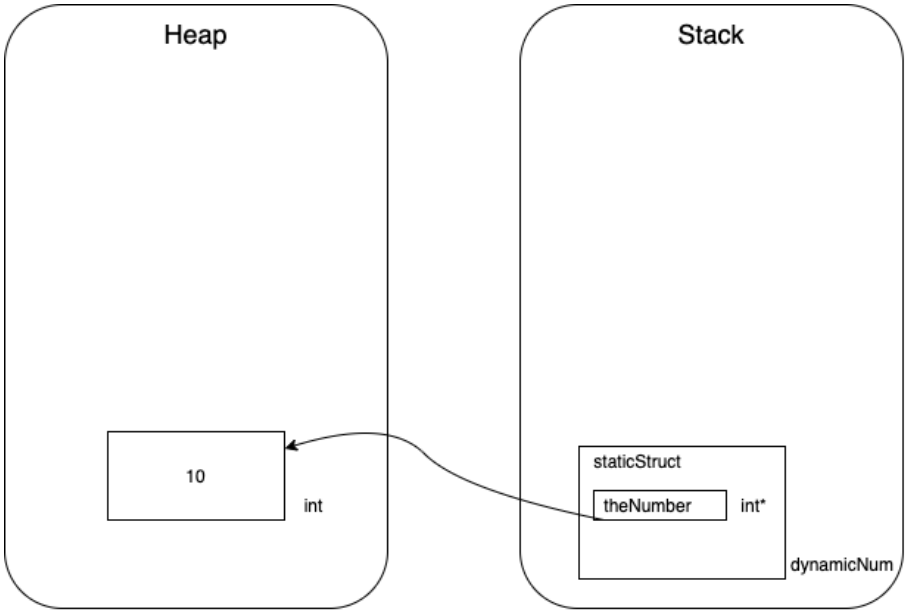
```
int * pointerArray[3];

pointerArray[0] = new int;
pointerArray[1] = new int;
pointerArray[2] = new int;

*(pointerArray[0]) = 1;
*(pointerArray[1]) = 10;
*(pointerArray[2]) = 50;
```

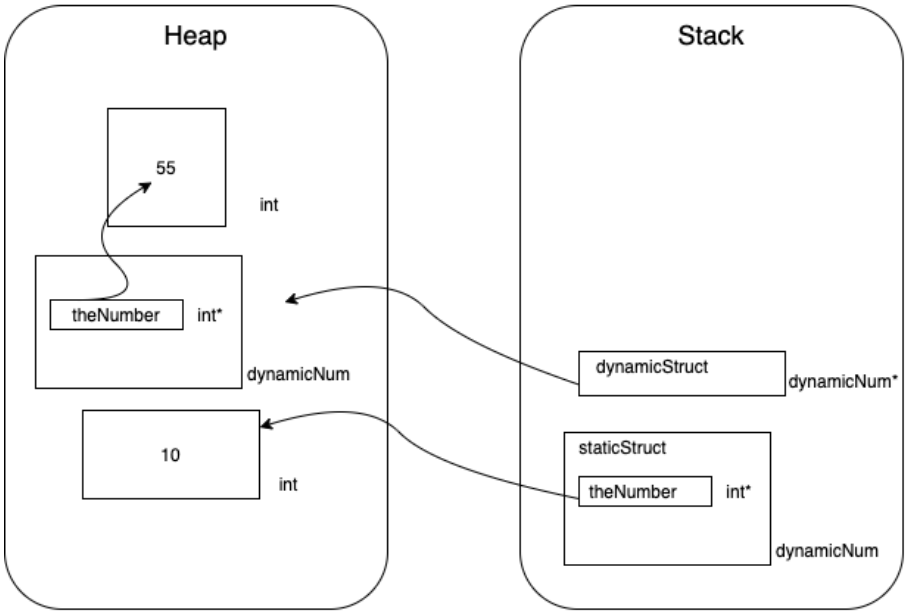


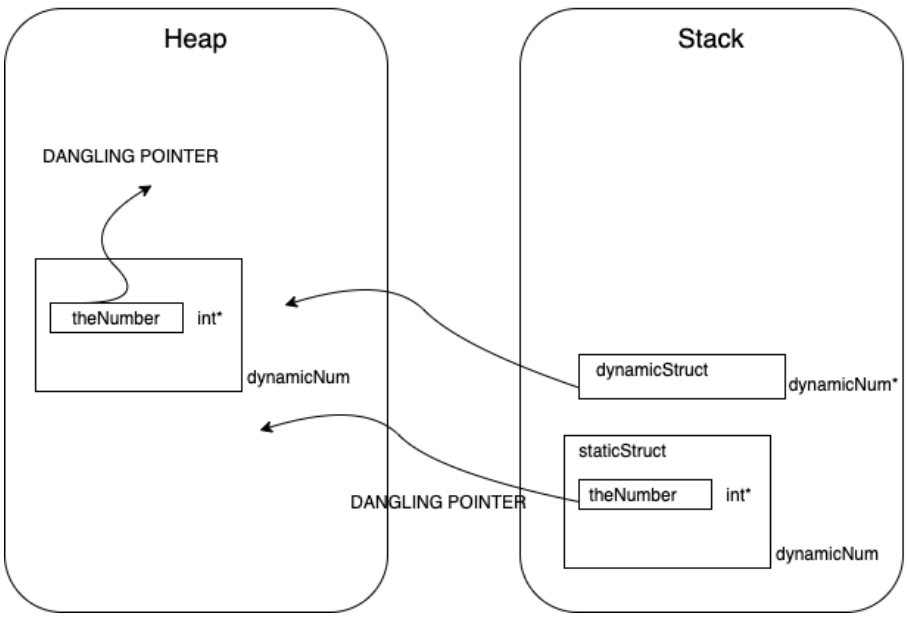
Question 3

Draw a box-and-arrow diagram that illustrates the state of the program's memory after lines 1, 2, and 3.

```
struct dynamicNum {
   int *theNumber;
};

int main() {
   dynamicNum staticStruct;
   staticStruct.theNumber = new int;
   *(staticStruct.theNumber) = 10; /* 1 */

   dynamicNum *dynamicStruct = new dynamicNum;
   dynamicStruct->theNumber = new int;
   *(dynamicStruct->theNumber) = 55; /* 2 */

   cout << "Static struct pointer: "
        << staticStruct.theNumber << endl;
   cout << "Static struct number: "
        << *(staticStruct.theNumber) << endl;

   cout << "Dynamic struct pointer: "
        << dynamicStruct->theNumber << endl;
   cout << "Dynamic struct number: "
        << *(dynamicStruct->theNumber) << endl;

   delete staticStruct.theNumber;
   delete dynamicStruct->theNumber; /* 3 */
   delete dynamicStruct;
}
```


Heap
Stack
10
int
staticStruct
theNumber
int*
dynamicNum


Heap
Stack
55
int
theNumber
int*
dynamicNum
10
int
dynamicStruct
dynamicNum*
staticStruct
theNumber
int*
dynamicNum


Heap
Stack
DANGLING POINTER
theNumber
int*
dynamicNum
dynamicStruct
dynamicNum*
staticStruct
theNumber
int*
DANGLING POINTER
dynamicNum