1. (50 pts) Concolic testing the circular queue of positive integers

- Given C program of the circular queue, convert asserts and environment model for concolic testing
  - You have to use **depth-first search (DFS)** to generate test inputs to cover all possible execution paths
- To do list:
  - Describe your assertion check routine in detail
  - Describe your environment model in detail
  - Describe run-time parameters of CROWN
  - Report concolic testing results
    - Assert violation if any
    - # of test inputs generated
    - Time spent
    - # of branches and branch coverage measured by gcov

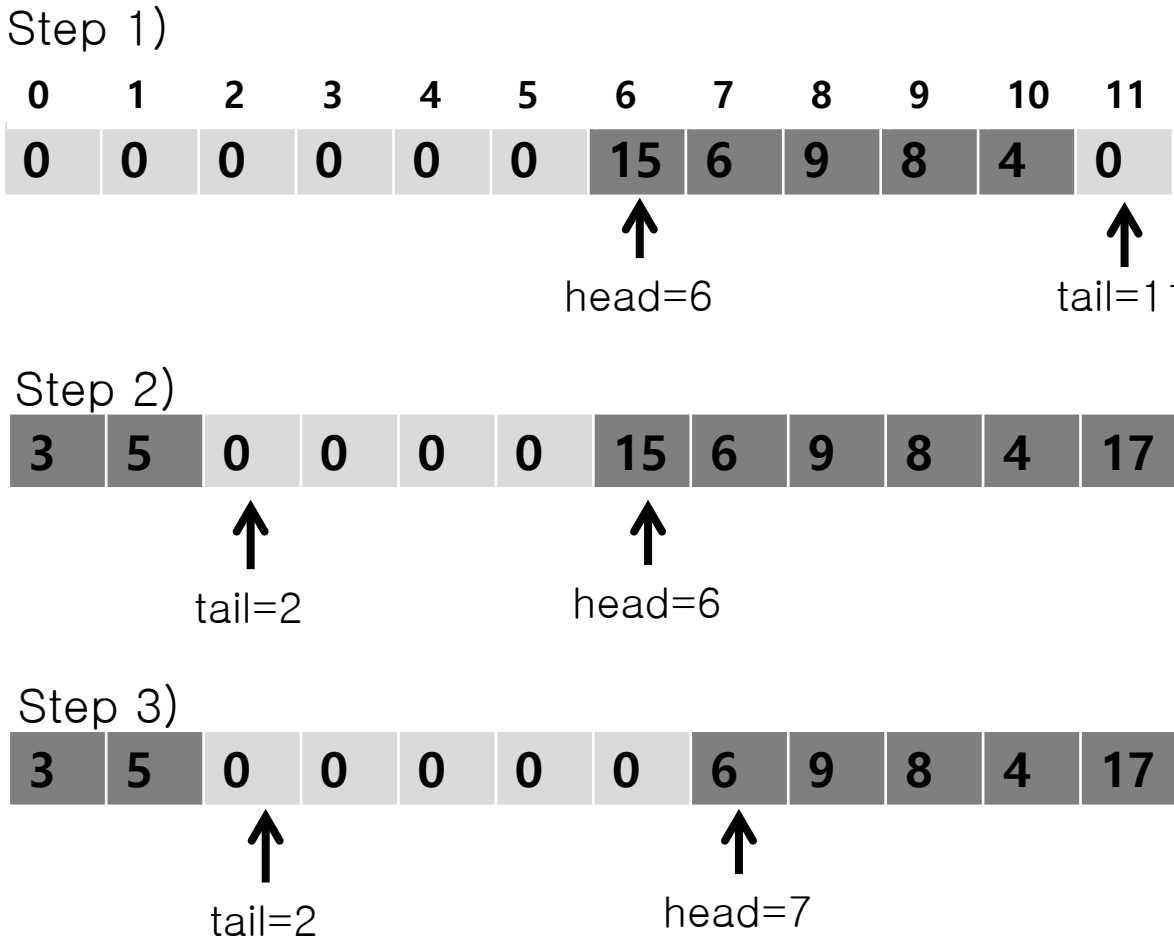
Circular Queue of Positive Integers

```
#include<stdio.h>
#define SIZE 12
#define EMPTY 0

// We assume that q[] is
// empty if head==tail
unsigned int q[SIZE],head,tail;

void enqueue(unsigned int x)
{
    q[tail]=x;
    tail=(++tail)%SIZE;
}

unsigned int dequeue() {
    unsigned int ret;
    ret = q[head];
    q[head]=0;
    head= (++head)%SIZE;
    return ret;}
```



```
void enqueue_verify() {
    unsigned int x, old_head, old_tail;
    unsigned int old_q[SIZE], i;
    __CPROVER_assume(x>0);

    for(i=0; i < SIZE; i++) old_q[i]=q[i];
    old_head=head;
    old_tail=tail;

    enqueue(x);

    assert(q[old_tail]==x);
    assert(tail== ((old_tail +1) % SIZE));
    assert(head==old_head);
    for(i=0; i < old_tail; i++)
            assert(old_q[i]==q[i]);
    for(i=old_tail+1; i < SIZE; i++)
            assert(old_q[i]==q[i]);
}
```

```
int main() {// cbmc q.c –unwind
SIZE+2
    environment_setup();
    enqueue_verify();}
```

```
void dequeue_verify() {
    unsigned int ret, old_head, old_tail;
    unsigned int old_q[SIZE], i;

    for(i=0; i < SIZE; i++) old_q[i]=q[i];
    old_head=head;
    old_tail=tail;
    __CPROVER_assume(head!=tail);

    ret=dequeue();

    assert(ret==old_q[old_head]);
    assert(q[old_head]== EMPTY);
    assert(head==(old_head+1)%SIZE);
    assert(tail==old_tail);
    for(i=0; i < old_head; i++)
            assert(old_q[i]==q[i]);
    for(i=old_head+1; i < SIZE; i++)
            assert(old_q[i]==q[i]);}
```

```
int main() {// cbmc q.c –unwind
SIZE+2
    environment_setup();
    dequeue_verify();}
```

```
#include<stdio.h>
#define SIZE 12
#define EMPTY 0

unsigned int q[SIZE],head,tail;

void enqueue(unsigned int x)
{
    q[tail]=x;
    tail=(++tail)%SIZE;
}

unsigned int dequeue() {
    unsigned int ret;
    ret = q[head];
    q[head]=0;
    head= (++head)%SIZE;
    return ret;
}
```

```
// Initial random queue setting following the script
void environment_setup() {
    int i;
    for(i=0;i<SIZE;i++) { q[i]=EMPTY;}

    head=non_det();
    __CPROVER_assume(0<= head && head < SIZE);

    tail=non_det();
    __CPROVER_assume(0<= tail && tail < SIZE);

    if( head < tail)
        for(i=head; i < tail; i++) {
            q[i]=non_det();
            __CPROVER_assume(0< q[i]);
        }
    else if(head > tail) {
        for(i=0; i < tail; i++) {
            q[i]=non_det();
            __CPROVER_assume(0< q[i]);
        }
        for(i=head; i < SIZE; i++) {
            q[i]=non_det();
            __CPROVER_assume(0< q[i]);
        }
    } // We assume that q[] is empty if head==tail
}
```

2. (50 pts) Concolic testing max_heapify()

- Given C program of max_heapify(), convert asserts and environment model for concolic testing
  - You have to use **depth-first search (DFS)** to generate test inputs to cover all possible execution paths
- To do list:
  - Describe your assertion check routine in detail
  - Describe your environment model in detail
  - Describe run-time parameters of CROWN
  - Report concolic testing results
    - Assert violation if any
    - # of test inputs generated
    - Time spent
    - # of branches and branch coverage measured by gcov

`max_heapify(int x[],int i,int h_size)`

- `x[]` is an array to contain a max-heap
- `i` is the index to the node that may violate the max-heap property
- `h_size` is a total number of nodes in the max-heap:
- `max_heapify` makes a subtree whose root is `x[i]` a max heap with the following assumptions.

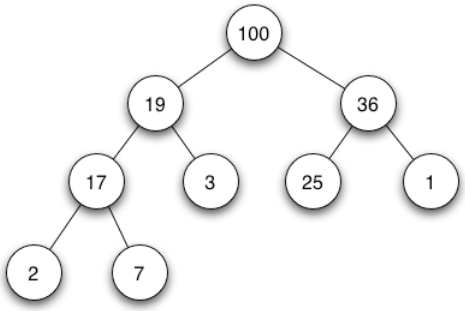
Assumptions

1. A given tree satisfies the shape property

2. The right and left sub-trees of node i are max heaps, but that `x[i]` may be smaller than its children

3. The size of `x[]` is **16**.

- A *max heap* is a heap data structure created using a binary tree with two constraints:
  - *The shape property*: the tree is a complete binary tree; that is, all levels of the tree, except possibly the last one (deepest) are fully filled, and, if the last level of the tree is not complete, the nodes of that level are filled from left to right.
  - *The max-heap property:* each node is greater than or equal to each of its children according to a comparison predicate defined for the data structure.



Max heap can be implemented using an array as follows (note that array index starts from 1):

Index	1	2	3	4	5	6	7	8	9
value	100	19	36	17	3	25	1	2	7

```c
/* Example code */
#include<stdio.h>
#define MAX 16
#define H_SIZE 10
#define parent(i)(i/2)
#define left(i) (2*i)
#define right(i)(2*i+1)

/* Ignore the first 0, since max heap
 contents start at index 1 */
int a[MAX] = {0,16,4,10,14,7,9,3,2,8,1,};

void  max_heapify(int x[],int i,int h_size){
  int largest, tmp;
  int l=left(i);
  int r=right(i);

  if (l<=h_size &&  x[l]>x[i]) largest=l;
  else  largest=i;
  if(r<=h_size && x[r]>x[largest]) largest=r;
  if (largest!=i)  {
    tmp=x[i];
    x[i]=x[largest];
    x[largest]=tmp;
    max_heapify(x,largest,h_size);
  }
}
```

```c
int main(){
  int i;
  max_heapify(a,2,H_SIZE);
  for (i=1;i<=H_SIZE;i++) printf("%d ",a[i]);
  return 0;
} /* Output: 16 14 10 8 7 9 3 2 4 1 */
```