Key Difference between Manual Testing and Concolic/Symbolic Testing

- Manual testing
  - A user should test one concrete execution scenario by checking a pair of concrete input values and the expected concrete output values
- Concolic/symbolic testing
  - A user should imagine all possible execution scenarios and model a general environment that can enable all possible executions
  - A user should describe general invariants on input values and output values

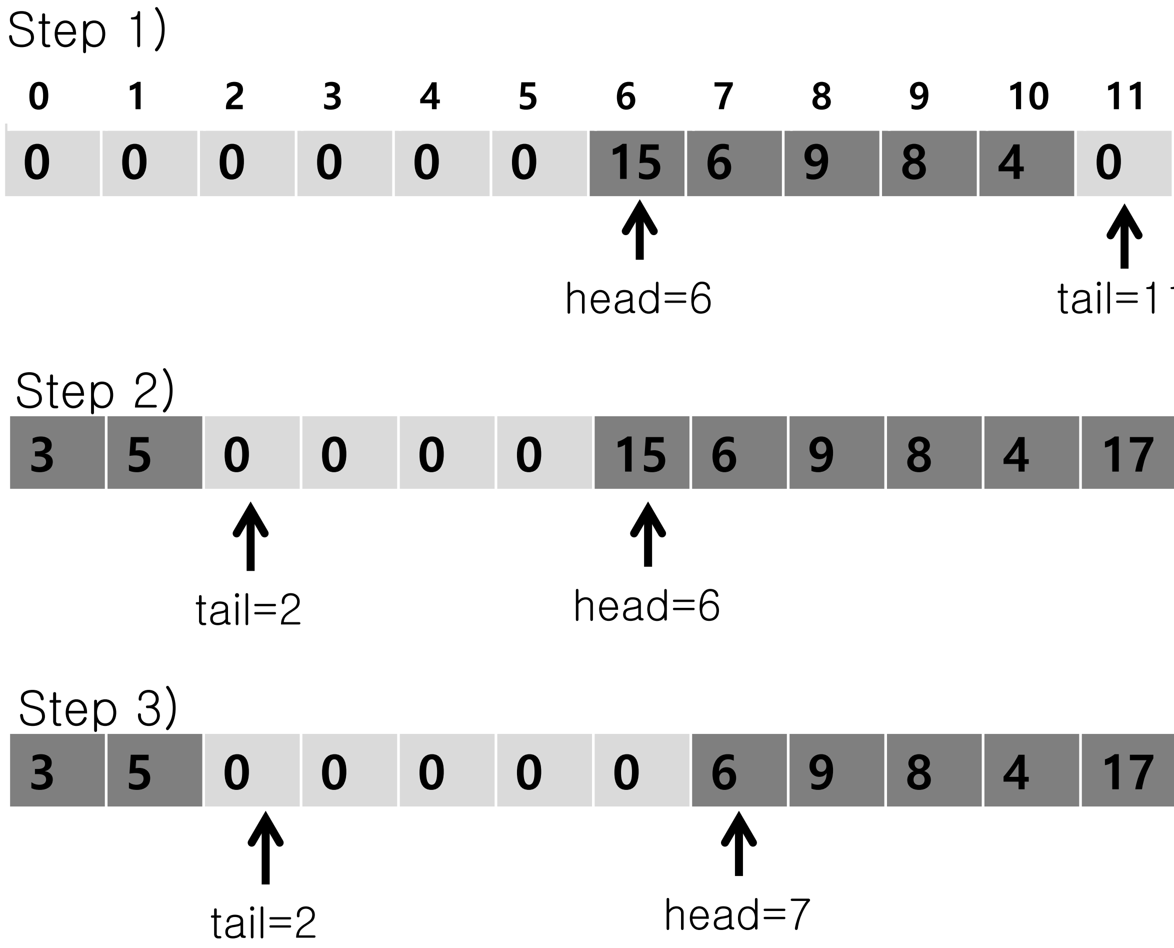
Ex1. Circular Queue of Positive Integers

```
#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;
    __sym_assume(x>0);//if(!(x>0)) exit();

    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]);
}
```

```
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;
    __sym_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() {
    environment_setup();
    enqueue_test();}
```

```
int main() {
    environment_setup();
    dequeue_test();}
```

```c
#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;
}
```

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

    CREST_unsigned_int(head);
    __sym_assume(0<= head && head < SIZE);

    CREST_unsigned_int(tail);
    __sym_assume(0<= tail && tail < SIZE);

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

 printf("head:%u, tail:%u₩n",head, tail);
    if( head < tail)
        for(i=head; i < tail; i++) printf("q[%u]:%u₩n",i,q[i]);
    else if(head > tail) {
        for(i=0; i < tail; i++) printf("q[%u]:%u₩n",i,q[i]);
        for(i=head; i < SIZE; i++) printf("q[%u]:%u₩n",i,q[i]);
    }
}
```