Find K-largest element in a stream of elements

```
#include <stdio.h>
#include <stdlib.h>

void swap(int *a, int *b)
{
	int temp = *a;
	*a = *b;
	*b = temp;
}

void MinHeapify(int *arr, int index, int size)
{
	printf("%d", index);
	int left = 2*index + 1;
	int right = 2*index + 2;
	int smallest = index;
	if (left < size && arr[left] < arr[index])
        smallest = left;
    if (right < size && arr[right] < arr[smallest])
        smallest = right;
    if (smallest != index)
    {
        swap(&arr[index], &arr[smallest]);
        MinHeapify(arr, index, smallest);
    }
}

void buildMinHeap(int *arr, int size)
{
	for(int index = (size/2) - 1; index >= 0; index--)
		MinHeapify(arr, index, size);
}

int getMinimum(int *arr)
{
	return arr[0];
}

void replaceMinimum(int *arr, int newEle, int size)
{
	arr[0] = newEle;
	MinHeapify(arr, 0, size);
}

void KthLargestInStream(int k)
{
	int count = 0, newEle, flag = 1; //count is for total number of elements in stream seen so far

	//allocate memory
	int *arr = (int *)malloc(sizeof(int) * k);
```

```
        while(flag)
        {
                printf("Enter next element in stream\n");
                scanf("%d", &newEle);

                if(count < k-1)
                        arr[count] = newEle;
                else
                {
                        if(count == k-1)
                        {
                                arr[count] = newEle;
                                buildMinHeap(arr, k);
                        }
                        if(newEle > getMinimum(arr))
                                replaceMinimum(arr, newEle, k);

                        printf("Kth largest element is = %d\n", getMinimum(arr));

                }
                count++;
                // This is used to quit the program not required
                if(count == 10)
                        flag = 0;
        }
}

int main()
{
        int k;
        printf("Enter the value of k\n");
        scanf("%d", &k);
        KthLargestInStream(k);

        return 0;
}
```

Time complexity: O(logk)
Space complexity: O(1)