```
#include "HuffmanHeap.h"
#include <iostream>
using namespace std;


HuffmanHeap::HuffmanHeap(int size){
    capacity = size;
    heapSize = 0;
    storage = new HuffmanNode*[capacity];
    for(int i = 0; i < capacity; i++){
        storage[i] = NULL;
    }
}

HuffmanHeap::~HuffmanHeap(){
    if(storage != NULL){
        for(int i = 0; i < capacity; i++){
            if(storage[i] != NULL){
                delete storage[i];
                heapSize--;
            }
        }
        delete[] storage;
    }
}

void HuffmanHeap::insert(HuffmanNode* rhs){
    if(heapSize == capacity){
        return;
    }
    heapSize++;
    storage[heapSize-1] = rhs;
    percUp(heapSize-1);
}

void HuffmanHeap::percUp(int curIndex){
    int parentIndex;
    HuffmanNode* temp;
    if(curIndex != 0) {
        parentIndex = (curIndex-1)/2;

        if(storage[parentIndex]->getFrequency() > storage[curIndex]->getFrequency()){
            temp = storage[parentIndex];
            storage[parentIndex] = storage[curIndex];
            storage[curIndex] = temp;
            percUp(parentIndex);
        }
    }
}

int HuffmanHeap::numChildren(int index){
    int children = 0, left = 2*index+1, right = 2*index+2;
    if(left >= heapSize) return 0;
    if(storage[left] != NULL) children++;
    if (storage[right] != NULL) children++;
    return children;
}

void HuffmanHeap::percDown(int curIndex){
    int children = 0;
    if(children = numChildren(curIndex)){ //are we at the "bottom"?
        HuffmanNode* temp;
        int left = 2*curIndex+1, right = 2*curIndex+2;
        switch(children){ //1 child or 2 children
            case 1:
                //if the child is smaller, perk the current node down
```

```
                if(storage[curIndex]->getFrequency() > storage[left]->getFrequency()){
                    temp = storage[curIndex];
                    storage[curIndex] = storage[left];
                    storage[left] = temp;
                    percDown(left);
                }
                break;
            case 2:
                //2 children, determine which is smaller, perk down that direction
                if((storage[curIndex]->getFrequency() > storage[left]->getFrequency()) ||
                            (storage[curIndex]->getFrequency() > storage[right]->getFrequency())){
                    //left child is smaller
                    if((storage[left]->getFrequency() < storage[right]->getFrequency())){
                        temp = storage[curIndex];
                        storage[curIndex] = storage[left];
                        storage[left] = temp;
                        percDown(left);
                    }
                    else{ //right child is smaller
                        temp = storage[curIndex];
                        storage[curIndex] = storage[right];
                        storage[right] = temp;
                        percDown(right);
                    }
                }
                break;
        }
    }
    return;
}


HuffmanNode* HuffmanHeap::removeMin(){
    HuffmanNode* value = new HuffmanNode();
    *value = *storage[0];

    HuffmanNode* temp = storage[0];
    storage[0] = storage[heapSize-1];
    storage[heapSize-1] = temp;
    delete storage[heapSize-1];
    storage[heapSize-1] = NULL;
    heapSize--;
    percDown(0);
    return value;
}
```