```cpp
/*
Filename: p8.cpp
Author(s): Zachary Rea and Parker Ross
Date: March 8 2023
Description: The cpp for p8
*/
#include "P8.h"
#include <iostream>

using namespace std;

//*************************************************************************
//Constructors and Destructors
//*************************************************************************
//Constructor
//Written by Zach modified by Parker
intList::intList (int listCapacity) {
    this->listCapacity = listCapacity;
    a = new int[listCapacity];
    listSize = 0;
}

//*************************************************************************
//Destructor
//Written by Zach
intList::~intList() {
    delete[] a;
}

//*************************************************************************
//Public Functions
//*************************************************************************
//Function to insert a value at the start of the list
//Written by Parker modified by Zach
bool intList::insert(int key) {
    bool rc = listCapacity > listSize;
    if (rc) {
        for (int i = listSize; i > 0; i--) {
            a[i] = a[i-1];
        }
        a[0] = key;
        listSize++;
    }
    return rc;
}

//*************************************************************************
//Fucntion to add an int to the end of the list
//Written by Parker modified by Zach
bool intList::add(int key) {
    bool rc;
    if (listSize < listCapacity) {
        a[listSize] = key;
        listSize++;
        rc = true;
    }
    return rc;
}

//*************************************************************************
//Function to insert a value at a given index of the list
//Written by Parker modified by Zach
bool intList::insertAt(int index, int key) {
    bool rc = listCapacity > listSize && index < listSize && index >= 0;
    if (rc) {
        for (int i = listSize; i > index; i--) {
            a [i] = a[i-1];
        }
        a[index] = key;
```

```
70          listSize++;
71      }
72      return rc;
73  }
74
75  //*************************************************************************
76  //Function to delete a value at a given index of the list
77  //Written by Parker modified by Zach
78  bool intList::deleteAt(int index, int &key) {
79      bool rc = listSize > index && index >= 0;
80          if (rc) {
81              key = a[index];
82              for (int i = index; i < listSize; i++) {
83                  a[i] = a[i+1];
84              }
85          listSize--;
86      }
87      return rc;
88  }
89
90  //*************************************************************************
91  //Function to set the size to 0
92  //Written and modified by Zach
93  void intList::clear() {
94      listSize = 0;
95  }
96
97  //*************************************************************************
98  //Function to print n of the list
99  //Written and modified by Zach
100 void intList::printIt(int n) const{
101     cout << "printIt with list size: " << listSize << " capacity = ";
102     cout << listCapacity << "\n";
103     if (n < listSize) {
104         for (int i = 0; i < n; i++) {
105             cout << "At pos " << i << " there is " << a[i] << "\n";
106         }
107         cout << "At pos " << listSize - 1 << " there is " << a[listSize - 1];
108         cout << "\n";
109     } else {
110         for (int i = 0; i < listSize; i++) {
111             cout << "At pos " << i << " there is " << a[i] << "\n";
112         }
113     }
114 }
115
116 //*************************************************************************
117 //Function to return the index of a given value
118 //Written and modified by Zach
119 int intList::getIndex(int key) const{
120     int rc = -1;
121     for (int i = 0;i < listSize - 1;i++) {
122         if (a[i] == key) {
123             rc = i;
124             break;
125         }
126     }
127     return rc;
128 }
129
130 //*************************************************************************
131 //Function to read the value at a given index
132 //Written by Parker modified by Zach
133 int intList::readAt(int index, int &key) const{
134     int rc = -1;
135     if (index < listSize && index >= 0) {
136         rc = index;
137         key = a[index];
138     }
```

```
    return rc;
}

//*****************************************************************************
//Function to return the current size of the list
//Written and modiefied by Zach
int intList::capacity() const{
    return listCapacity;
}

//*****************************************************************************
//Fucntion to sort the array via bubble method
//Written by Zach
void intList::bubbleSort() {
    if (listSize > 0) {
        for (int i = listSize; i > 1; i--) {
            for (int j = 0; j < i-1; j++) {
                if (a[j] > a[j+1]) {
                    swap(a[j],a[j+1]);
                }
            }
        }
    }
}

//*****************************************************************************
//Function to sort the array via selection method
//Written by Parker
void intList::selectionSort() {
    for (int i = 0, j, least; i < listSize-1; i++){
        for (j = i+1, least = i; j < listSize; j++)
            if (a[j] < a[least])
                least = j;
                swap(a[least],a[i]);
    }
}
//*****************************************************************************
//Function to sort the array via the insertion method
//Written by Parker
void intList::insertionSort() {
    for (int i = 1, j; i < listSize; i++){
        int tmp = a[i];
        for (j = i; j > 0 && tmp < a[j-1]; j--)
            a[j] = a[j-1];
            a[j] = tmp;
    }
}

//*****************************************************************************
//Function to tell if the array is currently sorted
//Written by Zach
bool intList::isSorted() const{
    bool rc;
    for (int i = 0;i < listSize - 1; i++) {
        if (a[i] <= a[i + 1]) {
            rc = true;
        } else {
            rc = false;
            cout << "Index " << i << " is greater than index " << i + 1 << " ";
            break;
        }
    }
    return rc;
}
//*****************************************************************************
//Functions from program 6
//*****************************************************************************
//Function for finding left child
//Written by Zach modified by Parker
```

```
208 int intList::left(int index) const{
209     return (2*index)+1;
210 }
211
212 //*************************************************************************
213 //Function for finding right child
214 //Written by Zach modified by Parker
215 int intList::right(int index) const{
216     return (2*index)+2;
217 }
218
219 //*************************************************************************
220 //Function to swap two integers with each other
221 //Written by Zach modified by Parker
222 void intList::swap(int &x, int &y){
223     int s = x;
224     x = y;
225     y = s;
226 }
227
228 //*************************************************************************
229 //Function to perform heapify operation
230 //Written by Zach modified by Parker
231 void intList::heapify(int index) {
232     if (index >= 0) {
233         int larger = index;
234         int l = left(index);
235         if (l < heapSize) {
236             //check if left is larger
237             if (a[l] > a[larger]) {
238                 larger = l;
239             }
240             int r = right(index);
241             if (r < heapSize) {
242                 //check if right is larger
243                 if (a[r] > a[larger]) {
244                     larger = r;
245                 }
246             }
247         }
248         //swap if needed
249         if (index != larger) {
250             swap(a[index],a[larger]);
251             heapify(larger);
252         }
253     }
254 }
255
256 //*************************************************************************
257 //New Functions
258 //*************************************************************************
259 //Function to sort the heap
260 //Written by Zach
261 void intList::heapSort(){
262     heapSize = listSize;
263     buildHeap();
264     while (heapSize) {
265         swap(a[0],a[heapSize - 1]);
266         heapSize--;
267         heapify(0);
268     }
269 }
270
271 //*************************************************************************
272 //Function to build the superimposed heap
273 //Written by Zach
274 void intList::buildHeap(){
275     if (listSize > 0) {
276         int parent = (heapSize/2) - 1;
```

```
277             for (int i = parent;i >= 0; i--) {
278                 heapify(i);
279             }
280         }
281     }
```