This is a complex lab. Be sure to read the chapter notes about Heaps before starting.

1. Download the following files and create a workspace using them:

<https://markbowman.org/231/Lab23.zip>

These files contain partial code for implementing a heap sort, using an array of strings.

2. Compile and run the program. You should see this displayed:

Sample Run

Enter file name: ***Lab23.txt***

Array

----------------

Mark Frank Alisa Susan Steve Kathy Michael Richard Lisa David Rochelle John Lailee Carlos Ann

Text

Description automatically generated

3. Start by implementing the heap\_left(), heap\_right(), and heap\_parent() functions. Each should return the appropriate index for an array implementation of a binary tree.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Left()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//function to return the index of left child of node at index pos

//for inserting or removing nodes accordingly

inline int heap\_left(int pos)

{

return 2 \* pos + 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Right()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// function to return the index of right child of node at index pos

//for inserting or removing nodes accordingly

inline int heap\_right(int pos)

{

return 2 \* pos + 2;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Parent()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// function to return the index of parent child (or parent at root node) of node at index pos

inline int heap\_parent(int pos)

{

return (pos - 1) / 2;

}

4. Implement the heap\_insert() function. It will use the heap\_parent() in-line function you just created. Follow the algorithm explained in class:

* + Set map[n] = value
  + Set pos = n
  + Set parent = heap\_parent(pos)
  + Loop while pos>0 and map[pos]>map[parent]
* Swap map[pos] and map[parent]
* Set pos = parent
* Set parent = heap\_parent(pos)
  + Increment n

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Insert()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// function to insert val into array map to maintain the max heap

void heap\_insert(string map[], int& n, string value)

{

map[n] = value; // insert value at the end of map

int pos = n; // set pos to n

// get the index of parent of pos through call to heap parent

int parent = heap\_parent(pos);

// loop that continues until we reach the root of heap i.e pos is 0 and element at pos > element at parent

while ((pos > 0) && (map[pos] > map[parent]))

{

// swap the elements at pos and parent

string temp = map[pos];

map[pos] = map[parent];

map[parent] = temp;

pos = parent; // set pos to parent

parent = heap\_parent(pos); // get the index of parent of pos

}

n++; // increment n

}

5. Update the input loop in main(). Switch the commented lines.

while(!in.eof())

{ in >> name;

Comment out this line

if(in.good()) map[n++] = name;

// if(in.good()) heap\_insert(map,n,name);

Uncomment this line

};

6. Compile and run the program. If your heap\_insert() function works correctly, you should see this displayed:

Sample Run

Enter file name: ***Lab23.txt***

Array

----------------

Susan Steve Michael Richard Rochelle Lailee Kathy Frank Lisa David Mark Alisa John Carlos Ann

Text

Description automatically generated

7. Implement the heapify() function. It will take an unordered array and turn it into a heap. Follow the algorithm explained in class:

* Set i = 0
* Loop while i<n
  + Call heap\_insert(map,i,map[i]);

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heapify()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//function that takes unorded array and turns it into a heap

//based on process of heapification - requires heap\_insert

void heapify(string map[], int n)

{

//counter

int i=0;

//while less than size of array, insert elements into heap

while (i < n)

{

heap\_insert(map, i, map[i]);

}

}

8. Update the main() function so that the input loop reverts to storing the data in the order read from the file. After closing the file, call the heapify() function. Display the array before, and after the call to heapify(), using appropriate labels.

9. Compile and run the program. If your heapify() function works correctly, you should see this displayed:

Sample Run

Enter file name: ***Lab23.txt***

Array

----------------

Mark Frank Alisa Susan Steve Kathy Michael Richard Lisa David Rochelle John Lailee Carlos Ann

Heapified Array

----------------

Susan Steve Michael Richard Rochelle Lailee Kathy Frank Lisa David Mark Alisa John Carlos Ann

Text

Description automatically generated

**Main.cpp (so far):**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Lab23.cpp

\* Twymun K. Safford

\* Last Updated:11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

#define ARRAY\_MAX 1000

#include "Heapsort.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* main()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{ int i,n;

string fname,name,map[ARRAY\_MAX];

fstream in;

// Initialize

n = 0;

// Get file name

cout << "Enter file name: ";

cin >> fname;

// Read from file

in.open(fname.data(),ios::in);

if(!in.is\_open()) return;

// Loop through file

while (!in.eof())

{

in >> name;

if (in.good()) map[n++] = name;

//if (in.good()) heap\_insert(map, n, name);

};

// Close file

in.close();

// Display array

cout << endl;

cout << "Array" << endl << "----------------" << endl;

for (i = 0; i < n; i++)

{

cout << map[i] << " ";

}

cout << endl;

cout << endl;

cout << "Heapified Array" << endl << "----------------" << endl;

// call heapify

heapify(map, n);

for (i = 0; i < n; i++)

{

cout << map[i] << " ";

}

cout << endl << endl;

return;

}

10. Draw a picture of the heap tree on a separate sheet of paper.

**Binary Heap Tree – Lab23.txt**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Susan | | | | | | | |
| Michael | | | | Steve | | | |
| Lisa | | Mark | | Rochelle | | Kathy | |
| Lailee | David | Alisa | Frank | Richard | John | Carlos | Ann |

11. Implement the heap\_remove() function. It will use the heap\_left() and heap\_right()in-line functions. Follow the algorithm explained in class:

* Set pos = 0
* Decrement n
* Swap map[0] and map[n]
* Loop while pos<n
  + Set index = pos
  + Set left = heap\_left(pos)
  + Set right = heap\_right(pos)
  + If left<n and map[left]>map[index]
* Set index = left
  + If right<n and map[right]>map[index]
* Set index = right
  + If index ≠ pos
* Swap map[pos] and map[index]
* Set pos = index
  + Otherwise
* Set pos = n
* Return map[n]

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Remove()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//function to remove nodes from a heap

//starts at root node, moves node at bottom of tree to root,

// and then another heapification algorithm is used:

// continues until the node to be removed is found

string heap\_remove(string map[], int& n)

{

int pos = 0;

//decrement n by one

n--;

//compare ccurrent node to two children and if it isn't

//bigger than them, swap with larger of the two

swap(map[pos], map[n]);

while (pos < n)

{

int index = pos;

int left = heap\_left(pos);

int right = heap\_right(pos);

//if the left child is larger than the root

if ((left < n) && (map[left] > map[index]))

{

//use the left child as the new index

index = left;

}

//if right child greater than the largest so far

if ((right < n) && (map[right] > map[index]))

{

//use the right child as the index

index = right;

}

//if the largest current value is not at the root

if (index != pos)

{

//then swap the two values

swap(map[pos], map[index]);

pos = index;

}

else

{

//sets position to n

pos = n;

}

}

return map[n];

}

12. Update the main() function so that it calls the heap\_remove() function. Display the value returned from the function, and the resulting array, using appropriate labels.

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Lab23.cpp

\* Twymun K. Safford

\* Last Updated:11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

#define ARRAY\_MAX 1000

#include "Heapsort.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* main()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{ int i,n;

string fname,name,map[ARRAY\_MAX];

fstream in;

// Initialize

n = 0;

// Get file name

cout << "Enter file name: ";

cin >> fname;

// Read from file

in.open(fname.data(),ios::in);

if(!in.is\_open()) return;

// Loop through file

while (!in.eof())

{

in >> name;

if (in.good()) map[n++] = name;

//if (in.good()) heap\_insert(map, n, name);

};

// Close file

in.close();

// Display array

cout << endl;

cout << "Array" << endl << "----------------" << endl;

for (i = 0; i < n; i++)

{

cout << map[i] << " ";

}

cout << endl;

cout << endl;

cout << "Heapified Array" << endl << "----------------" << endl;

// call heapify

heapify(map, n);

for (i = 0; i < n; i++)

{

cout << map[i] << " ";

}

cout << endl << endl;

//this section is used to show the removed value from the heapified array/tree and to show

//the array after the removal

cout << "Removed:" << heap\_remove(map, n) << endl << "----------------" << endl;

cout << "\nAfter Remove" << endl << "----------------" << endl;

for (i = 0; i < n; i++)

{

cout << map[i] << " ";

}

cout << endl;

return;

}

13. Compile and run the program. If your heapify() function works correctly, you should see this displayed:

Sample Run

Enter file name: ***Lab23.txt***

Array

----------------

Mark Frank Alisa Susan Steve Kathy Michael Richard Lisa David Rochelle John Lailee Carlos Ann

Heapified Array

----------------

Susan Steve Michael Richard Rochelle Lailee Kathy Frank Lisa David Mark Alisa John Carlos Ann

Removed: Susan

After Remove

----------------

Steve Rochelle Michael Richard Mark Lailee Kathy Frank Lisa David Ann Alisa John Carlos

Text

Description automatically generated

14. Implement the heap\_sort() function. It will use the heap\_remove()function in a loop. Follow the algorithm explained in class:

* Loop while n>0
  + Call heap\_remove(map,n)

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heapsort()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void heap\_sort(string map[], int n)

{

//sorts the elements based on employing a heapsort - since n decrements by one in heap remove,

//will run until n is equal to zero

while (n > 0)

{

//calls the heap remove function

heap\_remove(map, n);

}

}

15. Update the main() function so that instead of calling the heap\_remove() function, it calls the heap\_sort() function. Display the array after the call to heap\_sort(), using appropriate labels.

16. Compile and run the program. If your heap\_sort() function works correctly, you should see this displayed:

Sample Run

Enter file name: ***Lab23.txt***

Array

----------------

Mark Frank Alisa Susan Steve Kathy Michael Richard Lisa David Rochelle John Lailee Carlos Ann

Heapified Array

----------------

Susan Steve Michael Richard Rochelle Lailee Kathy Frank Lisa David Mark Alisa John Carlos Ann

After Sort

----------------

Alisa Ann Carlos David Frank John Kathy Lailee Lisa Mark Michael Richard Rochelle Steve Susan

Text

Description automatically generated

Heapsort.h:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heapsort.h

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <string>

using namespace std;

// Heap sort functions

inline int heap\_left(int pos);

inline int heap\_right(int pos);

inline int heap\_parent(int pos);

void heap\_insert(string map[], int& n, string value);

string heap\_remove(string map[], int& n);

void heapify(string map[], int n);

void heap\_sort(string map[], int n);

Heapsort.cpp:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heapsort.cpp

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "Heapsort.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Left()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//function to return the index of left child of node at index pos

//for inserting or removing nodes accordingly

inline int heap\_left(int pos)

{

return 2 \* pos + 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Right()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// function to return the index of right child of node at index pos

//for inserting or removing nodes accordingly

inline int heap\_right(int pos)

{

return 2 \* pos + 2;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Parent()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// function to return the index of parent child (or parent at root node) of node at index pos

inline int heap\_parent(int pos)

{

return (pos - 1) / 2;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Insert()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// function to insert val into array map to maintain the max heap

//new node gets compared to parent node, and if larger

//new value gets swapped with value above

void heap\_insert(string map[], int& n, string value)

{

map[n] = value; // insert value at the end of map

int pos = n; // set pos to n

// get the index of parent of pos through call to heap parent

int parent = heap\_parent(pos);

//at this point, since the new node is larger than its parent node it will be swapped

while ((pos > 0) && (map[pos] > map[parent]))

{

// swap the elements at pos and parent

string temp = map[pos];

map[pos] = map[parent];

map[parent] = temp;

pos = parent; // set pos to parent

parent = heap\_parent(pos); // get the index of parent of pos

}

n++; // increment n

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heapify()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//function that takes unorded array and turns it into a heap

//based on process of heapification - requires heap\_insert

void heapify(string map[], int n)

{

//counter

int i = 0;

//while less than size of array, insert elements into heap

while (i < n)

{

//use heap\_insert recursively for tree heapification of the current tree/sub-tree

heap\_insert(map, i, map[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Remove()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//function to remove nodes from a heap

//starts at root node, moves node at bottom of tree to root,

// and then another heapification algorithm is used:

//current node compared to two children for removal here

string heap\_remove(string map[], int& n)

{

int pos = 0;

//decrement n by one

n--;

//compare ccurrent node to two children and if it isn't

//bigger than them, swap with larger of the two

swap(map[pos], map[n]);

while (pos < n)

{

int index = pos;

int left = heap\_left(pos);

int right = heap\_right(pos);

//if the left child is larger than the root

if ((left < n) && (map[left] > map[index]))

{

//use the left child as the new index

index = left;

}

//if right child greater than the largest so far

if ((right < n) && (map[right] > map[index]))

{

//use the right child as the index

index = right;

}

//if the largest current value is not at the root

if (index != pos)

{

//then swap the two values

swap(map[pos], map[index]);

pos = index;

}

else

{

//sets position to n

pos = n;

}

}

return map[n];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heapsort()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void heap\_sort(string map[], int n)

{

//sorts the elements based on employing a heapsort - since n decrements by one in heap remove,

//will run until n is equal to zero

while (n > 0)

{

//calls the heap remove function

heap\_remove(map, n);

}

}

Main.cpp:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Lab23.cpp

\* Twymun K. Safford

\* Last Updated:11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

#define ARRAY\_MAX 1000

#include "Heapsort.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* main()

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

void main()

{

int i, n;

string fname, name, map[ARRAY\_MAX];

fstream in;

// Initialize

n = 0;

// Get file name

cout << "Enter file name: ";

cin >> fname;

// Read from file

in.open(fname.data(), ios::in);

if (!in.is\_open()) return;

// Loop through file

while (!in.eof())

{

in >> name;

if (in.good()) map[n++] = name;

//if (in.good()) heap\_insert(map, n, name);

};

// Close file

in.close();

// Display array

cout << endl;

cout << "Array" << endl << "----------------" << endl;

for (i = 0; i < n; i++)

{

cout << map[i] << " ";

}

cout << endl;

cout << endl;

cout << "Heapified Array" << endl << "----------------" << endl;

//call heapify to heapify the contents of the array

heapify(map, n);

for (i = 0; i < n; i++)

{

//prints out the elements of the heapified array

cout << map[i] << " ";

}

cout << endl << endl;

//this section is used to show the removed value from the heapified array/tree and to show

//the array after the removal

/\* cout << "Removed:" << heap\_remove(map, n) << endl << "----------------" << endl;

cout << "\nAfter Remove" << endl << "----------------" << endl;\*/

//used to show what the array looks like after sorting

cout << "\nAfter Sort" << endl << "----------------" << endl;

//call heap sort function to sort the array

heap\_sort(map, n);

for (i = 0; i < n; i++)

{

//prints out the elements of the sorted aray

cout << map[i] << " ";

}

cout << endl;

return;

}

Extra Credit (5 points)

Do this only after completing and handing in the lab.

Update your heap\_sort() function:

* Add the code from heapify() to create the initial heap order.
* Change the loop to incorporate the code from heap\_remove() to move values to the end of the array.
* Convert to a template function, so that different data types can be used.

Modify main() so that it tests the sort with strings, and an array of integers or floats. Hand in your heap\_sort() code and outputs.

**Output – Extra Credit:**

**Text

Description automatically generated**

**Heapsort.h (updated with templates):**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heapsort.h

\* Twymun K. Safford

\* Last Updated: 11/17/2021

\* For extra credit, implemetnted with

\* templates

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef HEAPSORT\_H

#define HEAPSORT\_H

#include <iostream>

#include <string>

using namespace std;

// Heap sort functions

inline int heap\_left(int pos);

inline int heap\_right(int pos);

inline int heap\_parent(int pos);

//template implementation of heap\_inseert

template <typename T> void heap\_insert(T map[], int& n, T value);

//void heap\_insert(string map[], int& n, string value);

//template implementation of heap remove

template <typename T> T heap\_remove(T map[], int& n);

//string heap\_remove(string map[], int& n);

//void heapify(string map[], int n);

//template implementation of heapify

template <typename T> void heapify(T map[], int n);

//template implementation of heap\_sort

template <typename T> void heap\_sort(T map[], int n);

//void heap\_sort(string map[], int n);

//include Heapsort.cpp to get the template functions to work

// - the compiler needs to have access to the implementation of the methods

//so implementattion for the templates are separate from declaration

#include "Heapsort.cpp"

#endif // HEAPSORT\_H

**Heapsort.cpp (updated with templates):**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heapsort.cpp

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#ifndef HEAPSORT\_CPP

#define HEAPSORT\_CPP

#include "Heapsort.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Left()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//function to return the index of left child of node at index pos

//for inserting or removing nodes accordingly

inline int heap\_left(int pos)

{

return 2 \* pos + 1;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Right()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// function to return the index of right child of node at index pos

//for inserting or removing nodes accordingly

inline int heap\_right(int pos)

{

return 2 \* pos + 2;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Parent()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// function to return the index of parent child (or parent at root node) of node at index pos

inline int heap\_parent(int pos)

{

return (pos - 1) / 2;

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Insert()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// function to insert val into array map to maintain the max heap

//new node gets compared to parent node, and if larger

//new value gets swapped with value above

template <typename T> void heap\_insert(T map[], int& n, T value)

//void heap\_insert(string map[], int& n, string value)

///Now updated to become a template function using the following:

{

//integers position and parent

int pos;

//parent

int parent;

map[n] = value; // insert value at the end of map

pos = n; // set pos to n

// get the index of parent of pos through call to heap parent

parent = heap\_parent(pos);

//declare the string variable temp

//string temp;

T temp; //changed to template type T

//at this point, since the new node is larger than its parent node it will be swapped

while ((pos > 0) && (map[pos] > map[parent]))

{

// swap the elements at pos and parent

temp = map[pos];

map[pos] = map[parent];

map[parent] = temp;

pos = parent; // set pos to parent

parent = heap\_parent(pos); // get the index of parent of pos

}

n++; // increment n

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heapify()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//function that takes unorded array and turns it into a heap

//based on process of heapification - requires heap\_insert

// change it to template function

template <typename T> void heapify(T map[], int n)

//void heapify(string map[], int n)

{

//counter

int i = 0;

//while less than size of array, insert elements into heap

while (i < n)

{

//use heap\_insert recursively for tree heapification of the current tree/sub-tree

heap\_insert(map, i, map[i]);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heap Remove()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//function to remove nodes from a heap

//starts at root node, moves node at bottom of tree to root,

// and then another heapification algorithm is used:

//current node compared to two children for removal here

// change it to template function

template <typename T> T heap\_remove(T map[], int& n)

//string heap\_remove(string map[], int& n)

{

int pos = 0;

//decrement n by one

n--;

//left, right, and index

int left;

int right;

int index;

//compare ccurrent node to two children and if it isn't

//bigger than them, swap with larger of the two

swap(map[pos], map[n]);

while (pos < n)

{

index = pos;

left = heap\_left(pos);

right = heap\_right(pos);

//if the left child is larger than the root

if ((left < n) && (map[left] > map[index]))

{

//use the left child as the new index

index = left;

}

//if right child greater than the largest so far

if ((right < n) && (map[right] > map[index]))

{

//use the right child as the index

index = right;

}

//if the largest current value is not at the root

if (index != pos)

{

//then swap the two values

swap(map[pos], map[index]);

pos = index;

}

else

{

//sets position to n

pos = n;

}

}

return map[n];

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Heapsort()

\* Twymun K. Safford

\* Last Updated: 11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//void heap\_sort(string map[], int n)

template <typename T> void heap\_sort(T map[], int n)

{

//set int i and set it to 0

int i = 0;

//heapify the array to create a max heap (greatest - down to -> lowest)

//loop while i < n

while (i < n)

//call heap\_insert(map, i, map[i]);

heap\_insert(map, i, map[i]);

//now, need to incor

//sorts the elements based on employing a heapsort - since n decrements by one in heap remove,

//will run until n is equal to zero

while (n > 0)

{

//calls the heap remove function - removes the max value from map and place at index n-1

heap\_remove(map, n);

}

}

#endif

**Main.cpp (updated with templates):**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Lab23.cpp

\* Twymun K. Safford

\* Last Updated:11/16/2021

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include <iostream>

#include <fstream>

#include <string>

#define ARRAY\_MAX 1000

#include "Heapsort.h"

using namespace std;

void main()

{

int i, n;

string fname, name, map[ARRAY\_MAX];

double value, dMap[ARRAY\_MAX];

int data, iMap[ARRAY\_MAX];

fstream in;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test here for strings

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// Initialize

n = 0;

// Get file name

cout << "Enter file name to test strings: ";

cin >> fname;

// Read from file

in.open(fname.data(), ios::in);

if (!in.is\_open())

{

return;

}

// Loop through file

while (!in.eof())

{

in >> name;

if (in.good())

{

map[n++] = name;

}

//if (in.good()) heap\_insert(map, n, name);

};

// Close file

in.close();

// Display array

cout << endl;

cout << "Array" << endl << "----------------" << endl;

for (i = 0; i < n; i++)

{

cout << map[i] << " ";

}

cout << endl << endl;

cout << "Heapified Array" << endl << "----------------" << endl;

// Call heapify

heapify(map, n);

for (i = 0; i < n; i++)

{

cout << map[i] << " ";

}

cout << endl << endl;

cout << "After Sort" << endl << "----------------" << endl;

heap\_sort(map, n); // sort array in ascending order

// display array after sort

for (i = 0; i < n; i++)

{

cout << map[i] << " ";

}

cout << endl << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test here for integers

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// reset n to 0

n = 0;

// Get file name

cout << "Enter file name to test for integers: ";

cin >> fname;

// Read from file

in.open(fname.data(), ios::in);

if (!in.is\_open()) return;

// Loop through file

while (!in.eof())

{

in >> data;

if (in.good()) iMap[n++] = data;

//if (in.good()) heap\_insert(map, n, name);

};

// Close file

in.close();

// Display array

cout << endl;

cout << "Array" << endl << "----------------" << endl;

for (i = 0; i < n; i++)

{

//print the map for integers

cout << iMap[i] << " ";

}

cout << endl << endl;

cout << "Heapified Array" << endl << "----------------" << endl;

// Call heapify

heapify(iMap, n);

for (i = 0; i < n; i++)

{

cout << iMap[i] << " ";

}

cout << endl << endl;

cout << "After Sort" << endl << "----------------" << endl;

heap\_sort(iMap, n); // sort the array in ascending order

// display array after sort

for (i = 0; i < n; i++)

{

cout << iMap[i] << " ";

}

cout << endl << endl;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Test here for doubles

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// reset n to 0

n = 0;

// Get file name

cout << "Enter file name to test for doubles: ";

cin >> fname;

// Read from file

in.open(fname.data(), ios::in);

if (!in.is\_open())

{

return;

}

// Loop through file

while (!in.eof())

{

in >> value;

if (in.good())

{

//print out the map for doubles

dMap[n++] = value;

}

//if (in.good()) heap\_insert(map, n, name);

};

// Close file

in.close();

// Display array

cout << endl;

cout << "Array" << endl << "----------------" << endl;

for (i = 0; i < n; i++)

{

cout << dMap[i] << " ";

}

cout << endl << endl;

cout << "Heapified Array" << endl << "----------------" << endl;

// Call heapify

heapify(dMap, n);

for (i = 0; i < n; i++)

{

//print out the map for doubles

cout << dMap[i] << " ";

}

cout << endl << endl;

cout << "After Sort" << endl << "----------------" << endl;

heap\_sort(dMap, n); // sort the array in ascending order

// display array after sort

for (i = 0; i < n; i++)

{

cout << dMap[i] << " ";

}

cout << endl << endl;

return;

}