WAYNE STATE UNIVERSITY
COLLEGE OF ENGINEERING

Computer Science Department

CSC 2201: Computer Science II – Lab
Lab11

Description:

You will implement Binary Search Tree using linked list.

Goals:

Learn how to implement Heap ADT using C++ pointers.

Book Reference:

Please read chapter 11 on the lab book carefully.

Where to Start:

1. Download Lab11_Work.zip from Blackboard
2. Unzip the file
3. Open the solution in Microsoft Visual Studio
4. Make sure that the project does not show any compile errors.
5. Implement methods and operations of the Heap class:

5.1 Implement `Heap<DataType,KeyType,Comparator>:: Heap ( int maxNumber )`

```
// Creates an empty heap. Allocates enough memory for maxNumber
// data items.
```

5.2 Implement `Heap<DataType,KeyType,Comparator>:: Heap ( const Heap& other )`

```
// Copy constructor.
```

5.3 Implement `BSTree<DataType, KeyType>::BSTree ( const BSTree<DataType,KeyType>& other )`

```
// Copy constructor.
```

5.4 Implement `operator = ( const Heap& other )`

```
// Sets a heap to be equivalent to the heap "other".
```

5.5 Implement `~Heap ()`

```
// Frees the memory used by a heap.
```

5.6 Implement `insert(const DataType &newDataItem)`

```
// Inserts newDataItem into a heap. This data item is initially
```

```
// inserted as the bottom rightmost data item in the heap. It is
then
// moved upward until a valid heap is produced.
```

5.7 Implement `remove ()`

```
// Removes the data item with the highest priority (the root) from a
// heap and returns it. This data item is replaced with the bottom
// rightmost data item, which is moved downward until a valid heap
is
// produced.
```

5.8 Implement `clear ()`

```
// Removes all the data items from a heap.
```

5.9 Implement `isEmpty ()`

```
// Returns true if a heap is empty. Otherwise, returns false.
```

5.10 Implement `isFull ()`

```
// Returns true if a heap is full. Otherwise, returns false.
```

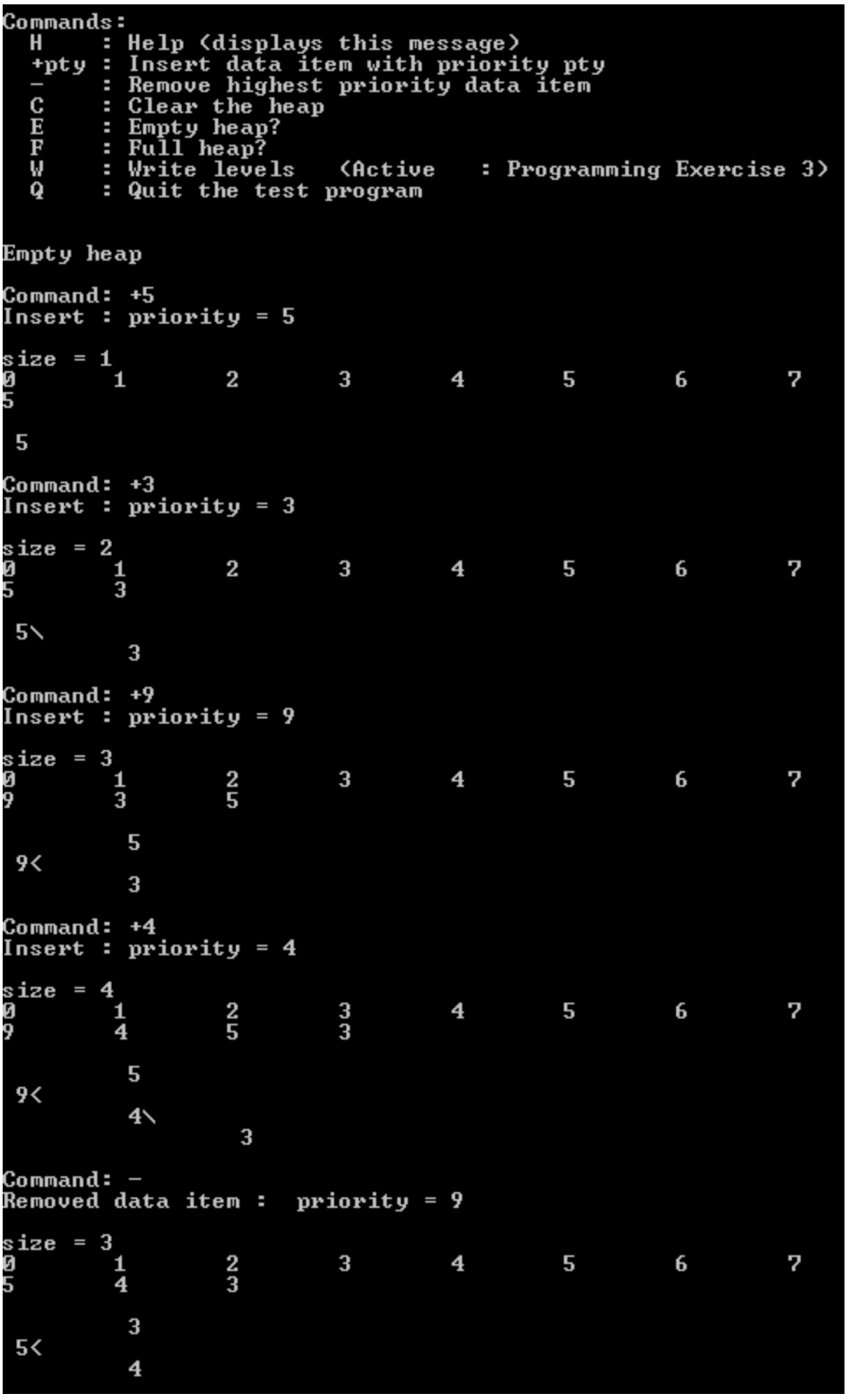
Activate `LAB11_TEST1` in the config.h file.

5.11 Implement `writeLevels ()`

```
// Outputs the data items in a heap in level order, one level per
line.
// Only outputs the priority for each data item.
```

6. Test your implementation using the program in the file test11.cpp.

7. Output:


Commands:
H : Help (displays this message)
+pty : Insert data item with priority pty
- : Remove highest priority data item
C : Clear the heap
E : Empty heap?
F : Full heap?
W : Write levels (Active : Programming Exercise 3)
Q : Quit the test program
Empty heap
Command: +5
Insert : priority = 5
size = 1
0 1 2 3 4 5 6 7
5
5
Command: +3
Insert : priority = 3
size = 2
0 1 2 3 4 5 6 7
5 3
5\
3
Command: +9
Insert : priority = 9
size = 3
0 1 2 3 4 5 6 7
9 3 5
5
9<
3
Command: +4
Insert : priority = 4
size = 4
0 1 2 3 4 5 6 7
9 4 5 3
5
9<
4\
3
Command: -
Removed data item : priority = 9
size = 3
0 1 2 3 4 5 6 7
5 4 3
3
5<
4

```
Command: E
Heap is NOT empty

size = 3
0       1       2       3       4       5       6       7
5       4       3

         3
 5<
         4

Command: F
Heap is NOT full

size = 3
0       1       2       3       4       5       6       7
5       4       3

         3
 5<
         4

Command: W
Levels :
5
4 3

size = 3
0       1       2       3       4       5       6       7
5       4       3

         3
 5<
         4

Command: C
Clear the heap

Empty heap
```

8. Implement methods and operations in a class called priorityQueue.cpp:

   8.1 Implement `PriorityQueue ( int maxNumber )`
   ```
   // Creates an empty priority queue.
   ```

   8.2 Implement `enqueue ( const DataType &newDataItem )`
   ```
   // Inserts newDataItem into a priority queue.
   ```

   8.3 Implement `dequeue ()`
   ```
   // Removes the least recently added (front) data item from a priority
   // queue and returns it.
   ```

9. Test your implementation using the program in the file test11pq.cpp.

10. Output:

```
Commands:
  +x : Enqueue data item with priority x
  -  : Dequeue data item
  C  : Clear the queue
  E  : Empty queue?
  F  : Full queue?
  H  : Print this help message
  Q  : Quit the test program


Empty heap

Command (H for help): +4
Enqueue :  pty = 4

size = 1
0       1       2       3       4       5       6       7
4

 4

Command (H for help): +6
Enqueue :  pty = 6

size = 2
0       1       2       3       4       5       6       7
4       6

 4\
         6

Command (H for help): +9
Enqueue :  pty = 9

size = 3
0       1       2       3       4       5       6       7
4       6       9

         9
 4<
         6

Command (H for help): -
Dequeued :  pty = 4

size = 2
0       1       2       3       4       5       6       7
9       6

 9\
         6

Command (H for help): E
Queue is NOT empty

size = 2
0       1       2       3       4       5       6       7
9       6

 9\
         6

Command (H for help): F
Queue is NOT full

size = 2
0       1       2       3       4       5       6       7
9       6

 9\
         6
```

11. Implement heap sort in a class called heapsort.cpp

12. Test your implementation using the program in the file test11hs.cpp.

13. Output:

```
Enter up to 10 priorities (end with EOF) : 1 5 8 62 4 23 12 85 45 21
Unsorted array : 1 5 8 62 4 23 12 85 45 21
Sorted array   : 1 4 5 8 12 21 23 45 62 85
Press any key to continue . . .
```

Create a Zip file of your solution:

1. Right click on your solution in Solution Explorer
2. Click on "Open Folder in File Explorer"
3. Go one level up in file explorer
4. Right click on your solution folder
5. Add it to archive by creating a zip file

Upload the zipped file on Blackboard:

1. Go to Blackboard
2. Click on this course (CSC 2201: Computer Science II Lab)
3. Go to the folder "Labs"
4. Click on the "Lab11_Work" assignment
5. Upload your zipped file