

|  |
| --- |
| Title: Implementation of priority queue program using Heap |

**Objective:** To understand the use of heap by performing Prority Queue program.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **3** | Demonstrate the priority queue using heap |

**Books/ Journals/ Websites referred:**

1. Data Structures A Psedocode Approach with C, Richard F. Gilberg & Behrouz A. Forouzan, second edition, CENGAGE Learning
2. Y. Langsam, M. Augenstin and A. Tannenbaum, Data Structures using C, Pearson Education Asia, Second Edition, 2002, ISBN 978-81-317-0229-1
3. E. Horowitz, S. Sahni, S.Anderson-freed, Fundamentals of Data Structures in C, 2nd Edition, University Press, ISBN 978-81-7371-605-8

Introduction to Data Structure and its Applications Jean-Paul Tremblay, P. G. Sorenson

**Abstract**:-

A heap is a useful and efficient way to store and look up data that must maintain order. The classic example is a priority queue abstract data type. A priority queue is a set of data where higher or lower valued data points bubble to the front of the queue and are therefore accessed first. We’re going to use that in our experiment to see how a heap can improve lookup efficiency

We will be creating a priority queue using heap.

**Related Theory: -**

A Priority Queue is an abstract data type to efficiently support finding an item with the highest priority across a series of operations. The basic operations are :   
1. Insert,  
2. Find - minimum (or maximum), and  
3. Delete-minimum (or maximum).

Some implementations also efficiently support, join two priority queues (meld), delete an arbitrary item, and increase the priority of a item (decrease-key).

A heap is a specialized tree-based data structure that satisfies the heap property :if B is a child node of A, then key(A) ≥ key(B). This implies that an element with the greatest key is always in the root node, and so such a heap is sometimes called a max-heap.

(Alternatively, if the comparison is reversed, the smallest element is always in the root node, which results in a min-heap). The several variants of heaps are the prototypical most efficient implementations of the abstract data type priority queues.

Priority queues are useful in many applications. In particular, heaps are crucial in several efficient graph algorithms.

Heaps are usually implemented in an array, and don't require pointers between elements.  
The operations commonly performed with a heap are :

* delete-max or delete-min : removing the root node of a max or min heap, respectively.
* increase-key or decrease-key : updating a key within a max or min heap, respectively.
* insert : adding a new key to the heap.
* Merge:joining two heaps to form a valid new heap containing all the elements of both.

**Implementation Details:**

**Program**

#include<stdio.h>

#include<math.h>

#define MAX 100

void swap(int\*,int\*);

main()

{

int choice,num,n,a[MAX],data,s;

void display(int[],int);

void insert(int[],int,int,int);

int del(int[],int,int);

n=0;

int lb=0;

while(1)

{

printf("\n.....MAIN MENU....\n");

printf("1.Insert\n");

printf("2.Delete\n");

printf("3.Display\n");

printf("4.Quit\n");

printf("Enter your choice : ");

scanf("%d",&choice);

switch(choice)

{

case 1:

printf("Enter data to be inserted : ");

scanf("%d",&data);

insert(a,n,data,lb);

n++;

break;

case 2:

s=del(a,n+1,lb);

if(s!=0)

printf("\nThe deleted value is : %d \n",s);

if(n>0)

n--;

break;

case 3:

printf("\n");

display(a,n);

break;

case 4:

return;

default:

printf("Invalid choice.n");

}

printf("nn");

}

}

void insert(int a[],int heapsize,int data,int lb)

{

int i,p;

int parent(int);

if(heapsize==MAX)

{

printf("Queue Is Full!!n");

return;

}

i=lb+heapsize;

a[i]=data;

while(i>lb&&a[p=parent(i)]<a[i])

{

swap(&a[p],&a[i]);

i=p;

}

}

int del(int a[],int heapsize,int lb)

{

int data,i,l,r,max\_child,t;

int left(int);

int right(int);

if(heapsize==1)

{

printf("Queue Is Empty!!n");

return 0;

}

t=a[lb];

swap(&a[lb],&a[heapsize-1]);

i=lb;

heapsize--;

while(1)

{

if((l=left(i))>=heapsize)

break;

if((r=right(i))>=heapsize)

max\_child=l;

else

max\_child=(a[l]>a[r])?l:r;

if(a[i]>=a[max\_child])

break;

swap(&a[i],&a[max\_child]);

i=max\_child;

}

return t;

}

int parent(int i)

{

float p;

p=((float)i/2.0)-1.0;

return ceil(p);

}

int left(int i)

{

return 2\*i+1;

}

int right(int i)

{

return 2\*i+2;

}

void display(int a[],int n)

{

int i;

if(n==0)

{

printf("Queue Is Empty!!n");

return;

}

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

printf("%d ",a[i]);

printf("\n");

}

void swap(int\*p,int\*q)

{

int temp;

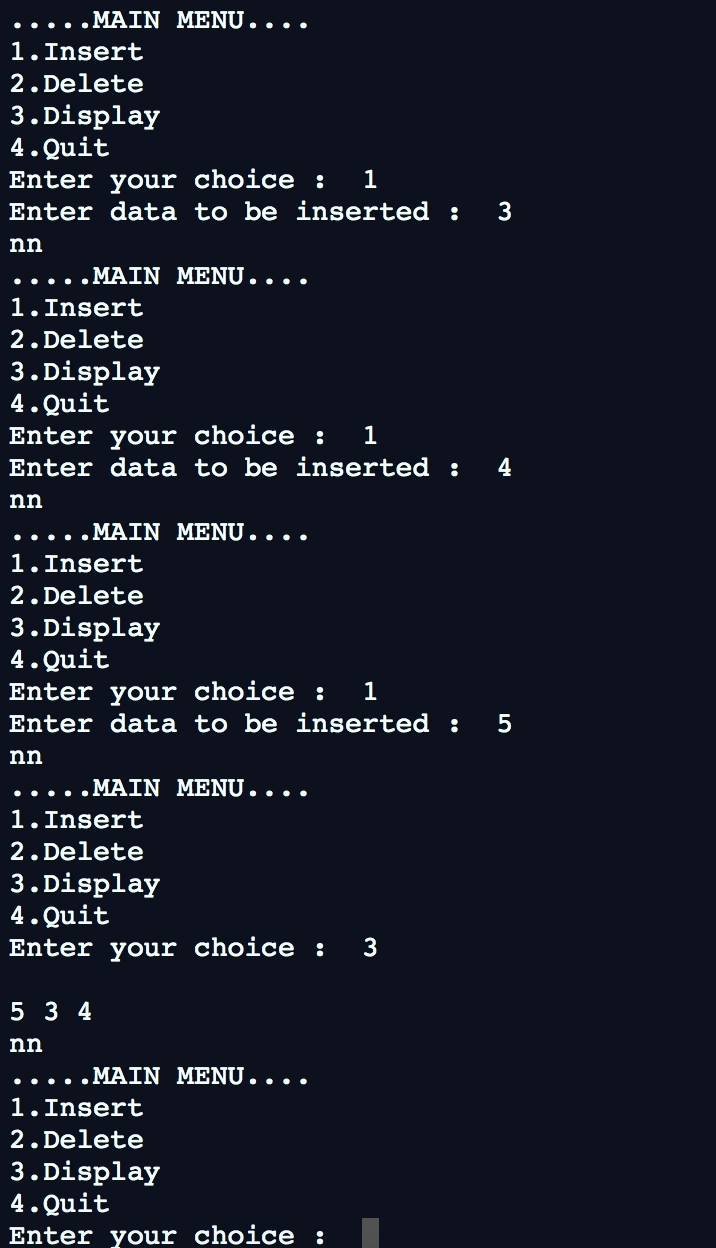
temp=\*p;

\*p=\*q;

\*q=temp;

}

**OUTPUT**

****

1. **Enlist all the Steps followed and various options explored**

**Algorithm**

Insertion.  
/\*This algorithm inserts an element in the queue\*/  
Algo\_insert(a[],int heapsize,data,lb)  
{  
if(heapsize==MAX)/\*MAX denotes maximum size of queue\*/  
{  
printf(Queue Is Full!);  
exit from program;  
}  
i=lb+heapsize;  
a[i]=data;  
while(i>lb AND a[p=parent(i)]<a[i])  
{  
swap(a[p],a[i]);  
i=p;  
}  
}

; Deletion.  
  
/\*This function deletes an element from the queue\*/  
int del\_hi\_priori(a[], heapsize, lb)  
{   
if(heapsize==1)  
{  
printf(Queue Is Empty!);  
exit from program;  
}  
t=a[lb];  
swap(a[lb],a[heapsize-1]);  
i=lb;  
heapsize=heapsize-1;  
while(1)  
{  
if((l=left(i))>=heapsize)  
exit;  
if((r=right(i))>=heapsize)  
max\_child=l;  
else  
max\_child=(a[l]>a[r])?l:r;  
if(a[i]>=a[max\_child])  
exit;  
swap(&a[i],&a[max\_child]);  
i=max\_child;  
}  
return t;  
}

**Explain the Importance of the approach followed by you**

To get better performance, priority queues typically use a heap as their backbone, giving O(log n) performance for inserts and removals.  
λ There are several applications of priority Queue :  
1. Priority queuing can be used to manage limited resources such as bandwidth on a transmission line from a network router.  
2. Another use of a priority queue is to manage the events in a discrete event simulation. The events are added to the queue with their simulation time used as the priority.  
3. When the graph is stored in the form of adjacency list or matrix, priority queue can be used to extract minimum efficiently when implementing Dijkstra's algorithm.

**Conclusion:- Successfully implemented Priority Queue using heap**