TREAP IMPLEMENTATION

- struct Treap:- It is an Structure with key ,priority right left and parent as its members.

  key is used to store data given by the user

  priority is used to store randomly generated value to satisfy the heap Property.

  right ,left, parent are the pointer to right Child ,Left child, Parent respectively of a node.

- Treap* newNode(int item):- This function is used to create a new node an set the

  values of key to item(given by the user). Priority has been assigned a randomvalue.

- struct Treap *RightRotate(struct Treap *y):- Perform right Rotation on Node y.
- struct Treap *LeftRotate(struct Treap *x):- Perform Left Rotation on Node X
- struct Treap* search(struct Treap* root , int key) :-In this function if root is NULL or key vale of root is equal to the value that user Want to search. the our search is over . But if the key value is greater than the root->value the cal the function recursively on Root->right. or if the key vale is less than the root->value then call the function recursively on left child.
- struct Treap* insert(struct Treap* Treap,int key):-To insert a node in the treap.*Treap is pointer to the root node, key is the value to enter. First arrange the node in such a way they satisfy the BST property and the arrange them on the basis of there priority value . To satisfy the MaxHEAP property. **Note-: I Have used max Heap**.
- int Height(struct Treap * root):- *root is the pointer to root node . In This function I have calculated size of left and right sub tree recursively and the compared the two , which ever is the greater is the height of the treap.
- void store(int val):- This function just store the Treap in an Array in ascending order.
- void Predecessor(int key) and void Successor(int key) :- key the value of which predecessor or successor has to be found out. Both these function simply search for the key value in the array and give there preceding and successive key value.
- void print(struct Treap * temp):- it simply print all the elements, with *temp as root node.
- struct Treap * deleteNode(struct Treap *root,int key) :- key is the value that has to be deleted ,
  - if root is null the return the function .
  - if key value is less than key value of the root node then go to the left sub tree. and perform deletion recursively of the left child of the root.
  - if key value is greater than key value of the root node then go to the lright sub tree. and perform deletion recursively of the right child of the root.
  - if key is at root and there is no left child make right child as root
  - if key is at root and there is no right Child , Then make the left child as root.
  - If key is at root and both left and right are not NULL then , perform right and left rotation respectively at key.
- int main()-:Asked the user to make choice whether he/she want to insert, delete, search,print all key value, Find predecessor or successor of any value. then performing the asked task.