**ADS LAB ASSIGNMENT-10**

**NAME:** Manasvi M. Patil

**CLASS:** SY-IT-C

**ROLL NO.:** 03

**AIM:**Write C/C++ program to create heap data structure and perform insertion.

**THEORY:**

The provided code revolves around implementing a max-heap data structure. A max-heap is a complete binary tree where each node satisfies the max-heap property: the value of each node is greater than or equal to the values of its children. This implementation employs an array-based representation for the heap, with indices representing the tree nodes. The concepts utilized in this code include:

1. **Array Representation:** The array-based representation of the heap allows easy access to parent and child nodes using index calculations (`parent = (index - 1) / 2`, `left\_child = 2 \* index + 1`, `right\_child = 2 \* index + 2`), simplifying heap manipulation.

2. **Insertion and Heapify Up:** The `insert` function adds a new element at the end of the array and maintains the max-heap property by using `heapifyUp`, which swaps the newly inserted element with its parent until the heap condition is satisfied, ensuring the maximum value is at the root.

3. **Heap Property Maintenance:** The code ensures that after every insertion, the max-heap property holds true, allowing efficient retrieval of the maximum element (root) and maintaining the integrity of the heap structure.

4. **Limitation Handling:** The code includes a limit on the heap size (`MAX\_HEAP\_SIZE`) to prevent overflow and warns when attempting to insert beyond this limit.

**CODE:**

#include <stdio.h>

#include <stdlib.h>

#define MAX\_HEAP\_SIZE 100

struct Heap {

    int array[MAX\_HEAP\_SIZE];

    int size;

};

// Function to swap two values

void swap(int \*a, int \*b) {

    int temp = \*a;

    \*a = \*b;

    \*b = temp;

}

// Function to maintain the heap property (upward)

void heapifyUp(struct Heap \*heap, int index) {

    int parent = (index - 1) / 2;

    while (index > 0 && heap->array[parent] < heap->array[index]) {

        swap(&heap->array[parent], &heap->array[index]);

        index = parent;

        parent = (index - 1) / 2;

    }

}

// Function to insert a new element into the heap

void insert(struct Heap \*heap, int value) {

    if (heap->size >= MAX\_HEAP\_SIZE) {

        printf("Heap overflow: Cannot insert more elements\n");

        return;

    }

    heap->array[heap->size] = value;

    heap->size++;

    heapifyUp(heap, heap->size - 1);

}

// Function to display heap elements

void displayHeap(struct Heap \*heap) {

    printf("Heap elements: ");

    for (int i = 0; i < heap->size; ++i) {

        printf("%d ", heap->array[i]);

    }

    printf("\n");

}

int main() {

    struct Heap heap;

    heap.size = 0;

    // Insert elements into the heap

    insert(&heap, 10);

    insert(&heap, 5);

    insert(&heap, 17);

    insert(&heap, 20);

    insert(&heap, 8);

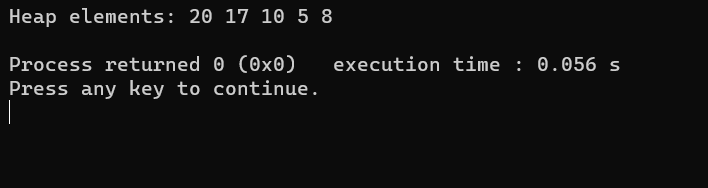
    // Display heap elements

    displayHeap(&heap);

    return 0;

}

**OUTPUT:**

****

**CONCLUSION:**

The presented C code establishes a basic max-heap data structure using an array-based representation. The `struct Heap` encompasses an array to store heap elements and tracks the current heap size. The `insert` function incorporates a mechanism to add new elements into the heap, employing the `heapifyUp` operation to maintain the heap's max-heap property by traversing upwards from the inserted element, ensuring that the parent nodes are consistently greater than their children. The `displayHeap` function facilitates the visualization of the heap's elements. Notably, the code has a predefined maximum heap size (`MAX\_HEAP\_SIZE`), limiting the number of elements the heap can accommodate, and warns of a "Heap overflow" condition if the insertion exceeds this limit. However, this implementation lacks other crucial heap functionalities like deletion, heapify down operations, or dynamic resizing, providing a rudimentary illustration focused primarily on insertion and heap property maintenance in a max-heap structure.

**THANK YOU!**