FALL – SEMESTER **Name: Nidhi Singh**
Course Code: MCSE501P **Reg. No:22MAI0015**
Course-Title: – Data Structures and Algorithms
DIGITAL ASSIGNMENT - IV
(LAB) **Slot-** L37+L38

Faculty: SARAVANAN R - SCOPE

List of programs :-

12. Write a program in C to perform bitonic sort.
13. Write a program in C to perform cocktail sort.
14. Write a program in C to perform linear search.
15. Write a program in C to perform interpolation search.
16. Write a program in C to exponential search.
17. Write a program in C to implement a binary tree and operations on it.
18. Write a program in C to implement a B tree and operations on it.

12. Write a program in C to perform bitonic sort.

```c
#include<stdio.h>
void exchange(int a[], int i, int j, int d)
{
    int temp;
    if (d==(a[i]>a[j]))
    {
        temp = a[i];
        a[i] = a[j];
        a[j] = temp;
    }
}
void merge(int a[], int beg, int c, int d)
{
    int k, i;
    if (c > 1)
    {
        k = c/2;
        for (i = beg; i < beg+k; i++)
            exchange(a, i, i+k, d);
        merge(a, beg, k, d);
        merge(a, beg+k, k, d);
    }
}
void bitonicSort(int a[],int beg, int c, int d)
{
    int k;
    if (c>1)
    {
        k = c/2;
        bitonicSort(a, beg, k, 1);
        bitonicSort(a, beg+k, k, 0);
        merge(a,beg, c, d);
    }
}
void sort(int a[], int n, int order)
{
    bitonicSort(a, 0, n, order);
}
 void print(int a[], int n)
    {
    int i;
    for(i = 0; i < n; i++)
    {
        printf("%d ",a[i]);
    }
```

```
    }
  //nidhi_singh_22mai0015-----------------------------------------------
int main()
{
    int n;
    printf("\nenter the size of array which is equal to power of 2:\t");
    scanf("%d", &n);
    int a[n];
    printf("\nenter the elements of array :\t");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    int order = 1;
    printf("Before sorting array elements are - \n");
    print(a, n);
    sort(a, n, order);
    printf("\nAfter sorting array elements are - \n");
    print(a, n);
    return 0;
}
```

Output :-

```
enter the size of array which is equal to power of 2:   16

enter the elements of array :   8
1
5
55
32
89
100
45
87
23
1
0
67
32
98
101
Before sorting array elements are -
8 1 5 55 32 89 100 45 87 23 1 0 67 32 98 101

After sorting array elements are -
0 1 1 5 8 23 32 32 45 55 67 87 89 98 100 101
PS C:\Users\User\Desktop\c program\DS LAB>
```

13. Write a program in C to perform cocktail sort.

```c
#include <stdio.h>
void cocktail(int a[], int n)
{
    int swap = 1;
    int beg = 0;
    int end = n - 1;
    int i, temp;
    while (swap)
    {
        swap = 0;
         for (i = beg; i < end; ++i)
        {
            if (a[i] > a[i + 1]) {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
                swap = 1;
            }
        }

        if (!swap)
            break;

        swap = 0;
         --end;
        for (int i = end - 1; i >= beg; --i)
        {
            if (a[i] > a[i + 1]) {
                temp = a[i];
                a[i] = a[i+1];
                a[i+1] = temp;
                swap = 1;
            }
        }
         ++beg;
    }
}

 void print(int a[], int n)
    {
    int i;
    for(i = 0; i < n; i++)
    {
        printf("%d ", a[i]);
    }
```

```
    printf("\n");
    }

  //nidhi_singh_22mai0015----------------------------------
int main()
{
    int n;
    printf("\nenter the size of array:\t");
    scanf("%d", &n);
    int a[n];
    printf("\nenter the elements of array :\t");
    for(int i=0;i<n;i++)
    {
        scanf("%d",&a[i]);
    }
    printf("Before sorting array elements are - \n");
    print(a, n);
    cocktail(a, n);
    printf("\nAfter sorting array elements are - \n");
    print(a, n);
    return 0;
}
```

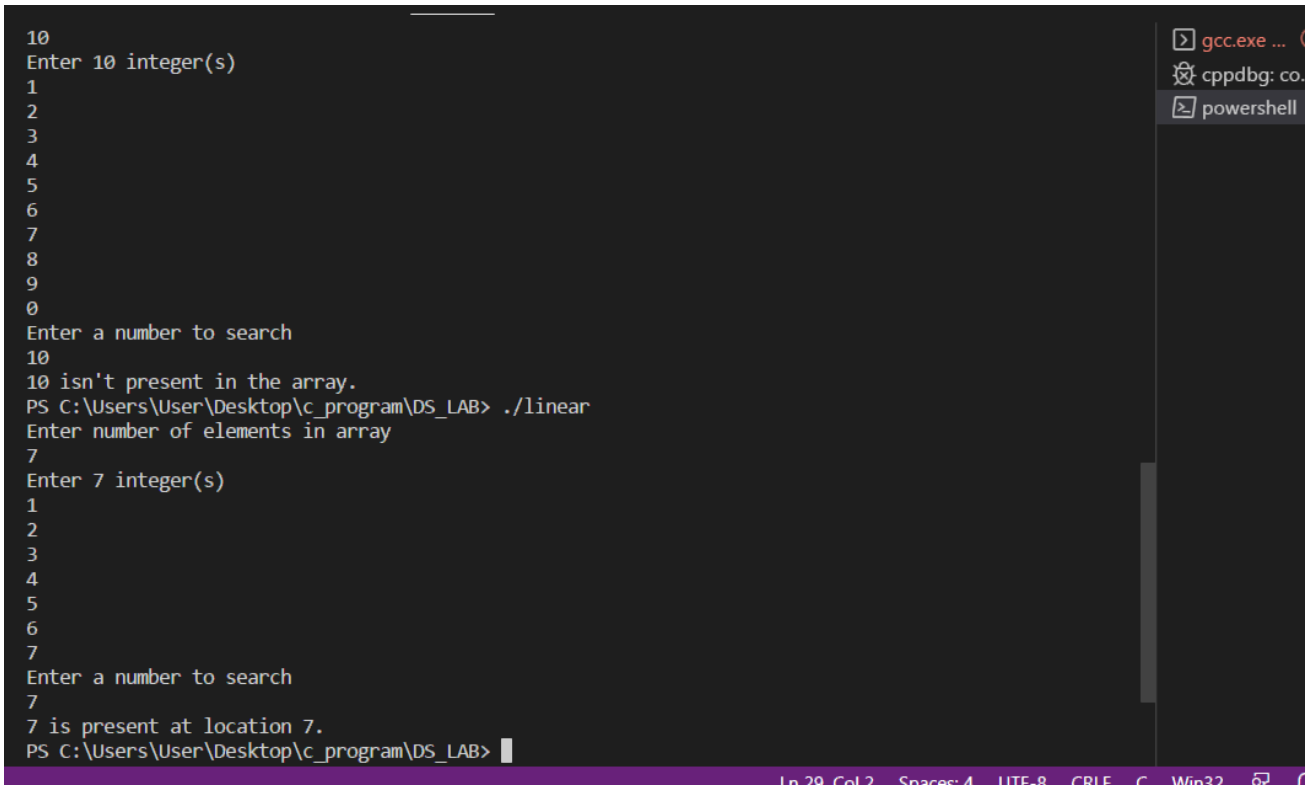
Output :-

```
PS C:\Users\User\Desktop\c_program\DS_LAB> gcc cocktail.c -o cock.exe
PS C:\Users\User\Desktop\c_program\DS_LAB> ./cock

enter the size of array:        10

enter the elements of array :   78
34
76
12
98
100
34
26
87
46
Before sorting array elements are -
78 34 76 12 98 100 34 26 87 46

After sorting array elements are -
12 26 34 34 46 76 78 87 98 100
PS C:\Users\User\Desktop\c_program\DS_LAB>
```

14. Write a program in C to perform linear search.

```c
#include <stdio.h>
int main()
{
  int array[100], search, c, n;

  printf("Enter number of elements in array\n");
  scanf("%d", &n);

  printf("Enter %d integer(s)\n", n);

  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
//nidhi_singh_22mai0015_____________________________
  printf("Enter a number to search\n");
  scanf("%d", &search);

  for (c = 0; c < n; c++)
  {
    if (array[c] == search)
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);

  return 0;
}
```

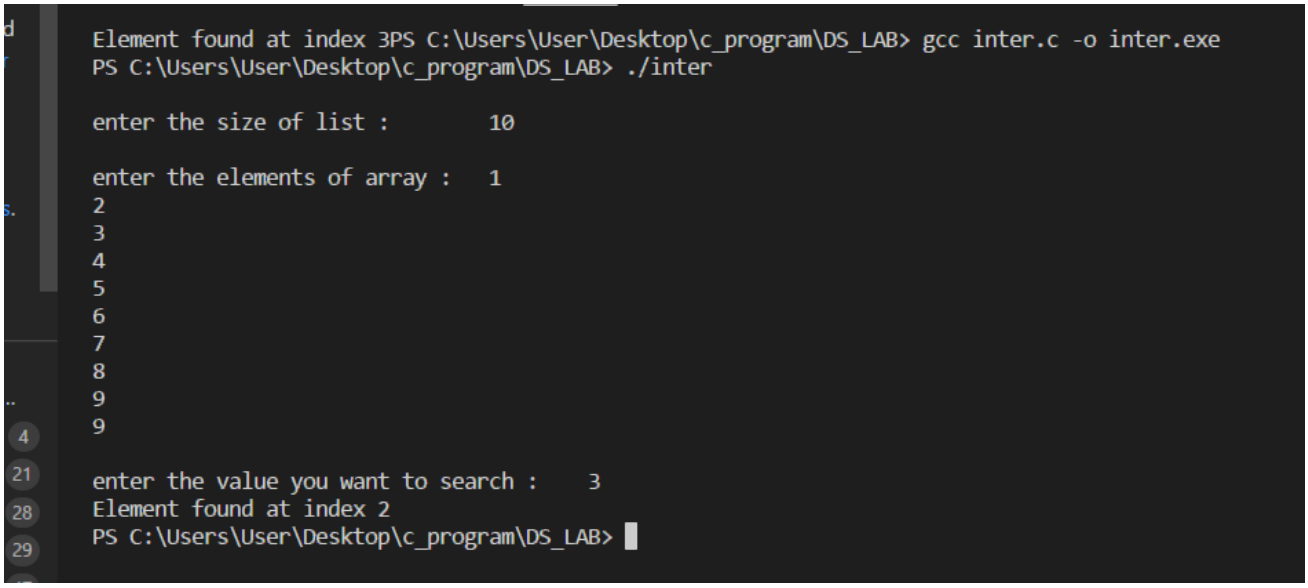
Output :-


10
Enter 10 integer(s)
1
2
3
4
5
6
7
8
9
0
Enter a number to search
10
10 isn't present in the array.
PS C:\Users\User\Desktop\c_program\DS_LAB> ./linear
Enter number of elements in array
7
Enter 7 integer(s)
1
2
3
4
5
6
7
Enter a number to search
7
7 is present at location 7.
PS C:\Users\User\Desktop\c_program\DS_LAB>
gcc.exe ...
cppdbg: co.
powershell

15. Write a program in C to perform interpolation search.

```c
 #include <stdio.h>
int interpolationSearch(int arr[], int lo, int hi, int x)
{
    int pos;
    if (lo <= hi && x >= arr[lo] && x <= arr[hi]) {
         pos = lo
            + (((double)(hi - lo) / (arr[hi] - arr[lo]))
                * (x - arr[lo]));
        if (arr[pos] == x)
            return pos;
        if (arr[pos] < x)
            return interpolationSearch(arr, pos + 1, hi, x);
    if (arr[pos] > x)
            return interpolationSearch(arr, lo, pos - 1, x);
    }
    return -1;
}
int main()
{
 int x,size;
    printf("\nenter the size of list :\t");
    scanf("%d", &size);
    int arr[size];
    printf("\nenter the elements of array :\t");
    for(int i=0;i<size;i++)
    {
        scanf("%d",&arr[i]);
    }
    printf("\nenter the value you want to search :\t");
    scanf("%d", &x);
    //nidhi_singh_22mai0015-------------------------
     int index = interpolationSearch(arr, 0, size - 1, x);
    if (index != -1)
        printf("Element found at index %d\n", index);
    else
        printf("Element not found.");
    return 0;
}
```

Output :-

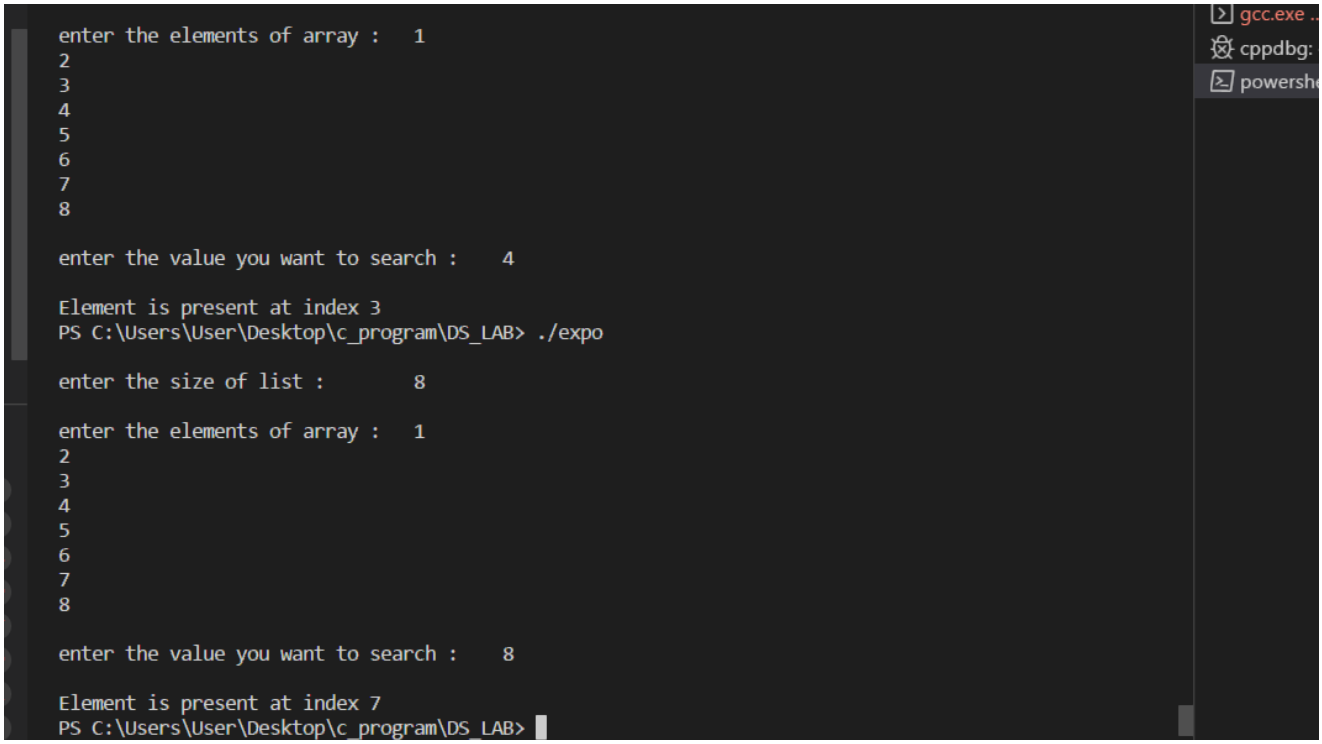

Element found at index 3PS C:\Users\User\Desktop\c_program\DS_LAB> gcc inter.c -o inter.exe
PS C:\Users\User\Desktop\c_program\DS_LAB> ./inter
enter the size of list : 10
enter the elements of array : 1
2
3
4
5
6
7
8
9
9
enter the value you want to search : 3
Element found at index 2
PS C:\Users\User\Desktop\c_program\DS_LAB>

16. Write a program in C to exponential search.

```c
// C++ program to find an element x in a
// sorted array using Exponential search.
#include <stdio.h>
#include <time.h>
#include <math.h>
#define min

int binarySearch(int arr[], int, int, int);

// Returns position of first occurrence of
// x in array
int exponentialSearch(int arr[], int n, int x)
{
    // If x is present at first location itself
    if (arr[0] == x)
        return 0;

    // Find range for binary search by
    // repeated doubling
    int i = 1;
    while (i < n && arr[i] <= x)
        i = i*2;

    // Call binary search for the found range.
    return binarySearch(arr, i/2,
                           min(i, n-1), x);
}

// A recursive binary search function. It returns
// location of x in given array arr[l..r] is
// present, otherwise -1
int binarySearch(int arr[], int l, int r, int x)
{
    if (r >= l)
    {
        int mid = l + (r - l)/2;

        // If the element is present at the middle
        // itself
        if (arr[mid] == x)
            return mid;

        // If element is smaller than mid, then it
        // can only be present n left subarray
        if (arr[mid] > x)
```

```
            return binarySearch(arr, l, mid-1, x);

        // Else the element can only be present
        // in right subarray
        return binarySearch(arr, mid+1, r, x);
    }

    // We reach here when element is not present
    // in array
    return -1;
}

// Driver code
int main(void)
{ int x,size;
    printf("\nenter the size of list :\t");
    scanf("%d", &size);
    int arr[size];
    printf("\nenter the elements of array :\t");
    for(int i=0;i<size;i++)
    {
        scanf("%d",&arr[i]);
    }
    printf("\nenter the value you want to search :\t");
    scanf("%d", &x);
    //nidhi_singh_22mai0015------------------------

int result = exponentialSearch(arr, size, x);
(result == -1)? printf("\nElement is notpresent in array") : printf("\nElement is
present at index %d",result);
printf("\n");
return 0;
}
```

Output :-


enter the elements of array : 1
2
3
4
5
6
7
8
enter the value you want to search : 4
Element is present at index 3
PS C:\Users\User\Desktop\c_program\DS_LAB> ./expo
enter the size of list : 8
enter the elements of array : 1
2
3
4
5
6
7
8
enter the value you want to search : 8
Element is present at index 7
PS C:\Users\User\Desktop\c_program\DS_LAB>
gcc.exe
cppdbg:
powersh

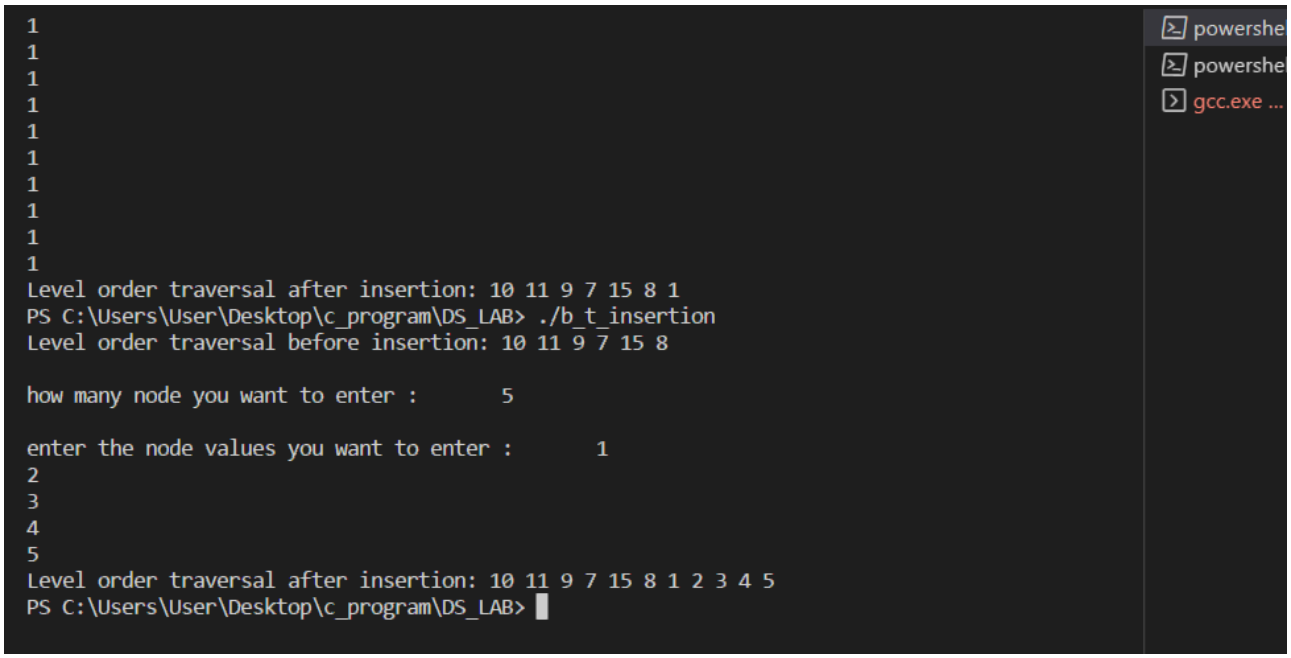
17. Write a program in C to implement a binary tree and operations on it.

```c
#include <stdio.h>
#include<stdlib.h>
/* A binary tree node has data, pointer to left child
and a pointer to right child */

struct Node {
    int data;
    struct Node * left;
    struct Node * right;
};

struct Node* newNode(int data)
{
    struct Node* Node = (struct Node*)malloc(sizeof(struct Node));
    Node->data = data;
    Node->left = NULL;
    Node->right = NULL;

    return (Node);
};

void printCurrentLevel(struct Node* root, int level);
int height(struct Node* node);

int height(struct Node* node)
{
    if (node == NULL)
        return 0;
    else {
        /* compute the height of each subtree */
        int lheight = height(node->left);
        int rheight = height(node->right);

        /* use the larger one */
        if (lheight > rheight)
            return (lheight + 1);
        else
            return (rheight + 1);
    }
}
void printLevelOrder(struct Node* root)
{
    int h = height(root);
    int i;
    for (i = 1; i <= h; i++)
        printCurrentLevel(root, i);
}
```

```c
/* Print nodes at a current level */
void printCurrentLevel(struct Node* root, int level)
{
    if (root == NULL)
        return;
    if (level == 1)
        printf("%d ", root->data);
    else if (level > 1) {
        printCurrentLevel(root->left, level - 1);
        printCurrentLevel(root->right, level - 1);
    }
}

struct node *insert(struct Node *root, int val)
{

    if(root == NULL)
        return newNode(val);

    if(root->data < val)
        root->right = insert(root->right,val);

    else if(root->data > val)
        root->left = insert(root->left,val);

    return root;
}
// Driver code
int main()
{
   struct Node* root = newNode(10);
    root->left = newNode(11);
    root->left->left = newNode(7);
    root->right = newNode(9);
    root->right->left = newNode(15);
    root->right->right = newNode(8);

    printf("Level order traversal before insertion: ");
    printLevelOrder(root);
    printf("\n");
   int n;
   printf("\nhow many node you want to enter :\t");
   scanf("%d", &n);
   int a[n];
   printf("\nenter the node values you want to enter :\t");
   for(int i=0;i<n;i++)
   {
    scanf("%d", &a[i]);
   }
    for(int i=0;i<n;i++)
```

```
  {
   root = insert(root, a[i]);
  }
   printf("Level order traversal after insertion: ");
   printLevelOrder(root);
   printf("\n");

   return 0;
}
```

Output :-



18. Write a program in C to implement a B tree and operations on it.

```c
#include <stdio.h>
#include <stdlib.h>

#define MAX 4
#define MIN 2

struct btreeNode {
      int val[MAX + 1], count;
      struct btreeNode *link[MAX + 1];
};

struct btreeNode *root;

/* creating new node */
struct btreeNode * createNode(int val, struct btreeNode *child) {
      struct btreeNode *newNode;
      newNode = (struct btreeNode *)malloc(sizeof(struct btreeNode));
      newNode->val[1] = val;
      newNode->count = 1;
      newNode->link[0] = root;
      newNode->link[1] = child;
      return newNode;
}

/* Places the value in appropriate position */
void addValToNode(int val, int pos, struct btreeNode *node,
                        struct btreeNode *child) {
      int j = node->count;
      while (j > pos) {
              node->val[j + 1] = node->val[j];
              node->link[j + 1] = node->link[j];
              j--;
      }
      node->val[j + 1] = val;
      node->link[j + 1] = child;
      node->count++;
}

/* split the node */
void splitNode (int val, int *pval, int pos, struct btreeNode *node,
   struct btreeNode *child, struct btreeNode **newNode) {
      int median, j;

      if (pos > MIN)
              median = MIN + 1;
      else
              median = MIN;
```

```
        *newNode = (struct btreeNode *)malloc(sizeof(struct btreeNode));
        j = median + 1;
        while (j <= MAX) {
                (*newNode)->val[j - median] = node->val[j];
                (*newNode)->link[j - median] = node->link[j];
                j++;
        }
        node->count = median;
        (*newNode)->count = MAX - median;

        if (pos <= MIN) {
                addValToNode(val, pos, node, child);
        } else {
                addValToNode(val, pos - median, *newNode, child);
        }
        *pval = node->val[node->count];
        (*newNode)->link[0] = node->link[node->count];
        node->count--;
}

/* sets the value val in the node */
int setValueInNode(int val, int *pval,
   struct btreeNode *node, struct btreeNode **child) {

        int pos;
        if (!node) {
                *pval = val;
                *child = NULL;
                return 1;
        }

        if (val < node->val[1]) {
                pos = 0;
        } else {
                for (pos = node->count;
                        (val < node->val[pos] && pos > 1); pos--);
                if (val == node->val[pos]) {
                        printf("Duplicates not allowed\n");
                        return 0;
                }
        }
        if (setValueInNode(val, pval, node->link[pos], child)) {
                if (node->count < MAX) {
                        addValToNode(*pval, pos, node, *child);
                } else {
                        splitNode(*pval, pval, pos, node, *child, child);
                        return 1;
                }
        }
        return 0;
}
```

```
/* insert val in B-Tree */
void insertion(int val) {
      int flag, i;
      struct btreeNode *child;

      flag = setValueInNode(val, &i, root, &child);
      if (flag)
              root = createNode(i, child);
}

/* copy successor for the value to be deleted */
void copySuccessor(struct btreeNode *myNode, int pos) {
      struct btreeNode *dummy;
      dummy = myNode->link[pos];

      for (;dummy->link[0] != NULL;)
              dummy = dummy->link[0];
      myNode->val[pos] = dummy->val[1];

}

/* removes the value from the given node and rearrange values */
void removeVal(struct btreeNode *myNode, int pos) {
      int i = pos + 1;
      while (i <= myNode->count) {
              myNode->val[i - 1] = myNode->val[i];
              myNode->link[i - 1] = myNode->link[i];
              i++;
      }
      myNode->count--;
}

/* shifts value from parent to right child */
void doRightShift(struct btreeNode *myNode, int pos) {
      struct btreeNode *x = myNode->link[pos];
      int j = x->count;

      while (j > 0) {
              x->val[j + 1] = x->val[j];
              x->link[j + 1] = x->link[j];
      }
      x->val[1] = myNode->val[pos];
      x->link[1] = x->link[0];
      x->count++;

      x = myNode->link[pos - 1];
      myNode->val[pos] = x->val[x->count];
      myNode->link[pos] = x->link[x->count];
      x->count--;
      return;
```

```
}

/* shifts value from parent to left child */
void doLeftShift(struct btreeNode *myNode, int pos) {
      int j = 1;
      struct btreeNode *x = myNode->link[pos - 1];

      x->count++;
      x->val[x->count] = myNode->val[pos];
      x->link[x->count] = myNode->link[pos]->link[0];

      x = myNode->link[pos];
      myNode->val[pos] = x->val[1];
      x->link[0] = x->link[1];
      x->count--;

      while (j <= x->count) {
              x->val[j] = x->val[j + 1];
              x->link[j] = x->link[j + 1];
              j++;
      }
      return;
}

/* merge nodes */
void mergeNodes(struct btreeNode *myNode, int pos) {
      int j = 1;
      struct btreeNode *x1 = myNode->link[pos], *x2 = myNode->link[pos - 1];

      x2->count++;
      x2->val[x2->count] = myNode->val[pos];
      x2->link[x2->count] = myNode->link[0];

      while (j <= x1->count) {
              x2->count++;
              x2->val[x2->count] = x1->val[j];
              x2->link[x2->count] = x1->link[j];
              j++;
      }

      j = pos;
      while (j < myNode->count) {
              myNode->val[j] = myNode->val[j + 1];
              myNode->link[j] = myNode->link[j + 1];
              j++;
      }
      myNode->count--;
      free(x1);
}

/* adjusts the given node */
```

```
void adjustNode(struct btreeNode *myNode, int pos) {
	if (!pos) {
		if (myNode->link[1]->count > MIN) {
			doLeftShift(myNode, 1);
		} else {
			mergeNodes(myNode, 1);
		}
	} else {
		if (myNode->count != pos) {
			if(myNode->link[pos - 1]->count > MIN) {
				doRightShift(myNode, pos);
			} else {
				if (myNode->link[pos + 1]->count > MIN) {
					doLeftShift(myNode, pos + 1);
				} else {
					mergeNodes(myNode, pos);
				}
			}
		} else {
			if (myNode->link[pos - 1]->count > MIN)
				doRightShift(myNode, pos);
			else
				mergeNodes(myNode, pos);
		}
	}
}

/* delete val from the node */
int delValFromNode(int val, struct btreeNode *myNode) {
	int pos, flag = 0;
	if (myNode) {
		if (val < myNode->val[1]) {
			pos = 0;
			flag = 0;
		} else {
			for (pos = myNode->count;
				(val < myNode->val[pos] && pos > 1); pos--);
			 if (val == myNode->val[pos]) {
				flag = 1;
			} else {
				flag = 0;
			}
		}
		if (flag) {
			if (myNode->link[pos - 1]) {
				copySuccessor(myNode, pos);
				flag = delValFromNode(myNode->val[pos], myNode->link[pos]);
				if (flag == 0) {
					printf("Given data is not present in B-Tree\n");
				}
			} else {
```

```
                                removeVal(myNode, pos);
                        }
                } else {
                        flag = delValFromNode(val, myNode->link[pos]);
                }
                if (myNode->link[pos]) {
                        if (myNode->link[pos]->count < MIN)
                                adjustNode(myNode, pos);
                }
        }
        return flag;
}

/* delete val from B-tree */
void deletion(int val, struct btreeNode *myNode) {
        struct btreeNode *tmp;
        if (!delValFromNode(val, myNode)) {
                printf("Given value is not present in B-Tree\n");
                return;
        } else {
                if (myNode->count == 0) {
                        tmp = myNode;
                        myNode = myNode->link[0];
                        free(tmp);
                }
        }
        root = myNode;
        return;
}

/* search val in B-Tree */
void searching(int val, int *pos, struct btreeNode *myNode) {
        if (!myNode) {
                return;
        }

        if (val < myNode->val[1]) {
                *pos = 0;
        } else {
                for (*pos = myNode->count;
                        (val < myNode->val[*pos] && *pos > 1); (*pos)--);
                if (val == myNode->val[*pos]) {
                        printf("Given data %d is present in B-Tree", val);
                        return;
                }
        }
        searching(val, pos, myNode->link[*pos]);
        return;
}

/* B-Tree Traversal */
```

```
void traversal(struct btreeNode *myNode) {
	int i;
	if (myNode) {
		for (i = 0; i < myNode->count; i++) {
			traversal(myNode->link[i]);
			printf("%d ", myNode->val[i + 1]);
		}
		traversal(myNode->link[i]);
	}
}

int main() {
	int val, ch;
	while (1) {
		printf("1. Insertion\t2. Deletion\n");
		printf("3. Searching\t4. Traversal\n");
		printf("5. Exit\nEnter your choice:");
		scanf("%d", &ch);
		switch (ch) {
			case 1:
				printf("Enter your input:");
				scanf("%d", &val);
				insertion(val);
				break;
			case 2:
				printf("Enter the element to delete:");
				scanf("%d", &val);
				deletion(val, root);
				break;
			case 3:
				printf("Enter the element to search:");
				scanf("%d", &val);
				searching(val, &ch, root);
				break;
			case 4:
				traversal(root);
				break;
			case 5:
				exit(0);
			default:
				printf("U have entered wrong option!!\n");
				break;
		}
		printf("\n");
	}
}
```

```
Enter your choice:4
0 1 3 4
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
Enter your choice:PS C:\Users\User\Desktop\c_program\DS_LAB>
PS C:\Users\User\Desktop\c_program\DS_LAB> ./BBTree
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
Enter your choice:2
Enter the element to delete:2
Given value is not present in B-Tree

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
Enter your choice:3
Enter the element to search:1

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
Enter your choice:1
Enter your input:6

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
Enter your choice:1
```

```
Enter your input:0

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
Enter your choice:1
Enter your input:1

1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
Enter your choice:3
Enter the element to search:0
Given data 0 is present in B-Tree
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
Enter your choice:4
0 1 3 6
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
Enter your choice:3
Enter the element to search:3
Given data 3 is present in B-Tree
1. Insertion    2. Deletion
3. Searching    4. Traversal
5. Exit
Enter your choice:
```

gcc.exe ...
cppdbg: co...
BBTree

Ln 150, Col 1 Spaces: 4 UTF-8 CRLF C Win32