Instructor :

Rana Saeed

Write algorithm and C++ code for Deletion Operation in an Array?

Program of deletion:

```
// C++ Program to Delete an Element from an Array
// ----codescracker.com----

#include<iostream>
using namespace std;
int main()
{
   int arr[10], tot=10, i, elem, j, found=0;
   cout<<"Enter 10 Array Elements: ";
   for(i=0; i<tot; i++)
      cin>>arr[i];
   cout<<"\nEnter Element to Delete: ";
   cin>>elem;
   for(i=0; i<tot; i++)
   {
      if(arr[i]==elem)
      {
         for(j=i; j<(tot-1); j++)
            arr[j] = arr[j+1];
         found++;
         i--;
         tot--;
      }
   }
   if(found==0)
```

```
        cout<<"\nElement doesn't found in the Array!";
    else
        cout<<"\nElement Deleted Successfully!";
    cout<<endl;
    return 0;
}
```

Now Algorithm starts here :

And the entered element say **3** (that has to be delete) gets stored in **elem**. Now the dry run (inside the **for** loop) of above program goes like:

- Initial value, **found=0**, **tot=10**
- At first evaluation of *for loop*, 0 gets initialized to **i**. The initialization (of *for loop*) happens first and only once. Now **i=0**
- The condition **i<tot** or **0<10** evaluates to be true, therefore program flow goes inside the loop
- And using **if**, the condition **arr[i]==elem** or **arr[0]==3** or **1==3** evaluates to be false, therefore program flow does not goes inside **if**'s body, rather it goes to the updatation part of **for loop** and increments the value of **i**. Now **i=1**
- And again the condition **i<tot** or **1<10** evaluates to be true, therefore program flow again goes inside the loop, and evaluates the condition of **if** block
- And again the condition **arr[i]==elem** or **arr[1]==elem** or **2==3** evaluates to be false, therefore program flow again increments **i**. Now **i=2**
- Because the third time, condition of *for loop* evaluates to be true, therefore at third time also, program flow again goes inside the loop
- But at this time, the condition of *if*, evaluates to be true, because element at index number 2 is **3** that equals the value stored in **elem** (element that has to be delete).
- That is, the condition **arr[i]==elem** or **arr[2]==3** or **3==3** evaluates to be true

- Therefore program flow goes inside the **if**'s body, and executes all of its statements
- That is, using **for** loop, all the element from this (current or second) index gets shifted one index back.
    - That is at first, the value of **i** (2) gets initialized to **j**. So **j=2**
    - And the condition, **j<(tot-1)** or **2<(10-1)** or **2<9** evaluates to be true, therefore program flow goes inside the loop and **arr[j+1]** or **arr[2+1]** or **arr[3]** or **4** (value at index number 3) gets initialized to **arr[j]** or **arr[2]**
    - Now the value of **j** gets incremented. So **j=3**
    - Again the condition, **j<(tot-1)** or **3<(10-1)** or **3<9** evaluates to be true, therefore program flow again goes inside the loop and **arr[j+1]** or **arr[3+1]** or **arr[4]** or **5** gets initialized to **arr[j]** or **arr[3]**
    - This process continues, until the condition **j<(tot-1)** evaluates to be false
    - When the condition evaluates to be false, the evaluation of second *for* loop gets ended for now
    - And the value of **found** gets incremented. Its value is initialized with 0 at start of the program, to check whether it contain its value (0) or not, after evaluating the both *for* loops. If it contains its value as 0, therefore program flow never goes inside the **if** block. That means, no any element from the array gets matched to the element entered as element to be delete. And if its value will not be 0, then the element gets matched and deleted from that array

Write algorithm and C++ code for Search and Update Operations in Linked list?

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

struct node {
   int data;
   struct node *next;
};

struct node *head = NULL;
struct node *current = NULL;

//Create Linked List
void insert(int data) {
   // Allocate memory for new node;
   struct node *link = (struct node*) malloc(sizeof(struct node));

   link->data = data;
   link->next = NULL;

   // If head is empty, create new list
   if(head==NULL) {
      head = link;
      return;
   }

   current = head;

   // move to the end of the list
   while(current->next!=NULL)
```

```
      current = current->next;

   // Insert link at the end of the list
   current->next = link;
}

void display() {
   struct node *ptr = head;

   printf("\n[head] =>");
   //start from the beginning
   while(ptr != NULL) {
      printf(" %d =>",ptr->data);
      ptr = ptr->next;
   }

   printf(" [null]\n");
}

void update_data(int old, int new) {
   int pos = 0;

   if(head==NULL) {
      printf("Linked List not initialized");
      return;
   }

   current = head;
   while(current->next!=NULL) {
      if(current->data == old) {
         current->data = new;
         printf("\n%d found at position %d, replaced with %d\n", old,
pos, new);
         return;
      }

      current = current->next;
      pos++;
   }

   printf("%d does not exist in the list\n", old);
}
```

```
int main() {
   insert(10);
   insert(20);
   insert(30);
   insert(1);
   insert(40);
   insert(56);

   display();
   update_data(40, 44);
   display();

   return 0;
}
```

Output

Output of the program should be −

```
[head] => 10 => 20 => 30 => 1 => 40 => 56 => [null]

40 found at position 4, replaced with 44

[head] => 10 => 20 => 30 => 1 => 44 => 56 => [null]
```

2nd Assignments

Write C/C++ code for implementation of stack?

Definition

A stack is a basic computer science data structure and can be defined in an abstract, implementation-free manner, or it can be generally defined as a linear list of items in which all additions and deletion are restricted to one end that is Top.

Stack Implementation using Array in C++

```
#include<iostream>
#define MAX 5

using namespace std;


int STACK[MAX],TOP;

//stack initialization
void initStack(){
    TOP=-1;
}
//check it is empty or not
int isEmpty(){
    if(TOP==-1)
        return 1;
    else
        return 0;
}

//check stack is full or not
int isFull(){
    if(TOP==MAX-1)
        return 1;
    else
```

```
        return 0;
}

void push(int num){
    if(isFull()){
        cout<<"STACK is FULL."<<endl;
        return;
    }
    ++TOP;
    STACK[TOP]=num;
    cout<<num<<" has been inserted."<<endl;
}

void display(){
    int i;
    if(isEmpty()){
        cout<<"STACK is EMPTY."<<endl;
        return;
    }
    for(i=TOP;i>=0;i--){
        cout<<STACK[i]<<" ";
    }
    cout<<endl;
}

//pop - to remove item
void pop(){
    int temp;
    if(isEmpty()){
        cout<<"STACK is EMPTY."<<endl;
        return;
    }

    temp=STACK[TOP];
    TOP--;
    cout<<temp<<" has been deleted."<<endl;
}
int main(){
    int num;
    initStack();
    char ch;
    do{
            int a;
```

```
            cout<<"Chosse \n1.push\n"<<"2.pop\n"<<"3.display\n";
            cout<<"Please enter your choice: ";
            cin>>a;
            switch(a)
            {
                case 1:
                    cout<<"Enter an Integer Number: ";
                    cin>>num;
                    push(num);
                break;

                case 2:
                    pop();
                    break;

                case 3:
                    display();
                    break;

                default :
                cout<<"An Invalid Choice!!!\n";


            }
            cout<<"Do you want to continue ? ";
            cin>>ch;
            }while(ch=='Y'||ch=='y');
    return 0;
}
```

Write algorithm/C++ code of Insert operation for binary search trees. Also explain Hash Tableand Hashing Technique?

Binary Search Tree vs Hash Table

Comparison: BST vs HashTable

Criteria	Binary Search Tree	Hash Table
Time complexity of Insertion/ deletion/ searching (Avg.)	O(log n)	O(1)
Sorted Order	Data is stored in Sorted order i.e. the in-order traversal will give the sorted data	Data is stored in random fashion. If you need sorted order, then you need some extra variable
Hash Function	There is no need of Hash Function in BST	In order to perform operations in O(1) time, we need to make proper hash function to generate a key.
Collision	There is no collision of data in BST	If some collision occurs, then we can use collision removal techniques like chaining and open addressing in Hash Table.
Input data size	There is no need of finding the input data size in advance	You have to find the input data size before making your Hash Table.
Range search	Range search is very fast in case of BST	Range search is very slow because each and every possible case is searched.

Program of Insert

```
#include <iostream>
using namespace std;

class BST
{
    int data;
    BST *left, *right;

public:
    // Default constructor.
    BST();

    // Parameterized constructor.
    BST(int);

    // Insert function.
    BST* Insert(BST*, int);

    // Inorder traversal.
    void Inorder(BST*);
};

// Default Constructor definition.
BST ::BST()
    : data(0)
    , left(NULL)
    , right(NULL)
{
}

// Parameterized Constructor definition.
BST ::BST(int value)
{
```

```
    data = value;
    left = right = NULL;
}

// Insert function definition.
BST* BST ::Insert(BST* root, int value)
{
    if (!root)
    {
        // Insert the first node, if root is NULL.
        return new BST(value);
    }

    // Insert data.
    if (value > root->data)
    {
        // Insert right node data, if the 'value'
        // to be inserted is greater than 'root' node
data.

        // Process right nodes.
        root->right = Insert(root->right, value);
    }
    else
    {
        // Insert left node data, if the 'value'
        // to be inserted is greater than 'root' node
data.

        // Process left nodes.
        root->left = Insert(root->left, value);
    }

    // Return 'root' node, after insertion.
    return root;
```

```
}

// Inorder traversal function.
// This gives data in sorted order.
void BST ::Inorder(BST* root)
{
    if (!root) {
        return;
    }
    Inorder(root->left);
    cout << root->data << endl;
    Inorder(root->right);
}

// Driver code
int main()
{
    BST b, *root = NULL;
    root = b.Insert(root, 50);
    b.Insert(root, 30);
    b.Insert(root, 20);
    b.Insert(root, 40);
    b.Insert(root, 70);
    b.Insert(root, 60);
    b.Insert(root, 80);

    b.Inorder(root);
    return 0;
}
```