```cpp
//1. Write a recursive and non recursive program to calculate the factorial of a number.
#include<iostream>
using namespace std;

long int multiplyNumbers(int n);

int main()
{
    int n;
    cout<<"Enter a positive integer";
    cin>>n;
    cout<<"Factorial of the number is"<<multiplyNumbers(n)<<endl;

    return 0;
}

long int multiplyNumbers(int n)
{
    if(n>=1)
        return n*multiplyNumbers(n-1);

    else
        return 1;
}

```

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

int main()
{

    int count;
    int sum;

    int arr[10]{1,2,3,4,5,6,7,8,9,10};

    for(count=0; count<=9; count++)
    {
        sum= sum + arr[count];

    }

    cout<<"Sum of Array elements is= "<<sum;

    return 0;
}
```

```cpp
//Write a program to perform linear search for a given value in unsorted list.
#include<iostream>
using namespace std;

int search(int arr[],int n,int x)
{
    for(int i=0; i<n; i++)
        if (arr[i]==x)
           return i;

    return -1;
}
int main()
{
int arr[]={2,4,0,1,9};
int x=1;
int n= sizeof(arr)/ sizeof(arr[0]);

int result = search(arr, n, x);

(result == -1)   ?  cout<<"Element not found"<<endl  : cout<<"Element found at index ="<<result<<endl;

return 0;
}
```

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

int binarySearch(int array[], int x, int low, int high)
{
if (high >= low)
{
int mid = low + (high-low) /2;


if (array [mid] == x)
    return mid;


if (array[mid]> x)
    return binarySearch(array, x, low, mid -1);



 return binarySearch (array, x, mid + 1, high );
 }
 return -1;
 }

 int main(void)
 {
int array[] = {3, 4, 5, 6, 7, 8, 9};
int n= sizeof(array)/ sizeof (array [0]);
int x = 4;
int result = binarySearch(array, x, 0, n - 1);

 if (result == -1)
cout<<"Not found";

else

cout<<"Element 1s found at index :"<<result<<endl;
 }
```

```cpp
// 7). Write a program that implements Selection sort, to sort a given list of integers in ascending order.

#include <iostream>
using namespace std;

// function to swap the the position of two elements
void swap(int *a, int *b) {
  int temp = *a;
  *a = *b;
  *b = temp;
}

// function to print an array
void printArray(int array[], int size) {
  for (int i = 0; i < size; i++) {
    cout << array[i] << " ";
  }
  cout << endl;
}

void selectionSort(int array[], int size) {
  for (int step = 0; step < size - 1; step++) {
    int min_idx = step;
    for (int i = step + 1; i < size; i++) {

      // To sort in descending order, change > to < in this line.
      // Select the minimum element in each loop.
      if (array[i] < array[min_idx])
        min_idx = i;
    }

    // put min at the correct position
    swap(&array[min_idx], &array[step]);
  }
}

// driver code
int main() {
  int data[] = {20, 12, 10, 15, 2};
  int size = sizeof(data) / sizeof(data[0]);
  selectionSort(data, size);
  cout << "Sorted array in Acsending Order:\n";
  printArray(data, size);
}
```

```cpp
// 8) Write a program that implement insertion sort a given list of inters in ascending order.

#include <iostream>
using namespace std;

// Function to print an array
void printArray(int array[], int size) {
  for (int i = 0; i < size; i++) {
    cout << array[i] << " ";
  }
  cout << endl;
}

void insertionSort(int array[], int size) {
  for (int step = 1; step < size; step++) {
    int key = array[step];
    int j = step - 1;

    // Compare key with each element on the left of it until an element smaller than
    // it is found.
    // For descending order, change key<array[j] to key>array[j].
    while (key < array[j] && j >= 0) {
      array[j + 1] = array[j];
      --j;
    }
    array[j + 1] = key;
  }
}

// Driver code
int main() {
  int data[] = {9, 5, 1, 4, 3};
  int size = sizeof(data) / sizeof(data[0]);
  insertionSort(data, size);
  cout << "Sorted array in ascending order:\n";
  printArray(data, size);
}
```

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

// A linked list node
struct Node
{
   int data;
   struct Node *next;
};
//insert a new node in front of the list
void push(struct Node** head, int node_data)
{
   /* 1. create and allocate node */
   struct Node* newNode = new Node;

   /* 2. assign data to node */
   newNode->data = node_data;

   /* 3. set next of new node as head */
   newNode->next = (*head);

   /* 4. move the head to point to the new node */
   (*head) = newNode;
}

//insert new node after a given node
void insertAfter(struct Node* prev_node, int node_data)
{
  /*1. check if the given prev_node is NULL */
if (prev_node == NULL)
{
   cout<<"the given previous node is required,cannot be NULL"; return; }

   /* 2. create and allocate new node */
   struct Node* newNode =new Node;

   /* 3. assign data to the node */
   newNode->data = node_data;

   /* 4. Make next of new node as next of prev_node */
   newNode->next = prev_node->next;

    /* 5. move the next of prev_node as new_node */
    prev_node->next = newNode;
}

/* insert new node at the end of the linked list */
void append(struct Node** head, int node_data)
{
/* 1. create and allocate node */
struct Node* newNode = new Node;

struct Node *last = *head; /* used in step 5*/

/* 2. assign data to the node */
newNode->data = node_data;

/* 3. set next pointer of new node to null as its the last node*/
newNode->next = NULL;

/* 4. if list is empty, new node becomes first node */
if (*head == NULL)
{
*head = newNode;
return;
}
```

```
67
68  /* 5. Else traverse till the last node */
69  while (last->next != NULL)
70  last = last->next;
71
72  /* 6. Change the next of last node */
73  last->next = newNode;
74  return;
75  }
76
77  // display linked list contents
78  void displayList(struct Node *node)
79  {
80     //traverse the list to display each node
81     while (node != NULL)
82     {
83        cout<<node->data<<"-->";
84        node = node->next;
85     }
86
87  if(node== NULL)
88  cout<<"null";
89  }
90  /* main program for linked list*/
91  int main()
92  {
93  /* empty list */
94  struct Node* head = NULL;
95
96  // Insert 10.
97  append(&head, 10);
98
99  // Insert 20 at the beginning.
100 push(&head, 20);
101
102 // Insert 30 at the beginning.
103 push(&head, 30);
104
105 // Insert 40 at the end.
106 append(&head, 40); //
107
108
109 insertAfter(head->next, 50);
110
111 cout<<"Final linked list: "<<endl;
112 displayList(head);
113
114 return 0;
115 }
```

```cpp
//10) Write a program that uses functions to perform deletion operation on singly linked list.
#include <bits/stdc++.h>
using namespace std;

/* Link list node */
class Node {
public:
    int data;
    Node* next;
};

/* Function to delete the entire linked list */
void deleteList(Node** head_ref)
{

    /* deref head_ref to get the real head */
    Node* current = *head_ref;
    Node* next = NULL;

    while (current != NULL)
    {
        next = current->next;
        free(current);
        current = next;
    }

    /* deref head_ref to affect the real head back
        in the caller. */
    *head_ref = NULL;
}

/* Given a reference (pointer to pointer) to the head
of a list and an int, push a new node on the front
of the list. */
void push(Node** head_ref, int new_data)
{
    /* allocate node */
    Node* new_node = new Node();

    /* put in the data */
    new_node->data = new_data;

    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref) = new_node;
}

/* Driver code*/
int main()
{
    /* Start with the empty list */
    Node* head = NULL;

    /* Use push() to construct below list
    1->12->1->4->1 */
    push(&head, 1);
    push(&head, 4);
    push(&head, 1);
    push(&head, 12);
    push(&head, 1);

    cout << "Deleting linked list";
    deleteList(&head);

```

```
    cout << "\nLinked list deleted";
}

// This is code is contributed by rathbhupendra
```

```cpp
// 11) Write  a program that implement stack and its operations(push and pop) using array.
#include <iostream>
using namespace std;

int stack[100], n=100, top=-1;

    void push (int val)
{
    if ( top>=n-1 )
        {
        cout<<"Stack Overflow "<<endl;
        }

       else
       {
         top++;
         stack[top]=val;
       }
}

    void pop()
{
     if (top<=-1)
        {
        cout<<"Stack Underflow"<<endl;
        }

        else
        {
         cout<<"The popped element is "<< stack[top] <<endl;
         top--;
        }

}
void display()
{
        if (top>=0)
          {
             cout<<"Stack elements are :";

                 for(int i=top; i>=0; i--)
             cout<<stack[i]<<" ";
             cout<<endl;
          }

          else
          {
          cout<<"Stack is empty";
          }
}

int main()
{
    int ch, val;
    cout<<"1) Push in stack"<<endl;
    cout<<"2) Pop from stack"<<endl;
    cout<<"3) Display stack"<<endl;
    cout<<"4) Exit"<<endl;

    do{
        cout<<"Enter choice :"<<endl;
        cin>>ch;
           switch(ch)
           {
             case 1:
             {
```

```
                cout<<"Enter value to be pushed :"<<endl;
                cin>>val;
                push(val);
                break;
               }
               case 2:
               {
                pop( );
                break;
               }
               case 3:
               {
                display();
                break;
               }
               case 4:
               {
                cout<<"Exit"<<endl;
                break;
               }
               default:
               {
                 cout<<"Invalid choice"<<endl;

               }
              }
         }
         while(ch!=4);
         return 0;

    }
```

```cpp
//12) Write a program that implements stack and its operation( push and pop) using linked list.
#include <iostream>
using namespace std;
struct Node {
   int data;
   struct Node *next;
};
struct Node* top = NULL;
void push(int val) {
   struct Node* newnode = (struct Node*) malloc(sizeof(struct Node));
   newnode->data = val;
   newnode->next = top;
   top = newnode;
}
void pop() {
   if(top==NULL)
   cout<<"Stack Underflow"<<endl;
   else {
      cout<<"The popped element is "<< top->data <<endl;
      top = top->next;
   }
}
void display() {
   struct Node* ptr;
   if(top==NULL)
   cout<<"stack is empty";
   else {
      ptr = top;
      cout<<"Stack elements are: ";
      while (ptr != NULL) {
         cout<< ptr->data <<" ";
         ptr = ptr->next;
      }
   }
   cout<<endl;
}
int main() {
   int ch, val;
   cout<<"1) Push in stack"<<endl;
   cout<<"2) Pop from stack"<<endl;
   cout<<"3) Display stack"<<endl;
   cout<<"4) Exit"<<endl;
   do {
      cout<<"Enter choice: "<<endl;
      cin>>ch;
      switch(ch) {
         case 1: {
            cout<<"Enter value to be pushed:"<<endl;
            cin>>val;
            push(val);
            break;
         }
         case 2: {
            pop();
            break;
         }
         case 3: {
            display();
            break;
         }
         case 4: {
            cout<<"Exit"<<endl;
            break;
         }
         default: {
            cout<<"Invalid Choice"<<endl;
```

```
67            }
68         }
69     }while(ch!=4);
70     return 0;
71 }
```