DYNAMIC ARRAY CODE (using classes)

……..

```
#include <iostream>
using namespace std;
class Array
{
   public:
   int *a= new int[size];
   int size;
   int top=0;
   void insert(int value)
   {
          if(top >= size)
          cout<<"size exceeded\n";
          else
          {
                 a[top]=value;
                 top++;
          }
   }
   void remove()
   {
          if (top==-1)
          cout<<"empty array\n";
          else
          top--;
   }
   void update(int number, int location)
   {
          a[location-1]= number;
   }
   void traverse()
   {
          cout<<"array is:\t";
          for (int i=0;i<top;i++)
          {
                 cout<<a[i]<<"\t";
          }
   }
};
void insert(int value);
void remove();
```

```
void update(int number, int location);
int main()
{
   int location;
   int value;
   int number;
   Array arr1;
   int option;
   cout<<"Enter the size of the array:\t";
   cin>>arr1.size;
   while(1)
   {
         cout<<"1.insert"<<endl; cout<< "2.update" <<endl; cout<<"3.delete"<<endl;
cout<<"4.exit"<<endl;
         cin>>option;
         if(option==1)
         {
               cout<<"enter the value to inserted:\t";
               cin>>value;
               arr1.insert(value);
         }
         if(option==3)
         arr1.remove();
         if(option==2)
         {
               cout<<"enter the location"<<endl; cout<<"enter the number"<<endl;
               cin>>location;
               cin>>number;
               arr1.update(number,location);
         }
        if(option==4)
         break;
   }
   arr1.traverse();

   return 0;
}
```

……

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

```
class linear_search
{
   int x;
   int *a;
   int size;
   public:
   int search()
   {
         for(int i=0; a[i]!= x;i++)
         {
                  {
                  }
                  return i;
         }
   }

   linear_search()
   {
         a= new int[size];
   }

   void setsize(int value)
   {
         value=size;
   }
   void setkey(int key)
   {
         key=x;
   }
};

int main()
{
   int value;
   int key;
   int element;
   linear_search array;
   int *a= new int[value];
   cout<<"enter the size of array";
   cin>>value;
   for (int i=0;i<value;i++)
   {
```

```
            cout<<"enter the element";
            cin>>element;
            a[i]=element;

    }
    array.setsize(value);
    cout<< "enter the number to be searched";
    cin>>key;
    array.setkey(key);
    array.search();

    return 0;
}
```

Linked List using classes

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

// Node class
class Node {
    int data;
    Node* next;

  public:
    Node() {};
    void SetData(int aData) { data = aData; };
    void SetNext(Node* aNext) { next = aNext; };
    int Data() { return data; };
    Node* Next() { return next; };
};

// List class
class List {
    Node *head;
  public:
    List() { head = NULL; };
    void Print();
    void Append(int data);
```

```
    void Delete(int data);
};

/**
 * Print the contents of the list
 */
void List::Print() {

    // Temp pointer
    Node *tmp = head;

    // No nodes
    if ( tmp == NULL ) {
    cout << "EMPTY" << endl;
    return;
    }

    // One node in the list
    if ( tmp->Next() == NULL ) {
    cout << tmp->Data();
    cout << " --> ";
    cout << "NULL" << endl;
    }
    else {
    // Parse and print the list
    do {
        cout << tmp->Data();
        cout << " --> ";
        tmp = tmp->Next();
    }
    while ( tmp != NULL );

    cout << "NULL" << endl;
    }
}

/**
```

```
 * Append a node to the linked list
 */
void List::Append(int data) {

    // Create a new node
    Node* newNode = new Node();
    newNode->SetData(data);
    newNode->SetNext(NULL);

    // Create a temp pointer
    Node *tmp = head;

    if ( tmp != NULL ) {
    // Nodes already present in the list
    // Parse to end of list
    while ( tmp->Next() != NULL ) {
        tmp = tmp->Next();
    }

    // Point the last node to the new node
    tmp->SetNext(newNode);
    }
    else {
    // First node in the list
    head = newNode;
    }
}

/**
 * Delete a node from the list
 */
void List::Delete(int data) {

    // Create a temp pointer
    Node *tmp = head;

    // No nodes
```

```
    if ( tmp == NULL )
    return;

    // Last node of the list
    if ( tmp->Next() == NULL ) {
    delete tmp;
    head = NULL;
    }
    else {
    // Parse thru the nodes
    Node *prev;
    do {
        if ( tmp->Data() == data ) break;
        prev = tmp;
        tmp = tmp->Next();
    } while ( tmp != NULL );

    // Adjust the pointers
    prev->SetNext(tmp->Next());

    // Delete the current node
    delete tmp;
    }
}

int main()
{
    // New list
    List list;

    // Append nodes to the list
    list.Append(100);
    list.Print();
    list.Append(200);
    list.Print();
    list.Append(300);
    list.Print();
```

```
    list.Append(400);
    list.Print();
    list.Append(500);
    list.Print();

    // Delete nodes from the list
    list.Delete(400);
    list.Print();
    list.Delete(300);
    list.Print();
    list.Delete(200);
    list.Print();
    list.Delete(500);
    list.Print();
    list.Delete(100);
    list.Print();
}
```