```
/*WAP to implement contiguous list using array*/
#include<iostream>
#include<stdlib.h>
#define max 15
using namespace std;
struct nodetype
{
   int info,next;
};
class list
{
   struct nodetype node[max];
   int avail=0;
public:
   int intialize_availlist()
   {
      int i;
      for(i=0; i<max-1; i++)
      {
         node[i].next=i+1;
      }
      node[max-1].next=-1;
   }
   int get_node()
   {
      int p;
      if(avail==-1)
      {
         cout<<"Overflow";
         exit(1);
      }
      p=avail;
      avail=node[avail].next;
      return p;
   }
   int freenode(int p)
   {
```

```
   node[p].next=avail;
   avail=p;
}
int insertnode(int &list1)
{
   int val,ptr,curptr,newnode=1;
   while(newnode==1)
   {
      if(list1==-1)
      {
         ptr=get_node();
         list1=ptr;
         cout<<"Enter the number: ";
         cin>>val;
         node[ptr].info=val;
         node[ptr].next=-1;
      }
      else
      {
         curptr=0;
         while(node[curptr].next!=-1)
         {
            curptr=node[curptr].next;
         }
         ptr=get_node();
         cout<<"Enter a Number: ";
         cin>>val;
         node[curptr].next=ptr;
         node[ptr].info=val;
         node[ptr].next=-1;
      }
      cout<<"Enter 1 for Newnode(-1 to stop): ";
      cin>>newnode;
   }
}
int displaynode()
{
```

```
    cout<<"**********Displaying The list**********"<<endl;
    int i;
    int ptr=0;
    if(avail==0)
    {
      cout<<"List Underflow"<<endl;
    }
    while(ptr!=-1)
    {
      cout<<"Index: "<<ptr<<" Value: "<<node[ptr].info<<" Next:
"<<node[ptr].next<<endl;
      ptr=node[ptr].next;
    }
  }
  int lastdeletenode(int &list1)
  {
    int ptr,curptr;
    if(list1==-1)
    {
      cout<<"List underflow";
    }
    else
    {
      curptr=0;
      ptr=curptr;
      while(node[curptr].next!=-1)
      {
        ptr=curptr;
        curptr=node[curptr].next;
      }
      freenode(curptr);
      node[ptr].next=-1;
    }
  }
  int insert_after(int ptr,int val)
  {
    int newptr;
```

```
      if(ptr==-1)
      {
         cout<<"Invalid Insertion";
      }
      else
      {
         newptr=get_node();
         node[newptr].info=val;
         node[newptr].next=node[ptr].next;
         node[ptr].next=newptr;
         cout<<"Inserted Node After "<<ptr<<" Value: "<<val<<" Index:
"<<newptr<<endl;
      }
   }
   int delete_after(int ptr)
   {
      int delptr,delval;
      if(ptr==-1 || node[ptr].next==-1)
      {
         cout<<"Invalid deletion after given ptr"<<endl;
      }
      else
      {
         delptr=node[ptr].next;
         delval=node[delptr].info;
         cout<<"Deleted Value is "<<delval<<endl;
         node[ptr].next=node[delptr].next;
         freenode(delptr);
      }
   }
};
int main()
{
   list l;
   l.intialize_availlist();
   int ch;
   int list1=-1;
```

```
   do
   {
      cout<<"1. Insert a new node: "<<endl;
      cout<<"2. Display Nodes: "<<endl;
      cout<<"3. Delete Last Node"<<endl;
      cout<<"4. Insert After Node"<<endl;
      cout<<"5. Delete After Node"<<endl;
      cout<<"6.Exit"<<endl;
      cin>>ch;
      switch(ch)
      {
      case 1:
         l.insertnode(list1);
         break;
      case 2:
         l.displaynode();
         break;
      case 3:
         l.lastdeletenode(list1);
         break;
      case 4:
         int ptr,val;
         cout<<"Enter the node index after which the new node has to be
inserted: ";
         cin>>ptr;
         cout<<"Enter the value to be inserted in the new node: ";
         cin>>val;
         l.insert_after(ptr,val);
         break;
      case 5:
         cout<<"Enter the node index after which the node has to be deleted: ";
         cin>>ptr;
         l.delete_after(ptr);
         break;
      case 6:
         exit(0);
         break;
```

```
      }
   }
   while(ch!=5);
   char c;
   cin>>c;
   return 0;
}

/*WAP to implement contiguous list using array*/
#include<iostream>
#define max 4
using namespace std;
class List
{
   int avail;
   struct nodeType
   {
      int info;
      int next;
   };
   struct nodeType node [max];
public:
   List()
   {
      avail=0;
      for(int i=0; i<max; i++)
      {
         node[i].next=i+1;
         node[i].info=0;
      }
      node[max-1].next=-1;
   }
   int getnode()
   {
      if (avail==-1)
      {
         cout<<"##############Overflow##############"<<endl;
```

```
    }
    else
    {
        int ptr;
        ptr=avail;
        avail=node[ptr].next;
        return ptr;
    }
}
void freenode(int ptr)
{
    node[ptr].next=avail;
    avail=ptr;
}
void ins()
{
    int num,ptr;
    cout<<"Enter the number:\t";
    cin>>num;
    cout<<"\n";
    ptr=getnode();
    if (ptr==0)
    {
        node[ptr].info=num;
        node[ptr].next=-1;
    }
    else
    {
        node[ptr].info=num;
        node[ptr].next=-1;
        bool test=true;
        int temp=0;
        while(test)
        {
            if (node[temp].next==-1)
            {
                test=false;
```

```
            node[temp].next=ptr;
          }
          temp=node[temp].next;
       }
    }
}
void insafter()
{
    int num,ptr,pos;
    cout<<"\nEnter n-1 position:\t";
    cin>>pos;
    cout<<"Enter the number:\t";
    cin>>num;
    cout<<"\n";
    ptr=getnode();
    if ((ptr==-1)||(ptr>max-1))
    {
       cout<<"##########Invalid#############"<<endl;
    }
    else
    {
       node[ptr].info=num;
       node[ptr].next=node[pos].next;
       node[pos].next=ptr;
    }
}
void del()
{
    int pos;
    cout<<"Enter n position:\t";
    cin>>pos;
    cout<<"\n";
    node[pos].info=0;
    bool test=true;
    int temp=0;
    if(pos==-1  || pos>max-1)
    {
```

```
      cout<<"#####Cant  REMOVE#####"<<endl;
   }
   else
   {
      while (test)
      {
         if (node[temp].next==pos)
         {
            test=false;
            node[temp].next=node[pos].next;
         }
         temp=node[temp].next;
      }
   }
   freenode(pos);
}
void delafter()
{
   int pos;
   cout<<"Enter n-1 position:\t";
   cin>>pos;
   cout<<"\n";
   if (pos==-1 || pos>max-1)
   {
      cout<<"####### Can't remove##########";
   }
   else
   {
      int delptr=node[pos].next;
      node[delptr].info=0;
      node[pos].next=node[delptr].next;
      freenode(delptr);
   }
}
void display()
{
   cout<<"-------------------------------------"<<endl;
```

```
      cout<<"-------------------------------------"<<endl;
      cout<<"\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\tavail= "<<avail<<endl;
      cout<<"Node\t\t\t\t\tInfo\t\t\t\t\tNext"<<endl;
      for (int i=0; i<max; i++)
      {
         cout<<i<<"\t\t\t\t\t"<<node[i].info<<"\t\t\t\t\t"<<node[i].next<<endl;
      }
      cout<<"-------------------------------------"<<endl;
      cout<<"-------------------------------------"<<endl;
   }
};
int main()
{
   int option;
   List lobj;
   do
   {
      cout<<"Choose:\n1.Insert\n2.Delete\n3.Insert After\n4.Delete
After\n5.Exit"<<endl;
      cin>>option;
      switch (option)
      {
      case 1:
      {
         lobj.ins();
         break;
      }
      case 2:
      {
         lobj.del();
         break;
      }
      case 3:
      {
         lobj.insafter();
         break;
      }
```

```
      case 4:
      {
         lobj.delafter();
         break;
      }
      default:
         cout<<"*****************Exiting**************"<<endl;
      }
      lobj.display();
   }
   while(option!=5);
   return 0;
}

/*WAP to implement contiguous list using array*/
#include<iostream>
#include<cstdlib>
#define MAX 10
using namespace std;
int avail =0;
struct nodetype
{
   int info,next;
};
class List
{
   nodetype node[MAX];
   int root;
   int getnode()
   {
      int p;
      if(avail==-1)
      {
         cout<<"\nOverflow\n";
         return -1;
      }
      p=avail;
```

```
    avail=node[avail].next;
    return p;
  }
  void freenode(int p)
  {
    node[p].info=-11;  /** -11 denotes empty*/
    node[p].next=avail;
    avail=p;
  }
  bool checklist()
  {
    if(root!=-1)
    {
      char ch;
      cout<<"\n\aThere is an existing node !!!   Do you want to replace it? y/n
";
      cin>>ch;
      if(ch=='y'||ch=='Y')
      {
        Initializearray();
        return true;
      }
      else
        return false;
    }
    else
      return true;
  }
  void Initializearray()
  {
    for(int i=0; i<=MAX-1; i++)
    {
      node[i].info=-11;   /** -11 denotes empty*/
      node[i].next=i+1;
    }
    node[MAX-1].next=-1;
  }
```

```
public:
   List():root(-1)
   {
      Initializearray();
   }
   void createlist()
   {
      if(checklist())
      {
         root = getnode();
         int num,ptr,point=root;
         cout<<"\nEnter the value of node : ";
         cin>>node[root].info;
         node[root].next=-1;
         while(1)
         {
            cout<<"\nEnter -1 to end input\nEnter the value : ";
            cin>>num;
            if(num==-1)
               break;
            ptr=getnode();
            node[point].next=ptr;
            point=ptr;
            node[ptr].info=num;
            node[ptr].next=-1;
         }
      }
   }
   void inslast(int num)
   {
      if(root==-1)
         cout<<"\nThere is no existing list\n";
      else
      {
         int ptr=root;
         while(node[ptr].next!=-1)
            ptr=node[ptr].next;
```

```
      node[ptr].next=getnode();
      ptr=node[ptr].next;
      node[ptr].info=num;
      node[ptr].next=-1;
   }
}
void insafter ( int ptr, int val)
{
   int newptr;
   if(ptr == MAX-1)
   {
      cout<<"\nInvalid Insertion\n";
   }
   else
   {
      newptr = getnode();
      if(newptr==-1)
      {
         cout<<"\nThere is no new available node\n";
      }
      else
      {
         node[newptr].info = val;
         int point=root;
         for(int i=1; i<ptr; i++)
         {
            point=node[point].next;
         }
         node[newptr].next = node[point].next;
         node[point].next = newptr;
      }
   }
}
void dellast()
{
   if(root==-1)
      cout<<"\nThere is no existing list\n";
```

```
    else
    {
       if(node[root].next==-1)
       {
          cout<<"\nThe deleted value is : "<<node[root].info<<"\n\n";
          freenode(root);
          root=-1;
       }
       else
       {
          int point,ptr;
          point=ptr=root;
          while(node[ptr].next!=-1)
          {
             point=ptr;
             ptr=node[ptr].next;
          }
          node[point].next=-1;
          cout<<"\nThe deleted value is : "<<node[ptr].info<<endl;
          freenode(ptr);
       }
    }
}
void delafter(int ptr)
{
    int delptr,delval;
    if((ptr== MAX-1)||node[ptr].next== -1)
       cout<<"\nInvalid deletion after the given pointer\n";
    else
    {
       int point=root;
       for(int i=1; i<ptr; i++)
       {
          point=node[point].next;
       }

       delptr=node[point].next;
```

```
            delval=node[delptr].info;
            cout<<"\nThe deleted value is : "<<delval<<endl;
            node[point].next=node[delptr].next;
            freenode(delptr);
        }
    }
    void displaylist()
    {
        if(root==-1)
            cout<<"\n\nThere is no existing list\n\n";
        else
        {
            int ptr=root;
            cout<<"\n\nThe list is : \n";
            while(node[ptr].next!=-1)
            {
                cout<<node[ptr].info<<"\t";
                ptr=node[ptr].next;
            }
            cout<<node[ptr].info<<"\n\n";
        }
    }
    void displayarr()
    {
        cout<<"\n\nIndex\tValue\tNext\n";
        for(int i=0; i<MAX; i++)
        {
            cout<<i<<"\t"<<node[i].info<<"\t"<<node[i].next<<endl;
        }
        cout<<endl;
    }
};
int main()
{
    List l;
    int choice,num,ptr;
    while(1)
```

```
  {
    cout<<"1. Create a list\n2. Insert node at last\n3. Insert node after certain node\n4. Delete last node\n5. Delete node after certain node\n6. Display list\n7. Display array\n8. Exit\nEnter your choice: ";
    cin>>choice;
    switch(choice)
    {
    case 1:
      l.createlist();
      break;
    case 2:
    {
      cout<<"\nEnter the value : ";
      cin>>num;
      l.inslast(num);
      break;
    }
    case 3:
    {
      cout<<"\nEnter the node : ";
      cin>>ptr;
      cout<<"\nEnter the value : ";
      cin>>num;
      l.insafter(ptr,num);
      break;
    }
    case 4:
      l.dellast();
      break;
    case 5:
    {
      cout<<"\nEnter the node : ";
      cin>>ptr;
      l.delafter(ptr);
      break;
    }
    case 6:
```

```
            l.displaylist();
            break;
        case 7:
            l.displayarr();
            break;
        default :
            exit(0);
        }
    }
}
```