```
1)def intersection(l1, l2):
    rev=[]
    for student in l1:
        if student in l2:
            rev.append(student)
    return rev

def union(l1, l2):
    rev=l2.copy()

    for student in l1:
        if not student in l2:
            rev.append(student)

    return rev

def diff(l1, l2):
    rev= []

    for student in l1:
        if not student in l2:
            rev.append(student)

    return rev
rev=[1,2,3,4,5,6,7] #jersy no of cricket players
rev=[2,3,6,7,10] #jersy no of badminton players
rev=[0,4,6,8,10,12] #jersy no of football players

print(intersection(a,b)) play cricket and badminton
print(diff(union(a,b),intersection(a,b))) #play either cricket or Badminton
print(diff(c,union(a,b))) #play neither cricket nor Badminton
print(diff(union(a,c),b)) #play cricket and football but not Badminton
```

```
2)marklist = ["20","30","80","20","none"]

# n=int(input("Enter the number of students"))

# for i in range(n):
#    mark=int(input("Enter mark of student: "))
#    marklist.append(mark)

#calculate avg
total=0
for mark in marklist:
    total += mark
#calculate max and min
max_val=marklist[0]
min_val=marklist[0]

absent_student =0
for mark in marklist:
    if mark==None:
        absent_student +=1
    else:
        total+=mark
        if max_val < mark:
            max_val = mark
        if min_val > mark:
            min_val = mark
print("a) Average of students marks: ",total/len(marklist))
print("b) maximum marks:",max_val,"minimum marks:",min_val)
print("c) absent student:",absent_student)
```

```
3)class bankUser:
    def __init__(self):
        self.balance=0
        print("Welcome to bank deposit ed withdraw")

    def deposit(self):
        print("How much Enter amt to be deposited")
        if amt>0:
            self.balance+=amt
            print("Amt deposited",amt)
        else:
            print("Invalid amt")

    def withdraw(self):
        print("How much Enter amt to be withdrawing")
        if amt>0:
            if amt<=self.balance:
                self.balance-=amt
                print("Amt after withdrawing ",amt)
            else:
                print("Insufficient balance")
        else:
            print("Invalid amt")

    def display(self):
        print("net bal ",self.balance)

u=bankUser()
while True:
    print("1.Deposite")
    print("2.Withdraw")
    print("3.Display net balance")
    print("4. Exit")

    ch=int(input("Enter choice"))
    if ch==1:
        u.deposit()
    elif ch==2:
        u.withdraw()
    elif ch==3:
        u.display()
    else:
        break
    print("Exit")
```

```
5)def SlectionSort(arr):
    for i in range(len(arr)):
        minindex=i
        for j in range (i+1,len(arr)):
            if(arr[j]<arr[minindex]):
                minindex=j
        arr[i],arr[minindex]=arr[minindex],arr[i]
    return arr

def BubbleSort:
    n=len(arr)
    for i in range(n-1):
        for j in range(0,n-i-1):
            if(arr[j]>arr[j+1]):
                arr[j],arr[j+1]=arr[j+1],arr[j]
    return arr

def Display(scores,top_n=3):
    top_scores=sorted( top_n,[:-1]
    print(" Top 5 scores ")
    for i,score in enumerate(top_scores,1):
        print(f"{i}.{score}")

ch=int(input("Enter:"))
lst=[]
for i in range(ch):
    no=int(input("enter marks"))
    lst.append(no)
print(lst)
s=SlectionSort(lst)
Display(s)
b=BubbleSi(lst)
Display(b)
```

```
6)def quick5(arr):
    if(len(arr)<=1):
        return arr
    else:
        pivot=arr[len(arr)//2]
        mid=[x for x in arr if x==pivot]
        left=[x for x in arr if x<pivot]
        right=[x for x in arr if x>pivot]
        return quick5(left)+mid+quick5(right)

def display(arr):
    if(len(arr)<=6):
        print(*arr[::-1])
    else:
        print(*arr[:len(arr)-6:-1])

lst=[]
n=int(input("Enter no.of students whos marks to be inserted : "))
for i in range(n):
    m=int(input("Enter marks : "))
    lst.append(m)

#Quick sort
a=quick5(lst)
print("sorted array:",,a)
b=display(a)
print("toppers:",,b)
```

```
7)Introduction to linked lists
using namespace std;

int size = 10000;

class Stack{
    public:
    int top = -1;
    char ar[10000];
    void push(char x){
        if(top==size-1){
            return;
        }
        else{
        top++;
        ar[top] = x;return;
        }
    }
    char pop(){
        if(top==-1){
            return ar[top];
        }
        else{
        char temp = ar[top];
        top--;
        return temp;
        }
    }
    char top(){
        return ar[top];
    }
    int isEmpty(){
        if(top==-1){
            return 1;
        }
        else{
            return 0;
        }
    }
};

void checkStack(){
    Stack s = Stack();
    cout<<"Pushing a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z,0,1,2,3,4,5,6,7,8,9 to stack\n";
    for(int i='a';i<='z';i++){
        s.push(i);
    }
    cout<<"Top of stack is "<<(char)s.top()<<endl;
    cout<<"Popping 5 elements\n";
    for(int i=0;i<5;i++){
        cout<<(char)s.pop()<<" ";
    }
    cout<<"\n";
    cout<<"Top of stack is "<<(char)s.top()<<endl;
    cout<<"Stack is empty? "<<s.isEmpty()<<endl;
}

int main(){
    checkStack();
    return 0;
}
```

```
8)Introduction to linked lists
using namespace std;

class Node{
    public:
    int data;
    Node* next;
    Node(int d){
        data = d;
        next = NULL;
    }
    ~Node(){
        int temp = data;
        while(next){
            int cur = next->data;
            Node* tmp = next;
            next = next->next;
            tmp->next = NULL;
            delete tmp;
        }
        cout<<"Memory freed for node with data "<<temp<<endl;
    }
};

void insertAtHead(Node* &head, int d){
    Node* temp = new Node(d);
    temp->next = head;
    head = temp;
}

void insertAtTail(Node* &tail, int d){
    Node* temp = new Node(d);
    tail->next = temp;
    tail = temp;
}

void insertAtPosition(Node* &head, Node* &tail, int position, int d){
    if (position == 1){
        insertAtHead(head, d);
        return;
    }
    Node* temp = head;
    int cnt = 1;
    while (cnt < (position-1)){
        temp = temp->next;
        cnt++;
    }
    if (temp->next == NULL){
        insertAtTail(tail, d);
        return;
    }
    Node* nodeToInsert = new Node(d);
    nodeToInsert->next = temp->next;
    temp->next = nodeToInsert;
}

void deleteNode(int position, Node* &head){
    if (position == 1){
        Node* temp = head;
        head = head->next;
        temp->next = NULL;
        delete temp;
        return;
    }
    Node* prev = NULL;
    int cnt = 1;
    while (cnt < position){
        prev = head;
        head = head->next;
        cnt++;
    }
    prev->next = head->next;
    head->next = NULL;
    delete head;
}

void print(Node* head){
    while (head){
        cout<<head->data<<" ";
        head = head->next;
    }
    cout<<endl;
}

int getLength(Node* head){
    int cnt = 0;
    while (head){
        cnt++;
        head = head->next;
    }
    return cnt;
}

int getMiddle(Node* head){
    int len = getLength(head);
    int mid = (len/2) + 1;
    while (mid > 1){
        head = head->next;
        mid--;
    }
    return head->data;
}

int getNthFromEnd(int n, int len, Node* head){
    int mid = (len - n) + 1;
    while (mid > 1){
        head = head->next;
        mid--;
    }
    return head->data;
}

void reverse(Node* &head, Node* &tail){
    if (head == NULL || head->next == NULL){
        return;
    }
    Node* prev = NULL;
    Node* curr = head;
    Node* next = NULL;
    while (curr != NULL){
        next = curr->next;
        curr->next = prev;
        prev = curr;
        curr = next;
    }
    tail = head;
    head = prev;
}

int main(){
    Node* head = NULL;
    Node* tail = NULL;
    int position, d;
    int len, n, mid;
    int temp;
    int cmd;
    while (true){
        cout<<"Enter command: \n";
        cout<<"1. Insert at head\n";
        cout<<"2. Insert at tail\n";
        cout<<"3. Insert at position\n";
        cout<<"4. Delete node\n";
        cout<<"5. Print\n";
        cout<<"6. Get length\n";
        cout<<"7. Get middle\n";
        cout<<"8. Get nth from end\n";
        cout<<"9. Reverse\n";
        cout<<"10. Exit\n";
        cin>>cmd;
        if (cmd == 1){
            cout<<"Enter data: ";
            cin>>d;
            insertAtHead(head, d);
            if (tail == NULL){
                tail = head;
            }
            cout<<"Inserted at head\n";
            print(head);
            cout<<"Head: "<<head->data<<endl;
            cout<<"Tail: "<<tail->data<<endl;
            cout<<"Length: "<<getLength(head)<<endl;
            cout<<"Middle: "<<getMiddle(head)<<endl;
            cout<<"Reverse: ";
            reverse(head, tail);
```