* Data structure used:

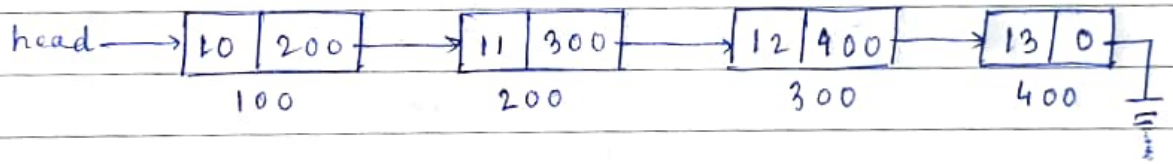
→ Linked List:-

- A linked list is an ordered collection of data in which each element contains minimum two values, data and link(s) to its successor (and/or predecessor).

→ Type of linked List used:-

- Singly linked List (SLL)

i) A linked list in which every node has one link field, to provide information about where the next node of list is, is called as singly linked list.



→ Pseudocode for class student

1. ~~class~~ declare class student
2. Declare int roll
   string name
   student *next
3. Create function student* create()
   void display (student *head)
   student* beg_add (student *head)
   student* end_add (student *head)
   student* btwn_add(student *head)
   student* beg_del (student *head)
   student* end_del (student *head)
   student* btwn_del (student *head)
   student* total (student *head)
   student* con(student *head, student *head1)

→ Pseudocode for student* create()

1. Declare int n
   student *head, *p
2. Initialize ~~initia~~ head = NULL
3. Display "Enter number of members:"
4. Read n
5. for i=0; i<n; i++ do

begin

if head == NULL then

declare head = new student

display "Enter pen of president:"

read head→roll
display "Enter name of president"
read head→name
initialize head→next = NULL
initialize p = head

else

initialize p→next = new student
initialize p = p→next
display "Enter pen of member:"
read p→roll
display "Enter name of member:"
read p→name
initialize p→next = NULL

end

6. return head

→ Pseudocode for void display (student *head)

1. Create student *p
2. for (p = head; p→next != NULL; p = p→next) do
   ~~do~~ begin
      display p→roll " " p→name "→"

   end
3. display p→roll " " p→name
4. return

→ Pseudocode for student* beg_add (student *head)

1. Create student *p
2. Initialize p = new student

3. Display "Enter pen of new president:"
4. read p→roll
5. Display "Enter name of new president:"
6. read p→name
7. Initialize p→next = NULL
8. Initialize p→next = head
9. Store head = p
10. return head

→ Pseudocode for student* end_add (student *head)

1.
1. Create student *p, *q
2. Initialize p = new student
3 Display "Enter pen of secretary:"
4. read p→roll
5 Display "Enter name of secretary:"
6. read p→name
7. Store p→next = NULL
8. for (q=head ; q→next ! NULL ; q=q→next) do
   begin
   end
9. Store q→next = p
10. return head

→ Pseudocode for student* btwn_add (student *head

1. Declare int y
2. Create student* *p, *q
3. initialize p = new student
4. Display "Enter pen of new member:"

2. for (q = head; q→next→next != NULL ; q = q→next) do
begin
end ~~store p = q→next~~
3. store p = q→next
4. delete p
5. Store q→next = NULL
6. return

→ Pseudocode for student* dtwn_del (student *head)

1. Declare int y
2. Create student *p, *q
3. Display "Enter pan of member which is to be deleted: "
4. read y
5. if head == NULL then
   display "Linked List is empty"
   return head
6. store p = head
7. while (p→next != NULL) do
begin
   if p→next→roll == y then
      store q = p→next
      store p→next = p→next→next
      delete q
   store p = p→next
end
8. return head

→ Pseudocode for student* total (student *head)

1. Create student *p
2. Initialize total = 0
3. for p = head; p != NULL; p = p→next do
   begon
   ~~tea~~ increment total
   end
4. Display "Total number of students are: " total
5. return

→ Pseudocode for student * con (student *head, student *head1)

1. Create student *p
2. for p = head; p→next != NULL; P = p→next do
   begon
   end
3. p→next = head1
4. return head

→ Pseudo code for int main()

1. Initialize flag = 1, flag1 = 1, flag2 = 1
2. Declare char, char1, char2
3. Create student *head, *head1
4. Create student obj
5. while (flag) do
   begon

```
Display "YOUR CHOICES ARE:"
Display "1. LINKED LIST1: 2. LINKED LIST2:
        3. CONCATENATE LINKED LIST1 & 2:
        4. Exit"
Display "Enter choice:"
read char O
switch (char O)
    case 1:
    while (flag) do
    begin
        Display "what operations on list1
        would you like to perform?"
        Display "1. Enter pen, name of
                members:"
        2. Add new president:
        3. Add secretary
        4. Add new members
        5. Remove president
        6. Remove secretary
        7. Remove members
        8. Total members present
        9. Exit
    Display "Enter choice;"
    read char1
    switch (char1)
        case 1:
            call function obj.create()
                obj.display(head)
            break
```

```
case 2:
    call function head = obj.beg_addinhead(
                         obj.display(head)
    break
case 3:
    call function head = obj.end_add(head)
                         obj.display(head)
    break
case 4:
    call function head = obj.btwn_add(head)
                         obj.display(head)
    break
case 5:
    Display "Removed president: "
    call function head = obj.beg_del(head)
                         obj.display(head)
    break
case 6:
    Display "Removed secretary: "
    call function head = obj.end_del(head)
                         obj.display(head)
    break
case 7:
    call function head = obj.btwn_del(head)
    Display "Member removed: "
    call function obj.display(head)
    break
case 8:
    call function obj.total(head)
    break
```

```
case 9:
        flag1 = 0
        break
default:
        display "Enter valid
                choice!!!"
        break
end
break
case 2:
while (flag2) do
begin
    display "what operations on list2
            would you like to perform?"
    display "1.Enter pan, name of members:"
            2. Add new president
            3. Add secretary
            4. Add new members
            5. Remove president
            6. Remove secretary
            7. Remove members
            8. Total members present
            9. Exit"
display "Enter choice:"
read char2
switch (char2)
    case 1:
        call function head1 = obj.create
                obj.display(head1)
    break
    case 2:
        call function head1 = obj.big_add(head1)
                obj.display(head1)
    break
```

```
case 3:
    call function head1 = obj.end_add(head1)
    obj.display(head1)
    break
case 4:
    call function head1 = obj.btun_add(head1)
                          obj.display(head1)
    break
case 5:
    display "Removed president: "
    call function head1 = obj.beg_del(head1)
                          obj.display(head1)
    break
case 6:
    display "Removed secretary: "
    call function obj.end_del(head1)
                  obj.display(head1)
    break
case 7:
    call function head1 = obj.btun_add(head1)
    display "Member removed: "
    call function obj.display(head1)
    break
case 8:
    call function obj.total(head1)
    break
case 9:
    store flag2 = 0
    break
default:
    display "Enter valid choice!!!"
    break
```

break

case 3:

call function head = obj.concheaad, head1)

obj.display (cheaad)

break

case 4:

store flag = 0

break

default:

~~cout <<~~ display "Enter valid choice; "

break

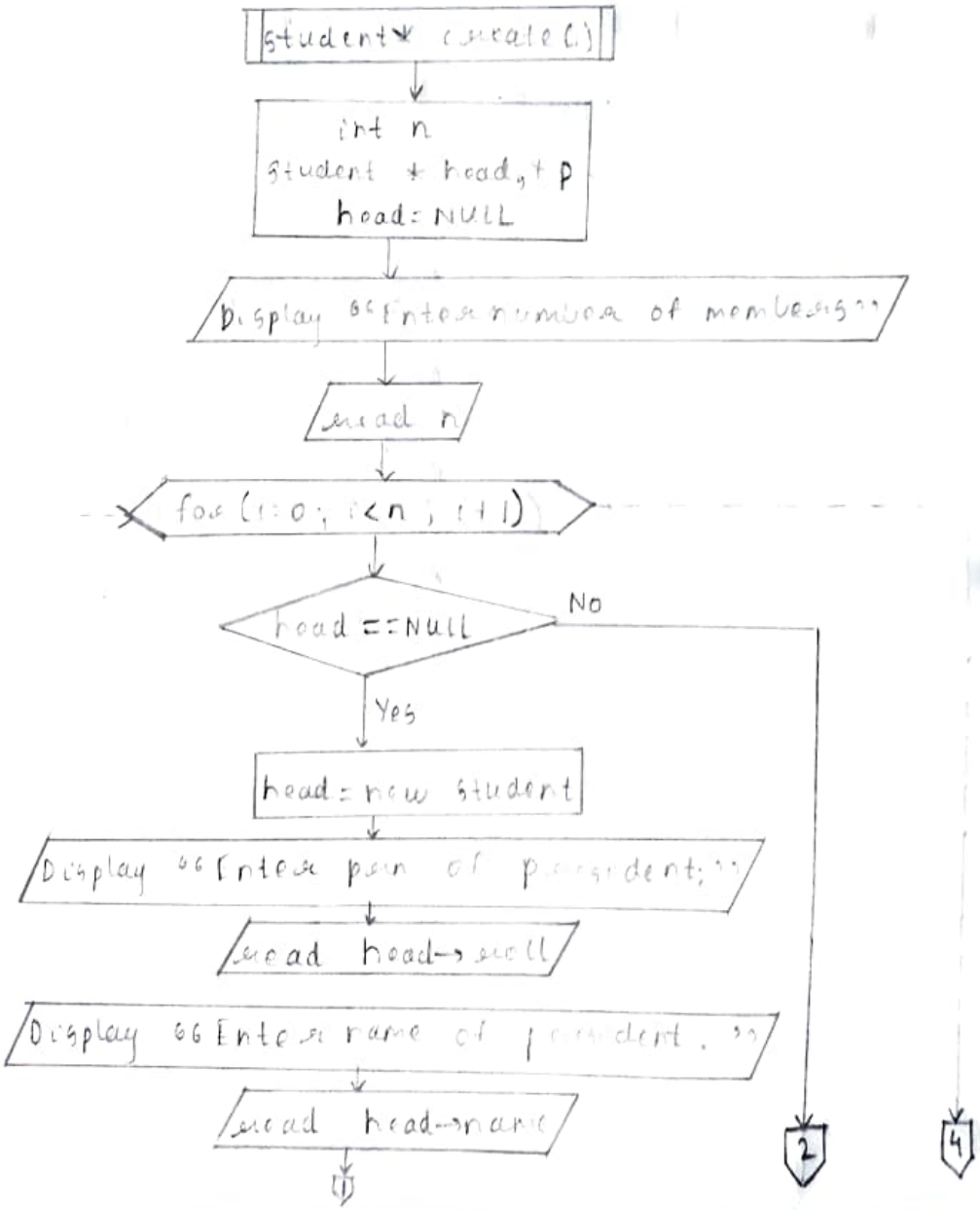
end

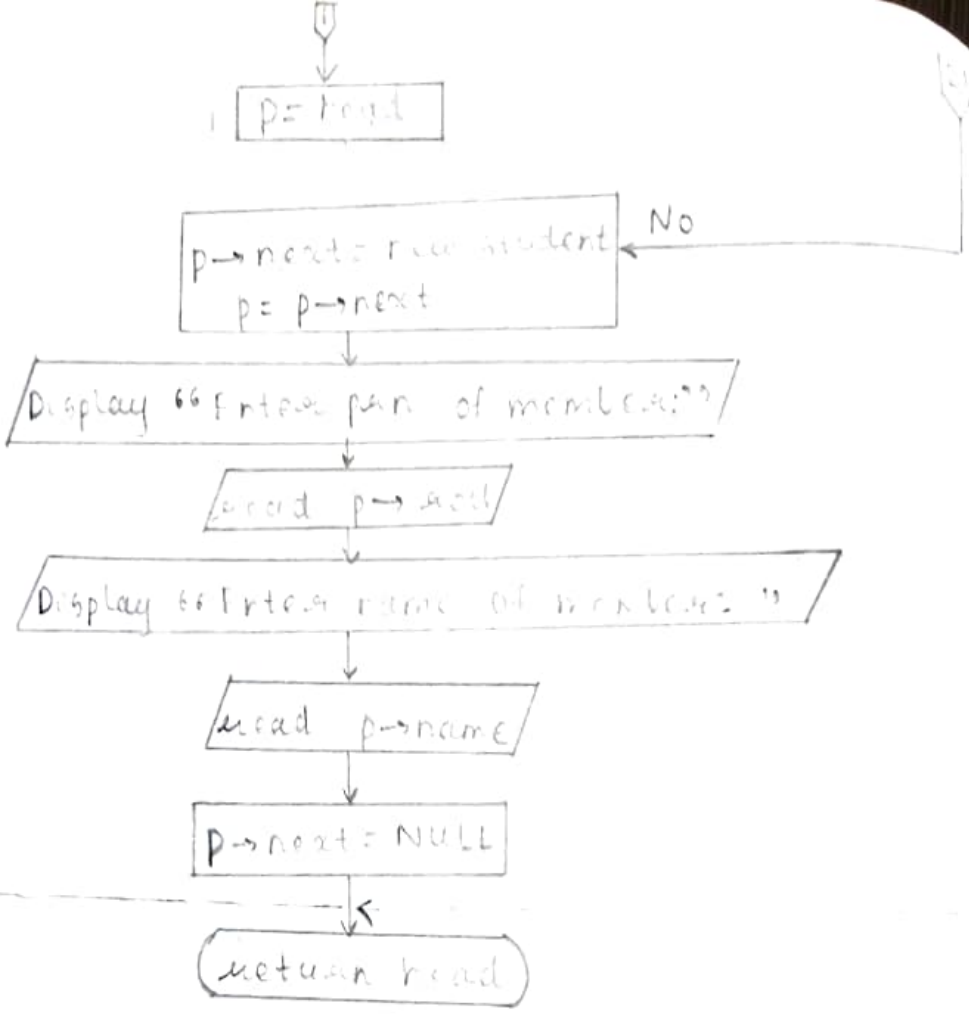
6. Stop

chart for class student

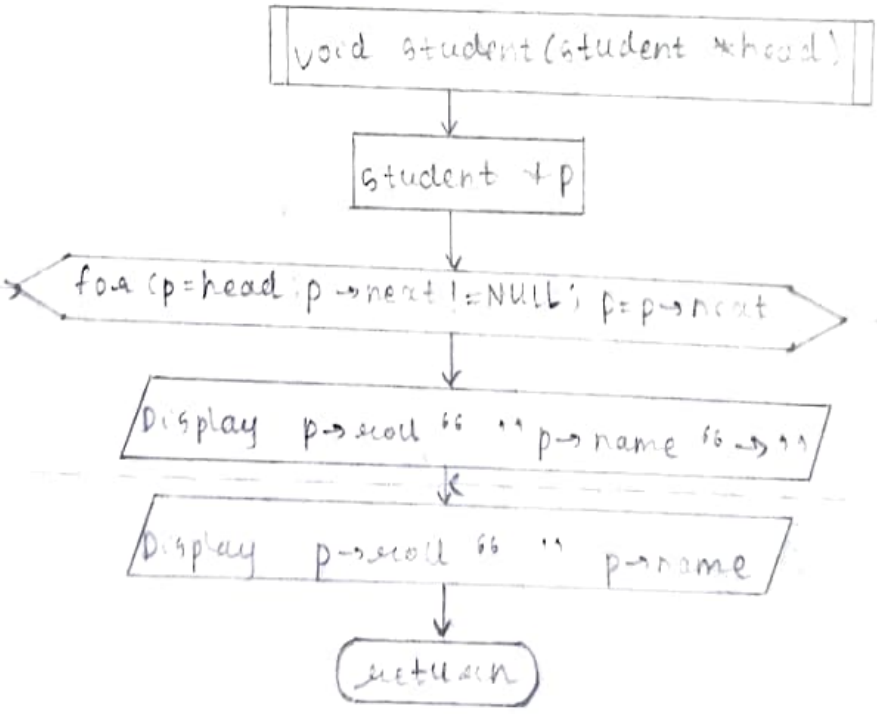
class student
int roll string name student *next
student* create() void display (student *head) student* beg_add (student *head) student* end_add (student *head) student* btw_add (student *head) student* beg_del (student *head) student* end_del (student *head) student* btw_del (student* head) student* con (student *head, student *head1)

→ Flowchart for student* create()

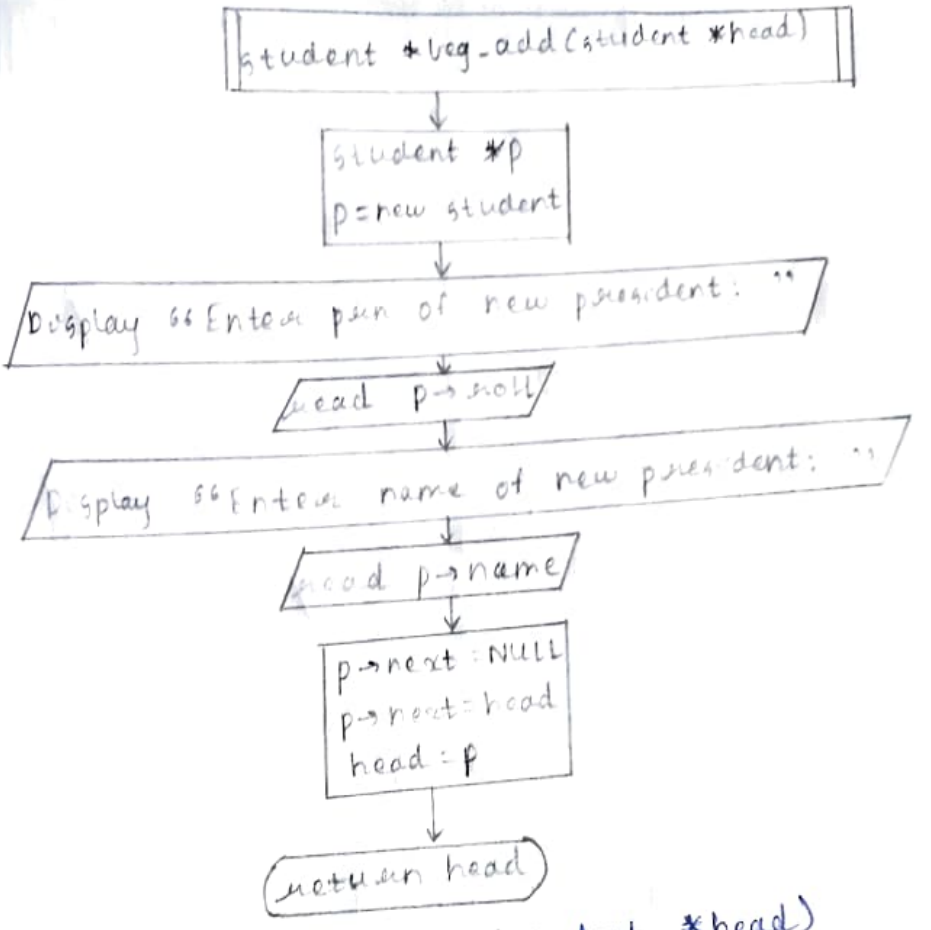




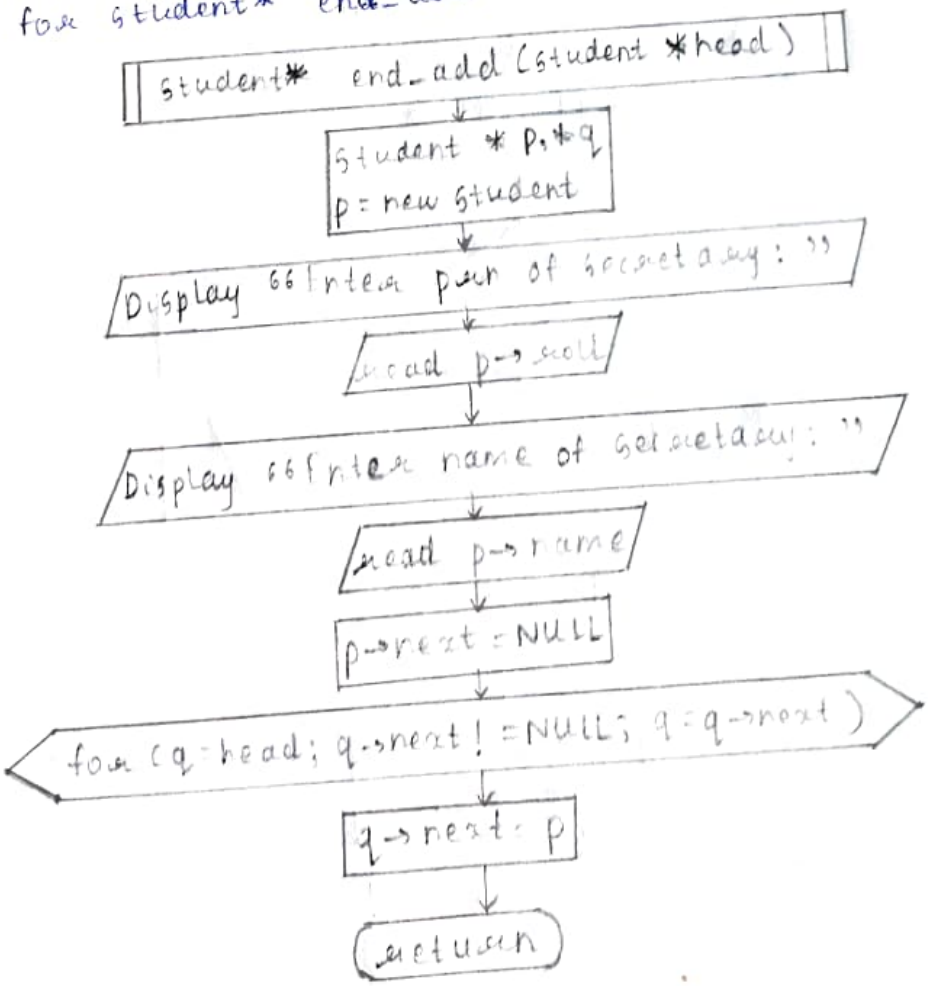
→ Flowchart for void display (Student *head)



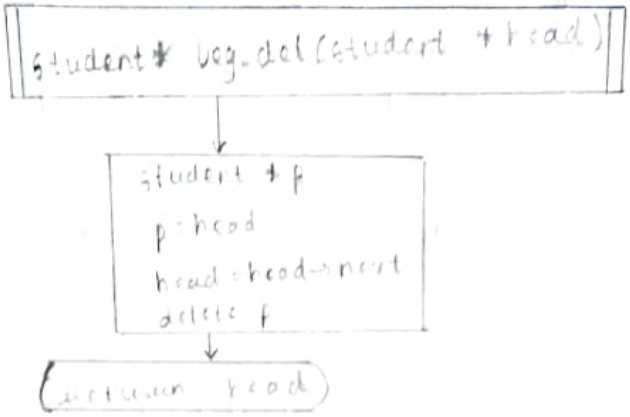
→ Flowchart for student* beg_add (student *head)



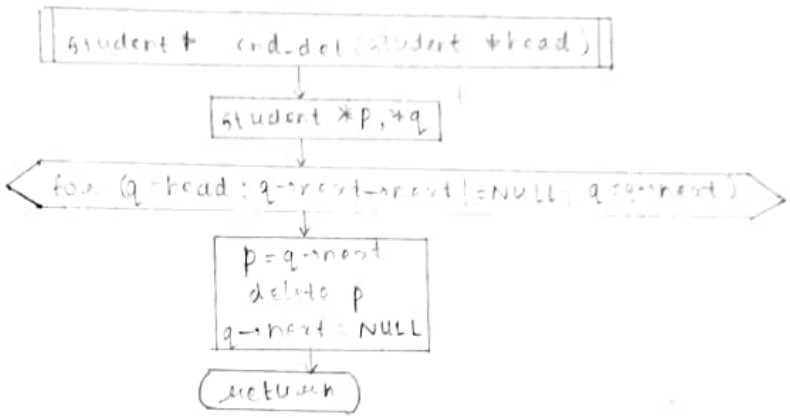
→ Flowchart for student* end_add (student *head)



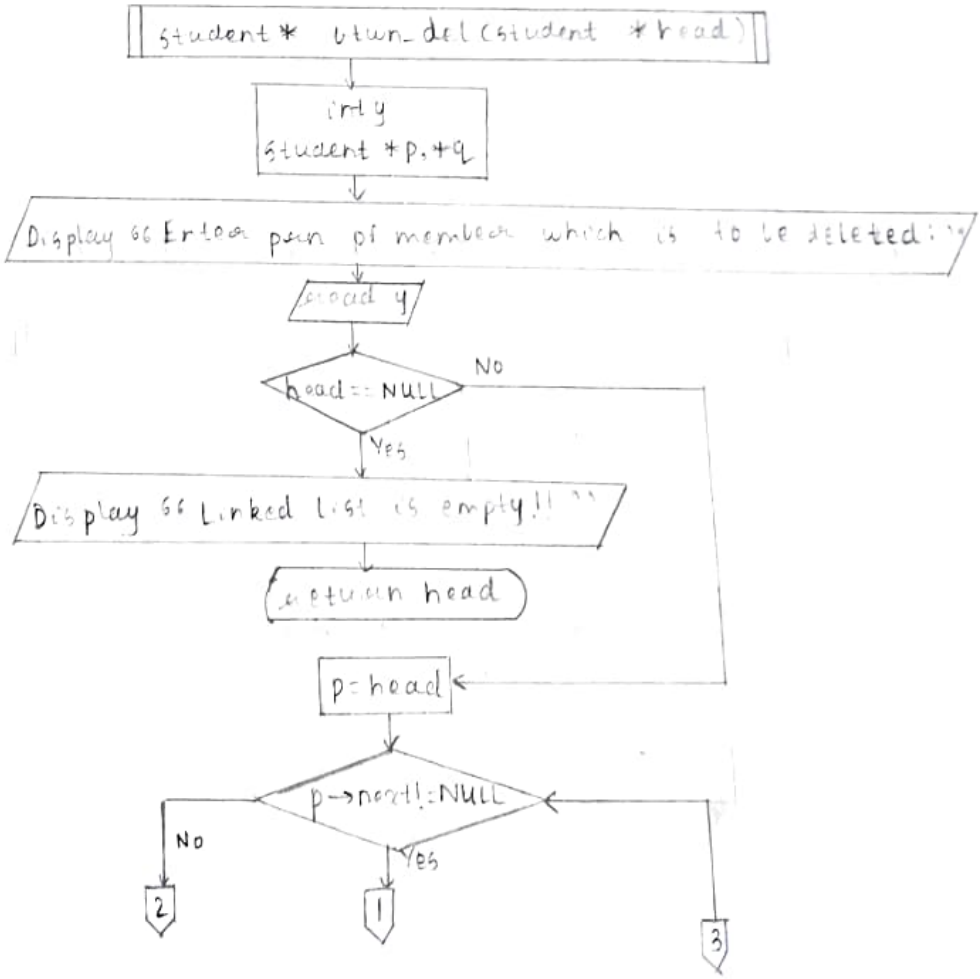
→ flowchart for student * beg_del (student *head)



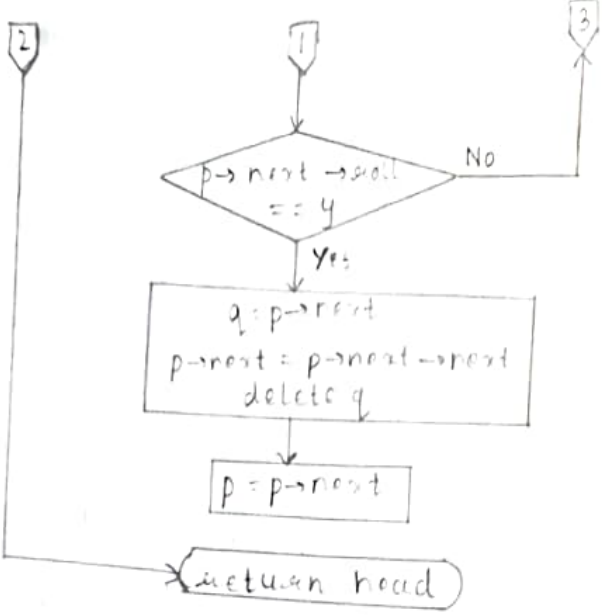
→ flowchart for student * end_del (student *head)



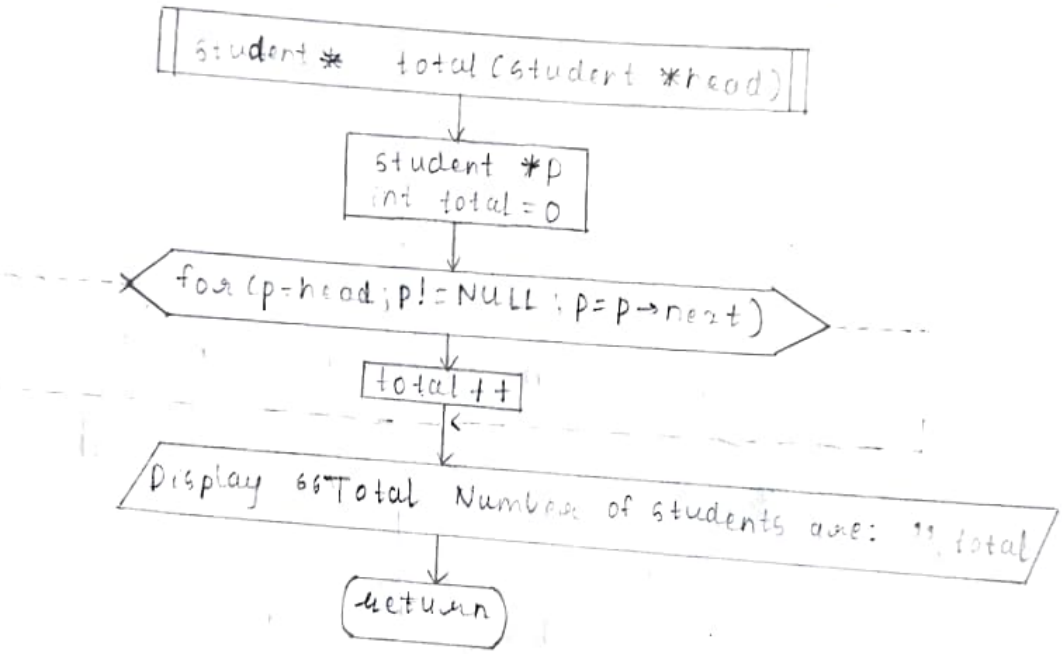
→ flowchart for student * btwn_del (student *head)



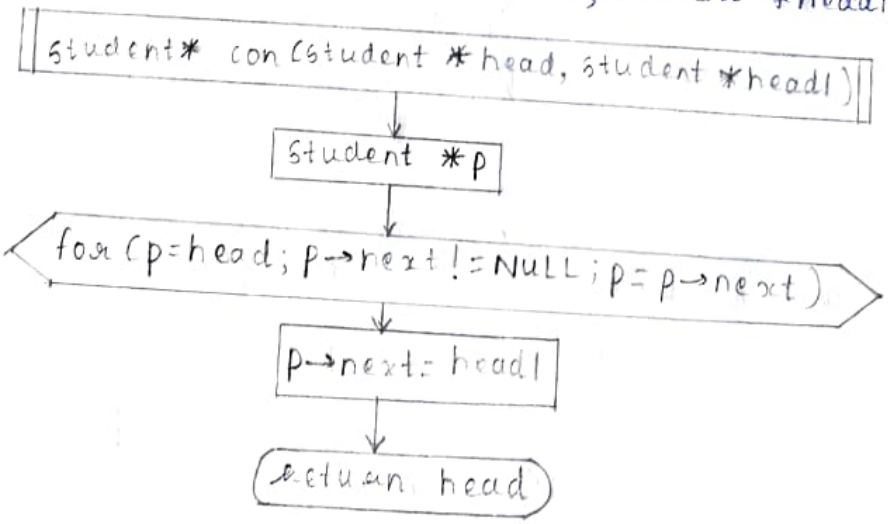
~~Flowchart~~

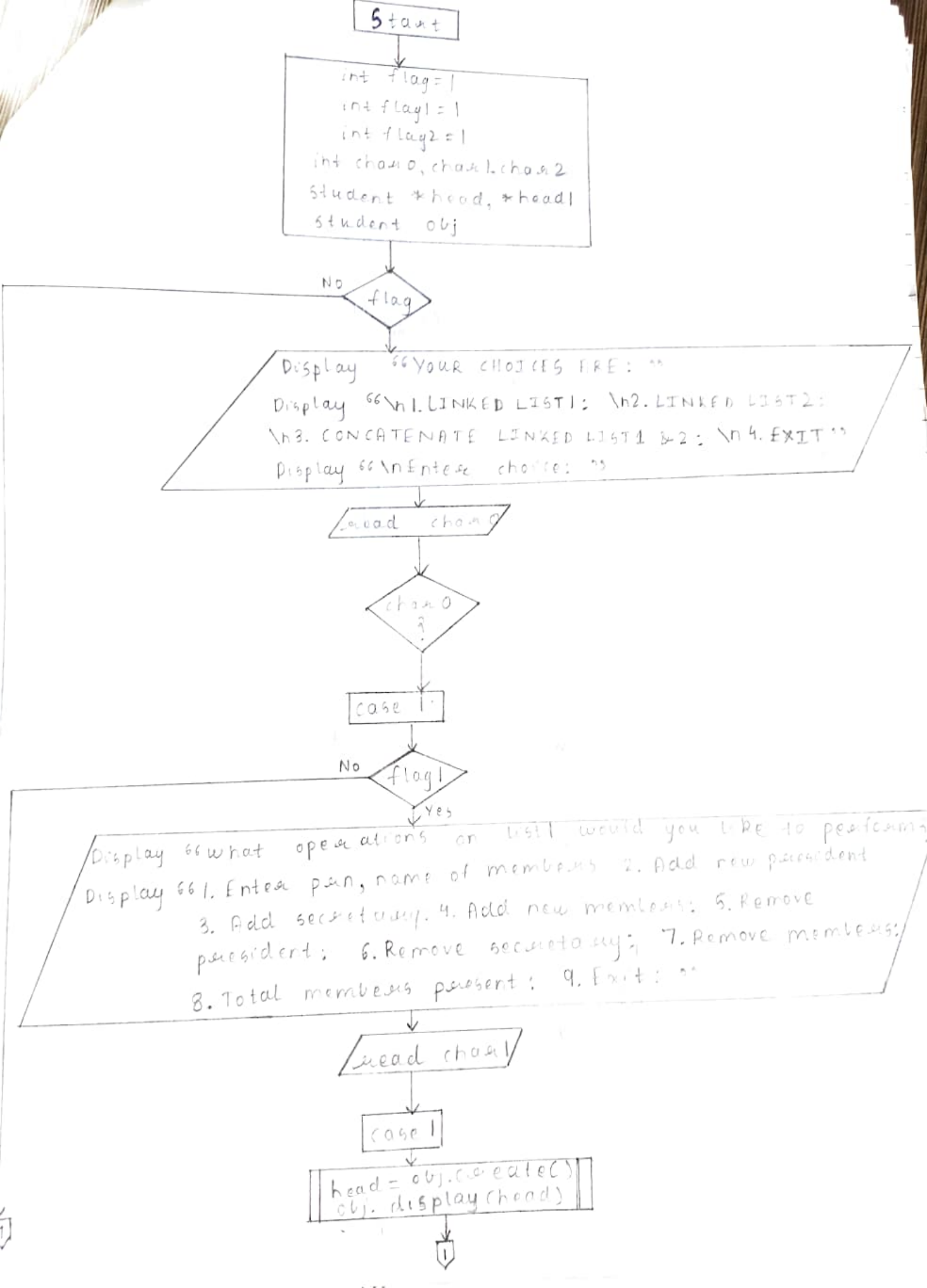


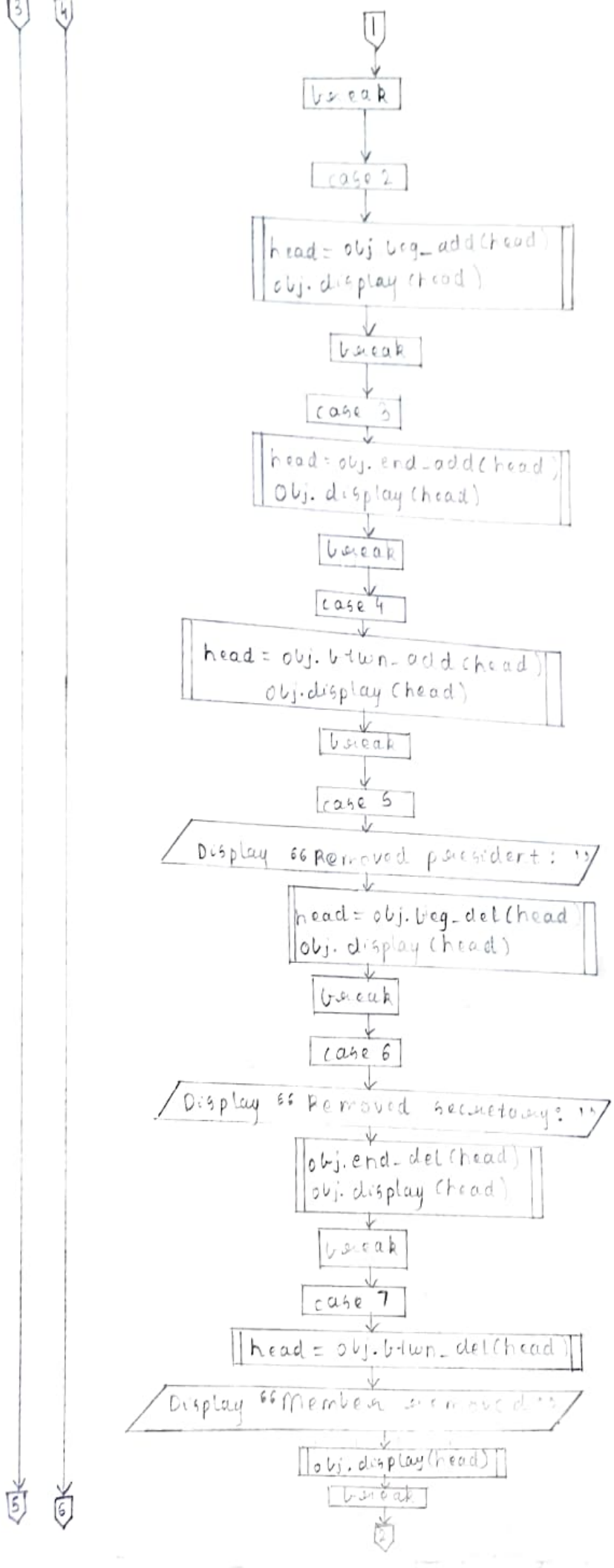
Flowchart for student* total (student *head)

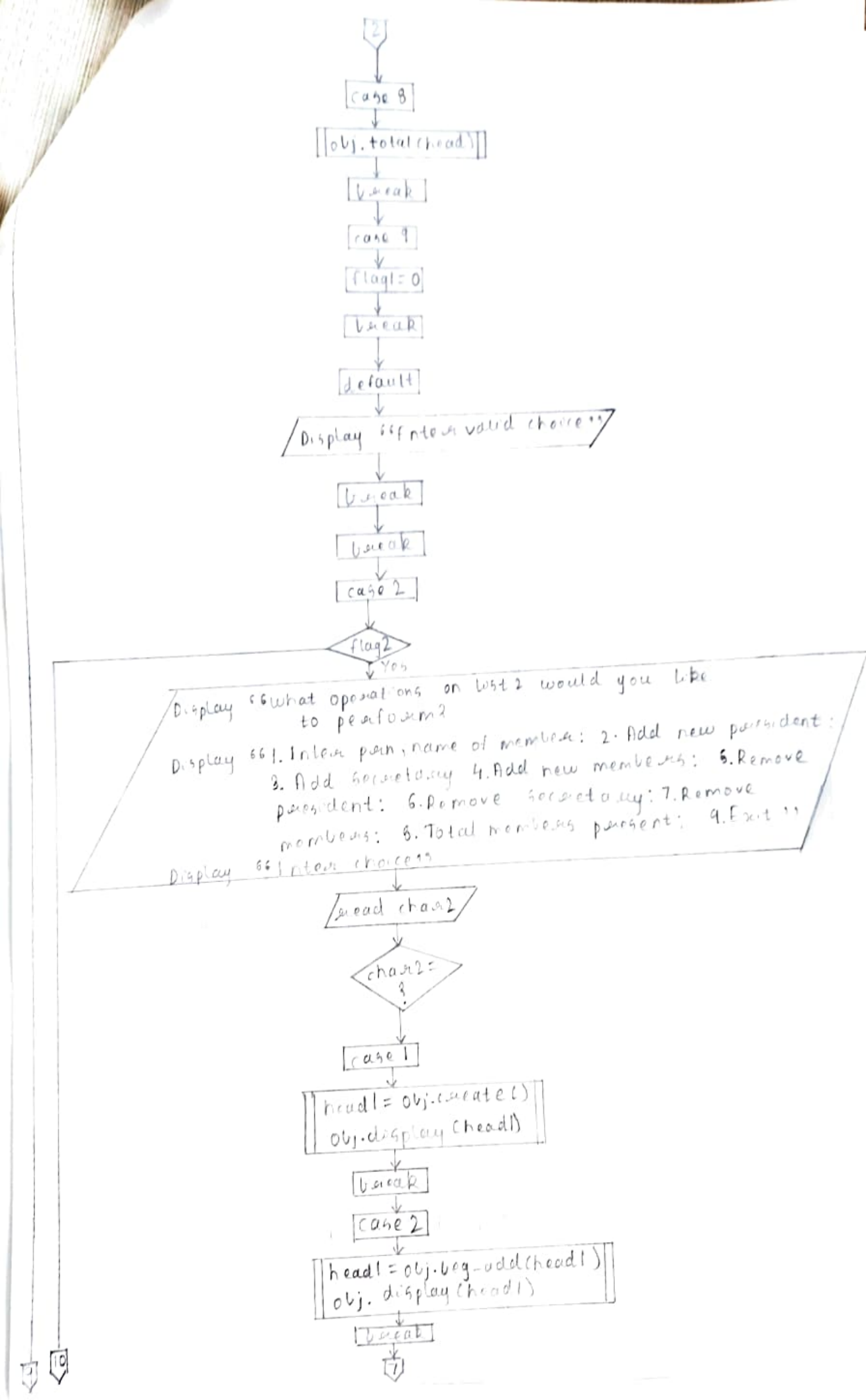


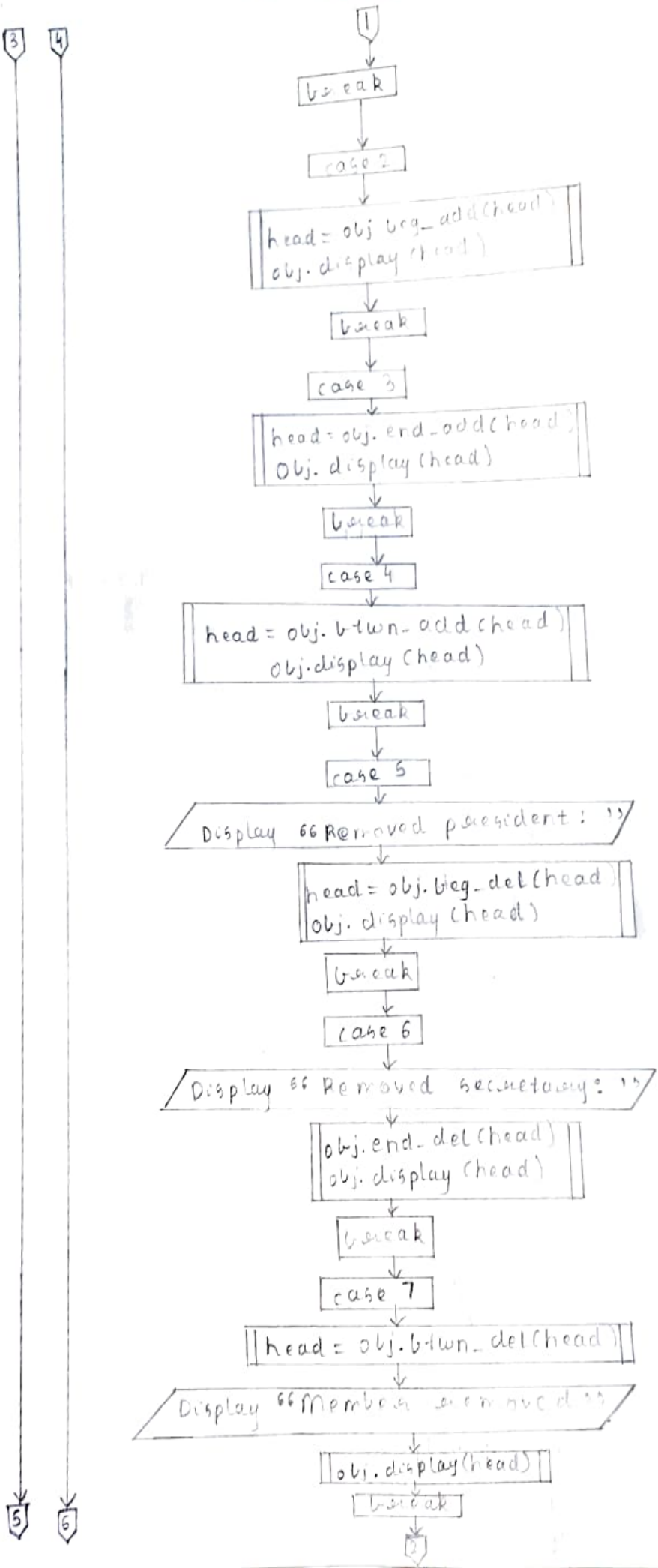
Flowchart student* con(student *head, student *head1)

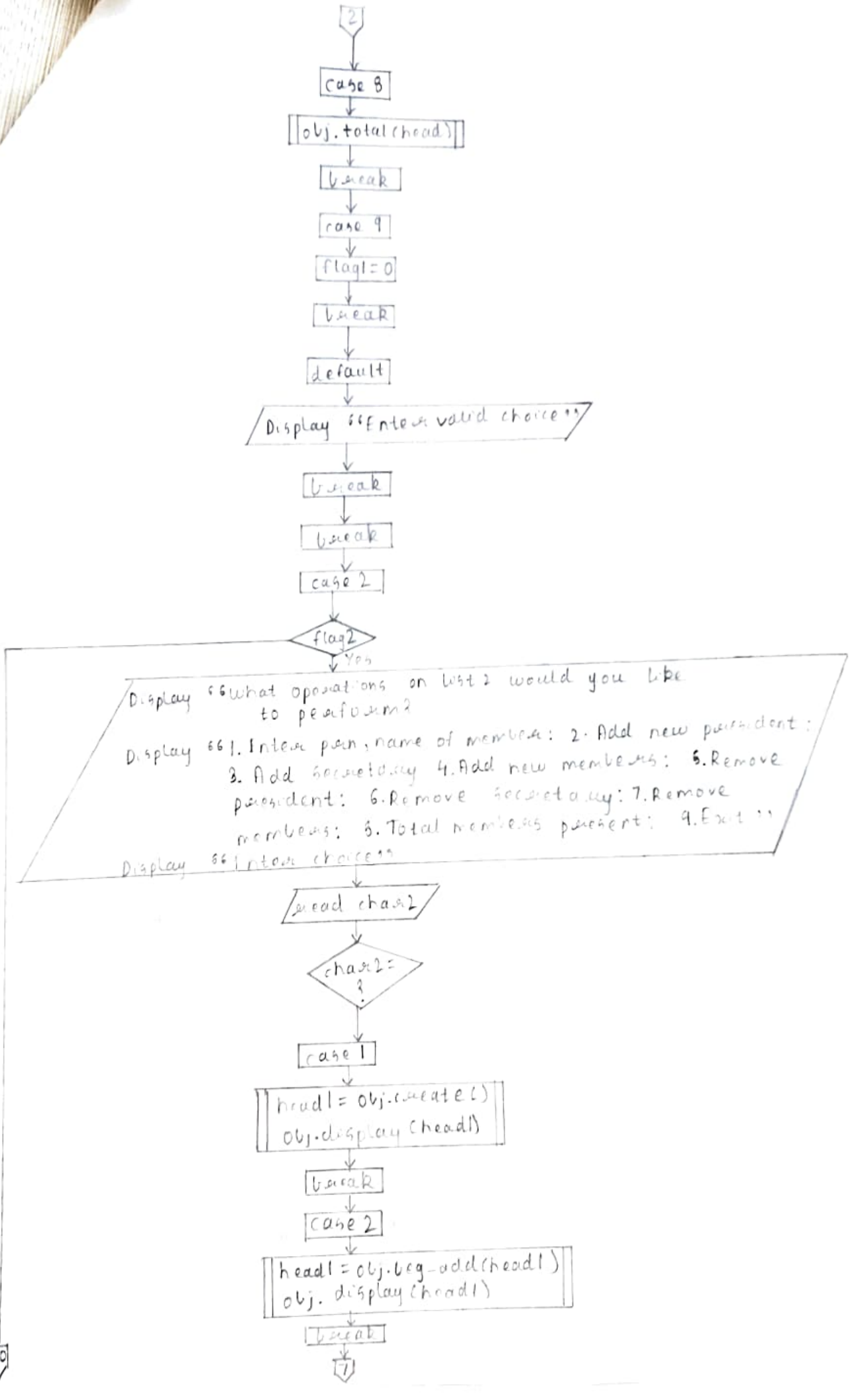


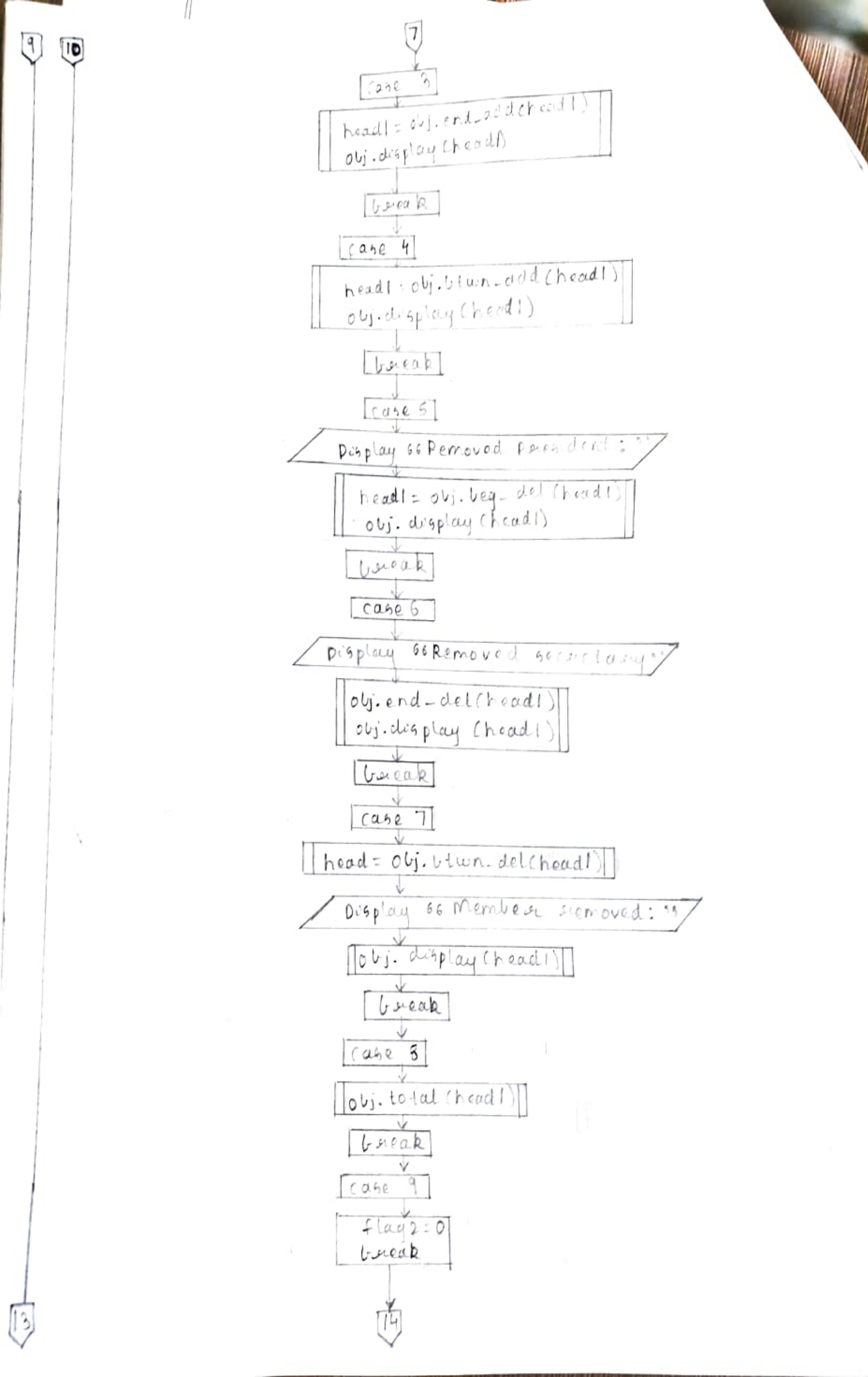

Start
int flag=1
int flag1=1
int flag2=1
int choice0, choice1, choice2
student *head, *head1
student obj
No
flag
Display "YOUR CHOICES ARE:"
Display "\n1. LINKED LIST1: \n2. LINKED LIST2:
\n3. CONCATENATE LINKED LIST 1 & 2: \n4. EXIT"
Display "\nEnter choice:"
read choice0
choice0
?
case 1:
No
flag1
Yes
Display "what operations on list would you like to perform
Display "1. Enter pin, name of members 2. Add new president
3. Add secretary. 4. Add new members: 5. Remove
president: 6. Remove secretary; 7. Remove members:
8. Total members present: 9. Exit:"
read choice1
case 1
head = obj.create()
obj. display(head)
1

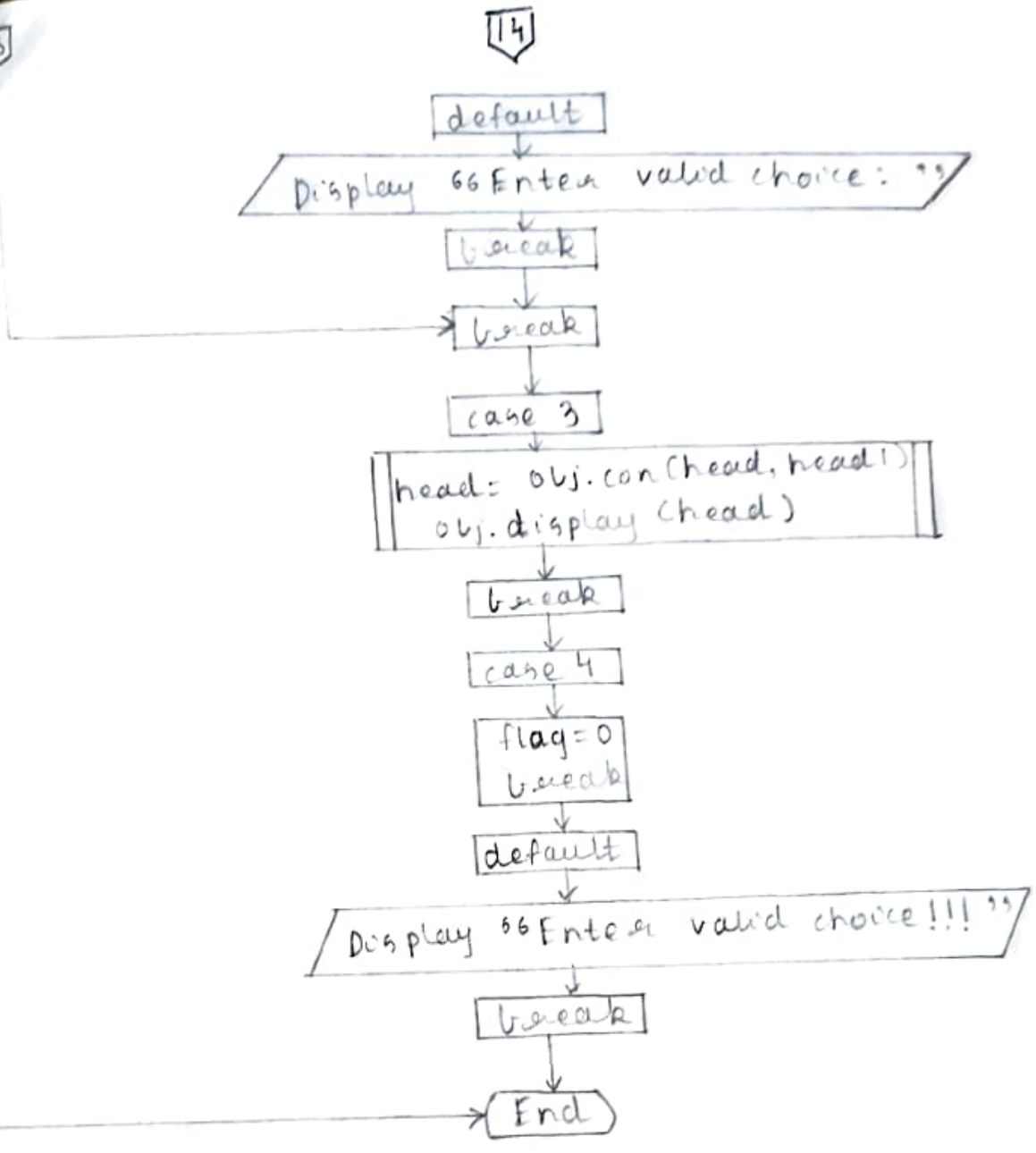

3
4
1
break
case 2
head = obj beg_add(head)
obj. display (head)
break
case 3
head = obj. end_add(head)
Obj. display (head)
break
case 4
head = obj. btwn_add(head)
obj.display (head)
break
case 5
Display "Removed president : "
head = obj. beg_del(head)
obj. display (head)
break
case 6
Display "Removed secretary : "
obj. end_del (head)
obj. display (head)
break
case 7
head = obj. btwn_del(head)
Display "Member removed"
obj. display(head)
break
2
5
6


2
case 8
obj.totalcheadl()
break
case 9
flag1=0
break
default
Display "Enter valid choice"
break
break
case 2
flag2
Yes
Display "What operations on list 2 would you like to perform?
Display "1. Enter pers name of members: 2. Add new president: 3. Add secretary 4. Add new members: 5. Remove president: 6. Remove secretary: 7. Remove members: 8. Total members present: 9. Exit"
Display "Enter choice"
read char2
char2=='1'
case 1
head1=obj.create()
obj.display(head1)
break
case 2
head1=obj.beg_add(head1)
obj.display(head1)
break
7
10


3
4
1
break
case 2
head = obj.beg_add(head)
obj.display(head)
break
case 3
head = obj.end_add(head)
Obj.display(head)
break
case 4
head = obj.btwn_add(head)
obj.display(head)
break
case 5
Display "Removed president:"
head = obj.beg_del(head)
obj.display(head)
break
case 6
Display "Removed secretary:"
obj.end_del(head)
obj.display(head)
break
case 7
head = obj.btwn_del(head)
Display "Member removed"
obj.display(head)
break
5
6
2


2
case 8
obj.total(head)
break
case 9
flag1 = 0
break
default
Display "Enter valid choice"
break
break
case 2
flag2
Yes
Display "What operations on list 2 would you like to perform?
Display "1. Enter pan, name of member: 2. Add new president: 3. Add secretary 4. Add new members: 5. Remove president: 6. Remove secretary: 7. Remove members: 8. Total members present: 9. Exit"
Display "Enter choice"
read char2
char2 = ?
case 1
head1 = obj.create()
obj.display(head1)
break
case 2
head1 = obj.beg_add(head1)
obj.display(head1)
break
10
1


9
10
7
case 3
head1 = obj.end_add(head1)
obj.display(head1)
break
case 4
head1 = obj.btwn_add(head1)
obj.display(head1)
break
case 5
Display "Removed president:"
head1 = obj.beg_del(head1)
obj.display(head1)
break
case 6
Display "Removed secretary"
obj.end_del(head1)
obj.display(head1)
break
case 7
head = obj.btwn_del(head1)
Display "Member removed:"
obj.display(head1)
break
case 8
obj.total(head1)
break
case 9
flag2 = 0
break
13
14


default
Display "Enter valid choice: "
break
break
case 3
head= obj.con(head, head1)
obj.display(head)
break
case 4
flag=0
break
default
Display "Enter valid choice!!!"
break
End

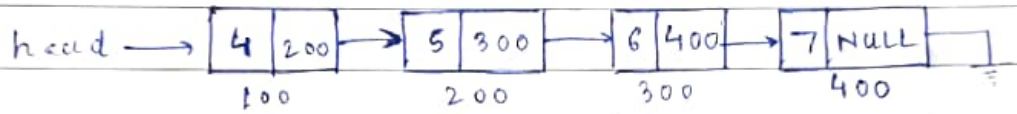
What is a linked list? State its types.

A linked list is an ordered collection of data in which each element contains minimum two values, data and link(s) to its successor (and/or) predecessor.

→ Types of linked list:-

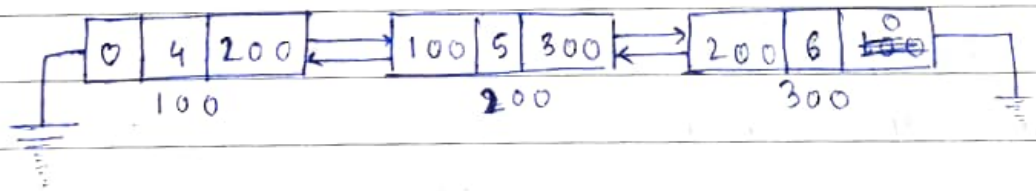
1. Singly linked list:-

A linked list in which every node has one link field, to provide information about where the next node of list is, is called as singly linked list.



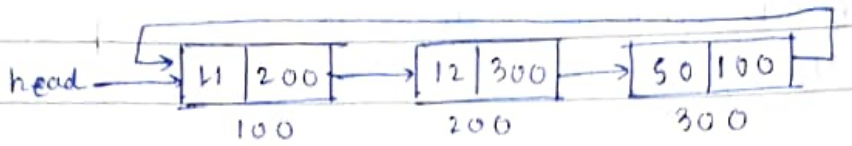
2. Doubly linked list

- In doubly linked list, each node has two link fields to store information about who is the next and also about who is ahead of the node.
- Hence each node has knowledge of its successor and also its predecessor.

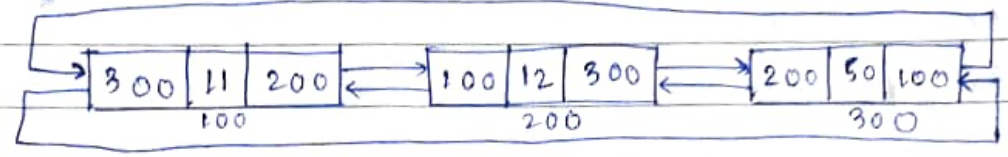


3. Circular linked list

- The link field of last node is set to NULL in linked list to mark end of list.
- This link field of last node can be set to point first node rather than NULL. Such a linked linked list is called circular linked list.



4. Circular doubly linked list
- Like the doubly linked list, it has an extra pointer called the previous pointer.
- Similar to the circular linked list, its last node points at the head node.



Q2. Write applications of SLL.

Ans. 1. Implementing stacks and queues:-
- Singly linked lists can be used to implement stacks and queues.
- In a stack, elements are added and removed from one end of the list, while in a queue, elements are added at one end and removed from the other end of the list.

2. Navigation in web browsers:-
- Singly linked lists can be used to store the browsing history on web browsers.

3. Implementing symbol tables:-
- Symbol tables are data structures used to store key-valued pairs.
- Singly-linked lists can be used to implement symbol tables, with each node storing a key-value pair.