**EXPERIMENT :1B SINGLY LINKED LIST DATE:**

**AIM:**

WAP to generate a student record using linked list.  
a) Find largest and smallest element of linked list and move smallest to the last.  
b) Swap adjacent elements by rearranging links  
    Ex: if a linked list is 1->2->3->4->5->6->7, then after swapping adjacent elements it should become 2->1->4->3->7>6  
c) Merge two linked list.  
d) Split linked list in two halves. (Based on Roll Number)

**THEORY:**

Linked Lists is a non-sequential collection of data items. For every data item in a linked list there is an associated pointer that contains the dress of the next the data item. The basic component of a linked list is a node having two field. One for holding the data and another for holding the address of the next node.

While the elements of an array occupy contiguous memory locations, those of a linked list are not constrained to be stored in adjacent locations. The order of the elements is maintained by explicit links between them.

TYPICAL NODE

|  |  |
| --- | --- |
| DATA | LINK |

A linked list is a collection of elements called Nodes. Each Node as two sections to it. The data part and the link part. The Null in the last Node indicates that it is the last Node in the list.

SYNTAX

A Node in C may be implemented by using a self-referential structure that has a a member that points to a structure of the same type.

NOTE: stdlib.h library should be included.

Struct node{

Int rollNo

Char name[20]

Int makrs[3]

Int total

Float avg;

Struct node \*link;

};

To Create A Node

Struct node \*temp;

temp=(struct node\*)malloc(sizeof(struct node));

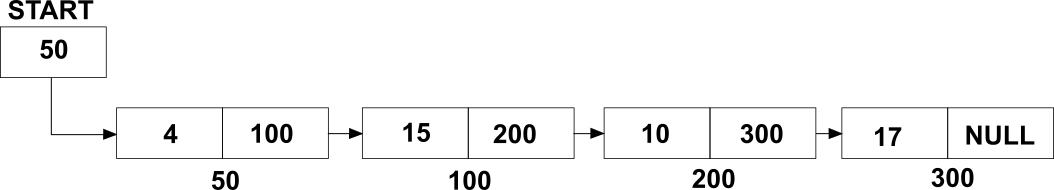
FUNCTIONS

(Design / Approach of Salient functions implemented in the program)

**Small()**

This function is designed to determine the Largest and the Smallest element in the Linked List. The smallest element is henceforth moved at the end of the Linked List by rearranging the Links.

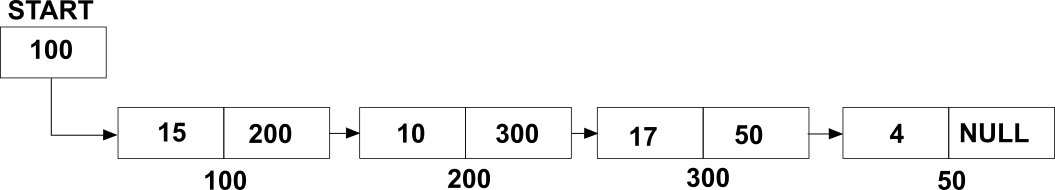
Before



Largest: 17

Smallest: 4

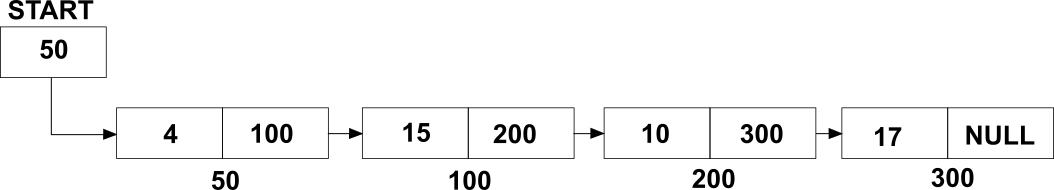
After



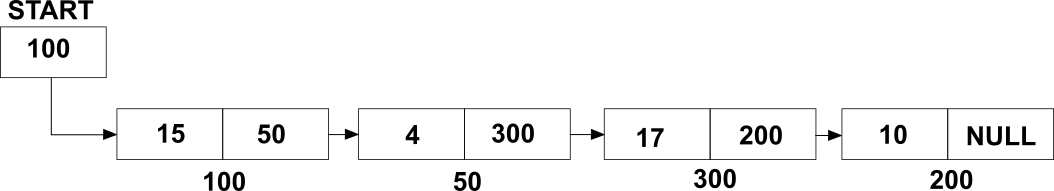
**Swap()**

This function is designed to manipulate the links of an existing Linked List by swapping the subsequent Node with the previous node.

Before



After

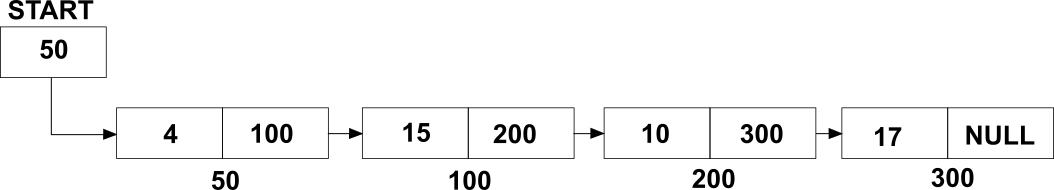


**Split()**

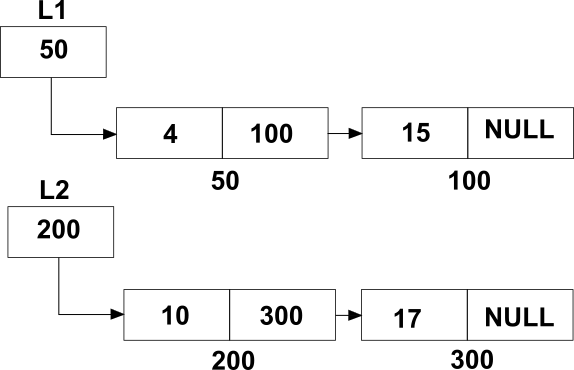
The split function breaks apart the linked list into 2 halves based on the Roll Number

The Starting address of the Linked List 1 is stored in a pointer S1 and the other Staring address is stored in pointer S2.

Before



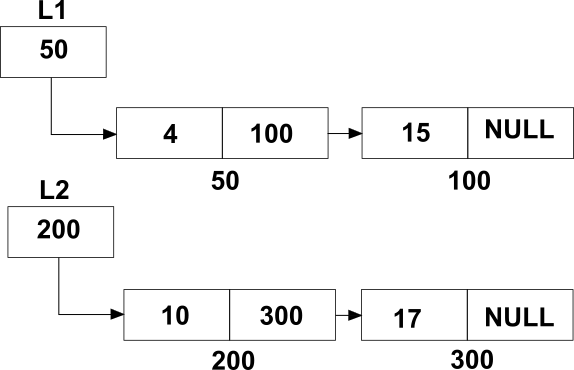
After



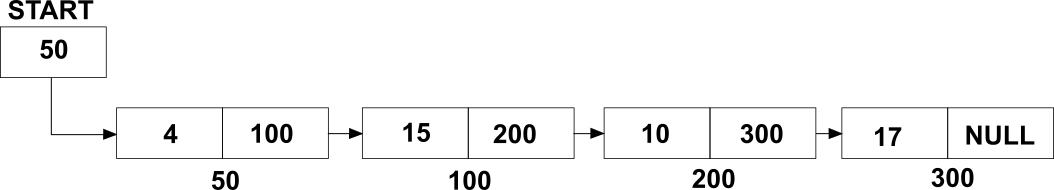
**Merge()**

This function is defined to merge 2 separate Linked lists. The link section of the last node of the first Linked List hold the starting address of the 2nd Linked list. Hence both the Linked Lists are concatenated.

Before



After

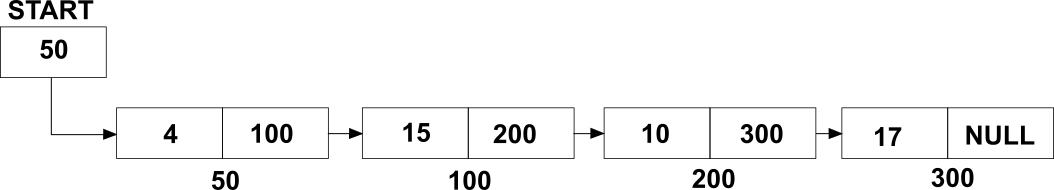


**Sort()**

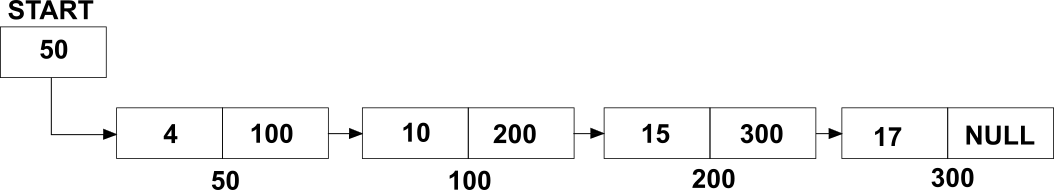
This function is used to sort the elements in a proper increasing order.

The sorting is done based on the Roll Numbers

Unsorted



Sorted



ALGORITHMS AND FLOWCHARTS

**struct node \*addEND(struct node \*start)**

1. Declare \*temp,\*p

2. Declare and initialize I, sum=0

3. p=start

4. Create a node (Using malloc)

5. Input temp->name

6. Input temp->rollNo

7. for i=0,1,…3

1 .input temp->marks[i]

2. sum=sum + temp->marks[i]

8. temp->avg=sum/3

9. while p->link != NULL

1.p=p->link

10. p->link=temp

11. temp->link=NULL

12. return start

**struct node \*createList(struct node \*start)**

1. Declare and initialize I, n, sum=0

2. Input N

3. start=NULL

4. if n == 0

1. return start

5. Declare \*temp

6. Create a Node (Using malloc)

7. Input temp->name

8. Input temp->rollNo

9. for i=0,1,…3

1 .input temp->marks[i]

2. sum=sum + temp->marks[i]

10. temp->avg=sum/3

11. temp->link=start

12. start=temp

13. for i=2,3,4,….n

Start=addEND(start)

14. return start

**struct node \*split(struct node \*start)**

1. Declare int i

2. Input r (roll Number)

3. Declare \*p1, \*p2

4. if start->rollNo ==r

1. print “List cannot be splited”

2. return start

5. p1=start

6. while p!= NULL

1. if p1 ->rollNo == r

1. p2=p1->link

2. p1->link=NULL

3. break

2. p1=p1->link

6. print “List 1”

7. display(start)

8. s1= start //s1,s2 global pointers

9. print “List 2”

10. display(p2)

11. s2=p2

**void display(struct node \*start)**

1. Declare \*p

2. Declare int i

3. if start == NULL

1. print “List Empty”

2. return

4. p=start

5. while p!= NULL

1. print p->name

2. print p->rollNo

3. for i=0,1,…3

1. print p->marks[i]

4. print p->avg

5.p=p->link

**struct node \*LARGE(struct node \*start)**

1. Declare \*temp, \*temp2, \*p

2. Declare and initialize MIN = start->avg

3. Declare and initialize Max=start->avg

4. while temp!=NULL

1. if temp->avg < MIN

1. MIN=temp->avg

2. p=temp

2.tep=temp->link

5. print MIN

6. temp=start->link

7. while temp!=NULL

1. if temp->avg >Max

1. Max=temp->avg

2. p=temp

2.tep=temp->link

8. print Max

9. temp=start

10. if p==start

1. start=start->link

2. while temp->link != NULL

1. temp=temp->link

3. temp->link=p

4. p->link = NULL

5. return start

Else if p!= start

1. temp2=start

2. while temp2->link != NULL

3. if temp2->link->avg== MIN

1.temp->link=p->link

2. temp2=temp2->link

4. temp2->link=p

5. p->link = NULL

6. return start

**void print(struct node \*start)**

1. Declare Int i.

2. Print start->name

3. Print start->rollNo

4. Print “Marks of 3 Subjects”

5. for i=0,1,…3

1. Print start->marks[i]

6. Print start->avg

**struct node \*merge()**

1. Declare node \*a, \*b

2. a=s1, b=s2

3. while a->link != NULL

1. a=a->link

4. print “Before Output”

5. print “List 1”

6. display(s1)

7. print “List 2:”

8. display(s2)

9. a->link=b

10. print “After Merge”

11. return(s1)

**struct node \*swap(struct node \*start)**

1. Declare and initialize \*prev= NULL, \*next,

\*cur=start

2. if start!= NULL && start->link=NULL

Return start

3. While cur != NULL && cur->link != NULL

1. next= cur->link

2. cur->link=next->link

3. next->link=cur

4. If prev != NULL

1, prev->link=next

5. prev=cur

6.prev=cur

7. cur=cur->link

4. display(start)

5. return start

**struct node \*sort(struct node \*start)**

1. Declare \*t, \*s

2. declare int x

3. t=start

4. while t != NULL

1. s=t->link

2.while s!= NULL

1. if t->rollNo > s->rollNo

x=t->rollNo

t->rollNo = s->rollNo

s->rollNo = x

2.s=s->link

3. t=t->link

5. return start

**SOURCE CODE:**

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct node{

int rollNo;

char name[20];

int marks[3];

int total;

float avg;

struct node \*link;

}\*s1,\*s2;

void print(struct node \*start)

{

int i;

printf("\nNAME: %s\n",start->name);

printf("ROLL NUMBER: %d\n",start->rollNo);

printf("MARKS OF 3 SUBJECTS: ");

for(i=0;i<3;i++)

printf("%d ",start->marks[i]);

printf("\nAVERAGE = %f",start->avg);

printf("\n");

}

struct node \*addEND(struct node \*start)

{

struct node \*temp,\*p;

int i;

float sum=0;

p=start;

temp=(struct node\*)malloc(sizeof(struct node));

printf("ENTER NAME: ");

scanf("%s",&temp->name);

printf("ENTER THE ROLL NUMBER: ");

scanf("%d",&temp->rollNo);

printf("ENTER MARKS OF 3 SUBJECTS:\n");

for(i=0;i<3;i++)

{

scanf("%d",&temp->marks[i]);

sum+=temp->marks[i];

}

temp->avg=sum/3;

while(p->link!=NULL)

p=p->link;

p->link=temp;

temp->link=NULL;

return start;

}

struct node \*createList(struct node \*start)

{

int i,n,sum=0;

printf("ENTER NUMBER OF STUDENTS\n");

scanf("%d",&n);

start=NULL;

if(n==0)

return start;

struct node \*temp;

temp=(struct node\*)malloc(sizeof(struct node));

printf("ENTER NAME: ");

scanf("%s",&temp->name);

printf("ENTER THE ROLL NUMBER: ");

scanf("%d",&temp->rollNo);

printf("ENTER MARKS OF 3 SUBJECTS:\n");

for(i=0;i<3;i++)

{

scanf("%d",&temp->marks[i]);

sum+=temp->marks[i];

}

temp->avg=sum/3;

temp->link=start;

start=temp;

for(i=2;i<=n;i++)

{

start=addEND(start);

}

return start;

}

void display(struct node \*start)

{

struct node \*p;

int i;

if(start==NULL)

{

printf("LIST EMPTY\n");

return;

}

p=start;

while(p!=NULL)

{

printf("\nNAME: %s\n",p->name);

printf("ROLL NUMBER: %d\n",p->rollNo);

printf("MARKS OF 3 SUBJECTS: ");

for(i=0;i<3;i++)

{

printf("%d ",p->marks[i]);

}

printf("\nAVERAGE: %f\n",p->avg);

p=p->link;

}

}

struct node \*LARGE(struct node \*start)

{

struct node \*temp,\*temp2,\*p,\*x;

float MIN=start->avg;

temp=start->link;

float Max=start->avg;

while(temp!=NULL)

{

if((temp->avg)<MIN)

{

MIN=temp->avg;

p=temp;

}

temp=temp->link;

}

printf("SMALLEST: %f\n",MIN);

temp=start->link;

while(temp!=NULL)

{

if((temp->avg)>Max)

{

Max=temp->avg;

}

temp=temp->link;

}

printf("LARGEST: %f\n",Max);

temp=start;

if(p==start)

{

start=start->link;

while(temp->link!=NULL)

{

temp=temp->link;

}

temp->link=p;

p->link=NULL;

return start;

}

else if(p!=start)

{

temp2=start;

while(temp2->link!=NULL)

{

if(temp2->link->avg==MIN)

temp2->link=p->link;

temp2=temp2->link;

}

temp2->link=p;

p->link=NULL;

return start;

}

}

struct node \*split(struct node \*start)

{

int r;

printf("ENTER THE ROLL NUMBER FROW WHERE YOU WANT TO SPLIT\n");

scanf("%d",&r);

struct node \*p1,\*p2;

if(start->rollNo==r)

{

printf("\nList cannot be splitted\n");

return start;;

}

p1=start;

while(p1!=NULL)

{

if(p1->rollNo==r)

{

p2=p1->link;

p1->link=NULL;

break;

}

p1=p1->link;

}

printf("\nLIST 1: \n");

display(start);

s1=start;

printf("\nLIST 2: \n");

display(p2);

s2=p2;

}

struct node \*swap(struct node \*start)

{

struct node \*prev=NULL, \*next, \*cur=start;

if(start!=NULL && start->link==NULL)

return start;

start=start->link;

while(cur!=NULL && cur->link!=NULL)

{

next=cur->link;

cur->link=next->link;

next->link=cur;

if(prev!=NULL)

prev->link=next;

prev=cur;

cur=cur->link;

}

display(start);

return(start);

}

struct node \*merge()

{

struct node \*a,\*b;

a=s1;

b=s2;

while(a->link!=NULL)

{

a=a->link;

}

printf("BEFORE MERGING: \n");

printf("LIST 1:\n");

display(s1);

printf("LIST 2:\n");

display(s2);

a->link=b;

printf("\n\nAFTER MERGING\n");

return(s1);

}

struct node \*sort(struct node \*start)

{

struct node \*t;

struct node \*s;

int x;

t=start;

while(t!=NULL)

{

s=t->link;

while(s!=NULL)

{

if(t->rollNo>s->rollNo)

{

x=t->rollNo;

t->rollNo=s->rollNo;

s->rollNo=x;

}

s=s->link;

}

t=t->link;

}

return start;

}

int main()

{

struct node \*start;

int choice,marks[3],i,rollNo;

char x[20];

do{

printf("\n\nSTUDENTS\n");

printf("CREATE STUDENT LIST........................1\n");

printf("DISPLAY CONTENT............................2\n");

printf("FIND THE LARGEST AND SMALLEST..............3\n");

printf("SPLIT INTO LIST............................4\n");

printf("SWAP CONTENTS..............................5\n");

printf("MERGE 2 LISTS..............................6\n");

printf("ENTER THE CHOICE: ");

scanf("%d",&choice);

switch(choice)

{

case 1:

start=createList(start);

break;

case 2:

display(start);

break;

case 3:

start=LARGE(start);

break;

case 4:

split(start);

break;

case 5:

start=swap(start);

printf("SWAPPED SUCCESSFULLY\n");

break;

case 6:

start=merge();

start=sort(start);

display(start);

break;

default:

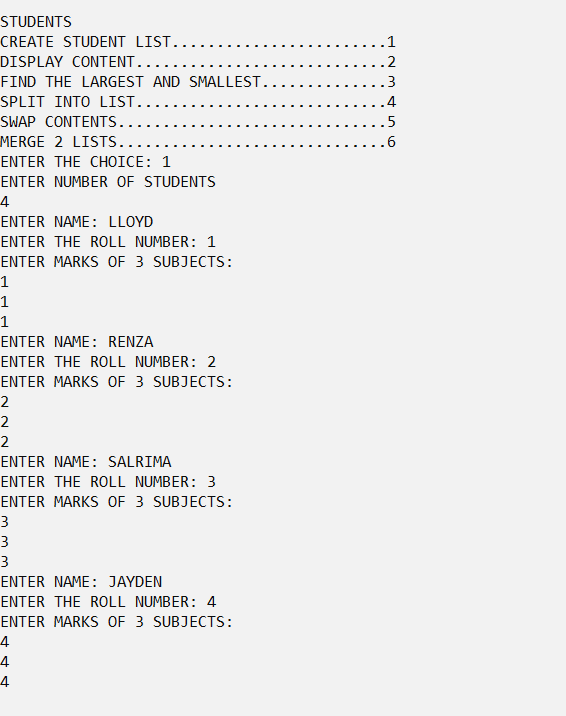
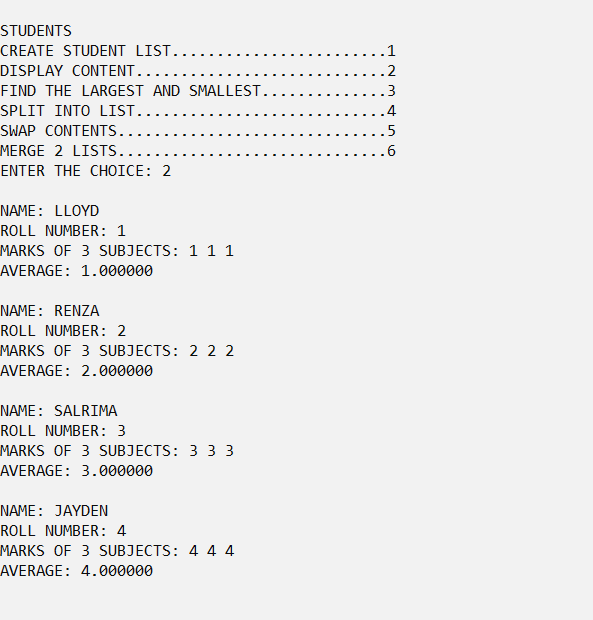
printf("INVALID CHOICE\n");

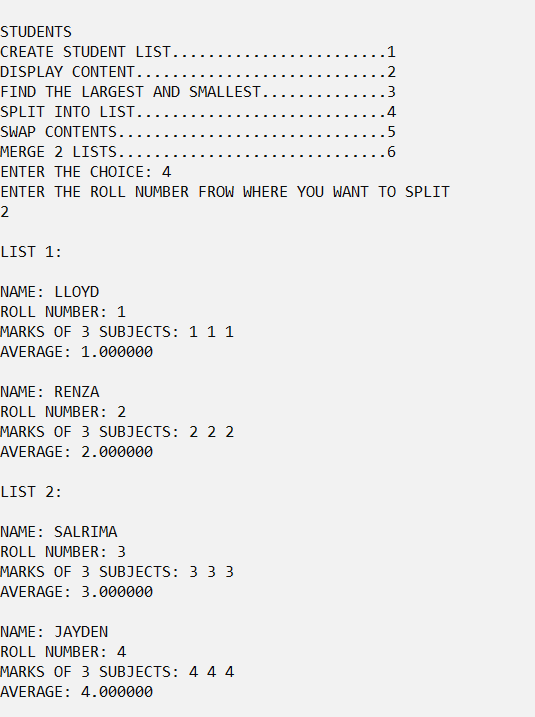
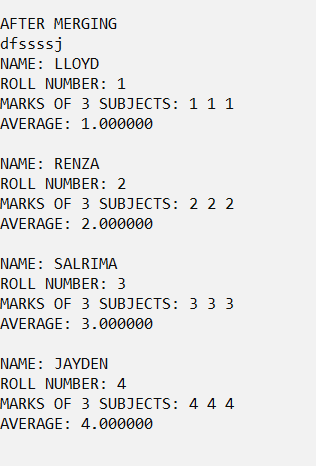
}

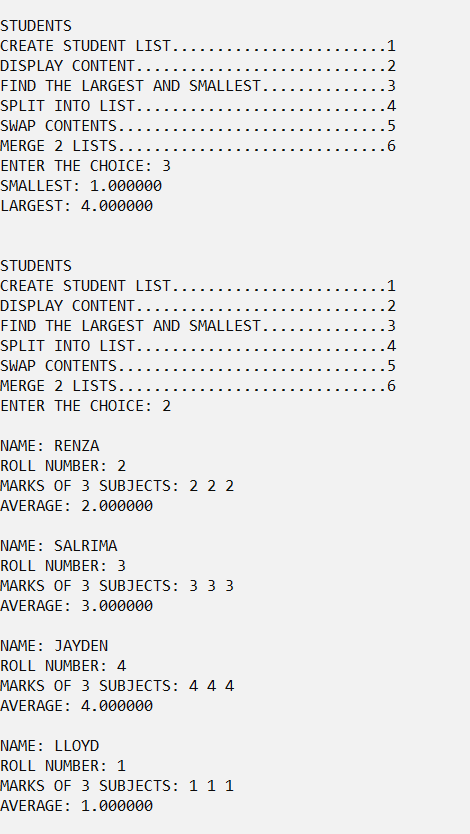
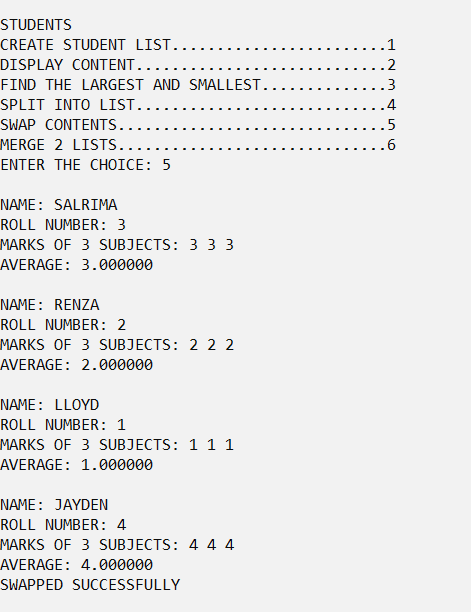
}while(1);

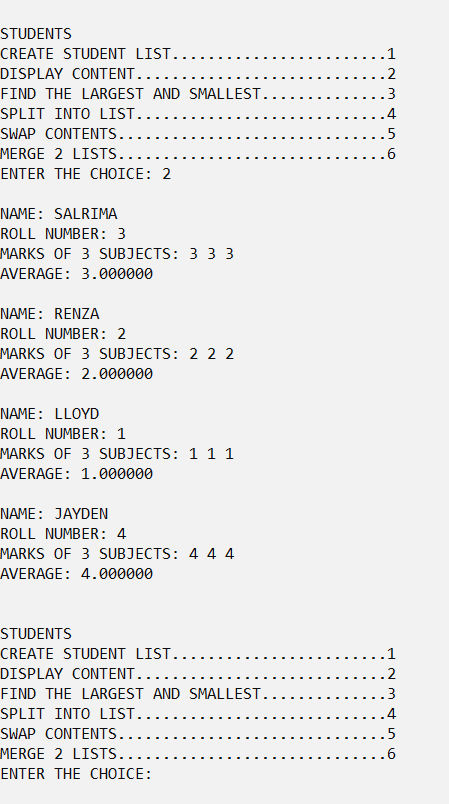
}

**OUTPUTS:**

****

****

****

****

**CONCLUSION:**

The Given Problem statement was successfully compiled and executed.

Learnings:

1) Concept of Singly Linked Lists.

2) Implementation of the same.

3) Its applications and uses.

.

Findings:

1) Use of Singly List can be Beneficial in many aspects as compared to Arrays.

2) Compilation Time: 0.8 secs