**Program 4**

**Aim :** To implement union and intersection of two sorted linked lists.

**Theory :** In computer science, a linked list is a linear collection of data elements, whose order is not given by their physical placement in memory. Instead, each element [points](https://en.wikipedia.org/wiki/Pointer_(computer_programming)) to the next. It is a [data structure](https://en.wikipedia.org/wiki/Data_structure) consisting of a collection of [nodes](https://en.wikipedia.org/wiki/Node_(computer_science)) which together represent a [sequence](https://en.wikipedia.org/wiki/Sequence). In its most basic form, each node contains: [data](https://en.wikipedia.org/wiki/Data_(computing)), and a [reference](https://en.wikipedia.org/wiki/Reference_(computer_science)) (in other words, a *link*) to the next node in the sequence. This structure allows for efficient insertion or removal of elements from any position in the sequence during iteration. More complex variants add additional links, allowing more efficient insertion or removal of nodes at arbitrary positions.

Union of two linked lists can be found by using merging the lists in a sorted manner.

Intersection of the two lists can be found by only taking common elements while merging the two lists.

It has been assumed that the linked lists are sorted.

**Algorithm :**

Union(L1,L2)

1. Declare node pointer output, outputTail as NULL
2. Repeat steps 3 to 9 while L1!=NULL AND L2!=NULL
3. Make a newNode and set its next = NULL
4. If L1->data < L2->data then

Set newNode->data = L1->data

Set L1 = L1->next

1. Else if L1->data > L2->data then

Set newNode->data = L2->data

L2 = L2->next

1. Else
2. Set Data = L1->data
3. Set newNode->data = Data
4. Repeat steps a) and b) while L1!=NULL AND L2!=NULL AND L1->data == Data AND L2->data == Data
5. Set L1 = L1->next
6. Set L2 = L2->next
7. If output == NULL then

Set Output = outputTail = newNode

1. Else

Set outputTail->next = newNode

Set outputTail = outputTail->next

1. Repeat steps 10 to 14 while L1!=NULL
2. Make a newNode
3. Set outputTail->next = newNode
4. Set outputTail = outputTail->next
5. Set outputTail->data = L1->data
6. Set L1 = L1->next

Repeat steps 15 to 19 while L2!=NULL

1. Make a newNode
2. Set outputTail->next = newNode
3. Set outputTail = outputTail->next
4. Set outputTail->data = L2->data
5. Set L2 = L2->next

Return output

Intersection(L1,L2)

1. If L1 or L2 is NULL then return NULL
2. Declare node pointers output, outputTail as null.
3. Repeat steps 4 to 6 while L1!=NULL AND L2!=NULL
4. If L1->data<L2->data then

Set L1 = L1->next

1. Else If L2->data<L1->data then

Set L2 = L2->next

1. Else
2. Declare and set data = L1->data
3. Make a newNode
4. Set newNode->data = data and newNode->next = NULL
5. If output == null then
6. Set output = outputTail = newNode
7. Else
8. Set outputTail->next = newNode
9. Set outputTail = outputTail->next
10. Repeat steps i and ii while L1!=NULL AND L2!=NULL AND L1->data == data AND L2->data == data
11. Set L1 = L1->next
12. Set L2 = L2->next

**Program :**

#include <stdio.h>

#include <stdlib.h>

struct node{

struct node\*next;

int data;

};

struct node \* Union(struct node \* L1, struct node \* L2){

struct node \* output = NULL;

struct node \* outTail = NULL;

while(L1&&L2){

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

newNode->next = NULL;

if(L1->data<L2->data){

newNode->data = L1->data;

L1 = L1->next;

}

else if(L1->data>L2->data){

newNode->data = L2->data;

L2 = L2->next;

}

else{

int data = L1->data;

newNode->data = data;

while(L1 && L2 && L1->data == data && L2->data == data){

L1 = L1->next;

L2 = L2->next;

}

}

if(!output)

output = outTail = newNode;

else{

outTail->next = newNode;

outTail = outTail->next;

}

}

while(L1){

outTail->next = (struct node \*) malloc(sizeof(struct node));

outTail = outTail->next;

outTail->data = L1->data;

L1 = L1->next;

}

while(L2){

outTail->next = (struct node \*) malloc(sizeof(struct node));

outTail = outTail->next;

outTail->data = L2->data;

L2 = L2->next;

}

outTail->next = NULL;

return output;

}

struct node \* intersection(struct node \* L1, struct node\* L2){

if(L1 == NULL || L2 == NULL)

return NULL;

struct node \* output = NULL;

struct node \* outTail = NULL;

while(L1&&L2){

if(L1->data<L2->data){

L1 = L1->next;

}

else if(L2->data<L1->data){

L2 = L2->next;

}

else{

int data = L1->data;

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

newNode->data = data;

newNode->next = NULL;

if(output == NULL){

outTail = output = newNode;

}

else{

outTail->next = newNode;

outTail = outTail->next;

}

while(L1 && L2 && L1->data == data && L2->data == data){

L1 = L1->next;

L2 = L2->next;

}

}

}

return output;

}

struct node \* createList(int listNum){

struct node \* list = NULL;

struct node \* list\_tail = NULL;

printf("Enter elements of List %d in increasing order\n",listNum);

char ch = 'y';

do{

int data;

printf("Enter element : ");

scanf("%d",&data);

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

newNode->data = data;

newNode->next = NULL;

if(list == NULL){

list = list\_tail = newNode;

}

else{

list\_tail->next = newNode;

list\_tail = list\_tail->next;

}

printf("Would you like to insert another element [Y/N] : ");

scanf(" %c",&ch);

}while(ch == 'y' || ch == 'Y');

return list;

}

void print(struct node \* list){

if(list == NULL){

printf("Empty List\n");

return;

}

while(list!=NULL){

printf("%d ",list->data);

list = list->next;

}

printf("\n");

}

int main() {

struct node \* L1 = NULL;

struct node \* L2 = NULL;

struct node \* L3 = NULL;

struct node \* L4 = NULL;

L1 = createList(1);

L2 = createList(2);

printf("List 1 : ");

print(L1);

printf("List 2 : ");

print(L2);

printf("Union : ");

L3 = Union(L1, L2);

print(L3);

printf("Intersection : ");

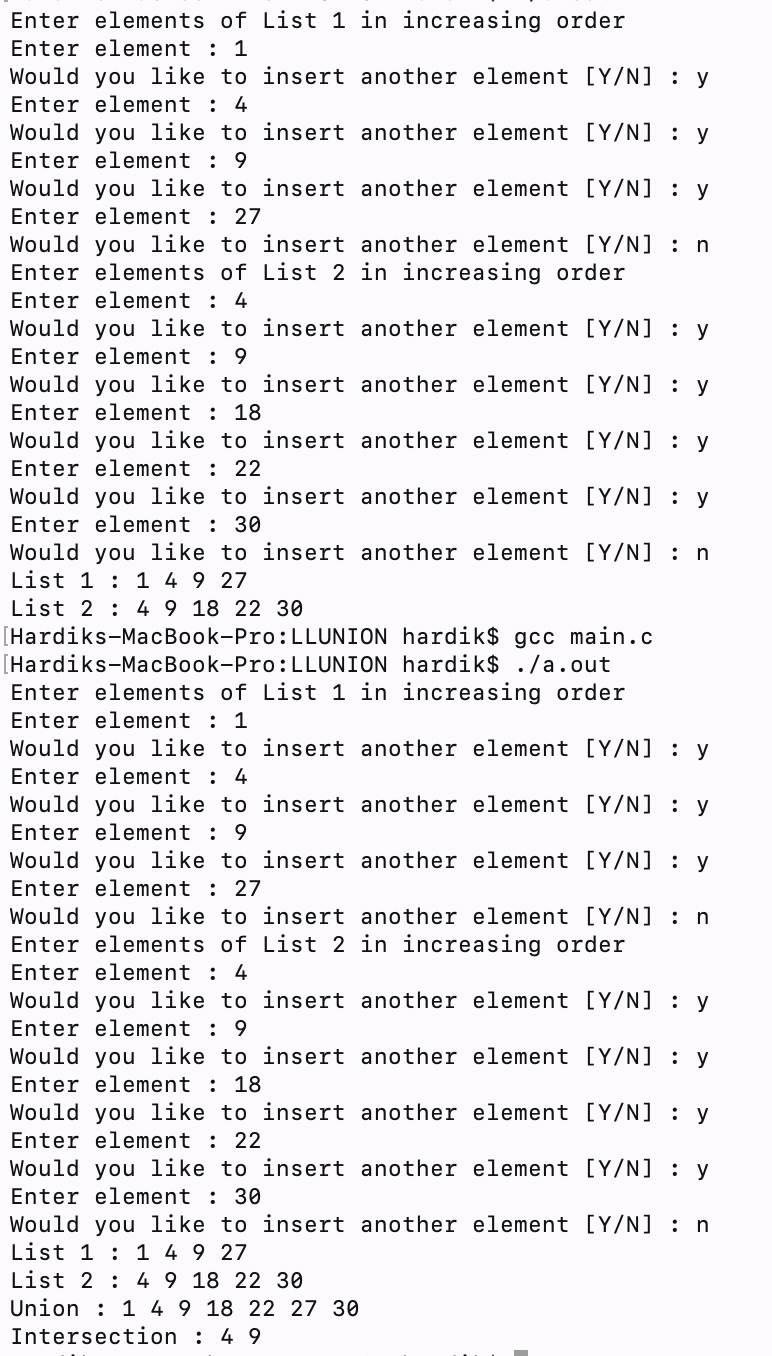
L4 = intersection(L1, L2);

print(L4);

return 0;

}

**OUTPUT :**



**Learning :** We learnt how to implement union and intersection on two sorted linked lists.