**Batch: A1 Roll No.: 1911004**

**Experiment / assignment / tutorial No. 2**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title:**  Implementation of Advanced operations on linked list-Concatenate two lists, Reverse a list, Intersection of two lists. |

**Objective:** To Understand and implement Advanced operations on linked list-Concatenate two lists, Reverse a list, Intersection of two lists.

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| 1 | Explain the different data structures used in problem solving |

**Books/ Journals/ Websites referred:**

1. *Fundamentals Of Data Structures In C –* Ellis Horowitz, Satraj Sahni, Susan Anderson-Fred
2. *An Introduction to data structures with applications –* Jean Paul Tremblay, Paul G. Sorenson
3. *Data Structures A Pseudo Approach with C –* Richard F. Gilberg & Behrouz A. Forouzan
4. <https://www.geeksforgeeks.org/data-structures/linked-list/>
5. <https://www.thecrazyprogrammer.com/2014/02/concatenation-of-two-linked-lists.html>

**Abstract**:

Linked lists are special lists of data elements linked to one another. The logical ordering is represented by having each element pointing to the next element. Each element is called a node, which has two parts, INFO part which stores the information and pointer which points to the next element.

The Last node’s Address field contains Null rather than a valid address. It’s a NULL Pointer and indicates the end of the list.

The linear order in Linked Lists is given by means of pointers

**Related Theory: -**

**Implement a menu-driven program for**

**concatenation- Accept two lists and concatenate them together. You may ask which list out of two goes first and which one goes later.**

**Reversal- Accept just one list and reverse the contents**

**intersection -Accept two lists as input and insert the common characters in the third list. The third list would be the output of program.**

1. **Concatenate two lists-**

**For this we implement a function concat(). We would need 2 lists for this function. We ask user for the 2 lists. Now we go through both lists to obtain our output List3 . We traverse through List1 & List2 , by asking user which list should be concatenated in List3.**

**Thus we get our output List3 . Since we have to traverse through both lists ;**

**n = no of elements in List1 ;**

**m = no of elements in List2 ;**

**for List3 no of elements would be n + m & this will take time complexity of O (n + m)**

1. **Reverse a list-**

**For this we implement a function reverse() . We would need 1 list for reversal of this list. We store the Reversed of List - 1 in output List – 2 . We ask user to enter the list . We traverse through the list and have the temporary address node pointer to store the data & address thus we store the node in reverse way using this method. Thus if**

**n = no of elements in List1 ;**

**n = would be no of elements in List1=2 ;**

**then it takes O ( n ) time complexity to solve this.**

1. **Intersection of two lists-**

**For this we implement a function intersection(). We would need 2 lists for this function. We ask user for the 2 lists. Now we go through both lists to obtain our output List3 . We traverse through List1 & List2 , to find out common elements both lists thus adding common to List3.**

**Thus we get our output List3 . Since we have to traverse through both lists ;**

**n = no of elements in List1 ;**

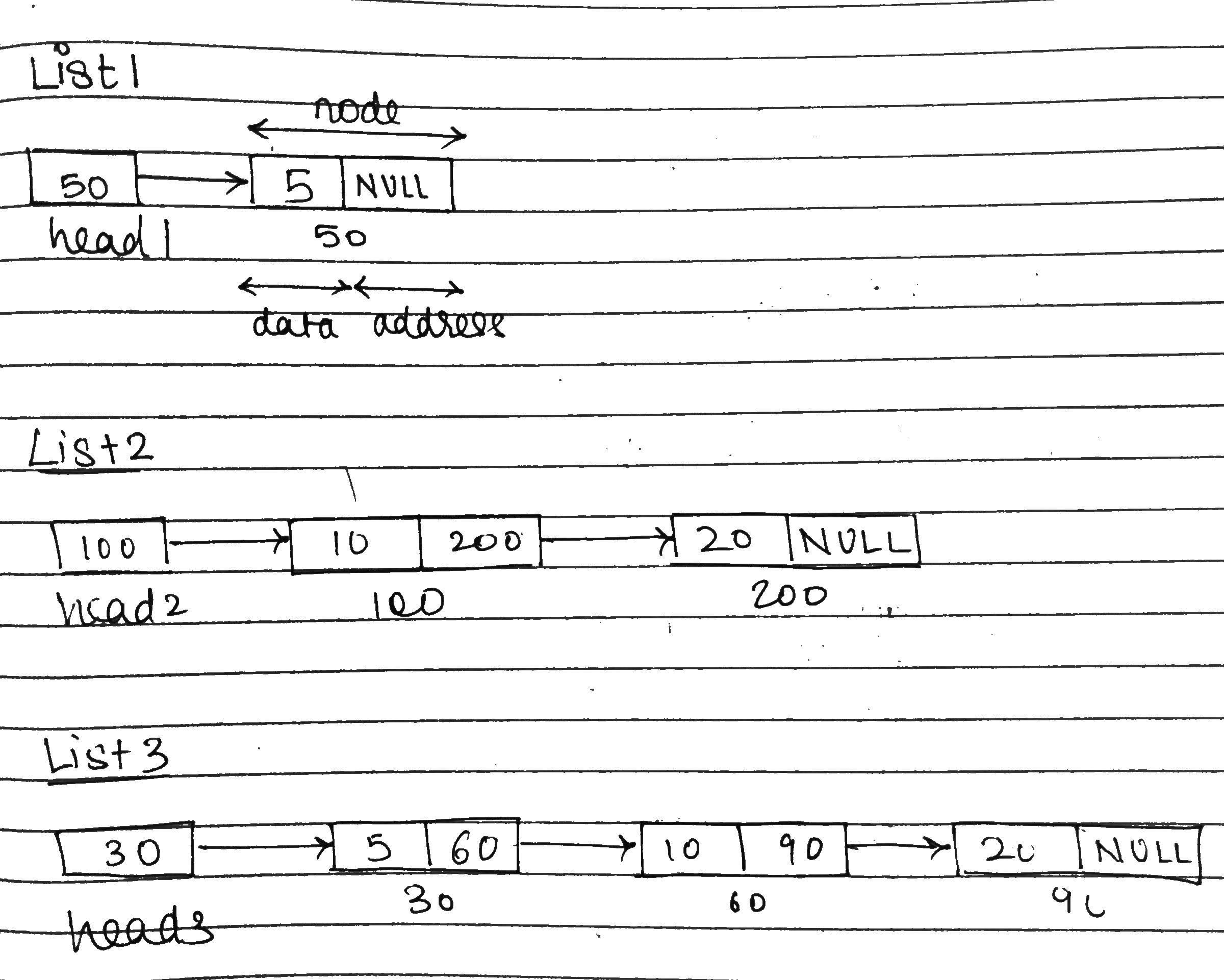
**m = no of elements in List2 ;**

**for List3 no of elements would be n ^ m & this will take time complexity of O (n \* m) as we have to find the common elements of both**

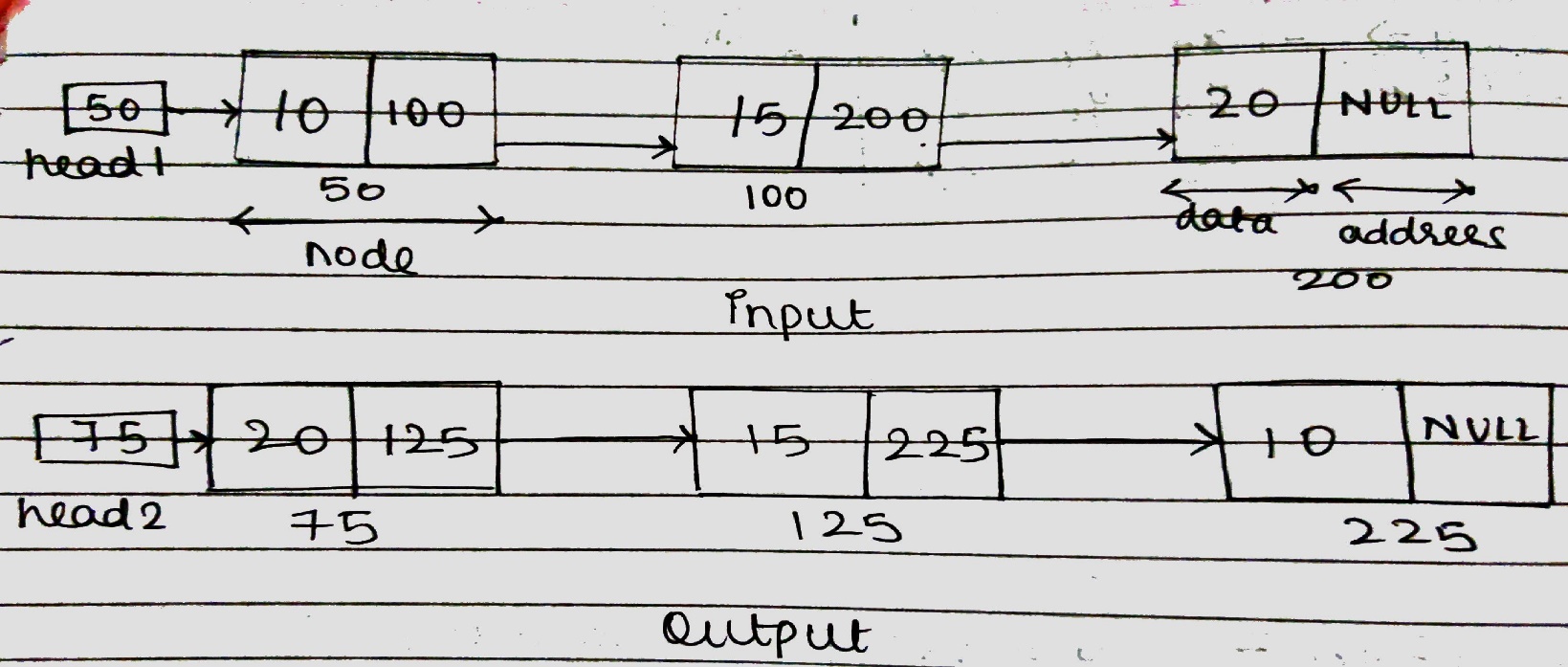
**^ = no of common elements**

**Diagram for each Advanced Operation:**

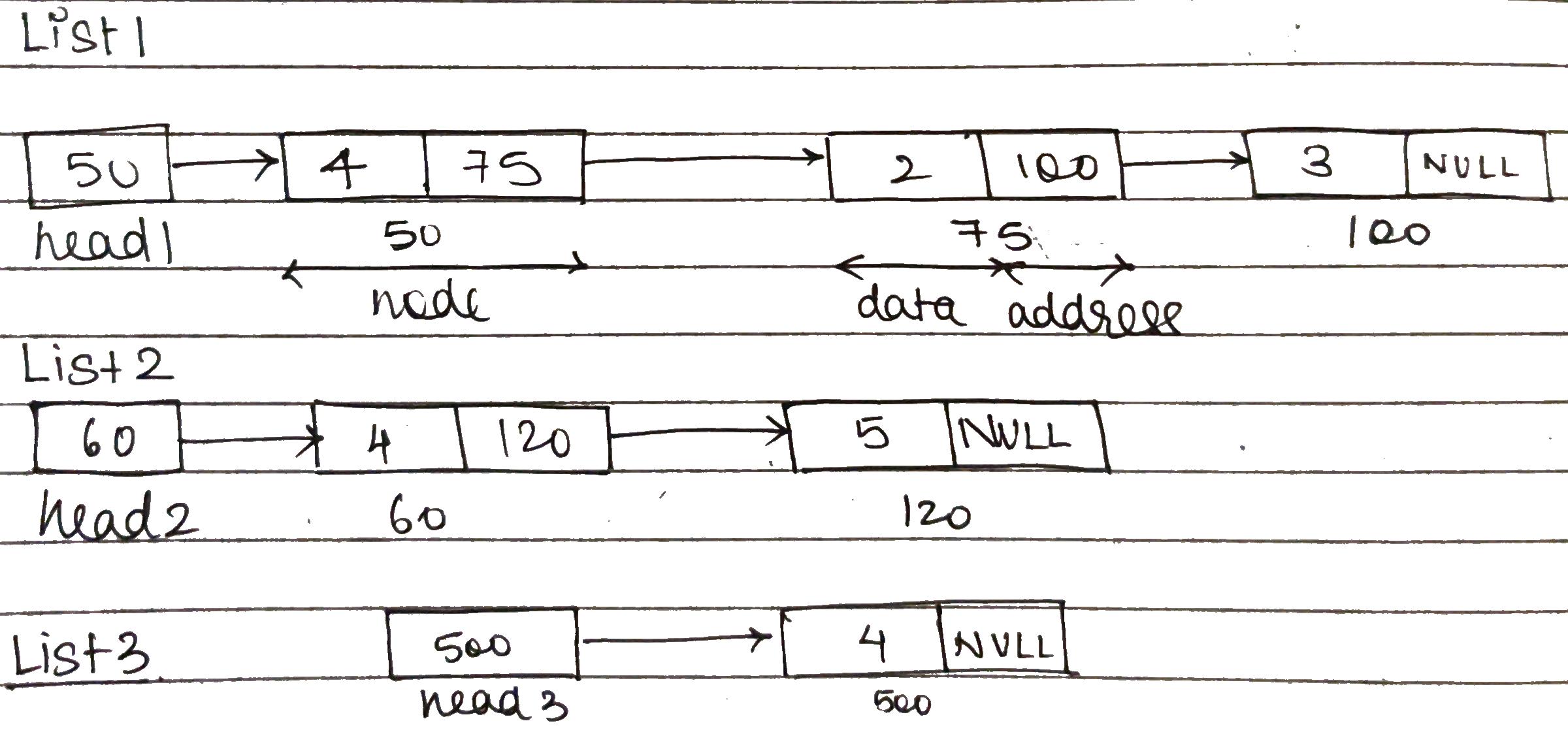
1. **Concatenate two lists-**



1. **Reverse a list-**



1. **Intersection of two lists-**



**Algorithm for Concatenation of two lists, Reversing a list, Intersection of two lists:**

1. **Start**
2. **Make struct node{**

**int data;**

**struct node \*next;**

**};**

1. **in struct node\* createlist()**

* **we create our link list by running loop n times where**
* **newN=(struct node\*)malloc(sizeof(struct node)); //for new node**
* **we ask user to enter the data in data part of newN , then we assign head pointer the address of 1st node , then 1st node has address of 2nd & so on.**

1. **in print we print the link list till last node is not reached by**

**t->address !=0**

**in void concat(struct node \*p,struct node \*q) we take in 2 list heads and concatenate the lists in a new lists by time complexity = O( n + m )**

**struct node \*ptr = p;**

* **we first add in 1st list then 2nd list in 3rd list while(ptr->next!=0){**

**ptr=ptr->next; }**

**ptr->next = q; }**

1. **in void reverse(struct node\*\* p){**

* **struct node\* prev = 0,\*current = \*p,next = 0;**

**here we manipulate the pattern in which node are connected in while loop using this time complexity = O( n )**

* **while (current != 0) {**

**next = current->next;**

**current->next = prev;**

**prev = current;**

**current = next;**

**}\*p = prev;}**

1. **in struct node\* intersection(struct node \*p, struct node \*q){**

**struct node \*head = 0, \*newN,\*ptr,\*q1;**

* **we iterate & traverse trough both the list**
* **but in a nested way so time complexity = O( n \* m )**
* **in each iteration we check for equal elements & increment pointers & add common elements in list 3.**

**q1=q;**

**while(p!=0){**

**q=q1;**

**while(q!=0){**

**if(q->data == p->data){**

**head = addnode(head,p->data);**

**}**

**q=q->next;**

**}**

**p=p->next;**

**} return head; }**

1. **End**

**Implementation Details:**

1. **Enlist all the Steps followed and various options explored.**

**We implemented code using menu driven approach . We give option of concatenation , reversal , intersection of lists ,exit . User enter the choice & till exit is not told do …while loop runs & we perform user’s choice .**

**Respective messages are shown on completion of operation for every selected choice. In each case the function of the respective operations are invoked & at the end of the case, a break statement is executed.**

* **Concatenate of 2 lists // calls concat()**
* **Reversal of list // calls reverse()**
* **Intersection of 2 lists // calls intersect()**

1. **Assumptions made for Input:**

**We assume that data taken as input would be integer type in concat() ,intersection() , reversal() on various linked - lists.**

1. **Built-In Functions Used:**

* **malloc()**

**Program source code for Concatenation of two lists, Reversing a list, Intersection of two lists. :**

**#include <stdio.h>**

**struct node{**

**int data;**

**struct node \*next;**

**};**

**struct node\* createlist(){**

**int n,i;**

**struct node \*first=0,\*newN,\*t;**

**printf("Enter the no of elements to be inserted = ");**

**scanf("%d",&n);**

**for(i=0;i<=n-1;i++){**

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

**printf("Enter the data = ");**

**scanf("%d",&newN->data);**

**newN->next=0;**

**if(first==0)**

**{**

**first=newN;**

**t=newN;**

**}**

**else**

**{**

**t->next=newN;**

**t=t->next;**

**}**

**}**

**return first;**

**}**

**struct node\* addnode(struct node\* head, int a){**

**struct node \*newN,\*ptr;**

**if(head == 0){**

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

**newN->data = a;**

**newN->next = 0;**

**head = newN;**

**}else{**

**ptr = head;**

**while(ptr->next!=0){**

**ptr=ptr->next;**

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

**newN->data = a;**

**newN->next = 0;**

**ptr->next = newN;**

**}**

**}**

**return head;**

**}**

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

**{**

**while(p!=0)**

**{**

**printf("%d->",p->data);**

**p=p->next;**

**}**

**printf("NULL");**

**}**

**void concat(struct node \*p,struct node \*q){**

**struct node \*ptr = 0;**

**ptr = p;**

**while(ptr->next!=0){**

**ptr=ptr->next;**

**}**

**ptr->next = q;**

**}**

**void reverse(struct node\*\* p)**

**{**

**struct node\* prev = 0,\* current = \*p,\* next = 0;**

**while (current != 0) {**

**next = current->next;**

**current->next = prev;**

**prev = current;**

**current = next;**

**}**

**\*p = prev;**

**}**

**struct node\* intersection(struct node \*p,struct node \*q){**

**struct node \*head = 0, \*newN,\*ptr,\*q1;**

**q1=q;**

**while(p!=0){**

**q=q1;**

**while(q!=0){**

**if(q->data == p->data){**

**head = addnode(head,p->data);**

**}**

**q=q->next;**

**}**

**p=p->next;**

**}**

**return head;**

**}**

**int main(void) {**

**struct node \*p=0;**

**struct node \*q=0;**

**struct node \*result=0;**

**int ch=-1;**

**do{**

**printf("1. Concatenation Of 2 Lists\n");**

**printf("2. Reversal of List\n");**

**printf("3. intersection of Common elements of 2 Lists\n");**

**printf("0. EXIT\n");**

**printf("\tEnter your choice \n");**

**scanf("%d",&ch);**

**switch(ch)**

**{**

**case 1:**

**p=createlist();**

**print(p);**

**printf("\n");**

**q=createlist();**

**print(q);**

**printf("\n");**

**printf("Concatenated List of 2 list\n\t");**

**concat(p,q);**

**print(p);**

**printf("\n\n");**

**break;**

**case 2:**

**printf("\nReversed List \n\t");**

**reverse(&p);**

**print(p);**

**printf("\n\n");**

**break;**

**case 3:**

**printf("Intersected List of 2 list \n\t");**

**result = intersection(p,q);**

**print(result);**

**printf("\n\n");**

**break;**

**case 0:**

**printf("EXITING ...");**

**break;**

**}**

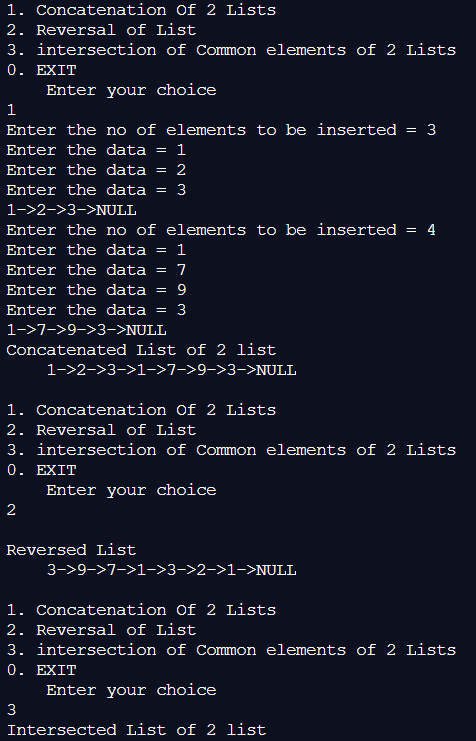
**}while(ch!=0);**

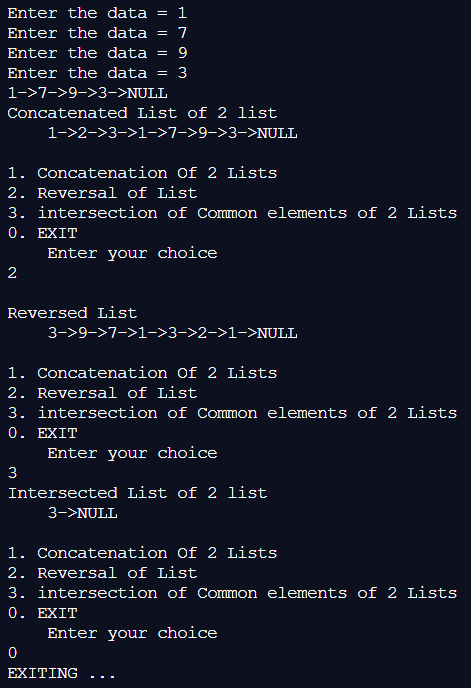
**printf("\n\n\n");**

**return 0;**

}

**Output:**





**Explain the Importance of the approach followed :**

* **We learnt creation of link lists**
* **We learnt about reversing it**
* **Concatenating 2 different link – list with different length**
* **Intersection of common elements in both lists**
* **Importance of Link -list**
* **Dynamic Nature of it**

**Conclusion:-**

**We understood the concept of the Link-Lists & Implemented the logic for reversal, intersection, concatenation in C ; getting the necessary correct output.**

**Post-Lab Questions:**

1. **Illustrate 2 Applications of Linked Lists.**

* **Implementation Of Stack**
* **A stack contains Top /Head pointer.**
* **Top/Head of the stack where pushing and popping elements takes place.**
* **1st first node will have null in address part**
* **2nd node address part have 1st node address & so on**
* **nth node address is in Top/Head pointer.**
* **Dynamic Memory Allocation**
* **We can create as many node as we want to work on .**
* **We can in middle delete unwanted nodes too**
* **we can insert & merge new node into Link-list too**

1. **Compare and Contrast between Arrays and Linked Lists.**

|  |  |
| --- | --- |
| **ARRAYS** | **LINKED - LIST** |
| **Static in size** | **Dynamic in Size** |
| **Binary Search is easy to perform** | **Binary Search is difficult to implement** |
| **Less memory needed for 1 element**  **i.e. int a[5]**  **a[0] is of 2-bytes** | **More memory needed to store 1 element node struct node{**  **int data; struct node\* next;**  **}** |
| **To get last element direct access possible** | **To get last element traversal through entire list is need** |