1. **Basic Operations :**

* Defined a basic linked list structure:

A linked list is a linear data structure, in which the elements are not stored at contiguous memory locations. The elements in a linked list are linked using pointers

* Initialize an empty list

struct Node {

 int data;

    struct Node\* next;

};

int main()

{

     struct Node\* head = NULL;

/\* linked list is empty, Becouse the value of the head is NULL.

* Destroy a list node by node

1. **Insertion and deletion operation :**

* Create and insert a new node at the start of the list :

**struct** LL{

**int** data;

LL \***next**;

};

**void** insertAtBeginning(LL\*\*head,**int** dataToBeInserted)

{

LL\*curr=**new** LL;

//make a new node with this data and next pointing to NULL

curr->data=dataToBeInserted;

curr->**next**=NULL;

**if**(\*head==NULL) //if list is empty then set the current formed node as head of list

\*head=curr;

**else** //make the next of the current node point to the present head and make the current node as the new head

{

curr->**next**=\*head;

\*head=curr;

}

}

* Create and insert a new node at the end of the list :

// A linked list node

struct Node

{

  int data;

  struct Node \*next;

};

/\* Given a reference (pointer to pointer) to the head

   of a list and an int, appends a new node at the end  \*/

void append(struct Node\*\* head\_ref, int new\_data)

{

    /\* 1. allocate node \*/

    struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

    struct Node \*last = \*head\_ref;  /\* used in step 5\*/

    /\* 2. put in the data  \*/

    new\_node->data  = new\_data;

    /\* 3. This new node is going to be the last node, so make next

          of it as NULL\*/

    new\_node->next = NULL;

    /\* 4. If the Linked List is empty, then make the new node as head \*/

    if (\*head\_ref == NULL)

    {

       \*head\_ref = new\_node;

       return;

    }

    /\* 5. Else traverse till the last node \*/

    while (last->next != NULL) {

        last = last->next;

}

    /\* 6. Change the next of last node \*/

    last->next = new\_node;

    return;

}

* Create and insert a new node at position k in the list

struct Node

{

  int data;

  struct Node \*next;

};

/\* Given a reference (pointer to pointer) to the head of a list and

 an int, inserts a new node on the front of the list. \*/

void push(struct Node\*\* head\_ref, int new\_data)

{

    /\* 1. allocate node \*/

    struct Node\* new\_node = (struct Node\*) malloc(sizeof(struct Node));

    /\* 2. put in the data  \*/

    new\_node->data  = new\_data;

    /\* 3. Make next of new node as head \*/

    new\_node->next = (\*head\_ref);

    /\* 4. move the head to point to the new node \*/

    (\*head\_ref)    = new\_node;

}

/\* Given a node prev\_node, insert a new node after the given

   prev\_node \*/

void insertAfter(struct Node\* prev\_node, int new\_data)

{

    /\*1. check if the given prev\_node is NULL \*/

    if (prev\_node == NULL)

    {

      printf("the given previous node cannot be NULL");

      return;

    }

    /\* 2. allocate new node \*/

    struct Node\* new\_node =(struct Node\*) malloc(sizeof(struct Node));

    /\* 3. put in the data  \*/

    new\_node->data  = new\_data;

    /\* 4. Make next of new node as next of prev\_node \*/

    new\_node->next = prev\_node->next;

    /\* 5. move the next of prev\_node as new\_node \*/

    prev\_node->next = new\_node;

}

1. **Input and output operation :**

* Given a 1D array of element , create a linked list from the array (one new node per element , adding the node to the end of the list each time )

// Representation of a node

struct Node {

int data;

Node\* next;

};

// Function to insert node

void insert(Node\*\* head, int item)

{

Node\* NewNode;

Node\* last;

temp->data = item;

temp->next = NULL;

if (\*head == NULL)

\*head = NewNode;

else {

last = \*head;

while (last ->next != NULL){

} last = last ->next;

last ->next = NewNode;

}

}

void display(Node\* head)

{

while (head != NULL) {

cout << head->data << " ";

head = head->next;

}

}

Node \*arrayToList(int arr[], int n)

{

Node \* head = NULL;

for (int i = 0; i < n; i++)

insert(&head, arr[i]);

return head;

}

// Driver code

int main()

{

int arr[] = { 1, 2, 3, 4, 5 };

int n = sizeof(arr) / sizeof(arr[0]);

Node\* head = arrayToList(arr, n);

display(head);

return 0;

}

* Given an unsorted array, create a sorted list by creating a new node for each array element, and inserting it into its appropriate position in the list.

struct node

{

int data;

struct node \*next;

};

void main()

{

int i;

int num ;

struct node \*first, \*nw, \*pre, \*new1, \*count;

clrscr();

printf("\n Enter the number of node you want to create: ");

scanf("%d", &num );

first->next = NULL;

nw = first;

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

{

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

nw = nw->next;

printf("\n Enter the node: %d: ", i+1);

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

nw->next = NULL;

}

new1 = first;

for( ; new1->next != NULL; new1 = new1->next)

{

for(count = new1->next; count != NULL; count = count->next)

{

if(new1->data > count->data)

{

int temp = new1->data;

new1->data = count->data;

count->data = temp;

}

}

}

nw = first->next;

printf("\n After sorting the Linked list is as follows:\n");

while (nw)

{

printf("%d\t", nw->data);

nw = nw->next;

}

getch();

}

* Make a copy of a list into a new list.
* #include <stdio.h>  
  #include <conio.h>  
  #include <alloc.h>  
  /\* structure containing a data part and link part \*/  
  struct node  
  {  
  int data ;  
  struct node \*next ;  
  } ;
* void append ( struct node \*\*, int ) ;  
  void copy ( struct node \*, struct node \*\* ) ;  
  void display ( struct node \* ) ;
* void main( )  
  {  
  struct node \*first, \*second ;  
  first = second = NULL ; /\* empty linked lists \*/  
  append ( &first, 1 ) ;  
  append ( &first, 2 ) ;  
  append ( &first, 3 ) ;  
  append ( &first, 4 ) ;  
  append ( &first, 5 ) ;  
  append ( &first, 6 ) ;  
  append ( &first, 7 ) ;  
  clrscr( ) ;  
  display ( first ) ;  
  copy ( first, &second ) ;  
  display ( second ) ;  
  }  
  /\* adds a node at the end of the linked list \*/  
  void append ( struct node \*\*head, int num )  
  {  
  struct node \*temp ;  
  temp = \* head;  
  if ( \* head == NULL ) /\* if the list is empty, create first node \*/  
  {  
  \* head = malloc ( sizeof ( struct node ) ) ;  
  temp = \* head;  
  }  
  else  
  {  
  /\* go to last node \*/  
  while ( temp -> next!= NULL )  
  temp = temp -> next;  
  /\* add node at the end \*/  
  temp -> next = malloc ( sizeof ( struct node ) ) ;  
  temp = temp -> next;  
  }  
  /\* assign data to the last node \*/  
  temp -> data = num ;  
  temp -> next = NULL ;  
  }

/\* copies a linked list into another \*/  
void copy ( struct node \* f, struct node \*\*s )  
{  
if (f!= NULL )  
{  
\*s = malloc ( sizeof ( struct node ) ) ;  
( \*s ) -> data = f -> data ;  
( \*s ) -> next = NULL ;  
copy (f -> next, &( ( \*s ) -> next) ) ;  
}  
}

/\* displays the contents of the linked list \*/  
void display ( struct node \* head)  
{  
printf ( "\n" ) ;  
/\* traverse the entire linked list \*/  
while (head!= NULL )  
{  
printf ( "%d ", head -> data ) ;  
head = head -> next;  
}  
}

1. **Basic Calculations:**

* Count the number of element in linked list

/\* Link list node \*/

struct Node

{

    int data;

    struct Node\* next;

};

/\* Given a reference (pointer to pointer) to the head

  of a list and an int, push a new node on the front

  of the list. \*/

void push(struct Node\*\* head\_ref, int new\_data)

{

    /\* allocate node \*/

    struct Node\* new\_node =

            (struct Node\*) malloc(sizeof(struct Node));

    /\* put in the data  \*/

    new\_node->data  = new\_data;

    /\* link the old list off the new node \*/

    new\_node->next = (\*head\_ref);

    /\* move the head to point to the new node \*/

    (\*head\_ref)    = new\_node;

}

/\* Counts no. of nodes in linked list \*/

int getCount(struct Node\* head)

{

    int count = 0;  // Initialize count

    struct Node\* current = head;  // Initialize current

   while (current != NULL)

    {

       count++;

        current = current->next;

    }

    return count;

}

* Find the maximum, minimum and average value of element in a list

/\* Linked list node \*/

struct Node {

    int data;

    struct Node\* next;

};

// Function that returns the largest element

// from the linked list.

int largestElement(struct Node\* head)

{

    // Declare a max variable and initialize

    // it with INT\_MIN value.

    // INT\_MIN is integer type and its value

    // is -32767 or less.

    int max = INT\_MIN;

    // Check loop while head not equal to NULL

    while (head != NULL) {

        // If max is less then head->data then

        // assign value of head->data to max

        // otherwise node point to next node.

        if (max < head->data)

            max = head->data;

        head = head->next;

    }

    return max;

}

// Function that returns smallest element

// from the linked list.

int smallestElement(struct Node\* head)

{

    // Declare a min variable and initialize

    // it with INT\_MAX value.

    // INT\_MAX is integer type and its value

    // is 32767 or greater.

    int min = INT\_MAX;

    // Check loop while head not equal to NULL

    while (head != NULL) {

        // If min is greater then head->data then

        // assign value of head->data to min

        // otherwise node point to next node.

        if (min > head->data)

            min = head->data;

        head = head->next;

    }

    return min;

}

// Function that push the element in linked list.

void push(struct Node\*\* head, int data)

{

    // Allocate dynamic memory for newNode.

    struct Node\* newNode =

         (struct Node\*)malloc(sizeof(struct Node));

    // Assign the data into newNode.

    newNode->data = data;

    // newNode->next assign the address of

    // head node.

    newNode->next = (\*head);

    // newNode become the headNode.

    (\*head) = newNode;

}

// Display linked list.

void printList(struct Node\* head)

{

    while (head != NULL) {

        printf("%d -> ", head->data);

        head = head->next;

    }

    cout << "NULL" << endl;

}

1. **List Manipulation**

* Move a node forward n positions in a list
* Swap two adjacent nodes in the list(not just the data )
* #include <stdio.h>
* #include <stdlib.h>
* // Data Structure to store a linked list node
* struct Node
* {
* int data;
* struct Node \*next;
* };
* // Function to pairwise swap adjacent nodes of a linked list
* void rearrange(struct Node \*\*headRef)
* {
* // if list is empty or contains just one node
* if (\*headRef == NULL || (\*headRef)->next == NULL)
* return;
* struct Node \*curr = \*headRef, \*prev = NULL;
* // consider two nodes at a time and swap their links
* while (curr != NULL && curr->next != NULL)
* {
* struct Node\* temp = curr->next;
* curr->next = temp->next;
* temp->next = curr;
* if (prev == NULL)
* \*headRef = temp;
* else
* prev->next = temp;
* prev = curr;
* curr = curr->next;
* }
* }
* // main function
* int main(void)
* {
* int arr[] = { 1, 2, 3, 4, 5, 6, 7, 8 };
* unsigned n = sizeof(arr)/sizeof(arr[0]);
* struct Node \*head = NULL;
* for (int i = n - 1; i >= 0; i--)
* push(&head, arr[i]);
* printList("Before", head);
* rearrange(&head);
* printList("After ", head);
* return 0;
* }
* <https://www.techiedelight.com/pairwise-swap-adjacent-nodes-linked-list/>
* Swap the mth and nth nodes in a list ( not just the data )

#include

#include

#include

**struct** node

{

**int** data;

**struct** node \*link;

}\*head,\*q,\*temp,\*t;

**int** cnt,num1,num2,m,n;

**void** **create**();

**void** **display**();

**void** **swap**();

**int** **count**();

**void** **main**()

{

**int** ch='y';

clrscr();

**while**(ch=='y'||ch=='Y')

{

create();

printf("**\n**Want to add another node(Y|N): ");

scanf("%s",&amp;ch);}

display();

swap();

count();

display();

getch();

}

**void** **create**()

{

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

printf("**\n**Enter the elements for singly linked list: ");

scanf("%d",&amp;temp-&gt;data);

temp-&gt;link=NULL;

**if**(head==NULL)

head=temp;

**else**{

q=head;

**while**(q-&gt;link!=NULL)

{

q=q-&gt;link;

}

q-&gt;link=temp;

}

}

**void** **display**()

{

**if**(head==NULL)

printf("**\n**linked list is empty");

**else**

{

printf("**\n**Element in singly linked list are: **\n**");

q=head;

**while**(q!=NULL)

{

printf("%d**\n**",q-&gt;data);

q=q-&gt;link;

}

}

}

**void** **swap**(){

**int** i;

printf("**\n**Enter Mth and Nth position of singly linked list for swaping: ");

scanf("**\n**%d%d",&amp;m,&amp;n);

count();

**if**(m&gt;**0**&amp;&amp;m&lt;=cnt&amp;&amp;n&gt;**0**&amp;&amp;n&lt;=cnt)

{

q=head;

i=**0**;

printf("**\n\n**Element after swapping");

**while**(q!=NULL)

{

i++;

**if**(m==i)

q-&gt;data=num2;

**if**(n==i)

q-&gt;data=num1;

q=q-&gt;link;

}

}

**else**

printf("**\n**you entered wrong position please try again");

}

**int** **count**()

{

cnt=**0**;

q=head;

**while**(q!=NULL)

{

cnt++;

**if**(cnt==m)

num1=q-&gt;data;

**if**(cnt==n)

num2=q-&gt;data;

q=q-&gt;link;

}

**return** cnt;

}

* Reverse the list (in- place).

#include <stdio.h>

#include <stdlib.h>

/\* Structure of a node \*/

struct node {

int data; //Data part

struct node \*next; //Address part

}\*head;

/\* Functions used in the program \*/

void createList(int n);

void reverseList();

void displayList();

int main()

{

int n, choice;

/\*

\* Create a singly linked list of n nodes

\*/

printf("Enter the total number of nodes: ");

scanf("%d", &n);

createList(n);

printf("\nData in the list \n");

displayList();

/\*

\* Reverse the list

\*/

printf("\nPress 1 to reverse the order of singly linked list\n");

scanf("%d", &choice);

if(choice == 1)

{

reverseList();

}

printf("\nData in the list\n");

displayList();

return 0;

}

/\*

\* Create a list of n nodes

\*/

void createList(int n)

{

struct node \*newNode, \*temp;

int data, i;

if(n <= 0)

{

printf("List size must be greater than zero.\n");

return;

}

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

/\*

\* If unable to allocate memory for head node

\*/

if(head == NULL)

{

printf("Unable to allocate memory.");

}

else

{

/\*

\* Read data of node from the user

\*/

printf("Enter the data of node 1: ");

scanf("%d", &data);

head->data = data; // Link the data field with data

head->next = NULL; // Link the address field to NULL

temp = head;

/\*

\* Create n nodes and adds to linked list

\*/

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

{

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

/\* If memory is not allocated for newNode \*/

if(newNode == NULL)

{

printf("Unable to allocate memory.");

break;

}

else

{

printf("Enter the data of node %d: ", i);

scanf("%d", &data);

newNode->data = data; // Link the data field of newNode with data

newNode->next = NULL; // Link the address field of newNode with NULL

temp->next = newNode; // Link previous node i.e. temp to the newNode

temp = temp->next;

}

}

printf("SINGLY LINKED LIST CREATED SUCCESSFULLY\n");

}

}

/\*

\* Reverse the order of nodes of a singly linked list

\*/

void reverseList()

{

struct node \*prevNode, \*curNode;

if(head != NULL)

{

prevNode = head;

curNode = head->next;

head = head->next;

prevNode->next = NULL; // Make first node as last node

while(head != NULL)

{

head = head->next;

curNode->next = prevNode;

prevNode = curNode;

curNode = head;

}

head = prevNode; // Make last node as head

printf("SUCCESSFULLY REVERSED LIST\n");

}

}

/\*

\* Display entire list

\*/

void displayList()

{

struct node \*temp;

/\*

\* If the list is empty i.e. head = NULL

\*/

if(head == NULL)

{

printf("List is empty.");

}

else

{

temp = head;

while(temp != NULL)

{

printf("Data = %d\n", temp->data); // Print the data of current node

temp = temp->next; // Move to next node

}

}

}

<https://codeforwin.org/2015/09/c-program-to-reverse-singly-linked-list.html>

* Concatenate two linked lists
* **#include** *<stdio.h>*
* **#include** *<stdlib.h>*
* **struct** node
* {
* **int** data;
* **struct** node \*next;
* };
* display(**struct** node \*head)
* {
* **if**(head == NULL)
* {
* printf("NULL\n");
* }
* **else**
* {
* printf("%d\n", head -> data);
* display(head->next);
* }
* }
* **void** concatenate(**struct** node \*a,**struct** node \*b)
* {
* **if**( a != NULL && b!= NULL )
* {
* **if** (a->next == NULL)
* a->next = b;
* **else**
* concatenate(a->next,b);
* }
* **else**
* {
* printf("Either a or b is NULL\n");
* }
* }
* **int** main()
* {
* **struct** node \*prev,\*a, \*b, \*p;
* **int** n,i;
* printf ("number of elements in a:");
* scanf("%d",&n);
* a=NULL;
* **for**(i=0;i<n;i++)
* {
* p=malloc(**sizeof**(**struct** node));
* scanf("%d",&p->data);
* p->next=NULL;
* **if**(a==NULL)
* a=p;
* **else**
* prev->next=p;
* prev=p;
* }
* printf ("number of elements in b:");
* scanf("%d",&n);
* b=NULL;
* **for**(i=0;i<n;i++)
* {
* p=malloc(**sizeof**(**struct** node));
* scanf("%d",&p->data);
* p->next=NULL;
* **if**(b==NULL)
* b=p;
* **else**
* prev->next=p;
* prev=p;
* }
* concatenate(a,b);
* **return** 0;
* }

<https://www.codesdope.com/blog/article/concatenating-two-linked-lists-in-c/>

* Combine two sorted lists into a single sorted linked list

/\* C program to merge two sorted linked lists \*/

#include<stdio.h>

#include<stdlib.h>

#include<assert.h>

/\* Link list node \*/

struct Node

{

int data;

struct Node\* next;

};

struct Node\* SortedMerge(struct Node\* a, struct Node\* b)

{

  struct Node\* result = NULL;

  /\* Base cases \*/

  if (a == NULL)

     return(b);

  else if (b==NULL)

     return(a);

  /\* Pick either a or b, and recur \*/

  if (a->data <= b->data)

  {

     result = a;

     result->next = SortedMerge(a->next, b);

  }

  else

  {

     result = b;

     result->next = SortedMerge(a, b->next);

  }

  return(result);

}

<https://www.geeksforgeeks.org/merge-two-sorted-linked-lists/>

* Given two linked lists create a new list containing their union( all the unique element s between them )

// C/C++ program to find union and intersection of two unsorted

// linked lists

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

/\* Link list node \*/

struct Node

{

int data;

struct Node\* next;

};

/\* A utility function to insert a node at the beginning of

a linked list\*/

void push(struct Node\*\* head\_ref, int new\_data);

/\* A utility function to check if given data is present in a list \*/

bool isPresent(struct Node \*head, int data);

/\* Function to get union of two linked lists head1 and head2 \*/

struct Node \*getUnion(struct Node \*head1, struct Node \*head2)

{

struct Node \*result = NULL;

struct Node \*t1 = head1, \*t2 = head2;

// Insert all elements of list1 to the result list

while (t1 != NULL)

{

push(&result, t1->data);

t1 = t1->next;

}

// Insert those elements of list2 which are not

// present in result list

while (t2 != NULL)

{

if (!isPresent(result, t2->data))

push(&result, t2->data);

t2 = t2->next;

}

return result;

}

/\* Function to get intersection of two linked lists

head1 and head2 \*/

struct Node \*getIntersection(struct Node \*head1,

struct Node \*head2)

{

struct Node \*result = NULL;

struct Node \*t1 = head1;

// Traverse list1 and search each element of it in

// list2. If the element is present in list 2, then

// insert the element to result

while (t1 != NULL)

{

if (isPresent(head2, t1->data))

push (&result, t1->data);

t1 = t1->next;

}

return result;

}

/\* A utility function to insert a node at the begining of a linked list\*/

void push (struct Node\*\* head\_ref, int new\_data)

{

/\* allocate node \*/

struct Node\* new\_node =

(struct Node\*) malloc(sizeof(struct Node));

/\* put in the data \*/

new\_node->data = new\_data;

/\* link the old list off the new node \*/

new\_node->next = (\*head\_ref);

/\* move the head to point to the new node \*/

(\*head\_ref) = new\_node;

}

/\* A utility function to print a linked list\*/

void printList (struct Node \*node)

{

while (node != NULL)

{

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

node = node->next;

}

}

/\* A utility function that returns true if data is

present in linked list else return false \*/

bool isPresent (struct Node \*head, int data)

{

struct Node \*t = head;

while (t != NULL)

{

if (t->data == data)

return 1;

t = t->next;

}

return 0;

}

/\* Drier program to test above function\*/

int main()

{

/\* Start with the empty list \*/

struct Node\* head1 = NULL;

struct Node\* head2 = NULL;

struct Node\* intersecn = NULL;

struct Node\* unin = NULL;

/\*create a linked lits 10->15->5->20 \*/

push (&head1, 20);

push (&head1, 4);

push (&head1, 15);

push (&head1, 10);

/\*create a linked lits 8->4->2->10 \*/

push (&head2, 10);

push (&head2, 2);

push (&head2, 4);

push (&head2, 8);

intersecn = getIntersection (head1, head2);

unin = getUnion (head1, head2);

printf ("\n First list is \n");

printList (head1);

printf ("\n Second list is \n");

printList (head2);

printf ("\n Intersection list is \n");

printList (intersecn);

printf ("\n Union list is \n");

printList (unin);

return 0;

}