Program 1:

Program to save names in a SLL, names in the list must be saved in ascending order based on ASCII value

struct node

{

char name[20];

struct node \*link;

}

#include<stdio.h>

#include<stdlib.h>

typedef struct name{

char names[50];

struct name \*link;

}nm;

nm\* insert(nm \*);

int main()

{

nm \*t,\*first=NULL;

int n;

scanf("%d",&n);

while(n--)

first=insert(first);

for(t=first;t!=NULL;t=t->link)

printf("%s\n",t->names);

return 0;

}

nm\* insert(nm \*f)

{

nm \*t,\*l,\*p;

t=(nm\*)malloc(sizeof(nm));

t->link=NULL;

scanf("%s",t->names);

if(f==NULL)

return t;

for(p=0,l=f;l!=NULL;p=l,l=l->link)

{

if(l->names[0]>t->names[0] && l==f){

t->link=l;

return t;

}

else if(l->names[0]>t->names[0]){

t->link=l;

p->link=t;

return f;

}

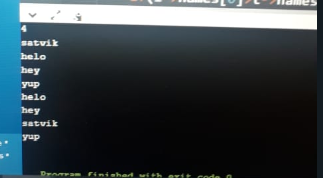
}

p->link=t;

l=t;

return f;

}



Program : 2Write separate functions to delete a node from SLL based on i) position ii) based on name use the previously defined structure.

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

typedef struct name{

char names[50];

struct name \*link;

}nm;

nm\* insert(nm \*);

nm\* del\_pos(nm \*);

nm\* del\_nam(nm \*);

int main()

{

nm \*t,\*first=NULL;

int n;

printf("Enter total no. of names to insert: ");

scanf("%d",&n);

while(n--)

first=insert(first);

printf("\nAfter insertion list\n");

for(t=first;t!=NULL;t=t->link)

printf("%s\n",t->names);

first=del\_pos(first);

printf("\nUpdated list\n");

for(t=first;t!=NULL;t=t->link)

printf("%s\n",t->names);

first=del\_nam(first);

printf("\nUpdated list\n");

for(t=first;t!=NULL;t=t->link)

printf("%s\n",t->names);

return 0;

}

nm\* insert(nm \*f)

{

nm \*t,\*l,\*p;

t=(nm\*)malloc(sizeof(nm));

t->link=NULL;

printf("Enter name: ");

scanf("%s",t->names);

if(f==NULL)

return t;

for(p=0,l=f;l!=NULL;p=l,l=l->link)

{

if(l->names[0]>t->names[0] && l==f){

t->link=l;

return t;

}

else if(l->names[0]>t->names[0]){

t->link=l;

p->link=t;

return f;

}

}

p->link=t;

l=t;

return f;

}

nm\* del\_pos(nm \*f)

{

nm \*t,\*p;

int pos;

printf("\nEnter pos to del item: ");

scanf("%d",&pos);

pos--;

for(p=0,t=f;t!=NULL;p=t,t=t->link,pos--)

{

if(pos==0)

{

if(t==f)

{

printf("Deleted name: %s\n",t->names);

t=t->link;

return t;

}

else

{

printf("Deleted name: %s\n",t->names);

p->link=t->link;

free(t);

return f;

}

}

}

printf("Position is not valid\n");

}

nm\* del\_nam(nm \*f)

{

char str[100];

printf("\nEnter string to del: ");

scanf("%s",str);

nm \*t,\*p;

for(p=0,t=f;t!=NULL;p=t,t=t->link)

{

if(strcmp(str,t->names)==0)

{

if(t==f)

{

printf("Deleted name: %s\n",t->names);

t=t->link;

return t;

}

else

{

printf("Deleted name: %s\n",t->names);

p->link=t->link;

free(t);

return f;

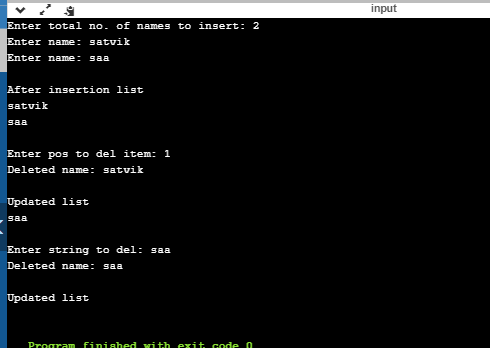
}

}

}

printf("Name not found\n");

}



Program 4:

\* C Program to Remove Duplicates from a Linked List

\*/

#include <stdio.h>

#include <stdlib.h>

struct node

{

int num;

struct node \*next;

};

void create(struct node \*\*);

void dup\_delete(struct node \*\*);

void release(struct node \*\*);

void display(struct node \*);

int main()

{

struct node \*p = NULL;

struct node\_occur \*head = NULL;

int n;

printf("Enter data into the list\n");

create(&p);

printf("Displaying the nodes in the list:\n");

display(p);

printf("Deleting duplicate elements in the list...\n");

dup\_delete(&p);

printf("Displaying non-deleted nodes in the list:\n");

display(p);

release(&p);

return 0;

}

void dup\_delete(struct node \*\*head)

{

struct node \*p, \*q, \*prev, \*temp;

p = q = prev = \*head;

q = q->next;

while (p != NULL)

{

while (q != NULL && q->num != p->num)

{

prev = q;

q = q->next;

}

if (q == NULL)

{

p = p->next;

if (p != NULL)

{

q = p->next;

}

}

else if (q->num == p->num)

{

prev->next = q->next;

temp = q;

q = q->next;

free(temp);

}

}

}

void create(struct node \*\*head)

{

int c, ch;

struct node \*temp, \*rear;

do

{

printf("Enter number: ");

scanf("%d", &c);

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

temp->num = c;

temp->next = NULL;

if (\*head == NULL)

{

\*head = temp;

}

else

{

rear->next = temp;

}

rear = temp;

printf("Do you wish to continue [1/0]: ");

scanf("%d", &ch);

} while (ch != 0);

printf("\n");

}

void display(struct node \*p)

{

while (p != NULL)

{

printf("%d\t", p->num);

p = p->next;

}

printf("\n");

}

void release(struct node \*\*head)

{

struct node \*temp = \*head;

\*head = (\*head)->next;

while ((\*head) != NULL)

{

free(temp);

temp = \*head;

(\*head) = (\*head)->next;

}

}

3. write a program to merge two SLL and the result must be sorted

#include <stdio.h>

#include <stdlib.h>

// A Linked List Node

struct Node

{

int data;

struct Node\* next;

};

// Helper function to print a given linked list

void printList(struct Node\* head)

{

struct Node\* ptr = head;

while (ptr)

{

printf("%d —> ", ptr->data);

ptr = ptr->next;

}

printf("NULL\n");

}

// Helper function to insert a new node at the beginning of the linked list

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

{

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

newNode->data = data;

newNode->next = \*head;

\*head = newNode;

}

// Function takes the node from the front of the source and moves it

// to the front of the destination

void moveNode(struct Node\*\* destRef, struct Node\*\* sourceRef)

{

// if the source list empty, do nothing

if (\*sourceRef == NULL) {

return;

}

struct Node\* newNode = \*sourceRef; // the front source node

\*sourceRef = (\*sourceRef)->next; // advance the source pointer

newNode->next = \*destRef; // link the old dest off the new node

\*destRef = newNode; // move dest to point to the new node

}

// Takes two lists sorted in increasing order and merge their nodes

// to make one big sorted list, which is returned

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

{

// a dummy first node to hang the result on

struct Node dummy;

dummy.next = NULL;

// points to the last result node — so `tail->next` is the place

// to add new nodes to the result.

struct Node\* tail = &dummy;

while (1)

{

// if either list runs out, use the other list

if (a == NULL)

{

tail->next = b;

break;

}

else if (b == NULL)

{

tail->next = a;

break;

}

if (a->data <= b->data) {

moveNode(&(tail->next), &a);

}

else {

moveNode(&(tail->next), &b);

}

tail = tail->next;

}

return dummy.next;

}

int main(void)

{

// input keys

int keys[] = { 1, 2, 3, 4, 5, 6, 7 };

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

struct Node \*a = NULL, \*b = NULL;

for (int i = n - 1; i >= 0; i = i - 2) {

push(&a, keys[i]);

}

for (int i = n - 2; i >= 0; i = i - 2) {

push(&b, keys[i]);

}

// print both lists

printf("First List: ");

printList(a);

printf("Second List: ");

printList(b);

struct Node\* head = sortedMerge(a, b);

printf("After Merge: ");

printList(head);

return 0;

}

