#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\*next;

};

struct node\*head;

print the list

void print()

{

struct node\*temp=head;

if(head==NULL)

{

printf("lsit is empty");

}

else{

temp=head;

while(temp!=0)

{

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

temp=temp->next;

}

}

}

Create the list

void create()

{

int choice=1;

struct node\* newnode,\*temp;

while(choice){

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

printf("eneter the data \n\n\n");

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

newnode->next=NULL;

if(head==0)

{

head=temp=newnode;

}

else

{

temp->next=newnode;

temp=newnode;

}

printf("do u want to add another node(0,1)?");

scanf("%d",&choice);

}

}

Get the length of the linked list

void get\_length()

{

int count;

struct node\* temp=head;

while(temp!=NULL)

{

count++;

temp=temp->next;

}

printf("length of linked is %d",count);

}

]

Sort of the linked list

void sortList() {

//Node current will point to head

struct node \*current = head;

struct node\*index=NULL;

int temp;

if(head == NULL) {

return;

}

else {

while(current != NULL) {

//Node index will point to node next to current

index = current->next;

while(index != NULL) {

//If current node's data is greater than index's node data, swap the data between them

if(current->data > index->data) {

temp = current->data;

current->data = index->data;

index->data = temp;

}

index = index->next;

}

current = current->next;

}

}

}

Reverse the linked list

void reverse()

{

struct node\*currentnode,\*previousnode,\*nextnode;

previousnode=0;

currentnode=nextnode=head;

while(nextnode!=0)

{

nextnode=nextnode->next;

currentnode->next=previousnode;

previousnode=currentnode;

currentnode=nextnode;

}

head=previousnode;

}

Remove duplicates from linked list

void RemoveDuplicate()

{

struct node\*head;

struct node\*current,\*nextNext;

if (head == NULL)

return;

current = head;

while (current->next != NULL)

{

if (current->data == current->next->data)

{

nextNext = current->next->next;

free(current->next);

current->next = nextNext;

}

else {

current = current->next;

}

}

}

Insertion sort for linked list

insertion\_sort() {

struct node \* p = head;

struct node \* currentNode = head->next;

struct node \* previousNode = head;

if (p == NULL || p->next == NULL) {

return 0;

}

while (currentNode != NULL) {

if(previousNode->data<=currentNode->data){

currentNode = currentNode->next;

previousNode = previousNode->next;

}

else{

if (head->data > currentNode->data) {

previousNode->next = currentNode->next;

currentNode->next = head;

head = currentNode;

}else {

p = head;

while (p->next != NULL && p->next->data < currentNode->data) {

p = p->next;

}

previousNode->next = currentNode->next;

currentNode->next = p->next;

p->next = currentNode;

}

}

currentNode = previousNode->next;

}

}

Sum of the linked list

int Sum()

{

int sum=0;

struct node\*temp=head;

while(temp!=NULL)

{

sum+=temp->data;

temp=temp->next;

}

printf("sum of linked list is %d \t",sum);

}

Get largest element in linked list

int largestelement()

{

struct node\*temp=head;

int max=INT\_MIN;

while(temp!=NULL)

{

if(max<temp->data)

max=temp->data;

temp=temp->next;

}

printf("largest element is %d \t", max);

}

Main function

int main()

{

create();

printf("list is:");

print();

printf("\n");

printf("\n");

printf("\n");

sortList();

printf("list after sorting \t");

print();

printf("\n");

printf("\n");

printf("\n");

printf("\n");

reverse();

printf("reversed linked is\t:");

print();

printf("\n");

printf("\n");

printf("\n");

printf("\n");

printf("list after removing duplicates \t");

RemoveDuplicate();

print();

printf("\n");

printf("\n");

printf("\n");

printf("elements after insertion sort\t:");

insertion\_sort();

print();

printf("\n");

printf("\n");

get\_length();

printf("\n");

printf("\n");

Sum();

largestelement();

print();

}