***LAB4:***

**Q1: Write a program that simulates a simple address book. Define a structure to store contact**

**information (name, email, phone number). Allow the user to add new contacts to the**

**address book dynamically. Use dynamic memory allocation for storing the contacts using**

**malloc and update the memory allocation using realloc when adding new contacts.**

**Implement an option to delete a contact and free the memory. Ensure that memory is**

**properly managed throughout the program's execution.**

struct contactinfo{

char name[50];

char phone\_no[20];

char email[50]};

void addcontact(struct contactinfo \*\*addressbook,int \*numcontact){

(\*numcontact)++;

struct contactinfo\*allocate=((struct contactinfo\*)malloc((\*numcontact)\*sizeof(struct contactinfo)));

\*addressbook=((struct contactinfo\*)realloc(\*addressbook,(\*numcontact)\*sizeof(struct contactinfo)));

printf("enter name\n");

scanf("%s",(\*addressbook)[\*numcontact-1].name);

printf("enter phone number\n");

scanf("%s",(\*addressbook)[\*numcontact-1].phone\_no);

printf("enter email\n");

scanf("%s",(\*addressbook)[\*numcontact-1].email);

printf("contact added succesfully");

}

void deletecontact(struct contactinfo \*\*addressbook,int \*numcontact,char contactname){

printf("enter the contact you want to delete");

scanf("%s",contactname);

int found=0;

for (int i=0;i<\*numcontact;i++){

if (strcmp((\*addressbook)[i].name,contactname)==0){

for (int j=i;j<\*numcontact-1;j++){

\*addressbook[j]=\*addressbook[j+1];

found++;

}

\*addressbook=((struct contactinfo\*)realloc(\*addressbook,(\*numcontact-1)\*sizeof(struct contactinfo)));

}

}

if (found==1){

printf("contact delete succesfully");

}

else{

printf("contact not found");

}

}

void displaycontact(struct contactinfo \*\*addressbook,int \*numcontact){

if (numcontact==0){

printf("no contact available");

}

else{

printf("contact list is\n");

for (int k=0; k<numcontact;k++){

printf("name is: %s",(\*addressbook)[k].name);

printf("phone number is: %s",(\*addressbook)[k].phone\_no);

printf("email is %s",(\*addressbook)[k].email);

printf("-----------------------------");

}

}

}

Int main(){

struct contactinfo \*addressbook=NULL;

int numcontact=0;

int choice;

char contactname[50];

do{

printf("1.Add contact\n 2.delete contact\n 3.display contact\n 4.exit\n");

printf("enter your choice");

scanf("%d",choice);

switch(choice){

case 1:

addcontact(&addressbook,&numcontact);

case 2:

deletecontact(&addressbook,&numcontact,contactname);

case 3:

displaycontact(&addressbook,&numcontact);

case 4:

free(addressbook);

printf("program exit");

}

}

while(choice!=4);

return 0;}

***Q2:***

***Write a C program to merge two sorted singly linked lists into a single sorted linked list.***

void append(struct node\*\* head,int data){

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

struct node\* temp=\*head;

newnode->data=data;

newnode->next= NULL;

if (\*head==NULL){

\*head=newnode;

return;

}

while(temp->next!=NULL)

{temp=temp->next;}

temp->next=newnode;

}

void printlist(struct node\* node){

while(node!=NULL){

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

node= node-> next;}

}

struct node\* mergesortedlist(struct node\* list1,struct node\*list2){

struct node\*mergedlist=NULL;

struct node\* newhead=NULL;

if(list1==NULL)

return list2;

else if(list2==NULL)

return list1;

if (list1 && list2){

if (list1->data<=list2->data){

mergedlist=list1;

list1=mergedlist->next;

}

else{

mergedlist=list2;

list2=mergedlist->next;

}

}

newhead=mergedlist;

while(list1&&list2){

if (list1->data<=list2->data){

mergedlist->next=list1;

mergedlist=list1;

list1=mergedlist->next;

}

else{mergedlist->next=list2;

mergedlist=list2;

list2=mergedlist->next;

}

}

if (list1==NULL)

mergedlist->next=list2;

if (list1==NULL)

mergedlist->next=list2;

return newhead;

}

Int main(){struct node\* list1=NULL;

struct node\* list2=NULL;

append(&list1,1);

append(&list1,3);

append(&list1,5);

printlist(list1);

append(&list2,2);

append(&list2,4);

append(&list2,6);

printlist(list2);

struct node\* mergedlist=mergesortedlist(list1,list2);

printlist(mergedlist);

return 0;}

**Q3:**

**Write a C program that converts a singly linked list into an array and returns it.**

int list\_to\_array(struct node\* head,int\* arraysize){

int count=0;

struct node\* temp=head;

while (temp!=NULL){

count++;

temp=temp->next;

}

int\* array=(int\*)malloc(count\* sizeof(int));

temp=head;

for (int i=1;i<count;i++){

array[i]=temp->data;

temp=temp->next;

}

\*arraysize=count;

return array;}

void printarray(int \*array,int \*arraysize){

printf("Array [");

for (int i=0;i<arraysize;i++){

printf("%d",array[i]);

printf(",");

} printf("]");

Int main(){

struct node\*\* head=NULL;

struct node\* temop=NULL;

int\*array=NULL;

int\* arraysize=NULL;

append(&head,1);

append(&head,2);

append(&head,3);

list\_to\_array(&head,&arraysize);

printarray(&array,&arraysize);}

***Q4:***

***Write a C program that removes elements with odd indices from a singly linked list.***

void deleteoddposition(){

struct node\*head;

struct node\*temp;

struct node\*nextnode;

int index=0;

temp=head;

if(head==NULL){

printf("list is empty");

}

while(temp!=NULL){

if(index%2 !=0){

nextnode->next=temp->next;

free(temp);

temp=nextnode->next;}

else{

nextnode=temp;

temp=temp->next;

}index++;

}

}

**Functions append and printlist already in ques 1 and 2.**

Int main(){

{struct node\*\* head=NULL;

struct node\* temop=NULL;

struct node\* newnode=NULL;

struct node\*nextnode=NULL;

struct node\* node=NULL;

append(&head,1);

append(&head,2);

append(&head,3);

append(&head,4);

append(&head,5);

append(&head,6);

deleteoddposition();

printlist(&node);

return 0;}