DSA

Assignment-4

Y.Shivaprasad
AP19110010242
CSE-G

1. Program to Insert and delete an element at Nth and Kth pos.

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int info;
struct node*next;
};
display(struct node* head)
{
if(head==NULL){
printf("It is empty");}
else{
printf("%d", head->data);
display(head->next);}
}
del(struct node* before_del)
{
struct node*temp;
temp=before_del->next;
before_del->next=temp->next;
free(temp);}
struct node*front(struct node*head, int num){
struct node*B;
B=malloc(sizeof(struct node));
B->data=num;
B->next=head;
return(B);
}
end(struct node* head, int num){
```

```
struct node* B,*A;
B=malloc(size of(struct node));
B->date=num;
B->next=null;
A=head;
while(A->next!=null)
{
A=A->next;
}
A->next=B;}
after(struct node*g,int num){
if(g->next!=null){
struct node*B;
B=malloc(size of(struct node));
B->date=num;
B->next=g->next;
g->next=B;}
else{
printf("Insert the number at the end");}
}
int main(){
struct node* Before,*head,*B;
int g,j;
printf("elements in total");
scanf("%d",&g);
head=null;
for(j=0;j<g;j++)
{
p=malloc(size of(struct node));
scanf("%d",&B->data);
B->next=Null;
if(head==null)
head=B;
else
Before->next=B;
Before= B;}
head= front(head, 5);
end(head,15);
after(head->next->next, 30);
del(head->next);
del(head->next->next);
display(head);
return0;
}
```

<u>output:-</u>

```
total elements:6
3
7
8
4
9
2

5
3
30
4
9
16
Null
```

2. New Linked List by merging Alternate nodes.

```
#include <stdio.h>
#include <stdlib.h>
Struct Node {
int value;
struct Node* next;
};
Void printList(struct Node* head)
{
   struct Node* ptr=head;
   while(ptr)
   {
     printf("%d->", ptr->dada);
      ptr=ptr->next;
    }
    printf("It is empty');}
Voidpush(str Node** head, int value)
 {
   struct Node* NewNode=(structNode*)malloc
          (sizeof(struct node));
     newnode->value=Value;
     newnode->next=*head;
      *head=newnode;
 }
Struct node* shufflemerge(struct node*q, struct
             node*p) {
   Struct node fake;
   Struct node* end=&fake;
    fake-next=null;
```

```
while(True)
{
 if(g==null)
  {
   end->next=p;
   break;
  }
  else if(p==null)
  {
   end->next=g;
   break;}
  else{
   end->next=g;
   end=g;
    g=g->next;
    end->next=p;
     end=p;
     p=p->next;
   }
 }
return fake.next;}
int main(void){
 int keys[]={1,2,3,4,5,6,7};
int n= sizeof(keys)/sizeof(keys[0]);
struct node*g=null,*p=null;
for(int i=n-1;i>=0;i=i-2)
  push(&g,keys[i]);
for(int i=n-2;i>=0;i=i-2)
    pust(&p,keys[i]);
```

```
printf("The first list:");
printlist(a);
printf("The second list:");
printlist(b);
Struct Node* head=Shuffle merge(a,b);
printf("Merging has done:");
PrintList(head);
return0;
}
```

output:-

```
The first list:
1 2 3
The second list:
4 5 6
merging has done:
1 4 2 5 3 6
```

3. Find all the elements in the stack whose sum is equal to k.

```
#include<stdio.h>
int top=-1;
int num;
Char Stack[50];
Void push(int num);
Char pop();
int main()
{
int j,n,g,t,k,b,total=0,count=1;
```

```
printf("total number of elements");
Scanf("%d",&n);
for(j=0;j<n;j++){
printf("next element");
Scanf("%d",&g);
push(g); }
printf("Sum to be checked");
Scanf("%d",&k);
for(j=0;j<n;j++)
{
t=pop();
total+=t
count+=1
if(total==k){
for(int i=0;i<count;i++)
printf("%d", Stack[i]);
b=1;
break; }
push(t);
}
if(b!=1)
print("Total elements in stack donot sum up"); }
void push(int num)
{
if(top=99)
{
printf(" Stack is Full");
return;
}
top=top+1
```

```
Stack[top]=num;
}
char pop()
{
if(Stack[top]==-1)
{
printf("Empty Stack");
return0;
}
num=Stack[top];
 top=top-1
   returnnum;
}
```

Output:-

```
Total number of elements : 5
 Enter next element: 4
  Next element      5
  Next element      9
  Next element      7
                     2
  Next element
Sum to be checked 3
The elements in Stack donot sum up.
```

4) i) Elements in a queue in reverse order.

```
#include<Stdio.h>
#include<Stdlib.h>
Structnode
{
int info;
Structnode* next;
};
Struct queue
{
Structnode* front;
Structnode* rear; };
Struct Stacknode{
intinfo;
StructStacknode*next;};
StructStacknode* push(StructStacknode* top, int x);
Struct queue* enqueue(Struct queue* q, int n);
int dequeue(Struct queue** q);
int pop(Struct Stack node**S);
int main(Void){
Struct queue* Q=Null;
Q=enqueue(Q, 13);
Q=enqueue(Q,9);
Q=enqueue(Q, 19);
Q=enqueue(Q, 130);
Q=enqueue(Q, 82);
Q=enqueue(Q, 77);
printQ(Q);
Struct Stacknode* S=null;
```

```
while( Q->front!=null)
 S=push(S. dequeue(&Q));
 Q=null;
 while(S!=null)
 Q=enqueue(Q.pop(&S));
printer(Q);
returno;}
Structstacknode* psh(Structstacknode* top,int x){
Structstacknode* temp= (Structstacknode*)malloc
                          (Sizeof(Structstacknode));
if(!temp){
printf("Stack is full");
return top;
}
temp->info=x;
temp->next=top;
returntemp;}
Structqueue*enqueue(Structqueue* q, int n){
Structnode*temp=(Structnode*)malloc(Sizeof(
          Structnode));
temp->info=n;
temp->next=null;
if(q==null){
q=(Structqueue*)malloc(Sizeof(Structqueue));
if(!q) {
printf("Exception of overflow");
return null;
}
q->font=temp;}
else
```

```
q->rear->next=temp;
q->rear=temp;
return q; }
int dequeue(struct queue**q){
int x=(*q)->front->data;
struct node* temp=(*q)->front;
(*q)->front: (*q)->front->next;
free(temp);
return y;
}
int pop(struct stacknode**s)
{
int y=(*s)->info;
struct stacknode* temp=*s;
*s=(*s)->next;
free(temp);
return x; }
void printer(struct queue*q){
struct node* y=q->front;
while(y!=null){
printf("%d", y->data);
 y=y->next; }
 printf(" /n"); }
```

output

```
13 9  19  130  82  77
77 82 130 19  9  13
```

5. (i)

ARRAY	Linked list
1. Insertion and deletion take more time.	1. Insertion and deletion process take less time.
2. It occupies less memory than a linked list for the same number of elements	2. It occupies more memory
3. Size of an array is fixed	3. Size of a list is not fixed.
4. memory utilization is Ineffective	4. memory utilization is efficient
5. memory required is less	5. memory required is more.

(ii) write a program to add the first element of one list to another list for example we have {1,2,3} in list1 and {4,5,6} in list2 we have to get {4,1,2,3} as output for list1 and {5,6} for list2.

```
#include<stdio.h>
#include<stdlib.h>
Struct Node
{
int info;
Struct Node*next;
};
Void push(structnode**head_ref, int data)
{
struct node*new_node=(structnode*)malloc(sizeof(
          (structnode));
new_node->info=data
new_node->next=(*head_ref);
(*head_ref)=new_node;
}
void printlist(structnode* head)
{
struct node* temp=head;
while(temp!=null)
{
printf("%d", temp->info);
temp=temp->next;
}
printf("\n");
}
void merge(struct node* B, structnode**a)
{
struct node* B-current=B, *a-current=*a;
struct node* B-next, *a-next;
while(B-current!=null && a-current!=null)
B-next=B-current->next;
a-next=a-current->next;
a-current->next=B-next;
B-current->next=a-current;
B-current=B-next;
a-current=a-next;
}
*a =a-curr;
}
int main(){
```

```
Structnode*B=null, *a=null;
push(&B, 2);
push(&B, 5);
push(&B, 7);
printf("first linked list");
printlist(B);
push(&a, 7);
push(&a, 9);
push(&a, 15);
push(&a, 16);
push(&a, 0);
printf("2nd linked list");
printlist(a);
merge(B, &a);
printf("first linked list after changing");
printlist(B);
printf("second linked list after changing");
printlist(a);
```

output:-

```
first linked list
2 5 7
Second linked list
7,9,15,16,0
first linked list after changing
 2 7 5 9 7 15
Second linked list after changing
 16 0
```