Computer Assignment-4

M.surya Prakash
AP19110010082

1. PROGRAM TO INSERT AND DELETE AN ELEMENT AT N^{TH} AND K^{TH} POS

```
#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);
}
}
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 Value)
{
struct node* p;
p= malloc( sizeof (struct node));
p -> data = Value ;
p -> next = head;
return (p);
}
end (struct node *head, int Value)
{
struct node * p, * q;
p= malloc (sizeof (struct node));
p -> data = Value;
p -> next = NULL;
q = head;
While (q -> next != NULL)
{
 q = q -> next;
}
q -> next = p;
}
after (struct node * a, int Value)
{
 if (a -> next != NULL)
```

```
{
struct node * p;
p = malloc (sizeof (structnode));
p -> data = value;
p -> next = a -> next;
a -> next = p;
}
else
{
printf ("USE END FUNCTION TO INSERT AT THE END\n");
}
}
int main ( )
{
struct node * prev, * head, * p;
int a, i ;
printf ("NUMBER OF ELEMENTS");
scanf (" %d", &a);
head = NULL;
for (i=0; i<a; i++)
{
p = malloc (sizeof (struct node));
scanf (" %d", &p -> data);
p -> next = NULL;
if (head == NULL)
 head = p;
```

```
else
prev -> next = p;
prev = p;
}
head = front(head, 10);
end(head, 20);
after(head -> next -> next, 30);
del(head -> next);
del(head -> next -> next);
display(head);
return 0;
}
```

OUTPUT

```
NUMBER OF ELEMENTS 5
 1
 2
 3
 4
 5
10
 1
30
 4
 5
20
NULL
```

20 New Linked List By Merging Alternate Nodes

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
int data;
struct Node * next;
};
Void push (struct Node ** head_ref, int new_data)
{
struct Node * new_node = (struct Node *)malloc(sizeof (struct Node));
new_node -> data = new_data;
new_node -> next = (*head_ref);
(*head_ref) = new_node;
}
Void printlist (struct Node * head)
{
struct Node * temp = head;
while (temp != NULL)
{
printf (" %d", temp -> data);
temp = temp -> next;
}
printf("\n");
}
```

```
Void merge (struct Node *P, struct Node **q)
{
struct Node *P_Curr = P, *q_Curr = *q;
struct Node *P_next, *q_next;
While (P_Curr != NULL && q_curr != NULL)
P_next = P_Curr -> next;
q_next = q_Curr -> next;
q_Curr -> next = P_next;
P_Curr -> next = q_Curr;
P_Curr = P_next;
q_Curr = q_next;
}
*q = q_Curr;
}
int main ( )
{
struct Node *P = NULL, *q = NULL;
Push (&P, 0);
Push (&P, 3);
Push (&P, 1);
Printf (" 1ST LINKED LIST \n");
printlist (P);
```

```
Push(&q, 2);
Push(&q, 8);
Push(&q, 0);
Printf("2ND LINKED LIST\n");
printlist(q);
merge(P, &q);
Printf("CHANGED LINKED LIST\n");
printlist(P);
return 0;
}
```

OUTPUT

1ST LINKED LIST

1 3 0

2ND LINKED LIST

0 8 2

CHANGED LINKED LIST

1 0 3 8 0 2

4 i) Elements in a Queue in Reverse Order

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
int data;
struct node * next;
};
struct queue
{
struct node *front;
struct node * rear;
};
struct stackNode
{
int data;
struct stackNode *next;
};
struct stackNode * push(struct stackNode * top, int element);
struct queue * enQueue(struct queue * q, int num);
int deQuene(struct queue **q);
int pop(struct stackNode ** s);

int main(Void)
{
struct queue * Q = NULL;
```

```
Q = enQueue (Q, 1);
Q = enQueue (Q, 7);
Q = enQueue (Q, 10);
Q = enQueue (Q, 130);
Q = enQueue (Q, 82);
Q = enQueue (Q, 77);
printes (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);
return 0;
}
struct stackNode * push (struct stackNode* top, int element)
{
struct stackNode * temp = (struct stackNode*) malloc
                          (sizeof (struct stackNode));
if (!temp)
{
```

```
printf ("STACK OVERFLOW");
return top;
}
temp -> data = element;
temp -> next = top;
return temp;
}
struct queue *enQueue (struct queue *q, int num)
{
struct node * temp = (struct node *) malloc (sizeof (struct node));
temp -> data = num;
temp -> next = NULL;
if (q == NULL)
{
q = (struct queue *) malloc (sizeof (struct queue));
if (!q)
{
printf ("OVERFLOW EXCEPTION");
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 x;
}
int pop (struct stackNode **s)
{
int x = (*s) ->data;
struct stackNode * temp = *s;
*s = (*s) -> next;
free (temp);
return x;
}
Void printer (struct queue * q)
{
struct node *x = q -> front;
While (x! = NULL)
{
printf(" %d", x ->data);
 x = x -> next;
}
printf ("\n");
}
```

Output

1 7 10 130 82 77

77 82 130 10 7 1

5o

i) Array

1o Size of an array is fixed

2o It Occupies less memory than a linked list for the same number of elements.

3o Deleting an element from an array is not possible.

4o Insertion and deletion take more time.

Linked List

1o Size of a list is not fixed

2o It Occupies more memory

3o Deleting an element is possible.

4o Insertion and deletion process take less time.

P.T.O

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
int data;
struct Node * next;
};
Void push (struct Node ** head_ref, int new_data)
{
struct Node * new_node = (struct Node *) malloc (sizeof (struct Node));
new_node -> data = new_data;
new_node -> next = (*head_ref);
(*head_ref) = new_node;
}
Void printlist (struct Node * head)
{
struct Node * temp = head;
while (temp != NULL)
{
printf (" %d", temp -> data);
temp = temp -> next;
}
printf("\n");
}
```

```
Void merge (struct Node *P, struct Node **q)
{
struct Node *P_Curr = P, *q_Curr = *q;
struct Node *P_next, *q_next;
while (P_Curr != NULL && q_curr != NULL)
P_next = P_Curr -> next;
q_next = q_Curr -> next;
q_Curr -> next = P_next;
P_Curr -> next = q_Curr;
P_Curr = P_next;
q_Curr = q_next;
}
*q = q_Curr;
}
int main( )
{
struct Node *P = NULL, *q = NULL;
Push (&P, 0);
Push (&P, 3);
Push (&P, 1);
Printf (" 1ST LINKED LIST \n");
printlist (P);
```

```
Push(&q, 1);
Push(&q, 7);
Push(&q, 2);
Push(&q, 8);
Push(&q, 0);

Printf("2ND LINKED LIST\n");
printlist(q);

merge(P, &q);
printf("CHANGED 1ST LINKED LIST\n");
printlist(P);
printf("CHANGED 2ND LINKED LIST\n");
Printlist(q);
```

OUTPUT

1ST LINKED LIST

1 3 0

2ND LINKED LIST

0 8 2 7 1

CHANGED 1ST LINKED LIST

1 0 3 8 0 2

CHANGED 2ND LINKED LIST

7 1

3. STACK WHOSE SUM IS EQUAL TO C

```
#include<stdio.h>
#define MAX_SIZE 100
int stk1[MAX_SIZE], tops=-1;
int stk2[MAX_SIZE], topH=-1;
int stk1empty()
{
if(tops==-1)
return 1;
else
return 0;
}
int stk1pop()
{
tops--;
}
int stk1top()
{
return stk1[tops];
}
int stk1push(int x)
{
stk1[++tops]=x
}
```

```
int stk2 empty ( )
{
if (topH==-1)
 return 1;
else
  return 0;
}
int stk2 pop ( )
{
topH--;
}
int stk2 top ( )
{
return stk2[tops]
}
int stk2 push (int x)
{
 stk2[++topH]=x;
}
int Sum (int c)
{
int x;
While (stk1 empty( )!=1)
{
x= stk1 top ( );
```

```
stk1pop();
while(stk1empty()!=1)
{
if(x+stk1top()==c)
{
printf("%od,%od\n", x, stk1top());
}
stk2push(stk1top());
stk1pop();
}
while(stk2empty()!=1)
{
stk1push(stk2top());
stk2pop();
}
}
}
int main()
{
int a,i,b,c;
printf("ELEMENTS IN STACK\n");
Scanf("%od",&a);
for(i=0;i<a;i++)
{
Scanf("%od",&b);
stk1push(b);
}
```

```
printf (" SUMATION OF THE SUM\n");
Scanf (" %d", &c);
printf (" SUMATION OF STACK EQUAL TO c\n");
Sum (c);
}
```

OUTPUT

```
ELEMENTS IN STACK
3
1
7
8
SUMATION OF THE SUM
15
SUMATION OF STACK EQUAL TO C
(8,7)
```

4.

ii QUEUE IN ALTERNATE ORDER

```
#include <stdio.h>
#define MAX_SIZE 100
int que[MAX_SIZE], front=-1, rear=-1;
int quepush(int x)
{
if (front==-1)
{
que[++rear]=x;
front++;
}
else
que[++rear]=x;
}
int quepop()
{
front++;
}
int quefront()
{
return que[front];
}
int queempty()
{
```

```
if (front > rear)
return 1;
else
return 0;
}
int main()
{
int a,i,b;
Printf(" ELEMENTS IN QUEUE");
Scanf("%d",&a);
for (i=0;i<a,i++)
{
Scanf("%d",&b);
quepush(b);
}
i=0;
While (queempty()!=1)
{
if (i%2==0)
printf(" %d", quefront());
i++
quepop();
}
}
```

OUTPUT

ELEMENTS IN QUEUE 6

082
123
130
513
156
056

82130156