P. Hema Sri Harshitha
AP19110010130
CSE-G

ASSINGMENT - 4

1. Program to insert and delete an element at n^{th} and K^{Th} Position.

```
#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 (structnode * before - del)
{
Struct node * temp;
temp = before - del → next;
before - del → next = temp → next;
free (temp);
}
```

```
Structnode * front (struct node * head, int value)
{
struct node * P;
P = malloc (size of (structnode));
P -> data = value;
P -> next = head;
return (P);
}
end (struct node * head, int value)
{
struct node *P, * q;
P = malloc (size of (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 (size of (struct node));
P -> data = value
P -> next = a -> next;
```

```
a -> next = P;
}
else
{
printf ("End function is used to insert at the end \n");
}
}
int main ()
{
struct node * prev, * head, * P;
int a, i;
printf ("The number of elements");
scanf ("%d", &a);
head = NULL;
for (i=0; i<a; i++)
{
P = malloc (size of (struct node));
scanf ("%d", & p -> data);
P -> next = Null;
if (head == Null)
head = P;
else
prev -> next = p;
Prev = P;
}
head = front (head, 70);
end (head, 40);
after (head -> next -> next, 80);
del (head -> next);
```

```
del (head -> next -> next);
display (head);
return 0;
}
```

Output:

```
The numbers of elements 5
100
250
310
472
514
70
100
40
472
514
80
Null
```

2. New linked list by merging Alternative 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 88 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, 2 );
Push (&P, 7 );
Push (&P, 4 );
Printf (" The first linked list \n");
printlist (p);
Push (&q, 3 );
Push (&q, 2 );
Push (&q, 1 );
Printf (" The Second Linked list \n");
```

```
Print list (q);
merge (P&q);
printf ("The modified linked list \n");
Print list (P);
return 0;
}
```

Output:

The first linked list

4 7 2

The second linked list

1 2 3

The modified linked list

4 1 7 2 2 3

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

```
else
    return 0;
}
int stk & pop ()
{
top H - - ;
}
int stk & top ()
{
return stk & [tops]
}
int stk & push (int x)
{
stk & [++ top H] = x;
}
int sum (int c)
{
int x;
While (stk1 empty ()! = 1)
{
x = stk 1 top ();
stk1 pop ();
While (stk1 empty ()! = 1)
{
if (x + stk1 top () = = c)
{
Printf ("%.d, %.d \n", x, stk1 top ());
}
```

```
Stk 2 push (stk 1 top ());
Stk 1 pop ();
}
While (stk 2 empty ()!=1)
{
Stk 1 push (Stk 2 top ());
Stk 2 pop ();
}
}
}
int main ()
{
int a, i, b, c;
printf ("The elements in stack \n");
Scanf ("%.d", &a);
for (i=0; i<a; i++)
{
Scanf ("%.d", &b);
Stk 1 push (b);
}
Printf ("The Sumation of Sum \n");
Scanf ("%.d", &c);
Printf ("The Sumation of Stack is equal to c \n");
Sum (c);
}
```

Output:

The element in stack

4

10
20
30
40

The sumation of sum

50

The sumation of stack is equal to C

(30, 20) (10, 40)

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 Stack Node * top, int element);
struct queue * enQueue (struct queue * q, int num);
int deQueue (struct queue * * q);
int pop (struct stack Node * * s);
int main (void)
{
Struct queue * Q = Null;
```

```
Q = enQueue (Q, 11);
Q = enQueue (Q, 77);
Q = enQueue (Q, 20);
Q = enQueue (Q, 551);
Q = enQueue (Q, 110);
Q = enQueue (Q, 1);

Printer (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 Stack Node * push (Struct StackNode * top, int element)
{
Struct StackNode * temp = (Struct StackNode *) malloc
                          (Size of (Struct StackNode));

if (! temp)
{
Printf ("The Overflow of Stack");
return top;
}
```

```
temp -> data = element;
temp -> next = top;
return temp;
}
Struct queue * enQueue (Struct queue * q, int num)
{
Struct node * temp = (Structnode *) malloc (sizeof (Structnode));
temp -> data = num;
temp -> next = null;
if (q == null)
{
q = (Struct queue *) malloc (Size of (struct queue));
if (! q)
{
qprintf ("The exception of Overflow");
return Null;
}
q -> font = temp;
}
else
q -> rear -> nxt = temp;
q -> rear = temp;
return q;
}
Int deQueue (Struct queue * * q)
{
```

```
int x = (*q) -> front -> data;
Structnode * temp = (*q) -> front;
(*q) -> front = (*q) -> front -> next;
free (temp);
return x;
}
int pop (Struct Stack Node * * S)
{
int x = (*S) -> data;
Struct Stack Node * 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;
}
Print ("\n");
}
```

Output:

11 77 20 551 110 1

1 110 551 20 77 11

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 (" The 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:

The elements in Queue 6

6

254
78
94
157
848
222

254 94 848

5. i) How array is different from the linked list

ARRAY	LINKED LIST
→ Size of an array is fixed	→ Size of a list is not fixed.
→ It occupies less memory than a linked list for the same number of elements.	→ It occupies more memory.
→ Deleting an element from an array is not possible.	→ Deleting an element is possible.
→ Insertion and deletion take more time.	→ Insertion and deletion process take less time.

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

```
#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 (size of
                                    (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, 1);
Push (&P, 7);
Push (&P, 4);
Printf ("The first linked list is \n");
Printlist (P);
Push (&q, 4);
Push (&q, 6);
Push (&q, 3);
Push (&q, 2);
Push (&q, 1);
Printf ("The Second linked list is \n")
Print list (q);
merge (P, &q);
Printf ("The modified linked list first is \n");
Printlist (P);
```

```
Printf ("The modified linked list second is \n");
Printlist (q);
```

Output:

```
The first linked list
  4  7  1
The second linked list
  1  2  3  6  4
The modified linked list first is
  4  1  7  2  1  3
The modified linked list second is
  6  4
```