20-4-20

J. Naveen
CSE-H
Ap19110010378

DSA - Assignment - 4

① => Insert a node at n^{th} position in linked list.

```
//insert at nth position
#include <stdio.h>
#include <stdlib.h>
struct Johny {
}*p;

struct Node {
  int data;
  struct Node *next;
};

struct Node *head;
void print()
{
  struct Node *temp = head;
  while (temp != NULL)
  {
    printf("%d", temp->data);
    temp = temp->next;
  }
  printf("\n");
}
void Insert(int data, int n)
{
```

```
int i;
struct Node * temp1 = (struct Node*) malloc(sizeof
                                    (struct Node));

temp1->data = data;
temp1 -> next = NULL;
if(n==1)
{
  temp1->next = head;
   head = temp1;
   return;
}
struct Node * temp2 = head;
for (i=0 ;i<n-2; i++).
{
  temp2 = temp2 -> next;
}
temp1 -> next = temp2 -> next;
temp2 -> next = temp1;
}
int main()
{
 head = NULL; // empty list
 insert (2, 1); // list : 2
 Insert (3,2); // list : 2, 3
 insert (4,1); // list : 4, 2, 3
 Insert (5,2); // list : 4, 5, 2, 3
 print();
 return 0;
}
```

output:

4 5 2 3

① Delete an element at nth position in a linked list.

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node * next;
};
struct Node *head;
void Insert(int data); // Insert an integer at the end of list
void print(); // print all elements in the list
void Delete(int n); // Delete at position n.

int main()
{
    head = NULL; // List is first empty.
    Insert(2);   // List : 2
    Insert(6);   // List : 2, 6
    Insert(8);   // List : 2, 6, 8
    Insert(7); // List 2, 6, 8, 7
    printf("List is: ");
    print();
    int n;
    printf("Enter a position:");
    scanf("%d", &n);
    Delete(n);
    print();

    return 0;
}
void Delete(n)
{
```

```
int i;
struct Node * temp1 = head;
if (n==1){
 head = temp1->next; // head points to second Node
  free(temp1);
  return;
}
for(i=0; i<n-2; i++)
{
  temp1 = temp1->next;
}// temp1 points to (n-1)th Node
void insert (int n)
{
  int i;
  struct Node * temp = (struct Node*) malloc (size of (struct node));
  temp->data = n;
  if(head == NULL);
  {temp->next = head;
    head = temp;
    return;
  }
  if(temp2->next == NULL)
  {
   temp2->next = temp;
   break;
  }
  temp2 = temp2->next;
  }

}
 void print()
{
 struct Node* temp = head;
  while (temp != NULL)
  {
```

```
        printf ("%d", temp -> data);
        temp = temp -> next;
    }
}
```

output:

```
List is : 2 6 8 7
Enter a position : 1
6 8 7
```

(a) construct a new linkest list by merging alternate nodes of two lists for example we have {1,2,3} in list1 and {4,5,6} in list2 we should get {4,1,2,3} as output for list1 & {5,6} as list2.

→

```
#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_current = p, * q_current = *q;
  struct Node *p_next = *q_next;
  while (p_current != NULL && q_current != NULL)
  {
    p_next = p_current->next;
    q_next = q_current->next;
    q_current->next = p_next;
    q_current->next = q_current;
    p_current = p_next;
    q_current = q_next;
  }
  *q = q_current;
}
int main()
{
  struct Node *p = NULL, *q = NULL;
  push(&p, 3);
  push(&p, 2);
  push(&p, 1);
  Printf("First linked list:\n");
  Print list(p);
  push(&q, 8);
  push(&q, 7);
  push(&q, 6);
  push(&q, 5);
  push(&q, 4);
  printf("second list:\n");
  Print list(q);
  merge(p, &q);
  printf("Modified first list:\n");
  Print list(p);
  Printf("Modified second list:\n");
  Print list(q);
  getchar();
  return 0;
}
```

output:

```
First linked list:
1 2 3
Second linked list:
4 5 6 7 8
Modified First linked list:
1 4 2 5 3 6
Modified 2nd list:
7 8
```

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

```
#include <stdio.h>
 int top = -1;
 int x;
 char stack[100];
 void push(int x);
 char pop();
 int main()
 {
int i, n, a, t, k, f, sum=0, count=1;
printf("enter the number of elements in the stack");
scanf("%d", &n);
for (i=0 ;i<n; i++){
 printf("enter next element");
 scanf("%d", &a);
 push(a);
 }
printf ("enter the sum to be checked");
 scanf("%d", &k);
 for(i=0; i<n; i++)
 {
 t=pop();
 sumt=t;
count++;
 if (sum==k){
for(int j=0; j<count; j++)
Print("%d", stack[j]);
 f=1;
break;
}
```

```
push(t);
}
if(f!=1)
printf("the elements in the stack dont add up
                to the sum");
}
void push(int x).
{
if(top==99)
{
printf("\n stack is FULL!!!\n");
return;
}
top= top+1;
stack[top] =x;
}
char pop()
{
if (stack[top]==-1)
{
printf("\n stack is EMPTY!!!\n");
return 0;
}
x= stack[top];
top= top-1;
      return x;
}
```

output:

```
Enter the numbers of elements in stack
                                    :6
Enter next element: 6
Enter next element: 7
Enter next element: 8
Enter next element: 12
Enter next element: 87
Enter next element: 45
Enter the sum to be checked
the elements in the stack dont
add up to the sum
```

④ (P) Write a program to print the elements in queue in reverse order:

→ // function to reverse the elements of queue.

```
void reverse queue (queue <int> & Queue)
{
    stack<int> stack;
    while (!queue.empty()) {
     stack.push (Queue.front());
     queue.pop();
    }
    while (! stack.empty()) {
      queue.push (stack.top());
      stack.pop();
    }
}
```

Input:

3 1 4 8 1 4 1

output:

1 4 1 8 4 1 3

45) Program to print the elements in a queue in reverse order.

```
void reverse queue first K elements (int K, queue <int> &queue)
{
  if (queue.empty().==true || K > queue.size())
    return;
  if (K<=0)
    return;
  stack<int> stack;
  for (int i=0; i<K; i++) {
      stack.push (queue.front());
      queue.pop();
  }
  while (!stack.empty()) {
     queue.push (stack.top());
     stack.pop();
  }
  for (int i=0; i< queue.size() - K; i++) {
  queue.push (queue.front());
  queue.pop();
  }
}
```

output:

5 4 3 2 1 6

output:

5 4 3 6 1 2

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

Ans: Differences between Array and Linked list

Array	Linked list
1. It has fixed size	1. It has dynamic size.
2. Elements are usually shifted.	2. Elements are Not shifted.
3. It has Random access i.e. efficient indexing where each element is associated with an index.	3. No random access. It is Not suitable for operations requiring accessing elements by Index like sorting.
4. There is wastage of memory in the array as if the array is full or almost full there is no wastage but in other case it has lots of wastage.	4. No wastage of memory because the memory is allocated dynamically (according to the need).

(5)(ii) Write a program to point or add the first element of one list to another list.

```
#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");
}
```

Output :

Enter the List 1 : 1 2 3

Enter the List 2 : 4 5 6

modified list 1 : 4 1 2 3

Modified List 2 : 5 6