DSA Assignment - 4

P. Saswanth Roy
AP19110010524
CSE-H

Write a Program to insert and delete an element at the n^{th} and k^{th} position in a linked list where n and k is taken from user.

Program :-

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
 struct node * next
};
struct node * cur, * temp;
void input (struct node)
void delete (struct node)
void main (void)
{
struct node * s;
int n;
s = null;
do
{
   Printf ("\n enter the element to insert :");
   Printf (" 2. Delete \n");
   Printf (" 3. Exit \n");
   Printf (" enter the choice :");
   Scanf (" %d", &n);
        switch (n)
        {
```

Scanned with CamScanner

```
            case 1: input (s);
                    break;
            case 2: delete (s);
                    break;
           }
          while (n != 3)
     }
void input (struct node * z)
{
   int pos, c = 1;
   curr = z;
   Printf ("enter the element to be inserted:");
   scanf ("%d", &pos);
   while (curr -> next != Null)
   {
     c++;
     if (c == pos)
     {
       temp = (struct node *) malloc (sizeof (struct node));
       Printf ("enter the numbers:");
       scanf ("%d", &temp -> n);
       temp -> next = curr -> next;
       curr -> next = temp;
       break;
      }
   }
}
void delete (struct node* z)
{
  int pos, c = 1;
```

Scanned with CamScanner

```
        curr = 2;
        printf("enter the element to be delete:");
        scanf("%d", &Pos);
        while (curr -> next != NULL)
        {
          C++;
          if (C == Pos)
          {
            temp = curr -> Next;
            curr -> next = curr -> next -> next;
            free (temp)
          }
          curr = curr -> next;
        }
    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;
```

Scanned with CamScanner

```
Push (&p, 1);
Push (&p, 2);
Push (&p, 3);
Printf (" First linked linst : \n ");
Print list (p);
Push (&q, 4);
Push (&q, 5);
Push (&q, 6);
Printf (" Second linked list : ");
Print list (q);
merge (p, &q);
Printf (" modified first linked list = ");
Print list (p);
Printf (" modified second linked list =");
Print list (q);
return 0;
}
```

Scanned with CamScanner

2) Construct a new linked list by merging of ~~two lists~~ alternate nodes of the two linked lists for example if the linked list 1. having node {1, 2, 3} and linked list 2. having {4, 5, 6} then by merging we should get {1, 4, 2, 5, 3, 6}

Ans:- Program:-

```
#include <stdio.h>
#include <stdlib.h>
Struct node
{
    int data;
    struct node * next;
};
Void Printlist (Struct node * head)
{
    struct node * Ptr = head;
    while (Ptr)
    {
        Printf ("%d" -> ", Ptr -> data);
        Ptr = Ptr -> next;
    }
    Printf ("null");
}
Void Push (Struct node ** head, int data)
{
  struct node * new node = (Struct node*)
                                malloc (sizeof (Struct node));
  newnode -> data = data;
  new node -> next = *head;
  *head = newnode;
```

Scanned with CamScanner

```
}
struct node * Shuffle merge (struct node * a, struct node * b)
{
   struct node dummy;
   struct node * tail = &dummy;
   dummy.next = null;
   While (1)
   {
      if (a == null)
      {
          tail -> next = b;
          break;
      }
      else if (b == null)
      {
         tail -> next = a;
         break;
      }
      else
      {
         tail -> next = a;
         tail = a;
         a = a -> next;
         tail -> next = b;
         tail = b;
         b = b -> next;
      }
   }
   return dummy.next;
}
```

Scanned with CamScanner

```
int main()
{
    int Keys[] = {1,2,3,4,5,6} = {1,4,2,5,3,6}
    int n = sizeof (Keys) / sizeof (Keys[0]);

    struct node *a = null, *b = null;
    for (i=n-1; i>=0; i=i-2)
        Push (&a, Keys[i]);
    for (i=n-2; i>=0; i=i-2)
        Push (&b, Keys[i]);
    Printf ("First list: ");
    Printlist (a);
    Printf ("Second list: ");
    Printlist (b);
    struct node* head = shuffle merge (a,b);
    Printf ("After merge: ");
    Print list (head);
    return 0;
}
```

Output :-

First list : 1 -> 2 -> 3 -> null

second list : 4 -> 5 -> 6 -> null

After merge : 1 -> 4 -> 2 -> 5 -> 3 -> 6 -> null

Scanned with CamScanner

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

Ans: Program :-

```
#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 no. 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();
    sum+=t;
    count+=1;
    if(sum==k)
    {
      for(j=0; j<count; j++)
```

Scanned with CamScanner

```
        Printf ("%d", stack[i]);
        f=1;
        break;
      }
    Push(t);
  }
  if(f!=1)
  Printf("the elements in the stack don't add up to the sum")
}
void Push(int x)
{
  if (top==99)
  {
    Printf("stack is full");
    return;
  }
  top=top+1;
  stack[top]=x;
}
char PoP()
{
  if (stack[top]==-1)
  {
    Printf("stack is empty");
    return 0;
  }
  x=stack[top];
  top=top-1;
  return x;
}
```

OUTPUT :-

```
Enter no. of elements : 3
Enter next element : 7
Enter next element : 14
Enter next element : 10
Enter the sum to be
      checked : 17
7  10
```

Scanned with CamScanner

4). Write a Program to Print the elements in a queue
(i) reverse order
(ii) Alternate order

(i)
```
#include <stdio.h>
#include "stack.h"
int main ()
{
  int n, arr[50], i, j;
  struct stack s;
  int stack (s);
  Printf ("enter the number of elements :");
  scanf ("%d", &n);
  for (i=0; i<n; i++)
  {
    Printf ("enter the values :");
    scanf ("%d", &arr[i]);
  }
  for (i=0; i<n; i++)
  {
    insert (arr[i]);
  }
  While (i!=n)
  {
    Push (&s, def(s));
    i++;
  }
  Printf (" The reverse order is :");
  While (stop != -1)
  {
    Printf ("%d", Pop[&s]);
  }
  Printf ("\n");
} return 0;
```

Scanned with CamScanner

(ii)
```
# include <stdio.h>
# include <stdlib.h>
struct node
{
  int data;
  struct node * next;
}
Void Print nodes (struct node * head)
{
   int count =0;
   while (head != null)
   {
      if (count %. 2 ==0)
      {
        Printf ("%.d", head ->data);
      }
      count ++;
      head = head -> nextt;
   }
}
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;
}
int main ()
{
   struct node* head_ null;
        Push (& head, 22);
        Push (& head, 29);
```

Scanned with CamScanner

```
    Push(&head, 31);
    Push(&head, 52);
    Push(&head, 20);
    return 0;
}
```

i) Output :-

~~20, 52 31 29 22~~

```
Enter no. of element : 3
Enter the elements: 1 2 3
reverse order : 3 2 1
```

ii) output

20 31 22

Scanned with CamScanner

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

Arrays	linked lists
1. Fixed size: Resizing in expensive	1. Dynamic size
2. Insertions and deletions are inefficient: Elements are usually shifted.	2. Insertions and deletions are efficient: No shifting.
3. Random access i.e., efficient indexing	3. No random access → Not suitable for operations requiring accessing elements by index such as sorting.
4. No memory waste if the arrays is full or almost full; otherwise may result in much memory waste.	4. Since memory is allocated dynamically there is no waste of memory.
5. sequential access is faster	5. Sequential access is slow.

Scanned with CamScanner

(ii)

```
#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");
}
```

Output :-

Enter number of nodes in list 1, list 2: 2,3

First list is : 3 -> 2 -> 1 second list is : 2 -> 1

Scanned with CamScanner