Sanchita Rawat
535
CSE-H

Camlin Page
Date / /

Assignment - 4

Q1.
```
#include<stdio.h>
struct node * head = NULL;
struct node
{ int data;
   struct node * next;
 };
void insert (int data)
 { struct node * temp = (struct node *) malloc
  sizeof(struct node));
  temp -> data = data;
  temp -> next = head;
   head = temp;
 }
void insert_at_pos(int d, int pos)
 { struct node *ptr = (struct node *) malloc(siz
 (struct node));
   ptr -> d = d;
   int i;
   struct node * temp = head;
   if (pos == 1)
  {  ptr -> next = temp;
       head = ptr;
   }
   for (i=1; i<pos-1; i++)
  {  temp = temp -> next;
   }
   ptr -> next = temp -> next;
    temp -> next = ptr;
 }
```

Camlin Page
Date / /

```
void print()
{ struct node *temp= head;
  printf("\n List:");
  while(temp != NULL)
  {  printf("\n %d", temp->data);
     temp= temp -> next;
   }
}
 void del()
{ struct node *temp= head;
  struct node * holder;
  if(temp != NULL)
 {  holder= temp ->next;
    free(temp);
    holder= holder;
  }
   else
      printf(" List is empty");
 }
void dprint()
{ ~~while~~ struct node * temp= head;
        printf( List: );
        while ( temp != NULL)
      { printf ("\n %d ", temp->data);
        temp= temp ->~~do~~ next;
      }
 }
void main()
{ insert(4);
  print();
  insert(5);
  print();
  insert (7);
```

Date / /

```
print();
 head = temp;
temp -> data = 4
struct node * o = (struct node *) malloc( sizeof(struct node)) 
struct node *t = (struct node *) malloc(sizeof(struct node));
struct node * k = (struct node *) malloc(sizeof (struct node));
 head = o;
 o -> data = 4;
 t -> data = 5;
 k -> data = 7;
 k -> next = NULL;
print();
 del();
 print();
}
```

2.
```
#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\n");
 }
void move node ( struct node ** dref, struct node **
                                        sref)
```

```
{ if (*sref = =NULL)
    return ;
 struct node *newnode = *source ref ;
  * sref = (*sref )->next;
  newNode -> next= *dref;
   *dref = newnode;
}
 struct node dummy;
  dummy.next = NULL;
  struct node * tail = & dummy;
 while(1)
 { if(a= =NULL)
     { tail->next = b ;
        break; }
   else if ( b= = NULL)
        {tail->next = a;
         break; }
   if (a->data <= b->data)
        movenode(&(tail->next), &a);
   else
        movenode(&(tail->next), &b);
   tail=tail->next ;
  }
   return dummy.next;
 }
```

4.
```
#include<stdio.h>
struct stackrecord
{ int *a;
  int cap;
  int tos;
 };
 typedef struct stackrecord *stack;
 stack createstack (int max)
 { stack s;
   s=malloc(sizeof (struct stackrecord));
   if(s==NULL)
     printf("Out of space");
   s->a=malloc((sizeof(int))*max);
   if(s->a==NULL)
     printf("Out of space);
    s->cap=max-1;
    s->tos=-1;
    return(s);
    int isfulls(stack s)
    { return s->tos==s->capacity;
     }
   void push(int x, stack s)
   { if(isfull(s))
      printf("Overflow");
     else
     { printf("\n%d is pushed",x);
       s->tos++;
       s->a[s->tos]=x;
      }
   }

 struct qrecord
{ int *a;
  int f;
int rear;
int cap;
};
typedef struct queuerecord *queue;
queue createq(int max)
{  queue q;
     q = malloc(sizeof(struct queuerecord));
     if(q==NULL)
     printf("Error")
     q-> array=malloc( sizeof (int) * max);
    if (q->array==NULL)
      q -> cap=max-1;
       q->f=-1;
        q->rear=-1;
       return q;
```

(ii)
```
#include<stdio>
struct node
{  int d;
   struct node *next;
};
void print(struct node * head)
{  int c=0;
   while(head!=NULL)
     {if (c%2==0)
        printf("%d", head->data ");
     }
  }

void main()
{   struct node * head= NULL;
        push(&head, 12);
        push(&head, 29);
      push(&head, 11);
```

Camlin Page
Date / /

```
push(&head, 23);
 print(head);
  }
```

5) The major difference is that array is index based data structure where each element has an index. Linked List relies on references where each node consists of data and reference to the previous and next element.