# Assignment - 4

Y. Nikhil

AP19110010514

1) Write a programme to insert and delete an element at the $n^{th}$ and $k^{th}$ pointer in a linked list where n and k are taken from the uses.

A)

```
#include <stdio.h>
#include <stdlib.h>
struct node {
int data;
struct node * next;
};
struct node * head;
void Insert (int data, int n) {
Node * temp = new node n;
temp → data = data;
temp → next = null;
if (n==1) {
temp → next = head;
    head = temp;
    return;
```

Scanned with CamScanner

```
Void Delete_(int k) {
    struct Node * temp = head;
    if (k==1) {
        head = temp -> next;
    free (temp);
      return;
    }
    Node * temp = head;
    for (int i=0; i<n-2; i++) {
    temp = temp -> next;
    }
     temp -> next = temp -> next;
     temp -> next = temp;
    }
     void print();
for (int i=0; i<k-2; i++)
      temp = temp -> next;
       free (temp);
    }
int main() {
int n, x, k;
```

Scanned with CamScanner

```
head = Null;
printf("Enter the position for inserting:");
scanf("%d", &n);
scanf("%d", &x);
Insert(x, n);
printf("Enter the position to delete");
scanf("%d", &k);
Delete(k);
print(x);
return ;
}
```

2) Construct a new linked list by merging alternative nodes and two lists for example in list 1 we have {1,2} and {2,4,6} and in the new we should have {1,4,2,5,3,6}

A)

```
# include <stdio.h>
# include <stdlib.h>
struct node {
int data;
struct node next;
}
```

Scanned with CamScanner

```
void print list (struct node * head)
{
printf ("%d →", (ptr → data)) ;
    ptr = ptr → next ; }
    printf ("Null/n") ;
}
void push (struct node * head, int dabai)
{
struct node * new = (struct node*) malloc
        (size of (struct node)) ;
new → data = data ;
new → next = * head ;
    * head = new ;
}
struct node * merge (struct node *a , struct node * b)
{
struct node fake ;
struct node * tail = fake ;
  fake. next = Null ;
  while (1) {
if (a == Null)
```

Scanned with CamScanner

```
        break;
    }
    else if (b != NULL)
    {
        fail->next = a;
        break;
    }
    else
    {
        fail->next = a;
        fail = a;
        a = a->next;
        fail->next = b;
    }
}
    return fake.next;
}
void main()
{
    int keys[] = {1,2,3,4,5,6,7}
int n = size of (keys)/ size of key [0]
struct node *a = null; b = null;
for (int i = n-i; i>0; i = i-a)
    push (fa, keys[i]);
```

Scanned with CamScanner

```
for (int i = n-2; i >= 0; i = i-2)
    push(&h, types[i]);
struct node *head = merge(a, h);
print(list(head));
}
```

3) find all the elements in the stack whose sum is equal to 15.

4)
```
#include <stdio.h>
void find(int arr[], int a, int k) {
int total = 0
int x=0, y=0;
while (total < k && y < a)
    total = arr[y]
        y++;
if (total == 0)
printf("find");
return;
total = arr[x];
}
```

Scanned with CamScanner

```
int main (void) {
    int arr[] = {9, 10, 12, 4, 1, 23}
    int k = 565;
    int a = size of (arr) / size of (arr[0]);
    find (arr, a, k);
    return 0;
}
```

4) A)

```
#include < stdio.h >
# define size 20
void insert (int);
void delete();
int queue [20], a = -1, b = -1,
void main() {
int num; choice;
while(1) {
printf ("\n * New \n");
printf ("1. Insert \n 2. delete \n 3. Print \n
        4. Reverse \n 5. Alternate \n 6. Exit");
printf("\n Enter your choice");
```

Scanned with CamScanner

```
scanf ("%d", &choice);
switch (choice) {
case 1: printf ("Enter the num to insert: ");
        scanf ("%d", &num);
        insert (num);
    break;
case 2: printf ("Reverse queue");
        for (int i=size, i>=0, i--)
          if (queue[i]==-0)
          continue;
        printf ("%d", queue[i]);
    }
    break;
case 3:
        printf ("Alternate elements");
        for (int i=0, i<size, i>0, i++2)
    {
        if (queue[i]=0)
            continue;
        printf ("%d", queue[i]);
    }
    break;
return 0;
}
```

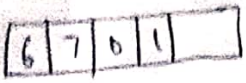
Scanned with CamScanner

1) (1) Array vs linked lists

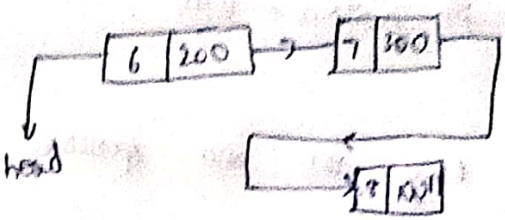
1) Both are the data structure. Both are used to store the data.

2) Cost of accessing the elements

Arrays




→ it takes at constant time O(1)



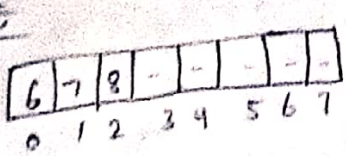

→ it depends on number of nodes in the linked list O(n)

2) Memory requirement and utilization.

Array

→ ineffective in memory utilization

Ex:



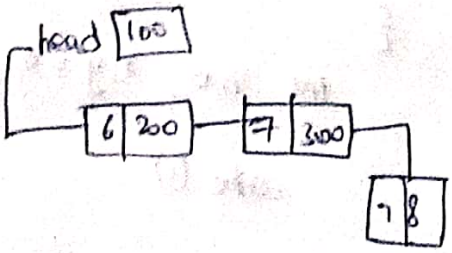

8 x 4 = 32 bytes

Used = 12

→ Requires memory in less

linked list

→ it is in dynamic byte




8 x 3 = 24 bytes.

2) More requirement

Scanned with CamScanner

4) Cost of insertion and cost of deletion

| | Array | linked list |
|---|---|---|
| Beginning | $O(n)$ | $O(1)$ |
| At end | $O(1)$ | $O(n)$ |
| ith position | $O(n)$ | $O(n)$ |

5) Easy, use and operations.

| Array | linked list |
|---|---|
| → easier to use | → less easier |
| → linear and binary | → linear |

(ii)
```
#include <stdio.h>
#include <stdlib.h>
int len[int a()]
{
  int i=0, x, y=0
    while (i)
    {
      if (x[i])
      {
        y++, i++;
      }
    else
      {
        break;
      }
```

Scanned with CamScanner

```
    }
    return xy;
}
void change list(int x[], int a[])
    {
        for (int i = len(x)-1, i>=0, i--)
        {
            x[i+1] = x[i]
        }
        x[0] = a[0];
        printf("\n Elements of old array (n);
for (int i=0; i< len(x); i++)
    {
        printf("%d", x[i]);
    }
for (int i=0, i< len(y); i++)
    {
        y[i] = y[i+1];
    }
printf("\n Elements of new array: \n")
for (int i=0; i< len[a]; i++)
{
    printf("\n %d", a[i]);
```

Scanned with CamScanner

```
int main()
{
    int x[10] = {1, 2, 3},  a[10], {4, 5, 6};
    change list = (a, b);
}
```

Scanned with CamScanner