DSA Assignment.

B. Deepika
CSE-G
AP19110010245

1. Program to insert and delete an element at n^{th} & k^{th} position in linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct linked_list
{ int num;
  struct linked_list *next;
};
typed of struct linked_list node;
node *head = NULL; * last = NULL;
Void create_linked_list ( )
void print_linked_list();
void insert_ab_list (int value);
void insert_after (int key, int value);
void delete_item ( int value);
Void search_item (int value);
int main ( )
{ int key, value;
  // create a linked list.
  printf ("Create linked list \n");
    create_linked_list ( );
    print_linked_list ( );
// Insert value at position to existing linked list.
```

```
printf("\n Enter a value to insert at last \n");
scanf("%d", &value);
insert_at_last(value);
print_linked_list();
// Insert value at first position to existing linked list.
printf("\n Insert new item at first \n");
scanf("%d", &value);
print_linked_list();
// Insert value after a defined value
printf("\n Enter a key (existing item of list), after that you want to insert a value\n");
Scanf("%d", &key);
printf("\n Insert new item after %d key\n", key);
scanf("%d", &value);
insert_after(key, value);
print_linked_list();
// search an item from linked list
printf("\n Enter an item to search it from list \n");
scanf("%d", &value);
search_item(value);
// Delete value from linked list.
printf("\n Enter a value, which you want to delete\n");
scanf("%d", value);
delete_item(value);
print_linked_list();
return 0;
} /* user defined functions */
```

```
Void create_linked_list ( )
{ int val;
  while ( )
  {
    printf("\nInput a number linked -1 to exit)\n");
    scanf ("%d", &value);
    if (val==-1)
      break;
    insert_at_last (val);
  }
}
Void insert_at_last (int value)
{ node* temp_node;
  temp_node = (node*) malloc (size of (nodes));
  temp_node → number = value;
  temp_node → next   = NULL;
  // For the 1st element
  if (head == NULL)
  { head = temp_node;
    last = temp_node;
  }
else
  { last → next = temp_node;
    last = temp_node;
  }
}
Void insert_after (int key, int value)
  { node* my node = head;
```

```
    int flag = 0.,
while (mynode!, = NULL)
{ if (my Node → Number = = Key)
{
  node* newnode = (node*) malloc(size of node));
  Newnode → number = value;
  Newnode → next = my Node → next;
  My Node → next → new node;
printf (" %d is inserted after %d \n",
                                value, key);
    flag = 1.,
    break,
}
else
my Node = my Node → Next;
}
If (flag = = 0)
  printf(" Key not found. \n");
}
Void delete_item (int value)
{ node *myNode = head, * previous = NULL;
    int flag = 0;
    while (my Node != NULL)
{ if (mynode → number = = value)
{ if (previous = = NULL)
      head = myNode → next;
```

```
else
  previous -> next = mynode -> next;
printf("%d is deleted from list \n", value);
      flag = 1;
      free (my node);
       break;
      }
    previous = my node;
    my node = my Node -> Next;
   }
if (flag == 0)
    printf("key not found! \n");
    }

Void print_linked_list()

{ printf ("\n your full linked list is \n");
     node* mylist;
      mylist = head;
     while ( mylist != NULL )
    { printf ("%d", mylist -> number);
       my list = mylist -> next;
          }
          puts (" ");
      }
```

Output:

Create a linked list

Input- a number - (Enter - 1 to exit)

1

Input a number. - (Enter - 1 to exit)

2

Input a number. - (Enter - 1 to exit)

3

Input a number. - (Enter - 1 to exit)

4

Input a number - (Enter - 1 to exit)

5

Input a number (Enter - 1 to exit)

-1

Your full linked list is

1 2 3 4 5 6

Insert new item at first.

0

Your full linked list is

0 1 2 3 4 5 6

Enter a key (existing item of list),

after what you have inserted a value.

6

Insert a new item.

7

7 is inserted after 6.

Your full linked list is

0 1 2 3 4 5 6 7

Enter a number to search from list.

3

3 is identified in list. Memory address is

1234867

Enter a value, which you want to delete from list

6

6 is deleted from the list.

Your full linked list is

0 1 2 3 4 5 7.

2.) Construct a new linked list by merging alternate nodes of two lists.

```
#include <stdio.h>
#include <stdlib.h>
// Data structure to store a linked list
Struct Node
{ int data;
  struct Node* next;
  };
Void print list(struct node* head)
{ struct Node* ptr= head;
  while(ptr)
  { printf("%d → ", ptr→data);
    ptr= ptr→ next;
  }
  printf("NULL\n");
  //Insert new node in beginning
Void push(struct Node** head, int data)
{ struct Node* new node = (struct Node*) malloc
                                  (size of
                                  struct
                                  Node);
```

```
New Node -> data = data;
New Node -> next = *head;

*head = newNode;
}
// function to construct a linked list by merging
                                    alternate node.
// two given linked lists using dist node
struct Node* shuffle merge (struct node* a,
                            struct Node* b)
{
  struct node dibal;
  struct Node* tail = &dibal;
   dibal.next = NULL;
   while(1)
  {
    // empty list cases.
    if (a == NULL)
    {
      tail->next = b;
      break;
    }
   else if(b == NULL)
    {
      tail->next = a;
          break;
    }
// move two nodes to tail
  else
    {
      tail->next = a;
      tail = a;
          a = a->next;
      tail->next = b;
          tail = b;
```

```
        b= b → next;
    }
    }
    return dibal.next;
}
int main(void)
{ int keys[] = {1,2,3,4,5,6,7};
  int n = size of (keys) / size of keys[0];
  struct node* a = NULL, *b = NULL.
  for (int i = n-1; i >= 0; i = i-2)
        push (&a, keys[i]);
  for (int i = n-2; i >= 0; i = i-2)
        push (&b, keys[i]);
  printf (" first list:");
    print list (a);
  printf (" second list:")
    print list (b);
  Struct Node* head = shuffle merge (a,b)
    printf (" After Merge:");
    print list (head);
    return 0;
}
```

Output:

first list: 1 → 3 → 5 → 7 → null

second list: 2 → 4 → 6 → null

After Merge: 1 → 2 → 3 → 4 → 5 → 6 → 7 → null.

3)

```
#include <stdio.h>
int stack[100], choice, n, top, x, i;
void push(void);
void display(void);
int main()
{
  top = -1;
printf("\n Enter the size of stack:");
scanf("%d", &n);
printf("\n\t Stack operations using ARRAY");
printf("\n\t 1. PUSH\n\t 2. DISPLAY\n\t 3. Sub
\n\t 4
Exit
do
{ printf("\n Enter the choice:");
  scanf("%d", &choice);
  switch(choice)
{ Case 1:
  { push()
    break;
  }
  Case 2:
  { display()
    break;
  }
  Case 3:
  { sub-Arraysum()
    break;
  }
Case 4:
```

```
{ printf ("\n\t exit point");
    break;
  }
  default:
  {
    printf("\n\t please enter a valid choice
                                (1/2/3/4");
    }
  }
 }
while (choice != 4)
  return 0;
}
Void push()
 { if (top >= n-1)
  { printf("\n\t stack is over flow");
  }
  else
  { printf("enter a value to be pushed");
    scanf("%d", &x);
    top++
    stack (top) = x;
  }
  }
Void display ()
 { if (top >= 0)
  { printf("\n the elements in stack \n");
    for (i= top; i>=0; i--)
    printf("\n %d", stack[i]);
   printf("\n press next choice");
      }else
```

```
    }
    printf("\n is the stack is empty");
  }
}
int subarraysum(int stack[7, int sum)
{
  int curr_sum, i, j;
  scanf("%d", &sum);
  for(i=0; i<n; i++)
  {curr_sum = stack[i] + stack[j];
  //try all sub arrays starting with i
    for(j=i+1; j<=n; j++)
    { if (curr_sum == sum)
    {
      printf("sum found %d and %d",
                    stack[i], stack[j]);
        return 1;
      }
if (curr_sum > sum || j==n)
      break;
  curr_sum = curr_sum + stack[j];
    }
}
printf("No sub array found");
    return 0;
  }
int main()
{
  int sum = 23;
  subarraysum(stack, n, sum);
    return 0;
  }
```

Output:

1. PUSH
2. DISPLAY
3. SUBARRAY
4. EXIT

enter choice: 1

enter a value to be pushed:

1

enter choice: 1

enter a value to be pushed:

2

Enter choice: 1

Enetel a value to be pushed:

3

enter choice: 1

enter a value to be pushed:

4

enter choice. 2

The elements in stack.

1
2
3
4.

press next choice. 3

3

Sum found 1, 2.

4.) Implementation of queue in reverse order.

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

#define max 20.
Void show( int stack[], int size, int top)
        { int i;
          for (i=0; i<size; i++)
```

```
{ printf ("\n value at %d is %d top, stack[top]);
    top = top - 1;
  }
}
void reverse (int stack[], int *t, int *f)
{ *f = 0;
  while (*t > -1)
  { *r = *r + 1;
    qu[*r] = stack[*t];
    *t = *t - 1;
  }
  while (*f <= *r)
  { *t = *t + 1;
    stack[*t] = qu[*f];
    *f = *f + 1;
  }
}
int main()
{ int size
  int item, t, i, stack[max], queue[max];
  int top = -1, front = -1, rear = -1;
  printf ("Enter size of stack");
  scanf ("%d", &size);
  for (i=0; i<size; i++)
  { top = top + 1;
    printf ("Enter value of for position %d", i);
    scanf ("%d", &item);
    stack[top] = item;
```

```
show (stack, size, top);
reverse (stack, queue, &top, &rear, &front);
printf("\n After reverse ---");
show(stack, size, top);
getch ();
}
```

Output:

```
Enter size of stack : 5
Enter value of for position 0 : 1
Enter value of for position 1 : 2
Enter value of for position 2 : 3
Enter value of for position 3 : 4
Enter value of for position 4 : 5
Value of 4 is 5
Value at 3 is 4
Value at 2 is 3
Value at 1 is 2
Value at 0 is 1

After reverse ----
Value at 4 is 1
Value at 3 is 2
Value at 2 is 3
Value at 1 is 4
Value at 0 is 5
```

(11) Program to print elements in a queue in alternate order.

```
#include <stdio.h>
#define MAX 50
```

```
Void insert ( );
Void alternate ( );
Void display ( );
int queue_array [max];
    int rear = -1,
    int front = -1, size;
    Scanf ("%d", &size);
        main ( )
    { int choice;
      while (1)
    { printf ("1. Insert elements to queue \n");
      printf ("2. Display elements from queue \n");
      printf ("3. Alternate elements");
      printf ("4. Quit \n");
      Printf ("Enter your choice:");
        scanf ("%d", &choice);
          switch (choice)
        { case 1:
            insert ( );
            break;

case 2:
    Display ( );
      break;
  case 3:
    alternate ( );
      break;
  case 4:
    Exit (1);
      default:
```

```
        printf("Wrong choice \n");
        }
        }
    }
Void insert()
{   int add_item;
    if (rear==max-1)
    printf("reverse overflow \n");
     else.
{   if (front==-1)
     front=0.,
    printf ("Insert the element in Queue");
      scanf ("%d", &add_item);
        rear= rear+1;
      queue_array[rear] = add_item.,
        }
    }
Void display()
  { int i;
    if (front ==-1)
     printf("Queue is empty is empty \n");
    else
    { printf("Queue is:)n");
      for(i=front; i<=rear; i++)
            printf("%d", queue_array[i]);
            printf("\n");
      }
    }
```

```
Void alternate()
{ int i, j, temp;
printf("\nAlternate elements are \n");
    for(i=0; i< size; i+=2)
     printf("%d\n", queue_array[i]);
}
```

Output:

Enter choice: 1

Insert the element in queue: 10.

Enter choice: 1

Insert the element in queue: 20

Enter choice: 1

Insert the element in queue: 30

Enter choice: 1

Insert the element in queue: 90

Enter choice: 1

Insert the element in queue: 50

Enter choice: 2

10
20
30
40
50

Enter choice: 3

10
30
50

Enter choice: 9

Exit

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

Array	Linked List
- An array is a collection of elements of a similar data type.	- Linked lists is an ordered collection of elements of same type in which each element is connected to next using pointers
- Array elements can be accessed randomly using the array index.	- Random accessing is not possible in linked lists the elements will have to be accesed.
- Data elements are stored in configuous locations in memory.	- New elements can be stored anywhere & reference is created.

(ii) Program to add first node of linked list to another linked list.

```
#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 push(struct Node** head, int data)
{ struct Node* New node = (struct node*) malloc
                          (size of (struct
                                    Node));
    new node -> data = data;
    New Node -> next = *head;
     * head = new node;
    }
// function take the node from the front of source.
// and move it to front of destination
Void moveNode (struct node** destRef, struct Node**
{                                      sourceref)
    if(* source ref == NULL)
        return;
    struct Node* New Node = * sourceref;
     * sourceref = (* source ref) -> next;
        new node -> next = * dest Ref;
     * dest Ref = new node;
    }
    int main (void)
    {
    int keys[] = {1, 2, 3}
    int n = size of (keys) / size of (keys[0]);
        struct Node* a = NULL;
        for and i = n-1; i>=0; i--)
```

```
// Construct 2nd linked list.
struct Node *b = NULL;
for (int i=0; i<n; i++)
    push(&b, keys[i]);
// move front node of b and move it to the front of a
MoveNode(&a, &b)
  printf("First list:");
    printlist(a);
printf("Second list:");
    printlist(b);
        return 0;
}
```

Output:

First list: 6 → 1 → 2 → 3 → null.

Second list: 4 → 2 → null.