Assignment-1

Q-1

write a c Program to scan two NxN matrices from the user. Perform addition of these matrices and store to the another matrices use a single 3D dynamically generated array via pointer for the same.

—>

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

int main()
{
    int N;
    Printf("Enter the size of the matrices (N) : ");
    scanf("%d", &N);

    int *** matrices = (int **) malloc( 3 * sizeof( int **));

    if (matrices == NULL)
    {
        Printf(" memory allocation failed \n");
        return 1;
    }
    for (int i=0; i<3; i++)
    {
        matrices[i] = int *a malloc(N * sizeof-(int *));
        if (matrices[i] == NULL)
        {
            Printf(" memory allocation failed \n");
            return 1;
```

```
for (int j=0; j<N; j++)
{
    matrices[i][j] = (int*) malloc (N*sizeof(int));
    if (matrice[i][j] == NULL){
        Printf("memory allocation is failed \n");
        return 1;
    }
  }
}

Printf("enter elements of first matrix : \n");
for(int i=0; i<N; i++),
{
    for (int j=0; j<N; j++)
    {
        Printf("enter element of 1 matrix [%d][%d] :"
        i,j);
        Scanf("%d", &matrices[0][i][j]);
    }
}

Printf("enter element of second matrix: \n");
for (int i=0; i<N; i++)
{
    for(int j=0; j<N; j++)
    {
        Printf("enter elment of 2 matrix [%d][%d]:"
        i,j);
        Scanf("%d", &matrices[1][i][j]);
    }
}
}
```

```
// addiction of matrics

for (int i=0; i<N ;i++)
{
   for (int j=0; j<N;j++)
   {
     matrices[2][i][j] = matrices[0][i][j] +
                         matrice [1][i][j];
   }
}
// Print matrices addiction
Printf(" matrices addiction is : \n");
for (int i=0; i<N; i++)
{
   for (int j=0; j<N; j++)
   {
      printf(" %d", matrices[2][i][j]);
   }
   Printf("\n");
}
// Free all the dynamic memory
for (int i=0; i<3 ; i++)
{
   for (int j=0; j<3; j++)
   {
      free(matrices[i][j]);
   }
   free(matrices[i]);
}
```

```
}free(matrices)
}
return 0;
```

output!

input: 1 2 3
4 5 6
7 8 9

10 20 30
40 50 60
70 80 90

output!

11 22 33
44 55 66
77 88 99

Q-2 Write a program to sort a stack using a temporary stack. Built a stack as input from the user

Ex:-
Input: [34, 3, 31, 98, 92, 23]
Output: [3, 23, 31, 34, 92, 98]

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

struct stack
{
    int capacity;
    int top;
    int *array;
};
struct stack *createStack(int capacity)
{
    struct stack *stack = (struct stack *) malloc
                          (sizeof(struct stack));
    if (!stack)
    {
        return NULL;
    }
    stack->capacity = capacity;
    stack->top = -1;
    stack->array = (int *) malloc(stack->capacity *
                                  sizeof(int));
```

```
if (!stack -> array)
{
    return NULL; }
return stack;
}
int IsEmpty(struct stack *stack)
{
    return stack -> top == -1;
}
int is_full (struct stack *stack)
{
    return stack -> top == stack -> capacity - 1;
}

void Push(struct stack *struct, int item)
{
    if (IsFull(stack))
        return;
    Stack -> array [++stack -> top] = item;
}
int Pop (struct stack *stack)
{
    if (isEmpty(stack))
        return -1;
    return stack -> array [stack -> top--];
}
```

```
int Peek(struct stack *stack)
{
    if(isEmpty(stack))
        return -1;
    return stack->array[stack->top];
}
void sortStack(struct stack *stack)
{
    struct stack *tempstack = createStack(stack->capacity);
    while(!isEmpty(stack))
    {
        int temp = Pop(stack);
        while(!isEmpty(tempstack) && Peek(tempstack) >
            temp)
        {
            Push(stack, Pop(tempstack));
        }
        Push(tempstack, temp);
    }
    while(!isEmpty(tempstack))
    {
        Push(stack, Pop(tempstack));
    }
    free(tempstack);
}
int main()
{
```

```
int capacity, num;
Printf("enter the capacity of the stack : ");
Scanf("%d", &capacity);
struct stack * stack = createstack(capacity);

Printf("enter %d elements to Push onto the
    stack : \n", capacity);
for(int i=0; i<capacity ; i++)
{
  Printf("enter stack element %d : ", i+1);
  Scanf("%d", &num);
  Push(stack, num);
}
Printf("\n original stack : ");
for(int i=0 ; i<capacity ; i++)
{
  Printf("%d", stack -> array[i]);
}
SortStack(stack);
Printf("\n sorted stack : ");
while( !isEmpty(stack))
{
    Printf("%d", Pop(stack));
}
free(stack -> array);
free(stack);
return 0;
}
```

→ Output :-

input [34, 3, 31, 98, 92, 23]
output [3, 23, 21, 31, 34, 92, 98]

input: [-3, 2, -2, 0, 9, 10]
output: [-3, -2, 0, 2, 9, 10]

Q-3 write a program to print the characters of the string in sorted order using stack.

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

Struct stack {
    int top;
    unsigned capacity;
    char *array;
};
struct stack *createstack(unsigned capacity)
{
    struct stack *stack = (struct stack *) malloc
        (sizeof(struct stack));
    stack -> capacity = capacity;
    stack -> top = -1;
    stack -> array = (char *) malloc (stack ->
        capacity * sizeof(char));
    return stack; }
```

```
int isfull(struct stack *stack)
{
  return stack->top == stack->capacity - 1;
}
int Isempty(struct stack * stack)
{
  return stack->top == -1;
}
void push(struct struct *stack, char item)
  {
    if (isfull(stack))
      return;
    stack->array[++stack->top] = item;
  }
char pop(struct stack *stack)
  {
    if(isempty(stack))
      return '\0';
    return stack->array[stack->top--];
  }
void Printstack(struct stack *stack)
  {
    while(!isempty(stack))
      {
        Printf("%c", pop(stack));
      }
  }
```

```
void sortstring(char *str)
{
    int len = strlen(str);
    struct stack *mainstack = createstack(len);
    struct stack *auxstack = createstack(len);
    push(mainstack, str[0]);
    for(int i=0; i<len; i++)
    {
        char current = str[i];
        while(!isempty(mainstack) && current >
              mainstack->array[mainstack->top])
              {
                push(auxstack, pop(mainstack));
              }
              push(mainstack, current);
              while(!isempty(auxstack))
              {
                push(mainstack, pop(auxstack));
              }
    }
    printf("output :");
    printstack(mainstack);
    free(mainstack->array);
    free(mainstack);
    free(auxstack->array);
    free(auxstack);
}
```

```
int main()
{
    char str[100];
    printf("Input : ");
    fgets(str, sizeof(str), stdin);
    str[strcspn(str, "\n")] = '\0';
    sortstring(str);
    return 0;
}
```

Output:

input: hello3568&%worldl234
output: 123456789&%abcdehllloorw.

4) Write a program to reverse a stack using recursion (you are not allowed to use loop constructs like while, for, etc). Use the fuction mentioned below.

- isEmpty(s)
- Push(s)
- Pop(s)

→

```
#include<stdio.h>
#include<stdlib.h>
struct stack {
        int top;
        unsigned capacity;
        int *array;
};
```

```
struct stack *createStack(unsigned capacity)
{
    struct stack *stack = (struct stack *) malloc
            (sizeof(struct stack));
    stack -> capacity = capacity;
    stack -> top = -1;
    stack -> array = (int *) malloc(stack -> capacity *
            sizeof(int));
    return stack;
}
int isFull(struct Stack *stack)
{
    return stack -> top == stack -> capacity - 1;
}
int isEmpty(struct stack *stack)
{
    return stack -> top == -1;
}
void Push(struct stack *stack, int item)
{
    if (isFull(stack))
        return;
    stack -> array[++stack -> top] = item;
}
```

11).

```
int Pop(struct stack *stack)
{
  if (isEmpty(stack))
    return -1;
  return stack->array[stack->top--];
}
void insertAtBottom(struct stack *stack, int item)
{
  if(isEmpty(stack))
{
    Push(stack, item);
}else{
    int temp= Pop(stack);
    insertAtBottom(stack, item);
    Push(stack, temp);
  }
}
void reverseStack(struct stack *stack)
{
  if(!isEmpty(stack))
  {
    int temp = Pop(stack);
    reverseStack(stack);
    insertAtBottom(stack, temp);
  }
}
```

```
int main()
{
    int capacity, num;
    Printf("enter the capacity of the stacks:");
    Scanf("%d", &capacity);
    Stack stack = createstack(capacity);
    Printf("enter %d element to push into the stack:\n",
        capacity);
    for(int i=0; i<capacity; i++)
    {
        Printf("enter stack element %d:", i+1);
        Scanf("%d", &num);
        Push(stack, num);
    }
    Printf("\n original stack:\n");
    for(int i=0; i<capacity; i++)
    {
        Printf("%d", stack->array[i]);
    }
    reversestack(stack);
    Printf("\n Reversed stack:\n");
    for(int i=0; i<capacity; i++)
    {
        Printf("%d", stack->array[i]);
    }
    free(stack->array);
    free(stack);
    return 0;
}
```

Output:-

Size: 5
Enter 5 element
Element 1 : 20
element 2 : -3
element 3 : -5
element 4 : 10
element 5 : 29

original stack : 20, -3, -5, 10, 29
Reversed stack : 29 10 -5 -3 20

(5) Write a Program to sort a stack using recursion case of any loop constructs like while, for, etc) is not allowed

Input: -3, 14, 18, -5, 30
Output: 30, 18, 14, -3, -5

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

struct StackNode{
    int data;
    struct StackNode * next;
};
struct StackNode * newNode( int data)
{
    struct StackNode * stackNode = (struct StackNode*)
                malloc(sizeof(struct StackNode));
    stackNode -> data = data;
    stackNode -> next = NULL;
    return stackNode;
}
int isEmpty(struct StackNode * root)
{
    return root;
}
```

```
int isEmpty(struct StackNode *root)
{
    return !root;
}
void push(struct StackNode **root, int data)
{
    struct StackNode *stackNode = newNode(data);
    stackNode->next = *root;
    *root = stackNode;
}
int pop(struct StackNode **root)
{
    if (isEmpty(*root))
        return -1;
    struct StackNode *temp = *root;
    *root = (*root)->next;
    int popped = temp->data;
    free(temp);
    return popped;
}
void sortedInsert(struct StackNode **root,
                  int data)
{
    if (isEmpty(*root) || data < (*root)->data)
    {
        push(root, data);
        return;
    }
}
```

```
    int temp = pop(root);
    sortedInsert(root, data);
    push(root, temp);
}

void sortstack(struct stackNode** root)
{
    if (!isEmpty(*root))
    {
        int temp = pop(root);
        sortstack(root);
        sortedInsert(root, temp);
    }
}

void printstack(struct stackNode* root)
{
    if (isEmpty(root))
        return;
    int data = pop(&root);
    printstack(root);
    printf("%d", data);
    push(&root, data);
}
int main()
{
    struct stackNode* root = NULL;
    int size, num;
    printf("enter the number of elements:\n");
    scanf("%d", &size);
```

```
Printf("Enter %d elements:\n", size);
for(int i=0; i<size; i++){
    Printf("enter stack element %d:", i+1);
    Scanf("%d", &num);
    Push(&top, num);
}
Printf("In original stack:\n");
PrintStack(top);

SortStack(&top);
Printf("In sorted stack:\n");
PrintStack(top);
Printf("\n");
return 0;
}
```

Outputs:

```
Enter size: 5
Element 1 : 10
Element 2 : -20
Element 3 : 20
Element 4 : 30
Element 5 : 0

original stack: 10 -20 20 30 0
Sorted stack: 30 20 10 0 -20
```

6) You are given a string s consisting of lowercase english letter. A duplicate remove consists of choosing two adjacent and equal letter and removing them we repeatedly duplicate removes on s until we no longer can write a program to return the final string after all such duplicate removals have been made.

Input - s = abbaca
output - : ca

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

struct Stack{
    int top;
    unsign capacity;
    char *array;
};
struct stack *createStack(unsigned capacity)
{
    struct Stack *stack = (struct Stack *) malloc
                          (sizeof (struct Stack));
    stack -> capacity = capacity;
    Stack -> top = -1;
    Stack -> array = (char *) malloc (stack ->
                     capacity * sizeof (char));
    return Stack; }
```

```
int isfull(struct Stack *Stack)
{
    return stack->top == Stack->capacity - 1;
}
int Empty(struct Stack *Stack)
{
    return Stack->top == -1;
}
void Push(struct stack *Stack, char Item)
{
    if (IsFull(Stack))
    {   return;
    }
    Stack->array[++Stack->top] = item;
}
char pop(struct Stack *Stack)
{
    if(Empty(Stack))
        return '\0';
    return Stack->array[Stack->top--];
}

char *removeDuplicates(char *s)
{
    int len = strlen(s);
    struct Stack *Stack = CreateStack(len);
```

```
for(int i=0; i<len; i++)
{
  if(!IsEmpty(stack) && stack->array[stack->top]
      == s[i])
  {
    Pop(stack);
  }else
  { Push(stack,s[i]);
  }
}
char *result = (char *) malloc(stack->top+2) *
                        sizeof(char));
int index=0;
while(!IsEmpty(stack))
{
  result[index++]=Pop(stack);
}
result[index]='\0';
int i=0; i!=index-1;
while(i<1);
{
  char temp=result[i];
  result[i]=result[j];
  result[j]=temp;
  i++;
  j--;
}
free(stack->array);
free(stack); return result; }
```

```
int main()
{
    char s[100];
    printf("Enter the string : ");
    fgets(s, sizeof(s), stdin);
    s[strcspn(s, "\n")] = '\0';

    char *result = removeDuplicates(s);
    printf("output : %s\n", result);
    free(result);
    return 0;
}
```

output

input : abbaccd
output : cd

7) write a program to implement queue using stacks?

→
```
#include<stdio.h>
#include<stdlib.h>
#define max_size 100
struct stack
{
    int top;
    int array[max_size];
};
```

```
Struct Stack *CreateStack()
{
  Struct Stack *Stack = (Struct Stack *) malloc
                        (sizeof (struct Stack));
  Stack->top = -1;
  return Stack;
}
Int isEmpty (struct Stack * Stack)
{
  return Stack -> top == -1;
}
int Full (struct Stack * Stack)
{
  return Stack -> top == max_size - 1;
}
Void Push (struct Stack* Stack, int item)
{
  If (Full (Stack))
  {
    Printf("Stack Underflow \n");
    return -1;
  }
  return Stack->array [Stack->top--];
}
int Peek (Struct Stack *Stack)
{
  If (IsEmpty (Stack))
    return -1;
  return Stack->array [Stack->top]; }
```

```
void InsertAtBottom(struct stack * stack, int data)
{
    if(IsEmpty(stack))
    {
        Push(stack, data);
    }
    else
    {
        int temp = Pop(stack);
        ReverseStack(stack);
        InsertAtBottom(stack, temp);
    }
}

Struct Queue * CreateAQueue(){
    struct queue * queue = (struct queue *) malloc
                           (sizeof(struct queue));
    queue->stack = CreateStack();
    return queue;
}

void enqueue(struct queue *queue, int data)
{
    ReverseStack(queue->stack);
    Push(queue->stack, data);
    ReverseStack(queue->stack);
}

int dequeue(struct Queue * queue)
{
    if(IsEmpty(queue->stack))
    {
        Printf("Queue is empty\n");
        return -1;
    }
```

```
    return Pop(queue->stack1);
}
void display(struct Queue *queue)
{
    if(isEmpty(queue->stack1))
    {
        printf("Queue is empty\n");
        return;
    }
    printf("Queue : ");
    struct Stack *tempStack = createStack();
    while(!isEmpty(queue->stack1))
    {
        int data = Pop(queue->stack1);
        printf("%d", data);
        Push(tempStack, data);
    }
    while(!isEmpty(tempStack))
    {
        Push(queue->stack1, Pop(tempStack));
    }
    printf("\n");
    free(tempStack);
}
```

```
int main()
{
    struct queue* queue = createQueue();
    int choice, data;
    do{
        printf("\n Queue Operation \n");
        printf(" 1. Enqueue \n");
        printf(" 2. Dequeue \n");
        printf(" 3. Display \n");
        printf(" 4. Exit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1:
                printf("Enter data to enqueue : ");
                scanf("%d", &data);
                enqueue(queue, data);
                printf("%d enqueued to the queue \n",
                data);
                break;
            case 2:
                data = dequeue(queue);
                if (data != -1)
                    printf("%d dequeued from the queue \n",
                        data);
                break;
```

queue.

```
case 3:
    display(queue);
    break;
case 4:
    printf("Exiting ---\n");
    break;
default:
    printf("Invalid choice Plase enter a valid
        operation \n");
}
} while(choice != 4);
free(queue->stack);
free(queue);
return 0;
}
```

output:-

Enter choise:-1
Enter data to enqueue:10
Enter choise:1
Enter data to enqueue:20
Enter choise:1
Enter data to enqueue:30

Enter choise:3
Queue 10 20 30

Ente choise:2
Enter choise:3
Queue: 20 30

8. Write a program to create a queue where insertion and deletion of elements can be done from both the ends.

```
-> #include <stdio.h>
   #include <stdlib.h>
   #define max_size 100

   struct Deque {
     int array[max_size];
     int front;
     int rear;
   };

   struct Deque* createDeque() {
       struct Deque * deque = (struct Deque *)
                              malloc (sizeof (struct Deque));
       deque -> front = -1;
       deque -> rear = -1;
       return deque;
     }
     int isEmpty (struct deque * deque)
     {
       return (deque -> front == -1 && deque -> rear == -1);
     }
     int isFull (struct Deque * deque)
     {
       return ((deque -> rear + 1) % max_size == deque ->
               front); }
```

```
void insertfront (struct Deque * deque, int data)
{
  if (isfull (deque))
  {
    Printf("Deque is full. cannot insert at front \n")
    return;
  }else if (isEmpty(deque))
  {
    deque->front =0;
    deque->rear = 0;
  }else if (deque -> front ==0)
    {
      deque ->front = max. size -1;
    }
    else{
        deque -> front = (deque ->front -1) %. max-
                size;
        }
    deque -> array [deque -> front] = data;
    Printf("%d inserted at the front \n", data);
  }
void insertRear (struct Deque * deque, int data)
{
  if(isfull (deque))
  {
    Printf(" Deque is full cannot insert at. rear \n");
    return;
  }
```

```
else if (isEmpty(deque))
{
    deque->front = 0;
    deque->rear = 0;
}
else {
    deque->rear = (deque->rear+1) % max_size;
}
deque->array[deque->rear] = data;
printf("%d Inserted at the rear \n", data);
}
void deleteFront(struct Deque *deque)
{
    if(isEmpty(deque))
    {
        printf("Deque is empty; cannot delete from
            front \n");
        return;
    } else if(deque->front == deque->rear)
    {
        printf("%d deleted from the front \n",
            deque->array[deque->front]);
        deque->front = -1;
        deque->rear = -1;
    }
    else
    { printf("%d deleted from the front \n",
        deque->array[deque->front]); }
```

```
Void deleterear(struct Deque * deque);
{
  if (isempty(deque))
  {
    Printf("Deque is empty cannot delete from
      rear\n");
    return;
  }
  else if (deque->front==2*deque->rear)
  {
    Printf("%d deleted from the rear\n",
      deque->array[deque->rear]);
    deque->front = -1;
    deque->rear = -1;
  } else if(deque->rear ==0)
    {
      Printf("%d deleted from the rear\n",
      deque->array[deque->rear]);
      deque->rear = max_size-1;
    } else
      Printf("%d deleted from the rear\n",
          deque->array[deque->rear]);
      deque->rear = (deque->rear-1)%max_size;
    }
}
```

```
void display (struct Deque * deque)
{
  if (isEmpty (deque))
  {
   printf (" Deque is empty \n");
   return;
  }
  Print (" Deque : ");
  int i = deque -> front;
  while (i != deque -> rear)
    {
      Printf("%d", deque -> array [i]);
      i = (i+1) % deque -> max-size;
    }
    Printf(" %d \n", deque -> array [i]);
}
int main()
{
  Struct Deque * deque = createDeque();
  int choice, data;
  do {
     Printf("\n Deque operations : \n");
     Printf(" 1. Insert at front \n");
     Printf(" 2. Insert at rear \n");
     Printf(" 3. Deleted from front \n");
     Printf(" 4. Deleted from rear \n");
```

```
Printf("5. display\n");
Printf("6. Exit\n");
Printf("Enter your choice : ");
Scanf("%d", &choice);
switch(choice)
{
    case 1:
        Printf("Enter data to insert at front : ");
        Scanf("%d", &data);
        insertfront(deque, data);
        break;
    case 2:
        Printf("Enter data to insert at rear : ");
        Scanf("%d", &data);
        insertRear(deque, data);
        break;
    case 3:
        deletefront(deque);
        break;
    case 4:
        deleterear(deque);
        break;
    case 5:
        display(deque);
        break;
    case 6:
        Printf("Exiting....\n");
        break;
```

```
default :
    Printf("Invalid choice ! Please enter a
        valid option \n");
    }
} while (choise != 0);
free(dequeue);
return 0;
}
```

output :-

```
Choise = 1
Enter data 10
Choise = 1
Enter data 20
Enter choise : 5
Queue : 20 10
Enter choise : 1
Enter data : 30
Enter choi = 2
Enter data : 40
Enter choise : 5
Queue : 20 10 30 40
Enter choise : 3
delet from front 20
choise : 4
delet from rear 40
choise -> 5
Queue : 10 30
```

9) The file system crawler keeps a log each time some user performs a change folder operation
-The operations are described below:
- "../" move to the parent folder of the current folder.
- "./" Remain in the same folder.
- "x/" move to the child folder named x.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define Max_size 100
struct Stacks
    int top;
    char * array[Max_size];
};
struct Stacks * CreateStacks()
{
   Struct Stacks * Stacks = (struct Stacks*)
                            malloc(sizeof(struct
                            Stacks));
   Stacks->top = -1;
   return Stacks
}
int IsEmpty(struct Stacks * Stacks)
{
    return Stacks->top == -1;
}
```

```
int isfull(struct stacks * stack)
{
    return stack->top == max_size-1;
}
void push(struct stacks * stack, char * folder)
{
    if(isfull(stack))
    { printf("stack overflow\n");
      return;
    }
    stack->array[++stack->top] = strdup(folder);
}
char * pop(struct stacks * stack)
{
    if(isempty(stack))
    {
        printf("stack underflow\n");
        return null;
    }
    return stack->array[stack->top--];
}
char * peek(struct stacks * stack)
{
    if(isempty(stack))
      return NULL
    return stack->array[stack->top];
}
```

```
void freestack(struct stack* s)
{
    while(!isEmpty(s))
    {
        free(pop(s));
    }
    free(s);
}

int main()
{
    struct stack* s = createStack();
    char operation[10];

    printf("Welcome to file system crawler\n");
    printf("Supported operations:\n");
    printf("- '..': move to the parent folder\n");
    printf("- '.': Remain in the same folder\n");
    printf("- 'x': move to the child folder
                named x\n");
    printf("- 'exit': exit the program\n\n");
    printf("current folder: /\n");
    printf("Enter file system operation\n");
    while(1)
    {
        scanf("%s", operation);
        if(strcmp(operation, "exit")==0)
            break;
```

```
if (strcmp(operation, "../") == 0)
{
  if (!isEmpty(stack))
      pop(stack);
} else if (strcmp(operation, "./") != 0)
  {
    push(stack, operation);
  }
  printf("current folder: ");
  if (isEmpty(stack))
    printf("/");
  else {
     for (int i=0; i<= stack->top; ++i)
     {
        printf("%s/", stack->array[i]);
     }
  }
  printf("\n");
}
free stack(stack);
printf("Exiting file system crawler
       Goodbye!\n");
return 0;
}
```

Output:-

Input: "d1/", "d2/", "./", "d3/", "../", "d31/")

Output 2 3

10) Write a Program to implement following stack operations using arrays with max element. Use an integer array At the start of the Program look for a command line argument for filename - if the file exists read file and load the stack and then start the Program else create a new stack and continue with the Program.
- Push, - Peep, - Change, - Empty
- Pop, - Display, - Quit, ~~- Ensure the bina~~

→
```
#include <stdio.h>
#include <stdlib.h>
#define max_size 100

struct stack {
    int top;
    int array [max_size];
};

struct stack* createStack()
{
  struct stack* stack = (struct stack*)
                malloc (sizeof (struct stack));
  stack->top = -1;
  return stack;
}
```

```
int IsEmpty(struct stack *stack)
{
    return stack->top == -1;
}
int IsFull(struct stack *stack)
{
    return stack->top == maxsize - 1;
}
void Push(struct stack *stack, int item)
{
    if(IsFull(stack))
    {
        Printf("stack overflow\n");
        return;
    }
    stack->array[++stack->top] = item;
    Printf("%d Pushed onto the stack\n", item);
}
int Pop(struct stack *stack)
{
    if(IsEmpty(stack))
    {
        Printf("stack underflow\n");
        return -1;
    }
    return stack->array[stack->top--];
}
```

```
int peek(struct stack * stack)
{
    if(isempty(stack))
      return -1;
    return stack -> array[stack -> top];
}
void display(struct stack* stack)
{
    if(isempty(stack))
    {
    printf("stack is empty\n");
    return;
    }
    printf("stack:\n");
    for(int i=0; i<=stack -> top; ++i);
    {
      printf("%d", stack -> array[i]);
    }
    printf("\n");
    }
}
void savefile(struct stack* stack, const
                                char* filename)
{
    FILE * file = fopen(filename, "w");
    if(file == NULL)
}
```

```
        Printf("Error opening file for writing \n");
        return;
    }
    for (int i=0; i<=stack->top; ++i)
    {
        fprintf(file, "%d \n", stack->array[i]);
    }
    fclose(file);
    Printf("Stack saved to file successfully \n");
}
void loadFromFile(struct Stack* stack, const
                  char* filename)
{
    FILE* file = fopen(filename, "r");
    if (file == NULL)
    {
        Printf("File not found or error opening
                file for reading \n");
        return;
    }
    int item;
    while (fscanf(file, "%d", &item) != EOF)
    {
        Push(stack, item);
    }
    fclose(file);
    Printf("Stack loaded from file
            successfully \n");
}
```

```
void freestack (struct stacks * stack)
{
    freestack(stack);
}
int main (int argc; char * argv[])
{
    struct Stack * stack;
    if (argc != 2)
    {
        printf("Usage: %s <filename>\n", argv[0]);
        return 1;
    }
    stack = createstack();
    loadfromfile (stack, argv[1]);
    int choice, data
    do {
        printf("\n stack operations :\n");
        printf("1. Push \n");
        printf("2. Pop \n");
        printf("3. Peep \n");
        printf("4. Display \n");
        printf("5. Save to file \n");
        printf("6. is full ? \n");
        printf("7. is empty ? \n");
        printf("8. Exit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
```

```
switch (choice)
{
  case 1:
    Printf("Enter data to Push: ");
    Scanf("%d", &data);
    Push(Stack, data);
    break;
  case 2:
    data = Pop(Stack);
    if (data != -1)
      Printf("Popped element: %d\n",
        data);
    break;
  case 3:
    data = Peek(Stack);
    if (data != -1)
      Printf("Top element: %d\n", data);
    break;
  case 4:
    Display(Stack);
    break;
  case 5:
    save_To_File(Stack, argv[1]);
    break;
  case 6:
    if (IsFull(Stack))
      Printf("Stack is full\n");
    else
      Printf("Stack is not full\n");
```

```
case 7:
    if (isEmpty(stack))
      printf("stack is empty\n");
    else
      printf("stack is not empty");
    break;
case 8:
    printf("Exiting...\n");
    break;
default:
    printf("Invalid choice Please enter
        a valid option.\n");
  }
}while(choice!=8);
freestack(stack);
return 0;
}
```

Output:- Create my

choice 1:
enter element: 10
choice 2: 1
enter element: 20
choice 3: 1
enter element: 30
choice: 2
~~enter item~~ Poped element 30
choice: 3
Of elemt 20

choice 4
10 20
Choice: 5
File saved.
Choice: 6
Stack is empty.
Choice: 7
Stack is empty.