Exercise 13

Aim:

To implement Stack storage allocation strategies using C program.

Algorithm:

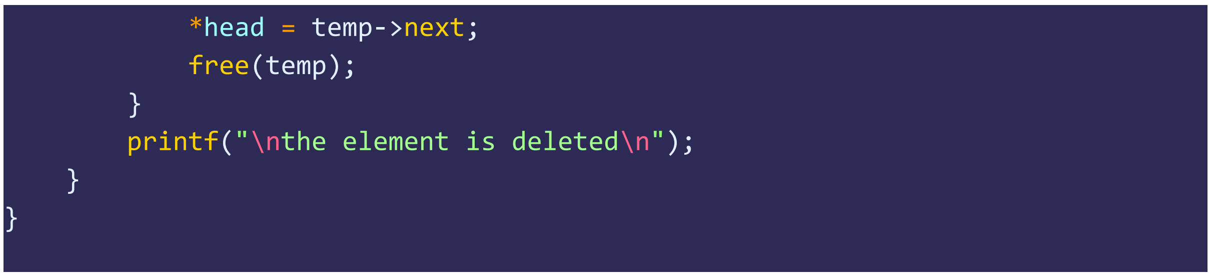
Step1: Initially check whether the stack is empty
Step2: Insert an element into the stack using push operation
Step3: Insert more elements onto the stack until stack becomes full
Step4: Delete an element from the stack using pop operation
Step5: Display the elements in the stack
Step6: Top the stack element will be displayed

Code :

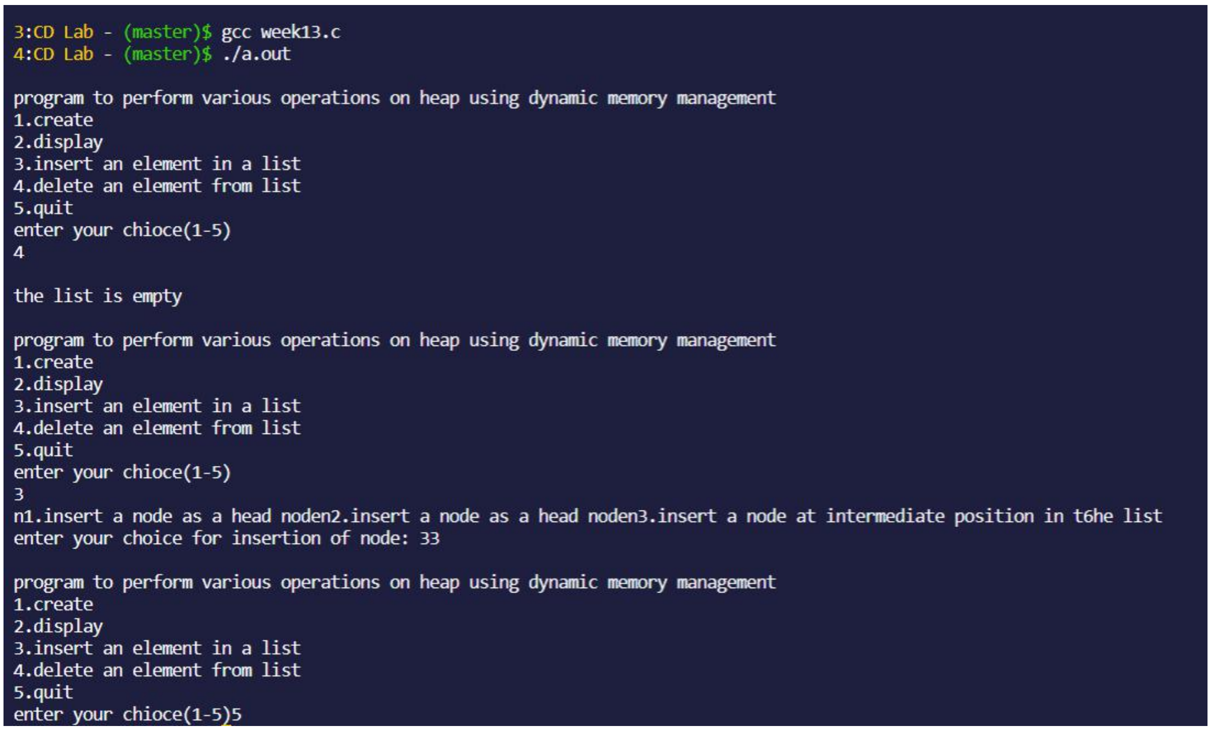
```
#include <stdio.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
typedef struct Heap
{
    int data;
    struct Heap *next;
} node;
node *create();
void main()
{
    int choice, val;
    char ans;
    node *head;
    void display(node *);
    node *search(node *, int);
    node *insert(node *);
    void dele(node **);
    head = NULL;
    do
    {
        printf("\nprogram to perform various operations on heap using dynamic
memory management");
        printf("\n1.create");
        printf("\n2.display");
        printf("\n3.insert an element in a list");
        printf("\n4.delete an element from list");
        printf("\n5.quit");
        printf("\nenter your chioce(1-5)");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            head = create();
            break;
        case 2:
            display(head);
            break;
        case 3:
            head = insert(head);
            break;
        case 4:
            dele(&head);
            break;
        case 5:
            exit(0);
        default:
            printf("invalid choice,try again");
        }
    } while (choice != 5);
}
node *create()
{
    node *temp, *New, *head;
    int val, flag;
    char ans = 'y';
    node *get_node();
    temp = NULL;
    flag = TRUE;
    do
    {
        printf("\n enter the element:");
        scanf("%d", &val);
        New = get_node();
        if (New == NULL)
            printf("\nmemory is not allocated");
        New->data = val;
        if (flag == TRUE)
        {
            head = New;
            temp = head;
            flag = FALSE;
        }
        else
        {
            temp->next = New;
            temp = New;
        }
        printf("\ndo you want to enter more elements?(y/n)");
    } while (ans == 'y');
    printf("\nthe list is created\n");
    return head;
}
node *get_node()
{
    node *temp;
    temp = (node *)malloc(sizeof(node));
    temp->next = NULL;
    return temp;
}
void display(node *head)
{
    node *temp;
    temp = head;
    if (temp == NULL)
    {
        printf("\nthe list is empty\n");
        return;
    }
    while (temp != NULL)
    {
        printf("%d->", temp->data);
        temp = temp->next;
    }
    printf("NULL");
}
node *search(node *head, int key)
{
    node *temp;
    int found;
    temp = head;
    if (temp == NULL)
    {
        printf("the linked list is empty\n");
        return NULL;
    }
    found = FALSE;
    while (temp != NULL && found == FALSE)
    {
        if (temp->data != key)
            temp = temp->next;
        else
            found = TRUE;
    }
    if (found == TRUE)
    {
        printf("\nthe element is present in the list\n");
        return temp;
    }
    else
    {
        printf("the element is not present in the list\n");
        return NULL;
    }
}
node *insert(node *head)
{
    int choice;
    node *insert_head(node *);
    void insert_after(node *);
    void insert_last(node *);
    printf("n1.insert a node as a head node");
    printf("n2.insert a node as a head node");
    printf("n3.insert a node at intermediate position in t6he list");
    printf("\nenter your choice for insertion of node:");
    scanf("%d", &choice);
    switch (choice)
    {
    case 1:
        head = insert_head(head);
        break;
    case 2:
        insert_last(head);
        break;
    case 3:
        insert_after(head);
        break;
    }
    return head;
}
node *insert_head(node *head)
{
    node *New, *temp;
    New = get_node();
    printf("\nEnter the element which you want to insert");
    scanf("%d", &New->data);
    if (head == NULL)
        head = New;
    else
    {
        temp = head;
        New->next = temp;
        head = New;
```

```c
    }
    return head;
}
void insert_last(node *head)
{
    node *New, *temp;
    New = get_node();
    printf("\nenter the element which you want to insert");
    scanf("%d", &New->data);
    if (head == NULL)
        head = New;
    else
    {
        temp = head;
        while (temp->next != NULL)
            temp = temp->next;
        temp->next = New;
        New->next = NULL;
    }
}
void insert_after(node *head)
{
    int key;
    node *New, *temp;
    New = get_node();
    printf("\nenter the elements which you want to insert");
    scanf("%d", &New->data);
    if (head == NULL)
    {
        head = New;
    }
    else
    {
        printf("\enter the element which you want to insert the node");
        scanf("%d", &key);
        temp = head;
        do
        {
            if (temp->data == key)
            {
                New->next - temp->next;
                temp->next = New;
                return;
            }
            else
                temp = temp->next;
        } while (temp != NULL);
    }
```

```
}
node *get_prev(node *head, int val)
{
    node *temp, *prev;
    int flag;
    temp = head;
    if (temp == NULL)
        return NULL;
    flag = FALSE;
    prev = NULL;
    while (temp != NULL && !flag)
    {
        if (temp->data != val)
        {
            prev = temp;
            temp = temp->next;
        }
        else
            flag = TRUE;
    }
    if (flag)
        return prev;
    else
        return NULL;
}
void dele(node **head)
{
    node *temp, *prev;
    int key;
    temp = *head;
    if (temp == NULL)
    {
        printf("\nthe list is empty\n");
        return;
    }
    printf("\nenter the element you want to delete:");
    scanf("%d", &key);
    temp = search(*head, key);
    if (temp != NULL)
    {
        prev = get_prev(*head, key);
        if (prev != NULL)
        {
            prev->next = temp->next;
            free(temp);
        }
        else
        {
```



```
            *head = temp->next;
            free(temp);
        }
        printf("\nthe element is deleted\n");
    }
}
```



```
3:CD Lab - (master)$ gcc week13.c
4:CD Lab - (master)$ ./a.out

program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)
4

the list is empty

program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)
3
n1.insert a node as a head noden2.insert a node as a head noden3.insert a node at intermediate position in t6he list
enter your choice for insertion of node: 33

program to perform various operations on heap using dynamic memory management
1.create
2.display
3.insert an element in a list
4.delete an element from list
5.quit
enter your chioce(1-5)5
```

Result:
we have successfully implemented stack storage allocation using heap.