In-Lab – 1(page No: 119)

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

struct Node* top = NULL;

void push(int ele)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if(newNode == NULL)
   {
      printf("Stack Overflow. Cannot push: %d\n", ele);
      return;
   }
   newNode->data = ele;
   newNode->next = top;
   top = newNode;
   printf("Pushed: %d\n", ele);
}
int pop()
{
   if(top == NULL)
   {
      printf("Stack is empty. Cannot pop.\n");
      return -1;
   }
   int ele = top->data;
   struct Node* temp = top;
   top = top->next;
   free(temp);
   printf("Popped: %d\n", ele);
   return ele;
}
int peek()
{
   if(top == NULL)
   {
      printf("Stack is empty. Cannot peek.\n");
      return -1;
```

```
    }
    printf("Peeked: %d\n", top->data);
    return top->data;
}

int isEmpty()
{
    return top == NULL;
}
int main()
{
    push(1);
    push(2);
    push(3);

    peek();
    pop();
    pop();
    pop();
    pop();
    peek();
    return 0;
}
```

In-Lab – 2(page No: 121)

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

struct Node* top = NULL;

void push(int value)
{
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (newNode == NULL)
    {
        printf("Stack Overflow. Cannot push: %d\n", value);
        return;
    }
    newNode->data = value;
```

```
    newNode->next = top;
    top = newNode;
}
void pop()
{
    if (top == NULL)
    {
        printf("Stack is Empty! Deletion is not possible.\n");
        return;
    }
    printf("%d\n", top->data);
    struct Node* temp = top;
    top = top->next;
    free(temp);
}
int topfun()
{
    if (top != NULL)
        return top->data;
    else
        return -1;
}
int getMin()
{
    struct Node *temp;
    temp=top;
    if(temp ==NULL) return -1;
    int min=top->data;
    while(temp!=NULL)
    {
        if(temp->data<min)
            min=temp->data;
        temp=temp->next;
    }
    return min;
}
int main()
{
    int n;
    scanf("%d", &n);
    char operation[10];
    int value;
    for (int i = 0; i < n; i++)
    {
```

```
        scanf(" %s", operation);
        if(strcmp(operation, "push") == 0)
        {
            scanf(" %d", &value);
            push(value);
        }
        else if(strcmp(operation, "pop") == 0)
        {
            pop();
        }
        else if(strcmp(operation, "top") == 0)
        {
            printf("%d\n", topfun());
        }
        else if(strcmp(operation, "getMin") == 0)
        {
            printf("%d\n", getMin());
        }
        else
        {
            printf("Invalid operation!\n");
        }
    }
    return 0;
}
```

In-Lab – 3(page No: 123)

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

#define MAX_SIZE 101
char a[MAX_SIZE];
int top = -1;

void push(char ele)
{
    if(top<=MAX_SIZE-1)
    {
        a[++top]=ele;
    }
    else
    {
        printf("Stack is full. Cannot push: %c\n",ele);
```

```
    }
}

char pop()
{
    if(top>=0)
    {
        char ele = a[top];
        top--;
        return ele;
    }
    else
    {
        printf("Stack is empty. Cannot pop.\n");
        return '-1';
    }
}
int isEmpty()
{
    return top==-1;
}
int isFull()
{
    return top>=MAX_SIZE;
}
int main()
{
    char originalString[] = "Hello, World!";
    int len = strlen(originalString);

    for (int i=0 ; i<len ; i++)
    {
        char c = originalString[i];
        push(c);
    }
    char reversed[len];
    int ind=0;
    while (!isEmpty())
    {
        reversed[ind]=pop();
        ind++;
    }
    printf("%s\n",reversed);
}
```

Post-Lab – 1(page No: 125)

```
#include <stdio.h>
int sum(int arr[], int size)
{
   int total = 0;
   for (int i = 0; i < size; i++)
   {
      total += arr[i];
   }
   return total;
}
int equalStacks(int h1[], int n1, int h2[], int n2, int h3[], int n3)
{
   int sum1 = sum(h1, n1);
   int sum2 = sum(h2, n2);
   int sum3 = sum(h3, n3);
   int i = 0, j = 0, k = 0;
   while (i < n1 && j < n2 && k < n3)
   {
      if (sum1 == sum2 && sum2 == sum3)
      {
         return sum1;
      }
      if (sum1 >= sum2 && sum1 >= sum3)
      {
         sum1 -= h1[i];
         i++;
      }
      else if (sum2 >= sum1 && sum2 >= sum3)
      {
         sum2 -= h2[j];
         j++;
      }
      else if (sum3 >= sum1 && sum3 >= sum2)
      {
         sum3 -= h3[k];
         k++;
      }
   }
   return 0;
}
int main()
```

```
{
   int n1, n2, n3;
   scanf("%d %d %d", &n1, &n2, &n3);

   int h1[n1], h2[n2], h3[n3];
   for (int i = 0; i < n1; i++)
   {
      scanf("%d", &h1[i]);
   }
   for (int i = 0; i < n2; i++)
   {
      scanf("%d", &h2[i]);
   }
   for (int i = 0; i < n3; i++)
   {
      scanf("%d", &h3[i]);
   }
   int result = equalStacks(h1, n1, h2, n2, h3, n3);
   printf("%d\n", result);

   return 0;
}
```

Post-Lab – 2(page No:127)

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

struct Node* top = NULL;

void push(int x)
{
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   if (newNode == NULL)
   {
      printf("Stack Overflow. Cannot push: %d\n", x);
      return;
   }
```

```
    newNode->data = x;
    newNode->next = top;
    top = newNode;
}

void pop()
{
    if (top == NULL)
    {
        printf("Stack is empty! Deletion not possible.\n");
        return;
    }
    struct Node* temp = top;
    top = top->next;
    free(temp);
}
int getMax()
{
        struct Node *temp;
        temp=top;
        if(temp ==NULL) return -1;
        int max=top->data;
        while(temp!=NULL)
        {
                if(temp->data>min)
                        max=temp->data;
                temp=temp->next;
    }
    return max;
}
int main()
{
    int n;
    scanf("%d", &n);
    for (int i = 0; i < n; i++)
    {
        char operation[10];
        scanf(" %s", operation);
        if (strcmp(operation, "1") == 0)
        {
            int value;
            scanf(" %d", &value);
            push(value);
        }
```

```
        else if (strcmp(operation, "2") == 0)
        {
            pop();
        }
        else if (strcmp(operation, "3") == 0)
        {
            printf("%d\n", getMax());
        }
    }
    return 0;
}
```

Skill Lab-1(page No:129)

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

struct node* top = NULL;

void push(int x)
{
    struct node* newnode=(struct node*)malloc(sizeof(struct node));
    if (newnode == NULL)
    {
        printf("Stack Overflow. Cannot push: %d\n", x);
        return;
    }
    newnode->data = x;
    newnode->next = top;
    top = newnode;
}

void printInOrder(struct node *temp)
{
    if (temp == NULL)
    {
        return;
    }
```

```
    else
    {
        printInOrder(temp->next);
        printf("%d ", temp->data);
    }
}
void display()
{
    if (top == NULL)
    {
        printf("Stack is empty!\n");
        return;
    }
    int i;
    struct node *temp=top;
    while(temp != NULL)
    {
        printf("%d ", temp->data);
        temp=temp->next;
    }
    printf("\n");
}

int main()
{
    push(5);
    push(2);
    push(1);
    push(6);
    push(8);
    printf("Current stack elements:\n");
    display();
    printf("Stack elements in insertion order: ");
    printInOrder(top);
    return 0;
}
```

Skill Lab-2(page No:130)

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

struct Node* top = NULL;

void push(int x)
{
   struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
   if (newNode == NULL)
   {
      printf("Stack Overflow. Cannot push: %d\n", x);
      return;
   }
   newNode->data = x;
   newNode->next = top;
   top = newNode;
}
float calculateAverage()
{
   if (top == NULL)
   {
      printf("Stack is empty!\n");
      return 0;
   }
   int sum = 0;
   int count = 0;
   struct Node* temp = top;
   while (temp != NULL)
   {
      sum += temp->data;
      count++;
      temp = temp->next;
   }
   return (float)sum / count;
}
void displayStack()
{
```

```
    if (top == NULL)
    {
        printf("Stack is empty!\n");
        return;
    }
    struct Node* temp = top;
    printf("Elements of the stack: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main()
{
    push(6);
    push(4);
    push(2);
    push(5);
    push(3);
    push(1);
    displayStack();
    float average = calculateAverage();
    printf("Average of the said stack values: %.2f\n", average);
    return 0;
}
```

Skill Lab-3(page No:132)

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
        int data;
        struct node *next;
};
struct node *top=NULL,*temp1,*p,*q;
void push()
{
        struct node *newnode;
        newnode=(struct node*)malloc(sizeof(struct node));
        if(newnode == NULL)
                return ;
```

```
        int val;
        printf("Enter data value");
        scanf("%d",&val);
        newnode->data=val;
        newnode->next=top;
        top=newnode;
}
void reverse()
{
        struct node *temp=top,*pretemp=NULL,*nexttemp=NULL;
        while(temp != NULL)
        {
                nexttemp=temp->next;
                temp->next=pretemp;
                pretemp=temp;
                temp=nexttemp;
        }
        temp1=pretemp;
}
int isPalindrome()
{
   reverse();
   p=top;
   q=temp1;
   int flag=0;
   while(p != NULL && q != NULL)
   {
      if(p->data != q->data)
      {
         flag=1;
         break;
      }
      p=p->next;
      q=q->next;
   }
   return flag==0;
}
int main()
{
   int n,i;
   printf("the number of elements");
   scanf("%d",&n);
   for(i=1;i<=n;i++)
   {
```

```
        push();
    }
    if(isPalindrome())
        printf("is Palindrome\n");
    else
        printf("is Not Palindrome\n");
}
```

Skill Lab-4(page No:134)

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
    int data;
    struct node *next;
};
struct node *top = NULL, *temp1, *p, *q;

void push()
{
    struct node *newnode;
    newnode = (struct node*)malloc(sizeof(struct node));
    if (newnode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    int val;
    printf("Enter data value: ");
    scanf("%d", &val);
    newnode->data = val;
    newnode->next = top;
    top = newnode;
}
void reverse()
{
    struct node *temp = top, *pretemp = NULL, *nexttemp = NULL;
    while (temp != NULL)
    {
        nexttemp = temp->next;
        temp->next = pretemp;
        pretemp = temp;
        temp = nexttemp;
    }
```

```
    temp1 = pretemp;
}
int isPalindrome()
{
    reverse();  // Reverse the linked list
    p = top;
    q = temp1;
    int flag = 0;
    while (p != NULL && q != NULL)
    {
        if (p->data != q->data)
        {
            flag = 1;
            break;
        }
        p = p->next;
        q = q->next;
    }
    reverse();  // Restore the original list
    return flag == 0;
}
int main()
{
    int n, i;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
    {
        push();
    }
    if (isPalindrome())
        printf("The list is a Palindrome\n");
    else
        printf("The list is Not a Palindrome\n");
    return 0;
}
```

Skill Lab-5(page No:136)

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

struct Node
{
```

```
    int data;
    struct Node* next;
};

struct Node* top = NULL;


void push(int value)
{
    struct Node* newNode=(struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = top;
    top = newNode;
}
int pop()
{
    if (top == NULL)
    {
        printf("Stack is empty.\n");
        return -1;
    }
    int value = top->data;
    struct Node* temp = top;
    top = top->next;
    free(temp);
    return value;
}
void decimalToBinary(int decimal)
{
    if (decimal == 0)
    {
        printf("The binary equivalent is: 0\n");
        return;
    }
    while (decimal > 0)
    {
        push(decimal % 2);
        decimal = decimal / 2;
    }
    printf("The binary equivalent is: ");
    while (top != NULL)
    {
        printf("%d", pop());
    }
```

```
    printf("\n");
}

int main()
{
    int decimal;
    printf("Input a decimal number: ");
    scanf("%d", &decimal);
    decimalToBinary(decimal);
    return 0;
}
```