Data Structure Assignment - 3

Problems

- Write a C program to perform push and pop operation on a stack array
- Write a C program to perform push and pop operation on a linked stack
- Write a C program to reverse a string using stack
- Write a C program to evaluate postfix notation
- Write a C program to infix notation into equivalent postfix notation
- Write a C program to to implement a circular queue
- Write a C program to implement input, output on restricted dequeues

Problem - 1

C Code

```
/**
* Data Structures Assignment - 3
* CS 392
* Problem 1:
*       Write a C program to perform push and pop operation on a stack array
*
* Joydeep Mukherjee
* CSE,3rd Sem, 11000116030
* GCETTS 2017
*
**/

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

int *stack;
int pos;
int size;


void init() {
  // Prompt the user to enter the size of stack
  printf("Enter Size of stack: ");
  scanf("%d",&size);
  stack = malloc(sizeof(int)*size);
  pos = 0;
}
void push(int elem) {
  if(size == pos) {
    printf("Stack Overflow, %d not added\n",elem);
    return;
  }
  stack[pos++] = elem;

}
int pop() {
  if(pos == 0) {
```

```
    printf("Stack is Empty\n");
    return 0;
  }
  return stack[--pos];
}


int main(void ){
  int temp;

  init();

  printf("Enter 0 to pop, -1 to exit\n");
  do{
    printf(">>> ");
    scanf("%d", &temp);

    if(temp)
      push(temp);
    else
      if(temp=pop())
        printf("Popped: %d\n",temp);

  } while(temp != -1);

  return 0;
}
```

Output

```
➜ ./3_1
Enter Size of stack: 5
Enter 0 to pop, -1 to exit
>>> 5
>>> 9
>>> 4
>>> 3
>>> 4
>>> 8
Stack Overflow, 8 not added
>>> 0
Popped: 4
>>> 0
Popped: 3
>>> 0
Popped: 4
>>> 0
Popped: 9
>>> 0
Popped: 5
>>> 0
Stack is Empty
>>> 0
Stack is Empty
>>> 5
>>> 6
>>> 1
>>> 0
Popped: 1
```

```
>>> 0
Popped: 6
>>> 0
Popped: 5
>>> 0
Stack is Empty
>>> -1

```

Problem - 2

C Code

```
/**
* Data Structures Assignment - 3
* CS 392
* Problem 2:
*       Write a C program to perform push and pop operation on a linked stack
*
* Joydeep Mukherjee
* CSE,3rd Sem, 11000116030
* GCETTS 2017
*
**/

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


typedef struct node_t {
  int val;
  struct node_t *next;
}node;

node *stack;

void init() {
  // Create Stack
  stack = NULL;
}
void push(int elem) {
  // Allocate New node
  node *new = malloc(sizeof(node));
  // Check for memory fault
  if(!new) {
    printf("Memory Fault\n");
    return;
  }
  new->val = elem;
  new->next = stack;

  stack = new;
}
int pop() {
  int temp;
  // Check for empty stack
  if(!stack) {
    printf("Stack is Empty\n");
    return 0;
```

```
  }
  temp = stack->val;
  stack = stack->next;

  return temp;
}


int main(void ){
  int temp;

  init();

  printf("Enter 0 to pop, -1 to exit\n");
  do{
    printf(">>> ");
    scanf("%d", &temp);

    if(temp)
      push(temp);
    else
      if(temp=pop())
        printf("Popped: %d\n",temp);

  } while(temp != -1);

  return 0;
}

```

Output

```
➜ ./3_2
Enter 0 to pop, -1 to exit
>>> 5
>>> 6
>>> 6
>>> 7
>>> 4
>>> 2
>>> 5
>>> 5
>>> 5
>>> 5
>>> 78
>>> 98
>>> 12
>>> 0
Popped: 12
>>> 0
Popped: 98
>>> 0
Popped: 78
>>> 0
Popped: 5
>>> 0
Popped: 5
>>> 0
Popped: 5
```

```
>>> 0
Popped: 5
>>> 0
Popped: 2
>>> 0
Popped: 4
>>> 0
Popped: 7
>>> 0
Popped: 6
>>> 0
Popped: 6
>>> 0
Popped: 5
>>> 0
Stack is Empty
>>> 0
Stack is Empty
>>> 4
>>> 6
>>> 6
>>> 7
>>> 0
Popped: 7
>>> 0
Popped: 6
>>> 0
Popped: 6
>>> 0
Popped: 4
>>> 0
Stack is Empty
>>> 0
Stack is Empty
>>> -1

```

Problem - 3

C Code

```

/**
* Data Structures Assignment - 3
* CS 392
* Problem 2:
*       Write a C program to reverse a string using stack
*
* Joydeep Mukherjee
* CSE,3rd Sem, 11000116030
* GCETTS 2017
*
**/

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


typedef struct node_t {
```

```
  int val;
  struct node_t *next;
}node;

node *stack;

void init() {
  // Create Stack
  stack = NULL;
}
void push(int elem) {
  // Allocate New node
  node *new = malloc(sizeof(node));
  // Check for memory fault
  if(!new) {
    printf("Memory Fault\n");
    return;
  }
  new->val = elem;
  new->next = stack;

  stack = new;
}
int pop() {
  int temp;
  // Check for empty stack
  if(!stack) {
    //printf("Stack is Empty\n");
    return 0;
  }
  temp = stack->val;
  stack = stack->next;

  return temp;
}


int main(void ){
  char c;

  init();
  printf("Enter String:\n>>>");
  while((c = getchar()) != '\n') {
    push(c);
  }
  printf("Reversed String: ");
  while(c = pop())
    printf("%c",c);
  printf("\n");


  return 0;
}


```

Output

```
➜ ./3_3
Enter String:
```

```
>>>joydeep mukherjee
Reversed String: eejrehkum peedyoj

```

Problem - 4

C Code

```
/**
* Data Structures Assignment - 3
* CS 392
* Problem 2:
*       Write a C program to evaluate a postfix expression
*
* Joydeep Mukherjee
* CSE,3rd Sem, 11000116030
* GCETTS 2017
*
**/

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


typedef struct node_t {
  int val;
  struct node_t *next;
}node;

node *stack;

void init() {
  // Create Stack
  stack = NULL;
}
void push(int elem) {
  // Allocate New node
  node *new = malloc(sizeof(node));
  // Check for memory fault
  if(!new) {
    printf("Memory Fault\n");
    return;
  }
  new->val = elem;
  new->next = stack;

  stack = new;
}
int pop() {
  int temp;
  // Check for empty stack
  if(!stack) {
    //printf("Stack is Empty\n");
    return 0;
  }
  temp = stack->val;
  stack = stack->next;

  return temp;
```

```
}


int main(void ){
  char c;
  int a,b;
  init();
  printf("Enter Postfix Expression:\n>>>");
  while((c = getchar()) != '\n') {
    if(c >= 48 && c < 58) {
        //printf("Pushed: %d\n",c-48);
        push(c-48);
    }
    else if (c != ' '){
      b = pop();
      a = pop();
      //printf("Popped: %d %d\n",a,b);
      switch (c) {
        case '+':
          push(a+b);
          break;
        case '-':
          push(a-b);
          break;
        case '*':
          push(a*b);
          break;
        case '/':
          push(a/b);
          break;
        default:
          break;
      }
    }
  }
  printf("Result of Expression: %d\n",pop());

  return 0;
}
```

Output

```
➜ ./3_4
Enter Postfix Expression:
>>>1 2 + 6 9 - /
Result of Expression: -1
```

Problem - 5

C Code

```
/**
* Data Structures Assignment - 3
* CS 392
* Problem 5:
*       Write a C program to convert an infix expression to a postfix expression
*
* Joydeep Mukherjee
```

```
 * CSE,3rd Sem, 11000116030
 * GCETTS 2017
 *
**/

#include <stdio.h>

char stack[20];
int top = -1;
void push(char x)
{
    stack[++top] = x;
}

char pop()
{
    if(top == -1)
        return -1;
    else
        return stack[top--];
}

int priority(char x)
{
    if(x == '(')
        return 0;
    if(x == '+' || x == '-')
        return 1;
    if(x == '*' || x == '/')
        return 2;
}

int main(void)
{
    char exp[20];
    char *e, x;
    printf("Enter the expression :: ");
    scanf("%s",exp);
    e = exp;
    while(*e != '\0')
    {
        if(isalnum(*e))
            printf("%c",*e);
        else if(*e == '(')
            push(*e);
        else if(*e == ')')
        {
            while((x = pop()) != '(')
                printf("%c", x);
        }
        else
        {
            while(priority(stack[top]) >= priority(*e))
                printf("%c",pop());
            push(*e);
        }
        e++;
    }
    while(top != -1)
    {
        printf("%c",pop());
```

```
    }
}

```

Problem - 6

C Code

```
/**
* Data Structures Assignment - 3
* CS 392
* Problem 6:
*           Write a C program to implement a circular queue
* Joydeep Mukherjee
* CSE,3rd Sem, 11000116030
* GCETTS 2017
*
**/

#include <stdio.h>

#define SIZE 5

int items[SIZE];
int front = -1, rear =-1;

int isFull()
{
    if( (front == rear + 1) || (front == 0 && rear == SIZE-1)) return 1;
    return 0;
}

int isEmpty()
{
    if(front == -1) return 1;
    return 0;
}

void enQueue(int element)
{
    if(isFull()) printf("\n Queue is full!! \n");
    else
    {
        if(front == -1) front = 0;
        rear = (rear + 1) % SIZE;
        items[rear] = element;
        printf("\n Inserted -> %d", element);
    }
}


int deQueue()
{
    int element;
    if(isEmpty()) {
        printf("\n Queue is empty !! \n");
```

```
        return(-1);
    } else {
        element = items[front];
        if (front == rear){
            front = -1;
            rear = -1;
        } /* Q has only one element, so we reset the queue after dequeing it. ? */
        else {
            front = (front + 1) % SIZE;

        }
        printf("\n Deleted element -> %d \n", element);
        return(element);
    }
}




void display()
{
    int i;
    if(isEmpty()) printf(" \n Empty Queue\n");
    else
    {
        printf("\n Front -> %d ",front);
        printf("\n Items -> ");
        for( i = front; i!=rear; i=(i+1)%SIZE) {
            printf("%d ",items[i]);
        }
        printf("%d ",items[i]);
        printf("\n Rear -> %d \n",rear);
    }
}

int main()
{
    // Fails because front = -1
    deQueue();

    enQueue(1);
    enQueue(2);
    enQueue(3);
    enQueue(4);
    enQueue(5);

    // Fails to enqueue because front == 0 && rear == SIZE - 1
    enQueue(6);

    display();
    deQueue();

    display();

    enQueue(7);
    display();

    // Fails to enqueue because front == rear + 1
    enQueue(8);

    return 0;
```

```
}
```

Problem - 7

C Code

```
/**
* Data Structures Assignment - 3
* CS 392
* Problem 7:
*           Write a C program to implement an restricted deque
* Joydeep Mukherjee
* CSE,3rd Sem, 11000116030
* GCETTS 2017
*
**/
#include <stdio.h>

#define MAX 30

typedef struct dequeue
{
    int data[MAX];
    int rear,front;
}dequeue;

void initialize(dequeue *p);
int empty(dequeue *p);
int full(dequeue *p);
void enqueueR(dequeue *p,int x);
void enqueueF(dequeue *p,int x);
int dequeueF(dequeue *p);
int dequeueR(dequeue *p);
void print(dequeue *p);

void main()
{
    int i,x,op,n;
    dequeue q;

    initialize(&q);

    do
    {

printf("\n1.Create\n2.Insert(rear)\n3.Insert(front)\n4.Delete(rear)\n5.Delete(front)");
        printf("\n6.Print\n7.Exit\n\nEnter your choice:");
        scanf("%d",&op);

        switch(op)
        {
            case 1: printf("\nEnter number of elements:");
                    scanf("%d",&n);
                    initialize(&q);
                    printf("\nEnter the data:");

                    for(i=0;i<n;i++)
                    {
                        scanf("%d",&x);
```

```
                        if(full(&q))
                        {
                            printf("\nQueue is full!!");
                            exit(0);
                        }
                        enqueueR(&q,x);
                    }
                    break;

            case 2: printf("\nEnter element to be inserted:");
                    scanf("%d",&x);

                    if(full(&q))
                    {
                        printf("\nQueue is full!!");
                        exit(0);
                    }

                    enqueueR(&q,x);
                    break;

            case 3: printf("\nEnter the element to be inserted:");
                    scanf("%d",&x);

                    if(full(&q))
                    {
                        printf("\nQueue is full!!");
                        exit(0);
                    }

                    enqueueF(&q,x);
                    break;

            case 4: if(empty(&q))
                    {
                        printf("\nQueue is empty!!");
                        exit(0);
                    }

                    x=dequeueR(&q);
                    printf("\nElement deleted is %d\n",x);
                    break;

            case 5: if(empty(&q))
                    {
                        printf("\nQueue is empty!!");
                        exit(0);
                    }

                    x=dequeueF(&q);
                    printf("\nElement deleted is %d\n",x);
                    break;

            case 6: print(&q);
                    break;

            default: break;
        }
    }while(op!=7);
}

```

```
114 void initialize(dequeue *P)
115 {
116     P->rear=-1;
117     P->front=-1;
118 }
119 
120 int empty(dequeue *P)
121 {
122     if(P->rear==-1)
123         return(1);
124 
125     return(0);
126 }
127 
128 int full(dequeue *P)
129 {
130     if((P->rear+1)%MAX==P->front)
131         return(1);
132 
133     return(0);
134 }
135 
136 void enqueueR(dequeue *P,int x)
137 {
138     if(empty(P))b
139     {
140         P->rear=0;
141         P->front=0;
142         P->data[0]=x;
143     }
144     else
145     {
146         P->rear=(P->rear+1)%MAX;
147         P->data[P->rear]=x;
148     }
149 }
150 
151 void enqueueF(dequeue *P,int x)
152 {
153     if(empty(P))
154     {
155         P->rear=0;
156         P->front=0;
157         P->data[0]=x;
158     }
159     else
160     {
161         P->front=(P->front-1+MAX)%MAX;
162         P->data[P->front]=x;
163     }
164 }
165 
166 int dequeueF(dequeue *P)
167 {
168     int x;
169 
170     x=P->data[P->front];
171 
172     if(P->rear==P->front)    //delete the last element
173         initialize(P);
174     else
```

```
        P->front=(P->front+1)%MAX;

    return(x);
}

int dequeueR(dequeue *P)
{
    int x;

    x=P->data[P->rear];

    if(P->rear==P->front)
        initialize(P);
    else
        P->rear=(P->rear-1+MAX)%MAX;

    return(x);
}

void print(dequeue *P)
{
    if(empty(P))
    {
        printf("\nQueue is empty!!");
        exit(0);
    }

    int i;
    i=P->front;

    while(i!=P->rear)
    {
        printf("\n%d",P->data[i]);
        i=(i+1)%MAX;
    }

    printf("\n%d\n",P->data[P->rear]);
}
```

Output

```
1.Create
2.Insert(rear)
3.Insert(front)
4.Delete(rear)
5.Delete(front)
6.Print
7.Exit
Enter your choice:1

Enter number of elements:3

Enter the data:4 6 7

1.Create
2.Insert(rear)
3.Insert(front)
4.Delete(rear)
5.Delete(front)
6.Print
```

```
7.Exit

Enter your choice:6

4
6
7

1.Create
2.Insert(rear)
3.Insert(front)
4.Delete(rear)
5.Delete(front)
6.Print
7.Exit

Enter your choice:4

Element deleted is 7

1.Create
2.Insert(rear)
3.Insert(front)
4.Delete(rear)
5.Delete(front)
6.Print
7.Exit

Enter your choice:7
```