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

struct Stack {
   int size;
   int top;
   char *S;
};
void create(struct Stack *, int);
void push(struct Stack *, char);
char pop(struct Stack *);
int isEmpty(struct Stack *);
int isBalance(struct Stack *, char *);
int main() {
   struct Stack st;
   char exp[100];

   printf("Enter the expression to check balance: ");
   scanf("%s", exp);

   create(&st, strlen(exp));

   if (isBalance(&st, exp)) {
      printf("Expression is balanced\n");
   } else {
      printf("Expression is not balanced\n");
   }

   free(st.S);
   return 0;
```

```
}

void create(struct Stack *st, int size) {
   st->size = size;
   st->top = -1;
   st->S = (char *)malloc(st->size * sizeof(char));
}

void push(struct Stack *st, char x) {
   if (st->top == st->size - 1) {
      printf("Stack overflow\n");
   } else {
      st->top++;
      st->S[st->top] = x;
   }
}

char pop(struct Stack *st) {
   if (st->top == -1) {
      printf("Stack underflow\n");
      return '\0';
   } else {
      char x = st->S[st->top];
      st->top--;
      return x;
   }
}

int isEmpty(struct Stack *st) {
   return st->top == -1;
}
```

```
int isBalance(struct Stack *st, char *exp) {
   int i;
   for (i = 0; exp[i] != '\0'; i++) {
      if (exp[i] == '(') {
         push(st, '(');
      } else if (exp[i] == ')') {
         if (isEmpty(st)) {
            return 0;
         }
         pop(st);
      }
   }
   return isEmpty(st) ? 1 : 0;
}
```

Output:

```
Enter the expression to check balance: ((a+b)*(c+d))
Expression is balanced
```

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
struct Stack {
   int size;
   int top;
```

```
    char *S;
};
void create(struct Stack *, int);
void push(struct Stack *, char);
char pop(struct Stack *);
int isEmpty(struct Stack *);
int precedence(char);
void infixtoPostfix(struct Stack*,char*,char*);
void reverseString(char*);
int main() {
    struct Stack st;
    char str[20],infix[100],postfix[100],prefix[100];
    printf("Enter the infix expression:");
    scanf("%s",infix);
    create(&st, strlen(infix));
    infixtoPostfix(&st,infix,postfix);
    printf("Postfix expression:%s\n",postfix);
    printf("Enter a string:");
    scanf("%s",str);
    reverseString(str);
    printf("Reversed string:%s\n",str);
    free(st.S);
    return 0;
}

void create(struct Stack *st, int size) {
    st->size = size;
    st->top = -1;
    st->S = (char *)malloc(st->size * sizeof(char));
}
```

```
void push(struct Stack *st, char x) {
    if (st->top == st->size - 1) {
        printf("Stack overflow\n");
    } else {
        st->top++;
        st->S[st->top] = x;
    }
}

char pop(struct Stack *st) {
    if (st->top == -1) {
        printf("Stack underflow\n");
        return '\0';
    } else {
        char x = st->S[st->top];
        st->top--;
        return x;
    }
}

int isEmpty(struct Stack *st) {
    return st->top == -1;
}

int precedence(char op)
{
    if(op=='+' || op=='-')
    {
        return 1;
    }
    if(op=='*' || op=='/')
```

```
    {
        return 2;
    }
    return 0;
}
void infixtoPostfix(struct Stack *st,char* infix,char*postfix)
{
    int i,j=0;
    for(i=0;infix[i]!='\0';i++)
    {
        char ch=infix[i];
        if(isalnum(ch))
        {
            postfix[j++]=ch;
        }
        else if(ch=='(')
        {
            push(st,ch);
        }
        else if(ch==')')
        {
            while(!isEmpty(st)&&st->S[st->top]!='(')
            {
                postfix[j++]=pop(st);
            }
            pop(st);
        }
        else
        {
            while(!isEmpty(st)&&precedence(st->S[st->top])>=precedence(ch))
            {
```

```
            postfix[j++]=pop(st);
        }
        push(st,ch);
    }
  }
    while(!isEmpty(st))
    {
        postfix[j++]=pop(st);
    }
    postfix[j]='\0';
}
void reverseString(char *str)
{
  int n=strlen(str);
  for(int i=0;i<n/2;i++)
  {
    char temp=str[i];
    str[i]=str[n-i-1];
    str[n-i-1]=temp;
  }
}
```

Output:

```
Enter the infix expression:a+b*c-d/e
Postfix expression:abc*+de/-
Enter a string:program
Reversed string:margorp
```

```
#include<stdio.h>
#include<stdlib.h>
struct Queue{
   int size;
   int front;
   int rear;
   int *Q;
};
void enqueue(struct Queue *,int);
int dequeue(struct Queue *);
int main()
{
   struct Queue q;
   printf("Enter the size:");
   scanf("%d",&q.size);
   q.Q=(int*)malloc(q.size*sizeof(int));
   q.front=q.rear=-1;
   enqueue(&q,10);
   enqueue(&q,20);
   enqueue(&q,30);
   enqueue(&q,40);
   enqueue(&q,50);
   printf("Dequeued:%d\n",dequeue(&q));
   return 0;
}
void enqueue(struct Queue *q,int x)
{
   if(q->rear==q->size-1)
   {
      printf("Queue is full");
   }
```

```
    else
    {
        q->rear++;
        q->Q[q->rear]=x;
        printf("Enqueued:%d\n",x);
    }
}
int dequeue(struct Queue *p)
{
    int x=-1;
    if(p->front==p->rear)
    {
        printf("Queue is empty");
    }
    else
    {
        p->front++;
        x=p->Q[p->front];
    }
    return x;
}
```

Output:

```
Enter the size:5
Enqueued:10
Enqueued:20
Enqueued:30
Enqueued:40
Enqueued:50
Dequeued:10
```

```
/*1.Simulate a Call Center Queue

Create a program to simulate a call center where
incoming calls are handled on a first-come, first-served basis.
Use a queue to manage call handling and provide options to
add, remove, and view calls.*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct Call{
   int id;
   char callerName[50];
};
struct Queue{
   int size;
   int front;
   int rear;
   struct Call *Q;
};
void enqueue(struct Queue *,int,char*);
struct Call dequeue(struct Queue *);
void display(struct Queue *);
int main()
{
   struct Queue q;
   printf("Enter the size of queue:");
   scanf("%d",&q.size);
   q.Q=(struct Call*)malloc(q.size*sizeof(struct Call));
   q.front=q.rear=-1;
   int choice,call=0;
```

```
char callerName[50];
while(1)
{
   printf("1.Add call(Enqueue)\n");
   printf("2.Remove call(Dequeue)\n");
   printf("3.View calls\n");
   printf("4.Exit\n");
   printf("Enter your choice:");
   scanf("%d",&choice);
   switch (choice)
   {
      case 1:
         if(q.rear==q.size-1)
         {
            printf("Queue is full\n");
         }
         else
         {
            printf("Enter caller name:");
            scanf("%s",callerName);
            enqueue(&q,call++,callerName);
         }
         break;
      case 2:
         if(q.front==q.rear)
         {
            printf("Queue is empty.No calls to handle\n");
         }
         else
         {
            struct Call handledCall=dequeue(&q);
```

```
                printf("Handled call id:%d, Caller name:%s\n",handledCall.id,handledCall.callerName);
            }
            break;
        case 3:
            display(&q);
            break;
        case 4:
            free(q.Q);
            printf("Exiting the queue \n");
            return 0;
        default:
            printf("Invalid choice");
      }
   }
   return 0;
}

void enqueue(struct Queue *q,int id,char *callerName)
{
   q->rear++;
   q->Q[q->rear].id=id;
   strcpy(q->Q[q->rear].callerName,callerName);
   printf("Added call ID %d, Caller Name:%s\n",id,callerName);
}
struct Call dequeue(struct Queue *q)
{
   q->front++;
   return q->Q[q->front];
}
void display(struct Queue *q)
{
```

```
    if(q->front==q->rear)
    {
        printf("Queue is empty.\n");
    }
    else
    {
        printf("Calls in queue:");
        for(int i=q->front+1;i<=q->rear;i++)
        {
            printf("Caller id:%d, Caller name:%s\n",q->Q[i].id,q->Q[i].callerName);
        }
    }
}
```

Output:

```
Enter the size of queue:6
1.Add call(Enqueue)
2.Remove call(Dequeue)
3.View calls
4.Exit
Enter your choice:1
Enter caller name:anu
Added call ID 0, Caller Name:anu
1.Add call(Enqueue)
2.Remove call(Dequeue)
3.View calls
4.Exit
Enter your choice:1
Enter caller name:arun
Added call ID 1, Caller Name:arun
1.Add call(Enqueue)
2.Remove call(Dequeue)
```

3.View calls

4.Exit

Enter your choice:1

Enter caller name:abhirami

Added call ID 2, Caller Name:abhirami

1.Add call(Enqueue)

2.Remove call(Dequeue)

3.View calls

4.Exit

Enter your choice:2

Handled call id:0, Caller name:anu

1.Add call(Enqueue)

2.Remove call(Dequeue)

3.View calls

4.Exit

Enter your choice:3

Calls in queue:Caller id:1, Caller name:arun

Caller id:2, Caller name:abhirami

1.Add call(Enqueue)

2.Remove call(Dequeue)

3.View calls

4.Exit

Enter your choice:4

Exiting the queue

```
/*2.Print Job Scheduler

Implement a print job scheduler where print requests
are queued. Allow users to add new print jobs, cancel
a specific job, and print jobs in the order they were added.*/

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

struct PrintJob {
   int id;
   char jobName[50];
};

struct Queue {
   int size;
   int front;
   int rear;
   struct PrintJob *Q;
};

void addJob(struct Queue *, int, char *);
void cancelJob(struct Queue *, int);
void viewJobs(struct Queue *);
int isEmpty(struct Queue *);

int main() {
   struct Queue q;
   printf("Enter the number of print jobs: ");
   scanf("%d", &q.size);
```

```
q.Q = (struct PrintJob *)malloc(q.size * sizeof(struct PrintJob));
q.front = q.rear = -1;

int choice, jobId = 0;
char jobName[50];

while (1)
{
   printf("1. Add Print Job\n");
   printf("2. Cancel Print Job\n");
   printf("3. View Pending Jobs\n");
   printf("4. Exit\n");
   printf("Enter your choice: ");
   scanf("%d", &choice);

   switch (choice) {
      case 1:
         if (q.rear == q.size - 1) {
            printf("Queue is full. Cannot add more jobs.\n");
         } else {
            printf("Enter job name: ");
            scanf(" %49s", jobName); // Limit input to avoid buffer overflow
            addJob(&q, ++jobId, jobName);
         }
         break;
      case 2:
         if (isEmpty(&q)) {
            printf("Queue is empty. No jobs to cancel.\n");
         } else {
            int cancelId;
            printf("Enter the job ID to cancel: ");
```

```
            scanf("%d", &cancelId);
            cancelJob(&q, cancelId);
        }
        break;
      case 3:
        viewJobs(&q);
        break;
      case 4:
        free(q.Q);
        printf("Exiting the Print Job Scheduler.\n");
        return 0;
      default:
        printf("Invalid choice. Please try again.\n");
    }
  }

  return 0;
}

int isEmpty(struct Queue *q) {
  return q->front == q->rear;
}

void addJob(struct Queue *q, int id, char *jobName) {
  q->rear++;
  q->Q[q->rear].id = id;
  strncpy(q->Q[q->rear].jobName, jobName, sizeof(q->Q[q->rear].jobName) - 1);
  q->Q[q->rear].jobName[sizeof(q->Q[q->rear].jobName) - 1] = '\0'; // Ensure null termination
  printf("Added Print Job ID: %d, Job Name: %s\n", id, jobName);
}
```

```
void cancelJob(struct Queue *q, int id) {
    int found = 0;
    for (int i = q->front + 1; i <= q->rear; i++) {
        if (q->Q[i].id == id) {
            found = 1;
            printf("Cancelled Print Job ID: %d, Job Name: %s\n", q->Q[i].id, q->Q[i].jobName);
            for (int j = i; j < q->rear; j++) {
                q->Q[j] = q->Q[j + 1];
            }
            q->rear--;
            break;
        }
    }
    if (!found) {
        printf("Print Job with ID %d not found.\n", id);
    }
}

void viewJobs(struct Queue *q) {
    if (isEmpty(q)) {
        printf("No pending print jobs.\n");
    } else {
        printf("Pending Print Jobs:\n");
        for (int i = q->front + 1; i <= q->rear; i++) {
            printf("Job ID: %d, Job Name: %s\n", q->Q[i].id, q->Q[i].jobName);
        }
    }
}
```

Output:

```
Enter the number of print jobs: 5
1. Add Print Job
```

2. Cancel Print Job

3. View Pending Jobs

4. Exit

Enter your choice: 1

Enter job name: PrintJob1

Added Print Job ID: 1, Job Name: PrintJob1

1. Add Print Job

2. Cancel Print Job

3. View Pending Jobs

4. Exit

Enter your choice: 1

Enter job name: PrintJob2

Added Print Job ID: 2, Job Name: PrintJob2

1. Add Print Job

2. Cancel Print Job

3. View Pending Jobs

4. Exit

Enter your choice: 3

Pending Print Jobs:

Job ID: 1, Job Name: PrintJob1

Job ID: 2, Job Name: PrintJob2

1. Add Print Job

2. Cancel Print Job

3. View Pending Jobs

4. Exit

Enter your choice: 2

Enter the job ID to cancel: 1

Cancelled Print Job ID: 1, Job Name: PrintJob1

1. Add Print Job

2. Cancel Print Job

3. View Pending Jobs

4. Exit

Enter your choice: 3

Pending Print Jobs:

Job ID: 2, Job Name: PrintJob2

1. Add Print Job

2. Cancel Print Job

3. View Pending Jobs

4. Exit

Enter your choice: 4

Exiting the Print Job Scheduler.

/*3.Design a Ticketing System

Simulate a ticketing system where people join a queue
to buy tickets. Implement functionality for people to
join the queue, buy tickets, and display the queue's
current state.*/

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

struct Customer {
    int id;
    char name[50];
```

```
};

struct Queue {
   int size;
   int front;
   int rear;
   struct Customer *Q;
};

void joinQueue(struct Queue *, int, char *);
struct Customer buyTicket(struct Queue *);
void displayQueue(struct Queue *);
int isQueueEmpty(struct Queue *);

int main() {
   struct Queue q;
   printf("Enter the maximum number of customers in the queue: ");
   scanf("%d", &q.size);
   q.Q = (struct Customer *)malloc(q.size * sizeof(struct Customer));
   q.front = q.rear = -1;

   int choice, customerId = 0;
   char customerName[50];

   while (1)
   {
      printf("1. Join the Queue\n");
      printf("2. Buy Ticket\n");
      printf("3. Display Queue\n");
      printf("4. Exit\n");
      printf("Enter your choice: ");
```

```
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                if (q.rear == q.size - 1) {
                    printf("Queue is full. No more customers can join.\n");
                } else {
                    printf("Enter customer name: ");
                    getchar();
                    scanf("%49s", customerName);
                    joinQueue(&q, ++customerId, customerName);
                }
                break;
            case 2:
                if (isQueueEmpty(&q)) {
                    printf("Queue is empty. No customers to buy tickets.\n");
                } else {
                    struct Customer servedCustomer = buyTicket(&q);
                    printf("Ticket issued to Customer ID: %d, Name: %s\n", servedCustomer.id,
servedCustomer.name);
                }
                break;
            case 3:
                displayQueue(&q);
                break;
            case 4:
                free(q.Q);
                printf("Exiting the Ticketing System.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
```

```
        }
    }

    return 0;
}

int isQueueEmpty(struct Queue *q) {
    return q->front == q->rear;
}

void joinQueue(struct Queue *q, int id, char *name) {
    q->rear++;
    q->Q[q->rear].id = id;
    strncpy(q->Q[q->rear].name, name, sizeof(q->Q[q->rear].name) - 1);
    q->Q[q->rear].name[sizeof(q->Q[q->rear].name) - 1] = '\0';
    printf("Customer ID %d, Name: %s joined the queue.\n", id, name);
}

struct Customer buyTicket(struct Queue *q) {
    q->front++;
    return q->Q[q->front];
}

void displayQueue(struct Queue *q) {
    if (isQueueEmpty(q)) {
        printf("No customers in the queue.\n");
    } else {
        printf("Customers in the queue:\n");
        for (int i = q->front + 1; i <= q->rear; i++) {
            printf("Customer ID: %d, Name: %s\n", q->Q[i].id, q->Q[i].name);
        }
```

```
    }
}
```

Output:

```
Enter the maximum number of customers in the queue: 3
1. Join the Queue
2. Buy Ticket
3. Display Queue
4. Exit
Enter your choice: 1
Enter customer name: Anu
Customer ID 1, Name: Anu joined the queue.
1. Join the Queue
2. Buy Ticket
3. Display Queue
4. Exit
Enter your choice: 1
Enter customer name: Ben
Customer ID 2, Name: Ben joined the queue.
1. Join the Queue
2. Buy Ticket
3. Display Queue
4. Exit
Enter your choice: 3
Customers in the queue:
Customer ID: 1, Name: Anu
Customer ID: 2, Name: Ben
1. Join the Queue
2. Buy Ticket
3. Display Queue
4. Exit
Enter your choice: 2
```

Ticket issued to Customer ID: 1, Name: Anu

1. Join the Queue

2. Buy Ticket

3. Display Queue

4. Exit

Enter your choice: 3

Customers in the queue:

Customer ID: 2, Name: Ben

1. Join the Queue

2. Buy Ticket

3. Display Queue

4. Exit

Enter your choice: 4

Exiting the Ticketing System.

```
//Implementation of queue using linked list
#include <stdio.h>
#include <stdlib.h>

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

struct Queue {
```

```
    struct Node *front;
    struct Node *rear;
};

void enqueue(struct Queue *, int);
int dequeue(struct Queue *);
void display(struct Queue *);

int main() {
    struct Queue q;
    q.front = q.rear = NULL;

    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
    enqueue(&q, 40);

    display(&q);

    printf("Dequeued: %d\n", dequeue(&q));
    printf("Dequeued: %d\n", dequeue(&q));

    display(&q);

    return 0;
}

void enqueue(struct Queue *q, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;
```

```
    if (q->rear == NULL) {
        q->front = q->rear = newNode;
        return;
    }

    q->rear->next = newNode;
    q->rear = newNode;
}

int dequeue(struct Queue *q) {
    if (q->front == NULL) {
        printf("Queue is empty.\n");
        return -1;
    }

    struct Node *temp = q->front;
    int value = temp->data;
    q->front = q->front->next;

    if (q->front == NULL) {
        q->rear = NULL;
    }

    free(temp);
    return value;
}

void display(struct Queue *q) {
    if (q->front == NULL) {
        printf("Queue is empty.\n");
```

```
        return;
    }

    struct Node *temp = q->front;
    printf("Queue: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

Output:

Queue: 10 20 30 40

Dequeued: 10

Dequeued: 20

Queue: 30 40