ASSIGNMENT

Implementation of Balanced Expression

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

struct Stack {
    int size;
    int top;
    int *S;
};
void create(struct Stack *);
void push(struct Stack *, int );
int pop(struct Stack *);
int isBalanced(char *);
int main() {
    char exp[100];
    printf("Enter an expression: ");
    scanf("%s", exp);
    if (isBalanced(exp)) {
        printf("The expression is balanced.\n");
    } else {
        printf("The expression is not balanced.\n");
    }
    return 0;
}
void create(struct Stack *st) {
    printf("Enter Size : ");
    scanf("%d", &st->size);
    st->top = -1;
    st->S = (int *)malloc(st->size * sizeof(int));
}
void push(struct Stack *st, int x) {
    if (st->top == st->size - 1) {
        printf("Stack Overflow\n");
    } else {
        st->top++;
        st->S[st->top] = x;
    }
}
int pop(struct Stack *st) {
    int x = -1;
    if (st->top == -1) {
        printf("Stack Underflow\n");
    } else {
        x = st->S[st->top];
        st->top--;
    }
    return x;
}
int isEmpty(struct Stack *st) {
    return st->top == -1;
}
int isBalanced(char *exp) {
    struct Stack st;
    create(&st);
    for (int i = 0; exp[i] != '\0'; i++) {
        if (exp[i] == '(') {
```

```
            push(&st, exp[i]);
        } else if (exp[i] == ')') {
            if (isEmpty(&st)) {
                return 0;
            }
            pop(&st);
        }
    }
    return isEmpty(&st);
}
```

Implement Infix to Postfix

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

#define MAX 100
typedef struct {
    char items[MAX];
    int top;
} Stack;
void push(Stack *s, char value) {
    if (s->top == MAX - 1) {
        printf("Stack is full!\n");
        return;
    }
    s->items[++(s->top)] = value;
}
char pop(Stack *s) {
    if (s->top == -1) {
        printf("Stack is empty!\n");
        return '\0';
    }
    return s->items[(s->top)--];
}
char peek(Stack *s) {
    if (s->top == -1) {
        return '\0';
    }
    return s->items[s->top];
}
int precedence(char op) {
    switch (op) {
        case '+':
        case '-':
            return 1;
        case '*':
        case '/':
            return 2;
        case '^':
            return 3;
        default:
            return 0;
    }
}
int isOperator(char c) {
    return c == '+' || c == '-' || c == '*' || c == '/' || c == '^';
}
```

```c
void infixToPostfix(char* infix, char* postfix) {
    Stack s;
    s.top = -1;
    int k = 0;
    for (int i = 0; i < strlen(infix); i++) {
        if (isdigit(infix[i]) || isalpha(infix[i])) {
            postfix[k++] = infix[i];
        } else if (infix[i] == '(') {
            push(&s, infix[i]);
        } else if (infix[i] == ')') {
            while (peek(&s) != '(') {
                postfix[k++] = pop(&s);
            }
            pop(&s);
        } else if (isOperator(infix[i])) {
            while (s.top != -1 && precedence(peek(&s)) >= precedence(infix[i])) {
                postfix[k++] = pop(&s);
            }
            push(&s, infix[i]);
        }
    }

    while (s.top != -1) {
        postfix[k++] = pop(&s);
    }
    postfix[k] = '\0';
}
int main() {
    char infix[MAX], postfix[MAX];
    printf("Enter infix expression: ");
    scanf("%s", infix);
    infixToPostfix(infix, postfix);
    printf("Postfix expression: %s\n", postfix);
    return 0;
}
```

Implementation of Queue

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

#define MAX_SIZE 100
struct Queue {
    int size;
    int front;
    int rear;
    int *arr;
};
void createQueue(struct Queue *q, int size) {
    q->size = size;
    q->front = q->rear = -1;
    q->arr = (int *)malloc(q->size * sizeof(int));
}
void enqueue(struct Queue *q, int data) {
    if (q->rear == q->size - 1) {
        printf("Queue is Full\n");
        return;
    }
    if (q->front == -1) {
        q->front = q->rear = 0;
```

```
    } else {
        q->rear = (q->rear + 1) % q->size;
    }
    q->arr[q->rear] = data;
}
int dequeue(struct Queue *q) {
    if (q->front == -1) {
        printf("Queue is Empty\n");
        return -1;
    }
    int data = q->arr[q->front];
    if (q->front == q->rear) {
        q->front = q->rear = -1;
    } else {
        q->front = (q->front + 1) % q->size;
    }
    return data;
}
int main() {
    struct Queue q;
    int size;
    printf("Enter the size of the queue: ");
    scanf("%d", &size);
    createQueue(&q, size);
    enqueue(&q, 10);
    enqueue(&q, 20);
    enqueue(&q, 30);
    printf("%d\n", dequeue(&q));
    printf("%d\n", dequeue(&q));
    enqueue(&q, 40);
    enqueue(&q, 50);
    while (q.front != -1) {
        printf("%d ", dequeue(&q));
    }
    return 0;
}
```

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

#define MAX_SIZE 100
struct Queue {
    int front;
    int rear;
    int Q[MAX_SIZE];
};
// Function declarations
void enQueue(struct Queue *q, int num);
int deQueue(struct Queue *q);
void displayQueue(struct Queue *q);
int isEmpty(struct Queue *q);
int isFull(struct Queue *q);
int main() {
    struct Queue q;
    q.front = -1;
    q.rear = -1;
   int choice, num;
    while (1) {
        printf("\nQueue Operations:\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 the element to enqueue: ");
                scanf("%d", &num);
                enQueue(&q, num);
                break;
            case 2:
                num = deQueue(&q);
                if (num != -1) {
                    printf("Dequeued element: %d\n", num);
                }
                break;
            case 3:
                displayQueue(&q);
                break;
            case 4:
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }
   return 0;
}
// Function definitions
void enQueue(struct Queue *q, int num) {
    if (isFull(q)) {
        printf("Queue is full. Dequeue some elements first.\n");
        return;
    }
    if (q->front == -1) {
        q->front = 0;
    }
    q->rear++;
    q->Q[q->rear] = num;
}
int deQueue(struct Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty. No elements to dequeue.\n");
        return -1;
    }
    int dequeued = q->Q[q->front];
    q->front++;
    if (q->front > q->rear) {
        q->front = -1;
        q->rear = -1;
    }
    return dequeued;
}
void displayQueue(struct Queue *q) {
    if (isEmpty(q)) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d ", q->Q[i]);
```

```
    }
    printf("\n");
}
int isEmpty(struct Queue *q) {
    return q->front == -1 || q->front > q->rear;
}
int isFull(struct Queue *q) {
    return q->rear == MAX_SIZE - 1;
}
```

Reverse a string using stack

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

#define MAX 100
struct Stack {
    int size;
    int top;
    char *S;
};
void create(struct Stack *);
void push(struct Stack *, char);
char pop(struct Stack *);
void display(struct Stack *);
void reverseString(char *);
int main() {
    char str[MAX];
    printf("Enter a string: ");
    scanf("%[^\n]",str);
    str[strcspn(str, "\n")] = '\0';
    printf("Original string: %s\n", str);
    reverseString(str);
    printf("Reversed string: %s\n", str);
    return 0;
}
void create(struct Stack *st) {
    printf("Enter Size: ");
    scanf("%d", &st->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->S[++(st->top)] = x;
    }
}
char pop(struct Stack *st) {
    if (st->top == -1) {
        printf("Stack Underflow\n");
        return '\0';
    } else {
        return st->S[(st->top)--];
    }
}
void display(struct Stack *st) {
    for (int i = st->top; i >= 0; i--) {
```

```
        printf("%c\n", st->S[i]);
    }
    printf("\n");
}
void reverseString(char *str) {
    int n = strlen(str);
    struct Stack st;
    st.size = n;
    st.top = -1;
    st.S = (char *)malloc(st.size * sizeof(char));
    for (int i = 0; i < n; i++) {
        push(&st, str[i]);
    }
    for (int i = 0; i < n; i++) {
        str[i] = pop(&st);
    }
    free(st.S);
}
```

```
/ 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>
#define MAX 100
typedef struct {
    int id;
    char callerName[50];
} Call;
typedef struct {
    Call queue[MAX];
    int front;
    int rear;
} CallQueue;
void initializeQueue(CallQueue* cq);
int isFull(CallQueue* cq);
int isEmpty(CallQueue* cq);
void addCall(CallQueue* cq, int id, char* callerName);
void handleCall(CallQueue* cq);
void viewCalls(CallQueue* cq);
int main() {
    CallQueue cq;
    int choice, id;
    char callerName[50];
    initializeQueue(&cq);
    while (1) {
        printf("\n--- Call Center Menu ---\n");
        printf("1. Add Incoming Call\n");
        printf("2. Handle Call\n");
        printf("3. View Calls\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                if (!isFull(&cq)) {
                    printf("Enter Call ID: ");
```

```
                    scanf("%d", &id);
                    printf("Enter Caller Name: ");
                    scanf("%s", callerName);
                    addCall(&cq, id, callerName);
                } else {
                    printf("Queue is full! Cannot add more calls.\n");
                }
                break;
            case 2:
                if (!isEmpty(&cq)) {
                    handleCall(&cq);
                } else {
                    printf("No calls to handle!\n");
                }
                break;
            case 3:
                viewCalls(&cq);
                break;
            case 4:
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
void initializeQueue(CallQueue* cq) {
    cq->front = -1;
    cq->rear = -1;
}
int isFull(CallQueue* cq) {
    return cq->rear == MAX - 1;
}
int isEmpty(CallQueue* cq) {
    return cq->front == -1 || cq->front > cq->rear;
}
void addCall(CallQueue* cq, int id, char* callerName) {
    if (isEmpty(cq)) {
        cq->front = 0;
    }
    cq->rear++;
    cq->queue[cq->rear].id = id;
    strcpy(cq->queue[cq->rear].callerName, callerName);
    printf("Call added: ID=%d, Caller=%s\n", id, callerName);
}
void handleCall(CallQueue* cq) {
    printf("Handling call: ID=%d, Caller=%s\n", cq->queue[cq->front].id, cq->queue[cq-
>front].callerName);
    cq->front++;
    if (cq->front > cq->rear) {
        cq->front = cq->rear = -1;
    }
}
void viewCalls(CallQueue* cq) {
    if (isEmpty(cq)) {
        printf("No calls in the queue.\n");
    } else {
        printf("\n--- Current Calls in Queue ---\n");
        for (int i = cq->front; i <= cq->rear; i++) {
            printf("ID=%d, Caller=%s\n", cq->queue[i].id, cq->queue[i].callerName);
```

```
        }
    }
}
```

```
// 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>
#define MAX 100
// Define a structure for a print job
typedef struct {
    int jobId;
    char documentName[50];
} PrintJob;
// Queue structure
typedef struct {
    PrintJob queue[MAX];
    int front;
    int rear;
} JobQueue;
// Function prototypes
void initializeQueue(JobQueue* q);
int isFull(JobQueue* q);
int isEmpty(JobQueue* q);
void addJob(JobQueue* q, int jobId, char* documentName);
void cancelJob(JobQueue* q, int jobId);
void printJobs(JobQueue* q);
int main() {
    JobQueue q;
    int choice, jobId;
    char documentName[50];
    initializeQueue(&q);
    while (1) {
        printf("\n--- Print Job Scheduler Menu ---\n");
        printf("1. Add Print Job\n");
        printf("2. Cancel Print Job\n");
        printf("3. Print All Jobs\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                if (!isFull(&q)) {
                    printf("Enter Job ID: ");
                    scanf("%d", &jobId);
                    printf("Enter Document Name: ");
                    scanf("%s", documentName);
                    addJob(&q, jobId, documentName);
                } else {
                    printf("Queue is full! Cannot add more print jobs.\n");
                }
                break;
            case 2:
                if (!isEmpty(&q)) {
                    printf("Enter Job ID to cancel: ");
                    scanf("%d", &jobId);
                    cancelJob(&q, jobId);
```

```
                } else {
                    printf("No jobs to cancel!\n");
                }
                break;
            case 3:
                printJobs(&q);
                break;
            case 4:
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
// Initialize the queue
void initializeQueue(JobQueue* q) {
    q->front = -1;
    q->rear = -1;
}
// Check if the queue is full
int isFull(JobQueue* q) {
    return q->rear == MAX - 1;
}
// Check if the queue is empty
int isEmpty(JobQueue* q) {
    return q->front == -1 || q->front > q->rear;
}
// Add a new print job to the queue
void addJob(JobQueue* q, int jobId, char* documentName) {
    if (isEmpty(q)) {
        q->front = 0;
    }
    q->rear++;
    q->queue[q->rear].jobId = jobId;
    strcpy(q->queue[q->rear].documentName, documentName);
    printf("Print job added: Job ID=%d, Document=%s\n", jobId, documentName);
}
// Cancel a specific print job from the queue
void cancelJob(JobQueue* q, int jobId) {
    int found = 0;
    for (int i = q->front; i <= q->rear; i++) {
        if (q->queue[i].jobId == jobId) {
            found = 1;
            printf("Cancelling print job: Job ID=%d, Document=%s\n", q->queue[i].jobId, q-
>queue[i].documentName);
            // Shift jobs to fill the gap
            for (int j = i; j < q->rear; j++) {
                q->queue[j] = q->queue[j + 1];
            }
            q->rear--;
            if (q->rear < q->front) {
                q->front = q->rear = -1; // Reset queue if empty
            }
            break;
        }
    }
    if (!found) {
        printf("Job ID %d not found in the queue.\n", jobId);
    }
```

```
}
// Print all jobs in the queue
void printJobs(JobQueue* q) {
    if (isEmpty(q)) {
        printf("No print jobs in the queue.\n");
    } else {
        printf("\n--- Current Print Jobs in Queue ---\n");
        for (int i = q->front; i <= q->rear; i++) {
            printf("Job ID=%d, Document=%s\n", q->queue[i].jobId, q->queue[i].documentName);
        }
    }
}
```

```
// 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>

#define MAX 100
typedef struct {
    int personId;
    char name[50];
} Person;
typedef struct {
    Person queue[MAX];
    int front;
    int rear;
} TicketQueue;
void initializeQueue(TicketQueue* tq);
int isFull(TicketQueue* tq);
int isEmpty(TicketQueue* tq);
void joinQueue(TicketQueue* tq, int personId, char* name);
void buyTicket(TicketQueue* tq);
void displayQueue(TicketQueue* tq);
int main() {
    TicketQueue tq;
    int choice, personId;
    char name[50];
    initializeQueue(&tq);
    while (1) {
        printf("\n--- Ticketing System Menu ---\n");
        printf("1. Join 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 (!isFull(&tq)) {
                    printf("Enter Person ID: ");
                    scanf("%d", &personId);
                    printf("Enter Name: ");
                    scanf("%s", name);
                    joinQueue(&tq, personId, name);
                } else {
                    printf("Queue is full! Cannot add more people.\n");
                }
```

```
                break;
            case 2:
                if (!isEmpty(&tq)) {
                    buyTicket(&tq);
                } else {
                    printf("No one is in the queue to buy tickets!\n");
                }
                break;
            case 3:
                displayQueue(&tq);
                break;
            case 4:
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
    return 0;
}
void initializeQueue(TicketQueue* tq) {
    tq->front = -1;
    tq->rear = -1;
}
int isFull(TicketQueue* tq) {
    return tq->rear == MAX - 1;
}
int isEmpty(TicketQueue* tq) {
    return tq->front == -1 || tq->front > tq->rear;
}
void joinQueue(TicketQueue* tq, int personId, char* name) {
    if (isEmpty(tq)) {
        tq->front = 0;
    }
    tq->rear++;
    tq->queue[tq->rear].personId = personId;
    strcpy(tq->queue[tq->rear].name, name);
    printf("Person added to queue: ID=%d, Name=%s\n", personId, name);
}
void buyTicket(TicketQueue* tq) {
    printf("Buying ticket for: ID=%d, Name=%s\n", tq->queue[tq->front].personId, tq->queue[tq-
>front].name);
    tq->front++;
    if (tq->front > tq->rear) {
        tq->front = tq->rear = -1;
    }
}
void displayQueue(TicketQueue* tq) {
    if (isEmpty(tq)) {
        printf("The queue is empty.\n");
    } else {
        printf("\n--- Current Queue ---\n");
        for (int i = tq->front; i <= tq->rear; i++) {
            printf("ID=%d, Name=%s\n", tq->queue[i].personId, tq->queue[i].name);
        }
    }
}
```