PHASE 1 DAY 23

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

struct Stack {
    int size;
    int top;
    int *S;
};

// Function Prototypes
void create(struct Stack *);
void push(struct Stack *, int);
void display(struct Stack *);
int pop(struct Stack *);
int peek(struct Stack *, int);
int isBalance(char *);
int isempty(struct Stack *);
void Infix_Postfix_conversion(char *, char *, struct Stack *);
int precedence(char);

int main()
{
    char Infix_exp[50], Postfix_exp[50];
    int result;
    struct Stack st;

    printf("Enter the Infix expression: ");
    scanf("%s", Infix_exp);

    create(&st);

    Infix_Postfix_conversion(Infix_exp, Postfix_exp, &st);
    printf("PostFix expression: %s\n", Postfix_exp);
    return 0;
}

void create(struct Stack *st)
{
    st->size=10;
    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;
    }
}

void display(struct Stack *st)
{
    for (int i = st->top; i >= 0; i--)
    {
        printf("%d\n", st->S[i]);
    }
    printf("\n");
}

int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    return st->S[st->top--];
}
```

```
int peek(struct Stack *st, int pos)
{
    if (pos <= 0 || pos > st->top + 1)
    {
        printf("Invalid position\n");
        exit(0);
    }
    return st->S[st->top - pos + 1];
}

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

int isBalance(char *expr)
{
    struct Stack st;
    create(&st);

    for (int i = 0; expr[i] != '\0'; i++)
    {
        if (expr[i] == '(')
        {
            push(&st, expr[i]);
        }
        else if (expr[i] == ')')
        {
            if (isempty(&st))
            {
                return 0;
            }
            pop(&st);
        }
    }
    return isempty(&st);
}

void Infix_Postfix_conversion(char *Infix_exp, char *Postfix_exp, struct Stack *stk)
```

```
{
    int i = 0, j = 0;

    while (Infix_exp[i] != '\0')
    {
        if (isdigit(Infix_exp[i]) || isalpha(Infix_exp[i]))
        {
            Postfix_exp[j++] = Infix_exp[i];
        }
        else
        {
            if (Infix_exp[i] == '(')
            {
                push(stk, Infix_exp[i]);

            }
            else if (Infix_exp[i] == ')')
            {
                while (stk->top != -1 && stk->S[stk->top] != '(')
                {
                    Postfix_exp[j++] = pop(stk);
                }

                pop(stk);
            }
            else
            {
                while (stk->top != -1 && precedence(Infix_exp[i]) <=
precedence(stk->S[stk->top]))
                {
                    Postfix_exp[j++] = pop(stk);
                }
                push(stk, Infix_exp[i]);
            }
        }
        i++;
    }

    while (!isempty(stk))
    {
        Postfix_exp[j++] = pop(stk);
    }

    Postfix_exp[j] = '\0';
}


int precedence(char opr)
{
    if (opr == '*' || opr == '/')
    {
        return 2;
    } else if (opr == '+' || opr == '-')
    {
        return 1;
    }
    return 0;
}
```

REVERSE STRING USING STACK

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

struct Stack {
    int size;
    int top;
    int *S;
};

// Function Prototypes
void create(struct Stack *,int);
void push(struct Stack *, int);
void display(struct Stack *);
int pop(struct Stack *);
int peek(struct Stack *, int);
int main()
{
    struct Stack st;
    char str1[10];
    printf("Enter the string :");
    scanf("%[^\n]",str1);
    getchar();
    int len1=strlen(str1);
    char str2[len1];
    int i=0;

    create(&st,len1);

    while(str1[i] !='\0')
    {
        push(&st,str1[i]);
        i++;
    }
    int j=0;
    while(st.top != -1)
    {
     str2[j]=pop(&st);
     j++;
    }
    str2[j]='\0';

    printf("Reversed string:%s",str2);
free(st->S);
    return 0;
}

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

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;
    }
}

void display(struct Stack *st)
{
    for (int i = st->top; i >= 0; i--)
    {
        printf("%d\n", st->S[i]);
    }
    printf("\n");
}

int pop(struct Stack *st)
{
    if (st->top == -1)
    {
        printf("Stack Underflow\n");
        return -1;
    }
    return st->S[st->top--];
}
```

QUEUE USING ARRAY

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

struct Queue {
    int size;
    int front, rear;
    int *Q;
```

```
};

// Function Prototypes
void enqueue(struct Queue *, int);
void display(struct Queue *);
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;

    // Test the queue
    enqueue(&q, 5);
    enqueue(&q, 10);
    enqueue(&q, 15);

    printf("Queue contents after enqueues:\n");
    display(&q);

    printf("Dequeued element: %d\n", dequeue(&q));
    printf("Queue contents after dequeue:\n");
    display(&q);

    free(q.Q);
    return 0;
}

void enqueue(struct Queue *q, int x)
{
    if (q->rear == q->size - 1)
    {
        printf("Queue full\n");
    }
    else
    {
        if (q->front == -1)
            q->front = 0;
        q->rear++;
        q->Q[q->rear] = x;
    }
}

int dequeue(struct Queue *q)
{
    int x = -1;
    if (q->front == -1 || q->front > q->rear)
    {
        printf("Queue empty\n");
    }
    else
    {
        x = q->Q[q->front];
        q->front++;
        if (q->front > q->rear)
        {
            q->front = q->rear = -1;
        }
    }
    return x;
}

void display(struct Queue *q)
{
    if (q->front == -1)
    {
        printf("Queue is empty\n");
    }
    else
    {
        for (int i = q->front; i <= q->rear; i++)
        {
            printf("%d ", q->Q[i]);
        }
        printf("\n");
    }
```

```
}
```

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 <string.h>



struct Queue {

    int size;

    int front, rear;

    char names[10][50];

    int phone_numbers[10];

};


void enqueue(struct Queue *, char *, int);

void dequeue(struct Queue *);

void display(struct Queue *);


int main() {

    struct Queue q;

    int choice;

    q.size = 10;
```

```
q.front = q.rear = -1;


while (1) {
    printf("\nMenu:\n");
    printf("1. Add call to queue\n");
    printf("2. Remove call from queue\n");
    printf("3. Display ongoing calls\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    getchar();

    switch (choice) {
        case 1: {
            if (q.rear == q.size - 1)
            {
                printf("Queue is full. Cannot add more calls.\n");
            } else {
                char name[50];
                int phone_number;

                printf("Enter the name: ");
                scanf("%[^\n]",name);
                getchar();
                printf("Enter phone number: ");
                scanf("%d", &phone_number);

                enqueue(&q, name, phone_number);
            }
            break;
        }
        case 2:
            dequeue(&q);
            break;
        case 3:
            display(&q);
            break;
        case 4:
            printf("Exiting...\n");
            return 0;

        default:
            printf("Invalid choice. Please try again.\n");
      }
   }
}

void enqueue(struct Queue *q, char *name, int phone_number)
{
   if (q->rear == q->size - 1) {
      printf("Queue is full. Cannot add more calls.\n");
      return;
   }
   if (q->front == -1)
   {
      q->front = 0;
   }
   q->rear++;
   strcpy(q->names[q->rear], name);
   q->phone_numbers[q->rear] = phone_number;
   printf("Call added to the queue.\n");
}

void dequeue(struct Queue *q)
{
   if (q->front == -1 || q->front > q->rear)
   {
      printf("Queue is empty. No calls to remove.\n");
      return;
   }
   printf("Removed call - Name: %s, Phone Number: %d\n", q->names[q->front],
q->phone_numbers[q->front]);
   q->front++;
   if (q->front > q->rear)
   {
      q->front = q->rear = -1;
   }
}

void display(struct Queue *q)
{
   if (q->front == -1 || q->front > q->rear)
   {
      printf("Queue is empty. No calls to display.\n");
      return;
   }
   printf("Calls in the queue:\n");
   for (int i = q->front; i <= q->rear; i++)
   {
      printf("%d. Name: %s, Phone Number: %d\n", i - q->front + 1, q->names[i],
q->phone_numbers[i]);
   }
}
```

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 Queue {
    int size;
    int front, rear;
    char jobs[10][50];
};

void addJob(struct Queue *, char *);
void cancelJob(struct Queue *, char *);
void printJobs(struct Queue *);

int main() {
    struct Queue q;
    q.size =10;
    q.front = q.rear = -1;

    int choice;
    char jobName[50]];

    while (1)
  {
        printf("\nMenu:\n");
        printf("1. Add Print Job\n");
        printf("2. Cancel Print Job\n");
        printf("3. Display Print Queue\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar(); // Clear input buffer

        switch (choice)
          {
            case 1:
                if (q.rear == q.size - 1)
                  {
                    printf("Queue is full. Cannot add more jobs.\n");
                }
              else
              {
                    printf("Enter print job name: ");
                    scanf(“%[^\n]”,jobname);
                    addJob(&q, jobName);
                }
                break;

            case 2:
                if (q.front == -1 || q.front > q.rear)
               {
                    printf("Queue is empty. No jobs to cancel.\n");
                }
               else
                 {
                    printf("Enter print job name to cancel: ");
                    scanf(“%[^\n]”,jobName);
                    cancelJob(&q, jobName);
                }
                break;

            case 3:
                printJobs(&q);
                break;

            case 4:
                printf("Exiting...\n");
                return 0;

            default:
                printf("Invalid choice. Please try again.\n");
        }
    }
}

void addJob(struct Queue *q, char *jobName)
 {
    if (q->rear == q->size - 1)
{
        printf("Queue is full. Cannot add job.\n");
        return;
    }
    if (q->front == -1)
 {
        q->front = 0;
    }
    q->rear++;
    strcpy(q->jobs[q->rear], jobName);
    printf("Job added to the queue.\n");
}

void cancelJob(struct Queue *q, char *jobName)
{
    int found = 0;
    for (int i = q->front; i <= q->rear; i++)
{
        if (strcmp(q->jobs[i], jobName) == 0)
 {
            found = 1;
            for (int j = i; j < q->rear; j++)
{
                strcpy(q->jobs[j], q->jobs[j + 1]);
            }
            q->rear--;
            if (q->rear < q->front)
 {
                q->front = q->rear = -1;
            }
            printf("Job '%s' canceled.\n", jobName);
            break;
        }
    }
    if (!found)
 {
        printf("Job '%s' not found in the queue.\n", jobName);
    }
}

void printJobs(struct Queue *q)
 {
    if (q->front == -1 || q->front > q->rear)
{
        printf("Queue is empty. No jobs to display.\n");
        return;
    }
    printf("Current Print Queue:\n");
    for (int i = q->front; i <= q->rear; i++) {
        printf("%d. %s\n", i - q->front + 1, q->jobs[i]);
    }
}
```

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 Node
 {
    char name[50];
    int ticketNumber;
    struct Node *next;
};

struct Queue
{
    struct Node *front, *rear;
};

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

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

    int choice;
    char name[50];
    int ticketNumber = 1;

    while (1)
      {
        printf("\nMenu:\n");
        printf("1. Join Ticket Queue\n");
        printf("2. Buy Ticket\n");
        printf("3. Display Queue\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        getchar();

        switch (choice)
          {
           case 1:
               printf("Enter name: ");
               fgets(name, 50, stdin);
               name[strcspn(name, "\n")] = '\0';
               joinQueue(&q, name, ticketNumber++);
               break;

           case 2:
               buyTicket(&q);
               break;

           case 3:
               displayQueue(&q);
               break;

           case 4:
               printf("Exiting...\n");
               return 0;

           default:
               printf("Invalid choice. Please try again.\n");
        }
    }
}

void joinQueue(struct Queue *q, char *name, int ticketNumber)
 {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    strcpy(newNode->name, name);
    newNode->ticketNumber = ticketNumber;
    newNode->next = NULL;
    if (q->rear == NULL) {
        q->front = q->rear = newNode;
    } else {
        q->rear->next = newNode;
        q->rear = newNode;
    }
    printf("Added %s to the queue with ticket number %d.\n", name, ticketNumber);
}

void buyTicket(struct Queue *q) {
    if (q->front == NULL) {
        printf("Queue is empty. No tickets to buy.\n");
        return;
    }

    struct Node *temp = q->front;
    printf("Ticket bought by %s (Ticket Number: %d).\n", temp->name, temp->ticketNumber);
    q->front = q->front->next;

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

void displayQueue(struct Queue *q)
 {
    if (q->front == NULL)
 {
        printf("Queue is empty. No one is waiting.\n");
        return;
    }

    struct Node *temp = q->front;
    printf("Current Ticket Queue:\n");
    while (temp != NULL)
 {
        printf("Name: %s, Ticket Number: %d\n", temp->name, temp->ticketNumber);
        temp = temp->next;
    }
}
```