DAY 23

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

struct stack{
    int top;
    int size;
    char *s;
};
void create(struct stack*st);
void push(struct stack*st,char x);
void display(struct stack*st);
int pop(struct stack *st);
int isEmpty(struct stack *st);
int balance(char*exp);

int main(){
    struct stack st;
    char arr[100];
    printf("enter the expression");
    scanf("%s",arr);

    if(balance(arr)){
        printf("this expression is balance");
    }
    else{
        printf("the expression is unbalance");
    }

}
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");
    }
    else{
        st->top++;
        st->s[st->top]=x;
    }
```

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

}
int pop(struct stack *st){
    int x=-1;
    if(st->top==-1){
        printf("empthy");
    }
    else{
        x=st->s[st->top];
        st->top--;
    }
    return x;
}

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

int balance(char*exp){
    struct stack st;
    create(&st);
    for(int i=0;i<strlen(exp);i++){
        char c = exp[i];
        if(c=='('){
            push(&st,exp[i]);
        }
        else if (c==')')
        {
            if(isEmpty(&st)){
                return 0;
            }
            else{
                pop(&st);
            }

        }
    }
    return isEmpty(&st);
}
```

2)
```
#include<stdio.h>
#include<string.h>
#include<stdlib.h>

struct stack {
   int top;
   int size;
   char *s;
};

void create(struct stack* st, char *arr);
void push(struct stack* st, char x);
void display(struct stack* st);
int pop(struct stack *st);
int isEmpty(struct stack *st);
int balance(char* exp);
int precedence(char c);
void infix_postfix(struct stack* st, char* infix, char* postfix);

int main() {
   struct stack st;
   char arr[100];
   char infix[100], postfix[100];
   int choice;

   do {

      printf("1. Check Parenthesis Matching\n");
      printf("2. Infix to Postfix\n");
      printf("3. Exit\n");
      printf("\nEnter choice:\n");
      scanf("%d", &choice);

      switch(choice) {
         case 1:
            printf("Enter the expression: ");
            scanf("%s", arr);
            if (balance(arr)) {
               printf("This expression is balanced\n");
            } else {
               printf("The expression is unbalanced\n");
            }
            break;
         case 2:
            printf("Enter infix expression: ");
            scanf("%s", infix);
            infix_postfix(&st, infix, postfix);
```

```
            break;
        }

    } while(choice != 3);

    return 0;
}

void create(struct stack* st, char *arr) {
    st->size = strlen(arr);
    st->top = -1;
    st->s = (char*)malloc(st->size * sizeof(char));  // Allocate memory for stack
}

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;
    }
}

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

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

```
        return 2;
    } else if(c == '^') {
        return 3;
    } else {
        return 0;
    }
}



int balance(char* exp) {
    struct stack st;
    create(&st, exp);  // Pass the correct string to create the stack
    for(int i = 0; i < strlen(exp); i++) {
        char c = exp[i];
        if(c == '(') {
            push(&st, c);
        } else if(c == ')') {
            if(isEmpty(&st)) {
                return 0;
            } else {
                pop(&st);
            }
        }
    }
    return isEmpty(&st);
}

void infix_postfix(struct stack* st, char* infix, char* postfix) {
    //create(&st, infix);

    int k = 0;

    for(int i = 0; infix[i] != '\0'; i++) {
        char c = infix[i];


        if((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
            postfix[k++] = c;
        }



        else if(c == '+' || c == '-' || c == '*' || c == '/' || c == '^') {
            while(!isEmpty(st) && precedence(c) <= precedence(st->s[st->top])) {
                postfix[k++] = pop(st);
            }
            push(st, c);
```

```
        }
    }


    while(!isEmpty(st)) {
        postfix[k++] = pop(st);
    }

    postfix[k] = '\0';
    printf("Postfix expression: %s\n", postfix);
}
```

1)

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

struct stack {
    int top;
    int size;
    char *s;
};

void create(struct stack* st, int size);
void push(struct stack* st, char x);
void display(struct stack* st);
char pop(struct stack *st);
int isEmpty(struct stack *st);
int balance(char* exp);
int precedence(char c);
void infix_postfix(char* infix, char* postfix);
void str_reverse(char *str);

int main() {
    char arr[100];
    char infix[100], postfix[100], str[100];
    int choice;

    do {
        printf("\n1. Check Parenthesis Matching\n");
        printf("2. Infix to Postfix\n");
        printf("3. String Reversal\n");
        printf("4. Exit\n");
        printf("\nEnter choice:\n");
        scanf("%d", &choice);
```

```
        switch (choice) {
            case 1:
                printf("Enter the expression: ");
                scanf("%s", arr);
                if (balance(arr)) {
                    printf("This expression is balanced\n");
                } else {
                    printf("The expression is unbalanced\n");
                }
                break;

            case 2:
                printf("Enter infix expression: ");
                scanf("%s", infix);
                infix_postfix(infix, postfix);
                break;

            case 3:
                printf("Enter a string to reverse: ");
                scanf("%s", str);
                str_reverse(str);

                break;
        }
    } while (choice != 4);

    return 0;
}

void create(struct stack* st, int size) {
    st->size = size;
    st->top = -1;
    st->s = (char*)malloc(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 -1;
    } else {
```

```
        return st->s[st->top--];
    }
}

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

int precedence(char c) {
    if (c == '+' || c == '-') {
        return 1;
    } else if (c == '*' || c == '/') {
        return 2;
    } else if (c == '^') {
        return 3;
    } else {
        return 0;
    }
}

int balance(char* exp) {
    struct stack st;
    create(&st, strlen(exp));
    for (int i = 0; i < strlen(exp); i++) {
        char c = exp[i];
        if (c == '(') {
            push(&st, c);
        } else if (c == ')') {
            if (isEmpty(&st)) {

                return 0;
            } else {
                pop(&st);
            }
        }
    }
    int isBalanced = isEmpty(&st);

    return isBalanced;
}

void infix_postfix(char* infix, char* postfix) {
    struct stack st;
    create(&st, strlen(infix));
    int k = 0;

    for (int i = 0; infix[i] != '\0'; i++) {
        char c = infix[i];
```

```
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z')) {
            postfix[k++] = c;
        } else if (c == '(') {
            push(&st, c);
        } else if (c == ')') {
            while (!isEmpty(&st) && st.s[st.top] != '(') {
                postfix[k++] = pop(&st);
            }
            pop(&st);
        } else {
            while (!isEmpty(&st) && precedence(c) <= precedence(st.s[st.top])) {
                postfix[k++] = pop(&st);
            }
            push(&st, c);
        }
    }

    while (!isEmpty(&st)) {
        postfix[k++] = pop(&st);
    }

    postfix[k] = '\0';
    printf("Postfix expression: %s\n", postfix);

}

void str_reverse(char *str) {
    struct stack st;
    create(&st, strlen(str));

    for (int i = 0; str[i] != '\0'; i++) {
        push(&st, str[i]);
    }

    int i = 0;
    while (!isEmpty(&st)) {
        str[i++] = pop(&st);
    }
    str[i] = '\0';
   printf("Reversed string: %s\n", str);
}
```

2)

QUEUE

1)
```
#include<stdio.h>
#include<stdlib.h>

typedef struct{
    int size;
    int front;
    int rear;
    int *q;


}Queue;


void create(Queue*,int);
void enque(Queue *,int);
int deque(Queue*);
void display(Queue *);

int main(){
    Queue st;

    int size,choice,value;
    printf("enter size of the queue:");
    scanf("%d",&size);

    create(&st,size);

    do{
        printf("1.Enqueue\n2.Dequeue\n3.Display\n\n");
        printf("enter choice:");
        scanf("%d",&choice);
        switch(choice){
            case 1:
                printf("Enter the value to enqueue: ");
                scanf("%d", &value);
                enque(&st,value);
                break;
            case 2:
                int delete=deque(&st);
                printf("deleted Value:%d",delete);
                break;
            case 3:
                display(&st);
                break;
        }
```

```
    }while(choice!=3);




    return 0;
}



void create(Queue *st,int size){
    st->size=size;
    st->q=(int*)malloc(size*sizeof(int));
    st->front=-1;
    st->rear=-1;
    printf("Queue Created Successfully!\n");
}
void enque(Queue *st,int value){

    if(st->rear==st->size-1){
        printf("Queue is full\n");
    }else{
         if (st->front == -1) {
            st->front = 0; // Update front when the first element is enqueued
        }
        st->rear++;
        st->q[st->rear]=value;
    }
}
int deque(Queue *st){
    int x=-1;
    if(st->front==st->rear){
        printf("Queue is empty\n");
    }else{

      x=st->q[st->front];
      st->front++;
    }
    return x;
}
void display(Queue *st){
    for(int i=st->front;i<=st->rear;i++){
        printf("%d\t",st->q[i]);
    }
```

}
2)/*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>

typedef struct{
    int size;
    int front;
    int rear;
    int *q;


}Queue;


void create(Queue*,int);
void enque(Queue *,int);
int deque(Queue*);
void display(Queue *);
void add_call(Queue*st,int);
int remove_call(Queue*st);
void view_call(Queue *st);

int main(){
    Queue st;

    int size,choice,value;
    printf("enter size of the queue:");
    scanf("%d",&size);

    create(&st,size);

    do{
        printf("1.Add call\n2.Remove Call\n3.View Call\n\n");
        printf("enter choice:");
        scanf("%d",&choice);
        switch(choice){
            case 1:
```

```
                add_call(&st,value);
                break;
            case 2:
                int delete=remove_call(&st);
                printf("%d phone number deleted\n",delete);
                break;
            case 3:
                view_call(&st);
                break;
        }

    }while(choice!=3);




    return 0;
}



void create(Queue *st,int size){
    st->size=size;
    st->q=(int*)malloc(size*sizeof(int));
    st->front=-1;
    st->rear=-1;
    printf("Queue Created Successfully!\n");
}
void enque(Queue *st,int value){

    if(st->rear==st->size-1){
        printf("Queue is full\n");
    }else{
         if (st->front == -1) {
            st->front = 0; // Update front when the first element is enqueued
        }
        st->rear++;
        st->q[st->rear]=value;
    }
}
int deque(Queue *st){
    int x=-1;
    if(st->front==st->rear){
        printf("Queue is empty\n");
    }else{

      x=st->q[st->front];
```

```
      st->front++;
    }
    return x;
}
void view_call(Queue *st){
    for(int i=st->front;i<=st->rear;i++){
        printf("%d\t",st->q[i]);
    }



}



void add_call(Queue *st,int num){
    printf("enter phone number to add:");
    scanf("%d",&num);
    enque(st,num);
}


int remove_call(Queue *st){
    return deque(st);

}
3)#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct printJob {
    int id;
    char description[100];
};

struct queue {
    int size;
    int front;
    int rear;
    struct printJob *jobs;
};

void add(struct queue job, int n, char des);
void display(struct queue *jobi);
void cancel(struct queue *jobi, int n);

int main() {
```

```
    struct queue job;
    printf("Enter the number of print jobs the queue can hold: ");
    scanf("%d", &job.size);
    job.jobs = (struct printJob*)malloc(job.size * sizeof(struct printJob));
    job.front = -1;
    job.rear = -1;

    while (1) {
        printf("\nEnter 1 to Add Job, 2 to Display Jobs, 3 to Cancel Job, 4 to Exit: ");
        int op;
        scanf("%d", &op);
        int id;
        char dis[100];
        int ca;

        switch (op) {
            case 1:
                printf("Enter Job ID: ");
                scanf("%d", &id);
                printf("Enter Job Description: ");
                scanf(" %[^\n]", dis);
                add(&job, id, dis);
                break;

            case 2:
                display(&job);
                break;

            case 3:
                printf("Enter Job ID to cancel: ");
                scanf("%d", &ca);
                cancel(&job, ca);
                break;

            case 4:
                printf("Exiting program...\n");
                free(job.jobs);
                return 0;

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

void add(struct queue jobi, int n, char des) {
    if (jobi->rear == jobi->size - 1) {
        printf("Queue is full. Cannot add more jobs.\n");
```

```
        return;
    }
    if (jobi->front == -1) {
        jobi->front = 0;
    }
    jobi->rear++;
    jobi->jobs[jobi->rear].id = n;
    strcpy(jobi->jobs[jobi->rear].description, des);
    printf("Job added successfully.\n");
}

void display(struct queue *jobi) {
    if (jobi->front == -1 || jobi->front > jobi->rear) {
        printf("There are no jobs available.\n");
        return;
    }
    printf("Current Print Jobs:\n");
    for (int i = jobi->front; i <= jobi->rear; i++) {
        printf("Id: %d\n Description: %s\n", jobi->jobs[i].id, jobi->jobs[i].description);
    }
}

void cancel(struct queue *jobi, int n) {
    if (jobi->front == -1 || jobi->front > jobi->rear) {
        printf("No jobs to cancel.\n");
        return;
    }
    int found = 0;
    int i;
    for (i = jobi->front; i <= jobi->rear; i++) {
        if (jobi->jobs[i].id == n) {
            found = 1;
            break;
        }
    }

    if (found) {
        for (int j = i; j < jobi->rear; j++) {
            jobi->jobs[j] = jobi->jobs[j + 1];
        }
        jobi->rear--;
        printf("Job with ID %d has been cancelled.\n", n);
        if (jobi->front > jobi->rear) {
            jobi->front = jobi->rear = -1;
        }
    } else {
        printf("No job found with ID %d.\n", n);
    }
```

```
}
```