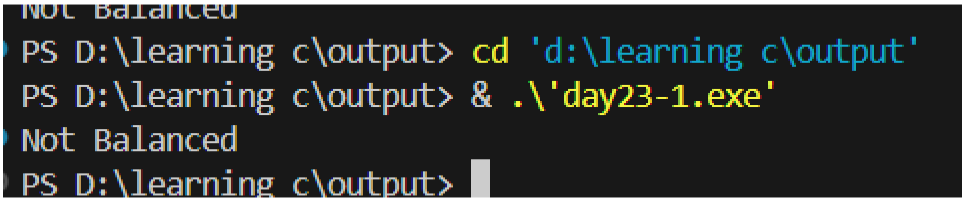
DAY 23

1. //parenthesis matching(exp is balance or not)

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

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

void create(struct stack *, char *);
void push(struct stack *, char );
char pop(struct stack *);
int isEmpty(struct stack *);
int isBalance(char *);

int main() {
    struct stack st;
    char expression[] = "((a+b)*(a-b)))";
    create(&st, expression);

    if (isBalance(expression)) {
        printf("Balanced\n");
    } else {
        printf("Not Balanced\n");
    }

    return 0;
}
```

```
void create(struct stack *st, char *exp) {
    st->size = strlen(exp);//the length of the expression
    st->top = -1;//in the begining stack is empty
    st->s = (char *)malloc(st->size * sizeof(char));
}

void push(struct stack *st, char exp) {
    if (st->top == st->size - 1) {
        printf("stack overflow \n");
    } else {
        st->top++;
        st->s[st->top] = exp;//pushing the parenthesis from the char expression
    }
}

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

int isEmpty(struct stack *st)
{

    if(st->top==-1)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}

int isBalance(char *exp) {
    struct stack st;
    create(&st, exp);//calling the structure creation fn

    for (int i = 0; exp[i] != '\0'; i++) {
        if (exp[i] == '(') {
            push(&st, exp[i]);
        } else if (exp[i] == ')') {
            if (isEmpty(&st)) {//checking whether stack is empty
                return 0;
            }
            pop(&st);//else poping the parenthesis in top
        }
    }
    return isEmpty(&st) ? 1 : 0;//if it is empty at the end return 1 means balanced else not balanced
}
```



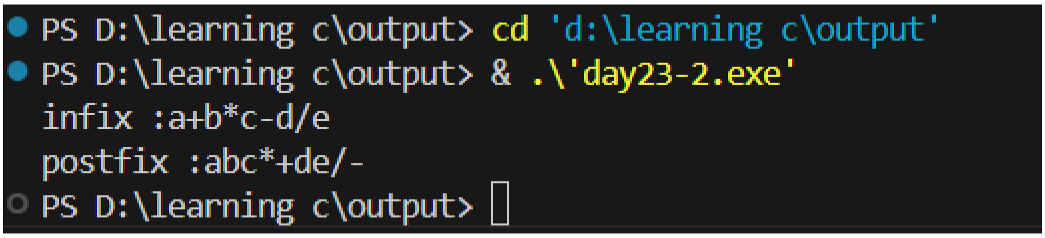
```
Not Balanced
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day23-1.exe'
Not Balanced
PS D:\learning c\output>
```

2. #include<stdio.h>

```
#include<stdlib.h>
#include<string.h>
struct stack
{
   int size;
   int top;
   char *s;
};
void create(struct stack *, char *);
void push(struct stack *, char );
char pop(struct stack *);
int isoperator(char );
int precedence(char );
void InfixToPostfix(struct stack*,char *,int ,char *);
int isoperator(char);
int main()
{
   char expression[]="a+b*c-d/e";
   int max=strlen(expression);
   char postfix[max+1];
   struct stack st;
   create(&st,expression);
   InfixToPostfix(&st,expression,max,postfix);
   printf("infix :%s \n",expression);
   printf("postfix :%s \n",postfix);
   return 0;




}
void create(struct stack *st, char *exp) {
   st->size = strlen(exp);//the length of the expression
   st->top = -1;//in the begining stack is empty
   st->s = (char *)malloc(st->size * sizeof(char));
}
void push(struct stack *st, char exp) {
   if (st->top == st->size - 1) {
       printf("stack overflow \n");
   } else {
       st->top++;
       st->s[st->top] = exp;
   }
}

char pop(struct stack *st) {
   char x;
   if (st->top == -1) {
       printf("stack underflow \n");
   } else {
       x = st->s[st->top];
       st->top--;
   }
   return x;
}
int isoperator(char exp)
{
       if(exp=='+'||exp=='-'||exp=='*'||exp=='/'||exp=='^')
       {
           return 1;
       }
       else
       {
           return 0;
       }



}
int precedence(char exp)
{
   if(exp=='+'||exp=='-')
   {
       return 1;
   }
   else if(exp=='*'||exp=='/')
   {
       return 2;
   }
   else if(exp=='^')
   {
       return 3;
   }
   else if(exp=='('||exp==')')
   {
       return 4;
   }
   return 0;
}
void InfixToPostfix(struct stack*st,char *exp,int max,char *postfix)
{
   int k=0;
   for(int i=0;i<max;i++)
   {
       int m=isoperator(exp[i]);
```

```
   if(!m)
   {
        postfix[k++]=exp[i];
   }
   else
   {
     if(st->top==-1)
     {
        //if stack is empty push the operator to the stack
        push(st,exp[i]);

     }
     else
     {
       while(st->top!=-1 && precedence(exp[i])<=precedence(st->s[st->top]))
       {
          postfix[k++]=pop(st);//if the stack is not empty check the precedence condition
        }

      push(st,exp[i]);//if precedence greater then push to the stack
     }


   }
  }
while(st->top!=-1)
{
    postfix[k++]=pop(st);
} //if reached end of expression pop all the elements in stack

postfix[k] = '\0';
```

```
}
```



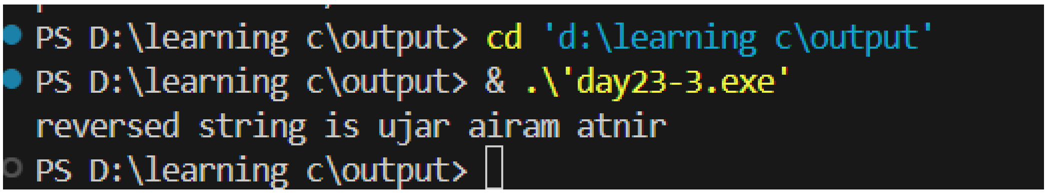
```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day23-2.exe'
infix :a+b*c-d/e
postfix :abc*+de/-
PS D:\learning c\output> 
```

3. //reverse a string using stack

```
#include<stdio.h>

#include<stdlib.h>

#include<string.h>

struct stack

{

   int size;

   int top;

   char *s;

};

void create(struct stack *, char *);

void push(struct stack *, char );

char pop(struct stack *);


int main()

{

   struct stack st;

   char expression[]="rinta maria raju";

   int m=strlen(expression);

   char reverse[m+1];

   create(&st, expression);

   for(int i=0;i<m;i++)//pushing each character of the expression to the stack
```

```
    {
        push(&st, expression[i]);


    }
    for(int i=0;i<m;i++)//poping out each character
    {
        reverse[i]=pop(&st);//since it is LIFO in stack add the popped char in order to the reverse array
    }
    reverse[m]='\0';
    printf("reversed string is %s \n",reverse);




    return 0;
}
void create(struct stack *st, char *exp) {
    st->size = strlen(exp);//the length of the expression
    st->top = -1;
    st->s = (char *)malloc(st->size * sizeof(char));
}
void push(struct stack *st, char exp) {
    if (st->top == st->size - 1) {
        printf("stack overflow \n");
    } else {
        st->top++;

        st->s[st->top] = exp;



    }
}
```

```
char pop(struct stack *st) {
   char x;
   if (st->top == -1) {
      printf("stack underflow \n");
   } else {
      x = st->s[st->top];
      st->top--;
   }
   return x;
}
```

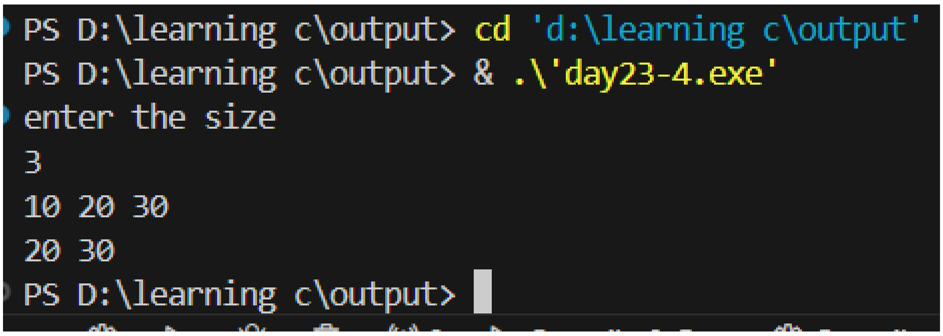


```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day23-3.exe'
reversed string is ujar airam atnir
PS D:\learning c\output>
```

4. //queue

```
#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 *);

void display(struct Queue *);

int main()
```

```
{
   //queue intialization
   struct Queue q;
   printf("enter the size \n");
   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);
   display(&q);
   Dequeue(&q);
   display(&q);
   return 0;
}
void Enqueue(struct Queue *q,int x)
{
   if(q->rear==q->size-1)
   {
      printf("queue is full \n");
   }
   else
   {
      q->rear++;
      q->Q[q->rear]=x;
      //printf("%d",x);
   }
}
int Dequeue(struct Queue *q)
{
  int x;
  if(q->front==q->rear)
  {
    printf("queue empty \n");
  }
  else
  {
    q->front++;
    x=q->Q[q->front];
    return x;
  }
}

void display(struct Queue *q)
{


       // printf("Queue: ");
        for (int i = q->front+1; i <= q->rear; i++)
        {
            printf("%d ", q->Q[i]);
        }
        printf("\n");



}
```



```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day23-4.exe'
enter the size
3
10 20 30
20 30
PS D:\learning c\output> 
```

5. /*.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>


struct Queue {

    int size;

    int front;

    int rear;

    long int *Q;

};


void Enqueue(struct Queue *q, long int );

long int Dequeue(struct Queue *q);

void display(struct Queue *q);


int main() {

    struct Queue q;

    printf("Enter the maximum number of calls: ");

    scanf("%d", &q.size);

    q.Q = (long int *)malloc(q.size *sizeof(long int));

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

```
int choice;
long int ph;
while (1) {
    printf("\n1. Add call\n");
    printf("2. Remove call\n");
    printf("3. View calls\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);
    switch (choice) {
        case 1:
            printf("Enter callid: ");
            scanf("%ld", &ph);
            Enqueue(&q, ph);
            break;
        case 2:
            Dequeue(&q);
            break;
        case 3:
            display(&q);
            break;
        case 4:
            free(q.Q);  // Free allocated memory
            return 0;
        default:
            printf("Invalid choice, please try again.\n");
            break;
    }
}
```

```
    return 0;
}

void Enqueue(struct Queue *q, long int ph) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->Q[q->rear] = ph;
    }
}

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

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("%ld ", q->Q[i]);
    }
    printf("\n");
  }
}
```

```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day23-5.exe'
Enter the maximum number of calls: 3

1. Add call
2. Remove call
3. View calls
4. Exit
Enter your choice: 1
Enter callid: 1

1. Add call
2. Remove call
3. View calls
4. Exit
Enter your choice: 1
Enter callid: 2

1. Add call
2. Remove call
3. View calls
4. Exit
Enter your choice: 1
Enter callid: 3

1. Add call
```

```
2. Remove call
3. View calls
4. Exit
Enter your choice: 3
Calls in queue: 1 2 3

1. Add call
2. Remove call
3. View calls
4. Exit
Enter your choice: 3
Calls in queue: 1 2 3

1. Add call
2. Remove call
3. View calls
4. Exit
Enter your choice: 4
PS D:\learning c\output>
```

```
6.  /*.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>

struct Queue {
   int size;
   int front;
```

```c
    int rear;
    long int *Q;
};

void Enqueue(struct Queue *q, long int job);
void DequeueUntil(struct Queue *q, long int jobId);
void display(struct Queue *q);

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

    int choice;
    long int job;
    while (1) {
        printf("\n1. Add job\n");
        printf("2. Remove job until specific ID\n");
        printf("3. View jobs\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter job ID: ");
                scanf("%ld", &job);
                Enqueue(&q, job);
                break;
            case 2:
                printf("Enter job ID to dequeue up to: ");
                scanf("%ld", &job);
                DequeueUntil(&q, job);
                break;
            case 3:
                display(&q);
                break;
            case 4:
                free(q.Q);  // Free allocated memory
                return 0;
            default:
                printf("Invalid choice, please try again.\n");
                break;
        }
    }

    return 0;
}

void Enqueue(struct Queue *q, long int job) {
    if (q->rear == q->size - 1) {
        printf("Queue is full\n");
    } else {
        q->rear++;
        q->Q[q->rear] = job;
    }
}

void DequeueUntil(struct Queue *q, long int jobId) {
    if (q->front == q->rear) {
        printf("Queue is empty\n");
    } else {
        while (q->front != q->rear && q->Q[q->front + 1] <= jobId) {
            q->front++;
            printf("Removed job: %ld\n", q->Q[q->front]);
        }
    }
}

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

```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day23-6.exe'
Enter the maximum number of jobs: 3

1. Add job
2. Remove job until specific ID
3. View jobs
4. Exit
Enter your choice: 1
Enter job ID: 1

1. Add job
```

```
2. Remove job until specific ID
3. View jobs
4. Exit
Enter your choice: 1
Enter job ID: 2

1. Add job
2. Remove job until specific ID
3. View jobs
4. Exit
Enter your choice: 1
Enter job ID: 3

1. Add job
2. Remove job until specific ID
3. View jobs
4. Exit
Enter your choice: 2
Enter job ID to dequeue up to: 2
Removed job: 1
Removed job: 2

1. Add job
2. Remove job until specific ID
3. View jobs
4. Exit
Enter your choice: 3
Jobs in queue: 3

1. Add job
2. Remove job until specific ID
```

```
3. View jobs
4. Exit
Enter your choice: 4
PS D:\learning c\output>
```

```
7.  /*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 Queue {
    int size;
    int front;
    int rear;
    char (*Q)[20];
};

void Enqueue(struct Queue *q, char *name);
void Dequeue(struct Queue *q);
void display(struct Queue *q);

int main() {
    struct Queue q;
    printf("Enter the maximum number of people in the queue: ");
    scanf("%d", &q.size);
    q.Q = (char (*)[20])malloc(q.size * sizeof(*q.Q));
    q.front = q.rear = -1;

    int choice;
    char name[20];
    while (1) {
        printf("\n1. Join Queue\n");
        printf("2. Buy Ticket\n");
        printf("3. View Queue\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter name: ");
                scanf("%s", name);
                Enqueue(&q, name);
                break;
            case 2:
                Dequeue(&q);
                break;
            case 3:
                display(&q);
                break;
            case 4:

                return 0;
            default:
                printf("Invalid choice, please try again.\n");
                break;
        }
    }

    return 0;
}
```

```
void Enqueue(struct Queue *q, char *name) {
   if (q->rear == q->size - 1) {
      printf("Queue is full\n");
   } else {
      q->rear++;
      strncpy(q->Q[q->rear], name, 20);
   }
}

void Dequeue(struct Queue *q) {
   if (q->front == q->rear) {
      printf("Queue is empty\n");
   } else {
      q->front++;
      printf("Ticket bought by: %s\n", q->Q[q->front]);
   }
}

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

```
PS D:\learning c\output> cd 'd:\learning c\output'
PS D:\learning c\output> & .\'day23-7.exe'
Enter the maximum number of people in the queue: 3

1. Join Queue
2. Buy Ticket
3. View Queue
4. Exit
Enter your choice: 1
Enter name: rinta

1. Join Queue
2. Buy Ticket
3. View Queue
4. Exit
Enter your choice: 1
Enter name: riya

1. Join Queue
2. Buy Ticket
3. View Queue
4. Exit
Enter your choice: 1
Enter name: raju

1. Join Queue
2. Buy Ticket
3. View Queue
4. Exit
Enter your choice: 2
Ticket bought by: rinta
```

```
1. Join Queue
2. Buy Ticket
3. View Queue
4. Exit
Enter your choice: 3
People in queue: riya raju

1. Join Queue
2. Buy Ticket
3. View Queue
4. Exit
Enter your choice: 4
PS D:\learning c\output>
```

8. //queue using linkedlist

```
#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 *q, int value);
int dequeue(struct Queue *q);
void display(struct Queue *q);
```

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

    int choice, value;
    while (1) {
        printf("\n1. 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 value to enqueue: ");
                scanf("%d", &value);
                enqueue(&q, value);
                break;
            case 2:
                value = dequeue(&q);
                if (value != -1) {
                    printf("Dequeued value: %d\n", value);
                }
                break;
            case 3:
                display(&q);
                break;
            case 4:
                return 0;
            default:
                printf("Invalid choice, please try again.\n");
        }
    }

    return 0;
}

void enqueue(struct Queue *q, int value) {
    struct Node *newNode = (struct Node *)malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed\n");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;
    if (q->rear == NULL) {
        q->front = q->rear = newNode;
    } else {
        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");
    } else {
        struct Node *temp = q->front;
        printf("Queue: ");
        while (temp != NULL) {
            printf("%d ", temp->data);
            temp = temp->next;
        }
        printf("\n");
    }
}
```

```
PS D:\learning c\output> & .\'day23-8.exe'

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 2

1. Enqueue
```

```
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 3

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 1
Enter value to enqueue: 4

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 2
Dequeued value: 2

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue: 3 4

1. Enqueue
2. Dequeue
3. Display
4. Exit
```

```
Enter your choice: 2
Dequeued value: 3

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 3
Queue: 4

1. Enqueue
2. Dequeue
3. Display
4. Exit
Enter your choice: 4
PS D:\learning c\output>
```