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

typedef struct stack{
   int size;
   int top;
   char *S;
}STACK;

void create(STACK *);
void push(STACK*,char);
void display(STACK*);
int pop(STACK*);
int peek(STACK*,int);
int isBalance(char*);

int main(){
   STACK st;
   char exp[]="((a+b)*(c-d))";
   //create(&st);
   int balanceValue=isBalance(exp);
   if(balanceValue==1){
      printf("expression is balanced\n");
   }
   else{
      printf("expression is not balanced\n");
   }
   return 0;
}

// void create(STACK *st){
//    printf("enter size:");
//    scanf("%d",&st->size);
//    st->top=-1;
//    st->S=(int *)malloc(st->size*sizeof(int));

// }

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

}

void display(STACK *st){
   int temp=st->top; //temp is used bcoz since directly passing the memory address the fn changes the index;

   while(temp>=0){
      printf("%d\n",st->S[temp]);
```

```
      temp--;
    }
  }
  //or
  /*
  void display(STACK *st){
    for(int i=st->top;i>=0;i--){
      printf("%d\n",st->S[i]);
    }
  }
  */
  int pop(STACK *st){
    int x=-1;
    if(st->top==-1){
      printf("stack underflow\n");
    }
    else{
      x=st->S[st->top];
      st->top--;
    }
    return x;
  }

  int peek(STACK *st,int index){
    int data=-1;
    int temp=st->top;
    if(temp<0){
      printf("empty stack\n");
    }
    while(index<=temp){
      if(temp==index){
        data=st->S[temp];
        return data;
      }
      else{
        temp--;
      }
    }
  }

  int isBalance(char *exp){
    STACK st;
    st.size=strlen(exp);
    st.top=-1;
    st.S=(char*)malloc(st.size*sizeof(char));
    for(int i=0;exp[i]!='\0';i++){
      if(exp[i]=='('){
        push(&st,exp[i]);
      }
      else if(exp[i]==')'){
        if(st.top<0){
          printf("stack is empty\n");
          return 0;
        }
        pop(&st);
      }
```

```
    }
    return (st.top<0)?1:0;
}
```

-_

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

typedef struct stack{
    int size;
    int top;
    char *S;
}STACK;

void create(STACK *,char*);
void push(STACK*,char);
void display(STACK*);
int pop(STACK*);
int peek(STACK*,int);

void infixTOpostfix(STACK*,char*);
int precedence(char);

int main(){
    STACK st;
    char exp[]="a+b*c-d";
    create(&st,exp);

    infixTOpostfix(&st,exp);
    return 0;
}

void create(STACK *st,char *exp){

    st->size=strlen(exp);
    st->top=-1;
    st->S=(char*)malloc(st->size*sizeof(char));

}

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

}

void display(STACK *st){
    int temp=st->top; //temp is used bcoz since directly passing the memory address the fn changes the index;
```

```
    while(temp>=0){
        printf("%d\n",st->S[temp]);
        temp--;
    }
}

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


int precedence(char op){
    if(op=='+'||op=='-'){
        return 1;
    }
    else if(op=='*'||op=='/'){
        return 2;
    }
}
void infixTOpostfix(STACK* st,char *exp){
    char *postfix=(char*)malloc(strlen(exp)+1);
    int j=0;
    for(int i=0;exp[i]!='\0';i++){
        char ch=exp[i];
        if(isalpha(ch)){
            postfix[j++]=ch;
        }
        else if(ch=='+'||ch=='-'||ch=='/'||ch=='*'){

            while(st->top!=-1 && precedence(st->S[st->top])>=precedence(ch)){
                postfix[j++]=pop(st);
            }
            push(st,ch);
        }
    }
    while(st->top!=-1){
        postfix[j++]=pop(st);
    }
    postfix[j]='\0';
    printf("postfix expression:%s\n",postfix);
    free(postfix);
}
```

-_

```
//Reverse a String Using Stack
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
```

```
typedef struct stack{
   int size;
   int top;
   char *S;
}STACK;

void create(STACK *,char *);
void insert(STACK *,char *);
void rev(STACK *);

int main(){
   char *exp;
   exp=(char *)malloc(20*sizeof(char));
   if(exp==NULL){
      printf("memory allocation failed\n");
   }
   STACK st; //creating a actual stack variable using structure
   STACK *reversed;
   printf("enter the string to be reversed:");
   scanf(" %[^\n]",exp);
   create(&st,exp);
   insert(&st,exp);
   rev(&st);

   return 0;

}

void create(STACK *st,char *exp){


   st->size=20;
   st->top=-1;
   st->S=exp;
}

void insert(STACK *st,char *exp){
   if(st->top==st->size-1){
      printf("stack is full\n");
      return;
   }
   for(int i=0;i<st->size && exp[i]!='\0';i++){
      st->top++;
      st->S[st->top]=exp[i];
   }

}

void rev(STACK *st){
   if(st->top==-1){
      printf("stack is empty\n");
   }
   while(st->top>=0){
      printf("%c",st->S[st->top]);
      st->top--;
```

```
   }

}
```

//queue_array_enqueue_dequeue

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

typedef struct queue{
   int size;
   int front;
   int rear;
   int *Q;
}QUEUE;

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

int main(){
   QUEUE q;
   int dequeVlaue;
   create(&q);
   enque(&q,10);
   enque(&q,20);
   enque(&q,30);
   display(&q);

   printf("dequed value is:%d\n",deque(&q));
   printf("dequed value is:%d\n",deque(&q));

   return 0;
}

void create(QUEUE *q){
   printf("enter size:");
   scanf("%d",&q->size);
   q->Q=(int*)malloc(q->size*sizeof(int));
   q->front=q->rear=-1;
}

void enque(QUEUE *q,int n){
   if(q->rear==q->size-1){
      printf("queue is full\n");
   }
   else{
      q->rear++;
      q->Q[q->rear]=n;
   }
}
void display(QUEUE *q){
   int i=0;
```

```
    if(q->rear==q->front){
        printf("queue is empty\n");
    }
    while(i<=q->rear){
        printf("%d\n",q->Q[i]);
        i++;
    }
}
int deque(QUEUE *q){
    int x=-1;
    if(q->front==q->rear){
        printf("queue is empty");
    }
    else{
        q->front++;
        x=q->Q[q->front];

    }
    return x;
}
```

-_

//**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>

typedef struct queue{
    int size;
    int front;
    int rear;
    char **Q;
}QUEUE;

void create(QUEUE*);
void add(QUEUE *,char*);
void view(QUEUE *);
char *callremove(QUEUE *);

int main(){
    QUEUE q;
    char *rem;
    create(&q);
    add(&q,"call 1");
    add(&q,"call 2");
    add(&q,"call 3");
    add(&q,"call 4");
    view(&q);
    rem=callremove(&q);
    printf("call removed is %s\n",rem);

    return 0;
}
```

```
void create(QUEUE *q){
  printf("enter size:");
  scanf("%d",&q->size);
  q->Q=(char**)malloc(q->size*sizeof(char*));
  q->front=q->rear=-1;
}

void add(QUEUE *q,char *exp){
  if(q->rear==q->size-1){
    printf("queue is full\n");
  }
  else{
    q->rear++;
    q->Q[q->rear]=(char*)malloc((strlen(exp)+1)*sizeof(char));
    strcpy(q->Q[q->rear],exp);
  }
}
void view(QUEUE *q){
  int i=0;
  if(q->rear==q->front){
    printf("queue is empty\n");
  }
  while(i<=q->rear){
    printf("%s\n",q->Q[i]);
   i++;
  }
}
char* callremove(QUEUE *q){

  if(q->rear==q->front){
    printf("queue is empty\n");

  }
  else{
    q->front++;
    char *x=q->Q[q->front];
    return x;
  }
}
```

-_