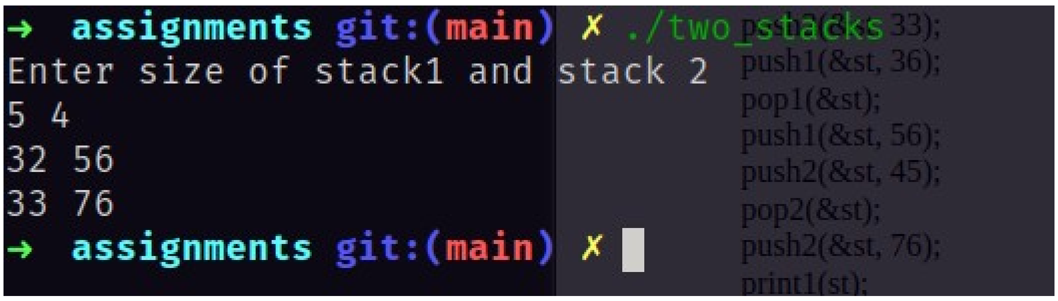
Name – Bankar Krushna Lahanubhau
MIS – 612303031
Batch – S2

Assignment 3

two_stacks.c

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

typedef struct two_stacks{
    int *A;
    int size1, size2;
    int top1, top2;
}two_stacks;

void init_two_stacks(two_stacks *st, int size1, int size2) {
    st->A = (int*)malloc(sizeof(int)*(size1 + size2));
    st->size1 = size1;
    st->size2 = size2;
    st->top1 = -1;
    st->top2 = -1;
    return;
}

int isFull1(two_stacks st) {
    if(st.top1 >= st.size1 - 1) return 1;
    return 0;
}

int isFull2(two_stacks st) {
    if(st.top2 >= st.size2 - 1) return 1;
    return 0;
}

int isEmpty1(two_stacks st){
    if(st.top1 == -1) return 1;
    return 0;
}

int isEmpty2(two_stacks st){
    if(st.top2 == -1) return 1;
    return 0;
}
void push1(two_stacks *st, int data) {
    if(isFull1(*st)) {
        printf("Stack 1 is full\n");
        return;
    }
    st->top1++;
    st->A[st->top1] = data;
    return;
}

void push2(two_stacks *st, int data) {
    if(isFull2(*st)) {
        printf("Stack 2 is full\n");
        return;
    }
    st->top2++;
    st->A[st->size1 + st->top2] = data;
    return;
}

void pop1(two_stacks *st) {
    if(isEmpty1(*st)){
        printf("Stack 1 is empty\n");
        return;
    }
    st->top1--;
    return;
}

void pop2(two_stacks *st) {
    if(isEmpty2(*st)){
        printf("Stack 2 is empty\n");
        return;
    }
    st->top2--;
    return;
}

void print1(two_stacks st) {
    for(int i = 0; i <= st.top1; i++) {
        printf("%d ", st.A[i]);
    }
    printf("\n");
    return;
}

void print2(two_stacks st) {
    for(int i = st.size1; i <= (st.size1 + st.top2); i++) {
        printf("%d ", st.A[i]);
    }
    printf("\n");
    return;
}
int main() {
    two_stacks st;
    printf("Enter size of stack1\n");
    int size1, size2;
    scanf("%d%d", &size1, &size2);
    init_two_stacks(&st, size1, size2);
    push1(&st, 32);
    push2(&st, 33);
    push1(&st, 36);
    pop1(&st);
    push1(&st, 56);
    push2(&st, 45);
    pop2(&st);
    push2(&st, 76);
    print1(st);
    print2(st);
    return 0;
}
```

OUTPUT:



```
→ assignments git:(main) ✗ ./two_stacks
Enter size of stack1 and stack 2
5 4
32 56
33 76
→ assignments git:(main) ✗
```

---

char_stack

stack.h

```
typedef struct node {
    char data;
    struct  node *next;
}node;

typedef struct stack {
    node *top;
}stack;

void init_stack(stack *s);
void push(stack *s, char data);
char pop(stack *s);
char peek(stack s);
int is_empty(stack s);
```

stack.c

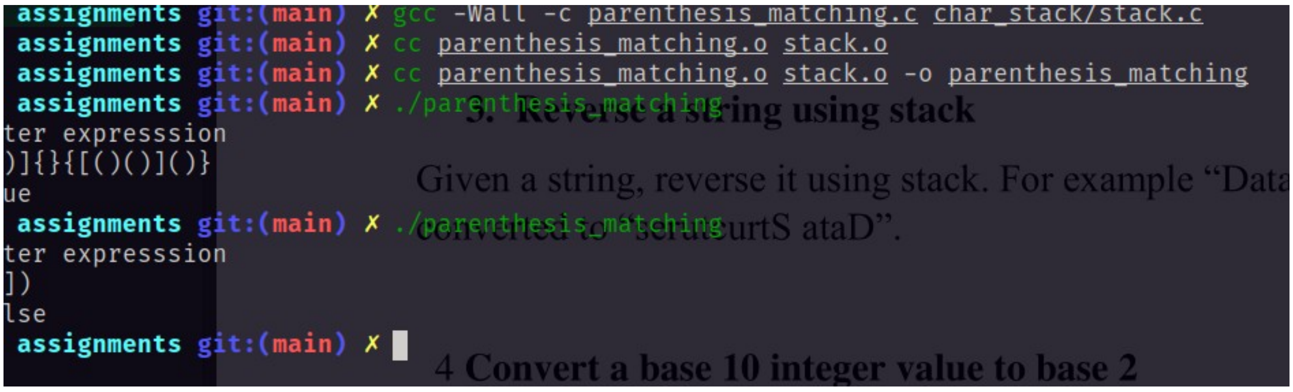
```
#include <stdlib.h>
#include "stack.h"
#include <limits.h>

void init_stack(stack *s) {
    s->top = NULL;
    return;
}
```

```
int is_empty(stack s) {
    if(s.top == NULL) {
        return 1;
    }
    return 0;
}

void push(stack *s, char data) {
    node *nn = (node*)malloc(sizeof(node));

    if(nn) {
        nn->data = data;
        nn->next = NULL;
    }
    else {
        return;
    }

    nn->next = s->top;
    s->top = nn;

    return;
}
char pop(stack *s) {
    node *p = s->top;
    char data;
    if(is_empty(*s)) return '\0';
    s->top = s->top->next;
    data = p->data;
    free(p);
    return data;
}

char peek(stack s) {
    return s.top->data;
}
```

paranthesis_matching.c

```
#include <stdio.h>
#include "int_stack/stack.h"

int is_matching_pair(char opening, char closing) {
    return (opening == '(' && closing == ')') ||
           (opening == '[' && closing == ']') ||
           (opening == '{' && closing == '}');
}

int check_parenthesis(char *str) {
    stack st;
    init_stack(&st);
    int i = 0;
    while(str[i] != '\0') {
        if(str[i] == '(' || str[i] == '[' || str[i] == '{') {
            push(&st, str[i]);
        }
        else {
            if(is_empty(st)) return 0;

            if(is_matching_pair(peek(st), str[i])){
                pop(&st);
            }
            else {
                return 0;
            }
        }
        i++;
    }
    return is_empty(st);
}

int main() {
    char str[100];
    printf("Enter expresssion\n");
    scanf("%[^\n]", str);
    if(check_parenthesis(str)) {
        printf("True\n");
    }
    else {
        printf("False\n");
    }
    return 0;
}
```

output



```
 assignments git:(main) ✗ gcc -Wall -c parenthesis_matching.c char_stack/stack.c
 assignments git:(main) ✗ cc parenthesis_matching.o stack.o
 assignments git:(main) ✗ cc parenthesis_matching.o stack.o -o parenthesis_matching
 assignments git:(main) ✗ ./parenthesis_matching
ter expression
)]{}{[()()]()}
ue
 assignments git:(main) ✗ ./parenthesis_matching
ter expression
])
lse
 assignments git:(main) ✗ 
```

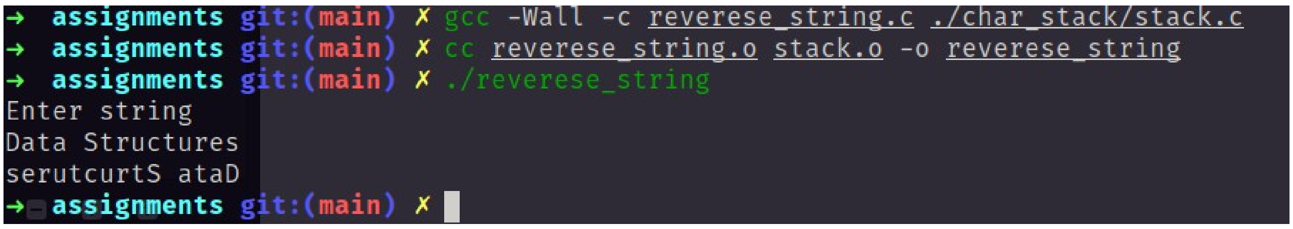
----

reverse_string.c

```
#include <stdio.h>
#include "char_stack/stack.h"
```

```
void reverse_string(char *str) {
    stack st;
    init_stack(&st);
    int i = 0;
    while(str[i] != '\0') {
        push(&st, str[i]);
        i++;
    }
    i = 0;
    while(!is_empty(st)) {
        str[i++] = pop(&st);
    }
    return;
}

int main() {
    char str[100];
    printf("Enter string\n");
    scanf("%[^\n]", str);
    reverse_string(str);
    printf("%s\n", str);
    return 0;
}
```

OUTPUT



```
→  assignments git:(main) ✗ gcc -Wall -c reverese_string.c ./char_stack/stack.c
→  assignments git:(main) ✗ cc reverese_string.o stack.o -o reverese_string
→  assignments git:(main) ✗ ./reverese_string
Enter string
Data Structures
serutcurtS ataD
→  assignments git:(main) ✗
```

----

decimal_to_binary.c

stack.h

```
typedef struct node {
    char data;
    struct  node *next;
}node;

typedef struct stack {
    node *top;
}stack;
```

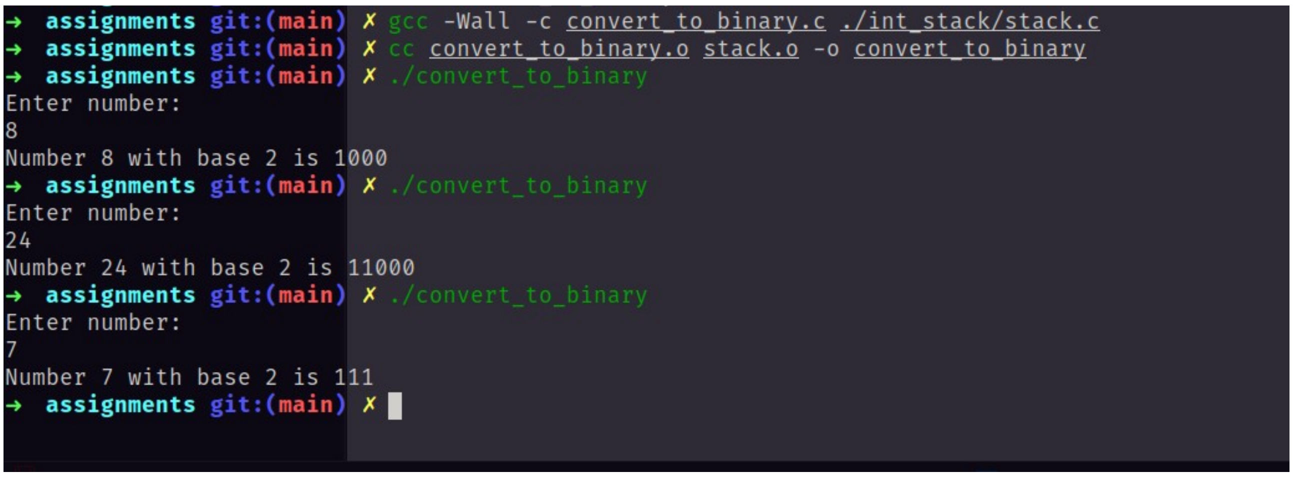
```
void init_stack(stack *s);
void push(stack *s, char data);
char pop(stack *s);
char peek(stack s);
int is_empty(stack s);
```

stack.c

```
#include <stdlib.h>
#include "stack.h"
#include <limits.h>

void init_stack(stack *s) {
   s->top = NULL;
   return;
}

int is_empty(stack s) {
   if(s.top == NULL) {
      return 1;
   }
   return 0;
}

void push(stack *s, char data) {
   node *nn = (node*)malloc(sizeof(node));

   if(nn) {
      nn->data = data;
      nn->next = NULL;
   }
   else {
      return;
   }

   nn->next = s->top;
   s->top = nn;

   return;
}
char pop(stack *s) {
   node *p = s->top;
   char data;
   if(is_empty(*s)) return '\0';
   s->top = s->top->next;
   data = p->data;
   free(p);
   return data;
}

char peek(stack s) {
    return s.top->data;
}
```

covert_to_binary.c

```
#include <stdio.h>
#include "int_stack/stack.h"

int conver_to_binary(int num) {
    stack st;
    int bin = 0;
    init_stack(&st);
    while(num) {
        push(&st, num % 2);
        num /= 2;
    }
    while(!is_empty(st)) {
        bin = bin * 10 + pop(&st);
    }
    return bin;
}

int main() {
    int num, bin;
    printf("Enter number: \n");
    scanf("%d", &num);
    bin = conver_to_binary(num);
    printf("Number %d with base 2 is %d\n", num, bin);
    return 0;
}
```



```
→ assignments git:(main) ✗ gcc -Wall -c convert_to_binary.c ./int_stack/stack.c
→ assignments git:(main) ✗ cc convert_to_binary.o stack.o -o convert_to_binary
→ assignments git:(main) ✗ ./convert_to_binary
Enter number:
8
Number 8 with base 2 is 1000
→ assignments git:(main) ✗ ./convert_to_binary
Enter number:
24
Number 24 with base 2 is 11000
→ assignments git:(main) ✗ ./convert_to_binary
Enter number:
7
Number 7 with base 2 is 111
→ assignments git:(main) ✗
```