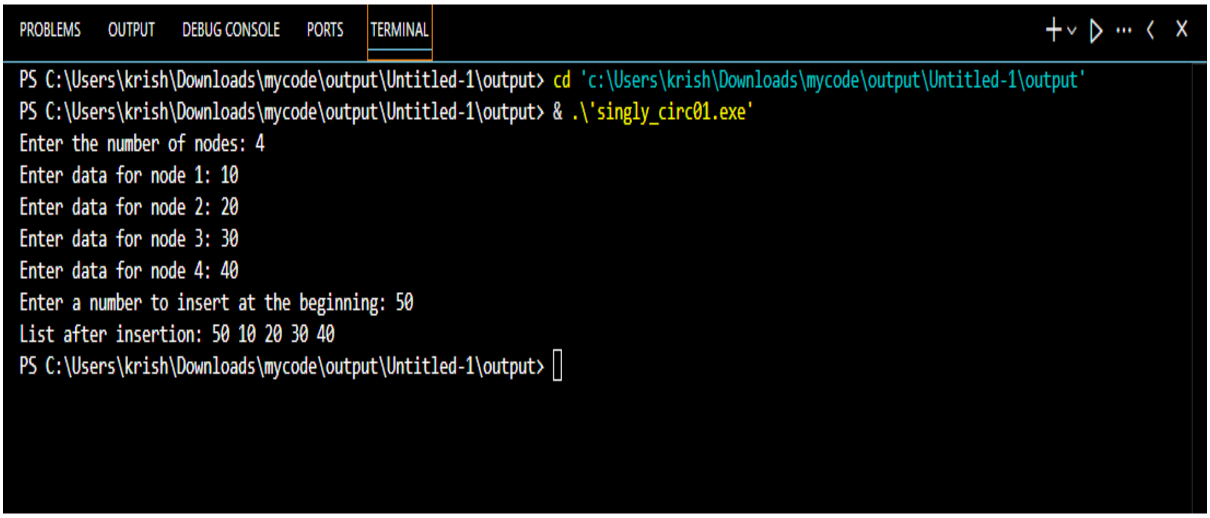
```
/*insertion at the begining of the circular singly list Input:number of nodes:4 (10 20 30 40)
Input:50 output:50 10 20 30 40 */

#include <stdio.h>

#include <stdlib.h>

struct node

{ // Define the node structure

   int data;

   struct node *link;

}; // Removed global head declaration here// Function declarations// Pass a pointer to the head
pointer to modify the original head

void add_at_begining(struct node **head_ref, int data);

void print_data(struct node *head); // Declaration for print function

int main()

{

   struct node *head = NULL;    // Initialize head locally in main

   struct node *newNode, *temp; // Declare necessary pointers

   int n, i, data;

   printf("Enter the number of nodes: ");

   scancf("%d", &n);

   if (n < 0)

   { // Input validation for number of nodes

      printf("Number of nodes cannot be negative.\n");

      return 1; // Indicate error

   }

   // Build the initial circular list

   for (i = 0; i < n; i++)

   {

      newNode = (struct node *)malloc(sizeof(struct node));

      // Check if malloc failed

      if (newNode == NULL)

      {
          perror("Memory allocation failed"); // Clean up already allocated memory before exiting (optional but good practice)  // ... (free list nodes) ...
          return 1;
      }
      printf("Enter data for node %d: ", i + 1);
      scanf("%d", &newNode->data);
      newNode->link = NULL; // Initialize link
      if (head == NULL)
      { // If list is empty
          head = newNode;
          head->link = head; // Point to itself to make it circular
      }
      else
      { // If list is not empty
          temp = head;
          // Traverse to the last node
          while (temp->link != head)
          {
              temp = temp->link;
          }
          temp->link = newNode; // Link last node to the new node
          newNode->link = head; // Link new node back to the head
      }
   } // Insert at the beginning if the list was created
   if (n >= 0)
   { // Proceed only if initial list creation was attempted
      printf("Enter a number to insert at the beginning: ");
      scanf("%d", &data);
      add_at_begining(&head, data); // Pass the address of head
      printf("List after insertion: ");
      print_data(head); // Print the updated list
```

```
}
else
{
    printf("List is empty, cannot insert.\n"); // Handle case where n was 0 initially
}
// Free allocated memory (important for circular lists)
if (head != NULL)
{
    struct node *current = head->link;
    struct node *nextNode;
    head->link = NULL; // Break the circle first
    while (current != NULL)
    {
        nextNode = current->link;
        free(current);
        current = nextNode;
    }
    // Simplified freeing for this example:
    if (head != NULL)
    {
        temp = head->link;
        while (temp != head)
        {
            struct node *next = temp->link;
            free(temp);
            temp = next;
            // Safety break if something went wrong
            if (temp == head->link)
                break;
        }
        free(head);
```

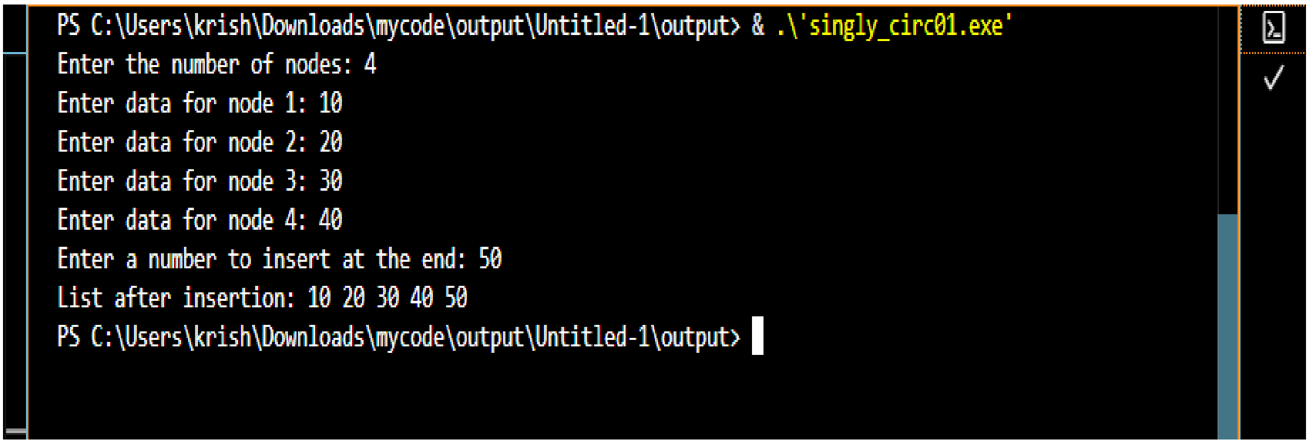
```
      }
    }
    return 0;
}
// Function to add a node at the beginning of the circular list
// Takes a pointer to the head pointer (**head_ref)
void add_at_begining(struct node **head_ref, int data)
{
    // Allocate memory for the new node
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    if (newNode == NULL)
    {
      perror("Memory allocation failed for new node");
      return; // Exit function if allocation fails
    }
    newNode->data = data;

    if (*head_ref == NULL)
    { // If the list is currently empty
      *head_ref = newNode;
      newNode->link = *head_ref; // Point to itself
    }
    else
    { // If the list is not empty
      struct node *last = *head_ref;
      // Find the last node (the one pointing to the current head)
      while (last->link != *head_ref)
      {
        last = last->link;
      }
      newNode->link = *head_ref; // New node points to the current head
```

```
        last->link = newNode;     // Last node points to the new node
        *head_ref = newNode;      // Update the head pointer to the new node
    }
}
// Function to print the data in the circular linked list
void print_data(struct node *head)
{
    if (head == NULL)
    {
        printf("List is empty.\n");
        return;
    }
    struct node *temp = head;
    do
    {
        printf("%d ", temp->data);
        temp = temp->link;
    } while (temp != head); // Continue until we loop back to the head
    printf("\n");
}
```



```
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> cd 'c:\Users\krish\Downloads\mycode\output\Untitled-1\output'
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> & .\'singly_circ01.exe'
Enter the number of nodes: 4
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30
Enter data for node 4: 40
Enter a number to insert at the beginning: 50
List after insertion: 50 10 20 30 40
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output>
```

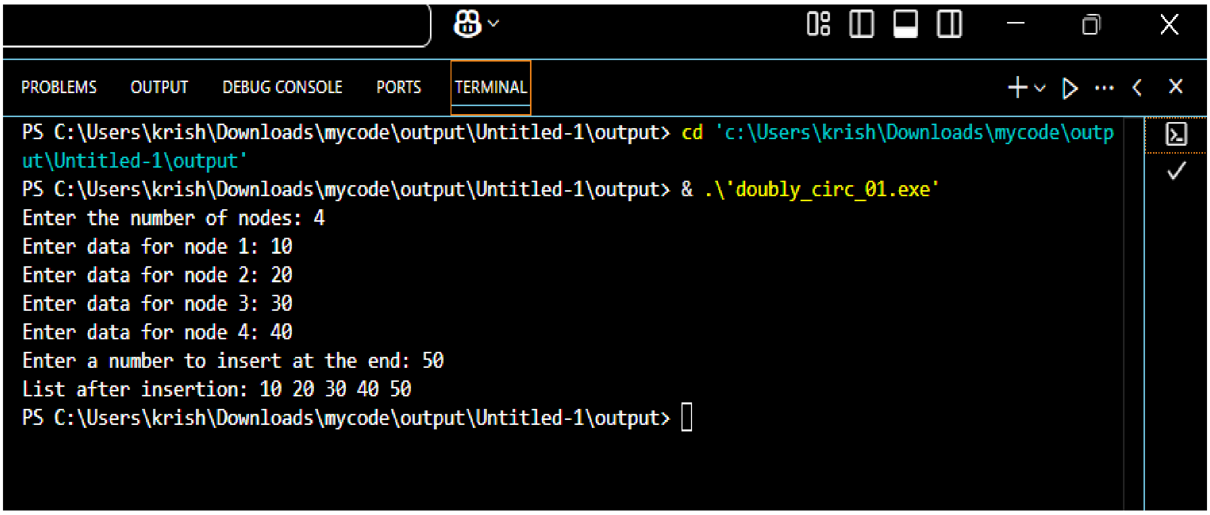
```
/*singly circular linked list to perform insertion at end Input:10 20 30 40 Input:50 output:10 20
30 40 50 */
#include<stdio.h>
#include<stdlib.h>
struct node {// Define the node structure
    int data;
    struct node *link;
};// Function declarations
void add_at_end(struct node **head_ref, int data);
void print_data(struct node *head);
int main() {
    struct node *head = NULL;
    struct node *newNode, *temp;
    int n, i, data;
    printf("Enter the number of nodes: ");
    scanf("%d", &n);
    if (n < 0) {
        printf("Number of nodes cannot be negative.\n");
        return 1;
    } // Build the initial circular list
    for (i = 0; i < n; i++) {
        newNode = (struct node *)malloc(sizeof(struct node));
        if (newNode == NULL) {
            perror("Memory allocation failed");
            return 1;
        }printf("Enter data for node %d: ", i + 1);
        scanf("%d", &newNode->data);
        newNode->link = NULL;
        if (head == NULL) {
            head = newNode;
            head->link = head;
        } else {
            temp = head;
            while (temp->link != head) {
                temp = temp->link;
            }temp->link = newNode;
            newNode->link = head;
        }
    } // Insert at the end if the list was created
    if (n >= 0) {
        printf("Enter a number to insert at the end: ");
        scanf("%d", &data);
        add_at_end(&head, data);
        printf("List after insertion: ");
        print_data(head);
    } else {
        printf("List is empty, cannot insert.\n");
    }
    return 0;
}// Function to add a node at the end of the circular list
void add_at_end(struct node **head_ref, int data) {
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    if (newNode == NULL) {
        perror("Memory allocation failed for new node");
        return;
    }newNode->data = data;
    if (*head_ref == NULL) {
        *head_ref = newNode;
        newNode->link = *head_ref;
    } else {
        struct node *temp = *head_ref;
        while (temp->link != *head_ref) {
            temp = temp->link;
        }
        temp->link = newNode;
        newNode->link = *head_ref;
    }
}// Function to print the data in the circular linked list
void print_data(struct node *head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }struct node *temp = head;
    do {
        printf("%d ", temp->data);
        temp = temp->link;
    } while (temp != head);
    printf("\n");
}
```



```
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> & .\'singly_circ01.exe'
Enter the number of nodes: 4
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30
Enter data for node 4: 40
Enter a number to insert at the end: 50
List after insertion: 10 20 30 40 50
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output>
```

```
/*doubly circular linked list to perform insertion at end Input:10 20 30 40 Input:50 output:10 20 30 40 50 */
#include <stdio.h>
#include <stdlib.h>

// Define the node structure
struct node {
    int data;
    struct node *prev;
    struct node *next;
};

// Function declarations
void add_at_end(struct node **head_ref, int data);
void print_data(struct node *head);

int main() {
    struct node *head = NULL;
    struct node *newNode, *temp;
    int n, i, data;

    printf("Enter the number of nodes: ");
    scanf("%d", &n);

    if (n < 0) {
        printf("Number of nodes cannot be negative.\n");
        return 1;
    }

    // Build the initial circular doubly linked list
    for (i = 0; i < n; i++) {
        newNode = (struct node *)malloc(sizeof(struct node));
        if (newNode == NULL) {
            perror("Memory allocation failed");
            return 1;
        }
        printf("Enter data for node %d: ", i + 1);
        scanf("%d", &newNode->data);
        newNode->next = newNode->prev = NULL;

        if (head == NULL) {
            head = newNode;
            head->next = head;
            head->prev = head;
        } else {
            temp = head->prev; // Get last node
            temp->next = newNode;
            newNode->prev = temp;
            newNode->next = head;
            head->prev = newNode;
        }
    }

    // Insert at the end if the list was created
    if (n >= 0) {
        printf("Enter a number to insert at the end: ");
        scanf("%d", &data);
        add_at_end(&head, data);
        printf("List after insertion: ");
        print_data(head);
    } else {
        printf("List is empty, cannot insert.\n");
    }

    return 0;
}

// Function to add a node at the end of the circular doubly linked list
void add_at_end(struct node **head_ref, int data) {
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    if (newNode == NULL) {
        perror("Memory allocation failed for new node");
        return;
    }
    newNode->data = data;

    if (*head_ref == NULL) {
        *head_ref = newNode;
        newNode->next = newNode;
        newNode->prev = newNode;
    } else {
        struct node *last = (*head_ref)->prev; // Get last node
        last->next = newNode;
        newNode->prev = last;
        newNode->next = *head_ref;
        (*head_ref)->prev = newNode;
    }
```

```
}

// Function to print the data in the circular doubly linked list
void print_data(struct node *head) {
  if (head == NULL) {
    printf("List is empty.\n");
    return;
  }

  struct node *temp = head;
  do {
    printf("%d ", temp->data);
    temp = temp->next;
  } while (temp != head);

  printf("\n");
}
```

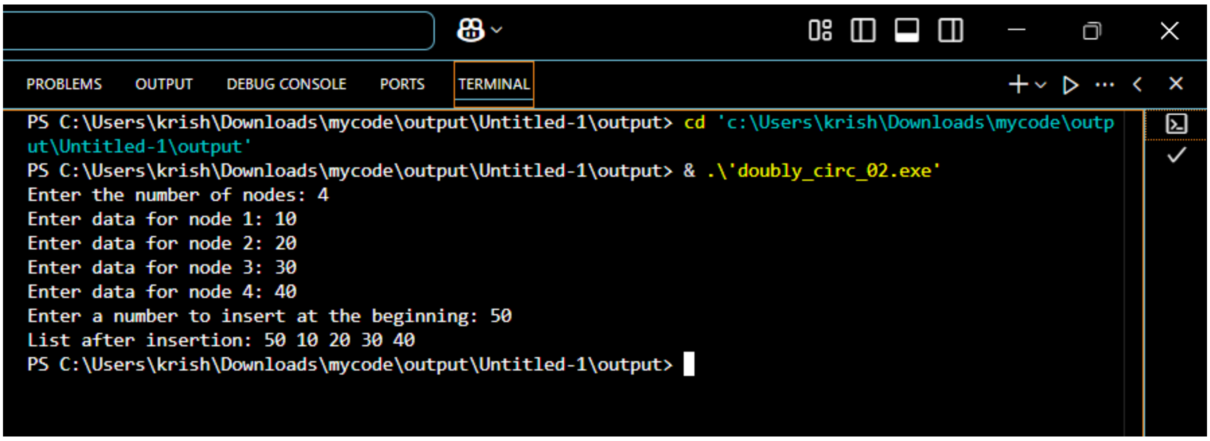


```
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> cd 'c:\Users\krish\Downloads\mycode\outp
ut\Untitled-1\output'
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> & .\'doubly_circ_01.exe'
Enter the number of nodes: 4
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30
Enter data for node 4: 40
Enter a number to insert at the end: 50
List after insertion: 10 20 30 40 50
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> 
```

```
/*doubly circular linked list to perform insertion at begining
Input:10 20 30 40
Input:50
output:10 20 30 40 50 */
#include <stdio.h>
#include <stdlib.h>
// Define the node structure
struct node {
  int data;
  struct node *prev;
  struct node *next;
};

// Function declarations
void add_at_beginning(struct node **head_ref, int data);
void print_data(struct node *head);

int main() {
  struct node *head = NULL;
  struct node *newNode, *temp;
  int n, i, data;

  printf("Enter the number of nodes: ");
```

```
scanf("%d", &n);

if (n < 0) {
    printf("Number of nodes cannot be negative.\n");
    return 1;
}

// Build the initial circular doubly linked list
for (i = 0; i < n; i++) {
    newNode = (struct node *)malloc(sizeof(struct node));
    if (newNode == NULL) {
        perror("Memory allocation failed");
        return 1;
    }
    printf("Enter data for node %d: ", i + 1);
    scanf("%d", &newNode->data);
    newNode->next = newNode->prev = NULL;

    if (head == NULL) {
        head = newNode;
        head->next = head;
        head->prev = head;
    } else {
        temp = head->prev; // Get last node
        temp->next = newNode;
        newNode->prev = temp;
        newNode->next = head;
        head->prev = newNode;
    }
}
```

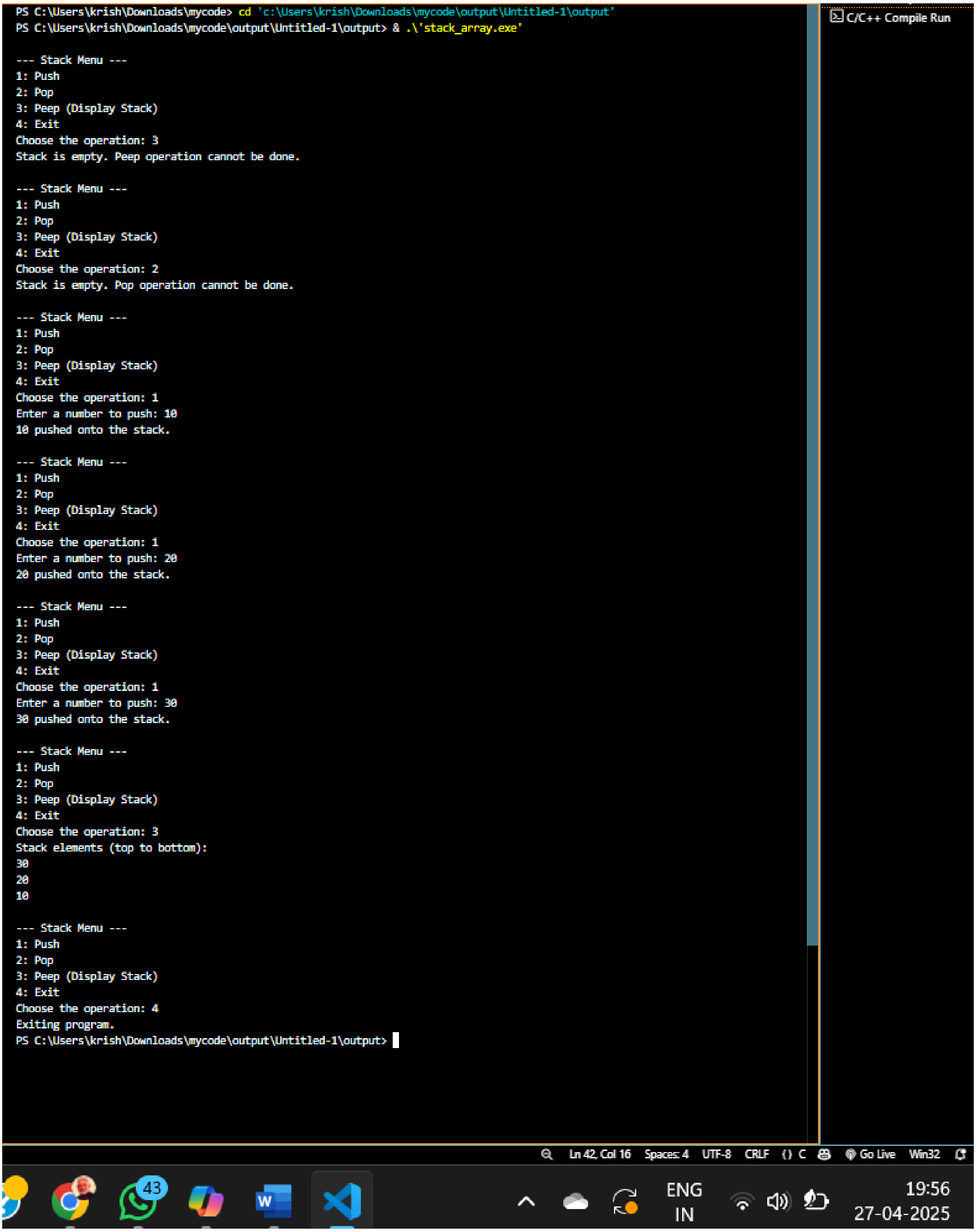
```
    // Insert at the beginning if the list was created
    if (n >= 0) {
        printf("Enter a number to insert at the beginning: ");
        scanf("%d", &data);
        add_at_beginning(&head, data);
        printf("List after insertion: ");
        print_data(head);
    } else {
        printf("List is empty, cannot insert.\n");
    }

    return 0;
}

// Function to add a node at the beginning of the circular doubly linked list
void add_at_beginning(struct node **head_ref, int data) {
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    if (newNode == NULL) {
        perror("Memory allocation failed for new node");
        return;
    }
    newNode->data = data;

    if (*head_ref == NULL) {
        *head_ref = newNode;
        newNode->next = newNode;
        newNode->prev = newNode;
    } else {
        struct node *last = (*head_ref)->prev; // Get last node
        newNode->next = *head_ref;
        newNode->prev = last;
        last->next = newNode;
        (*head_ref)->prev = newNode;
        *head_ref = newNode; // Update the head pointer
    }
}

// Function to print the data in the circular doubly linked list
void print_data(struct node *head) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct node *temp = head;
    do {
        printf("%d ", temp->data);
        temp = temp->next;
    } while (temp != head);

    printf("\n");
}
```



```
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> cd 'c:\Users\krish\Downloads\mycode\outp
ut\Untitled-1\output'
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> & .\'doubly_circ_02.exe'
Enter the number of nodes: 4
Enter data for node 1: 10
Enter data for node 2: 20
Enter data for node 3: 30
Enter data for node 4: 40
Enter a number to insert at the beginning: 50
List after insertion: 50 10 20 30 40
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output>
```

```
/*stack implementation using array*/
#include <stdio.h>
#include <stdlib.h> // Needed for exit()

#define MAX 4
int stack[MAX];
int top = -1; // Initialize top correctly

// Function Prototypes - Correct return types
int isempty();
int isfull();
void push();
void pop();
void peep(); // Also called display or peek

int main() {
    int choice; // Use a local variable for user choice

    // Loop to allow multiple operations
    while (1) {
        printf("\n--- Stack Menu ---\n");
        printf("1: Push\n");
        printf("2: Pop\n");
        printf("3: Peep (Display Stack)\n");
        printf("4: Exit\n");
        printf("Choose the operation: ");

        // Check if scanf successfully reads an integer
        if (scanf("%d", &choice) != 1) {
            printf("Invalid input. Please enter a number.\n");
            // Clear the input buffer
            while (getchar() != '\n');
            continue; // Ask for input again
        }

        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                peep();
                break;
            case 4:
                printf("Exiting program.\n");
                exit(0); // Exit successfully
            default:
                printf("Invalid choice. Please enter a number between 1 and 4.\n");
        }
    }

    // Although the loop is infinite, standard main should return int
    return 0;
```

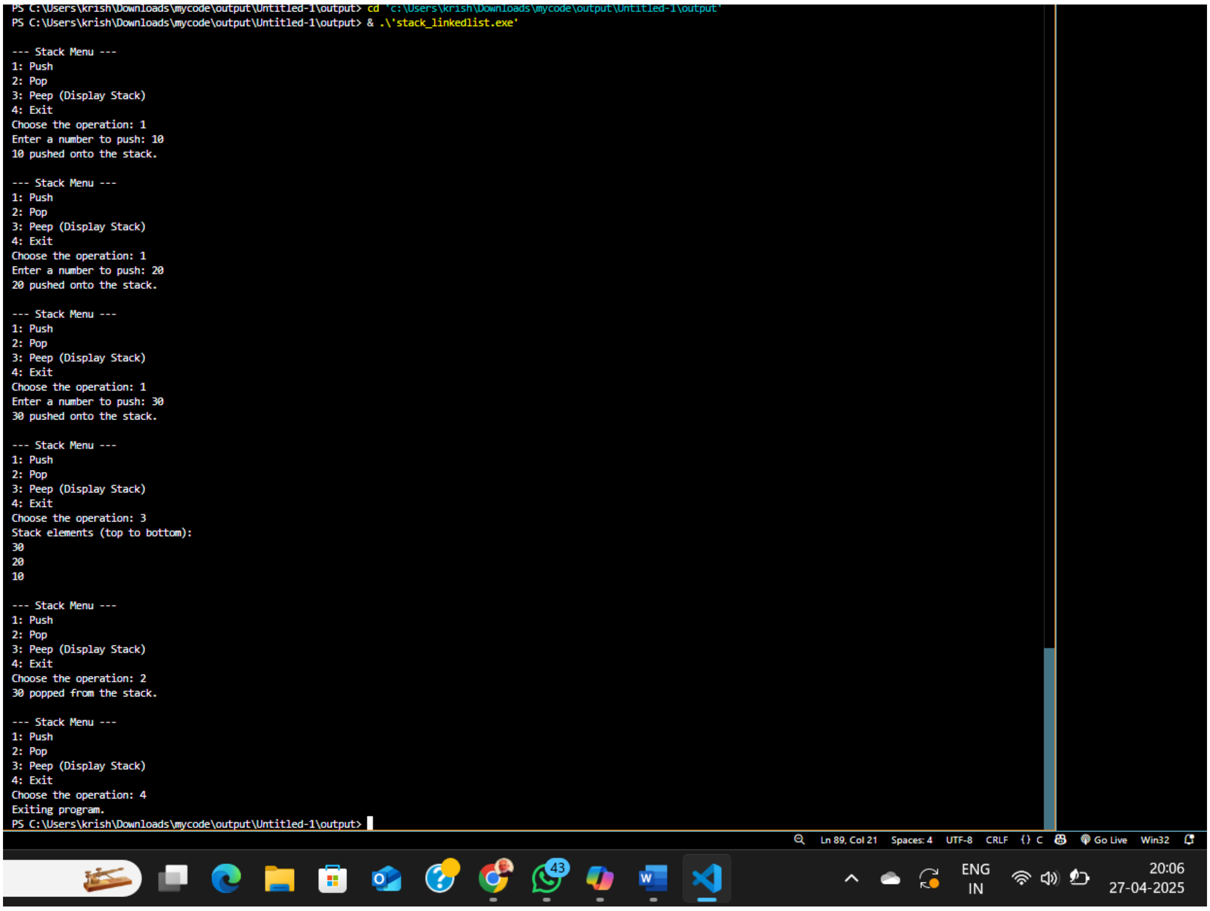
```
}

// Returns 1 if empty, 0 otherwise
int isempty() {
    if (top == -1) {
        return 1; // True, stack is empty
    }
    return 0; // False, stack is not empty
}

// Returns 1 if full, 0 otherwise
int isfull() {
    if (top == (MAX - 1)) {
        return 1; // True, stack is full
    }
    return 0; // False, stack is not full
}

// Push an element onto the stack
void push() {
    int value; // Local variable to store the value to push
    if (isfull()) {
        printf("Stack is full. Push operation cannot be done.\n");
    } else {
        printf("Enter a number to push: ");
        // Check if scanf successfully reads an integer
        if (scanf("%d", &value) != 1) {
            printf("Invalid input. Please enter a number.\n");
            // Clear the input buffer
            while (getchar() != '\n');
            return; // Don't push if input is invalid
        }
        top++;         // Increment top first
        stack[top] = value; // Store the value
        printf("%d pushed onto the stack.\n", value);
    }
}

// Pop an element from the stack
void pop() {
    if (isempty()) {
        printf("Stack is empty. Pop operation cannot be done.\n");
    } else {
        // Retrieve the value before decrementing top
        int popped_value = stack[top];
        top--; // Decrement top to "remove" the element
        printf("%d popped from the stack.\n", popped_value);
        // Note: We don't actually need to erase the value in the array
    }
}

// Display the elements in the stack
void peep() {
    int i; // Local loop counter
    if (isempty()) {
        printf("Stack is empty. Peep operation cannot be done.\n");
    } else {
        printf("Stack elements (top to bottom):\n");
        for (i = top; i >= 0; i--) {
            printf("%d\n", stack[i]);
        }
    }
```



```
PS C:\Users\krish\Downloads\mycode> cd 'c:\Users\krish\Downloads\mycode\output\Untitled-1\output'
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> & .\'stack_array.exe'

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 3
Stack is empty. Peep operation cannot be done.

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 2
Stack is empty. Pop operation cannot be done.

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 1
Enter a number to push: 10
10 pushed onto the stack.

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 1
Enter a number to push: 20
20 pushed onto the stack.

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 1
Enter a number to push: 30
30 pushed onto the stack.

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 3
Stack elements (top to bottom):
30
20
10

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 4
Exiting program.
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> 
```

```
/*stack implementation using linkedlist*/
#include <stdio.h>
#include <stdlib.h>

// Define the node structure
struct node {
    int data;
    struct node *next;
};

// Top pointer (head of the stack)
struct node *top = NULL;

// Function Prototypes
int isempty();
void push();
void pop();
void peep();

int main() {
    int choice;

    while (1) {
        printf("\n--- Stack Menu ---\n");
        printf("1: Push\n");
        printf("2: Pop\n");
        printf("3: Peep (Display Stack)\n");
        printf("4: Exit\n");
        printf("Choose the operation: ");

        if (scanf("%d", &choice) != 1) {
            printf("Invalid input. Please enter a number.\n");
            while (getchar() != '\n');  // Clear input buffer
            continue;
        }

        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                pop();
                break;
            case 3:
                peep();
                break;
            case 4:
                printf("Exiting program.\n");
                exit(0);
            default:
                printf("Invalid choice. Please enter a number between 1 and 4.\n");
        }
    }

    return 0;
}
```

```
// Function to check if the stack is empty
int isempty() {
    return top == NULL;
}

// Function to push an element onto the stack
void push() {
    int value;
    struct node *newNode = (struct node *)malloc(sizeof(struct node));
    if (newNode == NULL) {
        perror("Memory allocation failed");
        return;
    }

    printf("Enter a number to push: ");
    if (scanf("%d", &value) != 1) {
        printf("Invalid input. Please enter a number.\n");
        while (getchar() != '\n');  // Clear input buffer
        free(newNode);
        return;
    }

    newNode->data = value;
    newNode->next = top;
    top = newNode;  // Update top pointer

    printf("%d pushed onto the stack.\n", value);
}

// Function to pop an element from the stack
void pop() {
    if (isempty()) {
        printf("Stack is empty. Pop operation cannot be done.\n");
    } else {
        struct node *temp = top;
        int popped_value = temp->data;
        top = top->next;
        free(temp);  // Free the removed node
        printf("%d popped from the stack.\n", popped_value);
    }
}

// Function to display the stack elements
void peep() {
    if (isempty()) {
        printf("Stack is empty. Peep operation cannot be done.\n");
    } else {
        struct node *temp = top;
        printf("Stack elements (top to bottom):\n");
        while (temp != NULL) {
            printf("%d\n", temp->data);
            temp = temp->next;
        }
    }
}
```



```
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> cd 'c:\Users\krish\Downloads\mycode\output\Untitled-1\output'
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output> & .\'stack_linkedlist.exe'

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 1
Enter a number to push: 10
10 pushed onto the stack.

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 1
Enter a number to push: 20
20 pushed onto the stack.

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 1
Enter a number to push: 30
30 pushed onto the stack.

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 3
Stack elements (top to bottom):
30
20
10

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 2
30 popped from the stack.

--- Stack Menu ---
1: Push
2: Pop
3: Peep (Display Stack)
4: Exit
Choose the operation: 4
Exiting program.
PS C:\Users\krish\Downloads\mycode\output\Untitled-1\output>
```