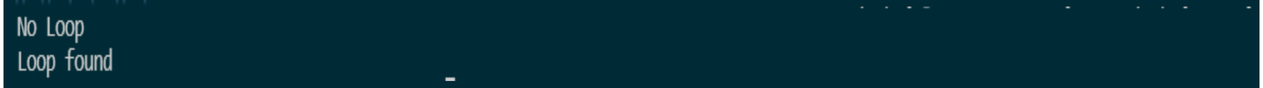
Q1.A linked list is said to contain a cycle if any node is visited more than once while traversing the list. WAP to detect a cycle in a linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node
{
  int data;
  struct Node *next;
};
void push(struct Node **head_ref, int new_data)
{
  struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
  new_node->data = new_data;
  new_node->next = (*head_ref);
  (*head_ref) = new_node;
}
int detectLoop(struct Node *list)
{
  struct Node *slow_p = list, *fast_p = list;
  while (slow_p && fast_p && fast_p->next)
  {
    slow_p = slow_p->next;
    fast_p = fast_p->next->next;
    if (slow_p == fast_p)
    {
      return 1;
    }
  }
  return 0;
}
int main()
{
  struct Node *head = NULL;
  push(&head, 20);
  push(&head, 4);
  push(&head, 15);
  push(&head, 10);

  head->next->next->next->next = NULL;
  if (detectLoop(head))
    printf("Loop found");
  else
    printf("No Loop");
  printf("\n");

  head->next->next->next->next = head;
  if (detectLoop(head))
    printf("Loop found");
  else
    printf("No Loop");
  return 0;
}
```

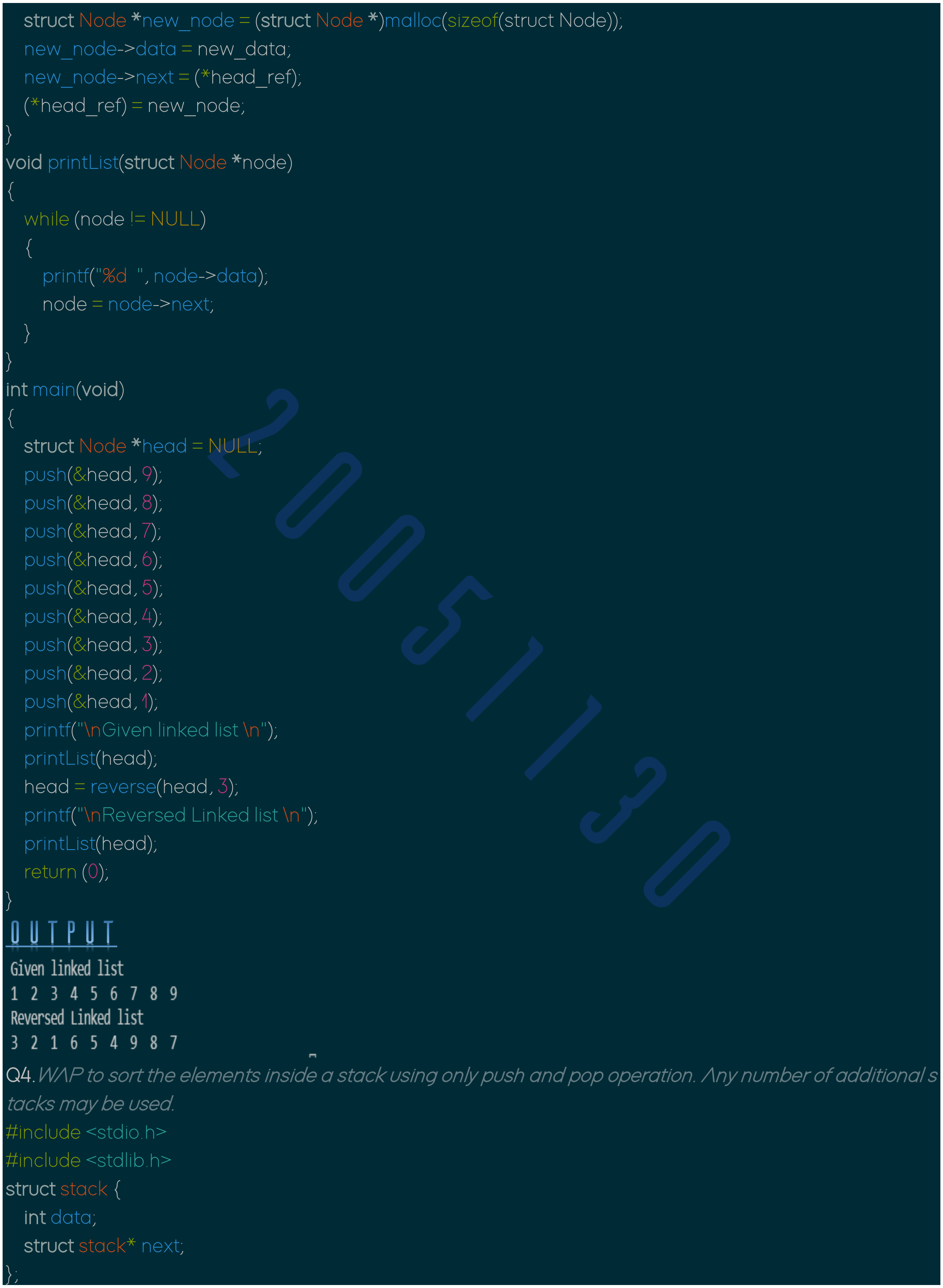
OUTPUT



```
No Loop
Loop found
```

Q2. Given a linked list, write a function to reverse every k nodes. (where k is an input to the function). If a linked list is given as 12->23->45->89->15->67->28->98->NULL and k = 3 then output will be 45->23->12->67->15->89->98->28->NULL.

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

struct Node
{
  int data;
  struct Node *next;
};
struct Node *reverse(struct Node *head, int k)
{
  if (!head)
    return NULL;
  struct Node *current = head;
  struct Node *next = NULL;
  struct Node *prev = NULL;
  int count = 0;
  while (current != NULL && count < k)
  {
    next = current->next;
    current->next = prev;
    prev = current;
    current = next;
    count++;
  }
  if (next != NULL)
    head->next = reverse(next, k);
  return prev;
}
void push(struct Node **head_ref, int new_data)
{
```



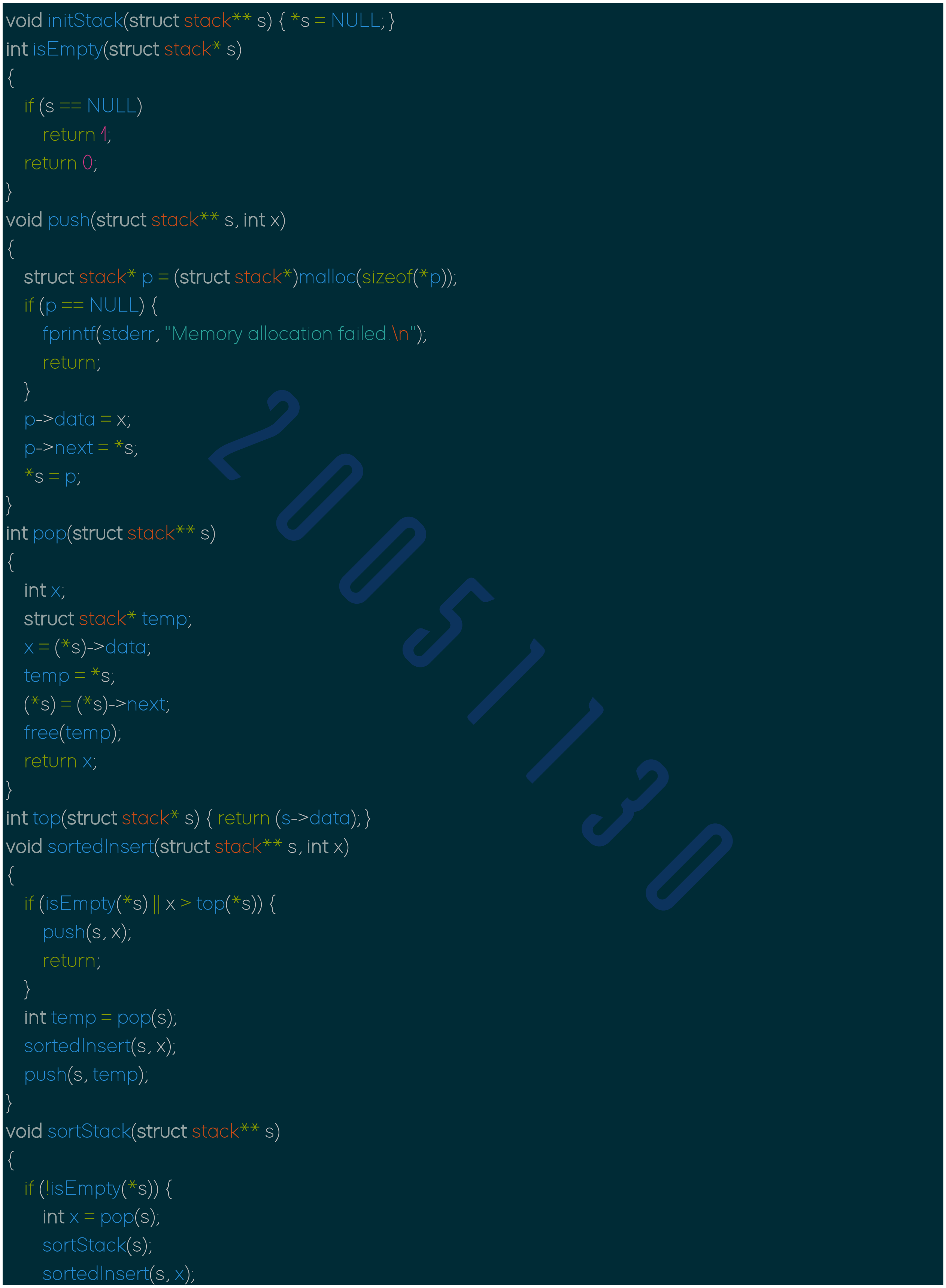
```
    struct Node *new_node = (struct Node *)malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}
void printList(struct Node *node)
{
    while (node != NULL)
    {
        printf("%d  ", node->data);
        node = node->next;
    }
}
int main(void)
{
    struct Node *head = NULL;
    push(&head, 9);
    push(&head, 8);
    push(&head, 7);
    push(&head, 6);
    push(&head, 5);
    push(&head, 4);
    push(&head, 3);
    push(&head, 2);
    push(&head, 1);
    printf("\nGiven linked list \n");
    printList(head);
    head = reverse(head, 3);
    printf("\nReversed Linked list \n");
    printList(head);
    return (0);
}
```

OUTPUT

```
Given linked list
1  2  3  4  5  6  7  8  9
Reversed Linked list
3  2  1  6  5  4  9  8  7
```

Q4. *WAP to sort the elements inside a stack using only push and pop operation. Any number of additional stacks may be used.*

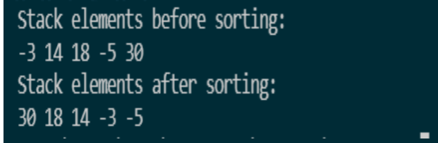
```
#include <stdio.h>
#include <stdlib.h>
struct stack {
    int data;
    struct stack* next;
};
```



```
void initStack(struct stack** s) { *s = NULL; }
int isEmpty(struct stack* s)
{
    if (s == NULL)
        return 1;
    return 0;
}
void push(struct stack** s, int x)
{
    struct stack* p = (struct stack*)malloc(sizeof(*p));
    if (p == NULL) {
        fprintf(stderr, "Memory allocation failed.\n");
        return;
    }
    p->data = x;
    p->next = *s;
    *s = p;
}
int pop(struct stack** s)
{
    int x;
    struct stack* temp;
    x = (*s)->data;
    temp = *s;
    (*s) = (*s)->next;
    free(temp);
    return x;
}
int top(struct stack* s) { return (s->data); }
void sortedInsert(struct stack** s, int x)
{
    if (isEmpty(*s) || x > top(*s)) {
        push(s, x);
        return;
    }
    int temp = pop(s);
    sortedInsert(s, x);
    push(s, temp);
}
void sortStack(struct stack** s)
{
    if (!isEmpty(*s)) {
        int x = pop(s);
        sortStack(s);
        sortedInsert(s, x);
```

```
  }
}
void printStack(struct stack* s)
{
  while (s) {
    printf("%d ", s->data);
    s = s->next;
  }
  printf("\n");
}
int main(void)
{
  struct stack* top;
  initStack(&top);
  push(&top, 30);
  push(&top, -5);
  push(&top, 18);
  push(&top, 14);
  push(&top, -3);
  printf("Stack elements before sorting:\n");
  printStack(top);
  sortStack(&top);
  printf("Stack elements after sorting:\n");
  printStack(top);
  return 0;
}
```

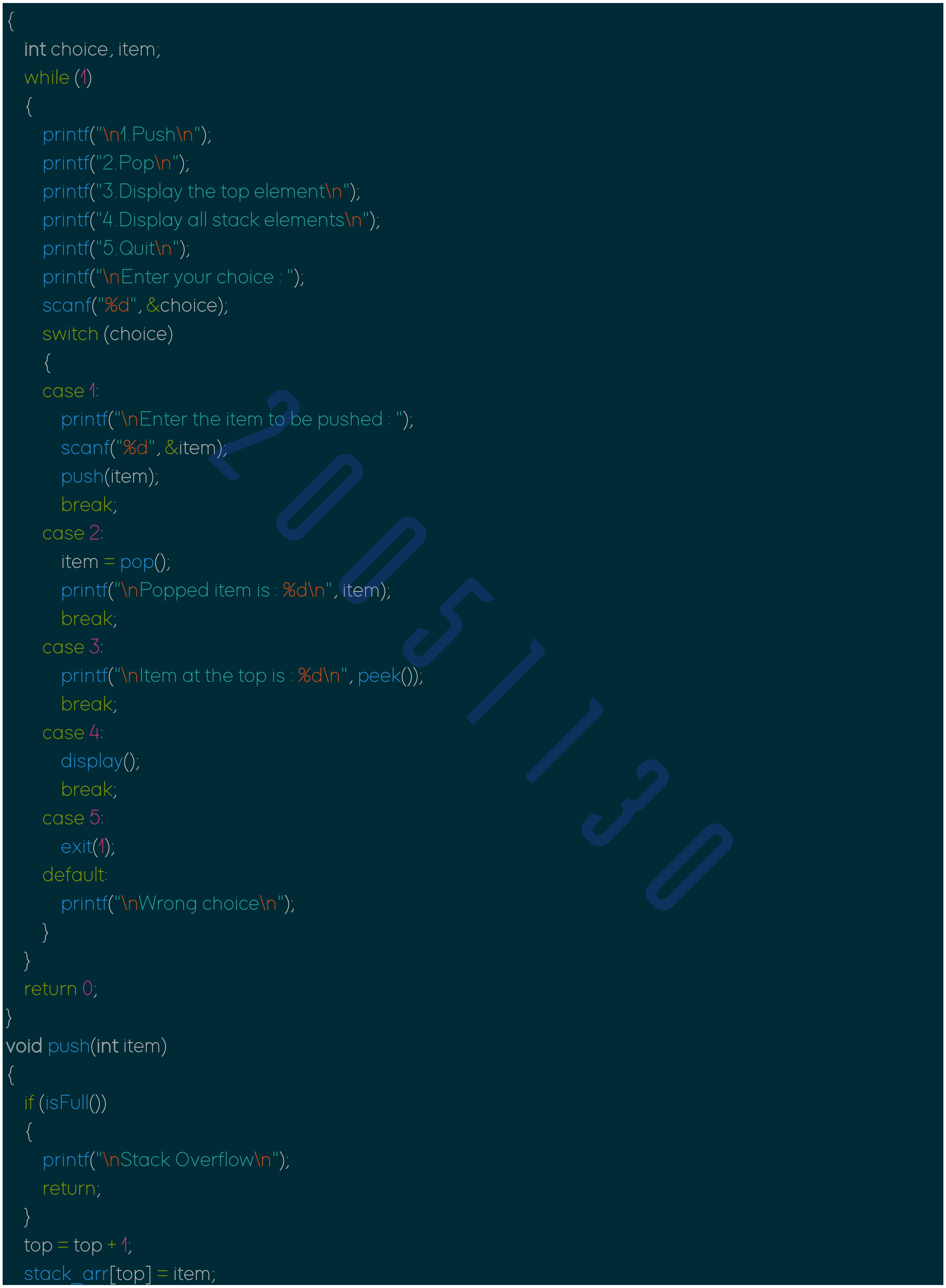
OUTPUT



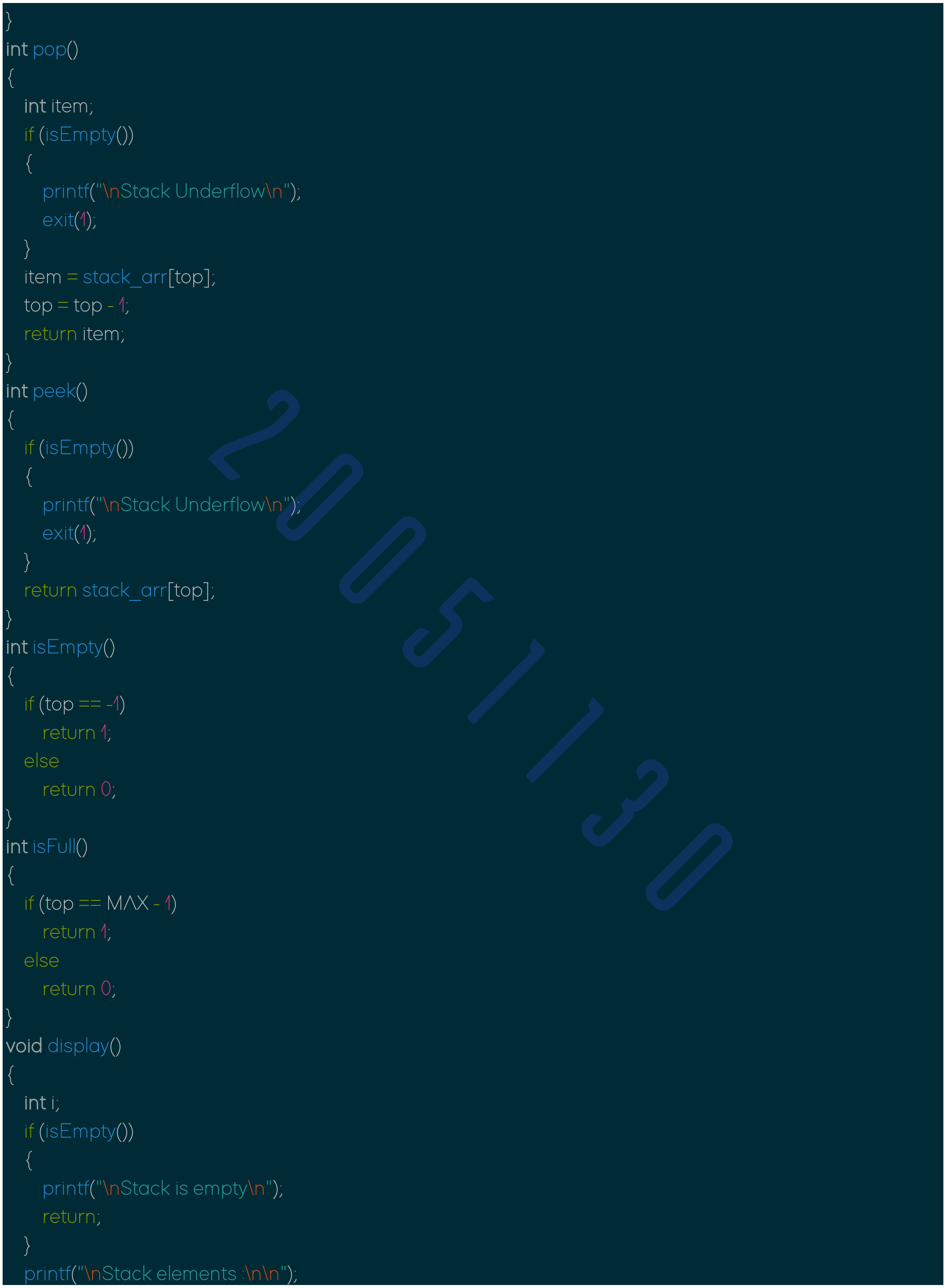
```
Stack elements before sorting:
-3 14 18 -5 30
Stack elements after sorting:
30 18 14 -3 -5
```

Q4. A stack data structure is given with push and pop operations. WAP to implement a queue using instances of stack data structure and operations on them.

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 10
int stack_arr[MAX];
int top = -1;
void push(int item);
int pop();
int peek();
int isEmpty();
int isFull();
void display();
int main()
```



```
{
    int choice, item;
    while (1)
    {
        printf("\n1.Push\n");
        printf("2.Pop\n");
        printf("3.Display the top element\n");
        printf("4.Display all stack elements\n");
        printf("5.Quit\n");
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("\nEnter the item to be pushed : ");
            scanf("%d", &item);
            push(item);
            break;
        case 2:
            item = pop();
            printf("\nPopped item is : %d\n", item);
            break;
        case 3:
            printf("\nItem at the top is : %d\n", peek());
            break;
        case 4:
            display();
            break;
        case 5:
            exit(1);
        default:
            printf("\nWrong choice\n");
        }
    }
    return 0;
}
void push(int item)
{
    if (isFull())
    {
        printf("\nStack Overflow\n");
        return;
    }
    top = top + 1;
    stack_arr[top] = item;
```

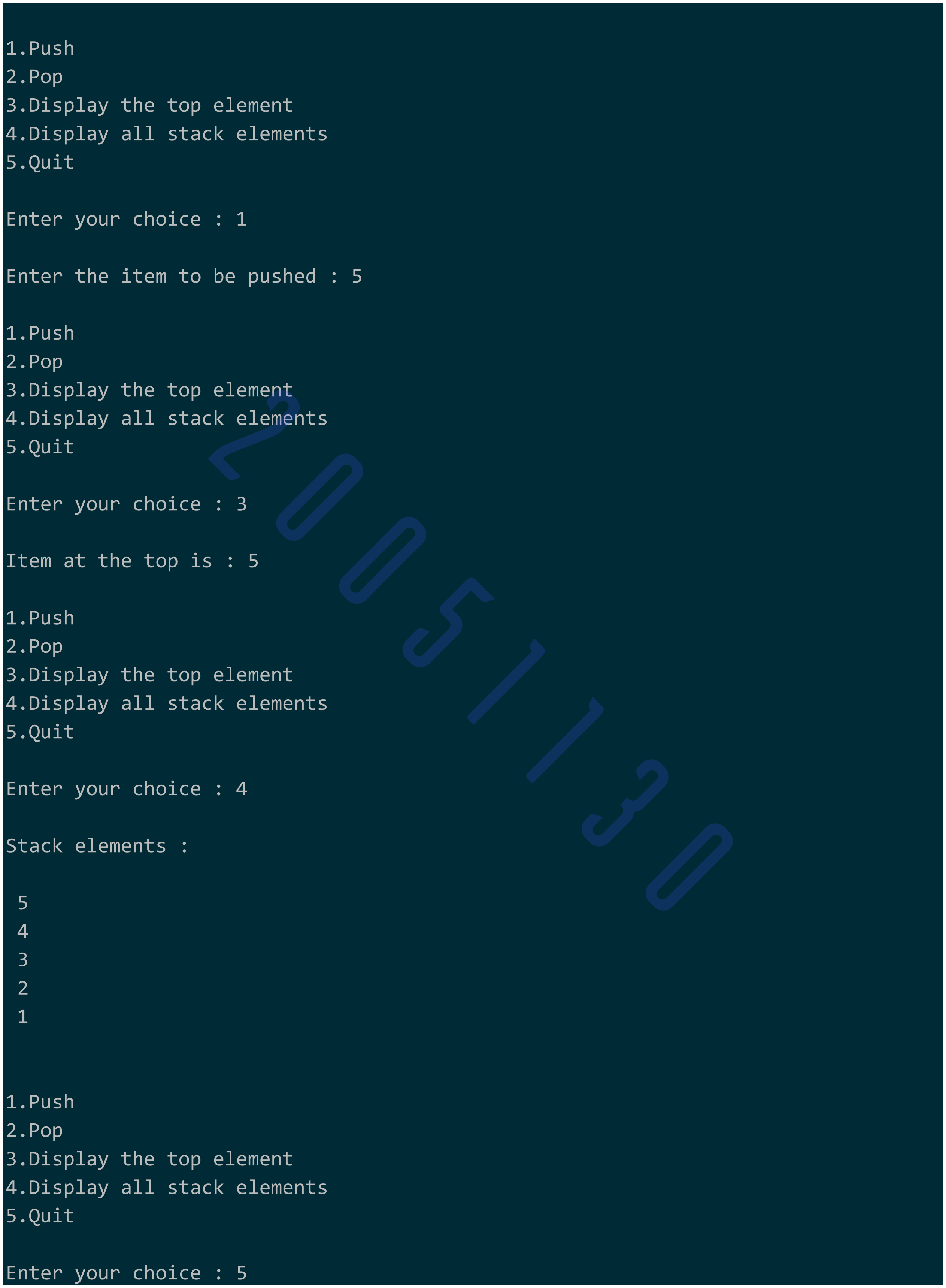


```
}
int pop()
{
    int item;
    if (isEmpty())
    {
        printf("\nStack Underflow\n");
        exit(1);
    }
    item = stack_arr[top];
    top = top - 1;
    return item;
}
int peek()
{
    if (isEmpty())
    {
        printf("\nStack Underflow\n");
        exit(1);
    }
    return stack_arr[top];
}
int isEmpty()
{
    if (top == -1)
        return 1;
    else
        return 0;
}
int isFull()
{
    if (top == MAX - 1)
        return 1;
    else
        return 0;
}
void display()
{
    int i;
    if (isEmpty())
    {
        printf("\nStack is empty\n");
        return;
    }
    printf("\nStack elements :\n\n");
```

```
  for (i = top; i >= 0; i--)
    printf(" %d\n", stack_arr[i]);
  printf("\n");
}
```

OUTPUT

```
1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 1

Enter the item to be pushed : 1

1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 1

Enter the item to be pushed : 2

1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 1

Enter the item to be pushed : 3

1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 1

Enter the item to be pushed : 4
```



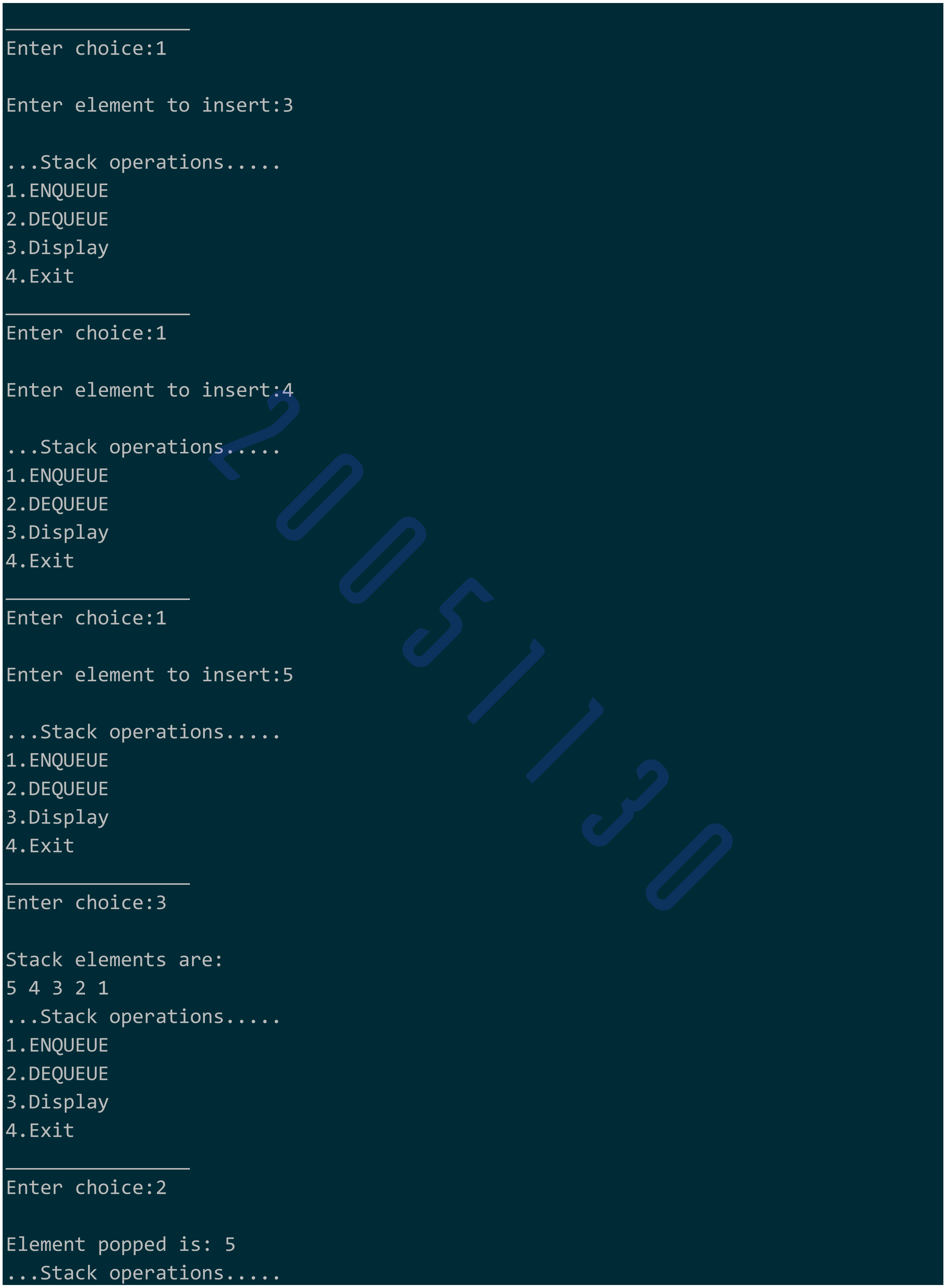
```
1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 1

Enter the item to be pushed : 5

1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 3

Item at the top is : 5

1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 4

Stack elements :

 5
 4
 3
 2
 1


1.Push
2.Pop
3.Display the top element
4.Display all stack elements
5.Quit

Enter your choice : 5
```

Q5. A queue data structure is given with enqueue and dequeue operations.WAP to implement a stack using instances of queue data structure and operations on them.

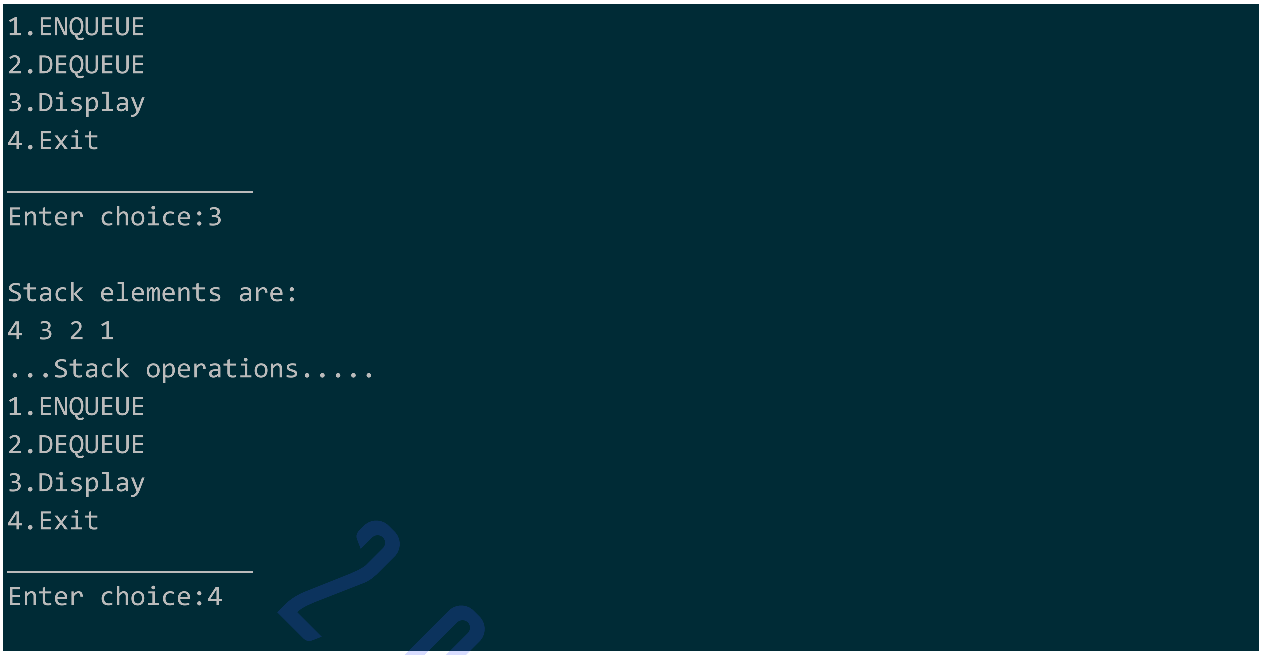
```
#include <stdio.h>
#include <stdlib.h>
int s[5], top = -1;
void push()
{
  if (top == 4)
    printf("\nStack overflow!!!!");
  else
  {
    printf("\nEnter element to insert:");
    scanf("%d", &s[++top]);
  }
}
void pop()
{
  if (top == -1)
    printf("\nStack underflow!!!");
  else
    printf("\nElement popped is: %d", s[top--]);
}
void disp()
{
  int t = top;
  if (t == -1)
    printf("\nStack empty!!");
  else
    printf("\nStack elements are:\n");
  while (t >= 0)
    printf("%d ", s[t--]);
}
int main()
{
  int ch;
  do
  {
    printf("\n...Stack operations.....\n");
    printf("1.ENQUEUE\n");
    printf("2.DEQUEUE\n");
    printf("3.Display\n");
    printf("4.Exit\n__________________\n");
    printf("Enter choice:");
    scanf("%d", &ch);
    switch (ch)
    {
    case 1:
      push();
      break;
    case 2:
      pop();
      break;
    case 3:
      disp();
      break;
    case 4:
      exit(0);
    default:
      printf("\nInvalid choice");
    }
  } while (1);
  return 0;
}
```

OUTPUT

```
...Stack operations.....
1.ENQUEUE
2.DEQUEUE
3.Display
4.Exit
________________
Enter choice:1

Enter element to insert:1

...Stack operations.....
1.ENQUEUE
2.DEQUEUE
3.Display
4.Exit
________________
Enter choice:1

Enter element to insert:2

...Stack operations.....
1.ENQUEUE
2.DEQUEUE
3.Display
4.Exit
```



```
________________
Enter choice:1

Enter element to insert:3

...Stack operations.....
1.ENQUEUE
2.DEQUEUE
3.Display
4.Exit
________________
Enter choice:1

Enter element to insert:4

...Stack operations.....
1.ENQUEUE
2.DEQUEUE
3.Display
4.Exit
________________
Enter choice:1

Enter element to insert:5

...Stack operations.....
1.ENQUEUE
2.DEQUEUE
3.Display
4.Exit
________________
Enter choice:3

Stack elements are:
5 4 3 2 1
...Stack operations.....
1.ENQUEUE
2.DEQUEUE
3.Display
4.Exit
________________
Enter choice:2

Element popped is: 5
...Stack operations.....
```



```
1.ENQUEUE
2.DEQUEUE
3.Display
4.Exit
________________
Enter choice:3

Stack elements are:
4 3 2 1
...Stack operations.....
1.ENQUEUE
2.DEQUEUE
3.Display
4.Exit
________________
Enter choice:4
```