DSA ASSIGNMENT-7

TANISHA KARMAKAR

21051950

CSE 37

Q1. WAP to create a Binary tree and perform the In-order traversal (non-recursive).

```c
#include<stdio.h>

struct node
{
    int data;
    struct node *lc, *rc;
};
void create(struct node *ptr)
{
    int choice;
    printf("Enter data : ");
    scanf("%d",&ptr->data);
    ptr->lc = NULL;
    ptr->rc = NULL;
```

```c
    printf("left child of %d?(0/1) : ",ptr->data);
    scanf("%d",&choice);
    if(choice == 1)
    {
        struct node *new = (struct node*)malloc(sizeof(struct node));
        ptr->lc = new;
        create(new);
    }
    printf("right child of %d?(0/1) : ",ptr->data);
    scanf("%d",&choice);
    if(choice == 1)
    {
        struct node *new = (struct node*)malloc(sizeof(struct node));
        ptr->rc = new;
        create(new);
    }

}
struct node *stack[100];
int top = -1;
void push(struct node *data)
{
    top++;
    stack[top] = data;
}
void pop()
{
    if(top == -1)
        printf("\nstack underflow!");
    else
    {
```

```
        printf("%d ",stack[top]->data);
        top--;
    }
}
```

```
void inorder(struct node *root)
{
```

```
    struct node *ptr;
    ptr = root;

    while(ptr != NULL || top != -1)
    {
        while(ptr != NULL)
        {
            push(ptr);
            ptr = ptr->lc;
        }
        ptr = stack[top];
        pop();
        ptr = ptr->rc;
    }
}
```

```
int main()
{
    struct node *root = (struct node *)malloc(sizeof(struct node));
    create(root);
    printf("non recursive inorder traversal : ");
    inorder(root);

}
```

OUTPUT:

```
Enter data : 7
left child of 7?(0/1) : 0
right child of 7?(0/1) : 0
right child of 10?(0/1) : 1
Enter data : 8
left child of 8?(0/1) : 1
Enter data : 9
left child of 9?(0/1) : 0
right child of 9?(0/1) : 0
right child of 8?(0/1) : 0
non recursive inorder traversal : 6 5 7 10 9 8
```

Q.2 WAP to create a Binary tree and perform the Pre-order traversal (non-recursive).

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

struct node
{
    int data;
    struct node *lc, *rc;
};
```

```
void create(struct node *ptr)
{
    int choice;
    printf("Enter data : ");
    scanf("%d",&ptr->data);
    ptr->lc = NULL;
    ptr->rc = NULL;
```

```
    printf("left child of %d?(0/1) : ",ptr->data);
    scanf("%d",&choice);
    if(choice == 1)
    {
        struct node *new = (struct node*)malloc(sizeof(struct node));
        ptr->lc = new;
        create(new);
    }
    printf("right child of %d?(0/1) : ",ptr->data);
    scanf("%d",&choice);
    if(choice == 1)
    {
        struct node *new = (struct node*)malloc(sizeof(struct node));
        ptr->rc = new;
        create(new);
    }

}
```

```
struct node *stack[100];
int top = -1;
void push(struct node *data)
{
    top++;
    stack[top] = data;
}
void pop()
{
    if(top == -1)
        printf("\nstack underflow!");
    else
    {
        printf("%d ",stack[top]->data);
        top--;
    }
}
```

```
void preorder(struct node *root)
{
```

```
    struct node *ptr;
    ptr = root;
    push(root);

    while(ptr != NULL || top != -1)
    {
        ptr = stack[top];
        pop();
        if(ptr->rc!=NULL)
        {
        push(ptr->rc);
        }
```

```
        if(ptr->lc!=NULL)
        {
        push(ptr->lc);
        }

    }
}
```

```
int main()
{
    struct node *root = (struct node *)malloc(sizeof(struct node));
    create(root);
    printf("non recursive preorder traversal : ");
    preorder(root);
```

```
}
```

OUTPUT:

```
Enter data : 10
left child of 10?(0/1) : 1
Enter data : 5
left child of 5?(0/1) : 1
Enter data : 6
left child of 6?(0/1) : 0
right child of 6?(0/1) : 0
right child of 5?(0/1) : 1
Enter data : 7
left child of 7?(0/1) : 0
right child of 7?(0/1) : 0
right child of 10?(0/1) : 1
Enter data : 8
left child of 8?(0/1) : 1
Enter data : 9
left child of 9?(0/1) : 0
right child of 9?(0/1) : 0
right child of 8?(0/1) : 0
non recursive preorder traversal : 10 5 6 7 8 9
```

Q.3 WAP to create a Binary tree and perform the Post-order traversal (non-recursive).

```c
/*wap to create a binary tree and perform the post-order traversal(non recursive)*/
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *lc, *rc;
};
void create(struct node *ptr)
{
    int choice;
    printf("data : ");
    scanf("%d",&ptr->data);
    ptr->lc = NULL;
    ptr->rc = NULL;

    printf("left child of %d? : ",ptr->data);
    scanf("%d",&choice);
    if(choice == 1)
    {
        struct node *new = (struct node*)malloc(sizeof(struct node));
        ptr->lc = new;
        create(new);
    }
    printf("right child of %d? : ",ptr->data);
    scanf("%d",&choice);
    if(choice == 1)
    {
        struct node *new = (struct node*)malloc(sizeof(struct node));
        ptr->rc = new;
        create(new);
    }

}
```

```c
struct node *stack[100];
int top = -1;
void push(struct node *data)
{
    top++;
    stack[top] = data;
}
void pop()
{
    if(top == -1)
        printf("\nstack underflow!");
    else
    {
        top--;
    }
}
void postorder(struct node *root)
{
    struct node *ptr = root;

    while(ptr != NULL)
    {
```

```
        push(ptr);
        if(ptr->rc != NULL)
            push(ptr->rc);
        ptr = ptr->lc;
    }
    ptr = stack[top];
    printf("%d ",stack[top]->data);
    pop();
    if(ptr->rc != NULL && ptr->rc == stack[top])
    {
        pop();
        push(ptr);
        ptr = ptr->rc;
    }
    else
        ptr = NULL;
```

```
    while(top != -1)
    {
        while(ptr != NULL)
        {
```

```
            push(ptr);
            if(ptr->rc != NULL)
                push(ptr->rc);
            ptr = ptr->lc;
        }
        ptr = stack[top];
        printf("%d ",stack[top]->data);
        pop();
        if(ptr->rc != NULL && ptr->rc == stack[top])
        {
            pop();
            push(ptr);
            ptr = ptr->rc;
        }
        else
            ptr = NULL;
    }
}
```

```
int main()
{
    struct node *root = (struct node*)malloc(sizeof(struct node));
    create(root);
    printf("non recursive\n");
    postorder(root);
}
```

OUTPUT:

```
data : 10
left child of 10? : 1
data : 5
left child of 5? : 1
data : 6
left child of 6? : 0
right child of 6? : 0
right child of 5? : 1
data : 7
left child of 7? : 0
right child of 7? : 0
right child of 10? : 1
data : 8
left child of 8? : 1
data : 9
left child of 9? : 0
right child of 9? : 0
right child of 8? : 0
non recursive
6 7 5 8 10
```