Complete the given function prototypes accordingly.

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

struct node
{
   int data;
   struct node *next;
};
struct node *start = NULL;

void display();      //display list
void insert_beg(); //insert at beginning
void insert_end(); //insert at end
void delete_beg(); //delete from beginning
void delete_end(); //delete from end

int main()
{
   int option;
   do
   {
     printf(“\n 1: Display the list”);
     printf(“\n 2: Add node at the beginning”);
     printf(“\n 3: Add node at the end”);
     printf(“\n 4: Delete node from beginning”);
     printf(“\n 5: Delete node from end”);
     printf(“\n 6: EXIT”);
     printf(“\n\n Enter your option:”);
     scanf_s(“%d”, &option);
     switch (option)
     {
          case 1: display(); break;
          case 2: insert_beg(); break;
          case 3: insert_end(); break;
          case 4: delete_beg(); break;
          case 5: delete_end(); break;
     }
   } while (option != 6);
   return 0;
}

void display()
{











}
```

```
void insert_beg()
{










}

void insert_end()
{










}


void delete_beg()
{









}

void delete_beg()
{











}
```

In order above pragram to be called **Stack** implementation which functions should be used?

In order above pragram to be called **Queue** implementation which functions should be used?

Complete the below AVL tree program accordingly.

```
// C program to Insert a node in Avl tree
#include <stdio.h>
#include <stdlib.h>

// An AVL tree node
struct Node
{
        int key;
        struct Node *left;
        struct Node *right;
        int height;
};

// A function to get maximum of two integers
int max(int a, int b) {

}

// A function to get the height of the tree
int height(struct Node *p) {

}

/* Helper function that allocates a new node with
the given key and NULL left and right pointers. */
struct Node *newNode(int key) {

}
```

```
//A utility function to right rotate subtree rooted
with y
struct Node *rightRotate(struct Node *y)
{

        //Perform rotation

        //Update heights

        //Return new root

}

//A utility function to left rotate subtree rooted
with x
struct Node *leftRotate(struct Node *x)
{

        //Perform rotation

        //Update heights

        //Return new root

}
```

Use **Hash Tables** to store values in an Open Addressing mode in order to handle collisions.

Complete the function prototypes of given implementation accordingly.

Insert function keeps probing until an empty slot is found.

Program prints warning when hash table is full or element is already exist.

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

//Hash table size
#define max 10
int ht[max];

void insert()
void display()

void main()
{
        for (int i = 0; i < max; i++)
                ht[i] = -1;
        printf(“1. Insert\n2, Display\n”
               “3. Exit”);

        int n;
        while (1)
        {
                printf(“\nEnter your choice: ”);
                sscanf_s(“%d”, &n);
                switch (n)
                {
                case 1:
                        insert();
                        break;
                case 2:
                        display();
                        break;
                case 3:
                        exit(0);
                default:
                        printf(“Wrong choice\n”);
                }
        }
}

void display(){




}
```

```
void insert(){




}
```

Complete the function prototypes of given binary search tree program accordingly.

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

typedef struct node
{
  int key;
  struct node *left, *right;
} node;

node *insert(node *root, int key)
node *delete (node *root, int key)
void inorderPrint(node *root)

int main()
{
  int i, data;
  node *root = NULL;
  while (1)
  {
        printf("1-Add\n 2-Delete\n"
                "3-Display\n 4-Exit\n");
        scanf_s("%d", &i);
        switch (i)
        {
        case 1:
                printf("Enter Value to add\n");
                scanf_s("%d", &data);
                root = insert(root, data);
                break;
        case 2:
                printf("Enter Value to delete\n");
                scanf_s("%d", &data);
                root = delete (root, data);
                break;
        case 3:
                inorderPrint(root);
                break;
        case 4:
                exit(1);
                break;
        default:
                printf("Wrong choice\n");
        }
  }
}

void inorderPrint(node* root){



}
```

```
node *insert(node *root, int key)
{



}


node *delete(node *root, int key)
{



}
```

1) Given a binary tree, print its nodes according to the preorder traversal.

```
void printInorder(node* root)
{



}
```

2) Write a minimal implementation of a ***stack*** with ***arrays*** without main function.

3) Write a minnimal implementation of a Queue without main function.