Name: Yashwant Chandrakant Bhosale
MIS: 612303039
SY Comp Div 1

Question:
Implement ADT Binary Search Tree(BST) using a linked list of nodes of structure having Month, left pointer, right pointer, and parent pointers pointing to left sub-trees, right sub-tree, and parent of the node.

Perform a series of insertions on keys - December, January, April, March, July, August, October, February, November, May, June. Write following functions:

initBST() // to initialize the tree.

insertNode() // non-recursive function to add a new node to the BST.

removeNode() // to remove a node from a tree.

traverse() // write any of the non-recursive traversal methods.

destroyTree() // to delete all nodes of a tree.

Write a menu driven program to invoke all above functions.

Solution:
I've represented the 12 months as integers in a mapping from January (0) to December (11). This enabled me to implement an integer BST with months in logical order rather than alphabetical, streamlining operations.

If months were represented as strings (e.g., "January," "February"), the binary search tree (BST) would organize them alphabetically rather than by their calendar order. This is due to the ASCII-based sorting of strings, so "April" would come before "February," disrupting the natural progression of months. As a result, performing chronological operations, like finding the next or previous month, would be difficult and inefficient. By mapping months to integers (from 0 for January to 11 for December), the BST can maintain a logical calendar order, making it much simpler to execute chronological operations and comparisons.

bst.h: File containing all function prototypes and struct declarations

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node {
    struct node *lchild, *rchild, *parent;
    int month;
} node;

typedef struct {
    node *root;
    int count;
} bst;

enum Months {
    January,
    February,
    March,
    April,
    May,
    June,
    July,
    August,
    September,
    October,
    November,
    December
};
void init_bst(bst *tree);
void insertNode(bst *tree, int month);
void removeNode(bst *tree, int month);
void inorder(bst *tree);
void destroyTree(bst *tree);
```

bst.c: File containing the function definitions for the functions related to Binary Search Tree

```
#include "bst.h"
#include <stdio.h>
#include <stdlib.h>
#include "stack.h"
void init_bst(bst *t) {
     t->root = NULL;
     t->count = 0;
     return;
}
```

```c
void insertNode(bst *t, int month) {
    node *nn = (node *)malloc(sizeof(node));
    if (!nn) {
        return;  // Memory allocation failed
    }

    nn->month = month;
    nn->lchild = nn->rchild = NULL;

    // Case 1: Empty tree
    if (t->root == NULL) {
        t->root = nn;
        nn->parent = NULL;  // Root has no parent
        t->count++;
        return;
    }

    // Case 2: Non-empty tree
    node *p = t->root, *q = NULL;
    while (p) {
        q = p;
        if (month < p->month) {
            p = p->lchild;
        } else if (month > p->month) {
            p = p->rchild;
        } else {
            // Duplicate value
            free(nn);
            return;
        }
    }

    // Set the new node as a child of q
    if (month > q->month) {
        q->rchild = nn;
    } else {
        q->lchild = nn;
    }

    // Set the parent of the new node
    nn->parent = q;
    t->count++;
    return;
}
```

```
void print_month(int month) {
    switch (month) {
        case January:
            printf("January, ");
            break;
        case February:
            printf("February, ");
            break;
        case March:
            printf("March, ");
            break;
          case April:
              printf("April, ");
              break;
          case May:
              printf("May, ");
              break;
        case June:
            printf("June, ");
            break;
        case July:
            printf("July, ");
            break;
        case August:
            printf("August, ");
            break;
        case September:
            printf("September, ");
            break;
        case October:
            printf("October, ");
            break;
        case November:
            printf("November, ");
            break;
        case December:
            printf("December, ");
            break;
        default:
            printf("%d is Invalid month, ", month);
    }
    return;
}
```

```
void inorder(bst *t) {
     stack s;
     init(&s, t->count);
     node *p = t->root;
     while (p != NULL || !is_empty(&s)) {
          // Traverse to the leftmost node

          while (p != NULL) {
               push(&s, p);
               p = p->lchild;
          }

          // Pop from the stack and print the month
          node *popped = pop(&s);
          if(popped){
          print_month(popped->month);
               // Move to the right child
               p = popped->rchild;
          }

     }

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

```c
void removeNode(bst *t, int month) {
    /*
        3 cases:
        1. node with no children (leaf node)
        2. node with single child
        3. node with two children

        for node with no children:
        we can simply detach the link and free the node

        for node with single child:
        we can replace the node with its child and free the node

        for node with two children:
        1. Search the node
        2. Find inorder successor
        3. Replace the node with inorder successor
        4. Delete the inorder successor

        Inorder successor is the leftmost node of the right
            subtree or minimum of the right subtree
        Inorder predecessor is the rightmost node of the left
            subtree or maximum of the left subtree
    */
    node *p = t->root, *q = NULL;

    // search the node
    while (p) {
        if (p->month == month) {
            break;
        }
        q = p;
        if (p->month < month) {
            p = p->rchild;
        } else {
            p = p->lchild;
        }
    }

    if (p == NULL) {
        return;  // Node not found
    }
```

```c
// case 1: node with no children
if (p->lchild == NULL && p->rchild == NULL) {
     if (q == NULL) {
          t->root = NULL;
     } else if (q->lchild == p) {
          q->lchild = NULL;
     } else {
          q->rchild = NULL;
     }
     free(p);
     t->count--;
     return;
}

// case 2: node with single child
if (p->lchild == NULL || p->rchild == NULL) {
     node *child = p->lchild ? p->lchild : p->rchild;
     if (q == NULL) {
          t->root = child;
     } else if (q->lchild == p) {
          q->lchild = child;
     } else {
          q->rchild = child;
     }
     free(p);
     t->count--;
     return;
}

// case 3: node with two children
node *successor = p->rchild, *successor_parent = p;
// Here, we may drop the parent pointer, as we have the
   parent pointer in the node structure

while (successor->lchild) {
     successor_parent = successor;
     successor = successor->lchild;
}

// Replace the node with the successor
p->month = successor->month;
```

```
    /*
        important case: if the successor's lchild is null, that
        is pointers did not move in the above while loop
        e.g. suppose 30 is the node to be deleted and 40 is the
        successor
            50
            /\
           30 60
          /\
         20 40
             \
             45
             /\
            43 44

        Here, we need to replace 30 with 40 and 40's right child
        should be 45
    */

    // pointer did not move in the above while loop
    if (successor_parent == p) {
        successor_parent->rchild = successor->rchild;
    }

    // pointer moved in the above while loop
    else {
        successor_parent->lchild = successor->rchild;
    }

    free(successor);
    t->count--;

    return;
}
```

```c
void destroyTree(bst *t) {
    node *p = t->root;

    stack s;
    init(&s, t->count);


    // similar to inorder traversal: visit each node in inorder fashion and free the node instead of printing
    while(p != NULL || !is_empty(&s)) {
        while(p != NULL) {
            push(&s, p);
            p = p->lchild;
        }

        node *popped = pop(&s);
        node *right = popped->rchild;
        free(popped);
        p = right;
    }

    t->root = NULL;
    t->count = 0;
    return;
}
```

stack.h: Contains the struct declarations and function prototypes of stack for the node pointers

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

typedef struct {
    node **arr;
    int size;
    int top;
} stack;

void init(stack *s, int size);
void push(stack *s, node *n);
node *pop(stack *s);
node *peek(stack *s);
short is_empty(stack *s);
```

stack.c: contains all the function definitions for the stack

```c
#include "bst.h"
#include "stack.h"
/* Function to initialize stack */
void init(stack *s, int size) {
     s->arr = (node **)malloc(size * sizeof(node *));
     s->top = -1;
     s->size = size;
     return;
}

/* Function to push an element in the stack */
void push(stack *s, node *n) {
     if (s->top >= s->size - 1) {
          return;  // stack full
     }

     s->top++;
     s->arr[s->top] = n;
     return;
}

/* Function to pop an element from the stack */
node *pop(stack *s) {
     if (is_empty(s)) {
          return NULL;
     }

     node *n = s->arr[s->top];
     s->top--;
     return n;
}

short is_empty(stack *s) {
     return s->top == -1;
}

/* Function to peek into the stack */
node *peek(stack *s) {
     if (is_empty(s)) {
          return NULL;
     }

     return s->arr[s->top];
}
```

main.c: Contains menu and main flow of the program.

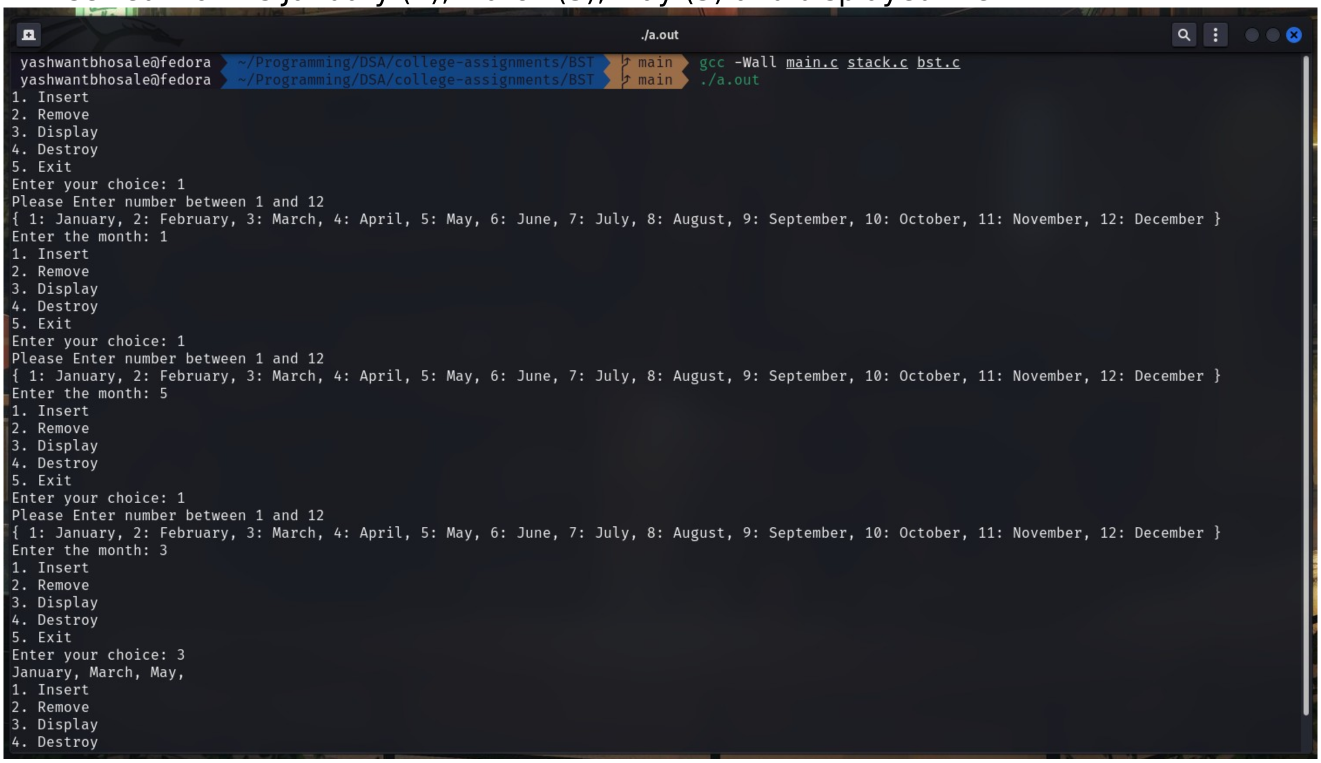
```
#include <stdio.h>
#include <stdlib.h>

#include "bst.h"

void displayMenu() {
     printf("1. Insert\n");
     printf("2. Remove\n");
     printf("3. Display\n");
     printf("4. Destroy\n");
     printf("5. Exit\n");
     return;
}

void evaluate_choice(int choice, bst *tree) {
     int month;
     switch (choice) {
          case 1:
               printf("Please Enter number between 1 and 12\n");
               printf("{ 1: January, 2: February, 3: March, 4:
April, 5: May, 6: June, 7: July, 8: August, 9: September, 10:
October, 11: November, 12: December }\n");

            printf("Enter the month: ");
               scanf("%d", &month);

               if (month < 1 || month > 12) {
                    printf("Invalid month\n");
                    break;
               }

               insertNode(tree, month - 1);
               break;
          case 2:
            printf("Please Enter number between 1 and 12\n");
               printf("{ 1: January, 2: February, 3: March, 4:
April, 5: May, 6: June, 7: July, 8: August, 9: September, 10:
October, 11: November, 12: December }\n");

            printf("Enter the month: ");
               scanf("%d", &month);

               if (month < 1 || month > 12) {
                    printf("Invalid month\n");
```

```
                    break;
                }
                removeNode(tree, month-1);
                break;

            case 3:
                inorder(tree);
                break;
            case 4:
                destroyTree(tree);
                break;
            case 5:
                exit(0);
            default:
                printf("Invalid choice\n");
        }
        return;
}

int main() {
        bst tree;
        init_bst(&tree);

        int choice;

        while (1) {
            displayMenu();
            printf("Enter your choice: ");
            scanf("%d", &choice);
            evaluate_choice(choice, &tree);
        }

        return 0;
}
```

OUTPUT

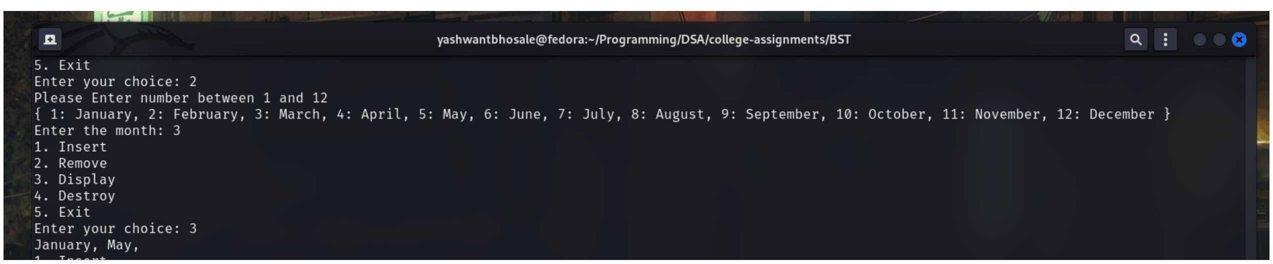
Note:
In my approach I Have represented **Months as Numbers from 0-11** (Using enum)
So when we insert months in the tree they are always inserted in order from jan to dec.
and when they get displayed they are displayed in **inorder** fashion so they are always sorted.

1. Inserted Months January (1), March (3), May (5) and displayed them



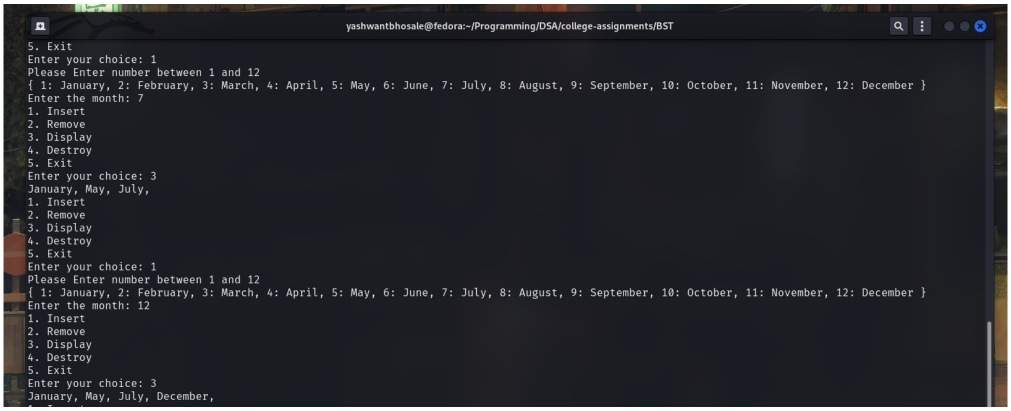
```
./a.out
yashwantbhosale@fedora ~/Programming/DSA/college-assignments/BST main gcc -Wall main.c stack.c bst.c
yashwantbhosale@fedora ~/Programming/DSA/college-assignments/BST main ./a.out
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 1
Please Enter number between 1 and 12
{ 1: January, 2: February, 3: March, 4: April, 5: May, 6: June, 7: July, 8: August, 9: September, 10: October, 11: November, 12: December }
Enter the month: 1
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 1
Please Enter number between 1 and 12
{ 1: January, 2: February, 3: March, 4: April, 5: May, 6: June, 7: July, 8: August, 9: September, 10: October, 11: November, 12: December }
Enter the month: 5
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 1
Please Enter number between 1 and 12
{ 1: January, 2: February, 3: March, 4: April, 5: May, 6: June, 7: July, 8: August, 9: September, 10: October, 11: November, 12: December }
Enter the month: 3
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 3
January, March, May,
1. Insert
2. Remove
3. Display
4. Destroy
```

2. removed month march (3) and displayed the tree



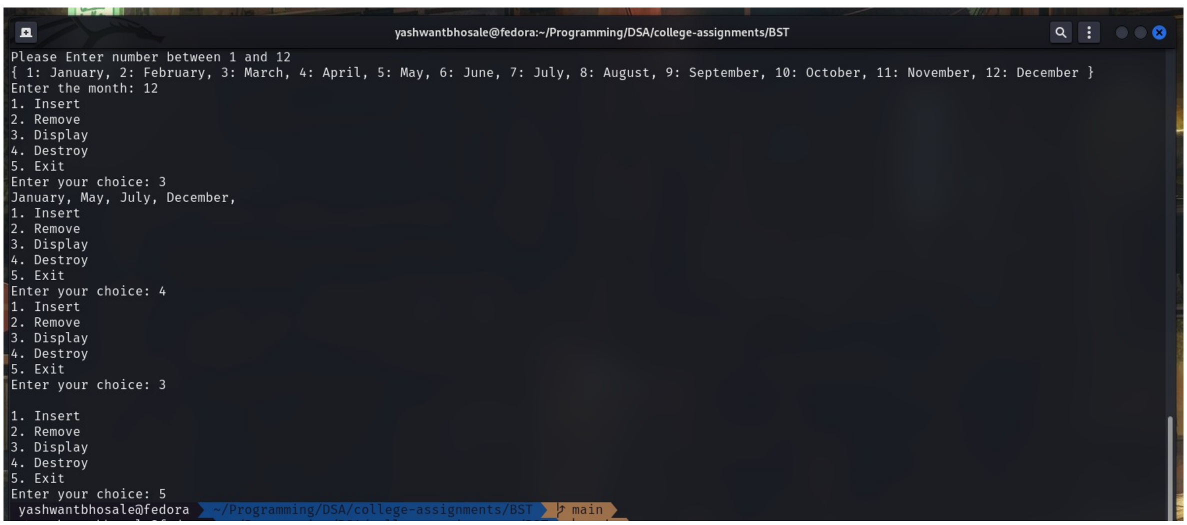
```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/BST
5. Exit
Enter your choice: 2
Please Enter number between 1 and 12
{ 1: January, 2: February, 3: March, 4: April, 5: May, 6: June, 7: July, 8: August, 9: September, 10: October, 11: November, 12: December }
Enter the month: 3
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 3
January, May,
```

3. Inserted months July (5) and December (12) into the tree and displayed them



```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/BST
5. Exit
Enter your choice: 1
Please Enter number between 1 and 12
{ 1: January, 2: February, 3: March, 4: April, 5: May, 6: June, 7: July, 8: August, 9: September, 10: October, 11: November, 12: December }
Enter the month: 7
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 3
January, May, July,
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 1
Please Enter number between 1 and 12
{ 1: January, 2: February, 3: March, 4: April, 5: May, 6: June, 7: July, 8: August, 9: September, 10: October, 11: November, 12: December }
Enter the month: 12
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 3
January, May, July, December,
```

4. Finally destroyed tree by choosing option 4, and displayed the tree. (Nothing was displayed as the tree is empty)



```
yashwantbhosale@fedora:~/Programming/DSA/college-assignments/BST
Please Enter number between 1 and 12
{ 1: January, 2: February, 3: March, 4: April, 5: May, 6: June, 7: July, 8: August, 9: September, 10: October, 11: November, 12: December }
Enter the month: 12
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 3
January, May, July, December,
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 4
1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 3

1. Insert
2. Remove
3. Display
4. Destroy
5. Exit
Enter your choice: 5
yashwantbhosale@fedora  ~/Programming/DSA/college-assignments/BST  main
```