**DAILY ONLINE ACTIVITIES SUMMARY**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Date:** | **23-06-2020** | | | | | **Name:** | **Ashwini** | |
| **Sem & Sec** | **6th  Sem ‘A’ Sec** | | | | | **USN:** | **4AL17CS017** | |
| **Online Test Summary** | | | | | | | | |
| **Subject** | | **java programming** | | | | | | |
| **Max. Marks** | | **--** | | **Score** | | | **--** | |
| **Certification Course Summary** | | | | | | | | |
| **Course** | **Workshop exercise solving** | | | | | | | |
| **Certificate Provider** | | | **--** | | **Duration** | | | **--** |
| **Coding Challenges** | | | | | | | | |
|  | | | | | | | | |
| **Status: done** | | | | | | | | |
| **Uploaded the report in Github** | | | | | **yes** | | | |
| **If yes Repository name** | | | | | <https://github.com/ashwiniachar/online-report> | | | |
| **Uploaded the report in slack** | | | | | **yes** | | | |

1. Write a C Program to Sort a stack using a temporary stack

We follow this algorithm.

1. Create a temporary stack say tmpStack.
2. While input stack is NOT empty do this:  
   • Pop an element from input stack call it temp  
   • while temporary stack is NOT empty and top of temporary stack is greater than temp,  
   pop from temporary stack and push it to the input stack  
   • push temp in temporary stack
3. The sorted numbers are in tmpStack

#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("\n\n");

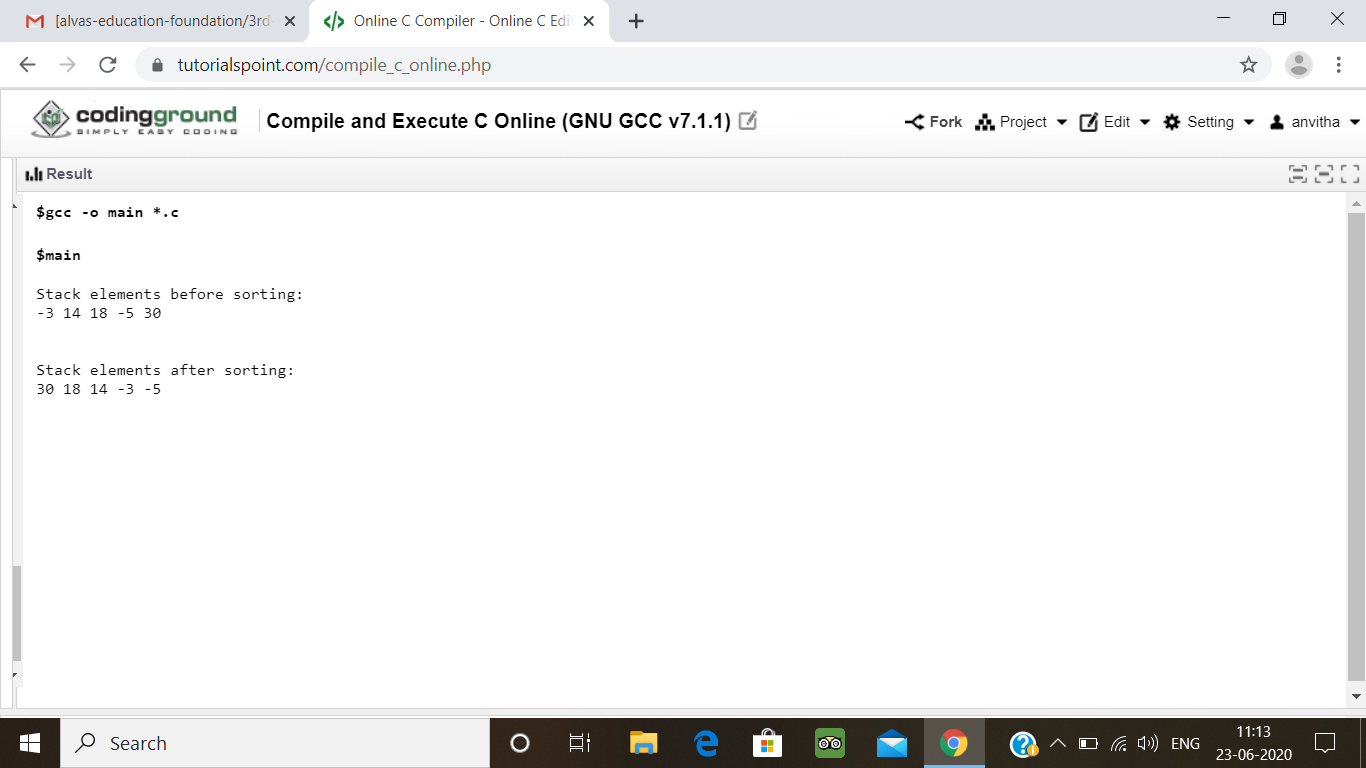
printf("Stack elements after sorting:\n");

printStack(top);

return 0;

}

**Output:**



2. Write a Java Program to traverse a binary tree using PreOrder traversal without recursion

import java.util.Stack;

public class BinaryTreePreOrder {

public static class TreeNode

{

int data;

TreeNode left;

TreeNode right;

TreeNode(int data)

{

this.data=data;

}

}

public void preorder(TreeNode root) {

if(root != null) {

System.out.printf("%d ",root.data);

preorder(root.left);

preorder(root.right);

}

}

public void preorderIter(TreeNode root) {

if(root == null)

return;

Stack<TreeNode> stack = new Stack<TreeNode>();

stack.push(root);

while(!stack.empty()){

TreeNode n = stack.pop();

System.out.printf("%d ",n.data);

if(n.right != null){

stack.push(n.right);

}

if(n.left != null){

stack.push(n.left);

}

}

}

public static void main(String[] args)

{

BinaryTreePreOrder bi=new BinaryTreePreOrder();

// Creating a binary tree

TreeNode rootNode=createBinaryTree();

System.out.println("Using Recursive solution:");

bi.preorder(rootNode);

System.out.println();

}

public static TreeNode createBinaryTree()

{

TreeNode rootNode =new TreeNode(40);

TreeNode node20=new TreeNode(20);

TreeNode node10=new TreeNode(10);

TreeNode node30=new TreeNode(30);

TreeNode node60=new TreeNode(60);

TreeNode node50=new TreeNode(50);

TreeNode node70=new TreeNode(70);

rootNode.left=node20;

rootNode.right=node60;

node20.left=node10;

node20.right=node30;

node60.left=node50;

node60.right=node70;

return rootNode;

}

}

**Output:**

