Name : Bryan Orville Audric  
NIM: 2602160750

struct node

int Data

// 1 Red, 0 Black

int Color

struct node \*Parent

struct node \*right

struct node \*left

END Struct

struct node \*root = NULL

void SwapColor(int \*a, int \*b):

int temp = \*a

\*a = \*b

\*b = temp

END Function

void RightRotate(struct node \*Current):

struct node \*LeftCurrent = Current->left

Current->left=LeftCurrent->right

If Current->left is not NULL

Current->left->Parent=Current

END IF

LeftCurrent->Parent=Current->Parent

If Current->Parent is NULL

root = LeftCurrent

else if Current is Current->Parent->left

Current->Parent->left= LeftCurrent

else

Current->Parent->right=LeftCurrent

END IF

LeftCurrent->right=Current

Current->Parent=LeftCurrent

END Function

void LeftRotate(struct node \*Current):

struct node \*RightCurrent = Current->right

Current->right=RightCurrent->left

If Current->right is not NULL

Current->right->Parent=Current

END IF

RightCurrent->Parent=Current->Parent

If Current->Parent is NULL

root=RightCurrent

else if Current is Current->Parent->left

Current->Parent->left=RightCurrent

else

Current->Parent->right=RightCurrent

END IF

RightCurrent->left=Current

Current->Parent=RightCurrent

END Function

struct node \*BinarySearchInsertion(struct node \*root,struct node \*Current):

if root is NULL

return Current

END IF

If Current->Data<root->Data

root->left=BinarySearchInsertion(root->left,Current)

root->left->Parent=root

else if Current->Data>root->Data

root->right=BinarySearchInsertion(root->right,Current)

root->right->Parent=root

END IF

return root

END Function

void Fix(struct node \*root, struct node \*Current):

struct node \*GrandParent = NULL

struct node \*Parent = NULL

while(Current is not root && Current->Color is 1 && Current->Parent->Color is 1)

Parent = Current->Parent

GrandParent = Current->Parent->Parent

If Parent is GrandParent->left

struct node \*ParentSibling = GrandParent->right

if ParentSibling is not NULL && ParentSibling->Color is 1

GrandParent->Color=1

Parent->Color=0

ParentSibling->Color=0

Current=GrandParent

else

if Current is Parent->right

LeftRotate(Parent)

Current = Parent

Parent = Current->Parent

END IF

RightRotate(GrandParent)

SwapColor(&GrandParent->Color,&Parent->Color)

Current = Parent

END IF

else

struct node \*ParentSibling = GrandParent->left

if ParentSibling is not NULL && ParentSibling->Color is 1

GrandParent->Color=1;

Parent->Color=0;

ParentSibling->Color=0;

Current=GrandParent;

else

if Current is Parent->left

RightRotate(Parent)

Current=Parent

Parent = Current->Parent

END IF

LeftRotate(GrandParent)

SwapColor(&GrandParent->Color,&Parent->Color)

Current = Parent

END IF

END IF

END while

END Function

void Insertion():

int Array[] = {41,22,5,51,48,29,18,21,45,3}

int Length = sizeof(Array)/sizeof(Array[0])

for int i=0;i<Length;i++

struct node \*NewNode = (struct node\*)malloc(sizeof(struct node))

NewNode->Data=Array[i]

NewNode->Color=1

NewNode->left=NULL

NewNode->right=NULL

NewNode->Parent=NULL

root = BinarySearchInsertion(root,NewNode)

Fix(root,NewNode)

root->Color=0

END for

END Function

void PrintInOrder(struct node \*root):

if root is not NULL

PrintInOrder(root->left)

Display "%d ",root->Data

PrintInOrder(root->right)

END IF

END Function

int main():

int menu

do

Display "1. Insertion\n"

Display "2. PrintInorder\n"

Display "3. Exit\n"

Display ">> "

Input menu

switch(menu)

case 1:

system("cls")

Insertion()

Display"[41,22,5,51,48,29,18,21,45,3]\nInserted\n"

system("pause")

system("cls")

break

case 2:

system("cls")

Display "InOrder Traversal\n"

PrintInOrder(root)

Display"\n"

system("pause")

system("cls")

break

END Switch

While menu is not 3

END do while

End Function