

**Data Structures and Algorithms [Assignment 3]**

**Submission Date: 10-June-2022, 11:59 PM**

**NAME: IQRA**

**REF#: L1F20BSSE0260**

**SECTION: “P8”**

# Question 1 [Marks 4]

An arithmetic sequence is a sequence of numbers such that the difference **d** between each consecutive term is a constant.

a, a+d, a+2d, a+3d …

nth term, an = a + (n-1) d

Write a recursive code in C++ that generates above sequence. The function accept two inputs **a** and **d** and generates above sequence.

**Answer**

#include <iostream>

using namespace std;

int Nth\_of\_AP(int a, int d, int N)

{

return (a + (N - 1) \* d);

}

int main()

{

int a = 0;

cout << "Enter starting number: ";

cin >> a;

int d = 0;

cout << "Enter Common Difference: ";

cin >> d;

int N = 0;

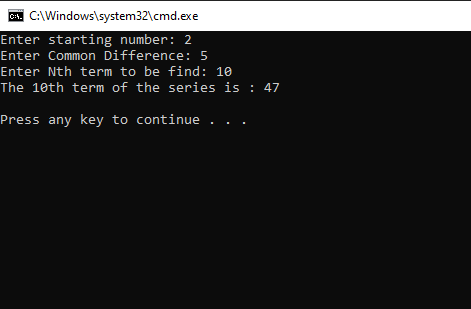
cout << "Enter Nth term to be find: ";

cin >> N;

cout << "The " << N << "th term of the series is : " << Nth\_of\_AP(a, d, N) << "\n\n";

return 0;

}



=========================

# Question 2 [Marks 3]

Write a recursive code in C++ that display all elements of a singly link list in a reverse order.

Example:

Input List: 2353439875Null

Output: 75, 98, 43, 53, 23

**Answer**

#include <iostream>

using namespace std;

struct Node {

int data;

struct Node\* next;

Node(int data)

{

this->data = data;

next = NULL;

}

};

struct LinkedList {

Node\* head;

LinkedList()

{

head = NULL;

}

Node\* reverse(Node\* head)

{

if (head == NULL || head->next == NULL)

return head;

Node\* rest = reverse(head->next);

head->next->next = head;

head->next = NULL;

return rest;

}

void print()

{

struct Node\* temp = head;

while (temp != NULL) {

cout << temp->data << " ";

temp = temp->next;

}

}

void push(int data)

{

Node\* temp = new Node(data);

temp->next = head;

head = temp;

}

};

int main()

{

LinkedList ll;

ll.push(23);

ll.push(53);

ll.push(43);

ll.push(98);

ll.push(75);

cout << "Given Linked List\n";

ll.print();

ll.head = ll.reverse(ll.head);

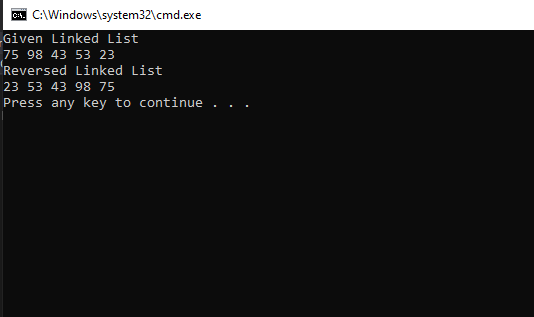
cout << "\nReversed Linked List \n";

ll.print();

cout << "\n";

return 0;

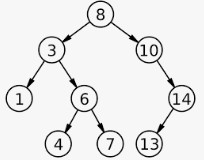
}



=======================

# Question 3 [Marks 3]

Let assume the following Binary Search Tree. Write a code in C++ that calculates the **sum** of the all values.



Example:

Output: 66

**Answer**

#include<iostream>

using namespace std;

struct Node

{

int data;

Node\* left;

Node\* right;

};

class Tree

{

protected:

Node\* root;

public:

Tree()

{

root = nullptr;

}

virtual void insert(int value) = 0;

virtual void inorder() = 0;

virtual void postorder() = 0;

virtual void preorder() = 0;

virtual bool deleteValue(int value) = 0;

};

class BST :public Tree

{

void INORDER(Node\* p);

void POSTORDER(Node\* p);

void PREORDER(Node\* p);

public:

void insert(int value);

void inorder();

void postorder();

void preorder();

bool deleteValue(int value);

int sum(Node \*root)

{

if (root == NULL)

return 0;

return (root->data + sum(root->left) + sum(root->right));

}

int sum()

{

return sum(root);

}

};

bool BST::deleteValue(int value)

{

if (root == nullptr)

return false;

else if (root->data == value && root->left == nullptr && root->right == nullptr) //single value placed on root that has to be deleted

{

delete root;

root = nullptr;

}

else

{

Node\*c = root;

Node\*p = root;

while (true)

{

if (value == c->data) //actual deletion

{

if (c->left != nullptr && c->right != nullptr)

{

Node\* t = c;

p = t;

c = t->right;

while (c->left != nullptr)

{

p = c;

c = c->left;

}

t->data = c->data;

}

if (c->left == nullptr && c->right == nullptr)

{

if (c->data < p->data)

{

delete c;

c = nullptr;

p->left = nullptr;

}

else

{

delete c;

c = nullptr;

p->right = nullptr;

}

}

else if (c->left != nullptr && c->right == nullptr)

{

if (c->data < p->data)

{

p->left = c->left;

delete c;

c = nullptr;

}

else

{

p->right = c->left;

delete c;

c = nullptr;

}

}

else if (c->left == nullptr && c->right != nullptr)

{

if (c->data < p->data) //left child of parent

{

p->left = c->right;

delete c;

c = nullptr;

}

else

{

p->right = c->right;

delete c;

c = nullptr;

}

}

return true;

}

else

{

if (value < c->data)

{

p = c;

c = c->left;

}

else

{

p = c;

c = c->right;

}

}

}

}

}

void BST::PREORDER(Node\* p)

{

if (p != nullptr)

{

cout << p->data << " ";

PREORDER(p->left);

PREORDER(p->right);

}

}

void BST::POSTORDER(Node\* p)

{

if (p != nullptr)

{

POSTORDER(p->left);

POSTORDER(p->right);

cout << p->data << " ";

}

}

void BST::INORDER(Node\* p)

{

if (p != nullptr)

{

INORDER(p->left);

cout << p->data << " ";

INORDER(p->right);

}

}

void BST::preorder()

{

if (root == nullptr)

cout << "Tree is empty" << endl;

else

PREORDER(root);

}

void BST::postorder()

{

if (root == nullptr)

cout << "Tree is empty" << endl;

else

POSTORDER(root);

}

void BST::inorder()

{

if (root == nullptr)

cout << "Tree is empty" << endl;

else

INORDER(root);

}

void BST::insert(int value)

{

Node \*n = new Node;

n->left = nullptr;

n->right = nullptr;

n->data = value;

if (root == nullptr)

{

root = n;

}

else

{

Node\*t = root;

while (1)

{

if (value<t->data) //left child case

{

if (t->left == nullptr) //insert the value

{

t->left = n;

break;

}

else

{

t = t->left;

}

}

else //right child case

{

if (t->right == nullptr) //insert value

{

t->right = n;

break;

}

else

{

t = t->right;

}

}

}

}

}

int main()

{

BST bst1;

bst1.insert(8);

bst1.insert(3);

bst1.insert(10);

bst1.insert(1);

bst1.insert(6);

bst1.insert(14);

bst1.insert(4);

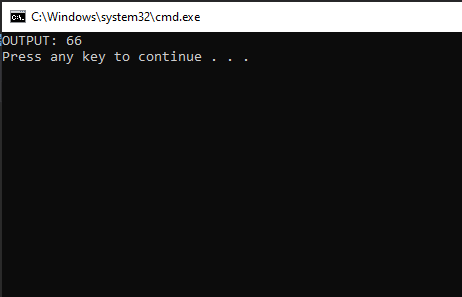
bst1.insert(7);

bst1.insert(13);

cout <<"OUTPUT: "<< bst1.sum() << "\n";

return 0;

}



===========================

THANK YOU