SET – 3

Ques1.

//source code

#include <iostream>

using namespace std;

template <class t>

class node

{

public:

t data;

node \*prev, \*next;

};

template <class t>

class DLinkedList

{

int n;

node<t> \*first, \*last;

public:

DLinkedList()

{

first = NULL;

last = NULL;

}

// create function

void create()

{

node<t> \*current, \*temp;

char ch;

first = new node<t>;

cout << "Enter data for first node:\n";

cin >> first->data;

current = first;

first->next = NULL;

first->prev = NULL;

last = first;

do

{

cout << "Want to enter more data(y/n):\n";

cin >> ch;

if (ch == 'y')

{

n = count();

this->insert(n + 1);

}

} while (ch == 'y');

}

// display function

void display()

{

node<t> \*current;

current = first;

cout << "The data in linked list:\n";

while (current != NULL)

{

cout << current->data << " <-> ";

current = current->next;

}

cout << "\n";

}

// reverse function

void reverse()

{

n = count();

// fflush(stdin);

node<t> \*current;

current = last;

cout << "The data after reversing the linked list:\n";

for (int i = 1; i <= n; i++)

{

cout << current->data << " -> ";

current = current->prev;

}

}

// count function

int count()

{

int c = 0;

node<t> \*current;

current = first;

while (current != NULL)

{

c++;

current = current->next;

}

return c;

}

// insert function

void insert(int n1)

{

int b = count();

if (n1 <= b + 1)

{

node<t> \*current, \*forward, \*temp;

current = first;

temp = new node<t>;

cout << "Enter data:\n";

cin >> temp->data;

temp->next = temp->prev = NULL;

if (n1 == 1)

{

temp->next = first;

first->prev = temp;

first = temp;

}

else if (n1 <= b)

{

for (int i = 1; i < n - 1; i++)

current = current->next;

forward = current->next;

temp->next = forward;

current->next = temp;

temp->prev = current;

forward->prev = temp;

}

else

{

last->next = temp;

temp->prev = last;

last = temp;

}

}

else

cout << "Can't be inserted\n";

}

// search function

void search()

{

int flag = 0;

cout << "Enter data to be searched:\n";

cin >> n;

node<t> \*current, \*previ, \*temp;

int b = count();

current = first;

for (int i = 1; i <= b; i++)

{

if (current->data == n)

{

flag = 1;

break;

}

current = current->next;

}

if (flag != 0)

{

previ = current->prev;

int c;

cout << "Data found";

}

else

cout << "Data not found:\n";

}

dlist operator+(dlist l)

{

dlist l6;

l6.first = first;

l6.last = last;

l6.last->next = l.first;

l.first->prev = l6.last;

return l6;

}

};

int main()

{

int choice, n;

char ch;

DLinkedList<int> l1, l3, l2;

l1.create();

l1.display();

do

{

cout << "\n\*\*\*\*\*\*\*\*Which operation you want to perform in Doubly Linked List \*\*\*\*\*\*\*\*\*\*\*";

cout << "\n1. Insert element(INSERTION).";

cout << "\n2. Deletion .";

cout << "\n3. Search an element.";

cout << "\n4. Reversing an element.";

cout << "\n5. Exit.";

cout << "\nEnter your choice: ";

cin >> choice;

switch (choice)

{

case 1:

// doing insertion

do

{

cout << "Want to insert a node(y/n):\n";

cin >> ch;

if (ch == 'y')

{

cout << "Enter the position of insertion;\n";

cin >> n;

l1.insert(n);

}

} while (ch == 'y');

cout << "The linked list after all insertions:\n";

l1.display();

case 2:

cout << "\nValue to be deleted from List. ";

cin >> n;

cout << "Data deleted:\n";

break;

break;

case 3:

cout << "search Element\n";

do

{

cout << "Want to search a data(y/n):\n";

cin >> ch;

if (ch == 'y')

l1.search();

} while (ch == 'y');

cout << "The linked list after searching and as so:\n";

l1.display();

break;

case 4:

cout << "Reversing The List.\n";

cout << "Want to see reversed linked list(y/n):\n";

cin >> ch;

if (ch == 'y')

{

l1.reverse();

}

cout << "The linked list after reversing:-\n";

l1.display();

break;

case 5:

cout << "\nExiting...";

exit(100);

break;

default:

cout << "\nWrong input!!!";

break;

}

cout << "\n\nReturn to menu(y/n): ";

cin >> ch;

} while (ch == 'y');

return 0;

}

Ques2.

//source code

#include<iostream>

using namespace std;

class Node{

public:

int val;

Node\* left;

Node\* right;

Node();

Node(int v){

val = v;

left = NULL;

right = NULL;

}

};

class BinarySearchTree{

public:

int data;

Node\* root;

BinarySearchTree(){

root = NULL;

}

void insertion();

void search();

};

void BinarySearchTree :: search(){

Node\* p;

int data;

bool flag = false;

cout << "Enter value to search : ";

cin >> data;

if(data == root->val){

cout << "Value found at root" << endl;

}

else{

p = root;

while(true){

if(data > p->val){

if(p->right == NULL){

flag = true;

break;

}

p = p->right;

}

else if(data < p->val){

if(p->left == NULL){

flag = true;

break;

}

p = p->left;

}

else{

cout << "Value found" << endl;

}

}

if(flag)

cout << "Value not found" << endl;

}

}

void BinarySearchTree :: insertion(){

Node\* p;

int value;

cout << "Value : ";

cin >> value;

if(root == NULL)

root = new Node(value);

else{

p = root;

while(true){

if(value > p->val){

if(p->right == NULL)

break;

p = p->right;

}

else{

if(p->left == NULL)

break;

p = p->left;

}

}

if(value > p->val)

p->right = new Node(value);

else

p->left = new Node(value);

}

}

void display(Node\* root){

if(root != NULL){

display(root->left);

display(root->right);

cout << root->val << " ";

}

}

void mirror\_image(Node\* root){

if(root!=NULL){

if(root->left){

mirror\_image(root->left);

}

if(root->right){

mirror\_image(root->right);

}

Node\* ptr=root->right;

root->right=root->left;

root->left=ptr;

return ;

}

return ;

}

int main() {

BinarySearchTree obj;

int choice;

while(1){

cout << "\n1.Insertion\n2.Mirror\n3.Display\n4.Search\n5.Exit\n";

cout << "Choice : ";

cin >> choice;

switch (choice)

{

case 1:

obj.insertion();

break;

case 2:

mirror\_image(obj.root);

break;

case 3:

display(obj.root);

break;

case 4:

obj.search();

break;

case 5:

exit(0);

default:

cout << " Wrong choice";

break;

}

}

return 0;

}