**NAME AREEB AHMED**

**REG. NO. FA19-BSE-022**

**ASSIGNMENT DSA LAB**

**SECTION B**

**SINGLY CIRCULAR LINKEDLIST:**

**CODE:**

#include<iostream>

using namespace std;

struct node{

char data;

node \*next;

};

class SinglyCircularLinkList{

private:

node\* head;

public:

SinglyCircularLinkList(){

head = NULL;

}

node\* create\_node(char data){

node\* link\_node = new node();

link\_node->data = data;

return link\_node;

}

void add\_node\_to\_tail(char data){

if (head == NULL){

head = create\_node(data);

head->next = head;

return;

}

node\* tail = get\_tail();

node\* link\_node = create\_node(data);

link\_node->next = tail->next;

tail->next = link\_node;

}

node\* get\_tail(){

node\* iter = head;

while(iter->next != head){

iter = iter->next;

}

return iter;

}

void print(){

node\* iter = head;

if (head == NULL) return;

do{

cout<<iter->data;

iter = iter->next;

}

while(iter != head);

cout<<endl;

}

node\* search\_node(char key){

node\* iter = head;

while(iter->data !=key){

iter = iter->next;

if (iter == head) return NULL;

}

return iter;

}

void insert\_at\_value(char info, char key){

if (head == NULL) return;

node\* search = search\_node(key);

node\* link\_node = create\_node(info);

link\_node->next = search->next;

search->next = link\_node;

}

node\* get\_prev(node\* random\_node){

node\* temp\_iter = head;

if (random\_node==NULL)

return NULL;

if (random\_node==head)return head;

while(temp\_iter->next!=random\_node){

temp\_iter=temp\_iter->next;

}

return temp\_iter;

}

void delete\_node(char key){

node\* del\_node = search\_node(key);

node\* tail = get\_tail();

node \* prev = get\_prev(del\_node);

if (del\_node == head){

tail->next = del\_node->next;

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

else head = head->next;

delete del\_node;

del\_node = NULL;

}

else {

prev->next = del\_node->next;

delete del\_node;

del\_node = NULL;

}

}

};

int main(){

SinglyCircularLinkList Ring;

Ring.add\_node\_to\_tail('h');

Ring.add\_node\_to\_tail('e');

Ring.add\_node\_to\_tail('l');

Ring.add\_node\_to\_tail('l');

Ring.add\_node\_to\_tail('o');

Ring.add\_node\_to\_tail(' ');

Ring.add\_node\_to\_tail('e');

Ring.add\_node\_to\_tail('v');

Ring.add\_node\_to\_tail('e');

Ring.add\_node\_to\_tail('r');

Ring.add\_node\_to\_tail('y');

Ring.add\_node\_to\_tail('b');

Ring.add\_node\_to\_tail('o');

Ring.add\_node\_to\_tail('d');

Ring.add\_node\_to\_tail('y');

Ring.add\_node\_to\_tail(' ');

Ring.add\_node\_to\_tail('h');

Ring.add\_node\_to\_tail('o');

Ring.add\_node\_to\_tail('w');

Ring.add\_node\_to\_tail(' ');

Ring.add\_node\_to\_tail('a');

Ring.add\_node\_to\_tail('r');

Ring.add\_node\_to\_tail('e');

Ring.add\_node\_to\_tail(' ');

Ring.add\_node\_to\_tail('y');

Ring.add\_node\_to\_tail('o');

Ring.add\_node\_to\_tail('u');

Ring.print();

return 0;

}

**DOUBLE LINKLIST**

**CODE:**

#include <iostream>

using namespace std;

struct Node

{

int data;

struct Node\* next;

struct Node\* prev;

};

void insert\_front(struct Node\*\* head, int new\_data)

{

struct Node\* newNode = new Node;

newNode->data = new\_data;

newNode->next = (\*head);

newNode->prev = NULL;

if ((\*head) != NULL)

(\*head)->prev = newNode;

(\*head) = newNode;

}

void insert\_After(struct Node\* prev\_node, int new\_data)

{

if (prev\_node == NULL)

{

cout << "Previous node is required , it cannot be NULL";

return;

}

struct Node\* newNode = new Node;

newNode->data = new\_data;

newNode->next = prev\_node->next;

prev\_node->next = newNode;

newNode->prev = prev\_node;

if (newNode->next != NULL)

newNode->next->prev = newNode;

}

void insert\_end(struct Node\*\* head, int new\_data)

{

struct Node\* newNode = new Node;

struct Node\* last = \*head;

newNode->data = new\_data;

newNode->next = NULL;

if (\*head == NULL) {

newNode->prev = NULL;

\*head = newNode;

return;

}

while (last->next != NULL)

last = last->next;

last->next = newNode;

newNode->prev = last;

return;

}

void displayList(struct Node\* node) {

struct Node\* last;

while (node != NULL) {

cout << node->data << "<==>";

last = node;

node = node->next;

}

if (node == NULL)

cout << "NULL";

}

int main() {

struct Node\* head = NULL;

insert\_end(&head, 40);

insert\_front(&head, 20);

insert\_front(&head, 10);

insert\_end(&head, 50);

insert\_After(head->next, 30);

cout << "Doubly linked list is as follows: " << endl;

displayList(head);

return 0;

}

**BST:**

**CODE:**

#include <iostream>

using namespace std;

struct node

{

int info;

node \*left,\*right;

};

class bin\_search\_tree

{

private :

node \*temp;

public :

node \*root;

int number;

bin\_search\_tree();

void options();

void b\_search\_tree(node \*);

void in\_order(node \*);

};

int main()

{

char ch;

bin\_search\_tree obj;

while( 4 )

{

obj.options();

cin >> ch;

switch(ch)

{

case '1':

cout<<"\n Enter number to add in a tree... \n";

cin>>obj.number;

obj.b\_search\_tree( obj.root );

break;

case '2':

obj.in\_order(obj.root);

break;

case '3':

exit(0);

break;

default :

exit(0);

break;

} // end of switch.

} // end of while.

}//-----------------------------------------------

bin\_search\_tree :: bin\_search\_tree()

{

root=temp=NULL;

}

//--------------------------------------------------------

void bin\_search\_tree :: b\_search\_tree(node \*temp)

{

if( root==NULL )

{

temp=new node;

temp->info = number;

temp->left=NULL;

temp->right=NULL;

root=temp;

return;

}

if( temp->info==number )

{

cout<<" \n Given number is already present in tree.\n";

return;

}

if(temp->info > number)

{

if( temp->left!=NULL )

{

b\_search\_tree(temp->left);

return;

}

else

{

temp->left=new node;

temp->left->info = number;

temp->left->left=NULL;

temp->left->right=NULL;

return;

}

}

if(temp->info < number)

{

if( temp->right!=NULL )

{

b\_search\_tree( temp->right );

return;

}

else

{

temp->right=new node;

temp->right->info = number;

temp->right->left=NULL;

temp->right->right=NULL;

return;

}

}

} //-------------------- Insertion Function Ends --------------------------

void bin\_search\_tree :: options()

{

cout<<"\n\n \*\*\*\*\*\*\*\*\*\*\*\*\*\* Select Option \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*.\n";

cout<<"\n Enter any of choices.\n";

cout<<"\n 1 : Adding (inserting) node in BST.\n";

cout<<"\n 2 : Print the whole BST .\n";

cout<<"\n 3 : Quitting the Program.\n";

}//--------------------------------------------------------------

void bin\_search\_tree :: in\_order(node \*temp)

{

if(root==NULL)

{

cout<<" Tree is empty.\n";

return;

}

if( temp->left!=NULL )

in\_order(temp->left);

cout<<temp->info<<" ";

if( temp->right!=NULL )

in\_order(temp->right);

return;

}//---------------------------- PROGRAM ENDS HERE ---------------------------------------