Stack by arrays

#include < iostream >

using namespace std;

class IntStack

{

public:

IntStack(int num) { top = 0; maxelem = num; s = new int[maxelem]; }

void push(int t)

{

if (top == maxelem) return;

s[top++] = t;

}

int pop()

{

if (top == 0) return -1;

return s[--top];

}

void display()

{

if (top == 0) { cout<<"(empty)\n"; return; }

for (int t=0 ; t < top ; t++) cout << s[t] << " ";

cout << "\n";

}

int empty() { return top == 0; }

private:

int \*s;

int top;

int maxelem;

};

void main()

{

IntStack \*s = new IntStack(100);

int d;

s->display();

s->push(1);

s->display();

s->push(2);

s->display();

s->push(3);

s->display();

s->push(4);

s->display();

s->pop();

s->display();

s->pop();

s->display();

s->push(10);

s->display();

s->pop();

s->display();

s->pop();

s->display();

s->pop();

s->display();

s->pop();

s->display();

s->pop();

s->display();

}

CONVERSION FROM DECIMAL TO BINARY

#include <iostream>

using namespace std;

class IntStack

{

public:

IntStack(int num) { top = 0; maxelem = num; s = new int[maxelem]; }

void decimal(int digit)

{

int reminder=0;

while(digit>0)

{

reminder=digit%2;

if (top == maxelem)

{ cout<<"OVERFLOW OCCUR"<<endl;

return;

}

else

s[top++] = reminder;

digit=digit/2;

}

}

void binary()

{

while(top!=0)

{

if (top == 0)

{ cout<<"an error"<<endl;

return;

}

else

cout<< s[--top];

}

}

private:

int \*s;

int top;

int maxelem;

};

void main()

{

int num;

cin>>num;021

IntStack obj(32);

obj.decimal(num);

obj.binary();

cout<<endl;

system("pause");

}

Circular queue

#include<iostream>

using namespace std;

class cqueue

{

int size;

int \*data;

int front,rear;

public:

cqueue(int num)

{

size=num;data=new int[size]; front=0; rear=0; }

void insert();

void remove();

};

void cqueue::insert()

{

if(rear==size && front==1 || front==rear+1)//when front is on start and rear is on end .ok afnan

{

cout << "\nCircular queue is full";

return;

}

else if(rear==0) //for initalization from start=1

{

rear++; front++;

}

else if(rear==size) //when rear is on end but there is space because front is checked in the first if condition that it is not on the start

rear=0;

else

rear++;

cout << "Enter Data : ";

cin >> data[rear];

}

void cqueue::remove()

{

if(front==0)

{

cout << "\n Circular Queue is empty";

return;

}

cout << data[front] << " deleted" << endl;

if(front==rear) //when all ints are deleted and front goes to rear.ok

{

front=0;rear=0;

}

else if(front==size)

front=1;

else

front++;

}

int main()

{

int qsize;

cout<<"Enter the size of queue ";

cin>>qsize;

cqueue cq(qsize);

int ch;

do

{

cout << "\n1. Insert\n2. Remove\n3. Quit\n Choice(1-3) ";

cin >> ch;

switch(ch)

{ case 1:

cq.insert();

break;

case 2:

cq.remove();

break;

}

}

while(ch!=3);

}

***TREES IMPLEMENTATION:***

#include<iostream>

#include <conio.h>

#include <fstream>

#include <stdlib.h>

using namespace std;

struct node

{

int info;

int repeat;

node \*right;

node \*left;

};

class btree

{

node \*temp;

public:

node \*root;

int number;

btree()

{

root=temp=0;

}

void insert(node \*temp)

{

if(root==NULL)

{

temp=new node;

temp->info=number;

temp->repeat=0;

temp->right=NULL;

temp->left=NULL;

root=temp;

return;

}

if(temp->info==number)

{

temp->repeat++;

cout<<"The number is present already ";

return;

}

if(temp->info>number)

{

if(temp->left!=NULL)

{

insert(temp->left);

return;

}

else

{

temp->left=new node;

temp->left->info=number;

temp->left->repeat=0;

temp->left->left=0;

temp->left->right=0;

return;

}

}

if(temp->info<number)

{

if(temp->right!=NULL)

{

insert(temp->right);

return;

}

else

{

temp->right=new node;

temp->right->info=number;

temp->right->repeat=0;

temp->right->right=NULL;

temp->right->left=NULL;

return;

}

}

}

void in\_order(node \*temp)

{

if(temp==NULL)

{ cout<<"THe tree is empty \n";

return;

}

if(temp->left!=NULL)

in\_order(temp->left);

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

if(temp->right!=0)

in\_order(temp->right);

return;

}

int depth(node \*temp)

{

if (temp==0)

{

return 0;

}

else

{ int smalldepth=depth(temp->left);

int largedepth=depth(temp->right);

if(smalldepth>largedepth)

return (smalldepth+1);

else

return (largedepth+1);

}

}

void pre\_order(node \*temp)

{

if(temp==NULL)

{ cout<<"THe tree is empty \n";

return;

}

if(temp->left!=NULL)

{

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

pre\_order(temp->left);

}

else cout<<"\t"<<temp->info<<" ";

if(temp->right!=0)

pre\_order(temp->right);

return;

}

void post\_order(node \*temp)

{

if(temp==NULL)

{ cout<<"THe tree is empty \n";

return;

}

if(temp->left!=NULL)

{

post\_order(temp->left);}

if(temp->right!=NULL)

{post\_order(temp->right);

}

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

return;

}

void deletecase(node \* &temp)

{

node \*temp2,\*a;

if(temp==NULL)

{cout<<"Empty tree "<<endl; return;}

if(temp->left==0 && temp->right==0)

{

delete temp;

temp=NULL; return;

}

if(temp->left!=0 && temp->right==0)

{

a=temp;

temp=temp->left;

delete a; return;

}

if(temp->left==0 && temp->right!=0)

{

a=temp;

temp=temp->right;

delete a; return;

}

if(temp->left!=0 && temp->right!=0)

{

temp2=temp->right;

while(temp2->left!=0)

{

temp2=temp->left;

}

temp->info=temp2->info;

a=temp2;

temp2=temp2->right;

delete a;

return;

}

}

//================================

void deletes(node \*&temp)

{

if(temp==NULL)

{

cout<<"Empty tree "<<endl;

return;

}

if(temp->info==number)

{

deletecase(temp);

}

else if(temp->info<number)

{

deletes(temp->right);

return;

}

else if(temp->info >number)

{

deletes(temp->left);

return;

}

else

cout<<"the number is not in the tree "<<endl;

}

void duplicate(node \*temp)

{

if(temp==0)

{

cout<<"The tree is empty "<<endl;

return;

}

if(temp->left!=0)

duplicate(temp->left);

cout<<temp->info<<"\t"<<temp->repeat<<endl;

if(temp->right!=0)

duplicate(temp->right);

}

void option()

{

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 : Deleting node in BST.\n";

cout<<"\n 3 : Print the whole BST (in order).\n";

cout<<"\n 3 : Print the whole BST (pre order).\n";

cout<<"\n 3 : Print the whole BST (post order).\n";

cout<<"\n 4 : Print the whole repition of BST .\n";

cout<<"\n 5 : Print the depth .\n";

cout<<"\n 6 : Quitting the Program.\n";

}

};

void main()

{

btree obj;

char ch;

ifstream in ;

in.open("in.txt");

while(!in.eof())

{

obj.option();

in>>ch;

switch(ch)

{

case '1':

system("cls");

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

in>>obj.number;

obj.insert(obj.root);

break;

case '2':

system("cls");

cout<<"Enter the number for deletion :";

cin>>obj.number;

obj.deletes(obj.root);

break;

case '3':

cout<<"in order : ";

obj.in\_order(obj.root);

cout<<"\npre order : ";

obj.pre\_order(obj.root);

cout<<"\npost order : ";

obj.post\_order(obj.root);

break;

case '4':

//system("cls");

cout<<"NUMBER\tNO OF REPEATITION"<<endl;

obj.duplicate(obj.root);

break;

case '5':

//system("cls");

cout<<"the depth "<<endl;

cout<<obj.depth(obj.root);

break;

case '6':

exit(0);

break;

default:

exit(0);

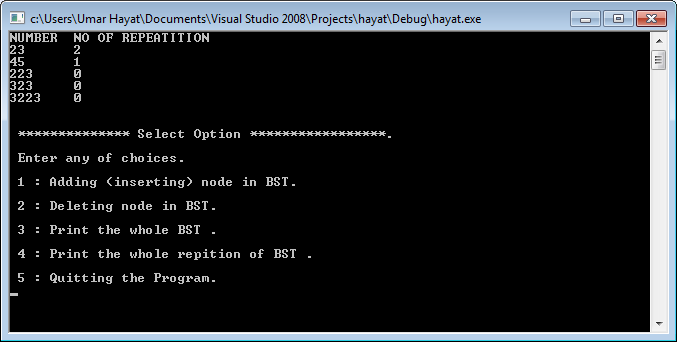
break;

}

}

system("pause");

}



***AVL***

#include <iostream>

#include<cctype>

#include <stdlib.h>

#include <conio.h>

using namespace std;

struct node

{

int element;

node \*left;

node \*right;

int height;

};

typedef struct node \*nodeptr;

class bstree

{

public:

void insert(int,nodeptr &);

void del(int, nodeptr &);

int deletemin(nodeptr &);

void find(int,nodeptr &);

nodeptr findmin(nodeptr);

nodeptr findmax(nodeptr);

void makeempty(nodeptr &);

void copy(nodeptr &,nodeptr &);

nodeptr nodecopy(nodeptr &);

void preorder(nodeptr);

void inorder(nodeptr);

void postorder(nodeptr);

int bsheight(nodeptr);

nodeptr srl(nodeptr &);

nodeptr drl(nodeptr &);

nodeptr srr(nodeptr &);

nodeptr drr(nodeptr &);

int max(int,int);

int nonodes(nodeptr);

};

// Inserting a node

void bstree::insert(int x,nodeptr &p)

{

if (p == NULL)

{

p = new node;

p->element = x;

p->left=NULL;

p->right = NULL;

p->height=0;

if (p==NULL)

{

cout<<"Out of Space\n"<<endl;

}

}

else

{

if (x<p->element)

{

insert(x,p->left);

if ((bsheight(p->left) - bsheight(p->right))==2)

{

if (x < p->left->element)

{

p=srl(p);

}

else

{

p = drl(p);

}

}

}

else if (x>p->element)

{

insert(x,p->right);

if ((bsheight(p->right) - bsheight(p->left))==2)

{

if (x > p->right->element)

{

p=srr(p);

}

else

{

p = drr(p);

}

}

}

else

{

cout<<"Element Exists\n"<<endl;

}

}

int m,n,d;

m=bsheight(p->left);

n=bsheight(p->right);

d=max(m,n);

p->height = d + 1;

}

// Finding the Smallest

nodeptr bstree::findmin(nodeptr p)

{

if (p==NULL)

{

cout<<"The tree is empty\n"<<endl;

return p;

}

else

{

while(p->left !=NULL)

{

p=p->left;

//return p;

}

return p;

}

}

// Finding the Largest node

nodeptr bstree::findmax(nodeptr p)

{

if (p==NULL)

{

cout<<"The tree is empty\n"<<endl;

return p;

}

else

{

while(p->right !=NULL)

{

p=p->right;

//return p;

}

return p;

}

}

// Finding an element

void bstree::find(int x,nodeptr &p)

{

if (p==NULL)

{

cout<<"Sorry! element not found\n"<<endl;

}

else

{

if (x < p->element)

{

find(x,p->left);

}

else

{

if (x>p->element)

{

find(x,p->right);

}

else

{

cout<<"Element found!\n"<<endl;

}

}

}

}

// Copy a tree

void bstree::copy(nodeptr &p,nodeptr &p1)

{

makeempty(p1);

p1 = nodecopy(p);

}

// Make a tree empty

void bstree::makeempty(nodeptr &p)

{

nodeptr d;

if (p != NULL)

{

makeempty(p->left);

makeempty(p->right);

d=p;

free(d);

p=NULL;

}

}

// Copy the nodes

nodeptr bstree::nodecopy(nodeptr &p)

{

nodeptr temp;

if (p==NULL)

{

return p;

}

else

{

temp = new node;

temp->element = p->element;

temp->left = nodecopy(p->left);

temp->right = nodecopy(p->right);

return temp;

}

}

// Deleting a node

void bstree::del(int x,nodeptr &p)

{

nodeptr d;

if (p==NULL)

{

cout<<"Sorry! element not found\n"<<endl;

}

else if ( x < p->element)

{

del(x,p->left);

}

else if (x > p->element)

{

del(x,p->right);

}

else if ((p->left == NULL) && (p->right == NULL))

{

d=p;

free(d);

p=NULL;

cout<<"Element deleted successfully\n"<<endl;

}

else if (p->left == NULL)

{

d=p;

free(d);

p=p->right;

cout<<"Element deleted successfully\n"<<endl;

}

else if (p->right == NULL)

{

d=p;

p=p->left;

free(d);

cout<<"Element deleted successfully\n"<<endl;

}

else

{

p->element = deletemin(p->right);

}

}

int bstree::deletemin(nodeptr &p)

{

int c;

cout<<"inside deltemin\n"<<endl;

if (p->left == NULL)

{

c=p->element;

p=p->right;

return c;

}

else

{

c=deletemin(p->left);

return c;

}

}

void bstree::preorder(nodeptr p)

{

if (p!=NULL)

{

cout<<p->element<<"\t";

preorder(p->left);

preorder(p->right);

}

}

// Inorder Printing

void bstree::inorder(nodeptr p)

{

if (p!=NULL)

{

inorder(p->left);

cout<<p->element<<"\t";

inorder(p->right);

}

}

// PostOrder Printing

void bstree::postorder(nodeptr p)

{

if (p!=NULL)

{

postorder(p->left);

postorder(p->right);

cout<<p->element<<"\t";

}

}

int bstree::max(int value1, int value2)

{

return ((value1 > value2) ? value1 : value2);

}

int bstree::bsheight(nodeptr p)

{

int t;

if (p == NULL)

{

return -1;

}

else

{

t = p->height;

return t;

}

}

nodeptr bstree:: srl(nodeptr &p1)

{

nodeptr p2;

p2 = p1->left;

p1->left = p2->right;

p2->right = p1;

p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1;

p2->height = max(bsheight(p2->left),p1->height) + 1;

return p2;

}

nodeptr bstree:: srr(nodeptr &p1)

{

nodeptr p2;

p2 = p1->right;

p1->right = p2->left;

p2->left = p1;

p1->height = max(bsheight(p1->left),bsheight(p1->right)) + 1;

p2->height = max(p1->height,bsheight(p2->right)) + 1;

return p2;

}

nodeptr bstree:: drl(nodeptr &p1)

{

p1->left=srr(p1->left);

return srl(p1);

}

nodeptr bstree::drr(nodeptr &p1)

{

p1->right = srl(p1->right);

return srr(p1);

}

int bstree::nonodes(nodeptr p)

{

int count=0;

if (p!=NULL)

{

nonodes(p->left);

nonodes(p->right);

count++;

}

return count;

}

int main()

{

//clrscr();

nodeptr root,root1,min,max;//,flag;

int a,choice,findele,delele;

char ch='y';

bstree bst;

//system("clear");

root = NULL;

root1=NULL;

cout<<"\n\t\t\t\tWELCOME TO AVL TREE"<<endl;

cout<<"\t\t\t\t:::::::::::::::::::\n"<<endl;

do

{

cout<<"\t\t::::::::::::::::::::::::::::::::::::::::::::::::"<<endl;

cout<<"\t\t::::Enter 1 to insert a new node::::::::::::::::"<<endl;

cout<<"\t\t::::Enter 2 to find the minimum value:::::::::::"<<endl;

cout<<"\t\t::::Enter 3 to find the max value:::::::::::::::"<<endl;

cout<<"\t\t::::Enter 4 to search a value:::::::::::::::::::"<<endl;

cout<<"\t\t::::Enter 5 to delete a value:::::::::::::::::::"<<endl;

cout<<"\t\t::::Enter 6 to display Preorder:::::::::::::::::"<<endl;

cout<<"\t\t::::Enter 7 to display Inorder::::::::::::::::::"<<endl;

cout<<"\t\t::::Enter 8 to display Postorder::::::::::::::::"<<endl;

cout<<"\t\t::::Enter 9 to display the height of the tree:::"<<endl;

cout<<"\t\t::::Enter 0 to exit:::::::::::::::::::::::::::::"<<endl;

cout<<"\t\t::::::::::::::::::::::::::::::::::::::::::::::::\n"<<endl;

cout<<"\nEnter the choice: ";

cin>>choice;

switch(choice)

{

case 1:

cout<<"\n\t\tADDING NEW NODE"<<endl;

cout<<"\t\t:::::::::::::\n"<<endl;

cout<<"Enter a new value: ";

cin>>a;

bst.insert(a,root);

cout<<"\nThe new value have been added to your tree successfully\n"<<endl;

break;

case 2:

if (root !=NULL)

{

min=bst.findmin(root);

cout<<"\nThe minimum element in the tree is: "<<min->element<<endl;

}

break;

case 3:

if (root !=NULL)

{

max=bst.findmax(root);

cout<<"\nThe maximum element in the tree is: "<<max->element<<endl;

}

break;

case 4:

cout<<"\nEnter node to search: ";

cin>>findele;

if (root != NULL)

{

bst.find(findele,root);

}

break;

case 5:

cout<<"\nEnter node to delete: ";

cin>>delele;

bst.del(delele,root);

bst.inorder(root);

cout<<endl;

break;

case 6:

cout<<"\n\t\tPRE-ORDER TRAVERSAL"<<endl;

bst.preorder(root);

cout<<endl;

break;

case 7:

cout<<"\n\t\tIN-ORDER TRAVERSAL"<<endl;

bst.inorder(root);

cout<<endl;

break;

case 8:

cout<<"\n\t\tPOST ORDER TRAVERSAL"<<endl;

bst.postorder(root);

cout<<endl;

break;

case 9:

cout<<"\n\t\tHEIGHT\n"<<endl;

cout<<"The height of the tree is: "<<bst.bsheight(root)<<endl;

break;

case 0:

cout<<"\n\tThank your for using AVL tree program\n"<<endl;

break;

default:

cout<<"Sorry! wrong input\n"<<endl;

break;

}

system("pause");

system("cls");

}while(choice != 0);

return 0;

}