

Student management system



**Project ID:** PM-872

**BY:**

20170090 حسام محمد عبود عبدالخالق

20180131 شاهيناز يحيى محمد ضيف

20180308 ندى حسام الدين محمود محمد

20180094 خالد اسامه الشاذلى يوسف كيلاني

June, 2020

**PREPARED FOR**

**Dr. Basheer Abdel Fatah**

Table of Contents

**A - Student management system 1**

**Header: linkedlist.h 1**

**Main.cpp 5**

**B - Solve the following problems 6**

**I – In-place merge two sorted arrays 6**

**Main.cpp 6**

**II – Assume you have the following Tree Struct. 8**

**Main.cpp 8**

**III – Balanced String 20**

**Main.cpp 20**

A - Student management system

**Header: linkedlist.h**

#ifndef LINKEDLIST\_H

#define LINKEDLIST\_H

#include <iostream>

using namespace std;

struct course

{

string name;

int total;

string grade;

double point;

course\* next;

};

struct lisst

{

string name;

string dept;

int courses\_num=0;

course\* course\_list=NULL;

lisst\* next;

};

class linkedlist

{

public:

linkedlist(){}

void studentinsert(string name,string dept)

{

lisst \*temp\_element=new lisst ;

temp\_element->name=name;

temp\_element->dept=dept;

temp\_element->course\_list=NULL;

temp\_element->next=NULL;

if(listhead==NULL)

{

listhead=temp\_element;

listtail=temp\_element;

}

else

{

listtail->next=temp\_element;

listtail=listtail->next;

}

}

void courseinsert(string name,int total,string grade,double point)

{

course \*temp\_element=new course ;

temp\_element->name=name;

temp\_element->total=total;

temp\_element->grade=grade;

temp\_element->point=point;

temp\_element->next=NULL;

if(listtail->course\_list==NULL)

{

listtail->course\_list=temp\_element;

listtail->courses\_num++;

}

else

{

course \*next\_element=new course ;

next\_element=listtail->course\_list;

while(true)

{

if(next\_element->next==NULL)

break;

next\_element=next\_element->next;

}

next\_element->next=temp\_element;

listtail->courses\_num++;

}

}

void print()

{

lisst\* print=new lisst;

print=listhead;

while(print)

{

cout<<"Student name: "<<print->name<<endl;

cout<<"Student department: "<<print->dept<<endl;

cout<<"Number of courses: "<<print->courses\_num<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

while(print->course\_list)

{

cout<<"Course name: "<<print->course\_list->name<<endl;

cout<<"Course total: "<<print->course\_list->total<<endl;

cout<<"Course grade: "<<print->course\_list->grade<<endl;

cout<<"Course point: "<<print->course\_list->point<<endl;

cout<<"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_"<<endl;

print->course\_list=print->course\_list->next;

}

cout<<"-----------------------------"<<endl;

print=print->next;

}

}

virtual ~linkedlist(){}

private:

lisst\* listhead=NULL;

lisst\* listtail=NULL;

};

#endif // LINKEDLIST\_H

**Main.cpp**

#include <iostream>

#include "linkedlist.h"

using namespace std;

int main()

{

linkedlist student;

student.studentinsert("Hossam Mohamed","CS");

student.courseinsert("DS",100,"A+",4);

student.courseinsert("OOP",89,"A",3.7);

student.courseinsert("Stat",70,"C+",2.7);

student.courseinsert("Math",95,"A+",4);

student.studentinsert("Mohamed Ali","IS");

student.courseinsert("DS",100,"A",3.7);

student.studentinsert("Seif Mahmoud","DS");

student.studentinsert("Ahmed Abd El-Rahman","AI");

student.courseinsert("Software-2",50,"D",2);

student.courseinsert("AI",98,"A",4);

student.courseinsert("Network",98,"A+",4);

student.courseinsert("Math-2",83,"B+",3.3);

student.courseinsert("Math-3",100,"A+",4);

student.studentinsert("Mahmoud Abd El-Rahman","IS");

student.courseinsert("Software-1",80,"B+",3.3);

student.courseinsert("IS",64,"D+",2.2);

student.studentinsert("Abd El-Rahman Seif","CS");

student.courseinsert("Software-2",78,"B",3);

student.courseinsert("IT",98,"D",4);

student.print();

return 0;}

B - Solve the following problems

**I – In-place merge two sorted arrays.**

**Main.cpp**

#include <iostream>

using namespace std;

void sort\_arraies(int x[],int y[],int m,int n);

void print\_arraies(int x[],int y[],int m,int n);

int main()

{

int x[]={2, 4, 6, 8, 10, 12, 14, 16, 18, 20};

int y[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

sort\_arraies(x,y,10,10);

print\_arraies(x,y,10,10);

int x1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

int y1[]={1, 3, 5, 7, 9, 11, 13, 15, 17, 19};

sort\_arraies(x1,y1,10,10);

print\_arraies(x1,y1,10,10);

int x2[]={1, 4, 7, 8, 10};

int y2[]={2, 3, 9};

sort\_arraies(x2,y2,5,3);

print\_arraies(x2,y2,5,3);

int x3[]={9, 100, 2000, 30000, 100000};

int y3[]={2, 3, 9};

sort\_arraies(x3,y3,5,3);

print\_arraies(x3,y3,5,3);

return 0;

}

void sort\_arraies(int x[],int y[],int m,int n)

{

for(int i=0 ; i<m ;i++){

for(int j=n-1 ;j>=0 ;j--){

if(x[i]>y[j]){

swap(x[i],y[j]);

}

}

}

}

void print\_arraies(int x[],int y[],int m,int n)

{

cout<<"---------------------------"<<endl;

cout<<"the first array is"<<endl;

for(int i=0 ; i<m ;i++){

cout<<x[i]<<endl;

}

cout<<"---------------------------"<<endl;

cout<<"the second array is"<<endl;

for(int j=0 ; j<n ;j++){

cout<<y[j]<<endl;

}

}

**II – Assume you have the following Tree Struct.**

**Main.cpp**

#include <iostream>

#include <queue>

#include <deque>

using namespace std;

template <class t>

struct tree

{

t value;

tree \*right,\*left;

tree();

tree(const t&v):value(v),left(NULL),right(NULL){}

};

template <class t>

class bt

{

tree <t> \*root;

int lengh=0;

queue<tree<int>\*> q;

int arr2[1000];

public:

bt()

{

root=NULL;

}

void insertt(t item)

{

tree <t> \*newtree=new tree <t> (item);

tree <t> \*current;

if(root == NULL)

{

root=newtree;

q.push(root);

}

else

{

current=q.front();

if(current->left==NULL)

{

current->left=newtree;

q.push(current->left);

}

else if(current->right==NULL)

{

current->right=newtree;

q.push(current->right);

q.pop();

}

}

}

tree <t> \*getroot()

{

return root;

}

void preorder1(struct tree <t> \*p)

{

int static x=0;

if (p==NULL)

{

return;

}

cout<<p->value<<endl;

preorder1(p->left);

preorder1(p->right);

}

void preorder()

{

if(root!=NULL)

preorder1(root);

}

void preorder(tree <t> \*p)

{

if(root!=NULL)

preorder1(p);}

void inorder1(tree <t> \*p)

{

int static x=0;

if (p==NULL)

{

return;

}

inorder1(p->left);

cout<<p->value<<endl;

inorder1(p->right);

}

void inorder(tree <t> \*p)

{

if(root!=NULL)

inorder1(p);

}

void inorder()

{

if(root!=NULL)

inorder1(root);

}

void postorder1(tree <t> \*p)

{

int static x=0;

if (p==NULL)

{

return;

}

postorder1(p->left);

postorder1(p->right);

cout<<p->value<<endl;

}

void postorder()

{

if(root!=NULL)

postorder1(root);}

void postorder(tree <t> \*p)

{

if(root!=NULL)

postorder1(p);

}

void doflipping(tree <t> \*treed)

{

if(treed==NULL)

return;

swap(treed->right,treed->left);

// if(treed->left!=NULL)

flipping(treed->left);

//if (treed->right!=NULL)

flipping(treed->right);

}

void flipping (tree <t> \*p)

{

doflipping(p);

}

void flipping ()

{

doflipping(root);

}

void highstvalue(tree <t> \*root)

{

int j=0;

int static z=1;

int maxi=NULL;

static int k=2;

if(root==NULL)

return;

queue <tree <t> \*>q;

deque <int>q2;

deque <int>q3;

q.push(root);

q3.push\_back(root->value);

while(!q.empty())

{

tree <t> \*curr=q.front();

if(curr->left!=NULL)

{

q.push(curr->left);

q2.push\_back(curr->left->value);

}

else

j++;

if(curr->right!=NULL)

{

q.push(curr->right);

q2.push\_back(curr->right->value);

}

else

j++;

q.pop();

while (q2.size()>=k-j&&j!=k)

{

//cout<<j;

for(int i=0;i<k-j;i++)

{

maxi=max((q2[i]),maxi);

q2.pop\_back();

}

// cout<<" MM"<<maxi <<" ";

q3.push\_back(maxi);

k\*=2;

j=0;

z++;

maxi=NULL;

}

}

for(int i=0;i<q3.size();i++)

{

cout<<q3[i]<<" ";

}

}

void printhighstvalue()

{

cout<<"[";

highstvalue(root);

cout<<"]";

}

int leaveNo()

{

return leavecount(root);

}

int leavecount(tree<int>\* p)

{

if(p!=NULL)

{

if((p->right!=NULL)||(p->left!=NULL))

{

return leavecount(p->left)+leavecount(p->right);

}

else

return 1;

}

}

void brunchesSum()

{

int arr[1000];

int sum=0;

int arr3[leavecount(root)];

searchforleave(root,arr,0,arr3);

for (int i=0 ;i<leavecount(root) ;i++)

{

sum+=arr3[i];

}

cout<<"sum= "<<sum<<" (";

for (int i=0 ;i<leavecount(root) ;i++)

{

cout<<arr3[i];

if(i!=leavecount(root)-1)

cout<<"+";

}

cout<<")"<<endl;

}

void Path(int arr[],int sizee,int arr3[])

{

static int index=-1;

index++;

int y=sizee-1;

cout<<"path->";

for(int i=0 ;i<sizee ;i++)

{

cout<<arr[i];

if(i!=sizee-1)

cout<<"->";

arr2[i]=1;

for(int j=0 ; j<y ; j++)

{

arr2[i]\*=10;

}

y--;

arr2[i]\*=arr[i];

}

for(int i=0 ;i<sizee-1; i++)

{

arr2[0]+=arr2[i+1];

}

arr3[index]=arr2[0];

cout<<" encodes "<<arr3[index]<<endl;

cout<<"------"<<endl;

}

void searchforleave(tree<int>\* p,int arr[],int x,int arr3[])

{

if (p!=NULL)

{

arr[x]=p->value;

x++;

if(p->left!=NULL || p->right!=NULL)

{

searchforleave(p->left,arr,x,arr3);

searchforleave(p->right,arr,x,arr3);

}

else

{

Path(arr,x,arr3);

}

}

}

};

int main()

{

bt<int>x;

x.insertt(1);

x.insertt(2);

x.insertt(3);

x.insertt(4);

x.insertt(5);

cout<<"TEST CASE 1"<<endl;

cout<<"preorder"<<endl;

x.preorder();

cout<<endl<<"postorder"<<endl;

x.postorder();

cout<<endl<<"inorder"<<endl;

x.inorder();

cout<<endl<<"Flipping and print by inorder"<<endl;

x.flipping();

x.inorder();

cout<<endl<<"Largest value"<<endl;

x.printhighstvalue();

cout<<endl;

x.brunchesSum();

cout<<endl<<endl;

cout<<"TEST CASE 2"<<endl;

bt<int>y;

y.insertt(1);

y.insertt(2);

y.insertt(3);

y.insertt(4);

y.insertt(5);

cout<<"preorder for leftnode"<<endl;

y.preorder(y.getroot()->left);

cout<<endl<<"postorder for leftnode"<<endl;

y.postorder(y.getroot()->left);

cout<<endl<<"inorder for leftnode"<<endl;

y.inorder(y.getroot()->left);

cout<<endl<<"Flipping for leftnode and print by inorder"<<endl;

y.flipping(y.getroot()->left);

y.inorder();

return 0; }

**III – Balanced String.**

**Main.cpp**

#include <iostream>

#include <deque>

using namespace std;

void Balanced(string s);

int main()

{

Balanced("{()}[][{()()}()]");

Balanced("{(])");

Balanced("({}{}{{}}[]{{[][][]}})()()");

Balanced("{}()[]");

Balanced(")({}{}[]");

return 0;

}

void Balanced(string s)

{

deque <int> dq;

for(int i=0;i<s.size();i++)

{

dq.push\_back(s[i]);

}

int y=0;

for(int i=0;i<s.size();i++)

{

int x=i+1;

while (x<=s.size())

{

if(dq[i]==0)

{

break;

}

if((dq[i]=='['&&dq[x]==']')||(dq[i]=='('&&dq[x]==')')|| (dq[i]=='{'&&dq[x]=='}'))

{

// cout<<"1"<<endl;

dq[i]=dq[x]=0;

break;

}

else

{

x=x+2;

}

}

}

for(int i=0;i<s.size();i++)

{

if(dq[i]!=0)

y=1;

}

if(y==0)

{

cout<<"Balanced"<<endl;

}

else

cout<<"NOt Balanced"<<endl;

}