Name: Swarnabh Paul

Section: Y

Roll no: 19CS8122

Assignment no: 8

Questions attempted: 1,2,3,4,5

**Question (1)**

**Code:**

#include<iostream>

**using** **namespace** std;

**class** **Salary**

{

**float** basic, gross;

**public:**

**static** **float** da;

**const** **int** bonus;

**const** **int** &dummy;

**float** **calcsal**();

**void** display() **const**;

Salary(**float**);

**static** **void** **dahike**(**float**);

**friend** **float** **average**(**const** Salary&,**const** Salary&);

};

**float** Salary::da=**1**;

**void** Salary::dahike(**float** d)

{

da+=d;

}

Salary::Salary(**float** b=**10000**): bonus(**2000**), dummy(bonus)

{

gross=basic=b;

}

**float** Salary::calcsal()

{

gross=basic+(bonus+(basic\*da/**100**));

**return** gross;

}

**void** Salary::display() **const**

{

cout<<"Basic: "<<basic<<endl;

cout<<"Bonus: "<<bonus<<endl;

cout<<"Da: "<<(basic\*da/**100**)<<endl;

cout<<"Gross: "<<gross<<endl;

}

**float** average(**const** Salary &s1,**const** Salary &s2)

{

**float** avg=(s1.gross+s2.gross)/**2**;

**return** avg;

}

**int** main()

{

Salary emp1(**50000**), emp2(**80000**);

emp1.calcsal();

emp2.calcsal();

cout<<"For emp1:"<<endl;

emp1.display();

cout<<endl<<"For emp2:"<<endl;

emp2.display();

cout<<endl<<"Average gross salary: "<<average(emp1,emp2);

**return** **0**;

}

**Output:**

For emp1:

Basic: 50000

Bonus: 2000

Da: 500

Gross: 52500

For emp2:

Basic: 80000

Bonus: 2000

Da: 800

Gross: 82800

Average gross salary: 67650

**Question (2)**

**Code:**

#include<iostream>

#include<stdbool.h>

**using** **namespace** std;

**class** **stackOverflow**

{

};

**class** **stackUnderflow**

{

};

**template** <**class** **Z**,**int** t=**5**>

**class** **stack**

{

**private:**

**int** top, size;

Z \*a;

**bool** **isempty**();

**bool** **isfull**();

**void** **initialize**(**int**);

**void** **deconstruct**();

**public:**

stack(**int** n=t);

~stack();

**void** **push**(Z);

Z **pop**();

**void** **display**();

};

**template** <**class** **Z**,**int** t>

**void** stack<Z,t>::push(Z x)

{

**if**(isfull())

{

**throw** stackOverflow();

}

top++;

a[top]=x;

}

**template** <**class** **Z**,**int** t>

Z stack<Z,t>::pop()

{

**if**(isempty())

{

**throw** stackUnderflow();

}

Z x=a[top];

top--;

**return** x;

}

**template** <**class** **Z**,**int** t>

stack<Z,t>::stack(**int** n)

{

initialize(n);

cout<<"Constructed stack of size "<<n<<endl;

}

**template** <**class** **Z**,**int** t>

stack<Z,t>::~stack()

{

deconstruct();

cout<<"Destroyed stack of size "<<size<<endl;

}

**template** <**class** **Z**,**int** t>

**void** stack<Z,t>::display()

{

**if**(isempty())

{

cout<<"Stack is empty**\n**";

**return**;

}

cout<<"Displaying stack from top to bottom:**\n**";

**for**(**int** i=top;i>=**0**;i--)

cout<<a[i]<<' ';

cout<<endl;

}

**template** <**class** **Z**,**int** t>

**bool** stack<Z,t>::isempty()

{

**return** (top==-**1**);

}

**template** <**class** **Z**,**int** t>

**bool** stack<Z,t>::isfull()

{

**return** (top==(size-**1**));

}

**template** <**class** **Z**,**int** t>

**void** stack<Z,t>::initialize(**int** n)

{

a=**new** Z[n];

top=-**1**;

size=n;

}

**template** <**class** **Z**,**int** t>

**void** stack<Z,t>::deconstruct()

{

**delete** []a;

}

**int** main(**void**)

{

stack<string,**10**> sch1(**3**);

try

{

sch1.push("Alter");

sch1.push("Deter");

sch1.push("Glue");

sch1.push("Critical");

}

**catch**(stackOverflow e)

{

cout<<"Stack overflow!"<<endl;

}

sch1.display();

try

{

**for**(**int** i=**0**;i<**4**;i++)

cout<<sch1.pop()<<' ';

cout<<endl;

}

**catch**(stackUnderflow e)

{

cout<<"**\n**Stack underflow!"<<endl;

}

**return** **0**;

}

**Output:**

Constructed stack of size 3

Stack overflow!

Displaying stack from top to bottom:

Glue Deter Alter

Glue Deter Alter

Stack underflow!

Destroyed stack of size 3

**Question (3)**

**Code:**

#include <iostream>

**using** **namespace** std;

**template** <**class** **Z**,**int** E=**0**>

**class** **node**

{

**public:**

Z data;

**int** e;

node \*link;

node(Z a,node \*l=NULL);

node();

};

**template** <**class** **Z**,**int** E>

node<Z,E>::node(Z x,node \*l)

{

data=x;

link=l;

e=E;

}

**template** <**class** **Z**,**int** E>

node<Z,E>::node()

{

link=NULL;

e=E;

}

**int** main()

{

node<string,**5**> \*n=**new** node<string,**5**>("Paul");

cout<<n->data<<' '<<n->e;

**return** **0**;

}

**Output:**

Paul 5

**Question (4)**

**Code:**

#include <iostream>

#include<stdbool.h>

**using** **namespace** std;

**template** <**class** **Z**,**int** E=**0**>

**class** **node**

{

**public:**

Z data;

**int** e;

node \*link;

node(Z a,node \*l=NULL);

node();

};

**template** <**class** **Z**,**int** E>

node<Z,E>::node(Z x,node \*l)

{

data=x;

link=l;

e=E;

}

**template** <**class** **Z**,**int** E>

node<Z,E>::node()

{

link=NULL;

e=E;

}

**template** <**class** **Z**>

**class** **sll**

{

node<Z> head;

node<Z>\* createNewNode(Z,node<Z>\*);

**bool** **isempty**();

**public:**

sll(node<Z> \*l=NULL);

sll(**const** sll&);

~sll();

**void** **deletesll**();

**void** **insertBeg**(Z);

**void** **Delete**(Z);

**bool** **search**(Z);

**void** **display**();

**int** **size**();

};

**template** <**class** **Z**>

sll<Z>::sll(node<Z> \*l)

{

head.e=**0**;

head.link=l;

**if**(l!=NULL)

{

**int** cnt=**1**;

node<Z> \*t=l;

**while**(t->link!=NULL)

{

t=t->link;

cnt++;

}

head.e=cnt;

}

}

**template** <**class** **Z**>

sll<Z>::sll(**const** sll &s)

{

head.e=s.head.e;

head.link=NULL;

node<Z> \*t=s.head.link;

**if**(t!=NULL)

{

insertBeg(t->data);

head.e--;

t=t->link;

node<Z> \*p=head.link;

**for**(**int** i=**1**;i<s.head.e;i++,t=t->link,p=p->link)

p->link=createNewNode(t->data,NULL);

}

}

**template** <**class** **Z**>

**void** sll<Z>::deletesll()

{

node<Z> \*t;

**for**(**int** i=**0**;i<head.e;i++)

{

t=head.link;

head.link=head.link->link;

**delete** t;

}

head.e=**0**;

}

**template** <**class** **Z**>

**bool** sll<Z>::isempty()

{

**return** (head.e==**0**);

}

**template** <**class** **Z**>

**void** sll<Z>::insertBeg(Z x)

{

head.link=createNewNode(x,head.link);

head.e++;

}

**template** <**class** **Z**>

node<Z>\* sll<Z>::createNewNode(Z x,node<Z> \*l)

{

node<Z> \*t=**new** node<Z>(x,l);

**return** t;

}

**template** <**class** **Z**>

sll<Z>::~sll()

{

deletesll();

}

**template** <**class** **Z**>

**void** sll<Z>::Delete(Z x)

{

**if**(isempty())

{

cout<<"List is empty**\n**";

**return**;

}

node<Z> \*p=head.link;

node<Z> \*q;

**if**(p->data==x)

{

head.link=p->link;

**delete** p;

head.e--;

}

**else**

{

**while**(p!=NULL&&p->data!=x)

{

q=p;

p=p->link;

}

**if**(p==NULL)

cout<<"No match :: deletion failed**\n**";

**else**

{

q->link=p->link;

**delete** p;

head.e--;

}

}

}

**template** <**class** **Z**>

**bool** sll<Z>::search(Z x)

{

node<Z> \*t=head.link;

**int** i;

**for**(i=**0**;i<head.e;i++,t=t->link)

**if**(t->data==x)

**return** true;

**return** false;

}

**template** <**class** **Z**>

**void** sll<Z>::display()

{

node<Z> \*t=head.link;

**for**(**int** i=**0**;i<head.e;i++,t=t->link)

{

cout<<t->data<<" --> ";

}

cout<<"||"<<endl;

}

**template** <**class** **Z**>

**int** sll<Z>::size()

{

**return** head.e;

}

**int** main()

{

sll<string> s1;

s1.insertBeg("str1");

s1.insertBeg("str5");

s1.insertBeg("str11");

s1.insertBeg("str15");

s1.insertBeg("str34");

s1.display();

sll<string> s2=s1;

s1.Delete("str11");

s1.display();

s2.display();

cout<<"Enter element to search for in s1: ";

string e;

getline(cin,e);

**if**(s1.search(e))

cout<<"Element found";

**else**

cout<<"Element not found";

**return** **0**;

}

**Output:**

str34 --> str15 --> str11 --> str5 --> str1 --> ||

str34 --> str15 --> str5 --> str1 --> ||

str34 --> str15 --> str11 --> str5 --> str1 --> ||

Enter element to search for in s1: str5

Element found

**Question (5)**

**Contents of MyLinkedListTemplate header file:**

**using** **namespace** std;

**template** <**class** **Z**,**int** E=**0**>

**class** **node**

{

**public:**

Z data;

**int** e;

node \*link;

node(Z a,node \*l=NULL);

node();

};

**template** <**class** **Z**,**int** E>

node<Z,E>::node(Z x,node \*l)

{

data=x;

link=l;

e=E;

}

**template** <**class** **Z**,**int** E>

node<Z,E>::node()

{

link=NULL;

e=E;

}

**template** <**class** **Z**>

**class** **sll**

{

**public:**

node<Z> head;

node<Z>\* createNewNode(Z,node<Z>\*);

**bool** **isempty**();

sll(node<Z> \*l=NULL);

sll(**const** sll&);

~sll();

**void** **deletesll**();

**void** **insertBeg**(Z);

**void** **Delete**(Z);

**bool** **search**(Z);

**void** **display**();

**int** **size**();

};

**template** <**class** **Z**>

sll<Z>::sll(node<Z> \*l)

{

head.e=**0**;

head.link=l;

**if**(l!=NULL)

{

**int** cnt=**1**;

node<Z> \*t=l;

**while**(t->link!=NULL)

{

t=t->link;

cnt++;

}

head.e=cnt;

}

}

**template** <**class** **Z**>

sll<Z>::sll(**const** sll &s)

{

head.e=s.head.e;

head.link=NULL;

node<Z> \*t=s.head.link;

**if**(t!=NULL)

{

insertBeg(t->data);

head.e--;

t=t->link;

node<Z> \*p=head.link;

**for**(**int** i=**1**;i<s.head.e;i++,t=t->link,p=p->link)

p->link=createNewNode(t->data,NULL);

}

}

**template** <**class** **Z**>

**void** sll<Z>::deletesll()

{

node<Z> \*t;

**for**(**int** i=**0**;i<head.e;i++)

{

t=head.link;

head.link=head.link->link;

**delete** t;

}

head.e=**0**;

}

**template** <**class** **Z**>

**bool** sll<Z>::isempty()

{

**return** (head.e==**0**);

}

**template** <**class** **Z**>

**void** sll<Z>::insertBeg(Z x)

{

head.link=createNewNode(x,head.link);

head.e++;

}

**template** <**class** **Z**>

node<Z>\* sll<Z>::createNewNode(Z x,node<Z> \*l)

{

node<Z> \*t=**new** node<Z>(x,l);

**return** t;

}

**template** <**class** **Z**>

sll<Z>::~sll()

{

deletesll();

}

**template** <**class** **Z**>

**void** sll<Z>::Delete(Z x)

{

**if**(isempty())

{

cout<<"List is empty**\n**";

**return**;

}

node<Z> \*p=head.link;

node<Z> \*q;

**if**(p->data==x)

{

head.link=p->link;

**delete** p;

head.e--;

}

**else**

{

**while**(p!=NULL&&p->data!=x)

{

q=p;

p=p->link;

}

**if**(p==NULL)

cout<<"No match :: deletion failed**\n**";

**else**

{

q->link=p->link;

**delete** p;

head.e--;

}

}

}

**template** <**class** **Z**>

**bool** sll<Z>::search(Z x)

{

node<Z> \*t=head.link;

**int** i;

**for**(i=**0**;i<head.e;i++,t=t->link)

**if**(t->data==x)

**return** true;

**return** false;

}

**template** <**class** **Z**>

**void** sll<Z>::display()

{

node<Z> \*t=head.link;

**for**(**int** i=**0**;i<head.e;i++,t=t->link)

{

cout<<t->data<<" --> ";

}

cout<<"||"<<endl;

}

**template** <**class** **Z**>

**int** sll<Z>::size()

{

**return** head.e;

}

**Code:**

#include<iostream>

#include<stdbool.h>

#include<math.h>

#include<fstream>

#include<sstream>

#include"MyLinkedListTemplate.h"

**using** **namespace** std;

**class** **not\_found\_exception**{};

**template** <**class** **Z**,**int** s=**293**>

**class** **hashing**

{

**public:**

sll<Z> \*ht;

**int** htsize;

**int** **hashfn**(**int**);

**int** **hashfn**(string);

hashing(**int** n=s);

hashing(**const** hashing&);

~hashing();

**bool** **Search**(Z);

**void** **Insert**(Z);

**void** **Delete**(Z);

**void** **Display**();

**void** **TableLoadDistr**();

**void** **SearchWithName**(string);

};

**template** <**class** **Z**,**int** s>

hashing<Z,s>::hashing(**const** hashing &h)

{

htsize=h.htsize;

ht=**new** sll<Z>[htsize];

**for**(**int** i=**0**;i<htsize;i++)

{

node<Z> \*t=h.ht[i].head.link;

ht[i].head.e=h.ht[i].head.e;

**if**(t!=NULL)

{

ht[i].head.link=**new** node<Z>(t->data,ht[i].head.link);

t=t->link;

node<Z> \*p=ht[i].head.link;

**for**(**int** j=**1**;j<ht[i].head.e;j++,p=p->link,t=t->link)

p->link=**new** node<Z>(t->data,NULL);

}

}

}

**template** <**class** **Z**,**int** s>

**int** hashing<Z,s>::hashfn(**int** x)

{

**return** (abs(x)%htsize);

}

**template** <**class** **Z**,**int** s>

**int** hashing<Z,s>::hashfn(string s1)

{

string str;

stringstream **sin**(s1);

**for**(**int** i=**1**;i<=**4**;i++)

getline(sin,str,',');

**int** n=str.length();

**int** cnt=**0**;

**int** m=htsize;

**for**(**int** i=**0**;i<n;i++)

{

cnt+=((i%m)\*(**int**(str[i])%htsize));

cnt%=htsize;

}

**return** cnt;

}

**template** <**class** **Z**,**int** s>

hashing<Z,s>::hashing(**int** n)

{

ht=**new** sll<Z>[n];

htsize=n;

}

**template** <**class** **Z**,**int** s>

hashing<Z,s>::~hashing()

{

**for**(**int** i=**0**;i<htsize;i++)

ht[i].deletesll();

**delete** []ht;

}

**template** <**class** **Z**,**int** s>

**bool** hashing<Z,s>::Search(Z x)

{

**int** index=hashfn(x);

**return** (ht[index].search(x));

}

**template** <**class** **Z**,**int** s>

**void** hashing<Z,s>::Insert(Z x)

{

**int** index=hashfn(x);

ht[index].insertBeg(x);

}

**template** <**class** **Z**,**int** s>

**void** hashing<Z,s>::Delete(Z x)

{

**int** index=hashfn(x);

**if**(Search(x))

{

ht[index].Delete(x);

}

**else**

{

**throw** not\_found\_exception();

}

}

**template** <**class** **Z**,**int** s>

**void** hashing<Z,s>::Display()

{

**for**(**int** i=**0**;i<htsize;i++)

ht[i].display();

}

**template** <**class** **Z**,**int** s>

**void** hashing<Z,s>::TableLoadDistr()

{

**int** totalElements=**0**;

**for**(**int** i=**0**;i<htsize;i++)

totalElements+=ht[i].size();

**double** mean=**double**(totalElements)/htsize, standardDev;

cout<<"Mean load in each bucket = "<<mean<<endl;

**for**(**int** i=**0**;i<htsize;i++)

standardDev+=((ht[i].size()-mean)\*(ht[i].size()-mean));

standardDev/=htsize;

standardDev=sqrt(standardDev);

cout<<"STD of the load = "<<standardDev<<endl;

}

**template** <**class** **Z**,**int** s>

**void** hashing<Z,s>::SearchWithName(string name)

{

**int** n=name.length();

**int** cnt=**0**;

**int** m=htsize;

**bool** found=false;

**for**(**int** i=**0**;i<n;i++)

{

cnt+=((i%m)\*(**int**(name[i])%htsize));

cnt%=htsize;

}

string str, sname;

**struct** node<Z> \*nd=ht[cnt].head.link;

**for**(**int** i=**0**;i<ht[cnt].size();i++,nd=nd->link)

{

str=nd->data;

stringstream **sin**(str);

**for**(**int** j=**1**;j<=**4**;j++)

getline(sin,sname,',');

**if**(name==sname)

{

cout<<"Record found. Record is:"<<endl;

cout<<str<<endl;

found=true;

**break**;

}

}

**if**(!found)

{

cout<<"Record not found."<<endl;

}

}

**int** main()

{

hashing<string> hstr;

ifstream fin;

fin.open("TENTATIVE 2nd Semester Roll Sheet 2020-2021.csv");

string line,word;

getline(fin,line);

getline(fin,line);

getline(fin,line);

**while**(getline(fin,line))

{

hstr.Insert(line);

}

hstr.TableLoadDistr();

fin.close();

string name;

cout<<"Enter name whose record is to be displayed: ";

getline(cin,name);

hstr.SearchWithName(name);

**return** **0**;

}

**Output:**

Mean load in each bucket = 3.06143

STD of the load = 1.80528

Enter name whose record is to be displayed: JHANSI KRISHNA KARU

Record found. Record is:

B.Tech-EE,20U10447 ,20G80045,JHANSI KRISHNA KARU,Male,9912795544,ramunaidu2001@gmail.com,jkk.20U10447@btech.nitdgp.ac.in