Name: Swarnabh Paul

Section: Y

Roll no: 19CS8122

Assignment no: 6

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

**Question (1)**

**Code:**

#include<iostream>

#include<stdbool.h>

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

**class** **Complex**

{

**float** real, img;

**public:**

Complex(**float**,**float**);

Complex **operator** + (**const** Complex&);

Complex **operator** - (**const** Complex&);

Complex **operator** \* (**const** Complex&);

Complex **operator** / (**const** Complex&);

Complex **operator** ! ();

**bool** **operator** == (**const** Complex&);

**bool** **operator** != (**const** Complex&);

**void** **operator** = (**const** Complex&);

**float** **operator** [] (**int**);

**friend** ostream& **operator** << (ostream&,**const** Complex&);

**friend** istream& **operator** >> (istream&,Complex&);

};

Complex::Complex(**float** r=**0**,**float** i=**0**)

{

real=r;

img=i;

}

Complex Complex::**operator** + (**const** Complex &z)

{

**return** Complex(real+z.real,img+z.img);

}

Complex Complex::**operator** - (**const** Complex &z)

{

**return** Complex(real-z.real,img-z.img);

}

Complex Complex::**operator** \* (**const** Complex &z)

{

**return** Complex(real\*z.real-img\*z.img,real\*z.img+img\*z.real);

}

Complex Complex::**operator** / (**const** Complex &z)

{

**float** d=z.real\*z.real+z.img\*z.img;

**return** **Complex**((real\*z.real+img\*z.img)/d,(img\*z.real-real\*z.img)/d);

}

Complex Complex::**operator** ! ()

{

**return** Complex(real,-img);

}

**bool** Complex::**operator** == (**const** Complex &z)

{

**return** ((real==z.real)&&(img==z.img));

}

**bool** Complex::**operator** != (**const** Complex &z)

{

**return** ((real!=z.real)||(img!=z.img));

}

**void** Complex::**operator** = (**const** Complex &z)

{

real=z.real;

img=z.img;

}

**float** Complex::**operator** [] (**int** index)

{

**if**(index==**0**)

**return** real;

**return** img;

}

ostream& **operator** << (ostream &f,**const** Complex &z)

{

f<<z.real;

**if**(z.img>=**0**)

f<<'+';

f<<z.img<<'j';

**return** f;

}

istream& **operator** >> (istream &f,Complex &z)

{

f>>z.real>>z.img;

**return** f;

}

**int** main()

{

Complex z1, z2, z3, z4;

cout<<"Input z1 and z2 respectively:**\n**";

cin>>z1>>z2;

cout<<z1+z2<<endl<<z1-z2<<endl<<z1\*z2<<endl<<z1/z2<<endl;

cout<<!z1<<endl<<!z2<<endl;

**if**(z1==z2)

cout<<"They are equal.**\n**";

**if**(z1!=z2)

cout<<"They are not equal.**\n**";

cout<<"Input z3: ";

cin>>z3;

z4=z3;

cout<<z3<<endl<<z4<<endl;

cout<<"Re(z3): "<<z3[**0**]<<endl;

cout<<"Im(z3): "<<z3[**1**]<<endl;

**return** **0**;

}

**Output:**

Input z1 and z2 respectively:

5 7

2 -1

7+6j

3+8j

17+9j

0.6+3.8j

5-7j

2+1j

They are not equal.

Input z3: 4 6

4+6j

4+6j

Re(z3): 4

Im(z3): 6

**Question (2)**

**Code:**

#include<iostream>

#include<algorithm>

#include<stdbool.h>

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

**class** **Fraction**

{

**int** a, b; /\* Fraction is represented by a/b, where a is numerator and b is denominator \*/

**public:**

Fraction(**int**,**int**);

Fraction **operator** + (**const** Fraction&);

Fraction **operator** - (**const** Fraction&);

Fraction **operator** \* (**const** Fraction&);

Fraction **operator** / (**const** Fraction&);

**void** **operator** \* ();

**bool** **operator** == (**const** Fraction&);

**bool** **operator** != (**const** Fraction&);

**bool** **operator** < (**const** Fraction&);

**bool** **operator** > (**const** Fraction&);

**void** **operator** = (**const** Fraction&);

**int** **operator** [] (**int**);

**friend** ostream& **operator** << (ostream&,**const** Fraction&);

**friend** istream& **operator** >> (istream&,Fraction&);

};

Fraction::Fraction(**int** n=**0**,**int** d=**1**)

{

a=n;

b=d;

}

Fraction Fraction::**operator** + (**const** Fraction &z)

{

**int** lcm=abs(b\*(z.b/\_\_gcd(abs(b),abs(z.b)))), n=a\*(lcm/b)+z.a\*(lcm/z.b);

**int** g=\_\_gcd(abs(n),abs(lcm));

**return** **Fraction**(n/g,lcm/g);

}

Fraction Fraction::**operator** - (**const** Fraction &z)

{

**int** lcm=abs(b\*(z.b/\_\_gcd(abs(b),abs(z.b)))), n=a\*(lcm/b)-z.a\*(lcm/z.b);

**int** g=\_\_gcd(abs(n),abs(lcm));

**return** **Fraction**(n/g,lcm/g);

}

Fraction Fraction::**operator** \* (**const** Fraction &z)

{

**int** n=a\*z.a, d=b\*z.b, g=\_\_gcd(abs(n),abs(d));

**if**(d<**0**)

{

n=-n;

d=-d;

}

**return** Fraction(n/g,d/g);

}

Fraction Fraction::**operator** / (**const** Fraction &z)

{

**int** n=a\*z.b, d=b\*z.a, g=\_\_gcd(abs(n),abs(d));

**if**(d<**0**)

{

n=-n;

d=-d;

}

**return** Fraction(n/g,d/g);

}

**void** Fraction::**operator** \* ()

{

**int** g=\_\_gcd(abs(a),abs(b));

a/=g;

b/=g;

**if**(b<**0**)

{

a=-a;

b=-b;

}

}

**bool** Fraction::**operator** == (**const** Fraction &z)

{

**int** lcm=abs(b\*(z.b/\_\_gcd(abs(b),abs(z.b))));

**return** ((a\*(lcm/b))==(z.a\*(lcm/z.b)));

}

**bool** Fraction::**operator** != (**const** Fraction &z)

{

**int** lcm=abs(b\*(z.b/\_\_gcd(abs(b),abs(z.b))));

**return** ((a\*(lcm/b))!=(z.a\*(lcm/z.b)));

}

**bool** Fraction::**operator** < (**const** Fraction &z)

{

**int** lcm=abs(b\*(z.b/\_\_gcd(abs(b),abs(z.b))));

**return** ((a\*(lcm/b))<(z.a\*(lcm/z.b)));

}

**bool** Fraction::**operator** > (**const** Fraction &z)

{

**int** lcm=abs(b\*(z.b/\_\_gcd(abs(b),abs(z.b))));

**return** ((a\*(lcm/b))>(z.a\*(lcm/z.b)));

}

**void** Fraction::**operator** = (**const** Fraction &z)

{

a=z.a;

b=z.b;

}

**int** Fraction::**operator** [] (**int** index)

{

**if**(index==**0**)

**return** a;

**return** b;

}

ostream& **operator** << (ostream &f,**const** Fraction &z)

{

f<<z.a<<'/'<<z.b;

**return** f;

}

istream& **operator** >> (istream &f,Fraction &z)

{

f>>z.a>>z.b;

**return** f;

}

**int** main()

{

Fraction f1, f2, f3, f4;

cout<<"Input f1 and f2 respectively:**\n**";

cin>>f1>>f2;

cout<<f1+f2<<endl<<f1-f2<<endl<<f1\*f2<<endl<<f1/f2<<endl;

cout<<"Input f3 and f4 respectively:**\n**";

cin>>f3>>f4;

**if**(f3==f4)

cout<<"They are equal.**\n**";

**if**(f3!=f4)

cout<<"They are not equal.**\n**";

**if**(f3<f4)

cout<<"f3 is lesser.**\n**";

**if**(f3>f4)

cout<<"f3 is greater.**\n**";

\*f3;

\*f4;

cout<<f3<<endl<<f4<<endl;

f4=f3;

cout<<f3<<endl<<f4<<endl;

cout<<"Numerator(f3): "<<f3[**0**]<<endl<<"Denominator(f3): "<<f3[**1**]<<endl;

**return** **0**;

}

**Output:**

Input f1 and f2 respectively:

3 4

4 5

31/20

-1/20

3/5

15/16

Input f3 and f4 respectively:

6 10

4 8

They are not equal.

f3 is greater.

3/5

1/2

3/5

3/5

Numerator(f3): 3

Denominator(f3): 5

**Question (3)**

**Code:**

#include<iostream>

#include<stdbool.h>

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

**class** **Matrix**

{

**int** \*\*a;

**int** r, c, maxr, maxc;

**int**\*\* **createMatrix**(**int**,**int**);

**public:**

Matrix(**int**);

Matrix(**int**,**int**);

Matrix(**const** Matrix&);

~Matrix();

Matrix **operator** + (**const** Matrix&);

Matrix **operator** - (**const** Matrix&);

Matrix **operator** \* (**const** Matrix&);

Matrix **operator** / (**int**);

**void** **operator** ! ();

**bool** **operator** == (**const** Matrix&);

**void** **operator** = (**const** Matrix&);

**int**\* **operator** [] (**int**);

**friend** ostream& **operator** << (ostream&,**const** Matrix&);

**friend** istream& **operator** >> (istream&,Matrix&);

**void**\* **operator** **new** (**size\_t**);

**void**\* **operator** **new** [] (**size\_t**);

**void** **operator** **delete** (**void**\*);

**void** **operator** **delete** [] (**void**\*);

};

**int**\*\* Matrix::createMatrix(**int** rows,**int** cols)

{

**int** \*\*res;

res=**new** **int**\*[rows];

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

res[i]=**new** **int**[cols];

**return** res;

}

Matrix::Matrix(**int** rows,**int** cols)

{

r=c=**0**;

maxr=rows;

maxc=cols;

a=createMatrix(rows,cols);

}

Matrix::Matrix(**int** Size=**10**)

{

r=c=**0**;

maxr=maxc=Size;

a=createMatrix(Size,Size);

}

Matrix::Matrix(**const** Matrix &m)

{

r=m.r;

c=m.c;

maxr=m.maxr;

maxc=m.maxc;

a=createMatrix(maxr,maxc);

**int** i, j;

**for**(i=**0**;i<r;i++)

**for**(j=**0**;j<c;j++)

a[i][j]=m.a[i][j];

}

Matrix::~Matrix()

{

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

**delete** []a[i];

**delete** []a;

}

Matrix Matrix::**operator** + (**const** Matrix &m)

{

Matrix result(r,c);

**if**((r!=m.r)||(c!=m.c))

{

cout<<"Invalid operation"<<endl;

**return** result;

}

result.r=r;

result.c=c;

**int** i,j;

**for**(i=**0**;i<r;i++)

**for**(j=**0**;j<c;j++)

result.a[i][j]=a[i][j]+m.a[i][j];

**return** result;

}

Matrix Matrix::**operator** - (**const** Matrix &m)

{

Matrix result(r,c);

**if**((r!=m.r)||(c!=m.c))

{

cout<<"Invalid operation"<<endl;

**return** result;

}

result.r=r;

result.c=c;

**int** i,j;

**for**(i=**0**;i<r;i++)

**for**(j=**0**;j<c;j++)

result.a[i][j]=a[i][j]-m.a[i][j];

**return** result;

}

Matrix Matrix::**operator** \* (**const** Matrix &m)

{

Matrix result(r,m.c);

**if**(c!=m.r)

{

cout<<"Invalid operation"<<endl;

**return** result;

}

result.r=r;

result.c=m.c;

**int** i, j, k;

**for**(i=**0**;i<r;i++)

**for**(j=**0**;j<m.c;j++)

{

result.a[i][j]=**0**;

**for**(k=**0**;k<c;k++)

result.a[i][j]+=(a[i][k]\*m.a[k][j]);

}

**return** result;

}

Matrix Matrix::**operator** / (**int** d)

{

Matrix result(r,c);

**if**(d==**0**)

{

cout<<"Invalid operation"<<endl;

**return** result;

}

result.r=r;

result.c=c;

**int** i,j;

**for**(i=**0**;i<r;i++)

**for**(j=**0**;j<c;j++)

result.a[i][j]=a[i][j]/d;

**return** result;

}

**void** Matrix::**operator** ! ()

{

**int** i, j, temp;

Matrix **t**(c,r);

**if**((r>maxc)||(c>maxr))

{

cout<<"Invalid operation"<<endl;

**return**;

}

**for**(i=**0**;i<c;i++)

**for**(j=**0**;j<r;j++)

{

t.a[i][j]=a[j][i];

}

temp=r;

r=c;

c=temp;

**for**(i=**0**;i<r;i++)

**for**(j=**0**;j<c;j++)

a[i][j]=t.a[i][j];

}

**bool** Matrix::**operator** == (**const** Matrix &m)

{

**int** i, j;

**if**((r!=m.r)||(c!=m.c))

**return** false;

**for**(i=**0**;i<r;i++)

**for**(j=**0**;j<c;j++)

**if**(a[i][j]!=m.a[i][j])

**return** false;

**return** true;

}

**void** Matrix::**operator** = (**const** Matrix &m)

{

**if**((m.r>maxr)||(m.c>maxc))

{

cout<<"Invalid operation"<<endl;

**return**;

}

r=m.r;

c=m.c;

**int** i, j;

**for**(i=**0**;i<r;i++)

**for**(j=**0**;j<c;j++)

a[i][j]=m.a[i][j];

}

**int**\* Matrix::**operator** [] (**int** index)

{

**if**(index<**0**||index>=r)

{

cout<<"Out of bound access**\n**";

**return** NULL;

}

**return** a[index];

}

ostream& **operator** << (ostream &f,**const** Matrix &m)

{

**int** i, j;

**for**(i=**0**;i<m.r;i++)

{

**for**(j=**0**;j<m.c;j++)

f<<m.a[i][j]<<' ';

f<<endl;

}

**return** f;

}

istream& **operator** >> (istream &f,Matrix &m)

{

cout<<"Enter order of matrix: ";

f>>m.r>>m.c;

**while**((m.r>m.maxr)||(m.c>m.maxc)||(m.r<**0**)||(m.c<**0**))

{

cout<<"Invalid input"<<endl;

cout<<"Enter order of matrix: ";

f>>m.r>>m.c;

}

cout<<"Enter matrix:**\n**";

**int** i, j;

**for**(i=**0**;i<m.r;i++)

**for**(j=**0**;j<m.c;j++)

f>>m.a[i][j];

**return** f;

}

**void**\* Matrix::**operator** **new** (**size\_t** size)

{

cout<<"Overloaded new with size "<<size<<endl;

**void** \*p=NULL;

p=malloc(size);

**return** p;

}

**void**\* Matrix::**operator** **new** [] (**size\_t** size)

{

cout<<"Overloaded new [] with size "<<size<<endl;

**void** \*p=NULL;

p=malloc(size);

**return** p;

}

**void** Matrix::**operator** **delete** (**void** \*p)

{

cout<<"Overloaded delete"<<endl;

free(p);

}

**void** Matrix::**operator** **delete** [] (**void** \*p)

{

cout<<"Overloaded delete []"<<endl;

free(p);

}

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

{

Matrix m1, m2;

cout<<"Enter m1 and m2:**\n**";

cin>>m1>>m2;

cout<<"**\n**Displaying matrices:**\n**";

cout<<m1<<endl<<m2<<endl<<m1+m2<<endl<<m1-m2<<endl;

**if**(m1==m2)

cout<<"m1 and m2 are equal**\n**";

Matrix m3, m4;

cout<<"Enter m3 and m4:**\n**";

cin>>m3>>m4;

cout<<"**\n**Displaying matrices:**\n**";

cout<<m3\*m4<<endl;

!m3;

cout<<m3<<endl;

**int** i, j;

cout<<"Enter indices of element to display in m3: ";

cin>>i>>j;

**if**(m3[i]!=NULL)

cout<<"Element is: "<<m3[i][j]<<endl;

Matrix m5(m3);

m4=m5;

cout<<"**\n**Displaying matrices:**\n**";

cout<<m4<<endl<<m5<<endl;

**int** n;

cout<<"Enter number to divide m5 by: ";

cin>>n;

cout<<"**\n**Displaying matrices:**\n**";

cout<<m5/n<<endl;

Matrix \*pm1, \*pm2;

pm1=**new** Matrix(m1);

pm2=**new** Matrix[**3**];

**for**(i=**0**;i<**3**;i++)

pm2[i]=m2;

cout<<"**\n**Displaying matrices:**\n**";

cout<<(\*pm1)<<endl;

**for**(i=**0**;i<**3**;i++)

cout<<pm2[i]<<endl;

**delete** []pm2;

**delete** pm1;

**return** **0**;

}

**Output:**

Enter m1 and m2:

Enter order of matrix: 2 2

Enter matrix:

1 2

3 4

Enter order of matrix: 2 2

Enter matrix:

-9 1

5 7

Displaying matrices:

1 2

3 4

-9 1

5 7

-8 3

8 11

10 1

-2 -3

Enter m3 and m4:

Enter order of matrix: 2 3

Enter matrix:

1 2 3

4 5 6

Enter order of matrix: 3 2

Enter matrix:

10 11

20 21

30 31

Displaying matrices:

140 146

320 335

1 4

2 5

3 6

Enter indices of element to display in m3: 2 0

Element is: 3

Displaying matrices:

1 4

2 5

3 6

1 4

2 5

3 6

Enter number to divide m5 by: 3

Displaying matrices:

0 1

0 1

1 2

Overloaded new with size 20

Overloaded new [] with size 64

Displaying matrices:

1 2

3 4

-9 1

5 7

-9 1

5 7

-9 1

5 7

Overloaded delete []

Overloaded delete

**Question (4)**

**Contents of Linked List header file:**

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

**class** **node**

{

**public:**

**int** data;

node \*link;

node(**int**,node\*);

};

node::node(**int** x=**0**,node \*l=NULL)

{

data=x;

link=l;

}

**class** **sll**

{

**public:**

node head;

node\* **createNewNode**(**int**,node\*);

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

sll(node \*l);

sll(**const** sll&);

~sll();

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

**void** **insertBeg**(**int**);

**void** **Delete**(**int**);

**bool** search(**int**) **const**;

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

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

};

sll::sll(node \*l=NULL)

{

head.data=**0**;

head.link=l;

**if**(l!=NULL)

{

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

node \*t=l;

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

{

t=t->link;

cnt++;

}

head.data=cnt;

}

}

sll::sll(**const** sll &s)

{

head.data=s.head.data;

head.link=NULL;

node \*t=s.head.link;

**if**(t!=NULL)

{

insertBeg(t->data);

head.data--;

t=t->link;

node \*p=head.link;

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

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

}

}

**void** sll::deletesll()

{

node \*t;

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

{

t=head.link;

head.link=head.link->link;

**delete** t;

}

head.data=**0**;

}

**bool** sll::isempty()

{

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

}

**void** sll::insertBeg(**int** x)

{

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

head.data++;

}

node\* sll::createNewNode(**int** x,node \*l)

{

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

**return** t;

}

sll::~sll()

{

deletesll();

}

**void** sll::Delete(**int** x)

{

**if**(isempty())

{

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

**return**;

}

node \*p=head.link;

node \*q;

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

{

head.link=p->link;

**delete** p;

head.data--;

}

**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.data--;

}

}

}

**bool** sll::search(**int** x) **const**

{

node \*t=head.link;

**int** i;

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

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

**return** true;

**return** false;

}

**void** sll::display() **const**

{

node \*t=head.link;

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

{

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

}

cout<<"||"<<endl;

}

**int** sll::size()

{

**return** head.data;

}

**Contents of Hash header file:**

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

**class** **hashing**

{

**public:**

sll \*ht;

**int** htsize;

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

hashing(**int**);

hashing(**const** hashing&);

~hashing();

**bool** Search(**int**) **const**;

**void** **Insert**(**int**);

**void** **Delete**(**int**);

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

};

hashing::hashing(**const** hashing &h)

{

htsize=h.htsize;

ht=**new** sll[htsize];

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

{

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

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

**if**(t!=NULL)

{

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

t=t->link;

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

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

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

}

}

}

**int** hashing::hashfn(**int** x) **const**

{

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

}

hashing::hashing(**int** n=**10**)

{

ht=**new** sll[n];

htsize=n;

}

hashing::~hashing()

{

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

ht[i].deletesll();

**delete** []ht;

}

**bool** hashing::Search(**int** x) **const**

{

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

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

}

**void** hashing::Insert(**int** x)

{

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

ht[index].insertBeg(x);

}

**void** hashing::Delete(**int** x)

{

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

**if**(Search(x))

{

ht[index].Delete(x);

}

**else**

cout<<"Element not found. Deletion not possible."<<endl;

}

**void** hashing::Display() **const**

{

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

ht[i].display();

}

**Code:**

#include<iostream>

#include<stdbool.h>

#include"MyLinkedList.h"

#include"MyHash.h"

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

**class** **Set**

{

hashing s;

**public:**

Set(**int**);

Set(**const** Set&);

~Set();

Set **operator** + (**const** Set&);

Set **operator** - (**const** Set&);

Set **operator** \* (**const** Set&);

**bool** **operator** < (**const** Set&);

**bool** **operator** <= (**const** Set&);

**bool** **operator** > (**const** Set&);

**bool** **operator** >= (**const** Set&);

**bool** **operator** == (**const** Set&);

**bool** **operator** != (**const** Set&);

**void** **operator** = (**const** Set&);

**friend** ostream& **operator** << (ostream&,**const** Set&);

**friend** istream& **operator** >> (istream&,Set&);

};

Set::Set(**int** n=**10**): s(n)

{

}

Set::Set(**const** Set &t): s(t.s)

{

}

Set::~Set()

{

}

Set Set::**operator** + (**const** Set &t)

{

Set res;

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

{

node \*t1=s.ht[i].head.link;

**for**(**int** j=**0**;j<s.ht[i].head.data;j++,t1=t1->link)

res.s.Insert(t1->data);

}

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

{

node \*t1=t.s.ht[i].head.link;

**for**(**int** j=**0**;j<t.s.ht[i].head.data;j++,t1=t1->link)

**if**(!res.s.Search(t1->data))

res.s.Insert(t1->data);

}

**return** res;

}

Set Set::**operator** - (**const** Set &t)

{

Set res;

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

{

node \*t1=s.ht[i].head.link;

**for**(**int** j=**0**;j<s.ht[i].head.data;j++,t1=t1->link)

**if**(!t.s.Search(t1->data))

res.s.Insert(t1->data);

}

**return** res;

}

Set Set::**operator** \* (**const** Set &t)

{

Set res;

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

{

node \*t1=s.ht[i].head.link;

**for**(**int** j=**0**;j<s.ht[i].head.data;j++,t1=t1->link)

**if**(t.s.Search(t1->data))

res.s.Insert(t1->data);

}

**return** res;

}

**bool** Set::**operator** < (**const** Set &t)

{

**int** size1=**0**, size2=**0**;

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

size1+=s.ht[i].head.data;

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

size2+=t.s.ht[i].head.data;

**if**(size1>=size2)

**return** false;

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

{

node \*t1=s.ht[i].head.link;

**for**(**int** j=**0**;j<s.ht[i].head.data;j++,t1=t1->link)

**if**(!t.s.Search(t1->data))

**return** false;

}

**return** true;

}

**bool** Set::**operator** <= (**const** Set &t)

{

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

{

node \*t1=s.ht[i].head.link;

**for**(**int** j=**0**;j<s.ht[i].head.data;j++,t1=t1->link)

**if**(!t.s.Search(t1->data))

**return** false;

}

**return** true;

}

**bool** Set::**operator** > (**const** Set &t)

{

**int** size1=**0**, size2=**0**;

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

size1+=s.ht[i].head.data;

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

size2+=t.s.ht[i].head.data;

**if**(size1<=size2)

**return** false;

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

{

node \*t1=t.s.ht[i].head.link;

**for**(**int** j=**0**;j<t.s.ht[i].head.data;j++,t1=t1->link)

**if**(!s.Search(t1->data))

**return** false;

}

**return** true;

}

**bool** Set::**operator** >= (**const** Set &t)

{

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

{

node \*t1=t.s.ht[i].head.link;

**for**(**int** j=**0**;j<t.s.ht[i].head.data;j++,t1=t1->link)

**if**(!s.Search(t1->data))

**return** false;

}

**return** true;

}

**bool** Set::**operator** == (**const** Set &t)

{

**int** size1=**0**, size2=**0**;

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

size1+=s.ht[i].head.data;

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

size2+=t.s.ht[i].head.data;

**if**(size1!=size2)

**return** false;

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

{

node \*t1=s.ht[i].head.link;

**for**(**int** j=**0**;j<s.ht[i].head.data;j++,t1=t1->link)

**if**(!t.s.Search(t1->data))

**return** false;

}

**return** true;

}

**bool** Set::**operator** != (**const** Set &t)

{

**return** !((\***this**)==t);

}

**void** Set::**operator** = (**const** Set &t)

{

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

s.ht[i].deletesll();

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

{

node \*t1=t.s.ht[i].head.link;

**for**(**int** j=**0**;j<t.s.ht[i].head.data;j++,t1=t1->link)

s.Insert(t1->data);

}

}

ostream& **operator** << (ostream &f,**const** Set &t)

{

f<<"{ ";

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

{

node \*t1=t.s.ht[i].head.link;

**for**(**int** j=**0**;j<t.s.ht[i].head.data;j++,t1=t1->link)

f<<t1->data<<" ";

}

f<<"}";

**return** f;

}

istream& **operator** >> (istream &f,Set &t)

{

**int** x;

f>>x;

**if**(!t.s.Search(x))

t.s.Insert(x);

**return** f;

}

**int** main()

{

**int** n;

Set s1, s2;

cout<<"For s1:**\n**";

cout<<"Input no: of elements: ";

cin>>n;

cout<<"Input elements: ";

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

cin>>s1;

cout<<"For s2:**\n**";

cout<<"Input no: of elements: ";

cin>>n;

cout<<"Input elements: ";

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

cin>>s2;

Set **s3**(s1);

cout<<"Displaying sets:**\n**";

cout<<s1<<endl<<s2<<endl<<s3<<endl<<s1+s2<<endl<<s1-s2<<endl<<s1\*s2<<endl;

s3=s1+s2;

cout<<s3<<endl;

Set s4, s5;

cout<<"For s4:**\n**";

cout<<"Input no: of elements: ";

cin>>n;

cout<<"Input elements: ";

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

cin>>s4;

cout<<"For s5:**\n**";

cout<<"Input no: of elements: ";

cin>>n;

cout<<"Input elements: ";

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

cin>>s5;

cout<<"Displaying sets:**\n**";

cout<<s4<<endl<<s5<<endl;

**if**(s4<s5)

cout<<"s4 is a proper subset of s5"<<endl;

**if**(s4<=s5)

cout<<"s4 is a subset of s5"<<endl;

**if**(s4>s5)

cout<<"s4 is a proper superset of s5"<<endl;

**if**(s4>=s5)

cout<<"s4 is a superset of s5"<<endl;

**if**(s4==s5)

cout<<"They are equal"<<endl;

**if**(s4!=s5)

cout<<"They are not equal"<<endl;

**return** **0**;

}

**Output:**

For s1:

Input no: of elements: 7

Input elements: 2 3 4 5 6 7 8

For s2:

Input no: of elements: 5

Input elements: 6 7 8 9 0

Displaying sets:

{ 2 3 4 5 6 7 8 }

{ 0 6 7 8 9 }

{ 2 3 4 5 6 7 8 }

{ 0 2 3 4 5 6 7 8 9 }

{ 2 3 4 5 }

{ 6 7 8 }

{ 0 2 3 4 5 6 7 8 9 }

For s4:

Input no: of elements: 8

Input elements: 3 4 5 6 7 8 9 0

For s5:

Input no: of elements: 6

Input elements: 3 4 5 6 7 8

Displaying sets:

{ 0 3 4 5 6 7 8 9 }

{ 3 4 5 6 7 8 }

s4 is a proper superset of s5

s4 is a superset of s5

They are not equal

**Question (5)**

**Code:**

#include<iostream>

#include<stdbool.h>

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

**class** **node**

{

**public:**

**int** data;

node \*link;

node(**int**,node\*);

};

node::node(**int** x=**0**,node \*l=NULL)

{

data=x;

link=l;

}

**class** **sll**

{

node head;

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

**public:**

sll(node\*);

sll(**const** sll&);

~sll();

sll **operator** + (**const** sll&);

**void** **operator** ! ();

**bool** **operator** == (**const** sll&);

**void** **operator** = (**const** sll&);

node\* **operator** [] (**int**);

**friend** ostream& **operator** << (ostream&,**const** sll&);

**friend** istream& **operator** >> (istream&,sll&);

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

**void**\* **operator** **new** (**size\_t**);

**void**\* **operator** **new** [] (**size\_t**);

**void** **operator** **delete** (**void**\*);

**void** **operator** **delete** [] (**void**\*);

};

sll::sll(node \*l=NULL)

{

head.data=**0**;

head.link=l;

**if**(l!=NULL)

{

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

node \*t=l;

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

{

t=t->link;

cnt++;

}

head.data=cnt;

}

}

sll::sll(**const** sll &s)

{

head.data=s.head.data;

head.link=NULL;

node \*t=s.head.link;

**if**(t!=NULL)

{

head.link=**new** node(t->data,head.link);

t=t->link;

node \*p=head.link;

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

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

}

}

**void** sll::deletesll()

{

node \*t;

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

{

t=head.link;

head.link=head.link->link;

**delete** t;

}

head.data=**0**;

}

sll::~sll()

{

deletesll();

}

sll sll::**operator** + (**const** sll &s)

{

sll res;

res.head.data=head.data+s.head.data;

node \*p=NULL, \*t=NULL;

**if**(head.data>**0**)

{

res.head.link=**new** node(head.link->data,res.head.link);

p=res.head.link;

t=head.link->link;

}

**for**(**int** i=**1**;i<head.data;i++)

{

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

t=t->link;

p=p->link;

}

**int** i=**0**;

**if**(head.data==**0**&&s.head.data>**0**)

{

res.head.link=**new** node(s.head.link->data,res.head.link);

p=res.head.link;

t=s.head.link->link;

i++;

}

**else**

t=s.head.link;

**for**(;i<s.head.data;i++)

{

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

t=t->link;

p=p->link;

}

**return** res;

}

**void** sll::**operator** ! ()

{

**if**(head.data==**0**||head.data==**1**)

{

**return**;

}

node \*prevnode=head.link, \*curnode=head.link->link;

head.link=head.link->link;

prevnode->link=NULL;

**for**(**int** i=**1**;i<head.data;i++)

{

head.link=head.link->link;

curnode->link=prevnode;

prevnode=curnode;

curnode=head.link;

}

head.link=prevnode;

}

**bool** sll::**operator** == (**const** sll &s)

{

**if**(head.data!=s.head.data)

**return** false;

node \*t=head.link;

node \*p=s.head.link;

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

**if**(t->data!=p->data)

**return** false;

**return** true;

}

**void** sll::**operator** = (**const** sll &s)

{

deletesll();

head.data=s.head.data;

node \*t=s.head.link;

**if**(t!=NULL)

{

head.link=**new** node(t->data,head.link);

t=t->link;

node \*p=head.link;

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

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

}

}

node\* sll::**operator** [] (**int** index)

{

**if**(index<**0**||index>=head.data)

{

cout<<"Index out of range"<<endl;

**return** NULL;

}

node \*t=head.link;

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

t=t->link;

**return** t;

}

ostream& **operator** << (ostream &f,**const** sll &s)

{

node \*t=s.head.link;

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

{

f<<t->data<<" --> ";

}

f<<"||";

**return** f;

}

istream& **operator** >> (istream &f,sll &s)

{

**int** x;

f>>x;

s.head.link=**new** node(x,s.head.link);

s.head.data++;

**return** f;

}

**void**\* sll::**operator** **new** (**size\_t** size)

{

cout<<"Overloaded new with size "<<size<<endl;

**void** \*p=NULL;

p=malloc(size);

**return** p;

}

**void**\* sll::**operator** **new** [] (**size\_t** size)

{

cout<<"Overloaded new [] with size "<<size<<endl;

**void** \*p=NULL;

p=malloc(size);

**return** p;

}

**void** sll::**operator** **delete** (**void** \*p)

{

cout<<"Overloaded delete"<<endl;

free(p);

}

**void** sll::**operator** **delete** [] (**void** \*p)

{

cout<<"Overloaded delete []"<<endl;

free(p);

}

**int** main()

{

sll l1, l2;

cout<<"Input list l1: ";

**for**(**int** i=**0**;i<**5**;i++)

cin>>l1;

cout<<"Input list l2: ";

**for**(**int** i=**0**;i<**5**;i++)

cin>>l2;

sll l3;

l3=l1;

sll **l4**(l2);

**if**(l1==l2)

cout<<"They are equal"<<endl;

cout<<"Displaying lists:"<<endl;

cout<<l1<<endl<<l2<<endl<<l3<<endl<<l4<<endl<<l1+l2<<endl;

!l3;

cout<<l3<<endl;

**int** i;

cout<<"Enter index for l3: ";

cin>>i;

node \*t=l3[i];

**if**(t!=NULL)

{

cout<<"Value at index "<<i<<" is: "<<t->data<<endl;

}

sll \*lp;

lp=**new** sll(l1);

sll \*lp2;

lp2=**new** sll[**3**];

**for**(**int** i=**0**;i<**3**;i++)

lp2[i]=l2;

cout<<"Displaying lists:"<<endl;

cout<<(\*lp)<<endl;

**for**(**int** i=**0**;i<**3**;i++)

cout<<lp2[i]<<endl;

**delete** []lp2;

**delete** lp;

**return** **0**;

}

**Output:**

Input list l1: 1 2 3 4 5

Input list l2: 6 7 8 9 10

Displaying lists:

5 --> 4 --> 3 --> 2 --> 1 --> ||

10 --> 9 --> 8 --> 7 --> 6 --> ||

5 --> 4 --> 3 --> 2 --> 1 --> ||

10 --> 9 --> 8 --> 7 --> 6 --> ||

5 --> 4 --> 3 --> 2 --> 1 --> 10 --> 9 --> 8 --> 7 --> 6 --> ||

1 --> 2 --> 3 --> 4 --> 5 --> ||

Enter index for l3: 3

Value at index 3 is: 4

Overloaded new with size 8

Overloaded new [] with size 28

Displaying lists:

5 --> 4 --> 3 --> 2 --> 1 --> ||

10 --> 9 --> 8 --> 7 --> 6 --> ||

10 --> 9 --> 8 --> 7 --> 6 --> ||

10 --> 9 --> 8 --> 7 --> 6 --> ||

Overloaded delete []

Overloaded delete