1. Complex Problem

```
#include<iostream>
using namespace std;
class Complex
	{
	private:
		double _real;
		double _imag;
	public:
		Complex():_real(0.0),_imag(0.0)
		{
		}
		Complex(double real,double imag):_real(real),_imag(imag)
		{
		}
		Complex  operator+(Complex & par)
		{
			Complex res;
			res._real=this->_real+par._real;
			res._imag=this->_imag+par._imag;
			return res;
		}
		Complex  operator+(int  par)
		{
			Complex res;
			res._real=this->_real;
			res._imag=this->_imag+par;
			return res;

		}
		friend ostream & operator<<(ostream & os,Complex & comp)
		{
			os<<comp._real<<(comp._imag>=0?"+":"")<<comp._imag<<"i"<<endl;
			return os;
		}
		friend Complex  operator+(int  par,const Complex & cpar)
		{
			Complex res;
			res._real=cpar._real+par;
			res._imag=cpar._imag;
			return res;

		}
		Complex& operator=(double value)
		{
			this->_real=this->_imag=value;
			return *this;
		}
		operator double()
		{
			return this->_real+this->_imag;
		}
	};

int main()
{

	Complex c1(12.34,11.23);
	Complex c2(10.22,15.46);
	Complex c3=c1+c2;
	cout<<c1;
	cout<<c2;
	cout<<"_________________"<<endl;
	cout<<c3;
	cout<<"_________________"<<endl;
	Complex c4=c1+10;//10 to be added to the imaginary
	Complex c5=10+c2;//10 to be added to the real
	c3=123.45;//this value to assigned to real and imag
	double magnitude=c5;//magnitude should accept real+imag
	cout<<c4<<c5<<c3<<magnitude<<endl;
  return 0;
}
```

2. On Heap

```
#include<iostream>
using namespace std;
class CA
{
	static bool globalHeap;
	static int count;
	bool IsOnHeap;
public:
	static void* operator new(size_t size)
		{
		   count=1;
			globalHeap=true;
			return malloc(size);
		}
	static void* operator new[](size_t size)
		{
		   count=size/sizeof(CA);
			globalHeap=true;
			return malloc(size);
		}
	CA():IsOnHeap(globalHeap)
	{
		count--;
		if(count<=0){
		   globalHeap=false;
		}

	}
	void ObjectLocation()
	{
		if(IsOnHeap==true)
		{
			cout<<"Is On Heap "<<endl;
		}
		else
		{
			cout<<"Is (Not) on heap"<<endl;
		}
	}
};
bool CA::globalHeap=false;
int CA::count=0;
int main() //Never Change Main
{
	CA obj1;
	CA *ptr1=new CA();
	CA *ptr3=new CA[5];
	CA obj2;
	CA *ptr2=new CA();

	obj1.ObjectLocation();
	obj2.ObjectLocation();
	cout<<"_____"<<endl;
	ptr1->ObjectLocation();
	ptr2->ObjectLocation();
	cout<<"_____"<<endl;
	for(int i=0;i<5;i++)
	{
		(ptr3+i)->ObjectLocation();
	}

	delete ptr1;
	delete ptr2;
	delete [] ptr3;
	return 0;
}
```

3. Memory Created only On Stack

```
#include<iostream>
using namespace std;
class CA
{
	static void* operator new(size_t size)
	{
		return NULL;
	}
	static void* operator new[](size_t size)
	{
		return NULL;
	}
	static void operator delete(void* pv)
	{
		//do nothing
	}
	static void operator delete[](void* pv)
	{
		//do nothing
	}
public:
	CA()
	{
		cout<<"CA Ctor"<<endl;
	}
	~CA()
	{
		cout<<"CA D-tor"<<endl;
	}
};
int main()
{
	CA obj1;

/*	CA *ptr=new CA();
	CA *ptr1=new CA[5];
	delete ptr;
	delete [] ptr1;     */
	return 0;
}
```

4. Create object only heap using smart pointer

```
#include<iostream>
using namespace std;
class CA{
	CA()
	{
		cout<<"CA Ctor"<<endl;
	}
	~CA()
	{
		cout<<"CA D-tor"<<endl;
	}
	static void Release(CA *ptr)
	{
		delete ptr;
	}
	static CA* CreateCA()
	{
	  return new CA();
	}
public:
	void fun()
	{
		cout<<"CA Fun"<<endl;
	}
	friend class SmartPointer;
};
class SmartPointer//heap prohibited
{
	static void* operator new(size_t size)
	{
		return NULL;
	}
	static void* operator new[](size_t size)
	{
		return NULL;
	}
	static void operator delete(void* pv)
	{
	}
	static void operator delete[](void* pv)
	{
	}
	CA *ptr;
public:
	SmartPointer():ptr(CA::CreateCA())
	{
	}
	CA* operator->()
	{
		return ptr;
	}
	~SmartPointer()
	{
		delete ptr;
	}
};

int main()
{
	SmartPointer wrap;
	wrap->fun();
	return 0;
}
```

5. Polar Rectangle

```
#include<iostream>
#include<math.h>
#define PI 3.14
using namespace std;
class Rect;//forward declaration
class Polar
{
	double distance;
	double angle;
public:
	friend class Rect;
	Polar(double distance,double angle):distance(distance),angle(angle)
	{
	}
	operator double()
	{
		return distance;
	}
	double getDistance()
	{
		return distance*(cos(angle*PI/180));
	}
	double getAngle()
	{
		return distance*(sin(angle*PI/180));
}
	friend ostream& operator<<(ostream & os,Polar & ob)
	{
		os<<ob.distance<<" "<<ob.angle<<endl;
		return os;
	}
	 operator Rect();
};

class Rect
{
	double x;
	double y;
public:
	Rect():x(0),y(0)
		{
		}
	Rect(double x,double y):x(x),y(y)
	{
	}
	operator double()
	{
		return x+y;
	}
	operator Polar()
	{
		double r1=sqrt((x*x)+(y*y));
		double thita=atan(y/x);
		return Polar(r1,thita);
	}
	Rect& operator=(Polar & p)
	{
		Rect res;
		res.x=p.getDistance();//distance*(cos(p.angle*PI/180));
		res.y=p.getAngle();
		return res;
	}
	friend ostream& operator<<(ostream & os,Rect & ob)
	{
		os<<ob.x<<" "<<ob.y<<endl;
		return os;
	}
};

Polar::operator Rect()
{
	return Rect(getDistance(),getAngle());
}
```

```
int main()//never change main
{
	Polar p(4.0,45.0);
	Rect r(3.0,4.0);
	double d1=p;//should return the distance
	double d2=r;//should return (xcor + ycor);
	p=r;
	r=p;
	Polar p1=r;
	Rect r1=p;
	cout<<"d1="<<d1<<endl;
	cout<<"d2="<<d2<<endl;
	cout<<p<<endl;
	cout<<r<<endl;
	cout<<p1<<endl;
	cout<<r1<<endl;
	return 0;
}
```

6. Deep Copy

```
#include<iostream>
using namespace std;

class CA
{
	CA()
	{
		cout<<"CA Ctor"<<endl;
	}
	CA(const CA& par)
	{
		cout<<"CA Copy"<<endl;
	}
	~CA(){
		cout<<"CA Dtor"<<endl;
	}
public:
	void Fun()
	{
		cout<<"CA Fun"<<endl;
	}
	friend class Smart;
};
class Smart
{
	CA *ptr;
public:
	Smart():ptr(new CA())
	{
	}
	//Smart(const Smart& par):ptr(par.ptr)//shallow copy
	Smart(const Smart& par):ptr(new CA(*par.ptr))//Deep copy
	{
	}
	CA* operator->()
	{
		return ptr;
	}
	~Smart()
	{
		delete ptr;
	}
};

void ClientFun(Smart sm)
{
	cout<<"Apple Pie"<<endl;
	sm->Fun();
	cout<<"American Chopsy"<<endl;
}
//ClientFun(sm1);
void main()
{
	Smart sm1;
	Smart sm2(sm1);
}
}
```

7. Deep Assignment

```
#include<iostream>
using namespace std;

class CA
{
	CA()
	{
		cout<<"CA Ctor"<<endl;
	}
	CA(const CA& par)
	{
		cout<<"CA Copy"<<endl;
	}
	~CA(){
		cout<<"CA Dtor"<<endl;
	}
	CA& operator=(const CA& par)
	{
		return *this;
	}
public:
	void Fun()
	{
		cout<<"CA Fun"<<endl;
	}
	friend class Smart;
};
class Smart
{
	CA *ptr;
public:
	Smart():ptr(new CA())
	{
	}

	Smart(const Smart& par):ptr(new CA(*par.ptr))
	{
	}
	CA* operator->()
	{
		return ptr;
	}
	Smart& operator=(const Smart& par)
	{
		*ptr=*par.ptr;
		return *this;
	}
	~Smart()
	{
		delete ptr;
	}
};

void main()
{
	Smart sm1;
	Smart sm2(sm1);


	sm1=sm2;
}
}
```

8. Ownership Transfer

```
#include<iostream>
using namespace std;

class CA
{
	CA()
	{
		cout<<"CA Ctor"<<endl;
	}

	~CA(){
		cout<<"CA Dtor"<<endl;
	}

public:
	void Fun()
	{
		cout<<"CA Fun"<<endl;
	}
	friend class Smart;
};
class Smart
{
	CA *ptr;
public:
	Smart():ptr(new CA())
	{
	}
	Smart(Smart& par):ptr(par.ptr)
	{
		par.ptr=NULL;
	}
	Smart& operator=(Smart &par)
	{
		this->ptr=par.ptr;
		par.ptr=NULL;
		return *this;
	}
	CA* operator->()
	{
		return ptr;
	}
	~Smart()
	{
		delete ptr;
	}
};

void main()
{
	Smart sm1;
	sm1->Fun();
	Smart sm2(sm1);
	sm1=sm2;
}
```

9. Reference counting

```
#include<iostream>
using namespace std;

class CA
{
	CA()
	{
		cout<<"CA Ctor"<<endl;
	}

	~CA(){
		cout<<"CA Dtor"<<endl;
	}
public:
	void Fun()
	{
		cout<<"CA Fun"<<endl;
	}
	friend class Smart;
};
class Smart
{
	CA *ptr;
	int *count;
public:

	Smart():ptr(new CA()),count(new int(1))
	{
	}
	Smart(Smart& par):ptr(par.ptr),count(par.count)
	{
		++(*count);
	}
	Smart& operator=(Smart &par)
	{
		this->Smart::~Smart();
		this->Smart::Smart(par);
		return *this;
	}
	CA* operator->()
	{
		return ptr;
	}
	~Smart()
	{
		--(*count);
		if((*count)==0)
		{
			delete ptr;
			delete count;
		}
	}
};


void main()
{
	Smart sm1;
	sm1->Fun();
	Smart sm2(sm1);


	Smart sm3;
	Smart sm4(sm3);
	Smart sm5(sm3);

	sm3=sm1;
}
}
```

10. Read – write / [] operator overload

```
#include<iostream>
using namespace std;

class CArray
{
	int arr[5];
public:
	CArray()
	{
		arr[0]=11;
		arr[1]=22;
		arr[2]=33;
		arr[3]=44;
		arr[4]=55;
	}

	int & operator[](int index)
	{
		return arr[index];
	}

	friend ostream& operator<<(ostream& os,CArray & cArr)
	{
		for(int i=0;i<5;i++)
		{
			os<<"arr["<<i<<"]="<<cArr.arr[i]<<endl;
		}
		return os;
	}
};
void main()
{
	CArray arr;
	int x=arr[2];		//reading //arr.operator[](2);
	arr[3]=999;		//writing
	arr[1]=arr[2];		//read/write
	cout<<arr;
	cout<<"x="<<x<<endl;
}
```

11. Lazy Loading

```
#include<iostream>
using namespace std;
class CArray
	{
		int arr[5];
	public:
		CArray()
	{
		arr[0]=11;
		arr[1]=22;
		arr[2]=33;
		arr[3]=44;
		arr[4]=55;
	}
		class	Helper
		{
			CArray *ptr;
			int index;
		public:
			Helper(CArray *ptr,int index):ptr(ptr),index(index)
			{
```

```
                        }
                        operator int()
                        {
                                cout<<"Reading Business"<<endl;
                                return ptr->arr[index];
                        }
                        Helper& operator=(int val)
                        {
                                cout<<"Writing Business"<<endl;
                                ptr->arr[index]=val;
                                return *this;
                        }
                        Helper& operator=(Helper par)
                        {
                                cout<<"Reading/Writing Business"<<endl;
                                ptr->arr[index]=par.ptr->arr[par.index];
                                return *this;
                        }
                };

                Helper operator[](int index)
                {
                        return Helper(this ,index);
                }

                friend ostream& operator<<(ostream& os,CArray & cArr)
                {
                        for(int i=0;i<5;i++)
                        {
                                os<<"arr["<<i<<"]="<<cArr.arr[i]<<endl;
                        }
                        return os;
                }
        };
        void main()
        {
                CArray arr;
                int x=arr[2];//reading //arr.operator[](2);
                arr[3]=999;//writing
                arr[1]=arr[2];//read/write

                cout<<arr;
                cout<<"x="<<x<<endl;
        }
}
```

12. call back first

```
#include<iostream>
using namespace std;

typedef void (*FPTR)();
void VendorBusiness(FPTR fp)
{
        cout<<"Vendor Business started"<<endl;
        fp();//callback
        cout<<"Vendor Business completed"<<endl;
}

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

void ClientFun()
{
        cout<<"Hi from client "<<endl;
}
int main()
{
  FPTR fp=&ClientFun;
        VendorBusiness(fp);
  return 0;
}
}
```

13 callback second

```
#include<iostream>
using namespace std;
class CA
{
	int x;
	int y;
public:
	CA(int x,int y):x(x),y(y)
	{
	}
	void India()
	{
		cout<<"India x="<<x<<endl;
	}
	void Bharath()
	{
		cout<<"Bharath y="<<y<<endl;
	}
};

typedef void (CA::*FPTR)();
void VendorBusiness(FPTR fp,CA& par)
{
	cout<<"Vendor Business started 123"<<endl;
	(par.*fp)();//callback
	cout<<"Vendor Business completed 123"<<endl;
}


void VendorBusinessNew(FPTR fp,CA* par)
{
	cout<<"Vendor Business started 123456"<<endl;
	(par->*fp)();//callback
	cout<<"Vendor Business completed 123456"<<endl;
}
//------------------------

class Smart
{
	CA *ptr;
	FPTR fptr;
public:
	Smart(int x,int y):ptr(new CA(x,y))
	{
	}
	CA* operator->()
	{
		return ptr;
	}
	Smart& operator->*(FPTR fp)
	{
		fptr=fp;
		return *this;
	}
	void operator()()
	{
		(ptr->*fptr)();
	}
	~Smart()
	{
		delete ptr;
	}
};

void ClientFlow(Smart & sm,FPTR fp)
{
	cout<<"Client Business started 123456"<<endl;
	(sm->*fp)();//callback
	cout<<"Client Business completed 123456"<<endl;
}


int main()
{
  CA obj1(111,222);
	FPTR fp=&CA::Bharath;
```

```
		//VendorBusiness(fp,obj1);
		//VendorBusinessNew(fp,&obj1);
		Smart sm1(11,22);
		ClientFlow(sm1,fp);
   return 0;
}
```

14. Day 6

```
#include<iostream>
#include<string>
using namespace std;

class DB //interface
{
public:
		virtual void OpenDb()=0;
		virtual void CloseDb()=0;
		operator DB*()
		{
			return this;
		}
};

class IProvider
{
public:
		virtual void DispatchSMS(string msg)=0;
};

class SMS
{
		static IProvider *prov;
public:
		static void SetProvider(IProvider *provider)
		{
			prov=provider;
		}
		static void SendSms(string msg)
		{
			prov->DispatchSMS(msg);
		}
};
IProvider * SMS::prov=NULL;

class Account//abstract class
{
protected:
		DB *db;
		virtual void ActualJob(int accId,int amount)=0;//pure
		virtual string SmsMsg()=0;
public:
		void SetDb(DB *db)//dependency passed
		{
			this->db=db;
		}
		void DoDebit(int accId,int amount)
		{
			db->OpenDb();
			this->ActualJob(accId,amount);
			db->CloseDb();
			SMS::SendSms(SmsMsg());
			cout<<"_____"<<endl;
		}
};

class SavingsAccount:public Account
{
protected:
		string SmsMsg()
		{
			return "Savings Debit done ";
		}
		void ActualJob(int accId,int amount)
		{
```

```
                cout<<"Doing Debit operation for Savings Account"<<endl;

        }
};

class CurrentAccount:public Account
{
public:
        string SmsMsg()
        {
                return "Current Debit done ";
        }
        void ActualJob(int accId,int amount)
        {
                cout<<"Doing Debit operation for Current Account"<<endl;
        }
};
//-------------------------
class SqlDb:public DB
{
public:
        void OpenDb()
        {
                cout<<"Open DB Sql"<<endl;
        }
        void CloseDb()
        {
                cout<<"Close DB Sql"<<endl;
        }
};

class OraDb:public DB
{
public:
        void OpenDb()
        {
                cout<<"Open DB Ora"<<endl;
        }
        void CloseDb()
        {
                cout<<"Close DB Ora"<<endl;
        }
};

class BSNL:public IProvider
{
public:
        virtual void DispatchSMS(string msg)
        {
                cout<<"BSNL Dispatched SMS ("<<msg<<")"<<endl;
        }
};

class Verizon:public IProvider
{
public:
        virtual void DispatchSMS(string msg)
        {
                cout<<"Verizon Dispatched SMS ("<<msg<<")"<<endl;
        }
};


class IProviderNew
{
public:
        virtual void SMSMessage(string msg)=0;
};

class JIO:public IProviderNew
{
        public:
                void SMSMessage(string msg)
                {
                        cout<<"JIO sent sms ("<<msg<<")"<<endl;
                }
};

class Adapter:public IProvider
```

```
{
	IProviderNew *prov;
public:
	Adapter(IProviderNew *prov):prov(prov)
	{
	}
	virtual void DispatchSMS(string msg)
	{
		prov->SMSMessage(msg);
	}
};

void main()
{
	JIO jio;

	SMS::SetProvider(new Adapter(&jio));
	SqlDb sqlDb;
	SavingsAccount sa;
	sa.SetDb(sqlDb);
	CurrentAccount ca;
	OraDb oraDb;
	ca.SetDb(oraDb);
	sa.DoDebit(101,30000);
	ca.DoDebit(102,20000);
}
```

14.Chrome firefox explolar

```
#include<iostream>
#include<string>
using namespace std;

class IAudio
{
public:
	virtual void CreateAudio()=0;
	virtual void AttachAudio()=0;
};
class IVideo
{
public:
	virtual void CreateVideo()=0;
	virtual void AttachVideo()=0;
};
class IMag
{
public:
	virtual void ZoomIn()=0;
	virtual void ZoomOut()=0;
};

class IFactory //Neeraj
{
public:
	virtual IAudio* BuildAudio()=0;
	virtual IVideo* BuildVideo()=0;
	virtual IMag* BuildMag()=0;
};

class ExpAudio :public IAudio
{
public:
	virtual void CreateAudio()
	{
		cout<<"Exp Audio Created"<<endl;
	}
	virtual void AttachAudio()
	{
		cout<<"Exp Audio Attached"<<endl;
	}
};


class ExpVideo:public IVideo
{
public:
```

```
	virtual void CreateVideo()
	{
		cout<<"Exp Video Created"<<endl;
	}
	virtual void AttachVideo()
	{
		cout<<"Exp Video Attached"<<endl;
	}
};

class ExpMag:public IMag
{
public:
	virtual void ZoomIn()
	{
		cout<<"Exp Zoom In"<<endl;
	}
	virtual void ZoomOut()
	{
		cout<<"Exp Zoom Out"<<endl;
	}
};

class ExpFactory:public IFactory
{
public:
	virtual IAudio* BuildAudio()
	{
		return new ExpAudio();
	}
	virtual IVideo* BuildVideo()
	{
		return new ExpVideo();
	}
	virtual IMag* BuildMag()
	{
		return new ExpMag();
	}
};


class ChromeAudio :public IAudio
{
public:
	virtual void CreateAudio()
	{
		cout<<"Chrome Audio Created"<<endl;
	}
	virtual void AttachAudio()
	{
		cout<<"Chrome Audio Attached"<<endl;
	}
};


class ChromeVideo:public IVideo
{
public:
	virtual void CreateVideo()
	{
		cout<<"Chrome Video Created"<<endl;
	}
	virtual void AttachVideo()
	{
		cout<<"Chrome Video Attached"<<endl;
	}
};

class ChromeMag:public IMag
{
public:
	virtual void ZoomIn()
	{
		cout<<"Chrome Zoom In"<<endl;
	}
	virtual void ZoomOut()
	{
		cout<<"Chrome Zoom Out"<<endl;
	}
```

```
};

class ChromeFactory:public IFactory
{
public:
	virtual IAudio* BuildAudio()
	{
		return new ChromeAudio();
	}
	virtual IVideo* BuildVideo()
	{
		return new ChromeVideo();
	}
	virtual IMag* BuildMag()
	{
		return new ChromeMag();
	}
};


class FireFoxAudio :public IAudio
{
public:
	virtual void CreateAudio()
	{
		cout<<"FireFox Audio Created"<<endl;
	}
	virtual void AttachAudio()
	{
		cout<<"FireFox Audio Attached"<<endl;
	}
};


class FireFoxVideo:public IVideo
{
public:
	virtual void CreateVideo()
	{
		cout<<"FireFox Video Created"<<endl;
	}
	virtual void AttachVideo()
	{
		cout<<"FireFox Video Attached"<<endl;
	}
};

class FireFoxMag:public IMag
{
public:
	virtual void ZoomIn()
	{
		cout<<"FireFox Zoom In"<<endl;
	}
	virtual void ZoomOut()
	{
		cout<<"FireFox Zoom Out"<<endl;
	}
};


class FireFoxFactory:public IFactory
{
public:
	virtual IAudio* BuildAudio()
	{
		return new FireFoxAudio();
	}
	virtual IVideo* BuildVideo()
	{
		return new FireFoxVideo();
	}
	virtual IMag* BuildMag()
	{
		return new FireFoxMag();
	}
};

IFactory * CreateFactory(int choice)
```

```
{
	if(10==choice)
	{
		return new ExpFactory();
	}
	else if(20==choice)
	{
		return new ChromeFactory();
	}
	else
	{
		return new FireFoxFactory();
	}
}

void main()
{
	IFactory *factory=CreateFactory(20);
	IAudio *aud=factory->BuildAudio();
	aud->CreateAudio();
	aud->AttachAudio();
	IVideo *video=factory->BuildVideo();
	video->CreateVideo();
	video->AttachVideo();
	IMag *mag=factory->BuildMag();
	mag->ZoomIn();
	mag->ZoomOut();
}
```

16. GAME

```
#include<iostream>
using namespace std;
namespace nm064
{
//To be solved in DS
class Game //has to be polymorphic for RTTI to work
{
public:
	virtual void Collides(Game & game)=0;
	virtual void Collided()=0;
};
class Ship:public Game{
public:
	void Collides(Game & game)
	{
		cout<<"Ship Collides with ";
		game.Collided();
	}
	void Collided()
	{
		cout<<"Ship"<<endl;
	}
};
class Station:public Game{
	public:
	void Collides(Game & game)
	{
		cout<<"Station Collides with ";
		game.Collided();
	}
	void Collided()
	{
		cout<<"Station"<<endl;
	}

};
class Asteroid:public Game{
	public:
	void Collides(Game & game)
	{
		cout<<"Asteroid Collides with ";
		game.Collided();
	}
	void Collided()
	{
		cout<<"Asteroid"<<endl;
```

```
	}
};

void ProcessCollission(Game & obj1,Game & obj2)
{
			obj1.Collides(obj2);
}

void main()//never change main
{
	Ship sp;
	Station st;
	Asteroid ast;
	ProcessCollission(sp,ast);
	ProcessCollission(sp,st);
	ProcessCollission(st,ast);
	ProcessCollission(ast,sp);

}
```

17. [][] overload

```
#include<iostream>
using namespace std;

class ArrayWrapper
{
	int arr[3][3];
public:
	ArrayWrapper()
	{
		for(int i=0;i<3;i++)
		{
			for(int j=0;j<3;j++)
			{
				arr[i][j]=0;
			}
		}
	}

	class Helper
	{
		int index;
		ArrayWrapper * ptr;
	public:
		Helper(ArrayWrapper * ptr,int index):ptr(ptr),index(index)
		{
		}
		int & operator[](int index)
		{
			return ptr->arr[this->index][index];
		}
	};


	Helper operator[](int index)
	{

		return Helper(this,index);
	}
	friend ostream& operator<<(ostream& os,ArrayWrapper & par)
	{
		for(int i=0;i<3;i++)
		{
			for(int j=0;j<3;j++)
			{
				os<<par.arr[i][j]<<"\t";
			}
			os<<endl;
		}
		return os;
	}
};

void main()
{
	ArrayWrapper smArr;
```

```
	smArr[1][1]=100;
	smArr[1][2]=999;
	int x=smArr[1][2];
	cout<<smArr<<endl;
	cout<<"x="<<x<<endl;
}
```

18. Singleton

```
#include<iostream>
using namespace std;

class CA
{
	CA()
	{
		cout<<"CA Ctor"<<endl;
	}
	~CA()
	{
		cout<<"CA D-tor"<<endl;
	}
	static CA* head;
public:

	void Display()
	{
		cout<<"CA Display"<<endl;
	}
	static CA* CreateCA()
	{
		if(head==NULL)
		{
			head=new CA();
		}
		return head;
	}
	static void ReleaseCA()
	{
		delete head;
		head=NULL;
	}
};
CA* CA::head=NULL;
class Smart
{
	CA* ptr;
	static int count;
	static void* operator new(size_t size)
	{
		return NULL;
	}
	static void* operator new[](size_t size)
	{
		return NULL;
	}
	static void operator delete(void *pv)
	{
	}
	static void operator delete[](void *pv)
	{
	}
public:
	Smart():ptr(CA::CreateCA())
	{
		count++;
	}
	CA*operator->()
	{
		return ptr;
	}
	Smart& operator=(Smart& par)
	{
		return *this;
	}
	~Smart()
	{
```

```
                        if((--count)==0)
                        {
                                CA::ReleaseCA();
                        }
        }
};
int Smart::count=0;
void main()//never change main
{
        Smart sm1;
        Smart sm2;
        Smart sm3;
        sm2->Display();
        sm2=sm3;
}
```