DATE :- 14.05.22

```
#include<iostream>
using namespace std;
class A{
	int a,b;
	int *p;
	int *ptr1=new int(10);  //value is assigned
//	delete ptr1;	//the dynamic memory allocated by the pointer is deleted
	int *ptr2=new int[10];  //array of 10 elements of same type is created from heap memory
//	delete[] ptr2;
	public:
/*		A(){}		//do-nothing constructor or empty constructor just to initialize garbage value because compiler is not
		//creating any default constructor as there are other constructor
		cout<<"do-nothing constructor called for object="<<this<<endl;
		*/
		A():a(5),b(10){ }	//Another way
/*		A(){		//Another way
		this->a=5;	//this hold the address of the calling object	//pointer to the object, so ->
		this->b=10;
		}*/
//		A(int,int);		//Another way
		A(int,int=5);	//Another way  //declaration within class	//Parameterized Constructor
		A(const A&);
		~A(){	//destructor	//In the order the objects are created,in reverse order the objects will be destructed
			cout<<"destructor called for object="<<this<<endl;
		}
};
A::A(int x,int y):a(x),b(y){	//definition outside the class
cout<<"parameterized constructor called for object="<<this<<endl;
}

A::A(const A& temp){
	this->a=temp.a;
	this->b=temp.b;
	this->ptr1=temp.p;
	cout<<"copy-constructor called for object="<<this<<endl;
}
int main(){
	A obj1; //obj1.A
	A obj2(10,20); //obj2.A(10,20)
	A obj3(10);	//obj2.A(10)	//direct initialization statement
	A obj4=obj3;	//obj4.A(obj3)  //copy constructor	//copy initialization
```

```
	//A obj4(obj3); //direct initialization
	A obj5=1;	//obj5.A(1)	//temporary object is created internally	//copy initialization statement
					//parameterized constructor is called then copy constructor
	A obj6=A();	//obj6.A(A()) //calls do-nothing cons then calls copy cons	//if do-nothing cons is not there parameterized cons will be called
	obj1=obj4;	//obj1.operator=obj4	//A constructor is not called

	return 0;
}
```

```
//DATE :- 28.05.22
//SHALLOW COPY vs DEEP COPY
#include<iostream>
using namespace std;
class stack{
	int sp;
	int *data;
	int maxsize;
public:
	stack(int=20);
	stack(const stack&);
bool push(int);
stack& operator=(const stack&);
bool pop(int&);
	~stack();
};
/* SHALLOW COPY
stack::stack(const stack& obj){
	this->sp=obj.sp;
	this->data=obj.data;
	this->maxsize=obj.maxsize;
}*/

/* DEEP COPY
stack::stack(const stack& obj){
	this->sp=obj.sp;
	this->maxsize=obj.maxsize;
	this->data=new int[maxsize];
	for(int i=0;i<maxsize;i++)
		this->data[i]=obj.data[i];
} */

/*
stack::stack(int m){
	this->sp=-1;
```

```
	this->maxsize=m;
	this->data=new int[maxsize];
}*/
/*
//compiler provides default overload assignment only for assignment operator
stack& stack::operator=(const stack& obj){
	this->sp=obj.sp;
	this->data=obj.data;
	this->maxsize=obj.maxsize;
	return(*this);	//deferencing this
}
*/

stack& stack::operator=(const stack& obj){
	if(this!=&obj){  //this if is for in case of self assignment
	this->sp=obj.sp;
	maxsize=obj.maxsize;
	delete[] data;   //so that there is no memory leak
	data=new int[maxsize];
	for(int i=0;i<maxsize;i++)
		this->data[i]=obj.data[i];
	}
	return(*this);	//deferencing this
}

stack::~stack(){}	//destructor code

int main(){
	stack s1;
	stack s2=s1;	//s2.stack(s1)
//	stack s3(s1);
	stack s3;	//s3.stack()
	stack s4;
	s4=s1; //s4.operator=s1
	s3=s2=s4=s1; //chain of assignment
	s4=s4; //self assignment

	return 0;
}
```

```
//DATE :- 28.05.22
#include<iostream>
using namespace std;
class stack{
	int sp;
	int *data;
```

```
	int maxsize;
public:
	stack(int=20);
	stack(const stack&);
bool push(int);
stack& operator=(const stack&);
bool pop(int&);
	~stack();
};
bool stack::push(int val){
	if(sp<maxsize-1){
		data[++sp]=val;
		return true;
	}
	else
		return false;
}
bool stack::pop(int &v){
	if(sp==-1)
		return false;
	else{
		v=data[sp--];
		return true;
	}
}
int main(){
	stack s1;
	char ch;
	cout<<"Enter your choice"<<endl;
	cin>>ch;
	switch(ch){
		case 1: int value;
				cout<<"Enter value to push"<<endl;
				cin>>value;
				if(s1.push(value))
					cout<<"successful"<<endl;
				else
					cout<<"overflow"<<endl;
				break;
		case 2: int val;
		if(s1.pop(val))
			cout<<"popped value"<<val<<endl;
		else
			cout<<"underflow"<<endl;
		break;
	}
	return 0;
}
```