Questions are NOT ALLOWED during exam, take reasonable assumptions where needed.

| Question No. | Marks | Marks Obtained |
|---|---|---|
| Q1 | 10 | |
| Q2 | 15 | |
| Q3 | 20 | |
| Q4 | 25 | |
| Q5 | 25 | |
| Total | 95 | |

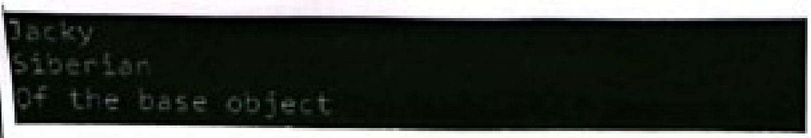
| | |
|---|---|
| cout<<"Swap Successful";<br>} | |
| template <><br>void my_swap (Person &one, Person &two)<br>{<br>cout<<"You cannot swap Person ";<br>} | |
| *int main()* | |
| { | |
| int a=10, b=20; | |
| cout <<a<<" "<<b<<endl; | 10 20 |
| my_swap(a,b); | Swap Successful |
| cout <<a<<" "<<b<<endl; | 20 10 |
| | |
| double *x= new double(10.5); | |
| double *y= new double(11.5); | |
| cout <<*x<<" "<<*y<<endl; | 10.5 11.5 |
| my_swap(*x,*y); | Swap Successful |
| cout <<*x<<" "<<*y<<endl; | **11.5 10.5** |
| | |
| //overloaded constructor takes account name as input | |
| Person P1("Ron"), Person P2("Harry"); | |
| cout<<P1<<" "<<P2; | Ron Harry |
| my_swap(P1, P2); | **You cannot swap person** |
| cout<<P1<<" "<P2; | **Ron Harry** |
| } | |
| How many instances (copies) of my_swap functions are created at compile time in above code? | **3, one for int one for double and one for person** |

```
  cin>>amount;

  if (amount<1)
    throw out_of_range("Enter positive number");
  if (amount>5000)
    throw exception("Amount should be less than 5001");
}

int main(){
   try{
        char a;
        getTransType(a);
        try{
              int amount;
              getAmount(amount);
              cout<<"Successful transaction";
        }
        catch(exception e){
              cout<<e.what();
        }
    }
    catch(invalid_argument ia) {
        cout<<ia.what();
    }
 }
```

| 1) User enters 'W' and then 0 | 2) User wants to enters 'D' and then 6000 | 3) User want to enter 'S' and then 6000 |
|---|---|---|
| Enter positive number | Amount should be less than 5001 | Incorrect Transaction type |

```
        string traits;
        double * age; //age in years
public:
        baseClass baseObj;

        petClass(string name="Bailey", string traits="Husky", double* age=NULL){
                this->name=name;
                this->traits=traits;
                this->age= age;
        }

        petClass(baseClass baseObj){
                cout<<baseObj.name<<endl<<baseObj.traits<<endl<<"Of the base object";
        }

        void showPet(){
                cout << this->name<<"\t"<< this->traits;
                cout<<"\t" << *this->age << endl;
        }
};

void main(){
        petClass *dog= new petClass();
        dog->baseObj.name="Jacky";
        dog->baseObj.traits="Siberian";
        dog->baseObj.age= 4.5;
        petClass * myDog = new petClass(dog->baseObj);
        myDog->showPet();
        delete myDog;
        delete dog;
}
```

**Output:**



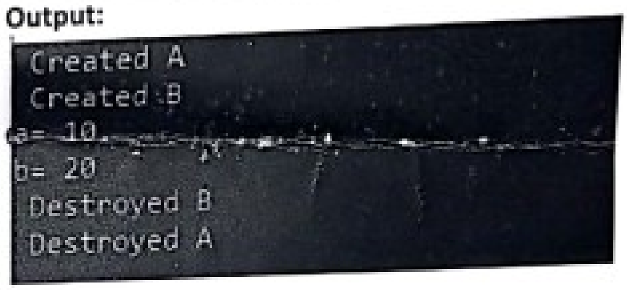
```
Jacky
Siberian
Of the base object
```

```
	{
		cout<<"a= "<<a<<endl;
	}

	virtual ~A()
	{
		cout<<" Destroyed A"<<endl;
	}
};
```

```
		cout<<"b= "<<b<<endl;
	}
	~B()
	{
		cout<<" Destroyed B"<<endl;
	}
};
```

```
int main()
{
	A *aPtr= new B(10, 20);
	aPtr->print();
	delete aPtr;

	return 0;
}
```

**Output:**



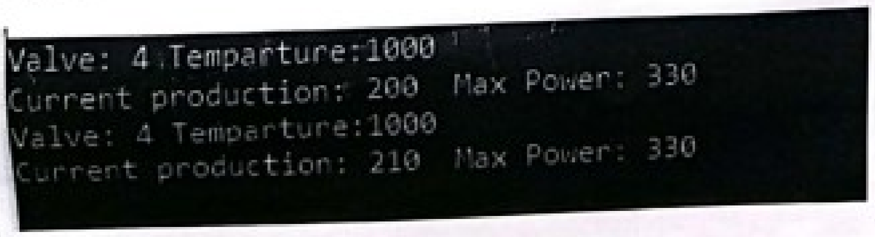
```
 Created A
 Created B
a= 10
b= 20
 Destroyed B
 Destroyed A
```

**Part (C)**

```
class ThermalReactor{
	int valve;
	float temprature;
public:
	ThermalReactor(int v, float t)
	{
		valve=v;
		temprature=t;
	}

	virtual void print(){
			cout<<"Valve: "<<valve;
			cout<<" Temparture:" <<temprature<<endl;
	}
};
```

```
	void signal(){ cout<<"Production cannot be increased"<<endl;}

	void increaseProd(float factor)
	{
		if((production+factor)<maxPower){
			production+=factor;
			print();
		}
		else signal();
	}

	void print(){
		ThermalReactor::print();
		cout<<"Current production: "<<production;
		cout<<"  Max Power: "<<maxPower<<endl;
	}
};

void Capacity(ThermalReactor * reactor ){
	reactor->print();
	dynamic_cast<MagnoxReactor *>(reactor)->increaseProd(10);

}

int main(){
	MagnoxReactor *MagRec= new MagnoxReactor(4, 1000, 330, 200);
	Capacity(MagRec); return 0;
}
```

**Output:**



```
Valve: 4 Temparture:1000
Current production: 200  Max Power: 330
Valve: 4 Temparture:1000
Current production: 210  Max Power: 330
```

CamScanner

```
void alloc(int* a, int size) {
      a = new int[size];
}

void main() {
      int* arr;
      alloc(arr, 10);
      arr[0] = 10;
}
```

**memory leakage:**
Inside function, pointer a is passed by value, So, inside main, arr pointer does not have any memory after function call.

```
void allocate(int** a2d, int rows, int cols)
{
      a2d = new int* [rows];
      int** endptr = a2d + cols;
      for   (int   **temp=a2d;   temp<endptr;
temp++)
            temp = new int*[cols];
}
void main()
{
      int** x=nullptr;
      allocate(x, 3, 6);
}
```

**memory leakage:**
Inside function, pointer a2d is passed by value, So, inside main, x pointer does not have any memory after function call.
Also, inside allocate function, temp is being initialized with array of pointers, which is again total lost

```
int* sum(int* a) {
      int s = *a + *a;
      return &s;
}
void main() {
      int num = 10;
      int *sumPtr = sum(&num);
      cout << *sumPtr << endl;

}
```

**dangling pointer:**
Variable s inside the function is a local variable, which will be destroyed by the end of function execution. So, inside main sumPtr would be dangling.