SOLUTION OF QUESTION 1:

1)

(a)

No matching template function call that can receive two different types as parameter.

(b)

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

template<typename T1, typename T2>
void print_max (const T1& a, const T2& b)
{
      cout<<((a>b)? a:b) << endl;
}

int main()
{
      print_max(4, 5.5);
      print_max(3.2, 1);
      return 0;
}
```

2)

```
class A
{
      int x;
      public:
      friend void set();
      display()
      {
            cout<<x<<endl;
      }
};

void set()
{
      A a;
      a.x = 10;
      a.display();
}
```

3)

(a)

“Int x” is a private data of class A. It can only be accessed inside the class A.

(b)

```
#include<iostream>
```

```
using namespace std;

class A
{
      int x;
      public:
      friend class B;
      display()
      {
            cout<<x;
      }
};

class B
{
      A a;
      display()
      {
            cout<<a.x;
      }
};

int main()
{
      B b;
}
```

4)

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

class shape
{
      int x;
      public:
      shape(int x)
      {
            this->x = x;
      }
};

class circle : virtual public shape
{
      int y;
      public:
      circle(int x, int y) : shape(x)
      {
            this->y = y;
      }
};
```

```
class square : virtual public shape
{
	int z;
	public:
	square(int x, int z) : shape(x)
	{
		this->z = z;
	}
};

class circle_on_square : public circle, public square
{
	int a;
	public:
	circle_on_square(int x, int y, int z, int a) : shape(x), circle(x,y), square(x,z)
	{
		this->a = a;
	}
};

int main()
{
	circle_on_square cc(1,2,3,4); //x=1, y=2, z=3, a=4
	return 0;
}
```

SOLUTION OF QUESTION 2:

- **a)** It violates encapsulation.
- **b)** When parent and child classes are using the same function as per their need and you need to have correct binding of the function.
- **c)** For a class to be abstract it needs to have a PURE Virtual Function. i.e. `abc() = 0`
- **d)** Child class access privileges allow child classes to access public members of their parent(s),
- **e)** Return Type is not considered when functions are bind.
- **f)** While, C++ views every file/entity in terms of an OBJECT no matter whatever is the extension.

SOLUTION OF QUESTION 3:

Task 1)

```
class Employer
{
	string emp_ID;
	Moderator* m;
	static int counter;	// for initializing unique ID

};

class Company: public Employer
{
	int active_projects;
};
class Pharma: public Employer
{
	int budget;
};
class Educational: public Employer
{
	int num_of_campuses;
};
class Candidate
{
	string candid_ID;
	string name, CNIC, degree, DOB, address;
	int expected_salary, experience;
	static int counter;	// for initializing unique ID
};
class Moderator
{
	string mod_ID;
	static int counter;	// for initializing unique ID
};
```

Task 2)

```
class Employer
{
	virtual void post_vacancy() = 0;
};

class Company: public Employer
{
	void post_vacancy()
	{
	cout << "We require individuals with ability to work in remote areas";
	}
};

class Pharma: public Employer
{
	void post_vacancy()
	{
	cout << "We require individuals with good analytical skills";
	}
};

class Bank: public Employer
{
	void post_vacancy()
	{
	cout << "We require individuals with good communication skills";
	}
};

class Educational: public Employer
{
	void post_vacancy()
	{
	cout << "We require individuals with ability to cope with pressure";
	}
};
```

Task 3)

```
		Employer(Moderator _m)
		{
			emp_ID = "e" + counter;
			m = &_m;
			++counter;
		}


		Company(int active_proj, Moderator _m): Employer(_m)
		{
			active_projects = active_proj;
		}


		Pharma(int _budget, Moderator _m): Employer(_m)
		{
			budget = _budget;
		}

		Bank(int numBranches, Moderator _m): Employer(_m)
		{
			num_of_branches = numBranches;
		}

		Educational(int numCampuses, Moderator _m): Employer(_m)
		{
			num_of_campuses = numCampuses;
		}

	Candidate(string name, string CNIC, string degree, string DOB, string address, int
expSal, int exp)
	{
		candid_ID = "c" + counter;
		++counter;
		this->name = name;
		this->CNIC = CNIC;
		this->degree = degree;
		this->address = address;
		this->DOB = DOB;
		expected_salary = expSal;
		experience = exp;
	}

	Moderator()
	{
		mod_ID = "m" + counter;
		++counter;
	}
```

Task 4)

```
Moderator    mod1;

Company      CapsuleCorp("Time machine", mod1);

Bank        HBL(10, mod1);

Educational NUCES(5, mod1);

Pharma            Pfizer(10000, mod1);

Candidate   candid1("Tom", "42301-11122355", "BS", "28-January", "Nowhere Street",
200000, 8);
```

Task 5)

```
	void receive_application(Candidate c)
	{
		++num_of_applications;
		select_candidate(c);
	}
```

Task 6)

```
	void select_candidate(Candidate c)
	{
		m = new Moderator();

		if(c.get_experiene() > 5)
		{
			cout << "Candidate selected"  << endl;
			m->func("Vacancy closed");
		}
	}
```

Task 7)

```
// add this statement to Candidate class
friend void operator < (Candidate,  Candidate);


// Global operator overload
void operator < (Candidate c1, Candidate c2)

{

      int exp1 = c1.get_experience();

      int exp2 = c2.get_experience();


      if(exp1 >= exp2)  // using >= since case of tie is undefined
            cout << "Candidate 1 is more experienced";

      else

            cout << "Candidate 2 is more experienced";

}
```

Task 8)

```
      // member function of Moderator class
      void find_num_of_applications()
      {
            cout << "Total applications: " << Employer::num_of_applications << endl;
      }
```

Task 9)

```
      void write_data()
      {
            int n = Employer::num_of_applications;
            ofstream o("D:\\info.txt");
            o.write << n;
            o.close();
      }
```

Task 10)

```
      void write_data(string msg)
      {
            ofstream o("D:\\messages.txt");
            o.write << msg;
            o.close();
      }
```

SOLUTION OF QUESTION 4:

1)

```
class Device
{
 int yearofmanufacture; // validity may be checked for range
public:
 int getYear() { return yearofmanufacture; }
};
class TrackingDevice: public Device
{
 float accuracy;
};
class LED: public Device
{
 string screensize, model;
 int supportedapps;
};
class Mobile: public Device
{
 string resolution, model;
};
class Tablet: public Device
{
string screensize, model;
};
class SmartRing: public Mobile, public TrackingDevice
{
};
```

2)
Diamond problem can occur in SmartRing since it inherits grandparent attribute yearofmanufacture from both Mobile and TrackingDevice. Since there are no overlapping attributes in Mobile and TrackingDevice, diamond problem can be resolved by using virtual inheritance.

3)
```
template <class T> void item_sort(T item1, T item2, T item3){

int y1 = item1.getYear();

int y2 = item2.getYear();

int y3 = item3.getYear();

// sort the three integers using your sorting logic e.g. Bubble Sort

}
```
4)
```
class Device{

int yearofmanufacture; // validity may be checked for range

public:

int getYear() { return yearofmanufacture; }

friend class RedTech;

};
```

5)
```
class MyException: public exception {

public:
const char* what() const throw()

{ return "Earlier than 2010";}
};
class RedTech{

void checker(TrackingDevice t)

{
try{
if(t.getYear() < 2010)

throw new MyException();

}

catch(MyException e)

{ cout << e.what();

}

}

};
```

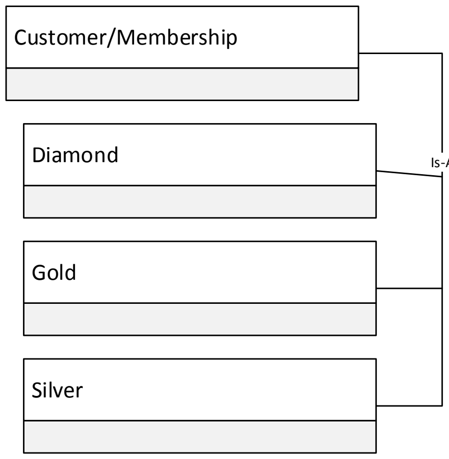
SOLUTION OF QUESTION 5:

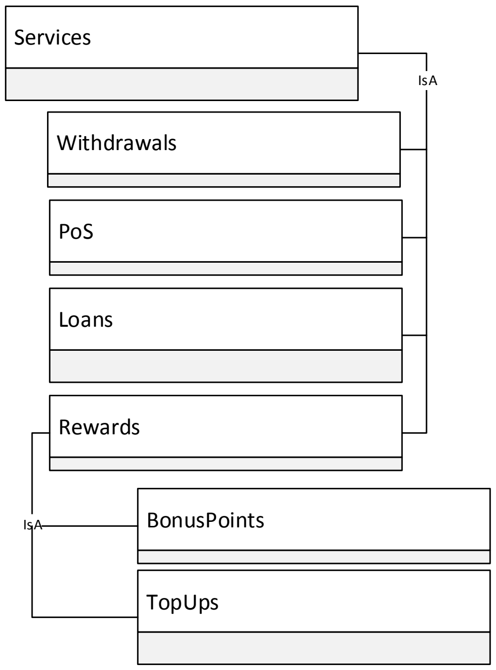
1)

```
Membership/Customers
 Diamond
 Gold
 Silver

Services
 Cash_Withdrawals
 Loans
 PoS
 Rewards
   Topups
   Bonus_Points
```

2)





3)

/* All the rules specified in text needed to be translated in code */

4)

// Code for any friend function it could be any REWARD calculation function with access to PoS class data, FREE RESPONSE