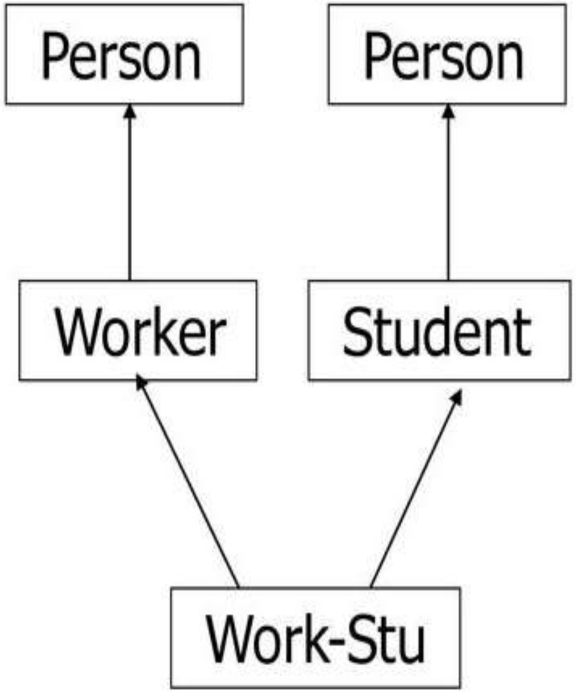
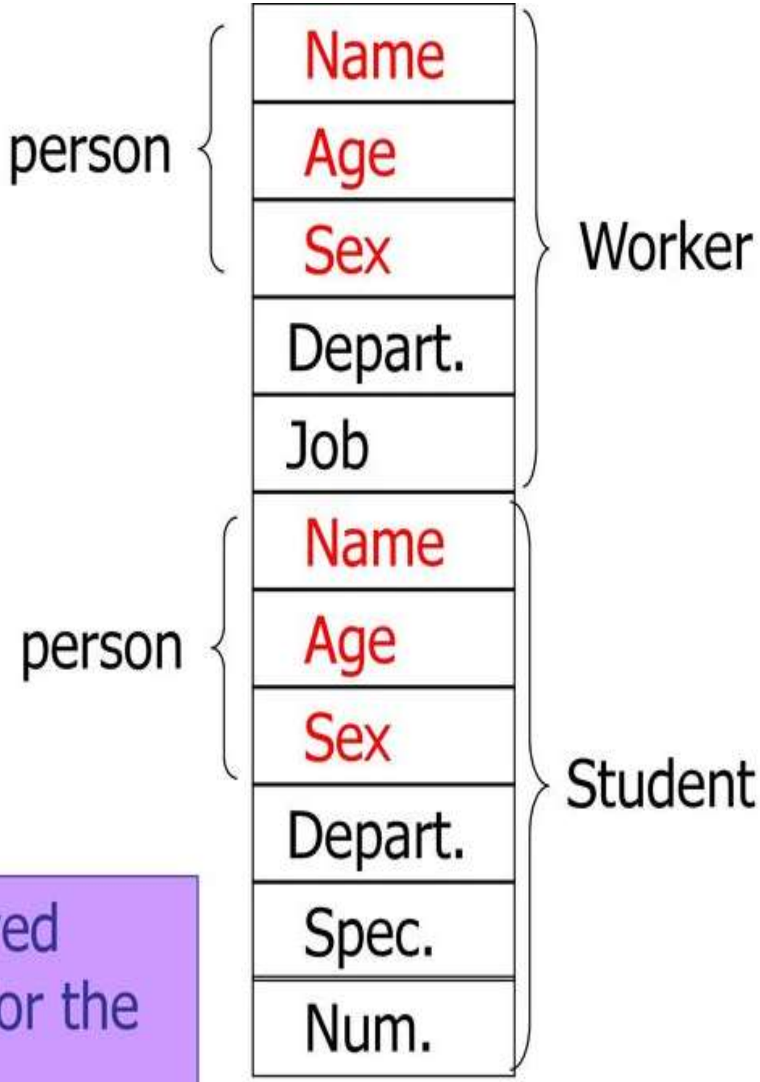
Virtual Function & Pure Virtual Function





The structure of the common derived class and the form of the storage for the member data.

```
class A{
public:
   void
   showa(){cout<<"A"<<endl;}
};
class B: public A{
public:
   void
   showb(){cout<<"B"<<endl;}
};
class C: public A{
public:
   void
   showc(){cout<<"C"<<endl;}
};
```

```
class D: public B, public C{
public:
   void
   showd(){cout<<"D"<<endl;}
};

void main()
{
    D d;
   d.showa();
}
```

Virtual Function

- A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class.
- When we refer to a derived class object using a pointer or a reference to the base class, we can call a virtual function for that object and execute the derived class's version of the function.
- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve runtime polymorphism.
- Functions are declared with a virtual keyword in base class.
- The resolving of function call is done at runtime.

Rules for Virtual Function

- Virtual functions cannot be static.
- A virtual function can be a friend function of another class.
- Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism.
- The prototype of virtual functions should be the same in the base as well as derived class.
- They are always defined in the base class and overridden in a derived class. It is not mandatory for the derived class to override (or re-define the virtual function), in that case, the base class version of the function is used.
- A class may have virtual destructor but it cannot have a virtual constructor.

Example 1

```
#include<iostream>
using namespace std;
  class base {
public:
    virtual void print() {
        cout << "print base class\n";
    }
   void show() {
        cout << "show base class\n";
    }
};
class derived : public base {
public:
    void print()  {
        cout << "print derived class\n";
    }
  void show()  {
        cout << "show derived class\n";
    }
};
  int main()   {
    base *bptr;
    derived d;
    bptr = &d;
  bptr->print();
 bptr->show();
return 0;
}
```

Example 2

```
#include<iostream>
using namespace std;
  class base {
public:
    void fun_1() { cout << "base-1\n"; }
    virtual void fun_2() { cout << "base-2\n"; }
    virtual void fun_3() { cout << "base-3\n"; }
    virtual void fun_4() { cout << "base-4\n"; }
};
class derived : public base {
public:
    void fun_1() { cout << "derived-1\n"; }
    void fun_2() { cout << "derived-2\n"; }
    void fun_4(int x) { cout << "derived-4\n"; }
};
  int main()  {
    base *p;
    derived obj1;
    p = &obj1;
   p->fun_1();
   // Late binding (RTP)
    p->fun_2();
   p->fun_3();
   p->fun_4();
    return 0;  }
```