Polymorphism

- Polymorphism is an ability to appear in many forms.
- In OOPS multiple routines sharing a common name is termed as Polymorphism.
- In SV, Polymorphism allows a parent class handler to hold sub class object and access the methods of those child classes from the parent class handler.
- To achieve this, functions/tasks in SV are declared as virtual functions/tasks which allow child classes to override the behaviour of the function/task.
- Properties can’t be virtual.

Example1

```
class shape;                              //Main Class
protected x, y, z;
virtual function void display();     //Function call can be
$display("I am shape");              // overridden, will call
endfunction                           //child function instead

virtual function void perimeter();
$display("I don't know perimeter");
endfunction

endclass
```

Example1

```
class rectangle extends shape;
virtual function void display();
$display("I am rectangle");
endfunction

virtual function void perimeter();
$display("perimeter=%0d", 2*(x + y));
endfunction
function new (int x, y); .....
endclass
```

Example1

```
class square extends rectangle;
function void display();          //This function call can show
 $display("I am square");        // polymorphism for rectangle class
   endfunction                     // and shape class

function void perimeter();
$display("perimeter=%0d", 4*x);
endfunction
function new (int x); .....
endclass
```

Example1

```
class triangle extends shape;
function void display();
$display(“I am  a triangle”);
endfunction

function void perimeter();
$display(“perimeter=%0d”, (x + y + z));
endfunction
function new (int x, y, z); .....
endclass
```

Example1

```
shape s1, s2;
rectangle r1,r2;
square sq1;
triangle t1;

initial begin
s1=new;
r1=new(2, 3);
sq1=new(4);
t1=new(1, 2, 3);
```

```
s1.display;     s1.perimeter;
r1.display;     r1.perimeter;
t1.display;     t1.perimeter;
s2=t1;
s2.display;     s2. perimeter;
r2=sq1;
r2.display;     r2. perimeter;
s2=r1;
s2.display;     s2. perimeter;  end
```

Example1

Result :

```
I am shape        I don’t know perimeter
I am rectangle     Perimeter= 10
I am triangle      Perimeter= 6
I am triangle      Perimeter= 6
I am square       Perimeter= 16
I am rectangle     Perimeter= 10
```

Example2

```
class parent;
int a=3;

function void d1();
$display(“Parent d1”);
endfunction

virtual function void d2();
$display(“Parent d2”);
endfunction
endclass
```

```
class child extends parent;
int b=8;

function void d1();
$display(“Child d1”);
endfunction

function void d2();
$display(“Child d2”);
endfunction
endclass
```

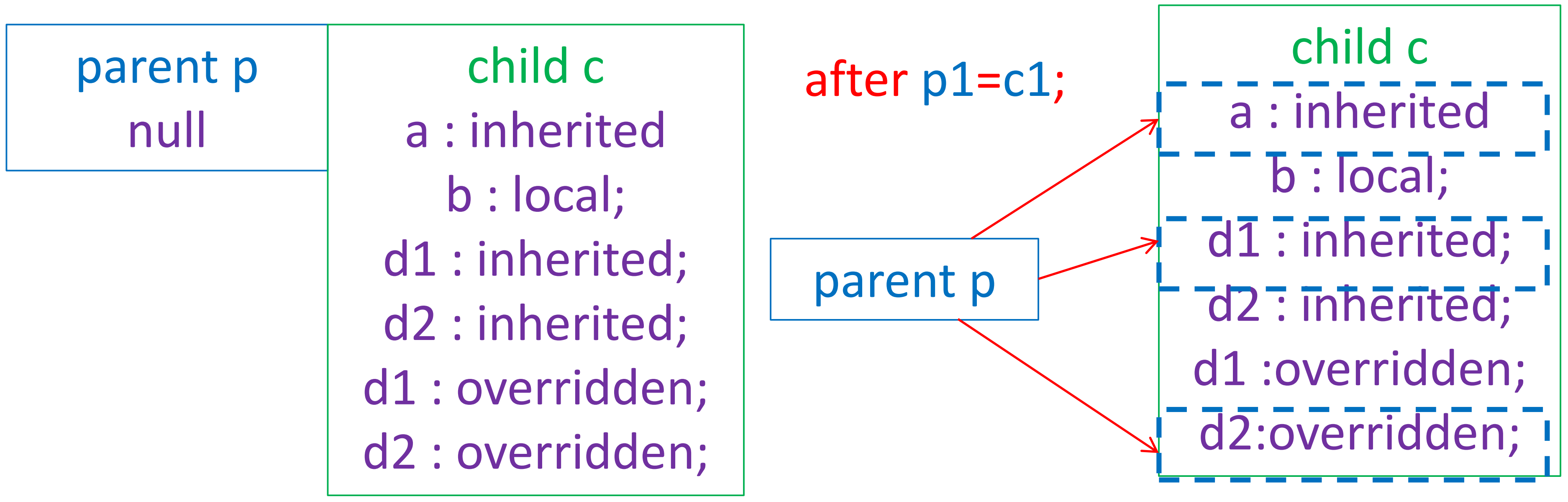
Example2

```
initial begin
parent p1; child c1;
c1=new;
$cast(p1, c1);       // checks run-time casting errors
//p1=c1;             //checks compile time casting errors
//properties and virtual methods in parent class
//points to one defined in child class
p1.d1;   p1.d2;
$display("p1.a=%0d", p1.a);  c1.a=9;
$display("p1.a=%0d", p1.a);
end
```

Result :

```
Parent d1
Child d2
p1.a=3
p1.a=9
```

Example2



parent points to child memory for inherited properties and virtual methods

Example3

```
class parent;
int a=3;

function void d1();
$display(“Parent d1”);
endfunction

virtual function void d2();
$display(“Parent d2”);
endfunction
endclass
```

```
class child extends parent;
int a=5; b=8;

function void d1();
$display(“Child d1”);
endfunction

function void d2();
$display(“Child d2”);
endfunction
endclass
```

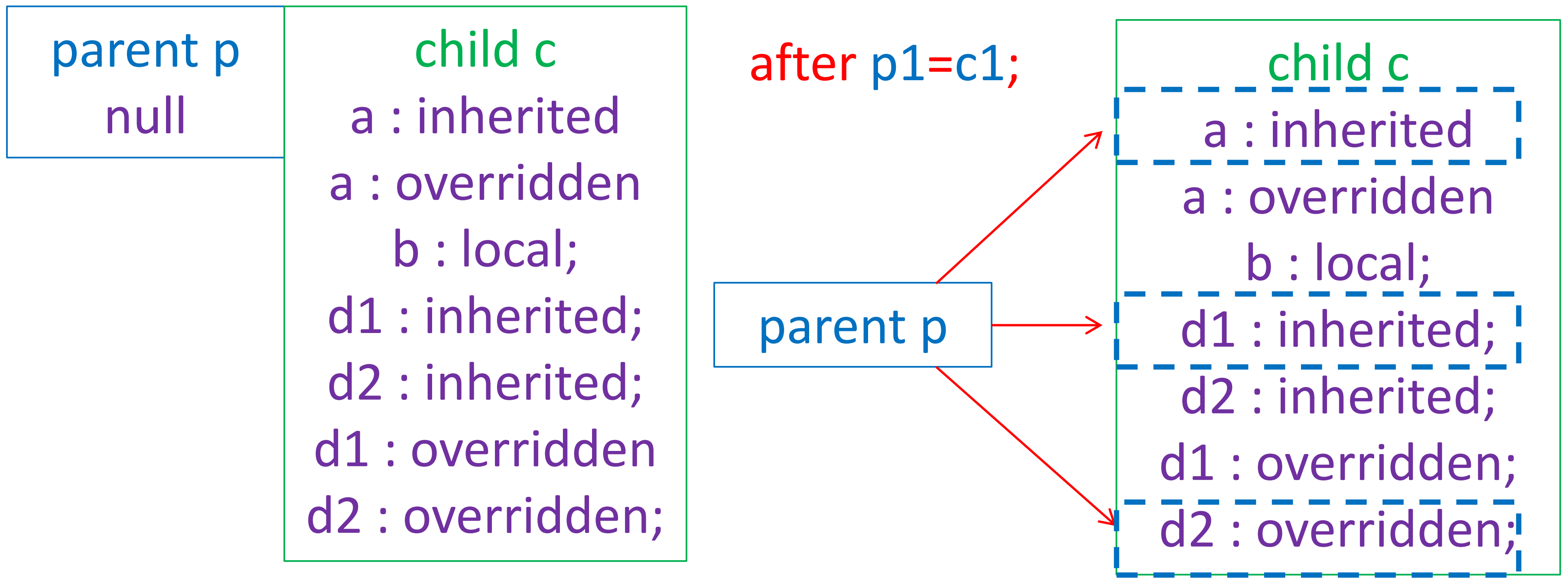
Example3

```
initial begin
parent p1; child c1;
c1=new;
p1=c1;                //Polymorphism occurs
//c1=p2;  will give compilation error
p1.d1;   p1.d2;
$display("p1.a=%0d", p1.a);  c1.a=9;
$display("p1.a=%0d", p1.a);
end
```

Result :

```
Parent d1
Child d2
p1.a=3
p1.a=3
```

Example3



Modifying parent's a will not modify child's a since it is overridden in child.

Abstraction

- Sometimes, it is useful to create a class without intending to create any objects of the class.
- The class exists simply as a base class from which other classes can be derived.
- In System Verilog this is called an abstract class and is declared by using the word virtual.
- A virtual class object can not be constructed but handle to the virtual class can be defined.

Abstraction

- Virtual methods can be declared without any body.
- These methods can be overridden in a derived class.
- The method overriding virtual method should have same signature i.e. (return type, number and type of arguments) must be the same as that of the virtual method.
- If a virtual method is defined as pure then these methods must be defined in child classes. A pure virtual method forces child classes to implement standard set of methods.

Example1

```
virtual class abstract;          //Abstract Class

virtual task display();          //Virtual Method
 endtask                         //Body not defined

function int increment(int x);
return x + 1;
endfunction
endclass
```

Example1

```
class abc extends abstract;

task display();       // display may or may not be defined
$display("abc");
endtask

function int increment(int x);  //Overriding
return x + 2;
endfunction

endclass
```

Example1

```
class xyz extends abstract;

task display();          // display may or may not be defined
$display("xyz");
endtask

//Increment function may not be defined
endclass
```

Example1

```
abstract ab;
abc a;
xyz x;

int p1, p2;
```

```
initial begin
//ab=new; not allowed will give compilation error
a=new;   x=new;
a.display;   x.display;
p1=a.increment(2);
p2=x.increment(5);
ab=x; ab.display;
ab=a; ab.display;
end
```

Results:

```
abc      xyz
4         6
xyz      abc
```

Example2

```
virtual class abstract;                    //Abstract Class

pure virtual task display();          //Pure Virtual Method

virtual function int increment(int x);          //Virtual Function
                                        //Body may not be defined

endclass
```

Example2

```
class abc extends abstract;

task display();          //display method needs to be defined
$display(“abc”);          //will give compilation error if not defined
 endtask

function int increment(int x);
 //Increment function may or may not be defined
return x + 2;
endfunction

endclass
```

Assignment-10

- **Declare a class eth_pac with following properties and methods:**
    - **payload: longint, SA:byte, DA: byte, pb: bit**
    - **Display method (virtual) to display all properties**
    - **Random method (virtual) to randomize payload**
    - **Random method(virtual) to randomize SA,DA.**
    - **Method to calculate pb by reduction and operator**
- **Create a child class good_pac from eth_pac and same methods name as parent class with following advancements**
    - **Display method to display all properties**
    - **Random method to randomize payload only between 0-100**
    - **Random method to randomize SA,DA between 2-24**
    - **Method to calculate pb by reduction xor operator**
- **Create object of parent class and child class and verify the concept of polymorphism**

- Provide the following modification in problem 1:
    - Convert the eth_pac as virtual
    - Random and display methods pure virtual
    - Now create the objects of parents and child class and verify the concept of abstraction

Reading assignment:

Object copy

Shallow copy

Deep copy