Prasthuthi Amin
Basic Coding Assignments:

```
1)program fork_join;
  initial begin
   #(10);
     $display(" BEFORE fork  time = %d ",$time );
     fork
        begin
           # (20);
           $display("time = %d # 20  ",$time );
        end
        begin
           #(10);
           $display("time = %d # 10  ",$time );
        end
        begin
           #(5);
           $display("time = %d # 5  ",$time );
        end
      join
   $display(" time = %d Outside the main fork ",$time );
  end
endprogram
```

Output:

```
BEFORE fork  time = 10
time = 15 # 5
time = 20 # 10
time = 30 # 20
time = 30 Outside the main fork
```

```
2)program fork_join_any;
  initial begin
   #(10);
     $display(" BEFORE fork  time = %d ",$time );
     fork
        begin
           # (20);
           $display("time = %d # 20  ",$time );
        end
        begin
           #(10);
           $display("time = %d # 10  ",$time );
        end
        begin
           #(5);
           $display("time = %d # 5  ",$time );
        end
      join_any
   $display(" time = %d Outside the main fork ",$time );
  end
  endprogram
```

Output:

```
BEFORE fork  time = 10
time = 15 # 5
time = 15 Outside the main fork
```

```
3)program fork_join_none;
  initial
  begin
   #10;
     $display(" BEFORE fork  time = %d ",$time );
     fork
        begin
           # (20);
```

```
       $display("time = %d # 20  ",$time );
     end
     begin
       #(10);
       $display("time = %d # 10  ",$time );
     end
     begin
       #(5);
       $display("time = %d # 5  ",$time );
     end
    join_none
  $display(" time = %d Outside the main fork ",$time );
 end
endprogram
```

Output:

```
BEFORE fork  time = 10
time = 10 Outside the main fork
```

```
4)module packed_and_unpacked();
// ADD_CODE:declare a packed array packed_array of 8 bits and initialize it to 8'hAA
// ADD_CODE:declare an unpacked array unpacked_array of 8 bits and initialize it to '{0,0,0,0,0,0,0,1}
initial
 begin
  //ADD_CODE:display the 0th element of the packed array
  //ADD_CODE:display the 0th element of the unpacked array
  //ADD_CODE:display the whole packed array. Is it possible???
  //ADD_CODE:display the whole unpacked array. Is it possible???
  #1 $finish;
end
endmodule
```

Ans:

```
module packed_and_unpacked();
bit [7:0]packed_array=8'hAA;
bit unpacked_array[0:7]='{0,0,0,0,0,0,0,1};
initial
begin
$display("0th element of the packed_array is %0b",packed_array[0]);
$display("0th element of the unpacked_array is %0p",unpacked_array[0]);
$display("whole packed_array is %0b",packed_array);
$display("whole unpacked_array is %0p",unpacked_array);
  #1 $finish;
end
endmodule
```

Output:

```
0th element of the packed_array is 0
0th element of the unpacked_array is 0
whole packed_array is 10101010
whole unpacked_array is 0 0 0 0 0 0 0 1
```

```
5)module basic_data_types_simulation();
  // Declaring and initializing the variables
  bit [ 7:0] data = 8'b0101_01xz;
  logic [ 7:0] address = 8'b010z_01xz;
  integer write_addr = 32'b01x1_01xz_01xz_01xz;
  int read_addr = 32'b01x0_01xz_01xz_01xz;
  byte wr_enable = 8'b0101_01xz;
  reg rd_enable = 8'b0101_01xz;
initial
 begin

  // Displaying the values of the variables for different data types
  $display ("Showing outputs for different datatypes:\n");
  $display ("Value of bit data = %b\n", data);
  $display ("Value of logic address = %b\n", address);
  $display ("Value of integer write address = %b\n", write_addr);
  $display ("Value of int read address = %b\n", read_addr);
  $display ("Value of byte wr_enable = %b\n", wr_enable);
  $display ("value of reg rd_enable = %b\n", rd_enable);
  $display ("Output for bit + integer is = %b\n", data + address);
  $display ("Output for logic + int is = %b\n", write_addr + read_addr);

  // Re-assigning the variables for the different data types
  data = 10;
  address = 20;
  write_addr = 30;
  read_addr = 40;
  wr_enable = 16'habcx;
  rd_enable = 16'habcx;

  // Displaying the values of the variables for different data types after re-assigning
   $display ("After changing values, output for bit + logic is = %b\n", data +
address);
  $display ("After changing values, output for integer + int is = %b\n", write_addr
+ read_addr);
  $display ("After changing values of byte wr_enable = %b\n", wr_enable);
  $display ("After changing values of reg rd_enable = %b\n", rd_enable);
 end
endmodule
```

Output:

```
Showing outputs for different datatypes:
Value of bit data = 01010100
Value of logic address = 010z01xz
Value of integer write address = 000000000000000001x101xz01xz01xz
Value of int read address = 00000000000000000100010001000100
Value of byte wr_enable = 01010100
value of reg rd_enable = z
Output for bit + integer is = xxxxxxxx
Output for logic + int is = xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

```
After changing values, output for bit + logic is = 00011110
After changing values, output for integer + int is = 00000000000000000000000001000110
After changing values of byte wr_enable = 11000000
After changing values of reg rd_enable = x
```

```
6)module queues();
// ADD_CODE:Declare a local variable i of type int for manipulation and initialize it to 1
// ADD_CODE:Declare a queue "b" of type int and initialize it to {3,4}
// ADD_CODE:Declare a queue "q" of type int and initialize it to {0,2,5}
initial
  begin
// ADD_CODE:Insert (1,j) into the queue q and display q using %p
// ADD_CODE:Insert (3,b[$]) into the queue q and display q using %p
// ADD_CODE:delete element (1) from the queue q and display q using %p
// ADD_CODE:push front (6) into the queue q and display q using %p
// ADD_CODE:pop back from the queue q, store the value in j and display j
// ADD_CODE:push back (8) into the queue q and display q using %p
// ADD_CODE:pop front from the queue q, store the value in j and display j
    end
endmodule
```

Ans:

```
module queues();
  int i=1;
  int j;
  int b[$]={3,4};
  int q[$]={0,2,5};

initial
  begin
      q.insert(1,"j");
    $display("size of q =%d",q.size);
    foreach(q[i])
```

```
      $display("q[%0p]=%p",i,q[i]);
    q.insert(3,b[$]);
    $display("size=%d",q.size);
    foreach(q[i])
      $display("q[%0p]=%p",i,q[i]);
        q.delete(1);
     $display("size=%d",q.size);
    foreach(q[i])
       $display("q[%0p]=%p",i,q[i]);
        q.push_front (6);
     $display("size=%d",q.size);
    foreach(q[i])
      $display("q[%0p]=%p",i,q[i]);
      j=  q.pop_back();
    $display("j=%d",j);
        q.push_back(8);
    foreach(q[i])
      $display("q[%0d]=%0p",i,q[i]);
      j= q.pop_front();
    $display("j=%d",j);
    end
endmodule
```

Output:

```
size of q = 4
q[0]=0
q[1]=106
q[2]=2
q[3]=5
size= 5
q[0]=0
q[1]=106
q[2]=2
q[3]=4
q[4]=5
```

```
size= 4
q[0]=0
q[1]=2
q[2]=4
q[3]=5
size= 5
q[0]=6
q[1]=0
q[2]=2
q[3]=4
q[4]=5
j= 5
q[0]=6
q[1]=0
q[2]=2
q[3]=4
q[4]=8
j= 6
```

```
7)module dynamic_array();
// ADD_CODE:Declare a dynamic array mem of 8 bits
initial begin
// ADD_CODE:Allocate the dynamic array for 4 locations
  $display ("Setting array size to 4");
 // ADD_CODE:Initialize the array with 0,1,2,3 values
  $display("Initialize the array with 0,1,2,3 values");
  // ADD_CODE:Doubling the size of dynamic array, with old content still valid
  // ADD_CODE:Display the current size of the dynamic array
  // ADD_CODE:Display the each value and the location of the dynamic array
  // ADD_CODE:Delete all the elements in the dynamic array
  // ADD_CODE:Display the current size of the dynamic array
  #1 $finish;
end
```

```
endmodule
```

Ans:

```
module dynamic_array();
   bit [7:0]mem[];

initial begin
  mem = new[4];
  $display ("Setting array size to 4");
  mem ={0,1,2,3};
  $display("Initialize the array with 0,1,2,3 values");
  foreach(mem[i])
    begin
  $display("elements are %d",mem[i]);
    end
  mem = new[8](mem);
    $display("size of the array=%d",mem.size);
     foreach(mem[i])
    begin
      $display("mem[%0d]=%0d",i,mem[i]);
    end
  mem.delete;
   $display("size of the array=%d",mem.size);
  #1 $finish;
end
endmodule
```

Output:

```
Setting array size to 4
Initialize the array with 0,1,2,3 values
elements are 0
elements are 1
elements are 2
elements are 3
size  of the array = 8
```

```
mem[0]=0
mem[1]=1
mem[2]=2
mem[3]=3
mem[4]=0
mem[5]=0
mem[6]=0
mem[7]=0
size of the array= 0
```

```
8)module associative_array ();
// ADD_CODE:Declare an associative array as_mem of type int and index type int
// ADD_CODE:Declare a local variable i of type int for manipulation
initial begin
// ADD_CODE:Add element to the associative array as follows:
// in the 100th location store value 101
// in the first location store value 100
// in the 50th location store value 99
// in the 256th location store value 77
// ADD_CODE:Display the size of the associative array
// ADD_CODE:Check if index 2 exists
// ADD_CODE:Check if index 100 exists
// ADD_CODE: Display the value stored in first index
// ADD_CODE:Display the value stored in last index
// ADD_CODE:Delete the first index
// ADD_CODE:Display the value stored in first index
 #1 $finish;
end
endmodule
```

Ans:

```
module associative_array ();
  int as_mem[int];
  int i;
  initial begin
  as_mem[100]=101;
  as_mem[0]=100;
as_mem[50]=99;
as_mem[256]=77;
    foreach(as_mem[i])
    begin
      $display("as_mem[%0d]=%0d",i,as_mem[i]);
    end
  $display("size of the array ",as_mem.size);
as_mem.exists(2);
    $display(" checks if index 2 exists=%d", as_mem[2]);
as_mem.exists(100);
  $display(" checks if index 100 exists=%d", as_mem[100]);
as_mem.first(i);
  $display("the value stored in first index",as_mem[i]);
as_mem.last(i);
  $display("the value stored in last index",as_mem[i]);
as_mem.delete(0);
  $display("Delete the first index=%d",as_mem[0]);
     foreach(as_mem[i])
    begin
      $display("as_mem[%0d]=%0d",i,as_mem[i]);
    End
as_mem.first(i);
  $display("the value stored in first index=%d",as_mem[1]);
   #1 $finish;
end
endmodule
```

Output:

```
as_mem[0]=100
as_mem[50]=99
as_mem[100]=101
```

```
as_mem[256]=77
size of the array 4
checks if index 2 exists= 0
checks if index 100 exists= 101
the value stored in first index 100
the value stored in last index 77
Delete the first index= 0
as_mem[50]=99
as_mem[100]=101
as_mem[256]=77
the value stored in first index= 0
```

9)Analyze the code for fork join, fork join any, fork join none
At what time clock is equal to 1 for the code below

```
initial

begin

clk =0;

#5

fork

#5 a = 0;

#10 b = 0;

join

clk= 1;

end
```

Ans: #15 clk=1

```
initial
 begin
 clk =0;
 #5
 fork
 #5 a = 0;
 #10 b = 0;
 join_any
 clk= 1;
 end
```

Ans: #10 clk=1

```
initial
 begin
 clk =0;
 #5
 fork
 #5 a = 0;
 #10 b = 0;
 join_none
 clk= 1;
 end
```

Ans: #5 clk=1

10) Description: Concept of class inheritance and constructor with example

//ADD_CODE: Write a class called base with property "value" of type int explicitly override the class constructor - function new() in the class base and initialize the variable value to 3 inside the function new()

ADD_CODE: Write a class called ext which is extended from class base with property "x" of type int explicitly override the class constructor - function new() in the class ext and initialize the variable "x" to 5 inside the function new()

```
program constructor1;
  initial
   begin
//ADD_CODE: Declare and create object for handle "e" of the class "ext"
//ADD_CODE: Display the variables "value" and "x" using the object "e"
    end
endprogram
```

Ans:

```
class base;
 int value;
          function new();
                    value=3;
          endfunction
endclass


class ext extends base;
int x;
```

```
    function new();
        x=5;
    endfunction
endclass


program constructor1;
initial
begin
  ext e=new();
  $display("value=%0d, x=%0d",e.value,e.x);
 end
endprogram
```

Output:

```
value=3, x=5
```

11) Description: Concept of classes data type with example

```
//ADD_CODE: Declare a class called "simple" with properties i and j of int data type and write a task called printf to display the properties i and j of the class
program simple_class;
 initial
  begin
   //ADD_CODE: Declare two handles to the class simple as obj_1 and obj_2
  //ADD_CODE: Create object for the handles obj_1 and obj_2
```

```
    //ADD_CODE: Access property i using obj_1 and initialize it to 2 and  Access property i using obj_2 and initialize it to 4

   //ADD_CODE: Call the task printf using obj_1 and obj_2

    end

endprogram
```

Ans:

```
class simple;

  int i,j;

   task printf();

    begin

      $display("properties of %0d and %0d",i,j);

    end

  endtask

endclass

program simple_class;

 initial

   begin

      simple obj_1;

      simple obj_2;

      obj_1=new();

      obj_2=new();
      obj_1.i=2;

      obj_2.j=4;

      obj_1.printf();

      obj_2.printf();

   end

endprogram
```

Output:

```
properties of 2 and 0

properties of 4 and 0
```

12) Description: Concept of polymorphism with example

```
program polymorphism;
```

// ADD_CODE: Write a class called "Packet" with property "data" of 32 bits and function "send" with return type int and expecting argument "data" of 32 bits. Inside the function "send" display "SENDING BASE PACKET"This is a Packet class, defining that there will be different types of packets to be sent

// ADD_CODE: Write a class called "Ethernet_packet" with property "ether_data" of 32 bits and function "send" with return type int and expecting argument "ether_data" of 32 bits. Inside the function "send" display "SENDING ETHERNET PACKET"

// ADD_CODE: Write a class called "Token_packet" with property "token_data" of 32 bits and function "send" with return type int and expecting argument "token_data" of 32 bits. Inside the function "send" display "SENDING ETHERNET PACKET"

//ADD_CODE: Declare an array of 10 handles for Packet class as pkts[10]

```
//ADD_CODE: Declare an handle for Ethernet_packet class as ep and create object for it

//ADD_CODE: Declare an handle for Token_packet class as tp and create object for it

  initial

  begin

// ADD_CODE: Make the base class handles point to objects "ep" and "tp" i.e. pkt[0]  points to ether packet and pkt[1] points to token packet pkts[0].send(); is the same as calling ep.send(), but the neat thing here is that a BFM only needs the base class handle, and doesn't need to be modifed if the functionality or data features change!!

// ADD_CODE: Call function "send" using handles pkts[0] and pkts[1], Also pass the value for "data" in the function's argument list

  end

endprogram
```

Ans:

```
class Packet;

bit [31:0]data;

 function int send(bit [31:0] data);

      $display("SENDING BASE PACKET");

      return 0;

  endfunction

endclass

class Ethernet_packet extends Packet;
```

```
bit [31:0] ether_data;
  function int send(bit [31:0] ether_data);
      $display("SENDING ETHERNET PACKET");
      return 0;
  endfunction
endclass
class Token_packet extends Packet;
bit [31:0] token_data;
  function int send(bit [31:0] token_data);
    $display("SENDING TOKEN PACKET");
      return 0;
  endfunction
endclass
module top;
  Packet pkts[10];
  Ethernet_packet ep=new();
  Token_packet tp=new();
  initial begin
    pkts[0]=ep;
    pkts[1]=tp;

    pkts[0].send(0);
    pkts[1].send(0);
  end
endmodule
```

Output:

```
SENDING BASE PACKET
SENDING BASE PACKET
```

13)Description: Concept of shallow copy with example program shallow_copy();

```
//ADD_CODE: Write a class "A" with property "j" of type int and initialize it to 5
//ADD_CODE: Write a class "B" with properties "i" of type int and initialize it to 1 and declare handle "a" for class "A" and create object for it
 initial
  begin
//ADD_CODE: Declare a handle "b1" for class "B" and Create an object for it
//ADD_CODE: Declare a handle "b2" for class "B"
//ADD_CODE: Make a shallow copy of "b1" to "b2"
//ADD_CODE: Display "b1.i, b2.i, b1.a.j, b2.a.j"
//ADD_CODE: Now change the value of "i" in "b2" to 10
//ADD_CODE: Display "b1.i, b2.i, b1.a.j, b2.a.j"
//ADD_CODE: Now change the value of "j" in "b2.a" to 50
//ADD_CODE: Display "b1.i, b2.i, b1.a.j, b2.a.j"
  end
endprogram
```

Ans:

```
program top;


class A;
  int j=5;
endclass




class B;
  int i=1;
    A a=new();
endclass


  initial begin
    B b1,b2;
    b1=new();
    b2= new b1;
      $display(" b1.i = %d, b2.i = %d, b1.a.j = %d, b2.a.j = %d", b1.i, b2.i, b1.a.j, b2.a.j);


    b2.i=10;
      $display(" b1.i = %d, b2.i = %d, b1.a.j = %d, b2.a.j = %d", b1.i, b2.i, b1.a.j, b2.a.j);
```

```
   b2.a.j=50;
     $display(" b1.i = %d, b2.i = %d, b1.a.j = %d, b2.a.j = %d", b1.i, b2.i, b1.a.j, b2.a.j);
  end
endprogram
```

Output:

```
b1.i = 1, b2.i = 1, b1.a.j = 5, b2.a.j = 5
b1.i = 1, b2.i = 10, b1.a.j = 5, b2.a.j = 5
b1.i = 1, b2.i = 10, b1.a.j = 50, b2.a.j = 50
```

14)Description: Concept of super in classes with example

//ADD_CODE: Write a class called parent with a task printf to display the message " THIS IS PARENT CLASS "

//ADD_CODE: Write a class called subclass which is extended from the parent class. Write a task printf inside the class subclass and call the task printf of the class parent using super

```
program super1;
  initial
  begin
```

```
//ADD_CODE: Declare handle "s" for class subclass
//ADD_CODE: Create object for handle "s"
//ADD_CODE: Call the task printf using object "s"
    end
endprogram
```

Ans:

```
class parent;
  task printf;
    $display("THIS IS PARENT CLASS");
  endtask
endclass

class subclass extends parent;
  task printf;
    super.printf();
  endtask
endclass

module super1;
  initial
    begin
      subclass s;
      s=new();
      s.printf();
    end
endmodule
```

Output:

```
THIS IS PARENT CLASS
```