```
module data_types();

bit       data_1bit;
byte      data_8bit;
shortint  data_16bit;
int       data_32bit;
longint   data_64bit;
integer   data_integer;

bit       unsigned data_1bit_unsigned;
byte      unsigned data_8bit_unsigned;
shortint  unsigned data_16bit_unsigned;
int       unsigned data_32bit_unsigned;
longint   unsigned data_64bit_unsigned;
integer   unsigned data_integer_unsigned;

initial begin
   data_1bit   = {32{4'b1111}};
   data_8bit   = {32{4'b1111}};
   data_16bit  = {32{4'b1111}};
   data_32bit  = {32{4'b1111}};
   data_64bit  = {32{4'b1111}};
   data_integer= {32{4'b1111}};
   $display("data_1bit    = %0d",data_1bit);
   $display("data_8bit    = %0d",data_8bit);
   $display("data_16bit   = %0d",data_16bit);
   $display("data_32bit   = %0d",data_32bit);
   $display("data_64bit   = %0d",data_64bit);
   $display("data_integer = %0d",data_integer);
   data_1bit   = {32{4'bzx01}};
   data_8bit   = {32{4'bzx01}};
   data_16bit  = {32{4'bzx01}};
   data_32bit  = {32{4'bzx01}};
   data_64bit  = {32{4'bzx01}};
   data_integer= {32{4'bzx01}};
   $display("data_1bit    = %b",data_1bit);
   $display("data_8bit    = %b",data_8bit);
   $display("data_16bit   = %b",data_16bit);
   $display("data_32bit   = %b",data_32bit);
   $display("data_64bit   = %b",data_64bit);
   $display("data_integer = %b",data_integer);
   data_1bit_unsigned   = {32{4'b1111}};
   data_8bit_unsigned   = {32{4'b1111}};
   data_16bit_unsigned  = {32{4'b1111}};
   data_32bit_unsigned  = {32{4'b1111}};
   data_64bit_unsigned  = {32{4'b1111}};
   data_integer_unsigned  = {32{4'b1111}};
   $display("data_1bit_unsigned  = %d",data_1bit_unsigned);
   $display("data_8bit_unsigned  = %d",data_8bit_unsigned);
   $display("data_16bit_unsigned = %d",data_16bit_unsigned);
   $display("data_32bit_unsigned = %d",data_32bit_unsigned);
   $display("data_64bit_unsigned = %d",data_64bit_unsigned);
   $display("data_integer_unsigned = %d",data_integer_unsigned);
   data_1bit_unsigned   = {32{4'bzx01}};
   data_8bit_unsigned   = {32{4'bzx01}};
   data_16bit_unsigned  = {32{4'bzx01}};
   data_32bit_unsigned  = {32{4'bzx01}};
   data_64bit_unsigned  = {32{4'bzx01}};
   data_integer_unsigned  = {32{4'bzx01}};
   $display("data_1bit_unsigned  = %b",data_1bit_unsigned);
   $display("data_8bit_unsigned  = %b",data_8bit_unsigned);
   $display("data_16bit_unsigned = %b",data_16bit_unsigned);
   $display("data_32bit_unsigned = %b",data_32bit_unsigned);
   $display("data_64bit_unsigned = %b",data_64bit_unsigned);
   $display("data_integer_unsigned = %b",data_integer_unsigned);
   #1 $finish;
end

endmodule
```

STRING

```
module string_ex ();

string my_string = "This is a orginal string";
string my_new_string;

initial begin
  $display ("My String = %s",my_string);
  // Assign new string of different size
  my_string = "This is new string of different length";
  $display ("My String = %s",my_string);
  // Change to uppercase and assign to new string
  my_new_string = my_string.toupper();
  $display ("My New String = %s",my_new_string);
  // Get the length of sting
  $display ("Length of new string %0d",my_new_string.len());
  // Compare variable to another variable
  if (my_string.tolower() == my_new_string.tolower()) begin
    $display("String Compare matches");
  end
  // Compare variable to variable
  if (my_string.toupper() == my_new_string) begin
    $display("String Variable Compare matches");
  end
  #1 $finish;
end

endmodule
```

LOOPS

Dowhile

```
module while_loop ();

byte a = 0;

initial begin
  do begin
    $display ("Current value of a = %g", a);
    a ++;
  end while  (a < 10);
  #1 $finish;
end

endmodule
```

Forloop

```
module for_loop ();

initial begin
  fork
    for (int i = 0 ; i < 4; i ++) begin
      #1 $display ("First  -> Current value of i = %g", i);
    end
    for (int i = 4 ; i > 0; i --) begin
      #1 $display ("Second -> Current value of i = %g", i);
    end
  join
  #1 $finish;
end

endmodule
```

Foreach

```
module foreach_loop ();

byte a [10] = '{0,6,7,4,5,66,77,99,22,11};

initial begin
  foreach (a[i]) begin
    $display ("Value of a is %g",i);
  end
  #1 $finish;
end

endmodule
```

FORK JOIN

Fork Join all

```
module fork_join_all_process();

task automatic print_value;
  input [7:0] value;
  input [7:0] delay;
  begin
    #(delay) $display("@%g Passed Value %d Delay %d",
      $time, value, delay);
  end
endtask

initial begin
  fork
    #1 print_value (10,7);
    #1 print_value (8,5);
    #1 print_value (4,2);
  join
  $display("@%g Came out of fork-join", $time);
  #20 $finish;
end
```

```
endmodule
```

Forkjoin any

```
module fork_join_any_process();

task automatic print_value;
  input [7:0] value;
  input [7:0] delay;
  begin
    #(delay) $display("@%g Passed Value %d Delay %d",
      $time, value, delay);
  end
endtask

initial begin
  fork
    #1 print_value (10,7);
    #1 print_value (8,5);
    #1 print_value (4,2);
  join_any
  $display("@%g Came out of fork-join", $time);
  #20 $finish;
end

endmodule
```

Forkjoin None

```
module fork_join_none_process();

task automatic print_value;
  input [7:0] value;
  input [7:0] delay;
  begin
    #(delay) $display("@%g Passed Value %d Delay %d",
      $time, value, delay);
  end
endtask

initial begin
  fork
    #1 print_value (10,7);
    #1 print_value (8,5);
    #1 print_value (4,2);
  join_none
  $display("@%g Came out of fork-join", $time);
  #20 $finish;
end

endmodule
```

ENUM

```
module enum_data();

enum integer {IDLE=0, GNT0=1, GNT1=2} state;
enum {RED,GREEN,ORANGE} color;
enum {BRONZE=4, SILVER, GOLD} medal;

// a=0, b=7, c=8
enum {a, b=7, c} alphabet;
// Width declaration
enum bit [3:0] {bronze='h1, silver, gold='h5} newMedal;
// Using enum in typedef
typedef enum { red, green, blue, yellow, white, black } Colors;

Colors Lcolors;

initial begin
  state = IDLE;
  color = RED;
  medal = BRONZE;
  alphabet = c;
  newMedal = silver;
  Lcolors = yellow;
  $display (" state = %0d", state);
  $display (" color = %s", color.name());
  $display (" medal = %s", medal.name());
  $display (" alphabet = %s", alphabet.name());
  $display (" newMedal = %s", newMedal.name());
  $display (" Lcolors = %s", Lcolors.name());
end

endmodule
```

Arrays

Packed Unpacked Array

```
module packed_unpacked_data();

// packed array
bit [7:0] packed_array = 8'hAA;
// unpacked array
reg unpacked_array [7:0] = '{0,0,0,0,0,0,0,1};

initial begin
  $display ("packed array[0]   = %b", packed_array[0]);
  $display ("unpacked array[0] = %b", unpacked_array[0]);
  $display ("packed array      = %b", packed_array);
  // Below one is wrong syntax
  //$display("unpacked array[0] = %b",unpacked_array);
  #1 $finish;
end

endmodule
```

Multidimensional Arrays

```
module arrays_data();

// 2 dimension array of Verilog 2001
reg [7:0] mem [0:3] = '{8'h0,8'h1,8'h2,8'h3};
// one more example of multi dimention array
reg [7:0] mem1 [0:1] [0:3] =
   '{'{8'h0,8'h1,8'h2,8'h3},'{8'h4,8'h5,8'h6,8'h7}};
// One more example of multi dimention array
reg [7:0] [0:4] mem2 [0:1] =
   '{{8'h0,8'h1,8'h2,8'h3},{8'h4,8'h5,8'h6,8'h7}};
// One more example of multi dimention array
reg [7:0] [0:4] mem3 [0:1] [0:1]  =
   '{'{{8'h0,8'h1,8'h2,8'h3},{8'h4,8'h5,8'h6,8'h7}},
   '{{8'h0,8'h1,8'h2,8'h3},{8'h4,8'h5,8'h6,8'h7}}};
// Multi arrays in same line declaration
bit [7:0] [31:0] mem4 [1:5] [1:10], mem5 [0:255];

initial begin
  $display ("mem[0]            = %b", mem[0]);
  $display ("mem[1][0]         = %b", mem[1][0]);
  $display ("mem1[0][1]        = %b", mem1[0][1]);
  $display ("mem1[1][1]        = %b", mem1[1][1]);
  #1 $finish;
end

endmodule
```

Dynamic Arrays

```
module dynamic_array_data();

// Declare dynamic array
reg [7:0] mem [];

initial begin
  // Allocate array for 4 locations
  $display ("Setting array size to 4");
  mem = new[4];
  $display("Initial the array with default values");
  for (int i = 0; i < 4; i ++) begin
    mem[i] = i;
  end
  // Doubling the size of array, with old content still valid
  mem = new[8] (mem);
  // Print current size
  $display ("Current array size is %d",mem.size());
  for (int i = 0; i < 4; i ++) begin
    $display ("Value at location %g is %d ", i, mem[i]);
  end
  // Delete array
  $display ("Deleting the array");
  mem.delete();
  $display ("Current array size is %d",mem.size());
  #1 $finish;
end

endmodule
```

Associative Arrays

```
module integer_associative_array ();

integer as_mem [integer];

integer i;

initial begin
  // Add element array
  as_mem[100] = 101;
  $display ("value stored in 100 is %d", 101);
  as_mem[1]   = 100;
  $display ("value stored in 1   is %d", 100);
  as_mem[50]  = 99;
  $display ("value stored in 50  is %d", 99);
  as_mem[256] = 77;
  $display ("value stored in 256 is %d", 77);
  // Print the size of array
  $display ("size of array is %d", as_mem.num());
  // Check if index 2 exists
  $display ("index 2 exists   %d", as_mem.exists(2));
  // Check if index 100 exists
  $display ("index 100 exists %d", as_mem.exists(100));
  // Value stored in first index
  if (as_mem.first(i)) begin
    $display ("value at first index %d value %d", i, as_mem[i]);
  end
  // Value stored in last index
  if (as_mem.last(i)) begin
    $display ("value at last index  %d value %d", i,  as_mem[i]);
  end
  // Delete the first index
  as_mem.delete(100);
  $display ("Deleted index 100");
  // Value stored in first index
  if (as_mem.first(i)) begin
    $display ("value at first index %d value %d", i, as_mem[i]);
  end
  #1 $finish;
end

endmodule
```

Queues

```
module queue_data();

// Queue is declated with $ in array size
integer queue[$] = { 0, 1, 2, 3, 4 };
integer i;

initial begin
  $display ("Initial value of queue");
  print_queue;
  // Insert new element at begin of queue
  queue = {5, queue};
  $display ("new element added using concate");
  print_queue;
  // Insert using method at begining
  queue.push_front(6);
  $display ("new element added using push_front");
  print_queue;
  // Insert using method at end
  queue.push_back(7);
  $display ("new element added using push_back");
  print_queue;
  // Using insert to insert, here 4 is index
  // and 8 is value
  queue.insert(4,8);
  $display ("new element added using insert(index,value)");
  print_queue;
  // get first queue element method at begining
  i = queue.pop_front();
  $display ("element poped using pop_front");
  print_queue;
  // get last queue element method at end
  i = queue.pop_back();
  $display ("element poped using pop_end");
  print_queue;
  // Use delete method to delete element at index 4 in queue
  queue.delete(4);
  $display ("deleted element at index 4");
  print_queue;
  #1 $finish;
end

task print_queue;
  integer i;
  $write("Queue contains ");
  for (i = 0; i < queue.size(); i ++) begin
    $write (" %g", queue[i]);
  end
  $write("\n");
endtask

endmodule
```