D - ASSOCIATIVE ARRAYS

Associative arrays have an index that is not necessarily an integer, and can be sparsely populated. The index for an associative array is called the key, and its type is called the KeyType.

Associative arrays are declared by placing the KeyType within the [ ] of an array declaration. A simple example for associative array is shown below.

```
import std.stdio;

void main ()
{
   int[string] e;      // associative array b of ints that are

   e["test"] = 3;
   writeln(e["test"]);

   string[string] f;

   f["test"] = "Tuts";
   writeln(f["test"]);

   writeln(f);

   f.remove("test");
   writeln(f);
}
```

When the above code is compiled and executed, it produces the following result:

```
3
Tuts
["test":"Tuts"]
[]
```

Initialization

A simple initialization of associative array is shown below.

```
import std.stdio;

void main ()
{
   int[string] days =
    [ "Monday"    : 0, "Tuesday" : 1, "Wednesday" : 2,
      "Thursday" : 3, "Friday"  : 4, "Saturday"  : 5,
      "Sunday"    : 6 ];
   writeln(days["Tuesday"]);
}
```

When the above code is compiled and executed, it produces the following result:

```
1
```

Properties

Property	Description
.sizeof	Returns the size of the reference to the associative array; it is 4 in 32-bit builds and 8 on 64-bit builds.

.length	Returns number of values in the associative array. Unlike for dynamic arrays, it is read-only.
.dup	Create a new associative array of the same size and copy the contents of the associative array into it.
.keys	Returns dynamic array, the elements of which are the keys in the associative array.
.values	Returns dynamic array, the elements of which are the values in the associative array.
.rehash	Reorganizes the associative array in place so that lookups are more efficient. rehash is effective when, for example, the program is done loading up a symbol table and now needs fast lookups in it. Returns a reference to the reorganized array.
.byKey	Returns a delegate suitable for use as an Aggregate to a ForeachStatement which will iterate over the keys of the associative array.
.byValue	Returns a delegate suitable for use as an Aggregate to a ForeachStatement which will iterate over the values of the associative array.
.get*Keykey, lazyValuedefVal*	Looks up key; if it exists returns corresponding value else evaluates and returns defVal.
.remove*Keykey*	Removes an object for key.

An example for using the above properties is shown below.

```
import std.stdio;

void main ()
{
   int[string] array1;

   array1["test"] = 3;
   array1["test2"] = 20;
   writeln("sizeof: ",array1.sizeof);
   writeln("length: ",array1.length);
   writeln("dup: ",array1.dup);

   array1.rehash;
   writeln("rehashed: ",array1);

   writeln("keys: ",array1.keys);
   writeln("values: ",array1.values);

   foreach (key; array1.byKey) {
      writeln("by key: ",key);
   }

   foreach (value; array1.byValue) {
      writeln("by value ",value);
   }

   writeln("get value for key test: ",array1.get("test",10));
   writeln("get value for key test3: ",array1.get("test3",10));

   array1.remove("test");
   writeln(array1);
}
```

When the above code is compiled and executed, it produces the following result:

```
sizeof: 8
length: 2
dup: ["test2":20, "test":3]
rehashed: ["test":3, "test2":20]
keys: ["test", "test2"]
values: [3, 20]
by key: test
by key: test2
by value 3
by value 20
get value for key test: 3
get value for key test3: 10
["test2":20]
```