PERL - REFERENCES

A Perl reference is a scalar data type that holds the location of another value which could be scalar, arrays, or hashes. Because of its scalar nature, a reference can be used anywhere, a scalar can be used.

> *You can construct lists containing references to other lists, which can contain references to hashes, and so on. This is how the nested data structures are built in Perl.*

Create References

It is easy to create a reference for any variable, subroutine or value by prefixing it with a backslash as follows −

```
$scalarref = \$foo;
$arrayref  = \@ARGV;
$hashref   = \%ENV;
$coderef   = \&handler;
$globref   = \*foo;
```

You cannot create a reference on an I/O handle *filehandleordirhandle* using the backslash operator but a reference to an anonymous array can be created using the square brackets as follows −

```
 $arrayref = [1, 2, ['a', 'b', 'c']];
```

Similar way you can create a reference to an anonymous hash using the curly brackets as follows −

```
$hashref = {
 'Adam'  => 'Eve',
 'Clyde' => 'Bonnie',
};
```

A reference to an anonymous subroutine can be created by using sub without a subname as follows −

```
$coderef = sub { print "Boink!\n" };
```

Dereferencing

Dereferencing returns the value from a reference point to the location. To dereference a reference simply use $, @ or % as prefix of the reference variable depending on whether the reference is pointing to a scalar, array, or hash. Following is the example to explain the concept −

```
#!/usr/bin/perl

$var = 10;

# Now $r has reference to $var scalar.
$r = \$var;

# Print value available at the location stored in $r.
print "Value of $var is : ", $$r, "\n";

@var = (1, 2, 3);
# Now $r has reference to @var array.
$r = \@var;
```

```
# Print values available at the location stored in $r.
print "Value of @var is : ",  @$r, "\n";

%var = ('key1' => 10, 'key2' => 20);
# Now $r has reference to %var hash.
$r = \%var;
# Print values available at the location stored in $r.
print "Value of %var is : ", %$r, "\n";
```

When above program is executed, it produces the following result –

```
Value of 10 is : 10
Value of 1 2 3 is : 123
Value of %var is : key220key110
```

If you are not sure about a variable type, then its easy to know its type using **ref**, which returns one of the following strings if its argument is a reference. Otherwise, it returns false –

```
SCALAR
ARRAY
HASH
CODE
GLOB
REF
```

Let's try the following example –

```
#!/usr/bin/perl

$var = 10;
$r = \$var;
print "Reference type in r : ", ref($r), "\n";

@var = (1, 2, 3);
$r = \@var;
print "Reference type in r : ", ref($r), "\n";

%var = ('key1' => 10, 'key2' => 20);
$r = \%var;
print "Reference type in r : ", ref($r), "\n";
```

When above program is executed, it produces the following result –

```
Reference type in r : SCALAR
Reference type in r : ARRAY
Reference type in r : HASH
```

Circular References

A circular reference occurs when two references contain a reference to each other. You have to be careful while creating references otherwise a circular reference can lead to memory leaks. Following is an example –

```
#!/usr/bin/perl

 my $foo = 100;
 $foo = \$foo;

 print "Value of foo is : ", $$foo, "\n";
```

When above program is executed, it produces the following result –

```
Value of foo is : REF(0x9aae38)
```

References to Functions

This might happen if you need to create a signal handler so you can produce a reference to a function by preceding that function name with \& and to dereference that reference you simply need to prefix reference variable using ampersand &. Following is an example −

```
#!/usr/bin/perl

# Function definition
sub PrintHash{
   my (%hash) = @_;

   foreach $item (%hash){
      print "Item : $item\n";
   }
}
%hash = ('name' => 'Tom', 'age' => 19);

# Create a reference to above function.
$cref = \&PrintHash;

# Function call using reference.
&$cref(%hash);
```

When above program is executed, it produces the following result −

```
Item : name
Item : Tom
Item : age
Item : 19
```