```
DB<4> l 5
5:      my %data = (
```

In this case, there's not much to see, but of course normally there's pages of stuff to wade through, and 'l' can be very useful. To reset your view to the line we're about to execute, type a lone period '.':

```
DB<5> .
main::(./data_a:4):     my $key = 'welcome';
```

The line shown is the one that is about to be executed **next**, it hasn't happened yet. So while we can print a variable with the letter '**p**', at this point all we'd get is an empty (undefined) value back. What we need to do is to step through the next executable statement with an '**s**':

```
DB<6> s
main::(./data_a:5):     my %data = (
main::(./data_a:6):             'this' => qw(that),
main::(./data_a:7):             'tom' => qw(and jerry),
main::(./data_a:8):             'welcome' => q(Hello World),
main::(./data_a:9):             'zip' => q(welcome),
main::(./data_a:10):    );
```

Now we can have a look at that first ($key) variable:

```
DB<7> p $key
welcome
```

line 13 is where the action is, so let's continue down to there via the letter '**c**', which by the way, inserts a 'one-time-only' breakpoint at the given line or sub routine:

```
DB<8> c 13
All OK
main::(./data_a:13):    print "$data{$key}\n";
```

We've gone past our check (where 'All OK' was printed) and have stopped just before the meat of our task. We could try to print out a couple of variables to see what is happening:

```
DB<9> p $data{$key}
```

Not much in there, lets have a look at our hash:

```
DB<10> p %data
Hello Worldziptomandwelcomejerrywelcomethisthat

DB<11> p keys %data
Hello Worldtomwelcomejerrythis
```

Well, this isn't very easy to read, and using the helpful manual (**h h**), the '**x**' command looks promising:

```
DB<12> x %data
0  'Hello World'
1  'zip'
2  'tom'
3  'and'
4  'welcome'
5  undef
6  'jerry'
7  'welcome'
8  'this'
9  'that'
```