Subject: modifying hash/dict elements
Posted by JDS on Fri, 23 May 2014 13:23:25 GMT
View Forum Message <> Reply to Message

I wonder why this doesn't work:

```
IDL> h=hash()
IDL> h['foo']++
% Type conversion error: Unable to convert given STRING to Long64.
```

nor:

```
IDL> h=dictionary()
IDL> h.foo++
% Data of this type cannot be used with increment or decrement operators:
  POINTER.
```

nor do keys auto-instantiate when referenced:

```
IDL> h.foo+=1
% Key does not exist: "FOO"
```

and yet:

```
IDL> h.foo=1
IDL> h.foo+=1
```

work fine.

I think dicts/hashes should auto-subtantiate keys on any left-hand access, not just assignment, and that all elements of dicts/hashes should be available for any left-hand operation, including increment/decrement.

JD

Subject: Re: modifying hash/dict elements
Posted by chris_torrence@NOSPAM on Fri, 23 May 2014 16:02:35 GMT
View Forum Message <> Reply to Message

Hi JD,

Thanks for the feedback. Regarding the increment ++ and decrement -- operators, these are not currently implemented for operator overloading. This is something that we could do in the future.

Regarding your other question about auto-instantiation, I'm not sure that would be a good idea. Why would it make sense to instantiate the key as an integer, as opposed to any other data value?

It seems like you should at least have to declare what type of data you want, before doing any math operations.

Also, I think it's cool enough that it automatically adds the key for you. For example:

```
IDL> h = dictionary()
IDL> h.foo = 1.0
IDL> h.foo += 1
IDL> print, h.foo
      2.00000
```

This way you can do things like create "dynamic" structures on the fly, etc.

Just to continue the discussion, are there other missing features for hash, dictionary, and orderedhash that you would like to see?

-Chris

Subject: Re: modifying hash/dict elements
Posted by JDS on Wed, 28 May 2014 23:48:57 GMT

View Forum Message <> Reply to Message

On Friday, May 23, 2014 12:02:35 PM UTC-4, Chris Torrence wrote:
> Hi JD,
>
>
>
> Thanks for the feedback. Regarding the increment ++ and decrement -- operators, these are not currently implemented for operator overloading. This is something that we could do in the future.
>
>
>
> Regarding your other question about auto-instantiation, I'm not sure that would be a good idea. Why would it make sense to instantiate the key as an integer, as opposed to any other data value?
>
>
>
> It seems like you should at least have to declare what type of data you want, before doing any math operations.
>
>
>
> Also, I think it's cool enough that it automatically adds the key for you. For example:
>
> IDL> h = dictionary()

>
> IDL> h.foo = 1.0
>
> IDL> h.foo += 1
>
> IDL> print, h.foo
>
>   2.00000
>
>
>
> This way you can do things like create "dynamic" structures on the fly, etc.
>
>
>
> Just to continue the discussion, are there other missing features for hash, dictionary, and orderedhash that you would like to see?

I think most users would expect the increment and decrement operators to work (since so much else does!). I'm probably biased by Perl's hash usage, where auto-instantiation is the norm, but in my view if you are performing a numerical (integer/float) or string operation on an undefined hash element, it would be natural to instantiate the relevant null flavor (0,0.0,''). Otherwise I end up with ugliness like:

```
  if found.hasKey(name) then found[name]+=1 else found[name]=1
```

when what I wanted was simply:

```
  found[name]++  ;)
```

I was all set to ask for hash slices, but on testing, I see you have implemented them!

```
  IDL> h=hash('a',1,'b',2,'c',3,'d',12)
  IDL> subhash=h[['a','d']]
```

And yet, for my uses they are a bit cumbersome. For example to pull out a list of numbers from a hash of type h[name]=number, you can use h.values(), but that just returns a LIST. So you need:

```
  (h.values()).toarray()
```

It occurs to me that if HASH included a toarray() method itself, like LIST, it could mean "turn the list of values of this HASH into a native IDL array, if possible". Then I could just pull arrays from hash slices in one step, like:

```
  h[slicearray].toarray()
```

Better. And yet, I'd do this so much I'd wish it were called something shorter, e.g. arr() instead of toarray(). And then, just to be really greedy, I'd want some syntactic sugar to do this for me, e.g.

just as concatenation overloads the array brackets for its own dimensional chicanery, a single element [hash] would be equivalent to hash.toarray() (losing only the fairly useless case of creating a single element array containing a hash).

Then I'd be all:

```
  plot,[h1[slicearray]],[h2[slicearray]]
```

ahhhhhh....

Thanks for your response.

JD