

---

Subject: Re: Old Timers ??

Posted by [John-David T. Smith](#) on Thu, 02 Nov 2000 16:01:16 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Eric Kihn wrote:

```
>
> Hello all,
>
> I've got some legacy code I'm trying to update and it looks like:
>
> ; Read text header from file
> ; until 'end header' reached.
> ;-----
> line = "
> header = "
> repeat begin
>   readf, unit, line
>   if (n_elements(header) eq 0) then $
>     begin
>       header = line
>     endif else $
>     begin
>       header = [header,line]
>     endelse
>   endrep until (strtrim(line,2) eq 'end header')
>
> ;-----
> ; Close file.
> ;-----
> close, unit & free_lun, unit
>
> ;-----
> ;Get relevant info from the header.
> ;-----
> dda_header.file_id = $
>   strtrim( get_keyword( header, 'file ID'), 2)
>
> This code is circa 95 and it's balking at the get_keyword() function. So my
> questions are two:
> 1) Did previous versions of IDL have a get_keyword() (I'm 5.3 now)?
> 2) It's clear this code is simulating a HASH with Arrays, does IDL 5.3 have
> native hash support of some kind?
```

1) No, but a search at

<http://www.astro.washington.edu/deutsch/idl/htmlhelp/index.html> reveals:

```
;+
; Project   : SOHO - CDS
;
```

```
; Name      : GET_KEYWORD
;
; Purpose   : Extract values in a string array that appear after
;             keyword construct such as: KEYWORD=VALUE
;             (e.g. extract all time values following
STARTIME=time_value)
;
```

Could be from that, but then again, it might be something else altogether.

2) No, sadly. The `get_keyword` routine probably does a linear search through the header line array, looking for "file ID" in this case. This is of course just the type of thing hashes were made for. The `get_keyword` routine I found just uses "where", as you might expect. This means not only is the search linear, all elements are considered, even after a match is found. It could have been made more efficient in IDL 5.4 if `array_equal()` had been implemented not just to return true as soon as a match is found, but return the index of that match. Oh well. Maybe an optional variable will be added to `array_equal` for this in 5.4.1. (hint)

It is possible to make a better simulation of a hash with a structure, but that is somewhat inflexible since adding new keywords (fields) is difficult, without copying the whole thing, though `create_struct()` can do it. Then there is the matter of indexing the structure with a variable field (the essence of what hashes do). Possible, but not efficient... in fact nearly as inefficient as a "where" on the full array. Unfortunately, it's not that easy to write a good hash type that scales well for small and large hashes, and doesn't eat too much memory.

JD

--

J.D. Smith | WORK: (607) 255-6263  
Cornell Dept. of Astronomy | (607) 255-5842  
304 Space Sciences Bldg. | FAX: (607) 255-5875  
Ithaca, NY 14853 |

---

Subject: Re: Old Timers ??  
Posted by [Paul van Delst](#) on Thu, 02 Nov 2000 16:31:30 GMT  
[View Forum Message](#) <> [Reply to Message](#)

---

"J.D. Smith" wrote:  
>

> Eric Kihn wrote:

>> 2) It's clear this code is simulating a HASH with Arrays, does IDL 5.3 have  
>> native hash support of some kind?

>

> 2) No, sadly. The `get_keyword` routine probably does a linear search  
> through the header line array, looking for "file ID" in this case. This  
> is of course just the type of thing hashes were made for. The  
> `get_keyword` routine I found just uses "where", as you might expect.  
> This means not only is the search linear, all elements are considered,  
> even after a match is found. It could have been made more efficient in  
> IDL 5.4 if `array_equal()` had been implemented not just to return true as  
> soon as a match is found, but return the index of that match. Oh well.  
> Maybe an optional variable will be added to `array_equal` for this in  
> 5.4.1. (hint)

>

> It is possible to make a better simulation of a hash with a structure,  
> but that is somewhat inflexible since adding new keywords (fields) is  
> difficult, without copying the whole thing, though `create_struct()` can  
> do it. Then there is the matter of indexing the structure with a  
> variable field (the essence of what hashes do). Possible, but not  
> efficient... in fact nearly as inefficient as a "where" on the full  
> array. Unfortunately, it's not that easy to write a good hash type that  
> scales well for small and large hashes, and doesn't eat too much  
> memory.

O.k., following the advice that "he who asks a question is a fool for 5 minutes, he who does not ask a question is a fool forever" I have to ask:

What is a hash?

I have seen the term used before regarding string searches but, being a mostly fortran number crunchy type programmer, I've never been one for searching through strings and what not.

The only dictionary definitions I found were:

- 1) To chop up meat/potatoes into small pieces
- 2) To confuse or muddle
- 3) To stick in one's pipe and smoke it.

:o)

paulv

--

Paul van Delst            Ph: (301) 763-8000 x7274  
CIMSS @ NOAA/NCEP       Fax: (301) 763-8545  
Rm.207, 5200 Auth Rd.   Email: pvandelst@ncep.noaa.gov

---

Subject: Re: Old Timers ??

Posted by [John-David T. Smith](#) on Thu, 02 Nov 2000 17:55:41 GMT

[View Forum Message](#) <> [Reply to Message](#)

---

Paul van Delst wrote:

- > O.k., following the advice that "he who asks a question is a fool for 5 minutes, he who
- > does not ask a question is a fool forever" I have to ask:
- >
- > What is a hash?
- >
- > I have seen the term used before regarding string searches but, being a mostly fortran
- > number crunchy type programmer, I've never been one for searching through strings and what
- > not.

Also known as a "named associative list", or "associative array", hashes (or "hash tables") are specialized lookup tables. You can think of them as arrays indexed, not with numeric indices, but with string values (or even other data types for some languages). In the example in this thread, a hash element would exist for each keyword... e.g.

```
hash["file ID"]="test"  
hash["otherkeyword"]=12.22
```

etc. Hashes are like infinitely malleable structures, which is why they are so nice. They are somewhat complicated to make (not for us, but for the language designers), but easy and fast to use. Here's a link I found: <http://www.math.utah.edu/~beebe/software/hash/hash.html>.

Hashes are built into Perl, which is why they are so popular these days, I think (no offense to the lisp programmers out there).

JD

--

J.D. Smith                   | WORK: (607) 255-6263  
Cornell Dept. of Astronomy |   (607) 255-5842  
304 Space Sciences Bldg.   | FAX: (607) 255-5875  
Ithaca, NY 14853           |

---