LEIDEN UNIVERSITY MEDICAL CENTER

Associative arrays

Jeroen F. J. Laros

Leiden Genome Technology Center

Department of Human Genetics

Center for Human and Clinical Genetics

LUMC

Introduction

Associative arrays.

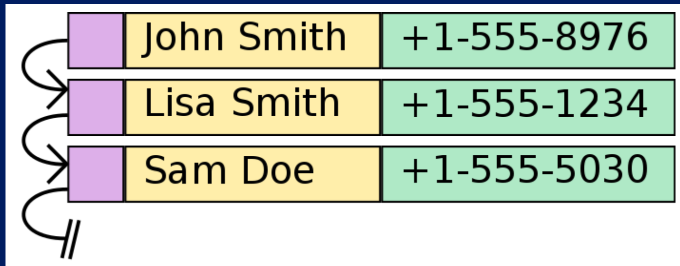

Figure 1 : A schematic view of an associative array.

LUMC

Variables

Variables in Bash and AWK.

We have seen variables before (in a **for**-loop).

```
1  for i in {1..10}; do
2    echo $i
3  done
```

Listing 1 : A variable in a for-loop (Bash).

Print the numbers 1 to 10 to the screen.

Variables

Variables in Bash and AWK.

We have seen variables before (in a **for**-loop).

```
1  for i in {1..10}; do
2    echo $i
3  done
```

Listing 1 : A variable in a for-loop (Bash).

Print the numbers 1 to 10 to the screen.

```
1  for (i = 1; i <= 10; i++)
2    print i;
```

Listing 2 : A variable in a for-loop (AWK).

LUMC

Variables

Arithmetic using variables.

We can use variables for other purposes as well.

```
1  x=10
2  y=20
3
4  echo $(($x * $y))
5  echo $((x * y))
```

Listing 3 : Multiplying two numbers (Bash).

Multiplication for example.

LUMC

Variables

Arithmetic using variables.

We can use variables for other purposes as well.

```
1  x=10
2  y=20
3
4  echo $(($x * $y))
5  echo $((x * y))
```

Listing 3 : Multiplying two numbers (Bash).

Multiplication for example.

```
1  x = 10;
2  y = 20;
3
4  print x * y;
```

Listing 4 : Multiplying two numbers (AWK).

LUMC

Variables

Many variables

Suppose we want to keep track of some properties in a population:

- Hair colour.
- Weight.

```
1 person1_hair_colour="white"
2 person1_weight=80
3 person2_hair_colour="blue"
4 person2_weight=71
```

Listing 5 : Keeping track of many people.

Variables

Many variables

Suppose we want to keep track of some properties in a population:

- Hair colour.
- Weight.

```
1    person1_hair_colour="white"
2    person1_weight=80
3    person2_hair_colour="blue"
4    person2_weight=71
```

Listing 5 : Keeping track of many people.

This is not practical if the population is large.

LUMC

Variables

Many variables

Suppose we want to keep track of some properties in a population:

- Hair colour.
- Weight.

```
1  person1_hair_colour="white"
2  person1_weight=80
3  person2_hair_colour="blue"
4  person2_weight=71
```

Listing 5 : Keeping track of many people.

This is not practical if the population is large.

And not possible if we do not know the population size in advance.

Arrays

First some terminology.

0	1	2	3	4	5	6
white	blue	gray				

Figure 2 : An array containing hair colours.

Arrays

First some terminology.

0	1	2	3	4	5	6
white	blue	gray				

Figure 2 : An array containing hair colours.

Term	Description	Example
Element	Smallest unit of the array	0: white
Index	Number of the element	0
Value	Value of the element	white
Type	Type of the content	String

Table 1 : Terminology.

LUMC

Arrays

Definition.

An *array* is a systematic arrangement of objects.

Also known as:

- Record.
- List.
- Vector.

Arrays

Definition.

An *array* is a systematic arrangement of objects.

Also known as:

- Record.
- List.
- Vector.

Arrays are *indexed*, we can ask for te n-th element.

LUMC

Arrays

Definition.

An *array* is a systematic arrangement of objects.

Also known as:

- Record.
- List.
- Vector.

Arrays are *indexed*, we can ask for te n-th element.

Going back to our example, we can make two arrays:

- Hair colour.
- Weight.

LUMC

Arrays

Indexing.

Most programming languages start counting at 0.

- C, C++, Pascal, ...
- You can not have element 2 without also having 0 and 1.

Arrays

Indexing.

Most programming languages start counting at 0.

- C, C++, Pascal, ...
- You can not have element 2 without also having 0 and 1.

Both Bash and AWK do not have much preferences.

- Gaps are allowed.
  - Element 2 and 4 may exist without element 3.
- In Bash, all indices need to be non-negative.
- In AWK, anything goes.

Arrays

Argument vector (argv).

We have seen arrays before:

```
1  #!/bin/bash
2
3  echo $1: $2
4  echo $3: $4
```

Listing 6 : The argument list in Bash.

Arrays

Argument vector (argv).

We have seen arrays before:

```
1  #!/bin/bash
2
3  echo $1: $2
4  echo $3: $4
```

Listing 6 : The argument list in Bash.

```
1  $ ./myscript hair white weight 85
2  hair: white
3  weight: 85
```

Listing 7 : Example usage of the argument list in Bash.

Many programming languages, use argv[1], argv[2], etc.

LUMC

Arrays

Array types.

Usually, an array only has elements of one *type*:

- Integer (numbers).
- Float (real valued numbers).
- String.
- …

LUMC

Arrays

Array types.

Usually, an array only has elements of one *type*:

- Integer (numbers).
- Float (real valued numbers).
- String.
- …

Not all programming languages support *mixed type* arrays.

- A mix of strings and integers is forbidden.
- C, C++, Pascal, Java do not support this.
- Python, Bash, AWK, Perl do support this.

Arrays

Element assignment and retrieval.

Array elements are referred to with square brackets.

```
1  my_array[0]=43
2  my_array[1]="hello"
```

Listing 8 : Assigning values to elements in Bash.

Arrays

Element assignment and retrieval.

Array elements are referred to with square brackets.

```
1    my_array[0]=43
2    my_array[1]="hello"
```

Listing 8 : Assigning values to elements in Bash.

To retrieve elements, we need to use additional braces (curly brackets).

```
1    $ echo ${my_array[1]}
2    hello
3    $ echo ${my_array[*]}
4    43 hello
```

Listing 9 : Retrieval of elements in Bash.

LUMC

Arrays

Element assignment and retrieval.

We can also initialise an array with one operation.

```
1    my_array=(43 "hello" 12 "world")
```

Listing 10 : Assigning multiple values at once in Bash.

The resulting array will be zero-based.

LUMC

Arrays

Element assignment and retrieval.

We can also initialise an array with one operation.

```
1    my_array=(43 "hello" 12 "world")
```

Listing 10 : Assigning multiple values at once in Bash.

The resulting array will be zero-based.

```
1    $ echo ${my_array[0]}
2    43
3    $ echo ${my_array[1]}
4    hello
```

Listing 11 : Example of element retrieval.

Arrays

Element assignment and retrieval.

In AWK, the syntax is a bit easier.

```
1    my_array[0] = 43;
2    my_array[1] = "hello";
3
4    print my_array[0];
5    print my_array[1];
```

Listing 12 : Element assignment and retrieval in AWK.

Arrays

Element assignment and retrieval.

In AWK, the syntax is a bit easier.

```
1    my_array[0] = 43;
2    my_array[1] = "hello";
3
4    print my_array[0];
5    print my_array[1];
```

Listing 12 : Element assignment and retrieval in AWK.

We can initialise an array by using the split function.

```
1    split("43 hello 12 world", my_array)
```

Listing 13 : Assigning multiple values at once.

The resulting array will be one-based.

Arrays

Looping through the values.

We frequently need to go through the entire array.

- Search for a particular value.
- Printing the content of the array.

LUMC

Arrays

Looping through the values.

We frequently need to go through the entire array.

- Search for a particular value.
- Printing the content of the array.

```
1 for value in ${my_array[*]}; do
2   print value
3 done
```

Listing 14 : Retrieve all values from an array in Bash.

LUMC

Arrays

Looping through the values.

We frequently need to go through the entire array.

- Search for a particular value.
- Printing the content of the array.

```
1   for value in ${my_array[*]}; do
2     print value
3   done
```

Listing 14 : Retrieve all values from an array in Bash.

The same thing, but now in AWK.

```
1   for (i in my_array)
2     print my_array[i];
```

Listing 15 : Retrieve all values from an aray in AWK.

Associative arrays

Definition.

An *associative array* is an *abstract data type* composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection.

Also known as:

- Dictionary.
- Hash.
- Map.
- Symbol table.

LUMC

Associative arrays

Definition.

An *associative array* is an *abstract data type* composed of a collection of (key, value) pairs, such that each possible key appears at most once in the collection.

Also known as:

- Dictionary.
- Hash.
- Map.
- Symbol table.

Think of a real dictionary.

Associative arrays.

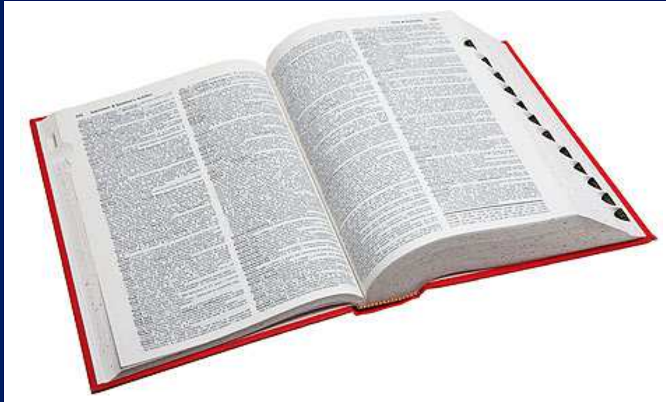

Figure 3 : Indexed books are easy to search.

LUMC

Associative arrays

Declaration of an associative array.

In Bash, we need to mention explicitly that we want an associative array.

```
1 declare -A my_array
2
3 my_array[hello]=world
```

Listing 16 : Tell Bash to make an associative array.

LUMC

Associative arrays

Declaration of an associative array.

In Bash, we need to mention explicitly that we want an associative array.

```
1   declare -A my_array
2
3   my_array[hello]=world
```

Listing 16 : Tell Bash to make an associative array.

We can also initialise the associative array at once.

```
1   my_array=([hello]=world [43]=12)
```

Listing 17 : Assigning multiple values at once in Bash.

Element retrieval is similar, but now we can use words.

Associative arrays

Retrieve the entire array.

Again, we can use the [*] construct to get all values.

```
1  $ echo ${my_array[*]}
2  world 12
```

Listing 18 : Retrieve all values.

Associative arrays

Retrieve the entire array.

Again, we can use the [*] construct to get all values.

```
1   $ echo ${my_array[*]}
2   world 12
```

Listing 18 : Retrieve all values.

To retrieve the keys, we need to use an additional exclamation mark.

```
1   $ echo ${!my_array[*]}
2   hello 43
```

Listing 19 : Retrieve all kays.

Associative arrays

Looping through associative arrays.

With the following code snippets we can print all keys and their associated values.

```
1  for key in ${!my_array[*]}; do
2    echo $key: ${my_array[$key]}
3  done
```

Listing 20 : Print the associative array in Bash.

Associative arrays

Looping through associative arrays.

With the following code snippets we can print all keys and their associated values.

```
1 for key in ${!my_array[*]}; do
2     echo $key: ${my_array[$key]}
3 done
```

Listing 20 : Print the associative array in Bash.

With as result:

```
1 hello: world
2 43: 12
```

Listing 21 : Output of the previous script.

Associative arrays

Looping through associative arrays.

In AWK this is exactly like looping through normal arrays.

```
for (i in my_array)
  print i ": " my_array[i];
```

Listing 22 : Print the associative array in AWK.

The reason for this is that AWK actually only has associative arrays.

LUMC

Examples

Putting it all together.

In the following example, we count the occurrence of letters.

```
1   declare -A my_array
2
3   for i in a a a a b b b b b b c; do
4     if [ ! ${my_array[$i]} ]; then
5       my_array[$i]=0
6     fi
7     my_array[$i]=$((${my_array[$i]} + 1))
8   done
9
10  for i in ${!my_array[*]}; do
11    echo $i: ${my_array[$i]}
12  done
```

Listing 23 : Count the number of a's, b's and c's.

LUMC

Examples

Putting it all together.

The same example, but now using AWK.

```
1    split("a a a a b b b b b b c", a);
2
3    for (i in a) {
4      if (! my_array[a[i]])
5        my_array[a[i]] = 0;
6      my_array[a[i]]++;
7    }
8
9    for (i in my_array)
10     print i ": " my_array[i];
```

Listing 24 : Count the number of a's, b's and c's.

Additional notes

Performance.

If the implementation is good, searching is very fast.

Method	$\ell = 1000$	$\ell = 100000$	Complexity
Unsorted list	1000	100000	$\mathcal{O}(\ell)$
Sorted list	10	17	$\mathcal{O}(\log(\ell))$
Hash	1	1	$\mathcal{O}(1)$

Table 2 : Number of operations when searching for stored values.

https://en.wikipedia.org/wiki/Computational_complexity_theory

LUMC

Additional notes

Performance.

If the implementation is good, searching is very fast.

Method	$\ell = 1000$	$\ell = 100000$	Complexity
Unsorted list	1000	100000	$\mathcal{O}(\ell)$
Sorted list	10	17	$\mathcal{O}(\log(\ell))$
Hash	1	1	$\mathcal{O}(1)$

Table 2 : Number of operations when searching for stored values.

The $\mathcal{O}$ notation describes the order of complexity.

https://en.wikipedia.org/wiki/Computational_complexity_theory

LUMC

Additional notes

Multidimensional arrays.

A multidimentional array is an array of arrays.

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 3 & 4 & 5 \\ 3 & 4 & 5 & 6 \\ 4 & 5 & 6 & 7 \end{pmatrix}$$

Table 3 : A 2-dimensional matrix.

Both for Bash and AWK: it is possible, but it gets very complicated. Better use a high level programming language.

LUMC

Questions?

Acknowledgements:

Magnus Palmblad
Rob Marissen
Michiel van Galen

https://humgenprojects.lumc.nl/trac/humgenprojects/wiki/scripting