`arrayManipulation` package

Lucas Franceschino

April 3, 2019

Storing arrays An array $myArray = [9, 8]$ consist of a macro `myArray.length` set to 3, a macro `myArray:0` set to 9, and a macro `myArray:1` set to 8.

`\arrayMake` `[`⟨*scope*⟩`]{`⟨*name*⟩`}`
Create a new empty array with name `[`⟨*name*⟩`]`. If `[`⟨*scope*⟩`]` is `local` (the default), the array is defined with a `def`, otherwise, it is declared with `gdef`.

`\arrayLen` `{`⟨*arrayName*⟩`}`
Get the length of an array. Fails if array not found.

`\arrayPush` `{`⟨*arrayName*⟩`}{`⟨*value*⟩`}`

`\arrayGet` `{`⟨*arrayName*⟩`}{`⟨*index*⟩`}`
Get the `{`⟨*index*⟩`}` value of `{`⟨*arrayName*⟩`}`.

`\arraySet` `{`⟨*arrayName*⟩`}{`⟨*index*⟩`}{`⟨*value*⟩`}`

`\arrayPop` `{`⟨*arrayName*⟩`}`
Pop something out of `{`⟨*arrayName*⟩`}`

`\arrayEmpty` `{`⟨*arrayName*⟩`}`
Empty properly the array `{`⟨*arrayName*⟩`}`

`\arrayFind` `{`⟨*arrayName*⟩`}{`⟨*keyVar*⟩`}{`⟨*valueVar*⟩`}{`⟨*predicate*⟩`}`
Find the index of a value from `{`⟨*arrayName*⟩`}` that matches `{`⟨*predicate*⟩`}`. If nothing found, returns -1.

`\arrayIndexOfC` `{`⟨*arrayName*⟩`}{`⟨*val*⟩`}` Search `{`⟨*val*⟩`}` in `{`⟨*arrayName*⟩`}`, and put its index in the global TEXcounter `arrayIndexOfResult`. -1 if not found.

`\arrayIndexOf` `{`⟨*arrayName*⟩`}{`⟨*val*⟩`}`
Search `{`⟨*val*⟩`}` in `{`⟨*arrayName*⟩`}`, and returns its index or -1.

`\arrayPushNoDuplicate` `{`⟨*arrayName*⟩`}{`⟨*val*⟩`}`
Push `{`⟨*val*⟩`}` in `{`⟨*arrayName*⟩`}` if and only if `{`⟨*val*⟩`}` doesn't exists already in `{`⟨*arrayName*⟩`}`.

`\arrayRemove` `{`⟨*arrayName*⟩`}{`⟨*val*⟩`}`
Remove the value `{`⟨*val*⟩`}` (if found) from `{`⟨*arrayName*⟩`}`.

`\arrayContains` `{`⟨*arrayName*⟩`}{`⟨*val*⟩`}`
Returns 1 if `{`⟨*arrayName*⟩`}` contains `{`⟨*val*⟩`}`, 0 otherwise.

`\arrayRemoveAt` `{`⟨*arrayName*⟩`}{`⟨*index*⟩`}`
Remove the value at position `{`⟨*index*⟩`}` in `{`⟨*arrayName*⟩`}`.

`\arrayMakeOrEmpty` `{`⟨*arrayName*⟩`}`
If no array named `{`⟨*arrayName*⟩`}` is found in the scope, it just create an empty array. Otherwise, it empties `{`⟨*arrayName*⟩`}`.
The result is that after the command, `{`⟨*arrayName*⟩`}` is an empty array.

`\arrayCopy` `{`⟨*arr1*⟩`}{`⟨*arr2*⟩`}`

`\arrayForeach` `[`⟨*keyVar*⟩`]{`⟨*valueVar*⟩`}{`⟨*arrayName*⟩`}{`⟨*body*⟩`}`
`{`⟨*body*⟩`}` is expanded for each key and value (commands named `{`⟨*valueVar*⟩`}` and `{`⟨*arrayName*⟩`}` make them accessible) of `{`⟨*arrayName*⟩`}`.

`\arrayForeachSkipping` `[`⟨*keyVar*⟩`]{`⟨*valueVar*⟩`}{`⟨*arrayName*⟩`}{`⟨*body*⟩`}{`⟨*n*⟩`}`
Same as `arrayForeach`, but skipping `{`⟨*n*⟩`}` first values.

`\arrayFold` `[`⟨*initial-value*⟩`]{`⟨*keyName*⟩`}{`⟨*accName*⟩`}{`⟨*valueName*⟩`}{`⟨*foldingFun*⟩`}`

`\arrayMakeRepeat` `{`⟨*arrayName*⟩`}{`⟨*value*⟩`}{`⟨*n*⟩`}`
Create an array `{`⟨*arrayName*⟩`}` (if not found), and push `{`⟨*n*⟩`}` times `{`⟨*value*⟩`}`.

`\arrayMakeFun` `{`⟨*arrayName*⟩`}{`⟨*keyName*⟩`}{`⟨*fun*⟩`}{`⟨*len*⟩`}`
Create an array `{`⟨*arrayName*⟩`}` of length `{`⟨*len*⟩`}`, with values given by the expansion of `{`⟨*fun*⟩`}` with a command named `{`⟨*keyName*⟩`}` set to $0, 1, \ldots,${`{`⟨*len*⟩`}`}.

`\arrayMap` `[`⟨*keyName*⟩`]{`⟨*srcArray*⟩`}{`⟨*outArray*⟩`}{`⟨*valName*⟩`}{`⟨*transformer*⟩`}`
For every index i from `{`⟨*srcArray*⟩`}`, `{`⟨*outArray*⟩`}` is updated at position i with the expansion of `{`⟨*transformer*⟩`}`, with commands `{`⟨*valName*⟩`}` and `[`⟨*keyName*⟩`]` defined as current key and value

`\arrayEnsureDefined` `{`⟨*arrayName*⟩`}`

`\arrayEnsureIndexable` `{`⟨*arrayName*⟩`}{`⟨*index*⟩`}`