5. Storage

5.2. Arrays

Where are we?
Array variables

```var
  holidays: Array[1..30] of Date;
```

**Definition (Array values and array variables)**

- An **Array Value** is a mapping from a set of indices to a set of values.
- An **Array Variable** is a realization of array value using variables, so that each of the image of each index may be changed at runtime.

Array values/variables can be modeled after

---

1 with slight variation due to the fact that arrays first index is 0 rather than 1.
Array variables

VAR
holidays: Array[1..30] of Date;

Definition (Array values and array variables)

- An **Array Value** is a mapping from a set of indices to a set of values.
- An **Array Variable** is a realization of array value using variables, so that each of the image of each index may be changed at runtime.

Array values/variables can be modeled after **Fortran** Integral Exponentiation

---

1 with slight variation due to the fact that arrays first index is 0 rather than 1
Array variables

VAR
  holidays: Array[1..30] of Date;

Definition (Array values and array variables)

- An **Array Value** is a mapping from a set of indices to a set of values.
- An **Array Variable** is a realization of array value using variables, so that each of the image of each index may be changed at runtime.

Array *values/variables* can be modeled after

**FORTRAN**  Integral Exponentiation
**C/Java**  Integral Exponentiation\(^1\)

---

\(^1\)with slight variation due to the fact that arrays first index is 0 rather than 1.
Array variables

VAR
  holidays: Array[1..30] of Date;

Definition (Array values and array variables)

- An **Array Value** is a mapping from a set of indices to a set of values.
- An **Array Variable** is a realization of array value using variables, so that each of the image of each index may be changed at runtime.

Array values/variables can be modeled after

- **FORTRAN** Integral Exponentiation
- **C/JAVA** Integral Exponentiation\(^1\)
- **PASCAL** Map\(^2\)

\(^1\) with slight variation due to the fact that arrays first index is 0 rather than 1
Array variables

```pseudocode
VAR
    holidays: Array[1..30] of Date;
```

**Definition (Array values and array variables)**

- An **Array Value** is a mapping from a set of indices to a set of values.
- An **Array Variable** is a realization of array value using variables, so that each of the image of each index may be changed at runtime.

Array values/variables can be modeled after

- **FORTRAN** Integral Exponentiation
- **C/Java** Integral Exponentiation\(^1\)
- **Pascal** Map\(^2\)

\(^1\) with slight variation due to the fact that arrays first index is 0 rather than 1
Why only now?

- Array *values* are not very useful\(^3\)
- But... array *variables* become very useful...
  - Efficient mapping into memory with the classical storage models
  - Foundation for many algorithms
  - Foundation for many data structures

\(^3\)Did you see any arrays in ML?
Why only now?

- Array *values* are not very useful\(^3\)
- But... array *variables* become very useful...
  - Efficient mapping into memory with the classical storage models
  - Foundation for many algorithms
  - Foundation for many data structures

\(^3\)Did you see any arrays in ML?
Why only now?

- Array *values* are not very useful\(^3\)
- But... array *variables* become very useful...
  - Efficient mapping into memory with the classical storage models
  - Foundation for many algorithms
  - Foundation for many data structures

\(^3\)Did you see any arrays in ML?
Why only now?

- Array *values* are not very useful\(^3\)
- But... array *variables* become very useful...
  - Efficient mapping into memory with the classical storage models
  - Foundation for many algorithms
  - Foundation for many data structures

\(^3\)Did you see any arrays in ML?
Why only now?

- Array *values* are not very useful\(^3\)
- But... array *variables* become very useful...
  - Efficient mapping into memory with the classical storage models
  - Foundation for many algorithms
  - Foundation for many data structures

\(^3\)Did you see any arrays in ML?
Where are we?

5. Storage

5.2. Arrays

5.2.1. Varieties of arrays
Where are we?

5. Storage

5.2. Arrays

5.2.1. Varieties of arrays

1. Preliminaries

2. Introduction

3. Values and types

4. Advanced typing

5. Storage

5.1. Storage models

5.2. Arrays

5.2.1. Varieties of arrays

5.2.2. Arrays with integral index types

5.2.3. Type of arrays
Varieties of arrays I/III

**Static** *size determined at compile time*

```c
const char* days[] = {
    "Sun", "Mon", "Tue",
    "Wed", "Thu", "Fri", "Sat"
}; // An array literal
int main(...) {...}
```

**Stack Based** *size determined at runtime, but cannot change after creation; allocated on the stack*

```c
void printPrimes(int n) {
    unsigned char sieve[n];
    ...
}
```
Varieties of arrays II/III

**Dynamic** *size determined at runtime, but cannot change after creation; allocated from the heap*

```c
int[] printPrimes(int n) {
    unsigned char sieve[n];
    ...
    int r[] = malloc(
        sum(sieve)
        * sizeof(int));
    ...
    return r;
}
```
Varieties of arrays III/III

Flexible  *size may change dynamically; array may expand or shrink,* found e.g., in Perl.

```perl
@a = 1..6;  # uninitialized; size 6
@a = (1,2,3);  # initialized; size 3
@a[13] = 17;  # size is now 13
@a[17] = 13;  # size is now 17
delete @a[17];  # size is now 13
delete @a[13];  # size is now 3
```

Associative  *any string (object) can serve as index,* as in, e.g., AWK
Associative arrays in scripting languages

Generalized (associative) arrays: mappings from non-integral types. Common in scripting PLs, e.g., AWK, JavaScript, PHP

```bash
$wives["Adam"] = "Eve";
$wives["Lamech"] = "Adah\ and\ Zillah";
$wives["Abraham"] = "Sarah";
$wives["Isaac"] = "Rebecca";
$wives["Jacob"] = "Leah\ and\ Rachel";
```

```
... 
...
...
```

```bash
echo $patriarch;
echo $wives[$patriarch];
```
Summary: determining the index set

When is the index set determined?

**Static Arrays** fixed at compile time.

**Dynamic Arrays** on creation of the array variable.

**Stack Based Arrays** on creation of the array variable.

**Flexible arrays** not fixed; bounds may change whenever index is changed.

**Associative Arrays** no “bounds” for the set of indices; the set changes dynamically as entries are added or removed from the array.
The unbelievable power of associative arrays

Using AWK to compute the frequency of words in the input stream:

```awk
#!/usr/bin/awk -f
{ for (i = 1; i <= NF; i++) a[$i]++;
}
END {
    for (w in a)
        if (a[w] in b)
            b[a[w]] = b[a[w]] ", " w;
        else {
            b[a[w]] = w;
            if (max < a[w]) max = a[w]
        }
    for (; max > 0; max--)
        if (b[max] != "") print max, b[max];
}
```

Explanation follows…
The unbelievable power of associative arrays

Computing Word Frequency in AWK

AWK’s implicit loop reads lines in turn, breaking each line to space-separated “fields”.

```bash
#!/usr/bin/awk -f

# implicitly executed
# for each input line
{
    for (i = 1; i <= NF; i++)
        a[$i]++;
    # optional semicolon (;)
    # The “$” character is special:
    # variable $i is the i^{th}
    # word in the current line
}

END {
    # after last line read
    # Accumulate in b[i] all words
    # that occur i times
    max = 0;
    # not really necessary
    for (w in a) {
        if (! (a[w] in b)) {
            b[a[w]] = w
            if (max < a[w])
                max = a[w]
        } else
            b[a[w]] = b[a[w]] "", w;
    }
    # Print array b in descending order
    for (; max > 0; max--)
        if (b[max] != "")
            print max, b[max];
}
```
Summary: determining the index set

When is the index set determined?

**Static Arrays** fixed at compile time.

**Dynamic Arrays** on creation of the array variable.

**Stack Based Arrays** on creation of the array variable.

**Flexible arrays** not fixed; bounds may change whenever index is changed.

**Associative Arrays** no “bounds” for the set of indices; the set changes dynamically as entries are added or removed from the array.
Arrays’ efficiency

**Static, Stack based, and Dynamic:** efficient implementation in the classical memory model.
- including range-based arrays, as in **PASCAL**
- including true multi-dimensional arrays, as in **FORTRAN**
- including arrays of arrays, as in **C**

**Flexible and Associative:** require more sophisticated data structure to map to the classical memory model.
Sophisticated data structures as part of PLs?

- Associative arrays are great!
Sophisticated data structures as part of PLs?

- Associative arrays are great!
- We want more, ...
Sophisticated data structures as part of PLs?

- Associative arrays are great!
- We want more, ...
  - sets!
Sophisticated data structures as part of PLs?

- Associative arrays are great!
- We want more, ...
  - sets!
  - multi-sets!!
Sophisticated data structures as part of PLs?

- Associative arrays are great!
- We want more, ...
  - sets!
  - multi-sets!!
  - stacks and queues and trees!!!
The sad story of **Pascal**'s sets

- simple implementation
- efficient implementation
- does not scale
- with scale, you need to carefully balance
  - operations repertoire
  - time
  - memory
  - parallelization
Dilemmas in language design

- Which, if any, sophisticated data structures should be part of the PL?
- Which, if any, sophisticated data structures be part of the library?
- Would it be possible to implement sophisticated data structures as part of the library?
- What PL structures can support the making of a better standard library of good data-structures.\(^4\)

---

\(^4\)yes, logic here is a bit confusing. Think about it this way: if you give the library designer better PL tools, he will be able to design a better datastructures library. Perfection and extensions to the protocol of the standard library would not require any changes to the PL.
Where are we?

5. Storage

5.2. Arrays

5.2.2. Arrays with integral index types
Where are we?

5. Storage
5.2. Arrays
5.2.2. Arrays with integral index types
Efficient but inflexible

Ordinary arrays are formed as mappings from integral types.

Pros

- Only values are stored, not indices.
- Simple description of legal indices (defined completely by higher bound, and in some PLs by lower bound as well)
- Efficient access using simple addition:

  **Explicit** in C and C++ pointer arithmetic is explicit

  \[ a[i] \equiv *(a+i) \equiv *(i+a) \equiv i[a] \]

  **Implicit** in, e.g., JAVA, array access it translated to simple machine instructions

  **Range Mapping** in, e.g., PASCAL, array access may require subtraction of the first index to compute the actual offset

Cons

- When data are sparse, packing techniques are needed.
- Inflexible programming.
Piddles

What are Piddles? (Quotes from the Perl manual)

- *Having no good term to describe their object, PDL developers coined the term “piddle” to give a name to their data type.*
- A piddle consists of a series of numbers organized as an N-dimensional data set...
- *Perl has a general-purpose array object that can hold any type of element...*
- *Perl arrays allow you to create powerful data structures..., but they are not designed for numerical work. For that, use piddles...*
Piddles

What are Piddles? (Quotes from the Perl manual)

- *Having no good term to describe their object, PDL developers coined the term “piddle” to give a name to their data type.*

- *A piddle consists of a series of numbers organized as an N-dimensional data set…*

- *Perl has a general-purpose array object that can hold any type of element…*

- *Perl arrays allow you to create powerful data structures…,*

- *but they are not designed for numerical work. For that, use piddles…*
Piddles

What are Piddles? (Quotes from the Perl manual)

- *Having no good term to describe their object, PDL developers coined the term “piddle” to give a name to their data type.*

- *A piddle consists of a series of numbers organized as an N-dimensional data set…*

- *Perl has a general-purpose array object that can hold any type of element…*

- *Perl arrays allow you to create powerful data structures…, but they are not designed for numerical work. For that, use piddles…*
Piddles

What are Piddles? (Quotes from the Perl manual)

- **Having no good term to describe their object, PDL developers coined the term “piddle” to give a name to their data type.**
- **A piddle consists of a series of numbers organized as an N-dimensional data set...**
- **Perl has a general-purpose array object that can hold any type of element...**
- **Perl arrays allow you to create powerful data structures...,**
- **but they are not designed for numerical work. For that, use piddles...**
Piddles

What are Piddles? (Quotes from the Perl manual)

- Having no good term to describe their object, PDL developers coined the term “piddle” to give a name to their data type.
- A piddle consists of a series of numbers organized as an N-dimensional data set...
- Perl has a general-purpose array object that can hold any type of element...
- Perl arrays allow you to create powerful data structures..., but they are not designed for numerical work. For that, use piddles...
Layout of multi-dimensional arrays

Two Main Strategies:

- Multi-Layered Memory Mapping:
  - row-major
  - column-major

- Multiple Dereferencing
Row-major layout of 2D arrays (e.g., PASCAL)

\[ A: \text{a } 4 \times 4 \text{ matrix} \]

For \( A \), an \( n \times m \) matrix,

\[
\text{offset}(A_{i,j}) = (i - 1)m + (j - 1)
\]
Column-major layout of 2D arrays (e.g., FORTRAN)

**A**: a $4 \times 4$ matrix

<table>
<thead>
<tr>
<th>$A_{1,1}$</th>
<th>$A_{1,2}$</th>
<th>$A_{1,3}$</th>
<th>$A_{1,4}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{2,1}$</td>
<td>$A_{2,2}$</td>
<td>$A_{2,3}$</td>
<td>$A_{2,4}$</td>
</tr>
<tr>
<td>$A_{3,1}$</td>
<td>$A_{3,2}$</td>
<td>$A_{3,3}$</td>
<td>$A_{3,4}$</td>
</tr>
<tr>
<td>$A_{4,1}$</td>
<td>$A_{4,2}$</td>
<td>$A_{4,3}$</td>
<td>$A_{4,4}$</td>
</tr>
</tbody>
</table>

For $A$, an $n \times m$ matrix,

$$
\text{offset}(A_{i,j}) = (j - 1)n + (i - 1)
$$
“Multiple dereferencing” layout of 2D arrays

4 × 4 matrix A

\[
\begin{array}{cccc}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\
A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} \\
A_{4,1} & A_{4,2} & A_{4,3} & A_{4,4} \\
\end{array}
\]

\textit{not all rows are fully interesting}

\[
\begin{array}{cccc}
A_{1,1} & A_{1,2} & A_{1,3} & A_{1,4} \\
A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\
A_{3,1} & A_{3,2} & A_{3,3} & A_{3,4} \\
A_{4,1} & A_{4,2} & A_{4,3} & A_{4,4} \\
\end{array}
\]

In C and JAVA, a 2D array is an array of arrays:
- may be \texttt{null}.
- may be of \textit{any} length
- even length 0 is OK

\textit{C/JAVA Representation}

\[
\begin{array}{cccc}
a[0][0] & a[0][1] & a[0][2] & a[0][3] \\
a[3][0] \\
\end{array}
\]
Example: triangular array in JAVA

```java
int k = 0;
int[][] iis = new int[][]
  { new int[k++], new int[k++],
    new int[k++], new int[k++],
  }; // An array initializer
...
for (int i = 0; i < k; i++)
  for (int j = 0; j < i; j++)
    iis[i][j] = i*j;
```

![Diagram of triangular array](image)
Where are we?

5. Storage

5.2. Arrays

5.2.3. Type of arrays
Where are we?

5. Storage

5.2. Arrays

5.2.3. Type of arrays
Arrays type?

The type of an array of values of type $\tau$ (first approximation)

- Integer Indexed: $\text{Integer} \rightarrow \tau$
- String Indexed: $\text{String} \rightarrow \tau$

But, the mapping is only partial; not all possible values of Integer/String indices are mapped into values of type $\tau$.

The Array Type Predicament

To properly define the type of arrays, one needs heavier type theory artillery, which is not really interesting in our course.
Array types in JAVA

Particularly simple situation

- The type array of $\tau$ includes all arrays of $\tau$, regardless of size.
- All these arrays are assignment compatible.

```java
double[] x, y, z;
x = new double[100];
y = new double[0];
z = x; x = y; y = z;
```
Array types in Ada

```ada
type Vector is array (Integer range <> ) of Float;

procedure ReadVector(v: out Vector) is ...;
-- Uses v'first and v'last

m: Integer := ...;

a: Vector(1..10);
b: Vector(0..m)
ReadVector(a);
ReadVector(b);

a := b; -- Succeeds only if array b has exactly 10 elements.
```