14 Composite Types - Arrays

Monday, November 9, 2020 9:18 AM

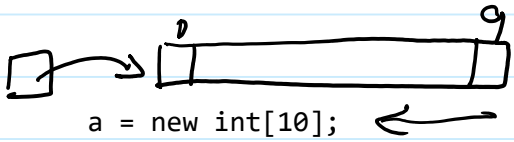
Declaration:

```
int a[10];    // C/C++
int[] a;      // Java
```

```
var a [10]int    // Go
# Python??
```

Declaration vs Creation/Construction:

```
int[] a;
int[] a = new int[10];
```




```
int[] a = {1,2,3,4,5,6,7,8,9,10};
```

More elaborate array declaration using constructors:

```
var a : array [1..10] of integer;   (* Pascal *)
```

Indexing:

```
a[i]
a(i)
a[i:j]
```

Memory Allocation

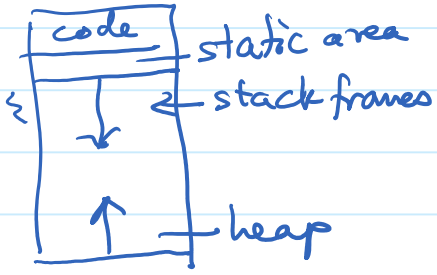
Where and when memory is used/allocated depends on whether the PL provides means for declaration separate from construction, and where an array is defined.

```
#include<stdio.h>

int main() {
   int a[10];
…
} // main()
```

```
#include<stdio.h>

int a[10];
int main() {
…
} // main()
```

Also depends on whether a **value** or a **reference** model is used:

```
int[] a;
a = new int[10];
```



Important Bindings for Arrays:

1. Name
2. Type
3. Size
4. Index bounds

14 2-D Arrays

Monday, November 9, 2020 8:47 AM

2 Dimensional Arrays

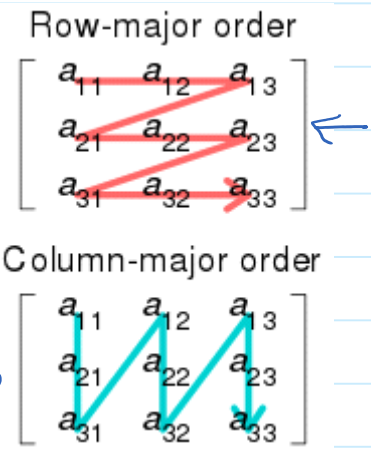
```
int a[3][4];    // MxN array where M=3, N=4
int[][] a;

a = new int[3][4];
```

a[i][j] - indexing

Memory Layout

Row-Major Form

Column Major Form



Cache implications for larger arrays.

```
for (int i=0; i < M; i++) { // Rows
   for (int j=0; j < N; j++) {
   …
   }
}
```

As opposed to…

```
for (int j=0; j < N; i++) { // Columns
   for (int i=0; i < M; j++) {
   …
   }
}
```

Linear storage and address computation:

Row-Major Form:

Address of
where is the word size (in bytes)
is the number of columns

Column-Major Form:

Address of
where is the word size (in bytes)
is the number of columns

Alternative Multi-Dimensional Array Representations:

- Sparse Matrices
- Lliffe Vectors

Alternative Multi-Dimensional Array Representations:

- Sparse Matrices
- Lliffe Vectors

14 Strings & Character Encoding

Monday, November 9, 2020 8:41 AM

"Hello" "Hello\nWorld"

Hello
World

`Hello
World` Go

"Hello" 'Hello'

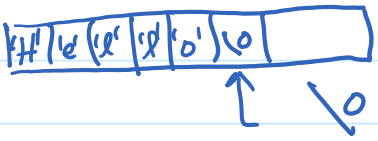
""" ___ """

''' ___ '''

"""Hello
World""" Python

char

- array of char
  null terminated.



\0 null character

- string data type

String s = "Hello"; ≡ String s = new String("Hello");

+

s = "Hello"

s[0] s[0:2]

s.charAt(1)

Encoding: 'A' 65 'A'+1

'B' 66

ASCII

ASCII → American Stand. Code for Inf. Interchange

7-bit $2^7 = 128$ ←

lexicographic ordering

UNICODE 1,112,064 chars

UTF → Unicode Transformation Format

UTF-8 Codepoints ☺ 128515

runes π 960

14 Type Checking

Monday, November 9, 2020 8:41 AM

Context

```
v1 : T1;
v2 : T2;

def f(v: T2) {
}
```

```
v1 = v2; ?
= . . . v1 + v2 . . . ?
f(v1); ?
```

Strong Typed — strict about types.

Static Typing — compile time

Dynamic Typing — runtime

v1 = v2 ?? ← · ok if T1 + T2 are equivalent
· ok if T1 + T2 are compatible

Type Equivalence

```
type T1
v1 : T1
```

```
type T2
v2 : T2
```

Name Equivalence

- same name

```
int a;
float b;      b = a;  X
```

Structural Equivalence

b = a; ok

Java

```
Celsius c;
Fahr f;
c = f; X
```

Go

```
type celsius float32
type fahr float32
var c celsius
var f fahr
c = f;
c = toCelsius(f)
```

C

```
typedef celsius float;
typedef fahr float;

celsius c;
fahr f;
c = f;   ✓ ok
```