ARRAYS I

ARRAYS

Concept a data structure or collection of similar typed variables

Example

```
int nums[10] = {10, 20, 30, 40, 50, 60, 70, 80, 90, 100};     // initialize an array
                                                              // of size 10

cout << a[0];                                                 // print first value
cout << a[9];                                                 // print last value
```

an array of size 10 has indexes 0 to 9 (0 to size-1) for accessing data

index	0	1	2	3	4	5	6	7	8	9
value	10	20	30	40	50	60	70	80	90	100
	9A60									

ex1_arrays.cpp

ARRAY INITIALIZATION

Purpose there are multiple ways to declare or initialize an array

Example

```
int nums[3];             // declare an array of size 3 (contains garbage)
                         // this might contain: {-123, A, 5050}

int nums[3] {};          // initialize an array of size 3 with default integer values
                         // this would contain: {0,0,0}

int nums[] = {4,2,9};    // initialize an array of size 3 with values
                         // this would contain: {4,2,9}

int nums[10]= {4,2,9};   // partially initialize an array of size 10 with some values
                         // this would contain: {4,2,9,0,0,0,0,0,0,0}
```

ex2_arrays_initialization.cpp

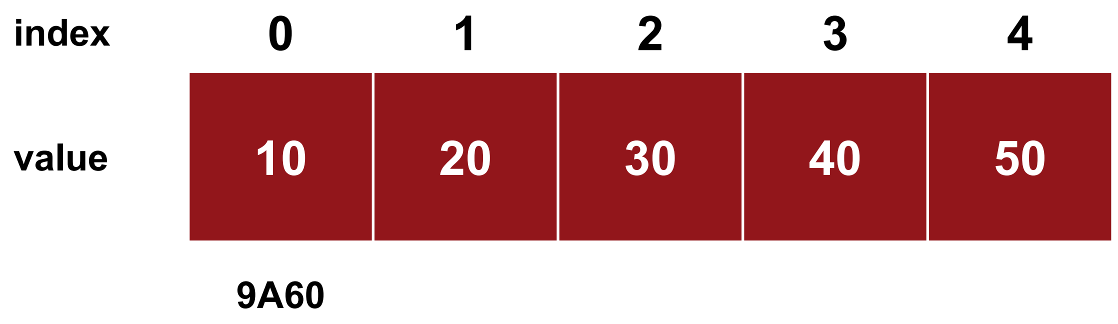
ARRAY ITERATION

Concept iterate through each value using the index such as nums[index]

Example

```
int nums[5] = {10, 20, 30, 40, 50};        // initialize an array of size 5

for(int i=0; i<5; ++i) {                    // iterate from 0 to 4 inclusive (0 to size-1)
      cout << nums[i] << " ";               // print each value at nums[i]
}
```



ARRAY ASSIGNMENT

Concept a data structure or collection of similar typed variables

Example

```
int nums[5] = {10, 20};          // partially initialize an array
```

index	0	1	2	3	4
value	10	20	0	0	0

```
nums[1] = 3000;                  // store 3000 into the 2nd index in the array
nums[4] = 5000;                  // store 5000 into the 4th index in the array
```

index	0	1	2	3	4
value	10	3000	0	0	5000

ex3_arrays_assignment.cpp

CONSTANTS FOR SIZE

Concept — use a constant for size to avoid errors and simplify program updates

Example

```
const int SIZE = 10;                // initialize a constant
int a[SIZE];                        // use a constant to declare an array

for(int=0; i<SIZE; ++i) {           // iterate from 0 to size-1
      a[i] = i;                     // store data in the array
}

for(int i=0; i<SIZE; ++i) {         // iterate from 0 to size-1
      cout << a[i] << “ “;          // print data
}
```

ARRAY MEMORY

Example

```
int nums[10] = {10,20,30,40,50,60,70,80,90,100};    // partially initialize an array
```

nums is the name of the array
nums stores the memory address of the array
nums stores the memory address of the first element in the array
memory address of subsequent elements is computed as such:
array memory address + (size of type * index)

9A60 + (4 * 3) is 9A6C, the address of the element at index 3

array memory is contiguous which makes it fast to access data

index	0	1	2	3	4	5	6	7	8	9
value	10	20	30	40	50	60	70	80	90	100
memory	9A60	9A64	9A68	9A6C	9A70	9A74	9A78	9A7C	9A80	9A84

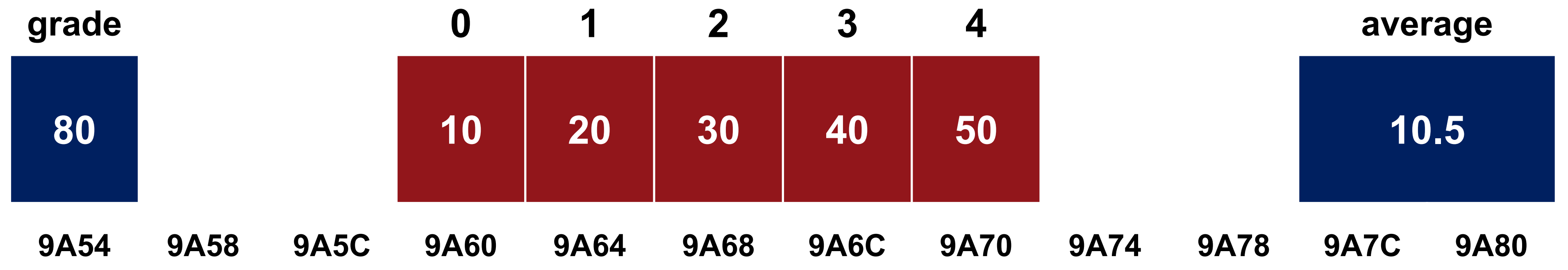
ARRAY OUT OF BOUNDS

Concept — array values exist between [0] and [size-1] which are the array boundaries

Example

```
double average = 10.5;               // initialize a double (8 bytes)
int grade = 90;                      // initialize an integer (4 bytes)
int nums[5] = {10, 20, 30, 40, 50};  // initialize an array of size 5 (20 bytes)

cout << a[-3];                       // access memory before the array (garbage)
cout << a[8];                        // access memory after the array (garbage)
a[-3] = 80;                          // overwrite memory that is not in the array
```



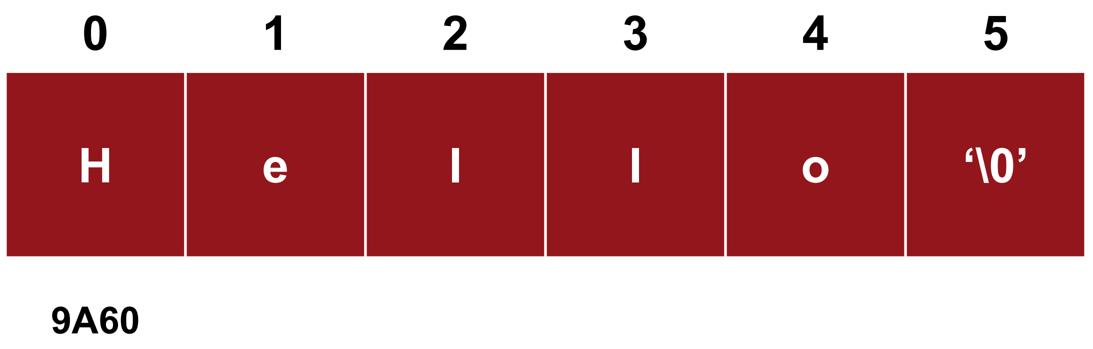
C-STRING

Concept older method of storing strings (before std::string) as character arrays

arrays have an extra null character ‘\0’ as the last element

Example

```
char word[] = ”hello”;        // initialize a c-string of size 6 (adds null char at end)
cout << word;                 // print “Hello”, this does not work for array types
```



TWO-DIMENSIONAL ARRAYS

Concept — arrays which contain a second dimension of data

Example

```
int a[2][3] = { {1,5,2}, {90,3,40} };          // initialize a 2x3 array

for(int i=0; i<2; ++i) {                        // iterate from 0 to 1 (1st dimension)
    for(int j=0; j<3; ++j) {                    // iterate from 0 to 2 (2nd dimension)
        cout << a[i][j] << “ “;                 // print every element at position i by j
    }
    cout << “\n”;                               // print each dimension on separate lines
}
```

	0	1	2
0	1	5	2
1	90	3	40

ex7_two_dimension_arrays.cpp

TWO-DIMENSIONAL ARRAY MEMORY

Example

```
int a[2][3] = { {1,5,2}, {90,3,40} };        // initialize a 2x3 array
```

	0	1	2
0	1	5	2
1	90	3	40

two dimensional arrays are stored as contiguous blocks of memory

index	[0][0]	[0][1]	[0][2]	[1][0]	[1][1]	[1][2]
value	1	5	2	90	3	40
memory	9A60	9A64	9A68	9A6C	9A70	9A74

MULTI-DIMENSIONAL ARRAYS

Concept	it is possible to have arrays with many dimensions
Example	`int a[2][3][2]` // initialize a three-dimensional array
	`int a[6][2][3][5];` // initialize a four-dimensional array
	this course focuses upon one and two-dimensional arrays