

CSCI 200: Foundational Programming Concepts & Design

Lecture 32



Static Arrays

Log into iClicker!

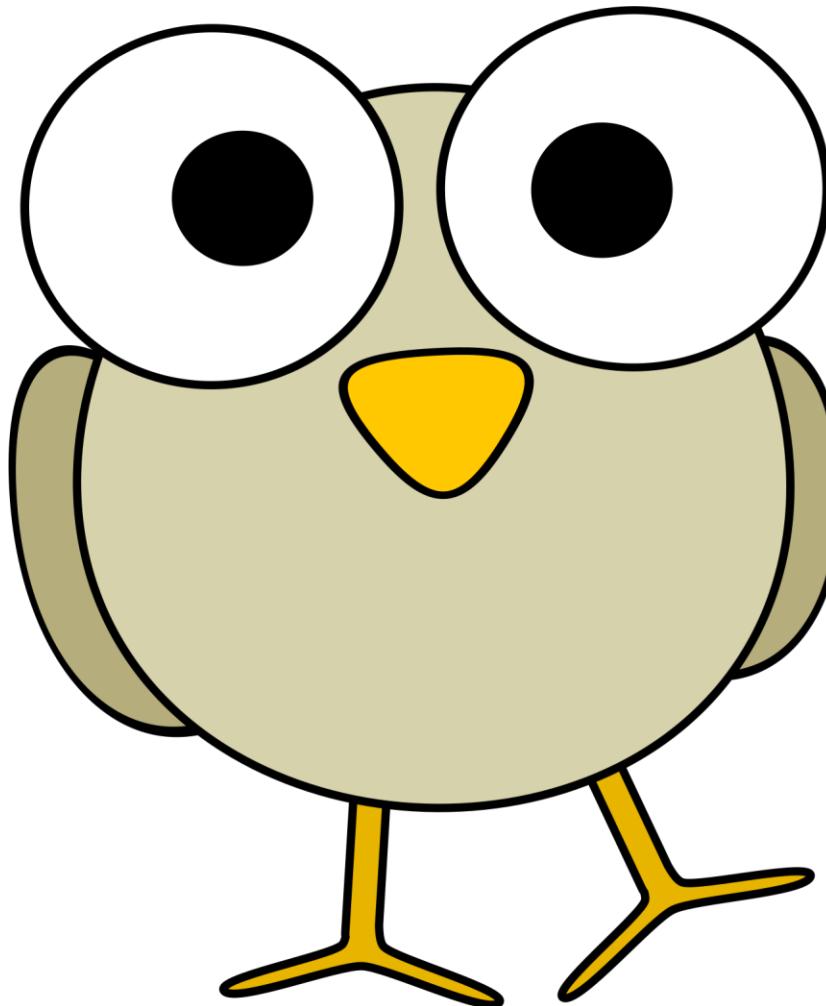
Download handout for the day

Previously in CSCI 200



- Design Principles
 - Abstract what's the same, encapsulate what varies.
 - Program to an interface, not an implementation
 - SOLID Principles
 - Single Responsibility Principle
 - Open/Closed Principle
 - Liskov Substitution Principle
 - Interface Segregation Principle
 - Dependency Inversion Principle
 - Favor composition over inheritance.

Questions?



Learning Outcomes For Today



- Evaluate the resultant output of a given code block containing an array.
- Sketch how an array is stored in memory denoting the base address and element step size.
- Construct a program using an array.
- Identify errors in a program involving an array.
- Analyze array operations using Big O Notation.

On Tap For Today



- Static Arrays (fixed size)
 - Creation
 - Memory Layout & Elemental Access
 - Special Case: Character Arrays
- Practice

On Tap For Today



- Static Arrays (fixed size)
 - Creation
 - Memory Layout & Elemental Access
 - Special Case: Character Arrays
- Practice

Static Arrays



- Simple “list-like” data structure
 - Fixed Size known at compile time
 - All elements are the same type



On Tap For Today



- Static Arrays (fixed size)
 - Creation
 - Memory Layout & Elemental Access
 - Special Case: Character Arrays
- Practice

Declaring an Array



- “Computer, create an array called stores to hold 10 integers”

```
int stores[10];
```

- “Computer, create an array called temps to hold 365 doubles”

```
double temps[365];
```

Declare & Initialize an Array



- “Computer, create an array called `omg` to store three characters. Here are three characters to put in the array.”

```
char omg[3] = { 'O', 'M', 'G' };
```

- “Computer, create an array called `count` that holds 5 integers. Here are five integers to put in the array.”

```
int count[5] = { 1, 7, 4, 2, 3 };
```

Initialize an Array



- “Computer, create an array called temp that will hold 365 doubles. Initialize ALL values to zero.”

```
double temp[365] = {0};
```

Practice!



```
int myCount[15] ;
```



Practice!



```
int myCount[15];           // declare myCount
                           // that holds 15 ints

double cool[2] = {2.5, 3.5}; // holds 2 doubles
                           // set to 2.5, 3.5

char myChar[4] = {'A'};     // holds 4 chars and
                           // init the 1st element
                           // to 'A', rest to 0

int z[] = {1, 2, 3};       // computer assumes an
                           // array size of 3
```

On Tap For Today



- Static Arrays (fixed size)
 - Creation
 - Memory Layout & Elemental Access
 - Special Case: Character Arrays
- Practice

Variables in Memory



- Every variable stored in memory (somewhere)
- Identifier points to memory address where value is stored
- How much memory does variable take up?
 - Depends on data type

Arrays in Memory



- “Computer, allocate enough **contiguous** memory for three integers. When I want a value, I’ll ask for it by **name** and **offset**.”

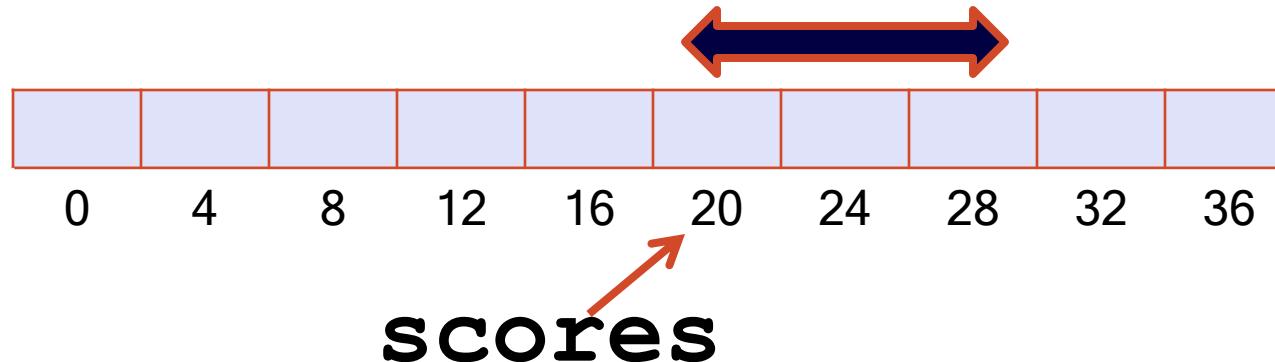
```
int scores[3];
```

Arrays in Memory



- “Computer, allocate enough **contiguous** memory for three integers. When I want a value, I’ll ask for it by name and offset.”

```
int scores[3];
```



(These addresses and the size of each box are just an example.)

Arrays in Memory



- “Computer, allocate enough contiguous memory for three integers. When I want a value, I’ll ask for it by name and offset.”

```
cout << scores[1];
```



scores

(These addresses and the size of each box are just an example.)

Arrays in Memory



identifier / name[offset]	memory address	value
	0x4f83821c	
scores / scores[0]	0x4f838220	14
scores[1]	0x4f838224	33
scores[2]	0x4f838228	21
	0x4f83822c	
	0x4f838230	

- The array **identifier** points to the **base address** of the array

Arrays in Memory



identifier / name[offset]	memory address	value
	0x4f83821c	
scores / scores[0]	0x4f838220	14
scores[1]	0x4f838224	33
scores[2]	0x4f838228	21
	0x4f83822c	
	0x4f838230	

- The **offset** is used to calculate the location of an individual element

Finding an Element in Memory



- The array **identifier** points to the **base address** of the array
- The **offset** is used to calculate the location of an individual element
- How much to offset'

/

$$\text{elementAddress} = \text{baseAddress} + (\text{dataTypeSize} * \text{offset})$$

Demo



```
const int SIZE = 3;  
  
int myArray[SIZE] = { 34, 5, 678 };  
  
cout << myArray << endl << endl;  
  
for( int i = 0; i < SIZE; i++ )  
  
    cout << "myArray[" << i << "] is:"  
        << myArray[i] << endl;  
  
  
for( int i = 0; i < SIZE; i++ )  
  
    cout << "The address of myArray["  
        << i << "] is:" << &myArray[i]  
        << endl;
```

Accessing Array Values



- “Computer, print the value at position 0 in scores”

```
int scores[2] = { 4, 2 };
```

```
cout << scores[0];
```

Accessing Array Values



- “Computer, add the values in scores and assign to sum”

```
int scores[2] = { 4, 2 };
```

```
int sum;
```

```
sum = scores[0] + scores[1];
```

Setting Array Values



- “Computer, assign the value 5 to position 0”
- “Computer, assign the value 100 to position 1”;

```
int scores[2];
```

```
scores[0] = 5;
```

```
scores[1] = 100;
```

Precedence Table

Category	Precedence	Operator	Associativity
Parenthesis	1	()	Innermost First
Scope Resolution	2	S::	
Postfix Unary Operators	3	a ⁺⁺ a ⁻⁻ a[] a. p-> f()	Left to Right
Prefix Unary Operators	4	⁺⁺ a ⁻⁻ a ^{+a} -a !a ^{~a} (type)a &a *p new delete	Right to Left
Binary Operators	5	a*b a/b a%b	
	6	a+b a-b	
Shift Operators	7	a<<b a>>b	
Relational Operators	8	a<b a>b a<=b a>=b	
	9	a==b a!=b	
Bitwise Operators	10	a&b	Left to Right
	11	a^b	
	12	a b	
Logical Operators	13	a&&b	
	14	a b	
Assignment	15	a=b a+=b a-=b a*=b a/=b a%==b a&=b a^=b a =b	Right to Left

Accessing Array Values



- Arrays have *indices* that refer to its values

'a'	'b'	'c'
0	1	2

- Array indices start with 0
- Array indices start with 0
- Array indices start with 0

Common Errors



```
int v[2] = { 4, 2 };

cout << v[5];      // index out of range

cout << v[-1];    // index out of range

cout << v[2];      // index out of range

int x[]; // no size and no init values

// In C++ you CANNOT change a static
// array's size after you declare it
```

Practice



- Runtime to
 - Access an element in an array?
 - Traverse an array?



Data Structure Operations



Operation	Array
Element Access	$O(1)$
Traversal	$O(n)$
Search	$O(n)$
Min / Max	$O(n)$

Practice



- Work through handout

On Tap For Today



- Static Arrays (fixed size)
 - Creation
 - Memory Layout & Elemental Access
 - Special Case: Character Arrays
- Practice

Arrays & Loops



- Need to iterate over array to input or output all the elements one at a time

```
const int SIZE = 15;  
  
int myIntArray[SIZE];  
  
for(int i < 0; i < SIZE; i++)  
  
    cin >> myIntArray[i];  
  
for(int i < 0; i < SIZE; i++)  
  
    cout << myIntArray[i] << endl;
```

- Except...
 - ...if array type is a character

C Strings I/O



```
const int SIZE = 15;  
  
char myCString[SIZE] = "I Love C++!";  
  
cout << myCString << endl;  
  
cout << "Enter a new string: ";  
  
cin >> myCString;  
  
cout << myCString << endl;
```

C-Style String



- Special case when array type is char
 - Size of string must still be known up front

```
const int SIZE = 15;  
char myCString[SIZE] = "I Love C++!";
```

- (Ends in '\0' aka null terminator)

String Functions



- Commonly used functions

```
#include <cstring>
```

```
strlen();
```

```
strcpy();
```

```
strncpy();
```

```
strcmp();
```

String Functions



- Commonly used functions

```
#include <cstring>
```

```
char myS[20] = "Hi!" ;
```

```
char myS1[7] = "Hey!", myS2[5] ;
```

```
strnlen();           cout << strnlen(myS, 20) << endl;
```

```
strncpy();          strncpy( myS2, myS1, 20 );
```

```
strncat();          strncat( myS, myS1, 20 );
```

```
strcmp();           if( strcmp( myS, myS1, 7 ) < 0 )
```

C Strings I/O



```
const int SIZE = 15;  
  
char myCString[SIZE] = "I Love C++!";  
  
cout << myString << endl;  
  
cout << "Length: "  
     << strlen( myCString, SIZE ) << endl;  
  
cin >> myString;  
  
cout << myString << endl;  
  
cout << "Length: "  
     << strlen( myCString, SIZE ) << endl;
```

Size of a **char** Array



- RULE: You should NOT exceed the bounds of the array

```
int myArray[10];
```

```
char myCString[10];
```

```
myArray[10] = 5;           // NOT OK!
```

```
cout << myCString[20];    // ERROR!
```

Size of a **char** Array



- RULE: You should NOT exceed the bounds of the data stored

```
char myCString[10] = "Hi !";
```

```
cout << myCString[6];      // ERROR!
```

On Tap For Today



- Static Arrays (fixed size)
 - Creation
 - Memory Layout & Elemental Access
 - Special Case: Character Arrays
- Practice

To Do For Next Time



- Continue on Set5 and Final Project
- 11/19 Inheritance & SOLID Quiz