CSCI 200: Foundational Programming Concepts & Design Lecture 33

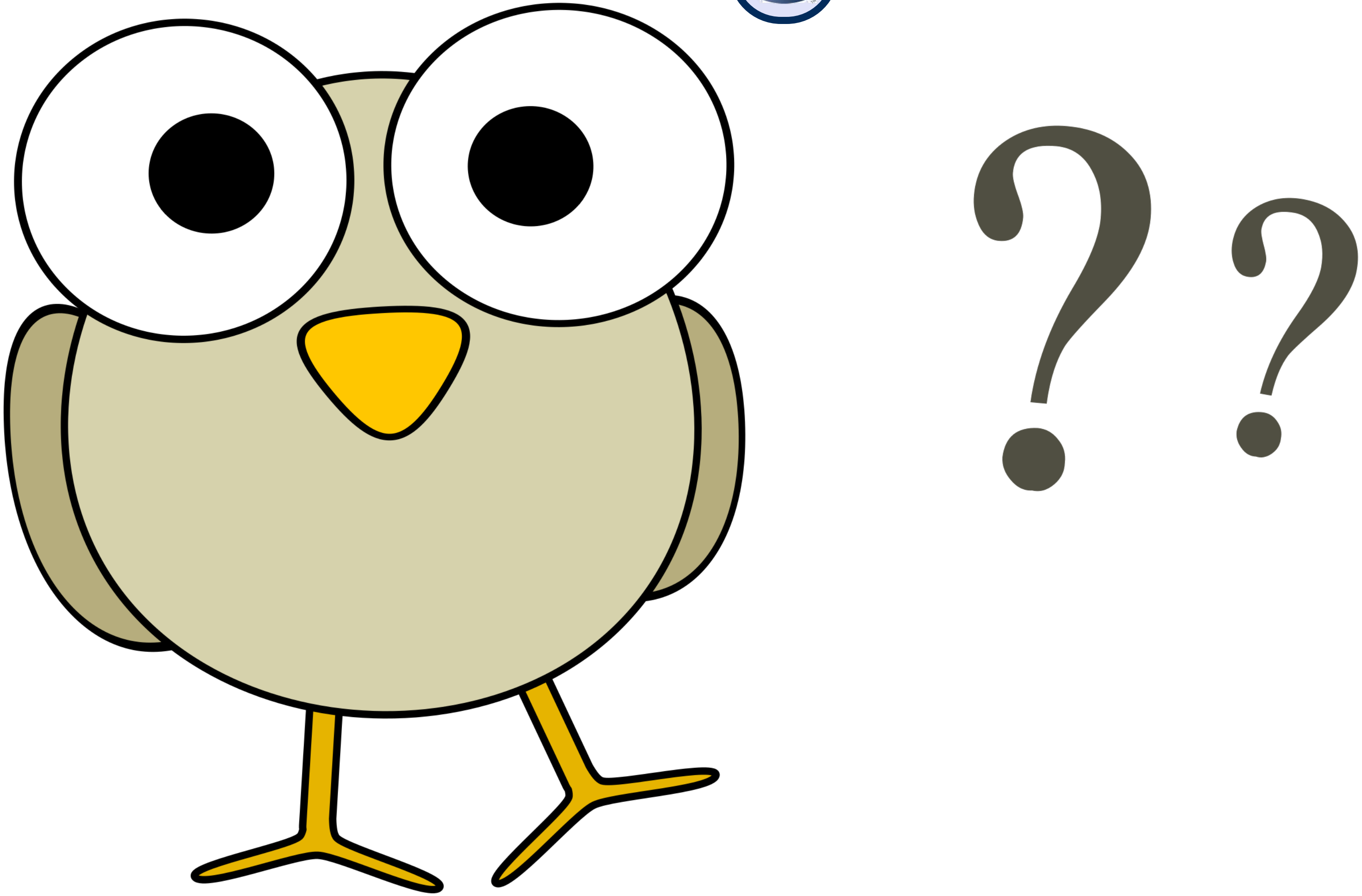
Dynamic Arrays

List Operations & Big-O Notation

Previously in CSCI 200

- Array identifier points to base address of array
- Array stored in one contiguous block of memory
- Offset used to determine memory location of specific element
- Array operations & Big O complexity

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.
- Create an array of variable size on the free store.
- Diagram the memory associated with pointers and where the values lie (either in the stack or the free store).
- Evaluate expressions involving pointer arithmetic.
- Diagram how pass-by-pointer works with pass-by-value and pass-by-reference in functions.
- Discuss causes of & solutions to memory leaks, segmentation faults, dangling pointers, null pointer exceptions, and other pointer related errors.

On Tap For Today

- Dynamic Arrays (variable size)
  - Pointer Math
- List Operations – Array Implementation
- Practice

On Tap For Today

- Dynamic Arrays (variable size)
  - Pointer Math
- List Operations – Array Implementation
- Practice

Storing Objects on the Free Store

- Use a pointer!

```
int *pNumCars = new int;     //*pNumCars  initialized to 0
int *pNumCars2 = new int(5); //*pNumCars2 initialized to 5
```

- **new** – “Computer, allocate enough memory in the free store for one object and tell me the starting address where the object will be stored.”

Dynamic Arrays

```cpp
int main() {
  unsigned int n;
  cin >> n;
  // Dynamically allocate one unsigned int object
  // whose value is initialized to n
  unsigned int *pSingleInt = new unsigned int(n);
  // Dynamically allocate an int array of size n
  // with all elements set to their default construction
  int *pDynArr = new int[n];
  // Return memory to the free store
  delete pSingleInt;  // delete one integer
  delete[] pDynArr;   // delete array
  return 0;
}
```

On Tap For Today

- Dynamic Arrays (variable size)
  - Pointer Math
- List Operations – Array Implementation
- Practice

Pointer Math

- Let’s walk through the following

```
int *pDynArr = new int[n];
int *pCurrArrSlot = pDynArr;
// *pCurrArrSlot == pCurrArrSlot[0]

for(unsigned int i = 0; i < n; i++) {
  *pCurrArrSlot = i;
  pCurrArrSlot++; // pCurrArrSlot += sizeof(int)
}
```

Pointer Math

```
int *pDynArr = new int[n];


pDynArr[i] == *(pDynArr + i)
```

Concerns

- Invalid Array Access…

```
int staticArr[10];
cout << staticArr[10] << endl; // what happens?


int *pDynArr = new int[10];
cout << pDynArr[10] << endl;   // what happens?
```

On Tap For Today

- Dynamic Arrays (variable size)
  - Pointer Math
- List Operations – Array Implementation
- Practice

Element Access

- Occurs in constant time - O(1)

Printing An Array

- Given an array of size *n*, how many elements need to be inspected to print the entire array?

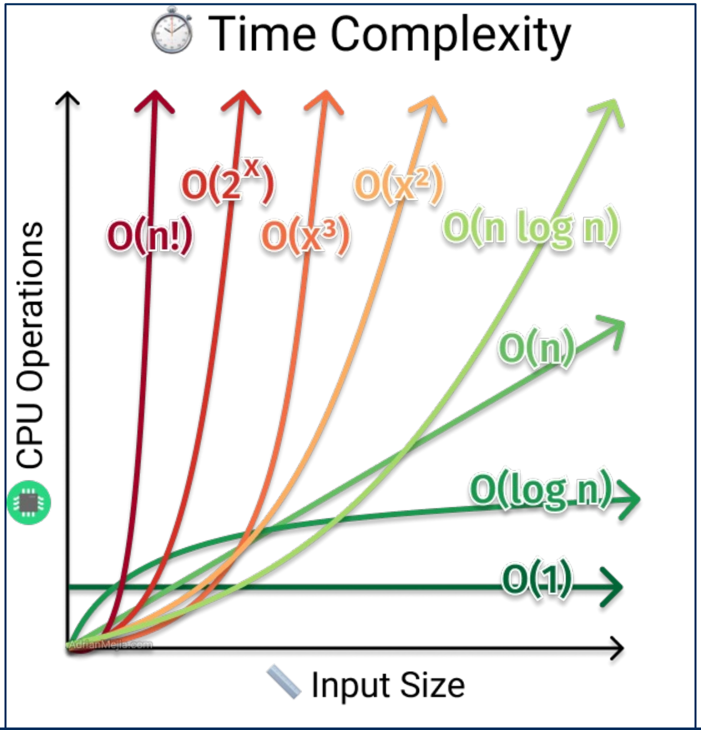
Printing An Array

- Given an array of size *n*, how many elements need to be inspected to print the entire array?

- Occurs in linear time - O(n)
  - O(n) > O(1)

Big O Dominance Relations

- Higher order polynomials dominate lower order
  - $O(n^n) > O(n!) > O(2^n) > O(n^3) > O(n^2) > O(n \log n) > O(n) > O(\log n) > O(1)$



How to resize an array

- Given

```
unsigned int n;
cin >> n;
// Dynamically allocate a double array of size n
double *pDynArr = new double[n];
for(unsigned int i = 0; i < n; i++) {
  cin >> pDynArr[i];
}
```

- How to resize the array to be size n+1?

Steps To Resize an Array

1. Make new array of larger size

```
double *pNewArray = new double[n+1];
```

2. Copy element by element from existing array to new array

```
for(unsigned int i = 0; i < n; i++)
  pNewArray[i] = pDynArr[i];
```

3. Delete existing array

```
delete[] pDynArr;
```

4. Assign pointer from existing array to new array

```
pDynArr = pNewArray;
```

The Vector Problem

- In-use size vs Allocated capacity

```
vector<unsigned int> vec;
cout << vec.size() << " " << vec.capacity() << endl;

for(unsigned int i = 0; i < 129; i++) {
  vec.push_back(i);
  cout << vec.size() << " " << vec.capacity() << endl;
}
```

- Allocating more than potentially necessary.
  - Why?
  - At what cost?
- The eternal trade-off: speed vs memory
- Want finer control of memory usage

Common Operations

- Insert at position `i`
- Remove from position `i`
- Find target value

Data Structure Operations

Operation	Array
Element Access	O(1)
Traversal	O(*n*)
Add	O(*n*)
Remove	O(*n*)
Search	O(*n*)
Min / Max	O(*n*)

On Tap For Today

- Dynamic Arrays (variable size)
  - Pointer Math
- List Operations – Array Implementation
- Practice

To Do For Next Time

- Continue on Set5 and Final Project
  - Set5 due Tuesday
- 11/15 Quiz 5 – Inheritance
- Can start L6A to complete Array test suite