Complexity

William Hendrix

Lecture 18

Outline

- Header guards
- Complexity
- Vectors
- 2D arrays

Header guards

- We define all of the objects and functions associated with a class in header files
  - If someone needs our class, they can include the header
  - Including a file "pastes" that the file contents at that location
- **Complication:** it's a syntax error to redefine functions or objects in the same scope
  - If we include a header that includes another, we "can't" include the second without causing an error
- **Solution:** header guards
  - Syntax

```
#ifndef FILENAME_H
#define FILENAME_H
//...
#endif
```

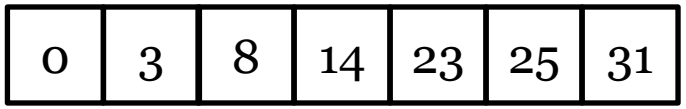
  - First time included, defines symbol
  - Second time included, code is excised
  - Symbol defined needs to be unique to file

Order of complexity

- Measures how computation increases w.r.t. problem size
  - E.g., the time needed to find the max value in an array is proportional to the array size
    - If sorted, one operation (constant)
- Measured with Big-Oh notation
  - **Definition:** *O(f(n))* means there is some coefficient *c* such that the number of operations is less than *cf(n)* for every problem of size *n* (upper bound)
    - *O(1)*: constant time (e.g., finding the min/max in a sorted array)
    - *O(n)*: linear time (e.g., searching an array for a given value)
    - $O(n^2)$: quadratic time (e.g., calculating max distance between vectors)
    - *O(ln(n))*: log time (e.g., binary search)
    - *O(nln(n))*: "linearithmic" time
- For sufficiently large problems, order of complexity outweighs effect of coefficient

Big-Oh example: Binary Search

- **Problem:** search for a given value in a sorted array
- **Strategy:** compare target to the middle, eliminate one half, then move to middle of remaining half, etc.
  - If only one element left, then target is not in array
  - **Example:**



Target: 21

```
int bin_search(int* arr, int target, int left, int right)
{
  int mid = (left + right) / 2;
  if (target == arr[mid]) return mid;
  else if (left == right) return -1;
  else if (target < arr[mid])
    return bin_search(arr, target, left, mid-1);
  else
    return bin_search(arr, target, mid+1, right);
}
```

- **Observation:** we need another round every time problem size doubles
  - Exponentially increasing problem size means complexity growth is logarithmic, *O(ln(n))*

Friday's exercise

- MutableArray needs to expand periodically to accommodate more elements
  - Allocate and copy
    - Cost is *O(n)*
- Adding 1 element each time, *O(n)* time per element added
  - Adding 2 or 10, *O(n)* time per *k* elements (still linear)
- Doubling size costs *O(n)*, but time between elements is exponential
  - Logarithmic per element added, *O(ln(n))*
  - Adding elements is "free" most of the time
  - Reallocation time increases drastically

Vectors

- Expandable array for any data type
- To use: add `#include <vector>` to preamble
- To declare: `vector<type> var_name;`
  - Can be any type
  - Optional: specify an initial size and value

    ```
    vector<type> var_name(size, init_value);
    ```

- Adding elements

  ```
  var_name.push_back(data);
  ```

  - Type of data should match `vector` type
  - Adds as last element to `vector`
- Size: `varname.size()`
- Accessing elements

  `var_name[index]` or `var_name.at(index)`
  - Will cause an exception if index out of bounds
  - Valid on LHS of assignment statement (returns `int&`)

Other operations

- Remove last element: `vec.pop_back();`
  - No return value
- Remove all elements: `vec.clear();`
- Copying vectors: `vec2 = vec1;`
  - Clears `vec2` and copies all elements of `vec1` into `vec2`
  - Types must match

2D arrays

- Multiple ways to store 2D (matrix) data
- **Preferred method:** dynamic array
  - Imagine assigning indices by row:
    - C/C++ are row-major
  - Element `(i, j)` at `i*ncol + j`

0	1	2
3	4	5
6	7	8

- **Alternative:** 2D static arrays
  - Syntax: `int mat[NROW][NCOL];`
  - Compiler converts to 1D array with index math
  - Quirk: need to indicate `NCOL` for function parameters

    ```
    double matrixMax(double matrix[][NCOL])
    ```

    - Wouldn't be able compute indices otherwise
- Not preferred: array of arrays
  - Dereferencing is relatively expensive operation
  - Can be useful if array is "ragged"

Tonight

Midterm tomorrow!

Lab 4 due Wednesday

Recommended reading: Section 8.1-8.4