Lecture 2

CS 105: Computer Programming C++
Arrays!

- Indexed data structure
  - Starts at zero
- How do we declare an array?
  - type arrayName[arraySize];
  - Ex: int array[5];
Array Initialization

- Loop
- Initializer list
- Use const array
Array Initialization

- Loop
  ```
  int array[5];
  for (int i = 0; i < 5; i++) { array[i] = i; }
  ```
- Initializer list
- Use const array size
Differences from Java

- No automatic `.length` for arrays
- No guaranteed compiler array bounds checks, if you go outside [0 through (arraySize-1)], undefined behavior
- Arrays always contiguous in memory
Character Arrays

- char chArray[] = "hello";
- What size is the array above?
Passing Arrays to Functions

- void modifyArray(int [], int);
- void modifyArray(int [] a, int aSize)
Passing Arrays to Functions

- Arrays are passed by reference (WHY?)
  - If copy, need to specify size!
- Name of array is address in memory of 1st element
- Need to pass size too, unlike Java
- Use const to make sure function can’t change array
  - void cannotModifyArray(const int b[], int size);
  - const is like saying array is, read-only
size_t

- Used when dealing with signs from any object
- Alias to an unsigned int
- size changes depending on the system
  - 32, 64, etc
sizeof

- returns the number of bytes (in size_t)
- is really an operator
  - can use without parens like ++
- works on arrays as well
  - returns the number of elements * size of type
References vs Pointers

- Both point to memory addresses
- References are more similar to an alias
A reference is like an alias, it refers to another variable
Can treat it like a normal variable afterwards
Cannot be reassigned
Generally safer than pointers (it can never be null)
ISO Standard "There shall be no references to references, no arrays of references, and no pointers to references"
References (& and *)

- How do we get a reference?
  - Use &
- & returns the "memory address" of the given variable
- * returns the value that the variable is pointing to
- & and * are inverses
A pointer is the address of another variable
Can be reassigned
Arithmetic possible
  ++
Is undefined when created
Arrays are pointers!
Arrays as Pointers

- `int myArray[10] = {}`
- `myArray` is a constant pointer to the first element of the array
  - `myArray` is equal to the address of the first element of the array (`myArray[0]`)
- `void changeArray(int [])` is equivalent to `void changeArray(int*)`
- `sizeof` returns the number of elements * type of elements (not like a pointer)
Pointer Arithmetic
References in Functions

- Efficient!
Reference

- Allows you to use without dereferencing (it knows what variable you are talking about)
Pointers
Pointers

- When to use (Only Guidelines)
  - pointer arithmetic
  - when you can return NULL values
Multidimensional Arrays

- Declaration
  - int a[10][3]; int b[10][5][3];
  - int a[2][3] = { {1,2,3}, {4,5,6} }; /// a 2x3 array

- Passing
  - When passing the arrays all but the first dimension must be known at compile time