Unit 11b

Functions
Pass-by-Reference & Array Arguments
Passing Arrays As Arguments

• Can we pass an array to another function?
  – YES!!

• Syntax:
  – **Step 1**: In the prototype/signature:
    Put empty square brackets after the parameter name if it is an array (e.g. `int data[]`)
  – **Step 2**: When you call the function, just provide the name of the array

```c
// Function that takes an array
int sum(int data[], int size);

int sum(int data[], int size)
{
    int total = 0;
    for(int i=0; i < size; i++){
        total += data[i];
    }
    return total;
}

int main()
{
    int vals[100];
    /* some code to initialize vals */
    int mysum = sum(vals, 100);
    cout << mysum << endl;
    // prints sum of all numbers
    return 0;
}
```
Pass-by-Value & Pass-by-Reference

• What are the pros and cons of emailing a document by:
  – Attaching it to the email
  – Sending a link (URL) to the document on some cloud service (etc. Google Docs)

• **Pass-by-value** is like emailing an attachment
  – A *copy* is made and sent

• **Pass-by-reference** means emailing a link to the original
  – No *copy* is made and any modifications by the other party are seen by the originator
Arrays And Pass-by-Reference

• Single (scalar) variables are passed-by-value in C/C++
  – Copies are passed
• Arrays are passed-by-reference
  – Links are passed
  – This means any change to the array by the function is visible upon return to the caller

```c
void dec(int);
int main()
{
    int y = 3;
    dec(y);
    cout << y << endl;
    return 0;
}

void dec(int y)
{
    y--;    }
```

```c
void init(int x[], int size);
int main()
{
    int data[10];
    init(data, 10);
    cout << data[9] << endl;
    // prints 0
    return 0;
}

void init(int x[], int size)
{
    // x is really a link to data
    for(int i=0; i < size; i++)
    {
        x[i] = 0; // changing data[i]
    }
}
```
But Why?

• If we used pass-by-value then we'd have to make a copy of a potentially HUGE amount of data (what if the array had a million elements)

• To avoid copying vast amounts of data, we pass a link

```c
// Function that takes an array
int sum(int data[], int size);

int sum(int data[], int size) {
    int total = 0;
    for(int i=0; i < size; i++){
        total += data[i];
    }
    return total;
}

int main() {
    int vals[100];
    /* some code to initialize vals */
    int mysum = sum(vals, 100);
    cout << mysum << endl;
    // prints sum of all numbers
    return 0;
}
```
So What Is Actually Passed?

- The "link" that is passed is just the starting address (e.g. 520) of the array in memory.
- The called function can now use 520 to access the original array (read it or write new values to it).

```c++
// Function that takes an array
int sum(int data[], int size);

int sum(int data[], int size)
{
    int total = 0;
    for(int i=0; i < size; i++){
        total += data[i];
    }
    return total;
}

int main()
{
    int vals[100];
    /* some code to initialize vals */
    int mysum = sum(vals, 100);
    cout << mysum << endl;
    // prints sum of all numbers
    return 0;
}
```
Analogy

- The first house on the 3600 block of Catalina Ave. has the address 3600.
- How many houses are on that block?

- There is no way to know!! We would have to count that separately.
Arrays in C/C++ vs. Other Languages

- Notice that if `sum()` only has the start address it would not know how big the array is.

- Unlike Java or other languages where you can call some function to find the size of an array, C/C++ require you to track the size yourself in a separate variable and pass it as a secondary argument.

```c
// Function that takes an array
int sum(int data[], int size);

int sum(int data[], int size) {
    int total = 0;
    for(int i=0; i < size; i++) {
        total += data[i];
    }
    return total;
}

int main() {
    int vals[100];
    /* some code to initialize vals */
    int mysum = sum(vals, 100);
    cout << mysum << endl;
    // prints sum of all numbers
    return 0;
}
```
Why Don't We Return Arrays from Functions

• In C++, we generally do **NOT** return arrays from a function...because we do **NOT** need to!

• **WHY?**
  – Because we modified the original array in the function

```cpp
// Function that takes an array
int[] fill(int data[], int size);
void fill(int data[], int size);

int[] fill(int data[], int size)
{
    for(int i=0; i < size; i++)
    {
        data[i] = i;
    }
}

int main()
{
    int vals[100];
    /* some code to initialize vals */
    fill(vals, 100);
    cout << mysum << endl;
    // prints sum of all numbers
    return 0;
}
```
Summary

• Syntax:
  – In the prototype/signature: Put empty square brackets after the parameter name if it is an array (e.g. void f1(int data[]))
  – When you call the function, just provide the name of the array (e.g. f1(data); )

• Functions only know what you pass them
  – You must pass the size of the array as an additional parameter in addition to the link to the array
  – Arrays are passed-by-reference meaning no copy is made and changes by a function are actually being made to the original

• The C++ std:: library provides some alternatives to "plain-old arrays" (like vectors), but you will learn about these in CS 103/104 and should not use them in CS 102