CS31 Introduction to Computer Science

FALL 2015

DISCUSSION SESSION 5

OCTOBER 30, 2015

Noor Abani

WHY DO WE NEED ARRAYS?

- Why do we need arrays?
  - Imagine keeping track of 5 test scores, or 100, or 1000 in memory
  - How would you name all the variables?
  - How would you process each of the variables?

ARRAYS

- Unlike regular variables, arrays can hold multiple values
- An array is a collection of variables of the same data type stored in a contiguous block of memory
- One-Dimensional Array:
  - stores values sequentially in memory
  - associates an index with each value (starting from 0)
  - uses index to quickly access any element

ARRAY DECLARATION

- We declare arrays using the following syntax:

  <type> <name>[size];

  ex 1. double scores[6];

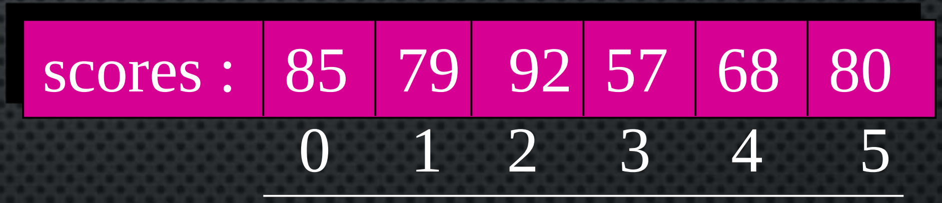
  Saying that:

  - each element is a double
  - there is space for _6 elements_
  - they are numbered _0 through 5_

  ex 2. int n[100];

  Saying that:

  - each element is an integer
  - there is space for _100 elements_
  - they are numbered _0 through 99_



Note: The value we declare for size MUST be a CONSTANT at compile time.

EXAMPLE

```
#include <iostream>
using namespace std;

int main(){
        int  num =5 ;
        int  scores[num];
       }
```

fix

```
#include <iostream>
using namespace std;

int main(){
        const int  num =5 ;
        int  scores[num];
       }
```

This code will not compile. The size of the array is not constant at compile time.

ACCESSING INDIVIDUAL COMPONENTS OF AN ARRAY

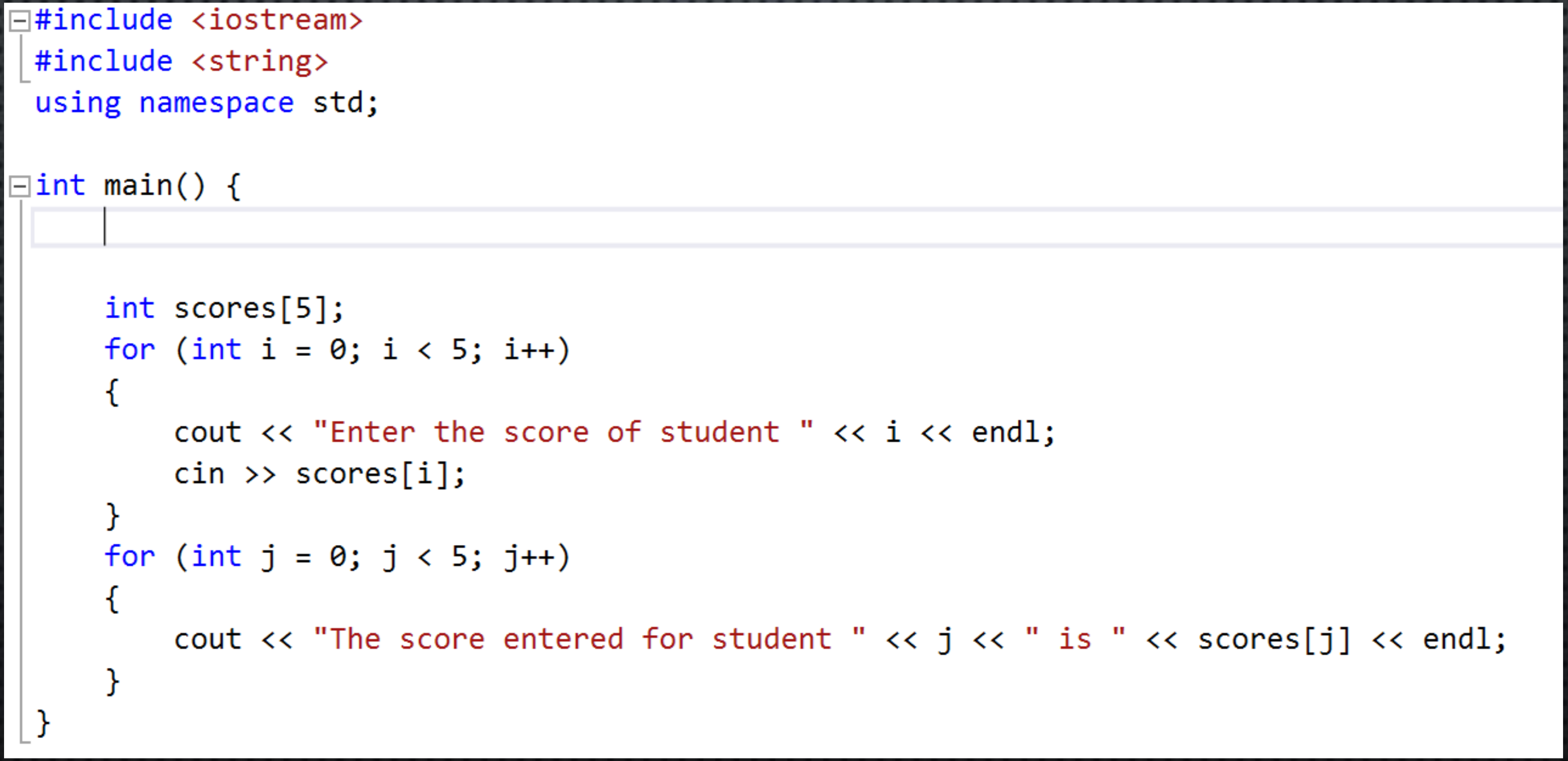
- Use the name of the array
- Followed by an integer expression inside the square brackets [ ]

The index can be a constant, a variable or an expression but it MUST be an integer

- scores[4]
- n[78]
- int x= 2; scores[x];
- int count = 10; n[count-1];

EXAMPLE

A program that reads the scores of 5 students and then displays them.



```
#include <iostream>
#include <string>
using namespace std;

int main() {
	


	int scores[5];
	for (int i = 0; i < 5; i++)
	{
		cout << "Enter the score of student " << i << endl;
		cin >> scores[i];
	}
	for (int j = 0; j < 5; j++)
	{
		cout << "The score entered for student " << j << " is " << scores[j] << endl;
	}
}
```

EXAMPLES

```
#include <iostream>
#include <string>
using namespace std;

int main() {
        int i[3];
        cout << i[1] << endl;
}
```

This code will compile. The output will be some junk value.

```
#include <iostream>
#include <string>
using namespace std;

 int main () {
     int arr[3.14];
     cout << arr[1] << endl;
 }
```

This code will not compile. Size is not an integer

```
#include <iostream>
#include <string>
using namespace std;

int main() {
        string text[2];
        cout << text[1] << endl;
}
```

This code will compile. The output will be the empty string.

INITIALIZING ARRAYS

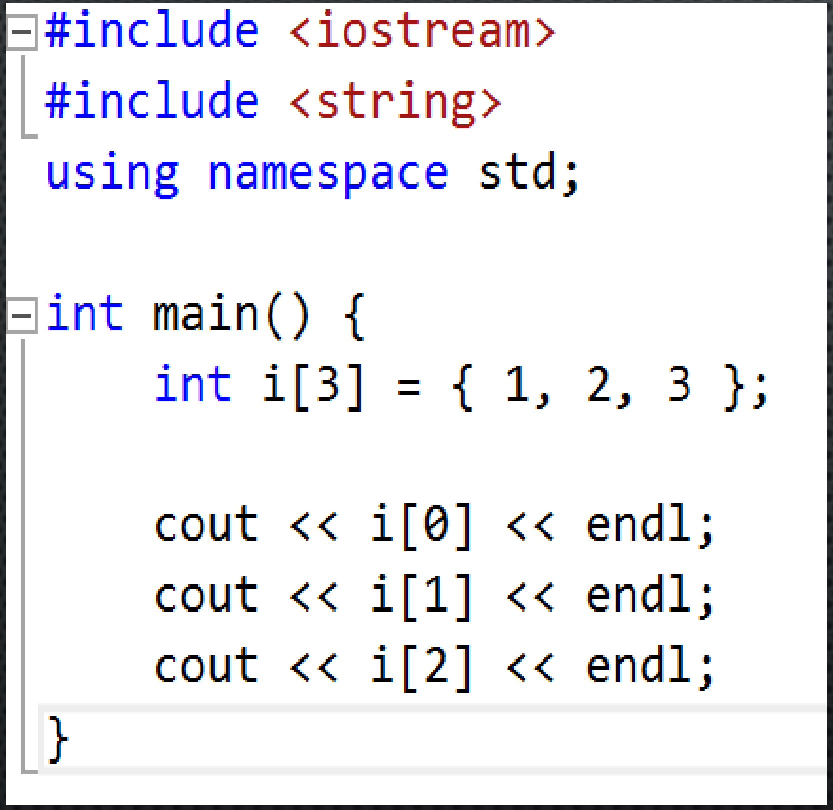
- Two ways to initialize an array:
  - Explicitly (using an assignment statement for each element)
  - Using a list. A list is denoted using braces, with element in the list separated by a comma

    <type> <name>[<size (optional)>] = {<value0>, <value1>, ...};

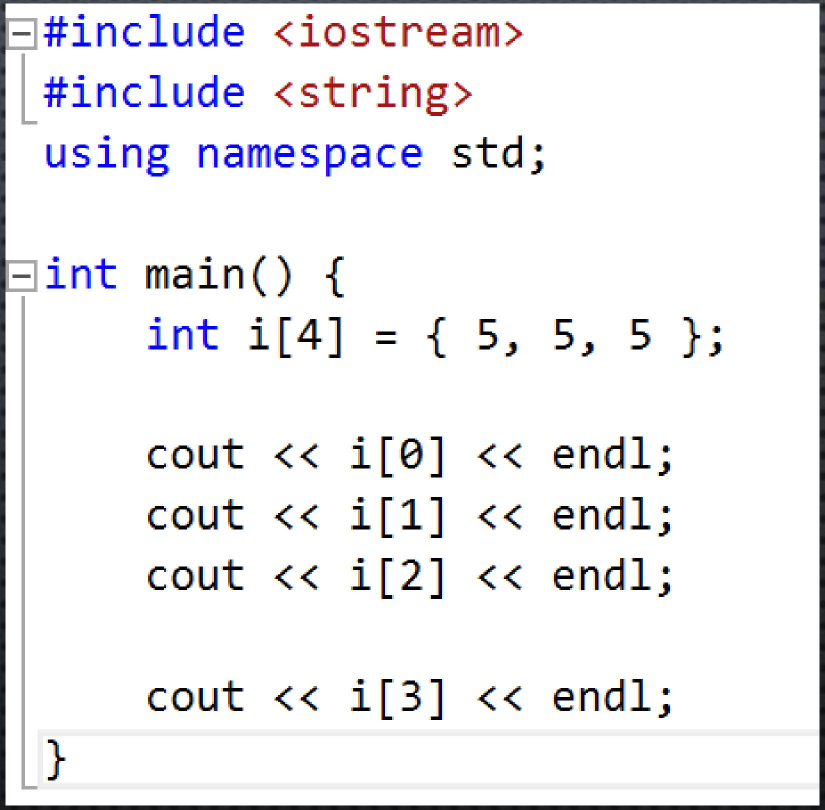
```
#include <iostream>
 #include <string>
 using namespace std;

 int main () {
    int i[] = {1, 2, 3};

    cout << i[0] << endl;
    cout << i[1] << endl;
    cout << i[2] << endl;
 }
```

NOTE: As long as you initialize an array at declaration, you do not need to specify the size; it will match the dimension of the initialization.

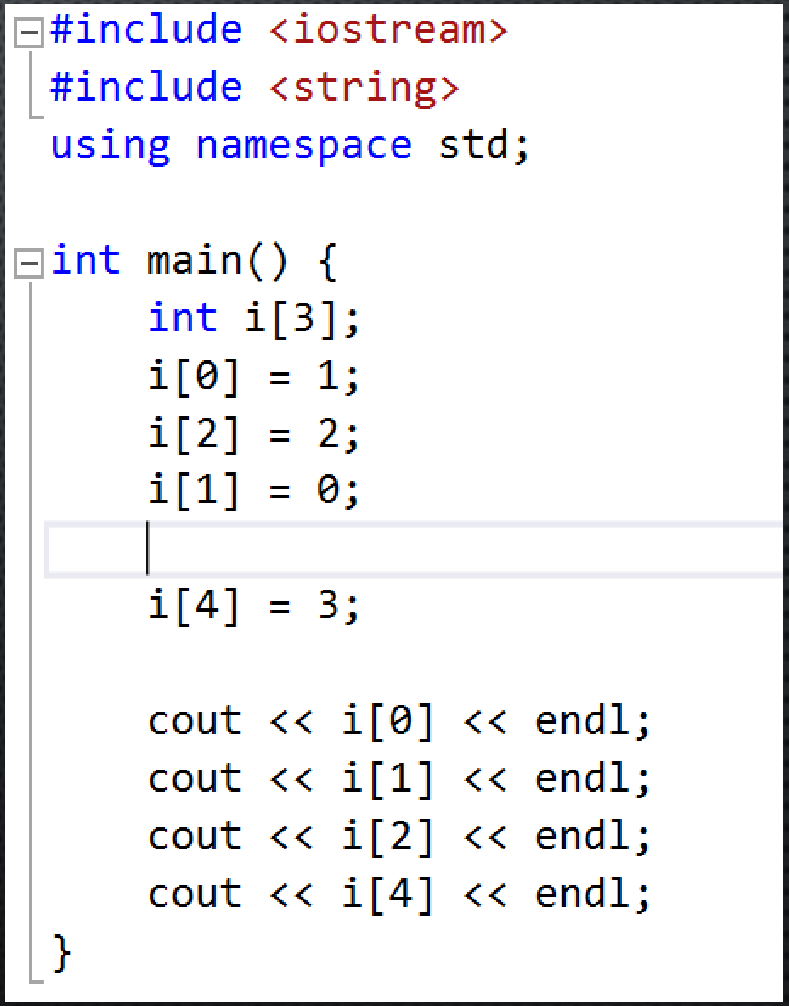
EXAMPLES



```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    int i[3] = { 1, 2, 3 };

    cout << i[0] << endl;
    cout << i[1] << endl;
    cout << i[2] << endl;
}
```



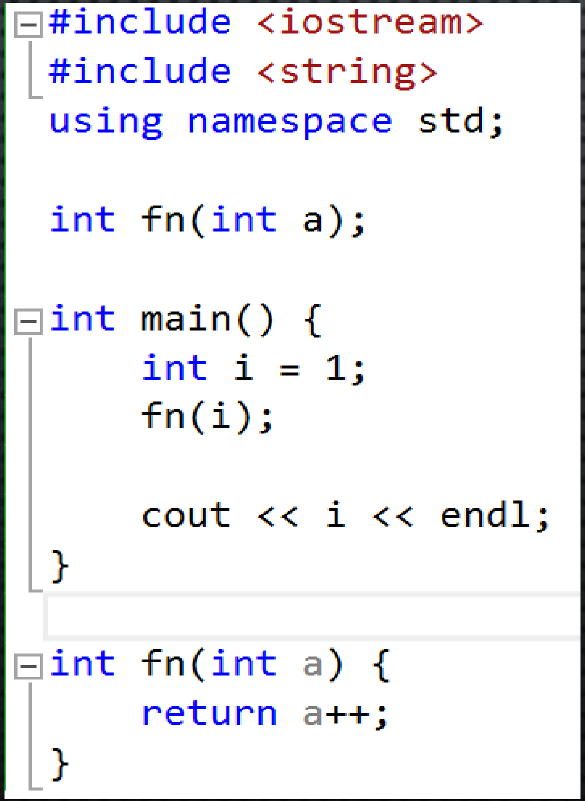
```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    int i[4] = { 5, 5, 5 };

    cout << i[0] << endl;
    cout << i[1] << endl;
    cout << i[2] << endl;

    cout << i[3] << endl;
}
```



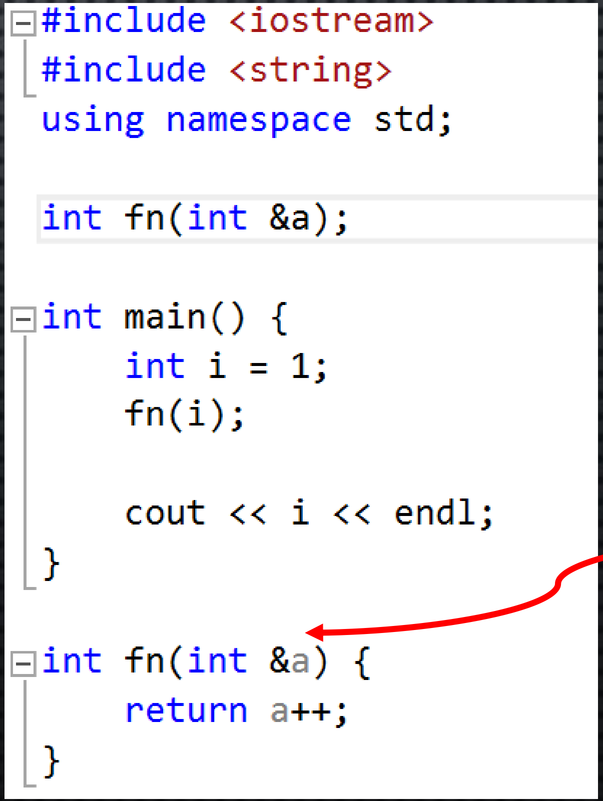
```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    int i[3];
    i[0] = 1;
    i[2] = 2;
    i[1] = 0;

    i[4] = 3;

    cout << i[0] << endl;
    cout << i[1] << endl;
    cout << i[2] << endl;
    cout << i[4] << endl;
}
```

ARRAYS & FUNCTIONS

- Reminder:
- Q: How do C++ functions pass arguments by default for basic types like int, double, and string?
  - A: **Pass by Value:** meaning the function parameters are **copies** of the arguments.



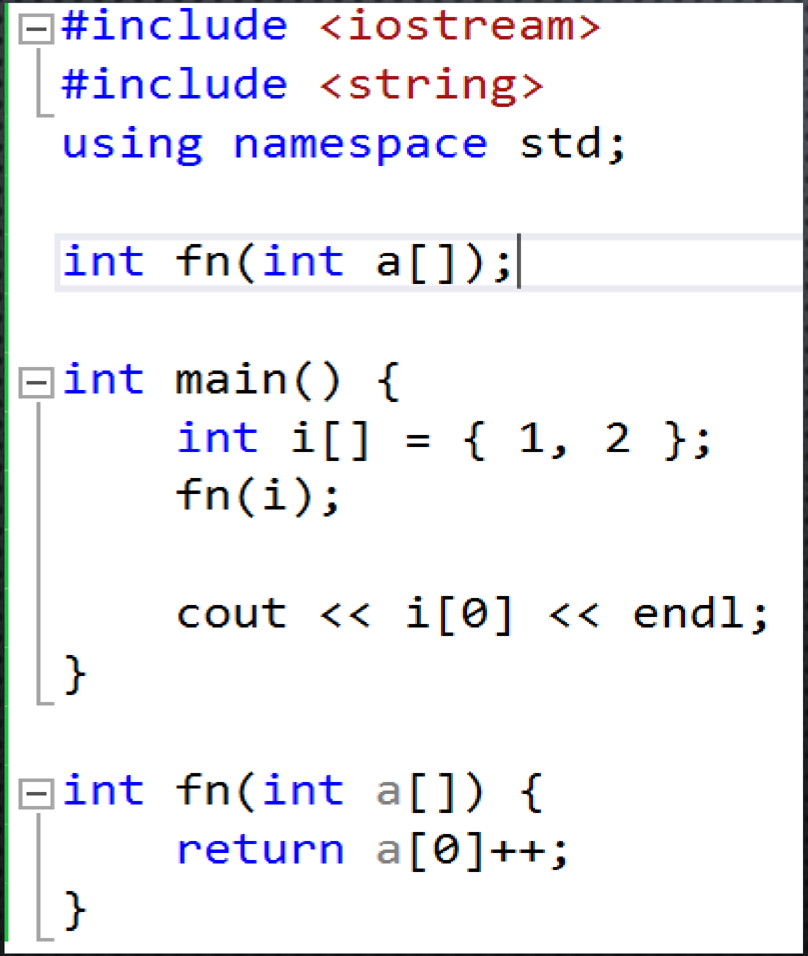
```cpp
#include <iostream>
#include <string>
using namespace std;

int fn(int a);

int main() {
    int i = 1;
    fn(i);

    cout << i << endl;
}

int fn(int a) {
    return a++;
}
```



```cpp
#include <iostream>
#include <string>
using namespace std;

int fn(int &a);

int main() {
    int i = 1;
    fn(i);

    cout << i << endl;
}

int fn(int &a) {
    return a++;
}
```

Pass by reference so the value of i is changed.

ARRAYS & FUNCTIONS - 2



```
#include <iostream>
#include <string>
using namespace std;

int fn(int a[]);

int main() {
    int i[] = { 1, 2 };
    fn(i);

    cout << i[0] << endl;
}

int fn(int a[]) {
    return a[0]++;
}
```

Output: 2

What does this tell us?

How do C++ functions pass arguments for arrays like int[], double[], and string[]?

Think of it as **passing by reference (though it's not):** meaning the function parameters are **pointers** to the argument; they refer to the same array in memory. (more on pointers later)

EXAMPLES

```
#include <iostream>
#include <string>
using namespace std;

int addItems(const int a[], int n);

int main () {
   int i[] = {1, 2, 3};
   int j =  addItems (i, 3);
   cout << j << endl;
}
int addItems (const int a[], int n) {
   int result = 0;
   for (int i = 0; i < n; i++) {
      result += a[i];
   }
   return result;
}
```

```
#include <iostream>
#include <string>
using namespace std;

int addItems(const int a[], int n);

int main() {
        int j = addItems({ 1, 2, 3 }, 3);
        cout << j << endl;
}
int addItems(const int a[], int n) {
        int result = 0;
        for (int i = 0; i < n; i++) {
        result += a[i];
                }
return result;
}
```

EXAMPLES

```
#include <iostream>
#include <string>
using namespace std;

void PlusOne(int yall[], int n);
int main() {
        const int i[] = { 1, 2, 3 };

        PlusOne(i, 3);
        cout << i[0] << endl;
        cout << i[1] << endl;
        cout << i[2] << endl;
}
void PlusOne(int a[], int n) {
        for (int i = 0; i < n; i++) {
        a[i]++;
        }
}
```

```
#include <iostream>
#include <string>
using namespace std;
void PlusOne(int yall[], int n);

int main() {
        int i[] = { 1, 2, 3 } ;// argument i has 3 elements
        PlusOne(i, 3);
        cout << i[0] << endl;
        cout << i[1] << endl;
        cout << i[2] << endl;
}
// the parameter lists size 2
void PlusOne(int a[2], int n)
{
        for (int i = 0; i < n; i++) {
                a[i]++; }
}
```

EXAMPLE

```
int A[5] = {11,22};              // initializes array values to 11,22,0,0,0
double Sum[7] = {0.0};       // initializes all 7 sums to 0.0
int B[ ] = {2,4,6,8};            // array B is assigned an array size of 4
char Vowels[8] = “aeiou”; // Vowels[0] = ‘a’, Vowels[1] = ‘e’, etc.
```

Array A
11
22
0
0
0

Array Sum
0
0
0
0
0
0
0

Array B
2
4
6
8

Array Vowels
a
e
i
o
u
\0
\0
\0

\0 is the “null character” →

PRACTICE

- Design a function intArraysEqual that compares **n** items starting at an index (called start) of two int arrays and determines (via bool return) if they are equivalent.

```
bool intArraysEqual(int arr1[], int arr2[], int start, int n)
```

- EX:

```
int i[] = {1, 2, 3};
int j[] = {0, 2, 3, 4, 5};
cout << intArraysEqual(i, j, 1, 2) << endl;
    // Above will print out: 1
    // because {2, 3} == {2, 3}
cout << intArraysEqual(i, j, 0, 2) << endl;
   // Above will print out: 0
   // because {1, 2} != {0, 2}
```

PRACTICE

- Design a function stringArrayPrint that does nothing but print the comma and space-separated values of the first n values of an array of strings.

EX:

```
 string s[] = {"print", “this", “here"};
     stringArrayPrint(s, 3);
     // Above will print out:
     // {"print", “this", “here"}

     stringArrayPrint(s, 0);
     // Above will print out:
     // {}
```