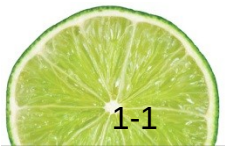
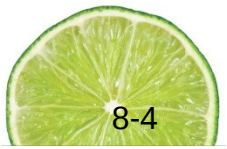
CSC2212
C++ Programming
Arrays
Lab Four



8.1 Arrays Hold Multiple Values

- Array: variable that can store multiple values of the same type
- Values are stored in consecutive memory locations
- Declared using `[]` operator

```
const int ISIZE = 5;
int tests[ISIZE];
```

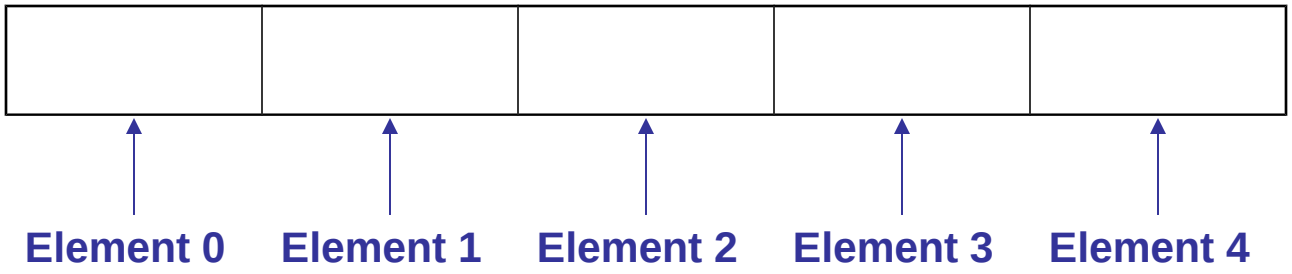



Array Storage in Memory

The definition

```
int tests[ISIZE];  // ISIZE is 5
```

allocates the following memory

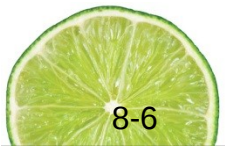


Array Terminology

In the definition `int tests[ISIZE];`

- **`int`** is the data type of the array elements
- **`tests`** is the name of the array
- **`ISIZE`**, in **`[ISIZE]`**, is the size declarator. It shows the number of elements in the array.
- The size of an array is the number of bytes allocated for it

*(number of elements) * (bytes needed for each element)*

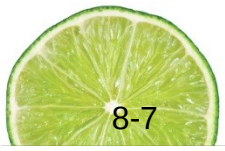


Array Terminology Examples

Examples:

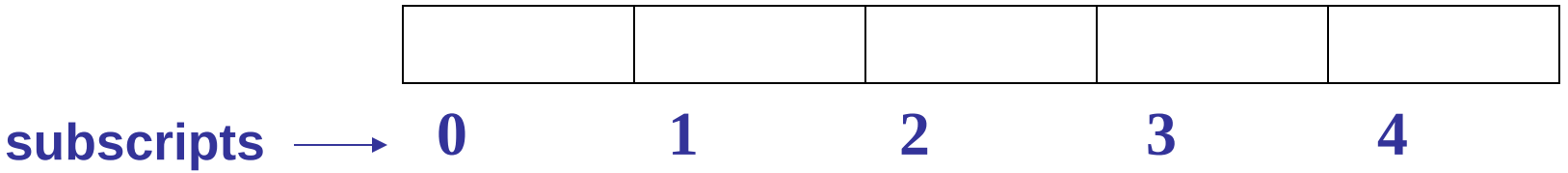
Assumes **`int`** uses 4 bytes and **`double`** uses 8 bytes

```
const int ISIZE = 5, DSIZE = 10;
int tests[ISIZE]; // holds 5 ints, array
                  // occupies 20 bytes
double volumes[DSIZE];// holds 10 doubles,
                      // array occupies
                      // 80 bytes
```



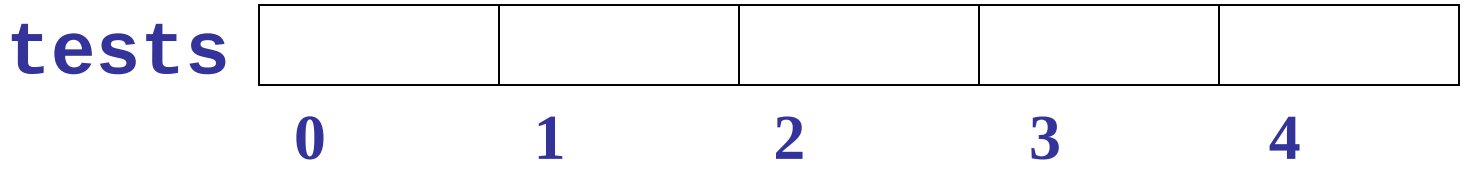
8.2 Accessing Array Elements

- Each array element has a subscript, used to access the element.
- Subscripts start at 0



Accessing Array Elements

Array elements (accessed by array name and subscript) can be used as regular variables

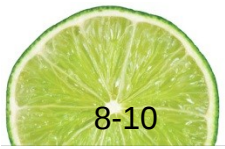



```
tests[0] = 79;
cout << tests[0];
cin  >> tests[1];
tests[4] = tests[0] + tests[1];
cout << tests; // illegal due to
               // missing subscript
```

8.3 Inputting and Displaying Array Contents

`cout` and **`cin`** can be used to display values from and store values into an array

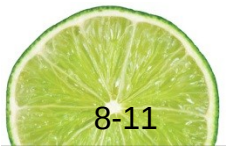
```
const int ISIZE = 5;

int tests[ISIZE]; // Define 5-elt. array
cout << "Enter first test score ";
cin  >>  tests[0];
```



Array Subscripts

- Array subscript can be an integer constant, integer variable, or integer expression
- Examples:

	Subscript is
`cin >> tests[3];`	int constant
`cout << tests[i];`	int variable
`cout << tests[i+j];`	int expression

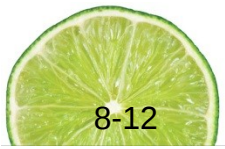


Accessing All Array Elements

To access each element of an array

- Use a loop
- Let the loop control variable be the array subscript
- A different array element will be referenced each time through the loop

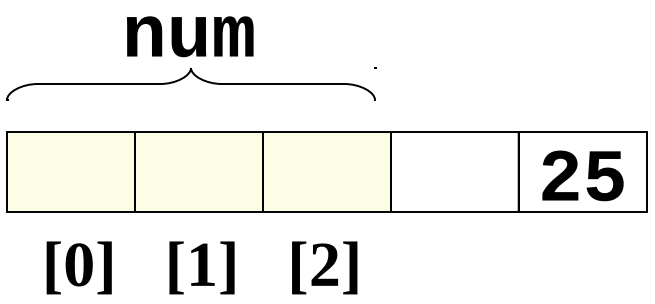
```
for (i = 0; i < 5; i++)
   cout << tests[i] << endl;
```



No Bounds Checking

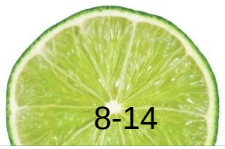
- There are no checks in C++ that an array subscript is in range
- An invalid array subscript can cause program to overwrite other memory
- Example:

```
const int ISIZE = 3;
int i = 4;
int num[ISIZE];
num[i] = 25;
```



Off-By-One Errors

- Most often occur when a program accesses data one position beyond the end of an array, or misses the first or last element of an array.
- Don’t confuse the ordinal number of an array element (first, second, third) with its subscript (0, 1, 2)



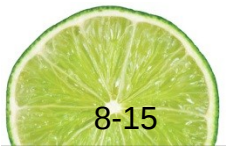
8.4 Array Initialization

- Can be initialized during program execution with assignment statements

```
tests[0] = 79;
tests[1] = 82; // etc.
```

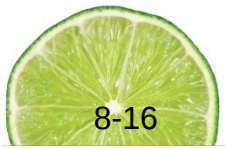
- Can be initialized at array definition with an initialization list

```
const int ISIZE = 5;
int tests[ISIZE] = {79,82,91,77,84};
```



Start at element 0 or 1?

- You may choose to declare arrays to be one larger than needed. This allows you to use the element with subscript 1 as the ‘first’ element, etc., and may minimize off-by-one errors.
- The element with subscript 0 is not used.
- This is most often done when working with ordered data, *e.g.*, months of the year or days of the week



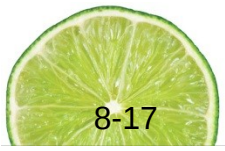
Partial Array Initialization

- If array is initialized at definition with fewer values than the size declarator of the array, remaining elements will be set to **0** or the empty string

```
int tests[ISIZE] = {79, 82};
```

79	82	0	0	0

- Initial values used in order; cannot skip over elements to initialize noncontiguous range
- Cannot have more values in initialization list than the declared size of the array



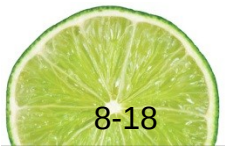
Implicit Array Sizing

- Can determine array size by the size of the initialization list

```
short quizzes[]={12,17,15,11};
```

12	17	15	11

- Must use either array size declarator or initialization list when array is defined



8.5 Processing Array Contents

- Array elements can be
  - treated as ordinary variables of the same type as the array
  - used in arithmetic operations, in relational expressions, etc.
- Example:

```
if (principalAmt[3] >= 10000)
  interest = principalAmt[3] * intRate1;
else
  interest = principalAmt[3] * intRate2;
```

Using Increment and Decrement Operators with Array Elements

When using **++** and **--** operators, don't confuse the element with the subscript

```
tests[i]++;  // adds 1 to tests[i]
tests[i++];  // increments i, but has
             // no effect on tests
```

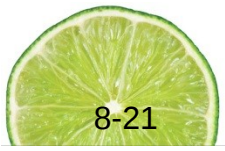
Copying One Array to Another

- Cannot copy with an assignment statement:

```
tests2 = tests;  //won't work
```

- Must instead use a loop to copy element-by-element:

```
for (int indx=0; indx < ISIZE; indx++)
   tests2[indx] = tests[indx];
```



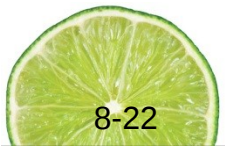
Are Two Arrays Equal?

- Like copying, cannot compare in a single expression:

```
if (tests2 == tests)
```

- Use a while loop with a boolean variable:

```
bool areEqual=true;
int indx=0;
while (areEqual && indx < ISIZE)
{
   if(tests[indx] != tests2[indx]
      areEqual = false;
}
```



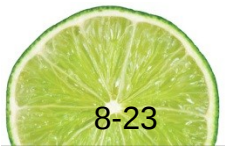
Sum, Average of Array Elements

- Use a simple loop to add together array elements

```
float average, sum = 0;
for (int tnum=0; tnum< ISIZE; tnum++)
    sum += tests[tnum];
```

- Once summed, average can be computed

```
average = sum/ISIZE;
```

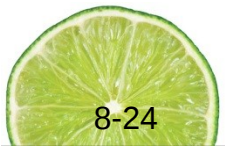


Largest Array Element

- Use a loop to examine each element and find the largest element (*i.e.*, one with the largest value)

```
int largest = tests[0];
for (int tnum = 1; tnum < ISIZE; tnum++)
{  if (tests[tnum] > largest)
       largest = tests[tnum];
}
cout << "Highest score is " << largest;
```

- A similar algorithm exists to find the smallest element

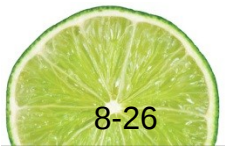


Using Arrays vs. Using Simple Variables

- An array is probably not needed if the input data is only processed once:
  - Find the sum or average of a set of numbers
  - Find the largest or smallest of a set of values
- If the input data must be processed more than once, an array is probably a good idea:
  - Calculate the average, then determine and display which values are above the average and which are below the average

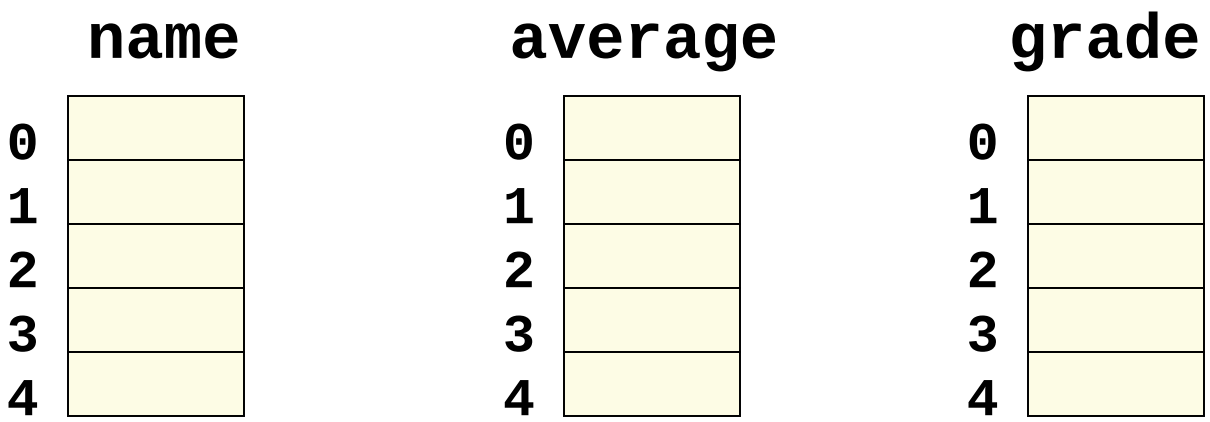
8.6 Using Parallel Arrays

- Parallel arrays: two or more arrays that contain related data
- Subscript is used to relate arrays
  - elements at same subscript are related
- The arrays do not have to hold data of the same type



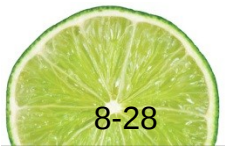
Parallel Array Example

```
const int ISIZE = 5;
string name[ISIZE];   // student name
float average[ISIZE]; // course average
char grade[ISIZE];    // course grade
```



Parallel Array Processing

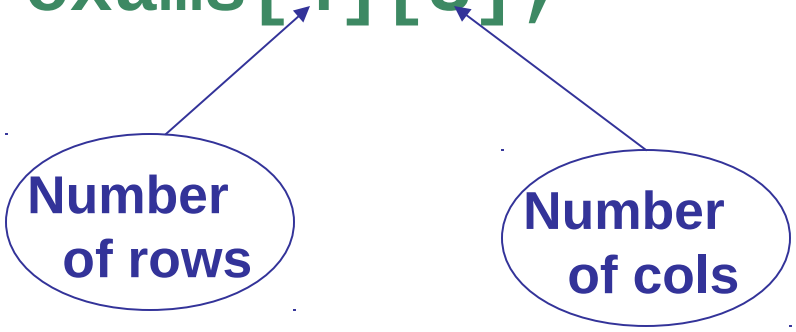
```
const int ISIZE = 5;
string name[ISIZE];   // student name
float average[ISIZE]; // course average
char grade[ISIZE];    // course grade
...
for (int i = 0; i < ISIZE; i++)
   cout << " Student: " << name[i]
        << " Average: " << average[i]
        << " Grade: "   << grade[i]
        << endl;
```



8.7 Two-Dimensional Arrays

- Can define one array for multiple sets of data
- Like a table in a spreadsheet
- Use two size declarators in definition

```
int exams[4][3];
```



Two-Dimensional Array Representation

```
int exams[4][3];
```

rows \ columns			
	exams[0][0]	exams[0][1]	exams[0][2]
	exams[1][0]	exams[1][1]	exams[1][2]
	exams[2][0]	exams[2][1]	exams[2][2]
	exams[3][0]	exams[3][1]	exams[3][2]

Use two subscripts to access element

```
exams[2][2] = 86;
```

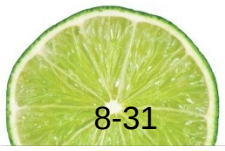
Initialization at Definition

- Two-dimensional arrays are initialized row-by-row

```
int exams[2][2] = { {84, 78},
                    {92, 97} };
```

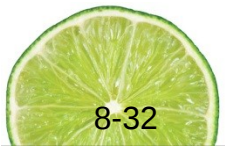
84	78
92	97

- Can omit inner **{ }**



2D Array Traversal

- Use nested loops, one for row and one for column, to visit each array element.
- Accumulators can be used to sum the elements row-by-row, column-by-column, or over the entire array.



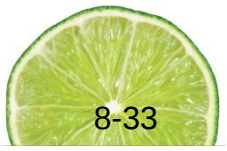
8.8Arrays with Three or More Dimensions

- Can define arrays with any number of dimensions

```
short rectSolid(2,3,5);
double timeGrid(3,4,3,4);
```

- When used as parameter, specify size of all but 1st dimension

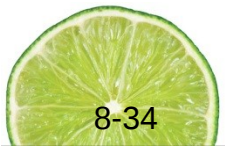
```
void getRectSolid(short [][3][5]);
```



8.9 Vectors

- Holds a set of elements, like an array
- Flexible number of elements - can grow and shrink
  - No need to specify size when defined
  - Automatically adds more space as needed
- Defined in the Standard Template Library (STL)
  - Covered in a later chapter
- Must include **`vector`** header file to use vectors

```
#include <vector>
```



Vectors

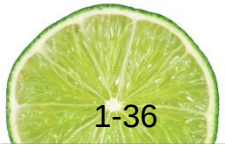
- Can hold values of any type
  - Type is specified when a vector is defined

```
vector<int> scores;
vector<double> volumes;
```

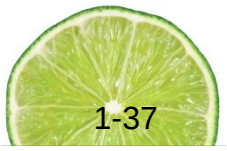
- Can use `[]` to access elements

Exercise 1

- Write program that stores employee work hours in an int array and uses one loop to read the hours and another loop to display them.



```
#include <iostream>
 using namespace std;
 int main()
 {
 const int NUM_EMPLOYEES = 6;
 int hours[NUM_EMPLOYEES];
// Holds hours worked for 6 employees
 int count; // Loop counter

 // Input the hours worked by each employee
 cout << "Enter the hours worked by " << NUM_EMPLOYEES << " employees: ";
 for (count = 0; count < NUM_EMPLOYEES; count++)
cin >> hours[count];

 // Display the contents of the array
 cout << "The hours you entered are:";
 for (count = 0; count < NUM_EMPLOYEES; count++)
 cout << " " << hours[count];
 cout << endl;
 return 0;
 }
```



Exercise 3

Modify exercise 1 to take the rate and compute the payment for each one and also store it in another array then display both the hours and corresponding payment