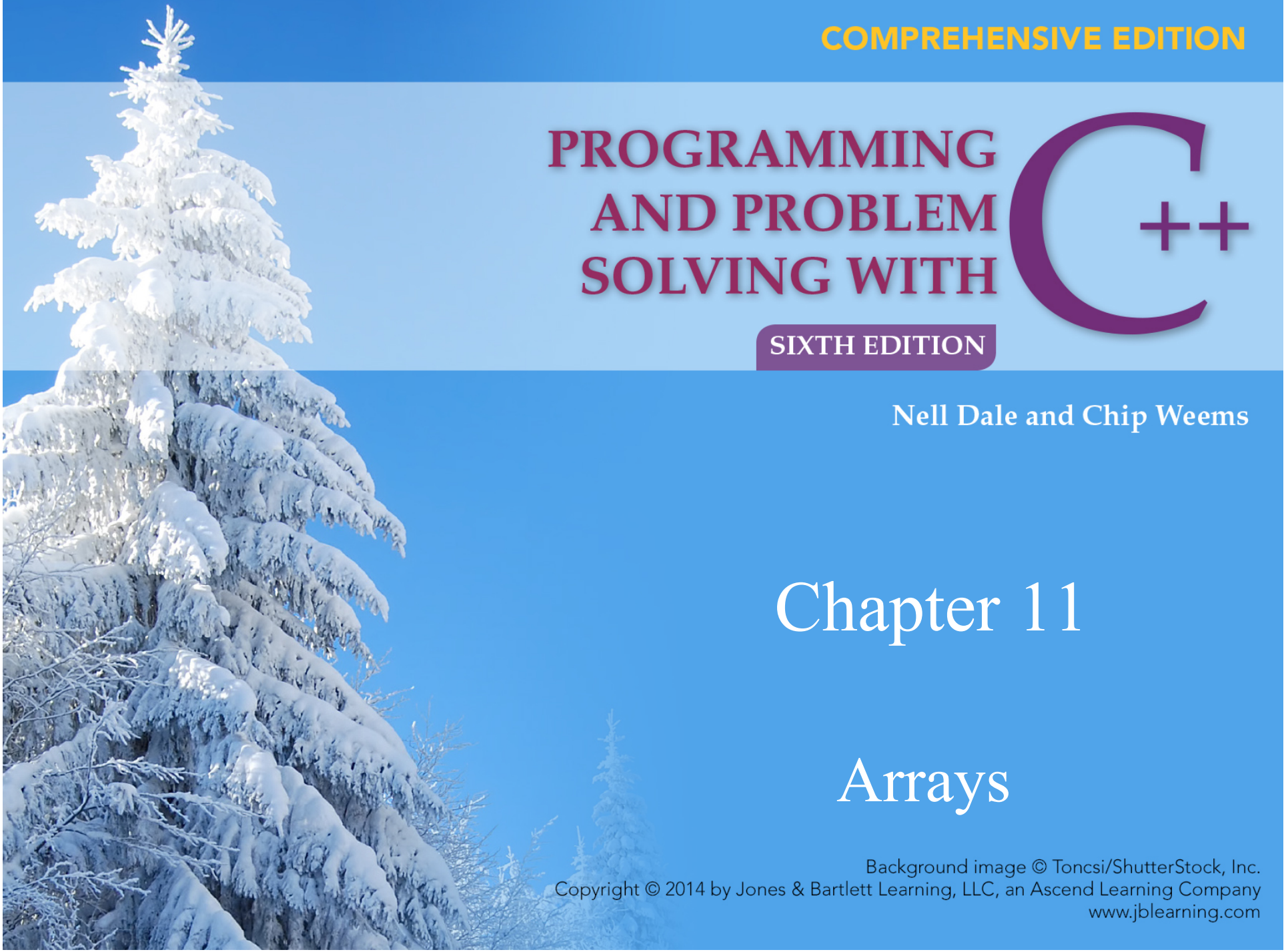


COMPREHENSIVE EDITION

PROGRAMMING AND PROBLEM SOLVING WITH C++

SIXTH EDITION

Nell Dale and Chip Weems

Chapter 11

Arrays

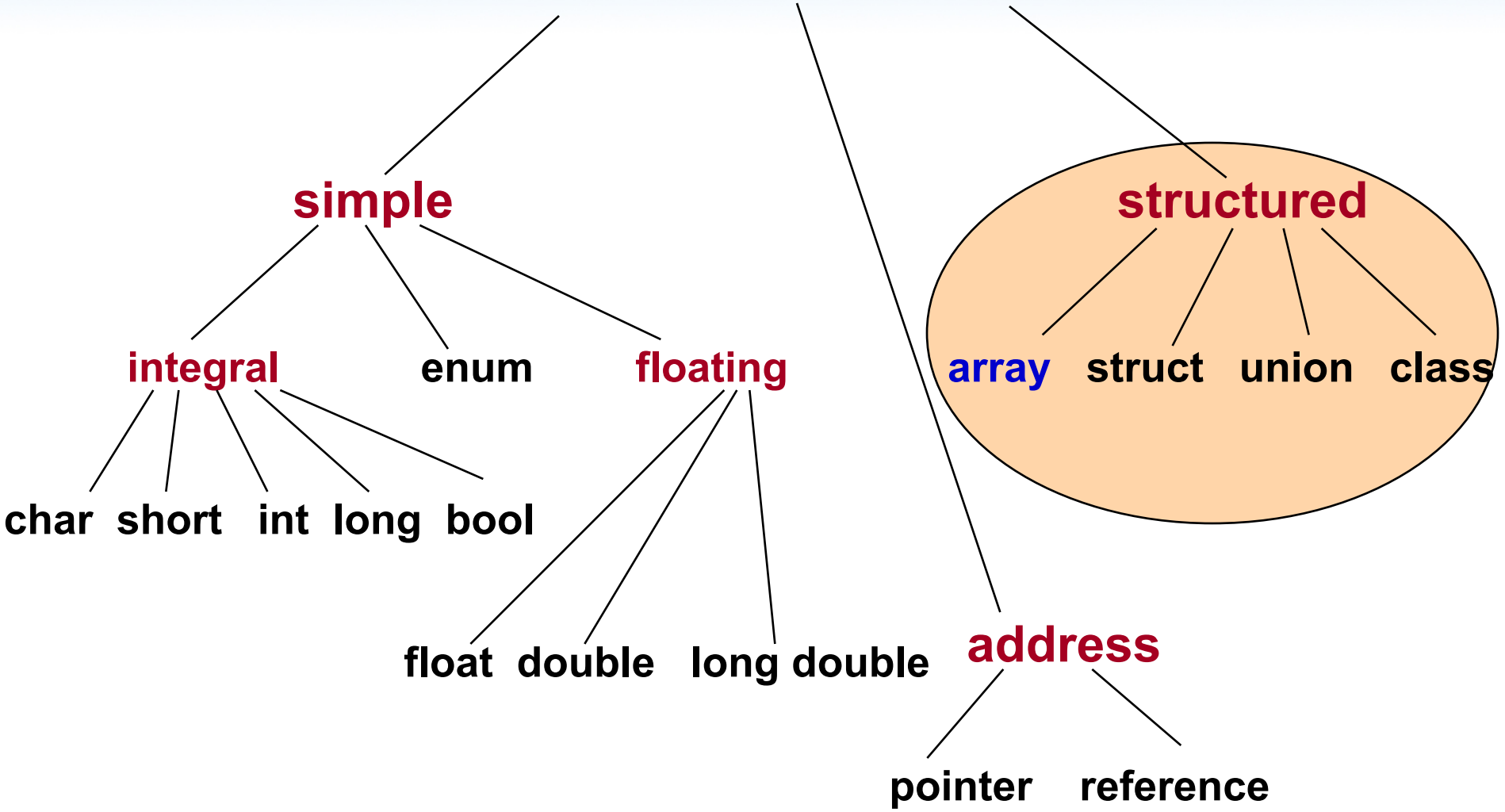
Chapter 11 Topics

- Declaring and Using a One-Dimensional Array
- Passing an Array as a Function Argument
- Using `const` in Function Prototypes
- Using an Array of `struct` or `class` Objects

Chapter 11 Topics

- **Using an `enum` Index Type for an Array**
- **Declaring and Using a Two-Dimensional Array**
- **Two-Dimensional Arrays as Function Parameters**
- **Declaring a Multidimensional Array**

C++ Data Types



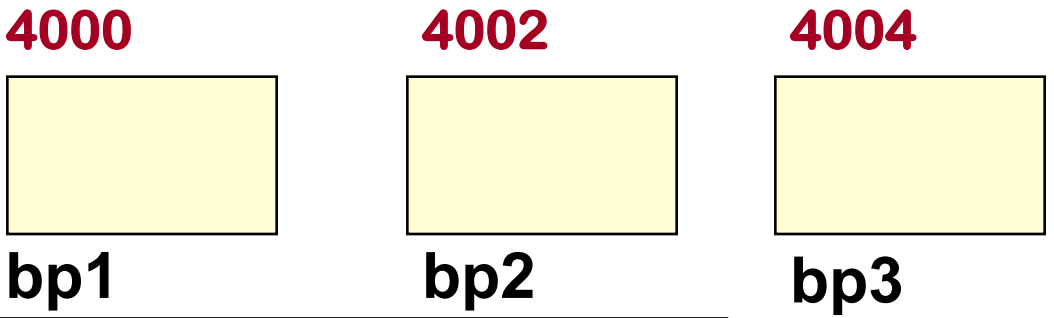
Structured Data Type

A **structured data type** is a type that

- Stores a collection of individual components with one variable name
- And allows individual components to be stored and retrieved by their position within the collection

Declare variables to store and total 3 blood pressures

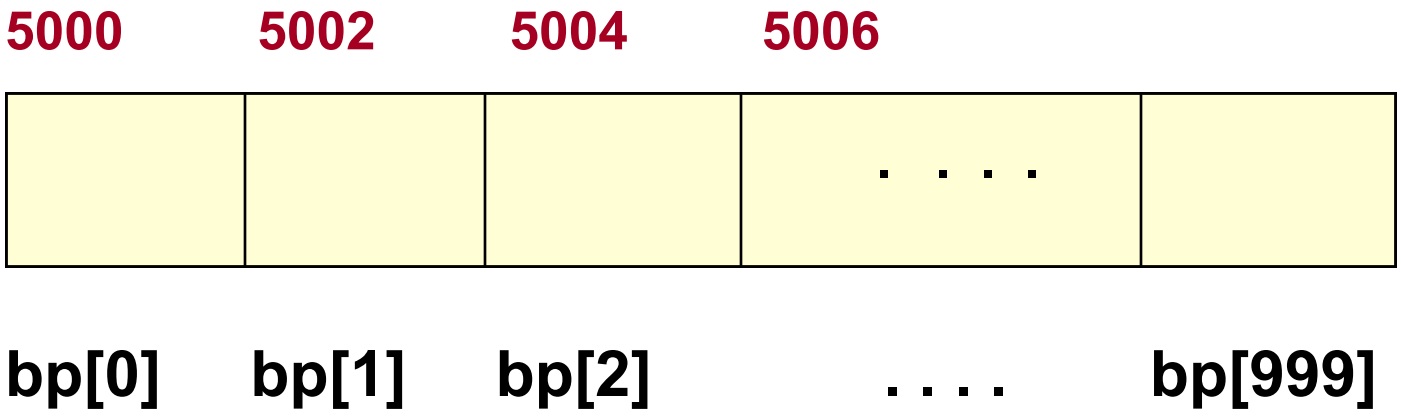
```
int  bp1,  bp2,  bp3;
int  total;
```




```
cin >> bp1 >>  bp2 >>  bp3;
total =   bp1 +  bp2 +  bp3;
```

What if you wanted to store and total 1000 blood pressures?

```
int  bp[1000];
// Declares an array of 1000 int values
```



One-Dimensional Array Definition

An array is a structured collection of components (called array elements):

Arrays are all of the same data type, given a single name, and stored in adjacent memory locations

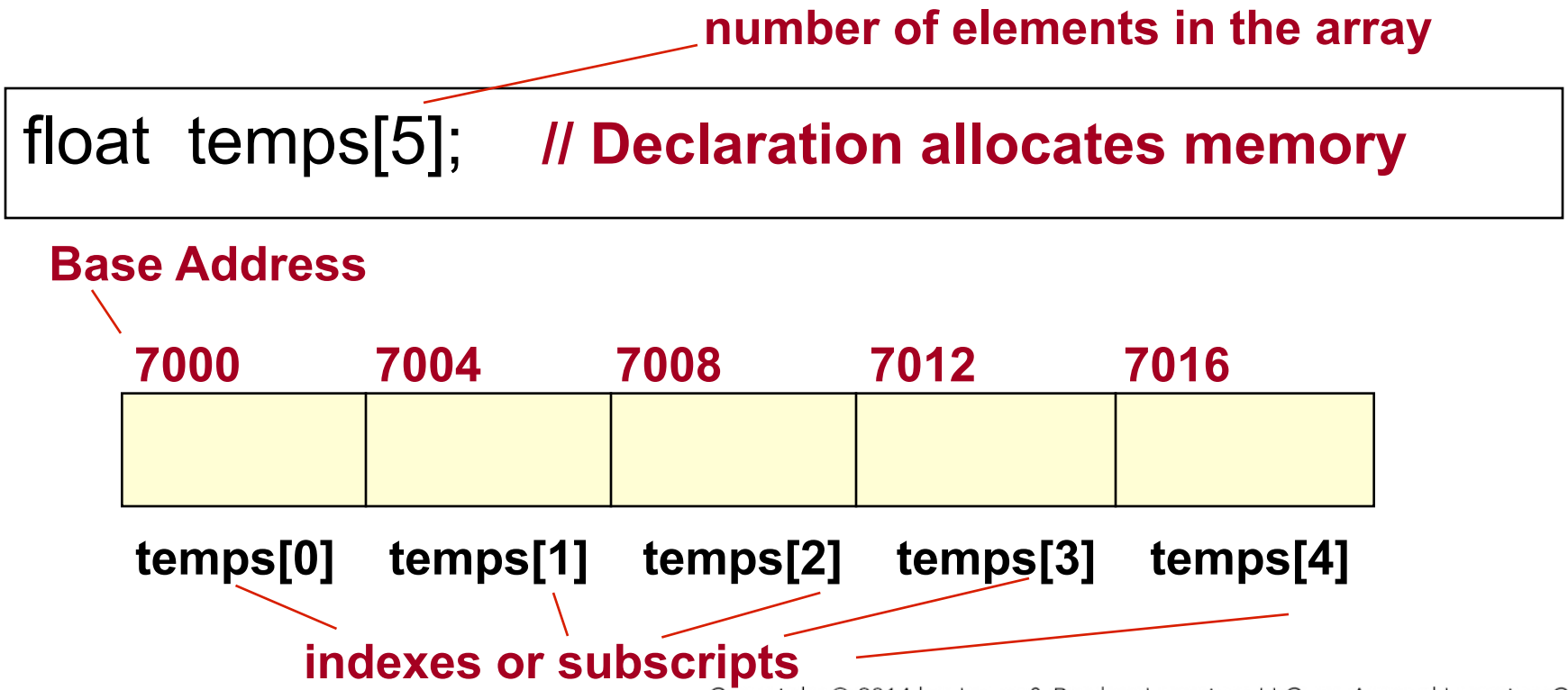
One Dimensional Array Definiton, cont...

The individual components are accessed by using the array name together with an integral valued index in square brackets

The index indicates the position of the component within the collection

Another Example

- **Declare an array called `temps` which will hold up to 5 individual float values**



Declaration of an Array

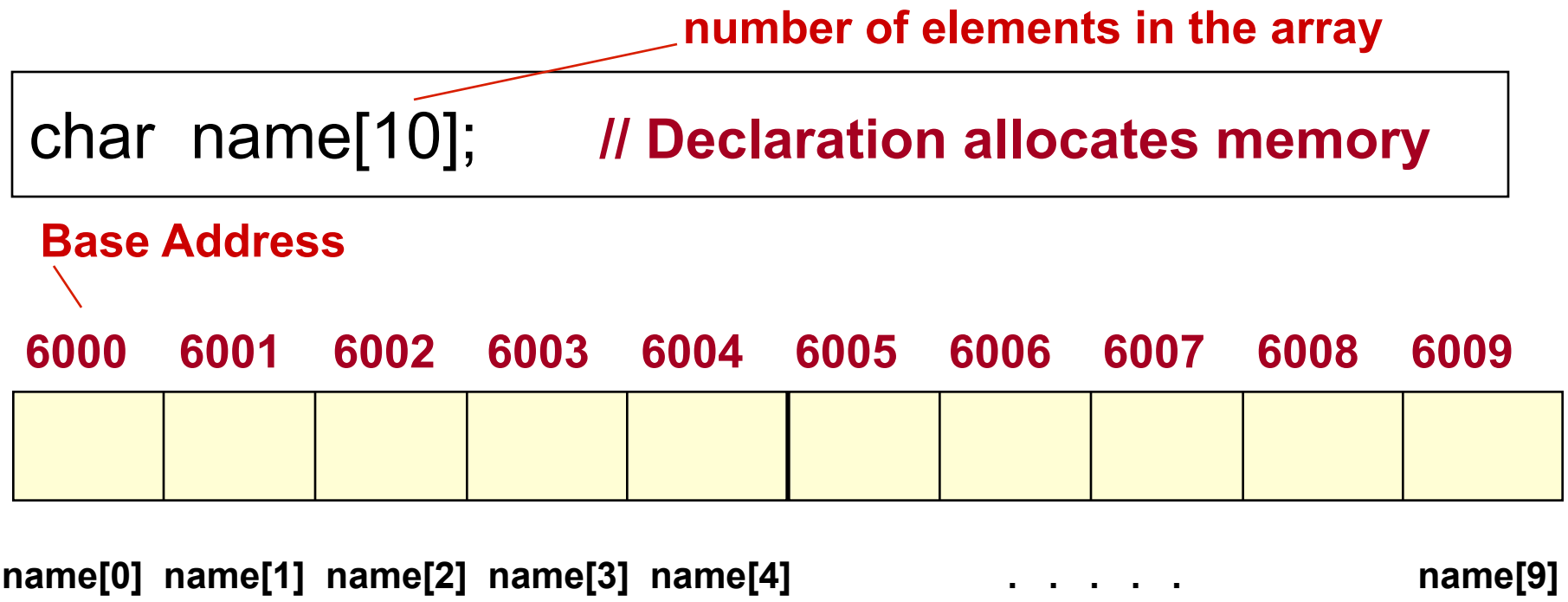
- The index is also called the subscript
- In C++, the first array element always has subscript 0, the second array element has subscript 1, etc.
- The base address of an array is its beginning address in memory

SYNTAX

```
DataType  ArrayName[ConstIntExpression];
```

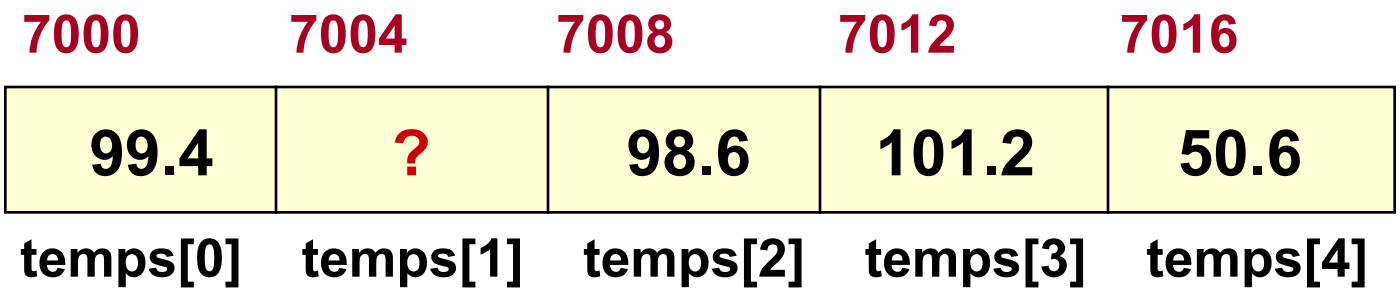
Yet Another Example

- **Declare an array called `name` which will hold up to 10 individual `char` values**



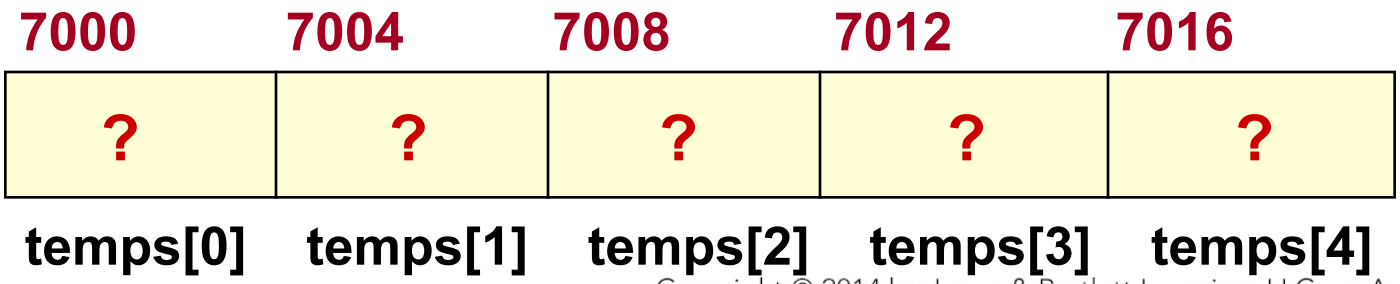
Assigning Values to Individual Array Elements

```
float temps[5]; int m = 4; // Allocates memory
temps[2] = 98.6;
temps[3] = 101.2;
temps[0] = 99.4;
temps[m] = temps[3] / 2.0;
temps[1] = temps[3] - 1.2;
// What value is assigned?
```



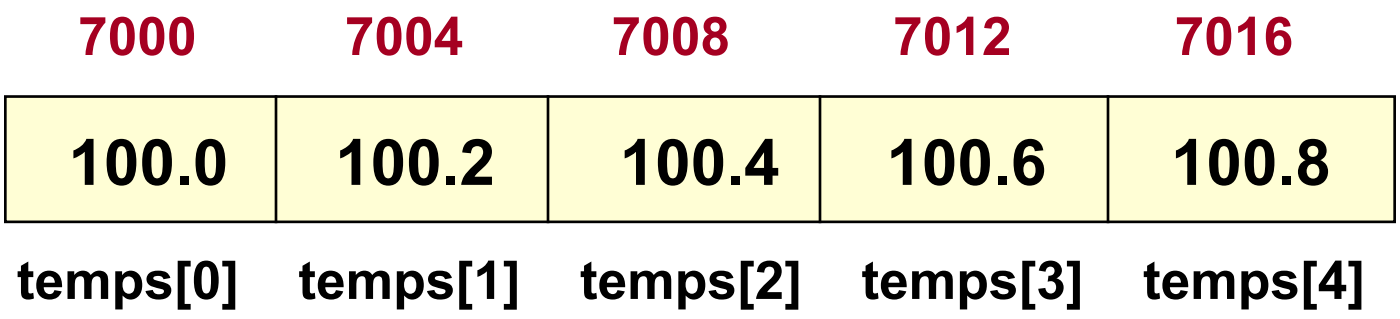
What values are assigned?

```
float temps[5]; // Allocates memory
int m;

for (m = 0; m < 5; m++)
{
     temps[m] = 100.0 + m * 0.2 ;
}
```



Now what values are printed?

```
float temps[5];  // Allocates memory
Int m;
 . . . . .
for (m = 4; m >= 0; m--)
{
     cout  <<  temps[m] << endl;
}
```

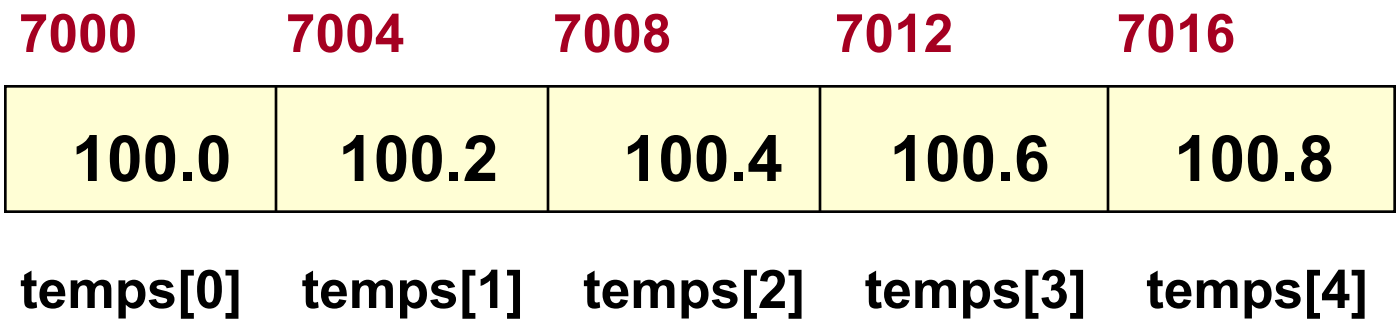


Variable Subscripts

```
float temps[5];  // Allocates memory
int m = 3;
 . . . . . .
```

What is temps[m + 1] ?

What is temps[m] + 1 ?

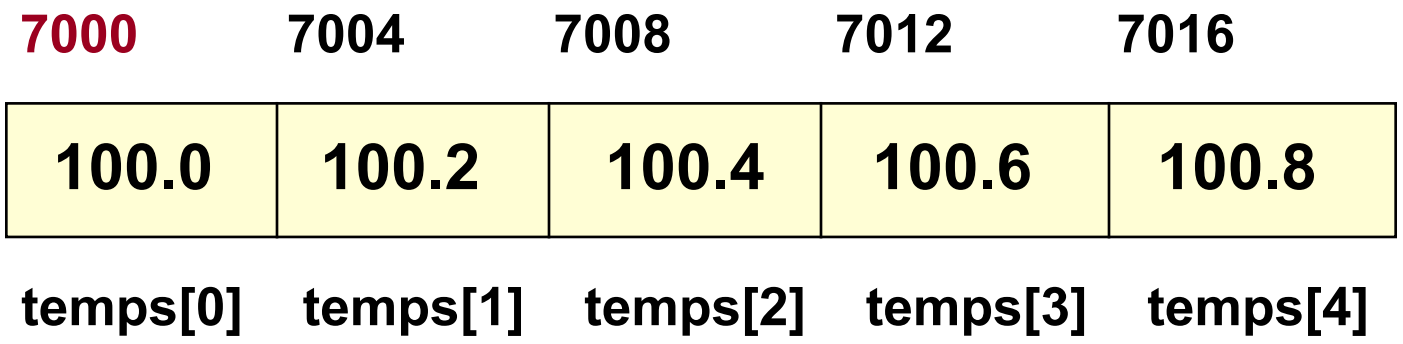


A Closer Look at the Compiler

```
float temps[5]; // Allocates memory
```

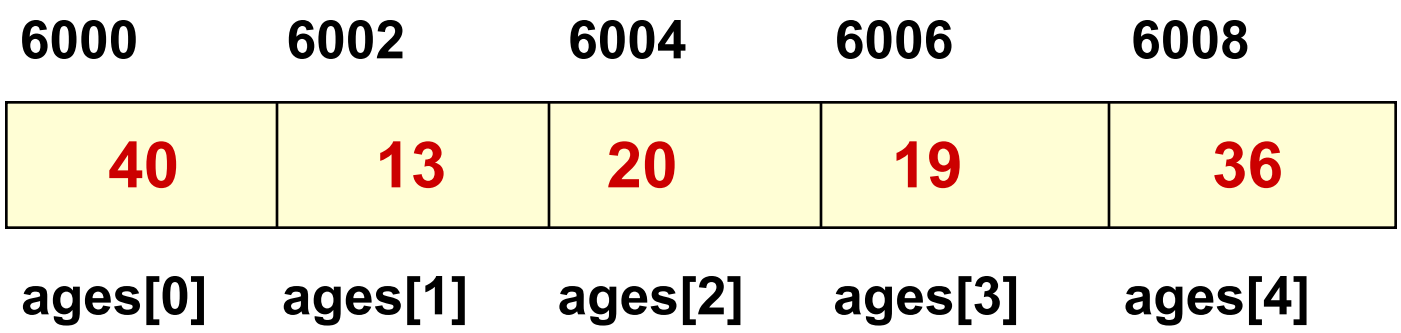
To the compiler, the value of the identifier `temps` is the base address of the array

We say `temps` is a pointer (because its value is an address); it "points" to a memory location



Initializing in a Declaration

```
int ages[5] ={ 40, 13, 20, 19, 36 };

for (int m = 0; m < 5; m++)
{
    cout  << ages[m];
}
```



Passing Arrays as Arguments

- In C++, arrays are *always* passed by reference
- Whenever an array is passed as an argument, its base address is sent to the called function

In C++, No Aggregate Array Operations

- The only thing you can do with an entire array as a whole (aggregate) is to pass it as an argument to a function

- Exception: aggregate I/O is permitted for C strings (special kinds of char arrays)

Using Arrays as Arguments to Functions

Generally, functions that work with arrays require two items of information:

- **The beginning memory address of the array (base address) and**
- **The number of elements to process in the array**

Example with Array Parameters

```
#include <iomanip>
#include <iostream>
void  Obtain (int[], int); // Prototypes here
void  FindWarmest (const  int[],  int , int&);
void  FindAverage  (const  int[],  int , int&);
void  Print (const  int[], int);

using  namespace  std;
```

Example continued...

```
int main ( )
{
    // Array to hold up to 31 temperatures
    int  temp[31
    int  numDays;
    int  average;
    int  hottest;
    int  m;
```

Example continued

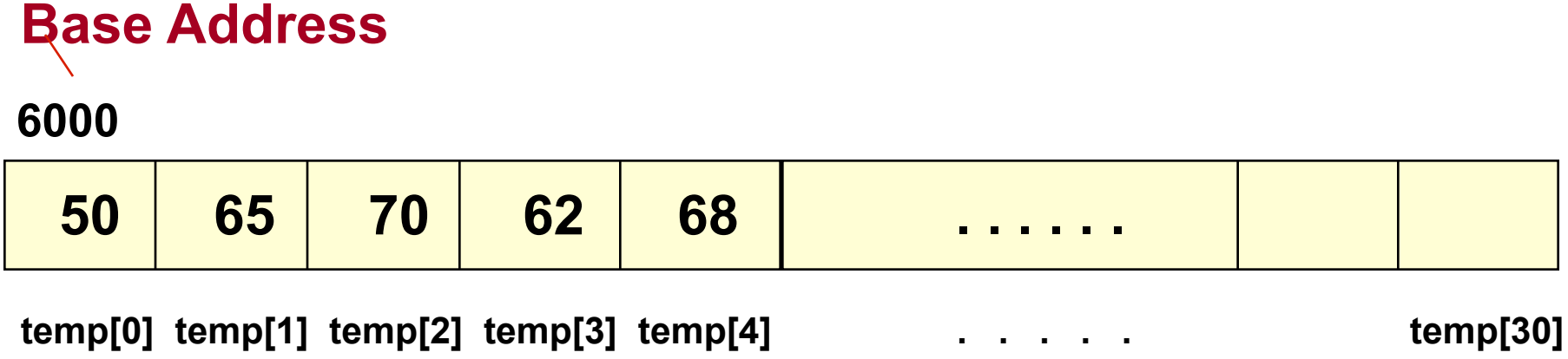
```
cout  <<  "How many daily temperatures? ";
cin  >>  numDays;

 Obtain(temp, numDays);
  // Call passes value of numDays
  // and address temp
cout  <<  numDays  <<  " temperatures"
      << endl;
Print (temp, numDays);
```

Example continued...

```
FindAverage (temp, numDays, average);
FindWarmest (temp, numDays, hottest);

cout  <<  endl   <<  "Average was:  "
      << average << endl;
cout  <<  "Highest was:  "
      << hottest  << endl;
    return 0;
}
```

Memory Allocated for Array

```
// Array to hold up to 31 temperatures
int temp[31];
```



```
void Obtain ( /* out */  int  temp[] ,
              /* in */   int  number )

// User enters number temperatures at keyboard

// Precondition:
//    number is assigned  &&  number > 0
// Postcondition:
//    temp[0 . . number -1]  are assigned
```

```
{
    int  m;

    for (m = 0; m < number;  m++)
    {
        cout << "Enter a temperature : ";
        cin >>  temp[m];
    }
}
```

```
void Print ( /* in */  const  int  temp[],
            /* in */  int  number )

// Prints number  temperature values to screen
// Precondition:
//    number is assigned  &&  number > 0
//    temp[0 . . number -1] are assigned
// Postcondition:
//    temp[0 . . number -1] printed 5 per line
```

```
{
   int  m;
   cout  <<   “You entered: “;
   for (m = 0; m < number;  m++)
   {
       if  (m % 5 == 0)
          cout  <<  endl;
           cout  <<  setw(7) << temp[m];
   }
}
```

Use of `const`

- Because the identifier of an array holds the base address of the array, & is never needed for an array in the parameter list :
- Arrays are always passed by reference
- To prevent elements of an array used as an argument from being unintentionally changed by the function:
- You place const in the function prototype and heading

Use of `const` in prototypes

Do not use const with outgoing array because function is supposed to change array values

```
void  Obtain (int[], int);

void  FindWarmest (const  int[],  int , int &);

void  FindAverage (const  int[],  int , int &);

void  Print (const  int[], int);
```

use const with incoming array values to prevent unintentional changes by function

Example, cont...

```
void FindAverage( /* in */ const int  temp[],
                  /* in */   int  number,
                  /* out */  int &  avg)
// Determines average of temp[0 . . number-1]
// Precondition:
//   number is assigned  &&  number > 0
//   temp[0 . . number -1] are assigned
// Postcondition:
//   avg == average of temp[0 . . number-1]
```

Example, cont...

```
{
      int  m;
      int  total = 0;
      for (m = 0; m < number;  m++)
      {
          total = total + temp[m];
      }
      avg =
          int (float(total) / float(number) + .5);
}
```

Another Example

```
void  FindWarmest ( /* in */ const int temp[],
                    /* in */       int   number,
                    /* out */      int& largest)


// Determines largest of temp[0 . . number-1]
// Precondition:
//    number is assigned  &&  number > 0
//    temp[0 . . number -1] are assigned
// Postcondition:
//    largest== largest value in
//    temp[0 . . number-1]
```

Another Example, cont...

```
{
    int  m;
    // Initialize to first element
    largest = temp[0];

    for (m = 0; m < number;  m++)
    {
        if (temp[m]  > largest)
            largest  =  temp[m];
    }
}
```

Using Arrays for Counters

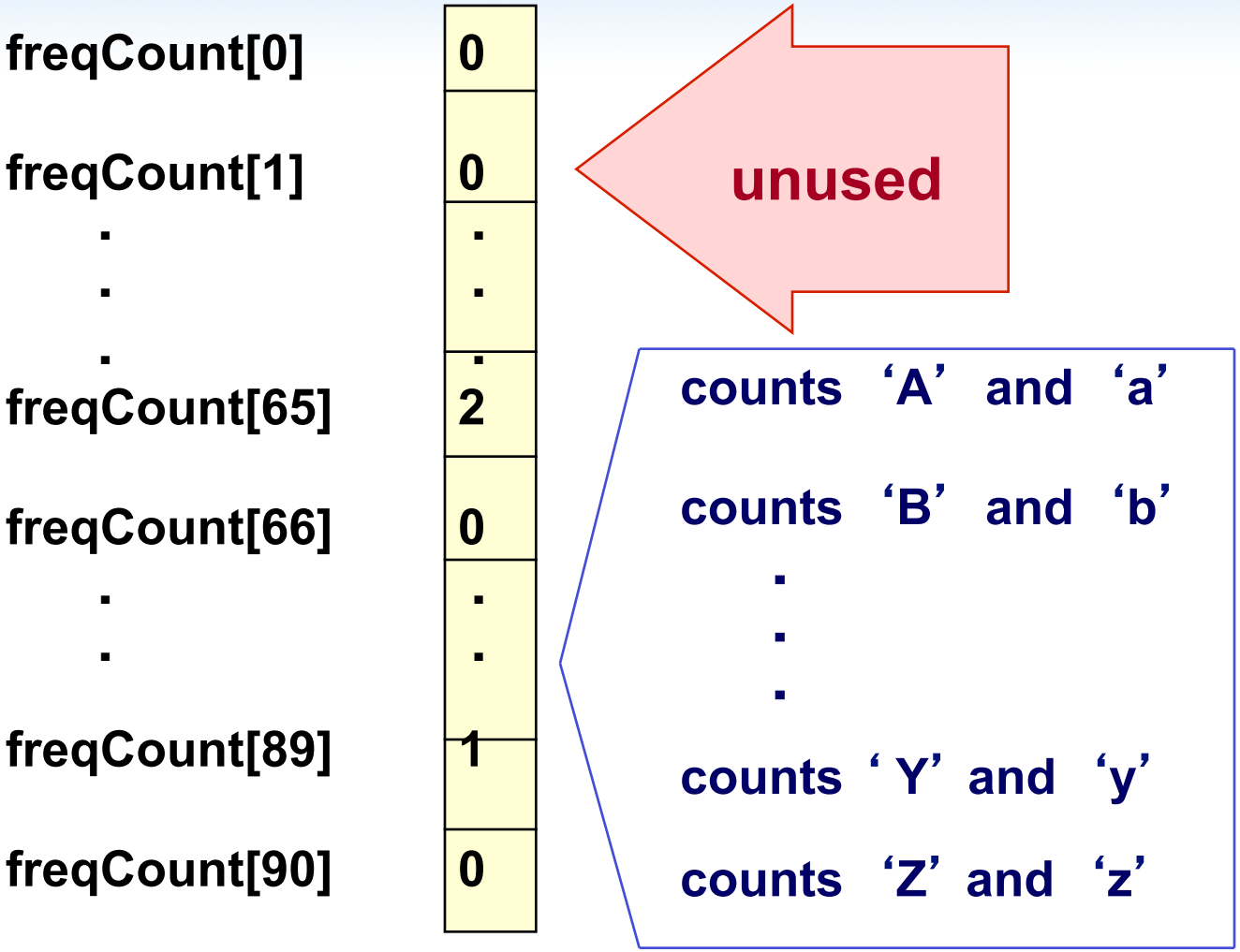
- **Write a program to count the number of each alphabetic letter in a text file**

letter	ASCII
'A'	65
'B'	66
'C'	67
'D'	68
.	.
.	.
.	.
'Z'	90

A:\my.dat

This is my text file.
It contains many things!
5 + 8 is not 14.
Is it?

```
const int SIZE 91;
int freqCount[SIZE];
```



Main Module Pseudocode

Level 0

```
Open dataFile (and verify success)
Zero out freqCount
Read ch from dataFile
WHILE NOT EOF on dataFile
   If ch is alphabetic character
        If ch is lowercase alphabetic
              Change ch to uppercase
        Increment freqCount[ch] by 1
   Read ch from dataFile
Print characters and frequencies
```

Counting Frequency of Alphabetic Characters

```
// Program counts frequency of each alphabetic
//   character in text file.

#include < fstream >
#include < iostream >
#include < cctype >

const int SIZE=91;
void PrintOccurrences(const  int[]); // Prototype
```

Counting Frequency of Alphabetic Characters

```
int  main ()
{
    ifstream dataFile;
    int  freqCount[SIZE];
    char ch;
    char index;
```

Counting Frequency of Alphabetic Characters

```
dataFile.open ("my.dat"); // Open
 if  (! dataFile)          // Verify success
 {
       cout  <<  " CAN'T OPEN INPUT FILE ! "
                 << endl;
       return  1;
      }
     for ( int  m = 0; m < SIZE;  m++) // Zero array
         freqCount[m]  =  0;
```

Counting Frequency of Alphabetic Characters

```
// Read file one character at a time
    dataFile.get (ch);  // Priming read
    while (dataFile)    // While read successful
     {
        if (isalpha (ch))
        {
            if (islower (ch))
                ch  =  toupper (ch);

            freqCount[ch]  =  freqCount[ch] + 1;
```

Counting Frequency of Alphabetic Characters

```
}
        dataFile. get (ch); // Get next character
    }
   PrintOccurrences (freqCount);
   return  0;
}
```

Counting Frequency of Alphabetic Characters

```
void PrintOccurrences (
    /* in */ const int  freqCount [])
// Prints each alphabet character and its frequency
// Precondition:
//    freqCount['A' . . 'Z'] are assigned
// Postcondition:
//    freqCount['A' . . 'Z'] have been printed
```

Counting Frequency of Alphabetic Characters

```
{
  char  index;
  cout  <<  "File contained "  << endl;
  cout  <<  "LETTER      OCCURRENCES"  << endl;
  for   ( index = 'A' ;  index < = 'Z';  index ++)
  {
      cout << setw(4) << index << setw(10)
           << freqCount[index] << endl;
  }
}
```

More about Array Indexes

- Array indexes can be any integral type including `char` and `enum` types
- The index must be within the range 0 through the declared array size minus one
- It is the programmer's responsibility to make sure that an array index does not go out of bounds

More About Array Indexes

- The index value determines which memory location is accessed
- Using an index value outside this range causes the program to access memory locations outside the array

Array with enum Index Type

DECLARATION

```
enum Department { WOMENS,  MENS,  CHILDRENS,
                  LINENS, HOUSEWARES,
  ELECTRONICS };
float  salesAmt[6];
Department which;
```

USE

```
for (which = WOMENS; which <= ELECTRONICS;
  which = Department(which + 1))
    cout  <<  salesAmt[which] << endl;
```

float salesAmt[6];

salesAmt[WOMENS] **(i. e. salesAmt[0])**

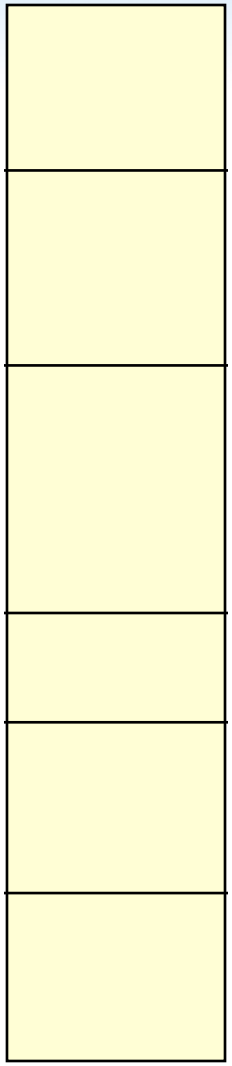
salesAmt[MENS] **(i. e. salesAmt[1])**

salesAmt[CHILDRENS] **(i. e. salesAmt[2])**

salesAmt[LINENS] **(i. e. salesAmt[3])**

salesAmt[HOUSEWARES] **(i. e. salesAmt[4])**

salesAmt[ELECTRONICS] **(i. e. salesAmt[5])**



Parallel Arrays

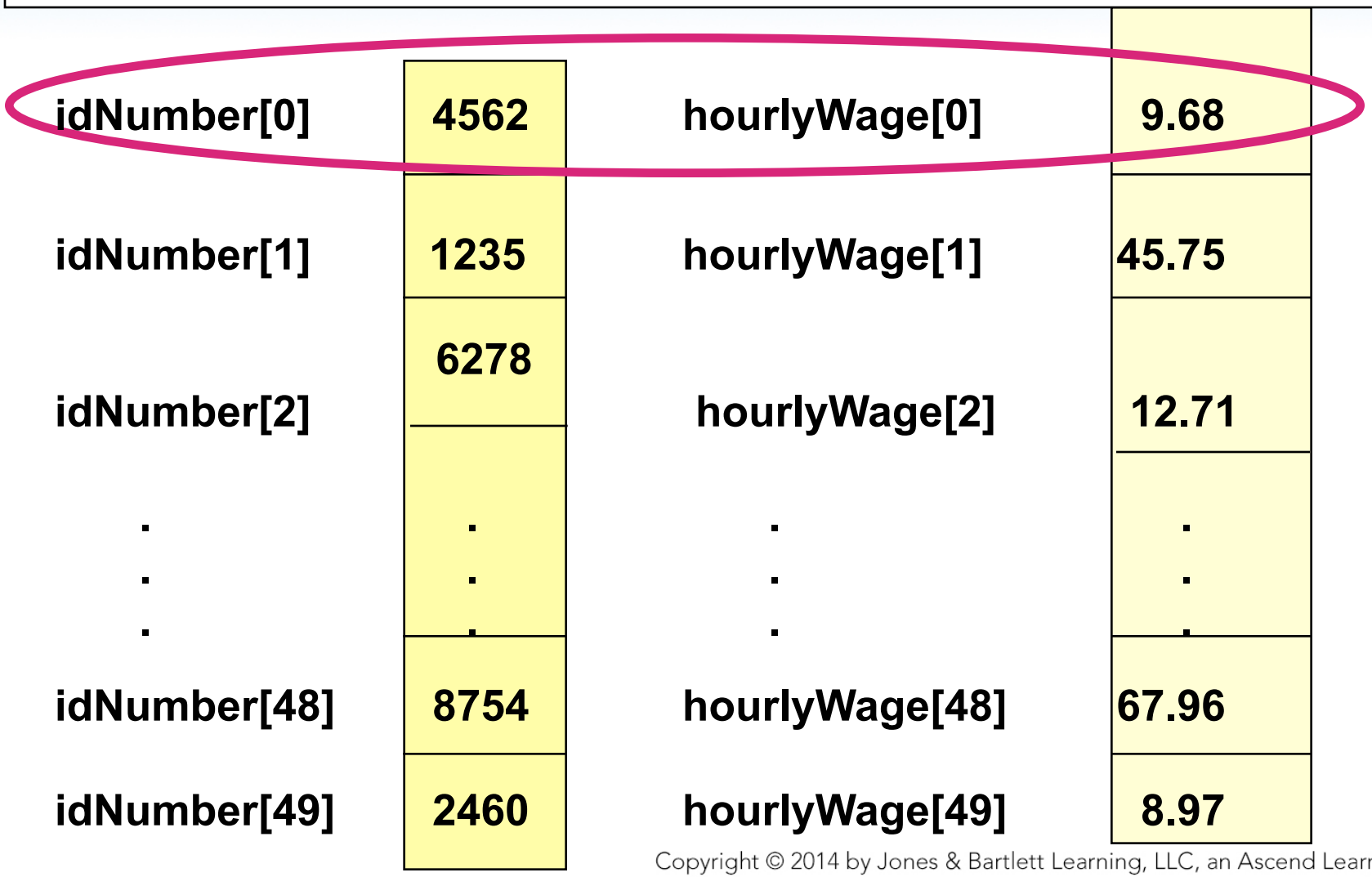
Parallel arrays are two or more arrays that have the same index range and whose elements contain related information, possibly of different data types

EXAMPLE

```
const  int  SIZE  50;
int      idNumber[SIZE];
float    hourlyWage[SIZE];
```

parallel arrays

```
const  int  SIZE  50;
int      idNumber[SIZE];       // Parallel arrays hold
float    hourlyWage[SIZE];     // Related information
```



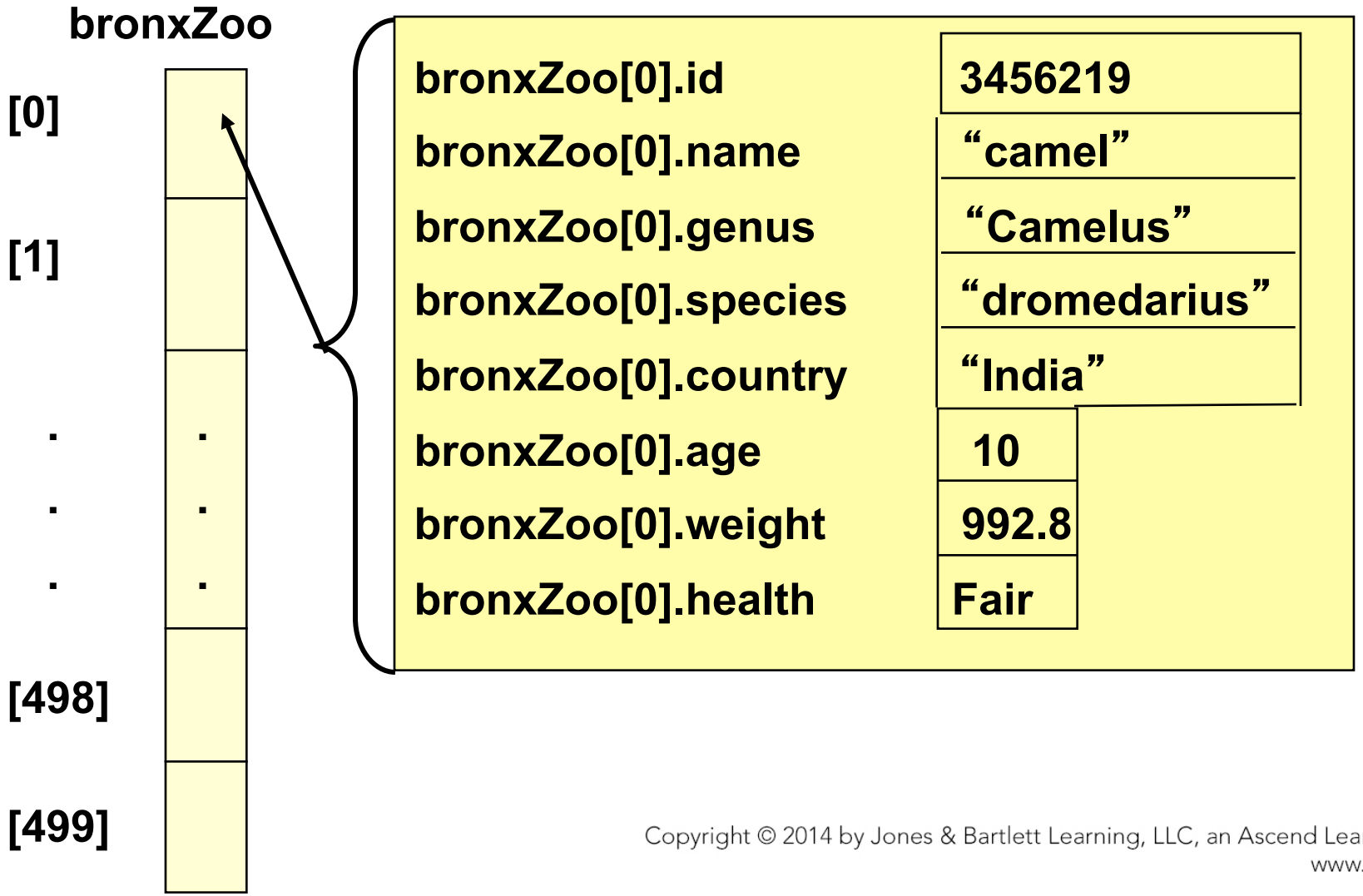
Array of Structures

```
const  int  MAX_SIZE = 500;

enum  HealthType { POOR, FAIR, GOOD,
  EXCELLENT };

struct  AnimalType  // Declares struct type
```

Array of Structures, cont...

```
{
    long    id;
    string  name;
    string  genus;
    string  species;
    string  country;
    int     age;
    float   weight;
    HealthType health;
};

// Declares array
AnimalType  bronxZoo[MAX_SIZE];
```

AnimalType bronxZoo[MAX_SIZE];



`AnimalType bronxZoo[MAX_SIZE];`

	.id	.name	.genus	.species	.country	.age	.weight	.health
bronxZoo[0]	3456219	“camel”	“Camelus”	“dromedarius”	“India”	10	992.8	Fair
bronxZoo[1]								
bronxZoo[2]								
bronxZoo[3]								
.		.						
.		.						
.		.						
bronxZoo[498]								
bronxZoo[499]								

Add 1 year to the age member of each element of the bronxZoo array

```
for (j = 0; j < MAX_SIZE; j++)
      bronxZoo[j].age = bronxZoo[j].age + 1;
```

OR,

```
for (j = 0; j < MAX_SIZE; j++)
      bronxZoo[j].age++;
```

Find total weight of all elements of the `bronxZoo` array

```
float  total = 0.0;

for (j = 0; j < MAX_SIZE; j++)
    total  +=
  bronxZoo[j].weight;
```

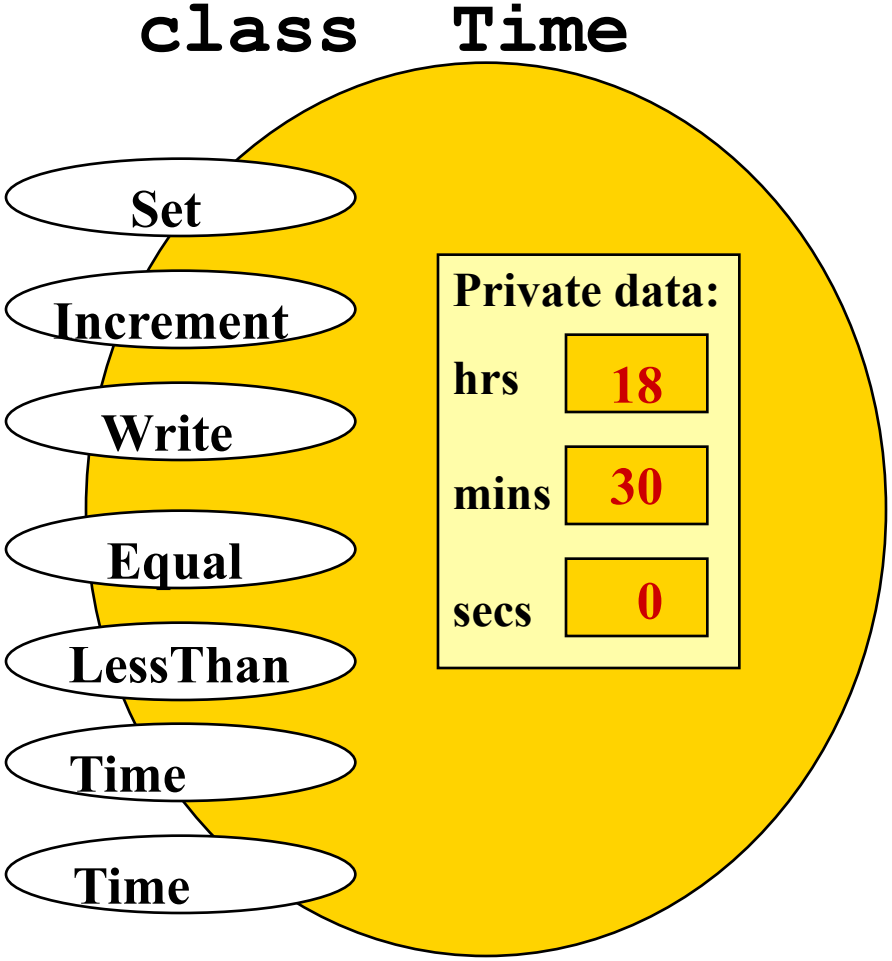
Specification of `Time`

```
class  Time          // "Time.h"
{
public : //  7 function members
     void Set (int hours, int  minutes, int
   seconds);
     void Increment ();
     void Write ()  const;
     bool  Equal (Time   otherTime)  const;
     bool  LessThan (Time  otherTime)  const;
```

Specification of `Time`

```
Time (int initHrs, int initMins, int initSecs);
     // Constructor
     Time ();
     // Default constructor

private :  // Three data members
   int  hrs;
   int  mins;
   int  secs;
};
```

`Time` Class Instance Diagram



Array of Class Objects

```
const   int   MAX_SIZE = 50;

// Declare array of class objects

 Time   trainSchedule[MAX_SIZE];
```

The default constructor, if there is any constructor, is invoked for each element of the array

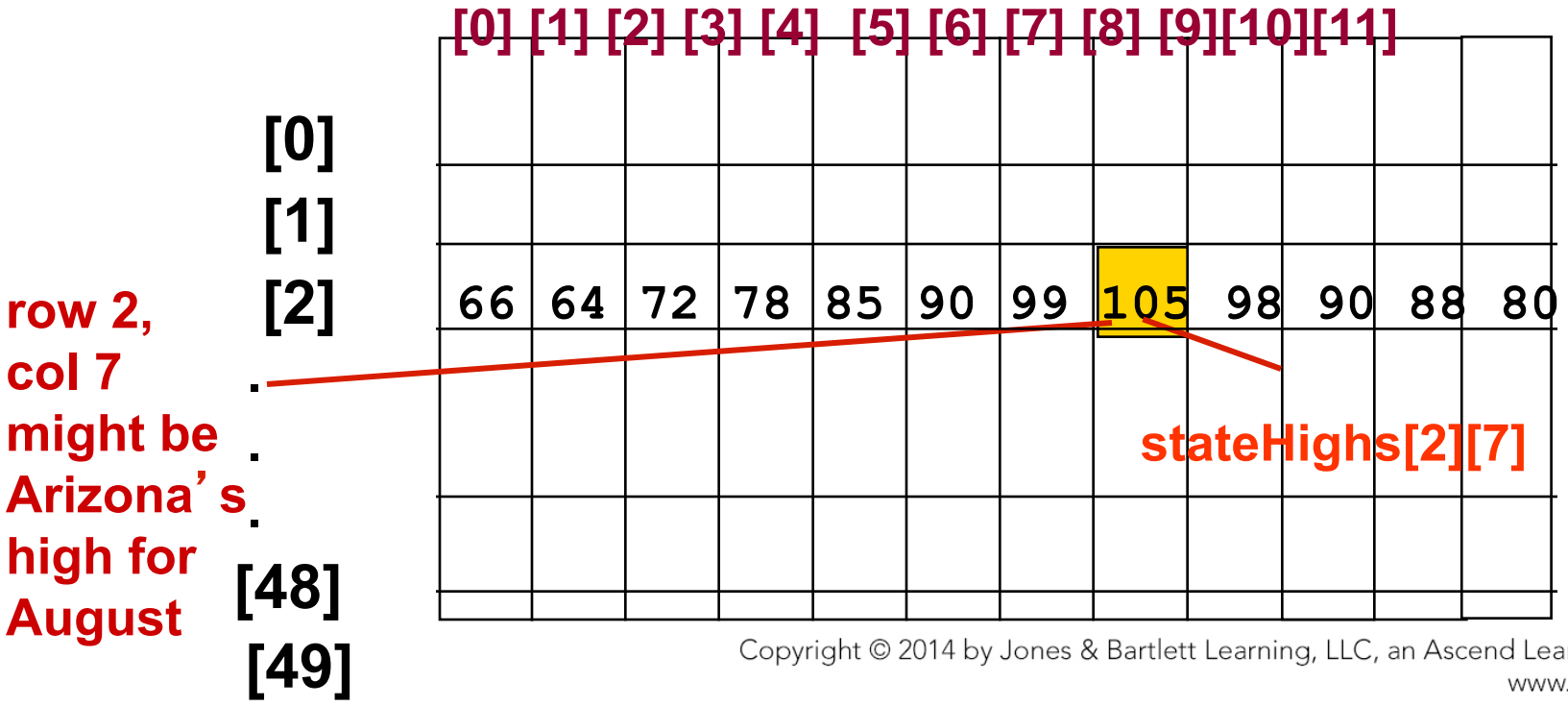
Two-Dimensional Array

- A two-dimensional array is a collection of components, all of the same type, structured in two dimensions, (referred to as rows and columns)
- Individual components are accessed by a pair of indexes representing the component's position in each dimension

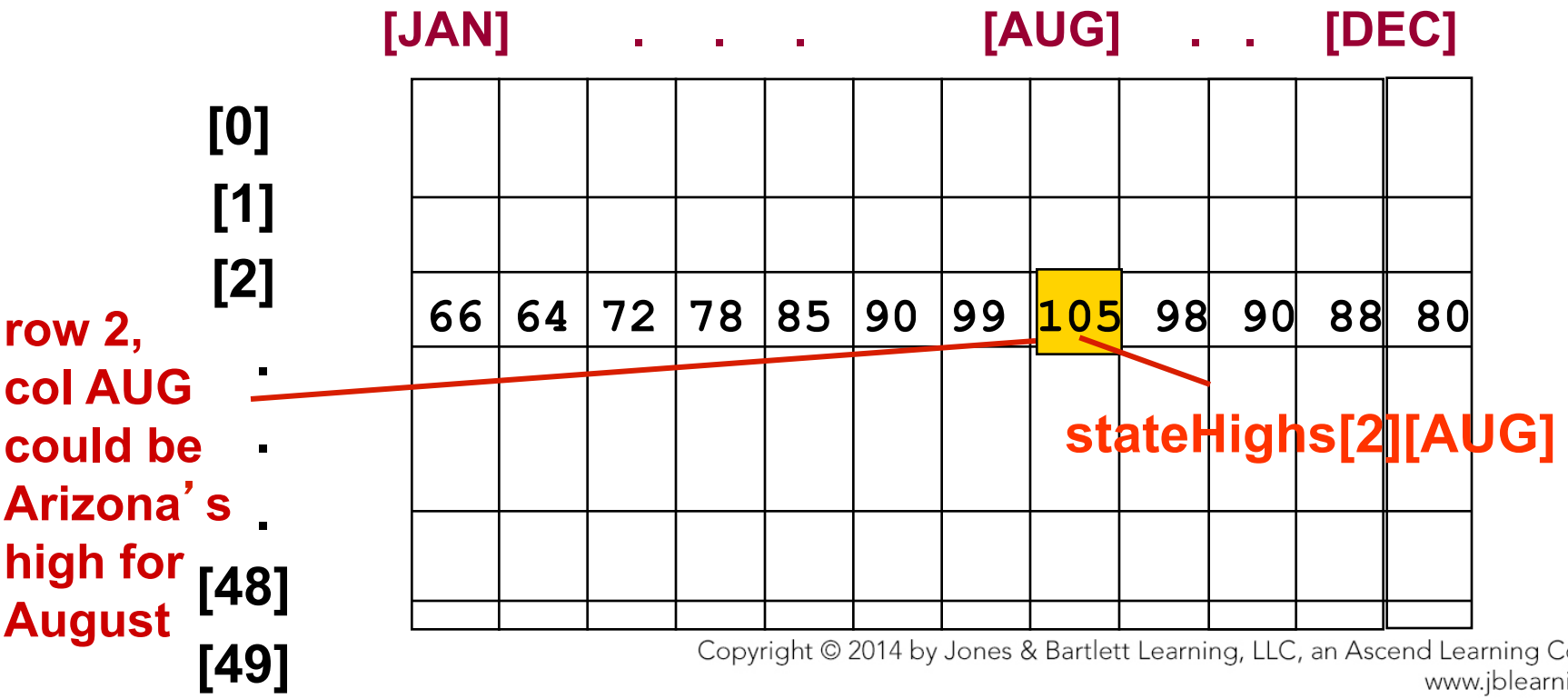
```
DataType ArrayName[ConstIntExpr][ConstIntExpr]...;
```

EXAMPLE -- Array for monthly high temperatures for all 50 states

```
const   int   NUM_STATES     =   50;
 const   int   NUM_MONTHS   =   12;
 int   stateHighs[NUM_STATES][NUM_MONTHS];
```



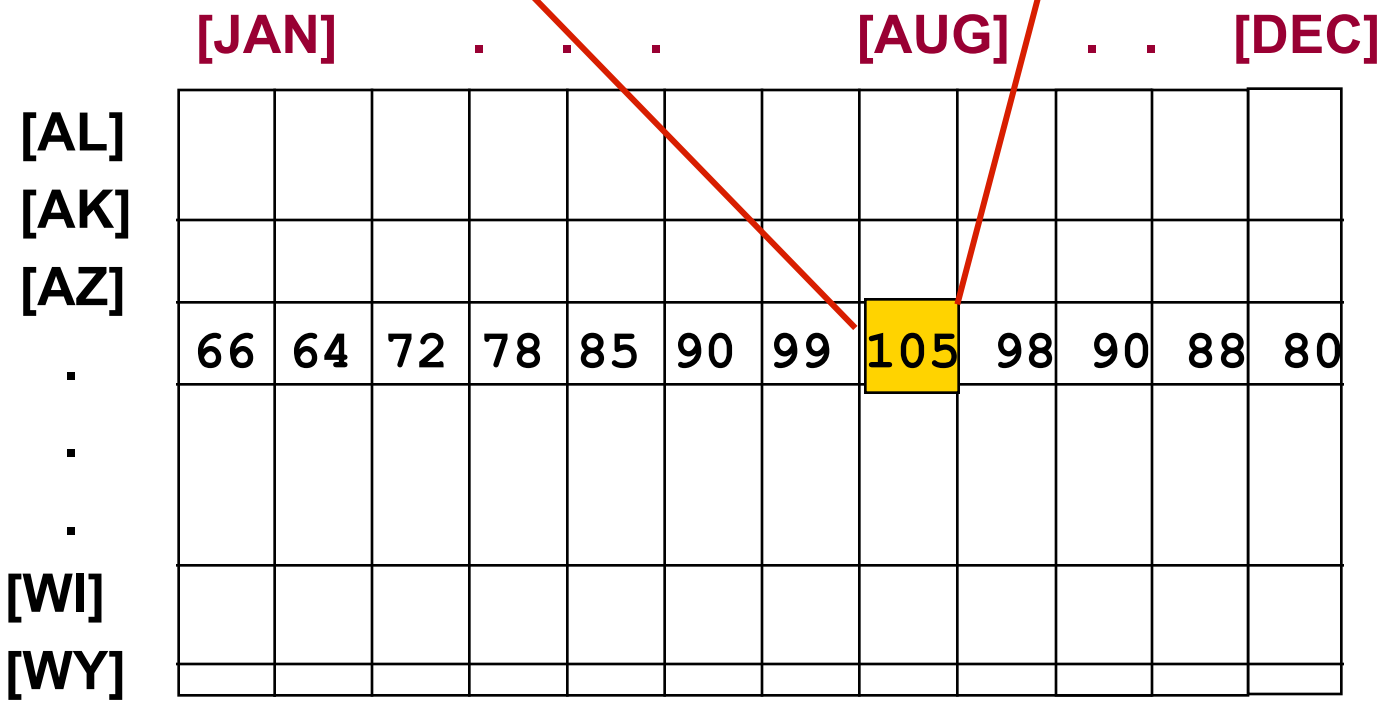
```
enum  Month { JAN, FEB, MAR, APR, MAY, JUN,
              JUL, AUG, SEP, OCT, NOV, DEC  };
const  int  NUM_MONTHS = 12;
const  int  NUM_STATES = 50;
int  stateHighs[NUM_STATES][NUM_MONTHS];
```



Array for Monthly High Temperatures for all 50 states, cont...

```
enum State  {  AL, AK, AZ, AR, CA, CO, CT, DE,
  FL, GA, HI, ID, IL, IN, IA, KS, KY, LA, ME,
  MD, MA, MI, MN, MS, MO, MT, NE, NV, NH, NJ,
  NM, NY, NC, ND, OH, OK, OR, PA, RI, SC, SD,
  TN, TX, UT, VT, VA, WA, WV, WI, WY };

enum  Month {  JAN, FEB, MAR, APR, MAY, JUN,
  JUL,
               AUG, SEP, OCT, NOV, DEC  };
const  int  NUM_MONTHS  =  12;
const  int  NUM_STATES    =  50;
int  stateHighs[NUM_STATES][NUM_MONTHS];
```

row AZ, col. AUG holds stateHighs[AZ][AUG]
Arizona's high for August



Finding the Average High Temperature for Arizona

```
int  total  = 0;
int  month;              // Without  enum  types
int  average;
for (month = 0; month < NUM_MONTHS; month ++)
    total = total + stateHighs[2][month];
average  = int (total / 12.0  + 0.5);
```

average | 85

Finding the Average High Temperature for Arizona, cont...

```
int                    total  = 0;
Month month;  // With  enum  types defined
int average;
for (month = JAN; month <= DEC; month = Month(month
  +1))
      total = total + stateHighs[AZ][month];
average  = int (total / 12.0  + 0.5);
```

average	85

```
const   int   NUM_STATES     =  50;
const   int   NUM_MONTHS   =  12;
int   stateHighs[NUM_STATES][NUM_MONTHS];
```

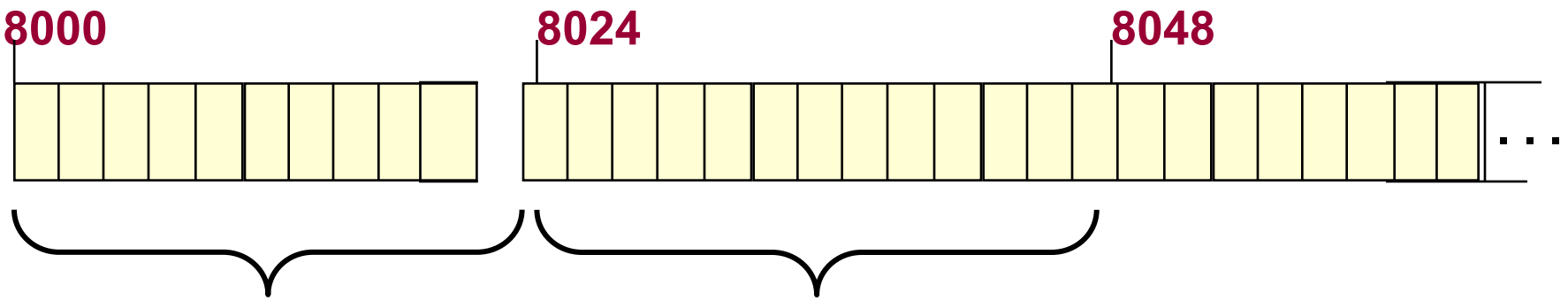
rows

columns

STORAGE

- In memory, C++ stores arrays in row order; the first row is followed by the second row, etc.

Base Address

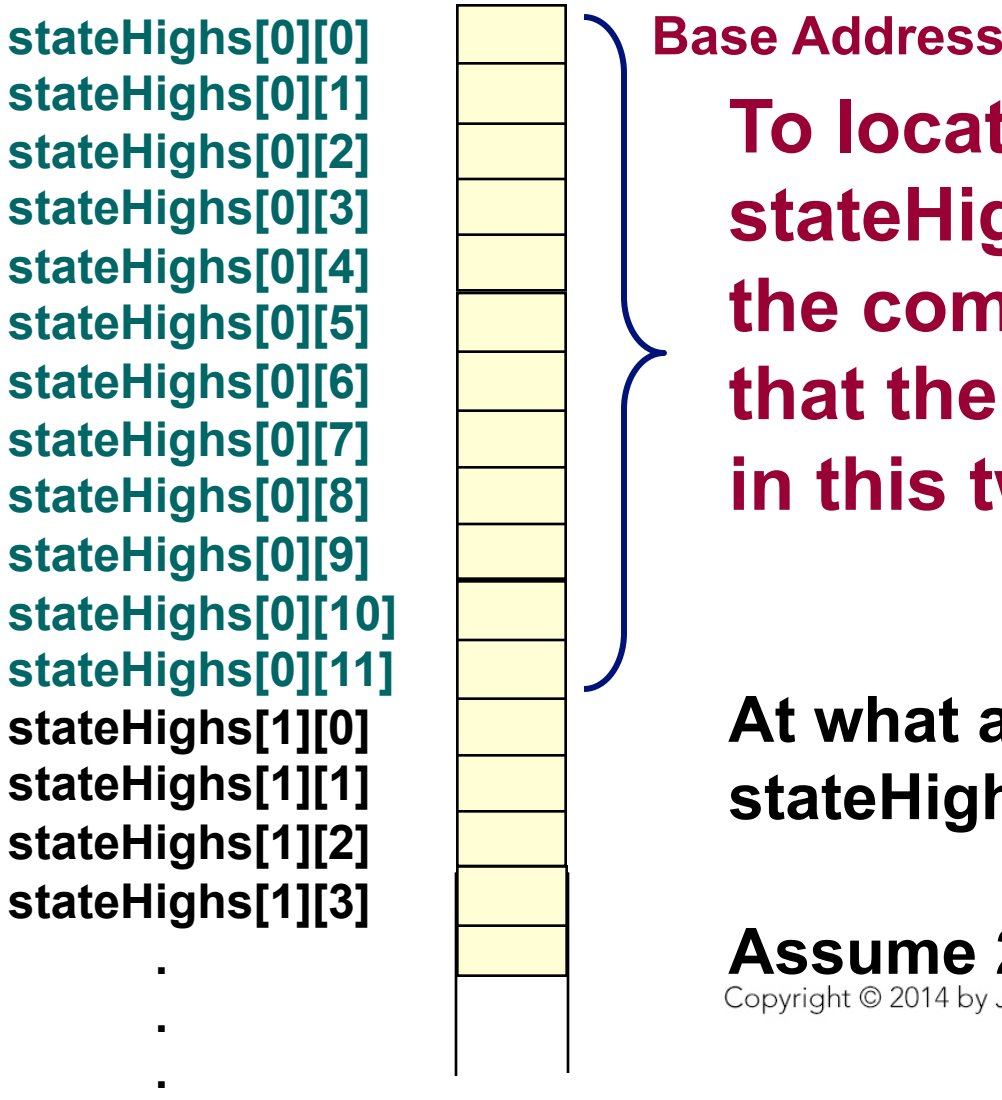


12 highs for state 0
Alabama
first row

12 highs for state 1
Alaska
second row

etc.

Viewed another way . . .



Base Address 8000

To locate an element such as stateHighs[2][7] the compiler needs to know that there are 12 columns in this two-dimensional array.

At what address will stateHighs[2][7] be found?

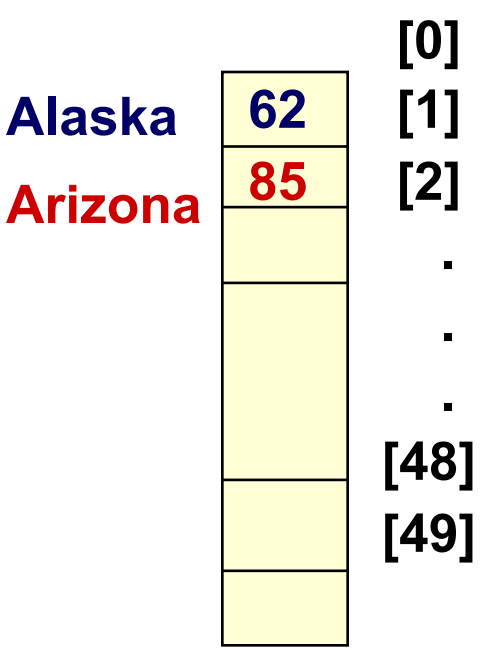
Assume 2 bytes for type int.

Arrays as Parameters

- As with a one-dimensional array, when a two- (or higher) dimensional array is passed as an argument, the base address of the caller's array is sent to the function
- The size of all dimensions except the first must be included in the function heading & prototype
- The sizes of those dimensions in the function's parameter list must be exactly the same as those declared for the caller's array

Write a function using the two-dimensional stateHighs array to fill a one-dimensional stateAverages array

```
const   int   NUM_STATES   =   50;
const   int   NUM_MONTHS   =   12;
int   stateHighs[NUM_STATES][NUM_MONTHS];
int   stateAverages[NUM_STATES];
```



[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
43	42	50	55	60	78	80	85	81	72	63	40
66	64	72	78	85	90	99	105	98	90	88	80

```
void FindAverages(
  /* in */ const int stateHighs[][NUM_MONTHS],
  /* out */       int stateAverages[])
//PRE:stateHighs[0..NUM_STATES][0..NUM_MONTHS]
//    assigned
// POST:stateAverages[0..NUM_STATES] contains
//      rounded high temperature for each state
```

```
{
     int  state;
     int  month;
     int  total;
     for  (state = 0;  state  <  NUM_STATES;  state++)
      {
         total = 0;
         for (month = 0; month < NUM_MONTHS; month++)
             total += stateHighs[state][month];
         stateAverages[state] = int(total/12.0 + 0.5);
      }
}
```

Using typedef with Arrays

The typedef statement helps eliminate the chances of size mismatches between function arguments and parameters. FOR EXAMPLE,

```
typedef int StateHighs [NUM_STATES][NUM_MONTHS];

typedef int StateAverages [NUM_STATES];

void FindAverages(
  /* in */ const StateHighs stateHighs,
  /* out */       StateAverages stateAverages)
{
}
```

Declaring Multidimensional Arrays

Example of three-dimensional array

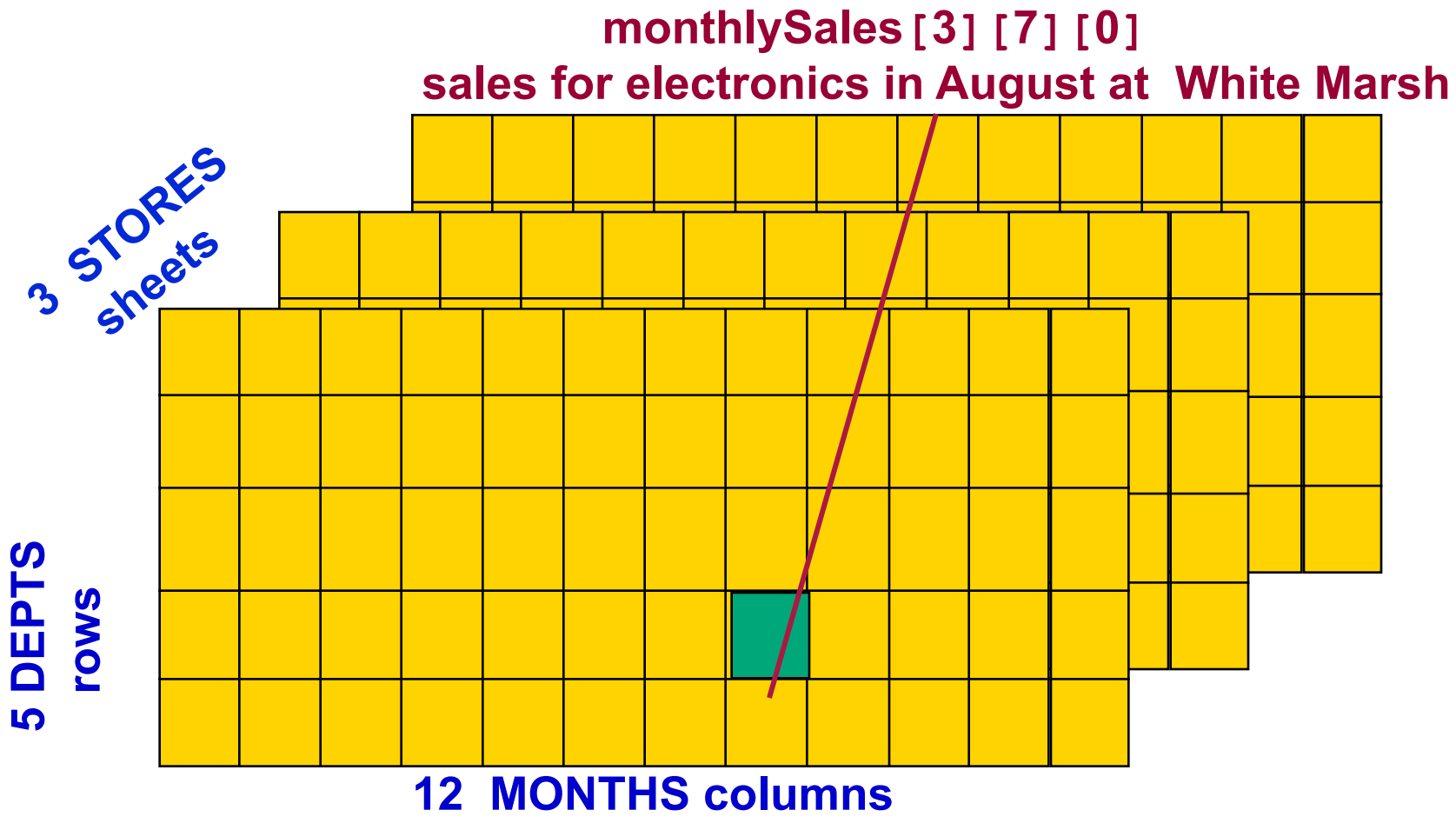
```
const NUM_DEPTS = 5;
// mens, womens, childrens, electronics, furniture
const NUM_MONTHS = 12;
const NUM_STORES = 3; // White Marsh,  Owings Mills, Towson

int  monthlySales[NUM_DEPTS][NUM_MONTHS][NUM_STORES];
```

rows columns sheets

OR USING TYPEDEF

```
typedef  int  MonthlySales [NUM_DEPTS][NUM_MONTHS][NUM_STORES];

MonthlySales  monthlySales;
```

```
const   NUM_DEPTS   = 5;
// mens, womens, childrens, electronics, furniture
const   NUM_MONTHS   = 12;
const   NUM_STORES   = 3; // White Marsh,  Owings Mills, Towson
int  monthlySales[NUM_DEPTS][NUM_MONTHS][NUM_STORES];
```



Print Sales for Dec. by Department

COMBINED DEPT #	SALES FOR DEPT NAME	December SALES $
0	Mens	12345
1	Womens	13200
2	Childrens	11176
3	Electronics	22567
4	Furniture	11230

Print sales for Jan. by department

COMBINED SALES FOR		January
DEPT #	DEPT NAME	SALES $
0	Mens	8345
1	Womens	9298
2	Childrens	7645
3	Electronics	14567
4	Furniture	21016

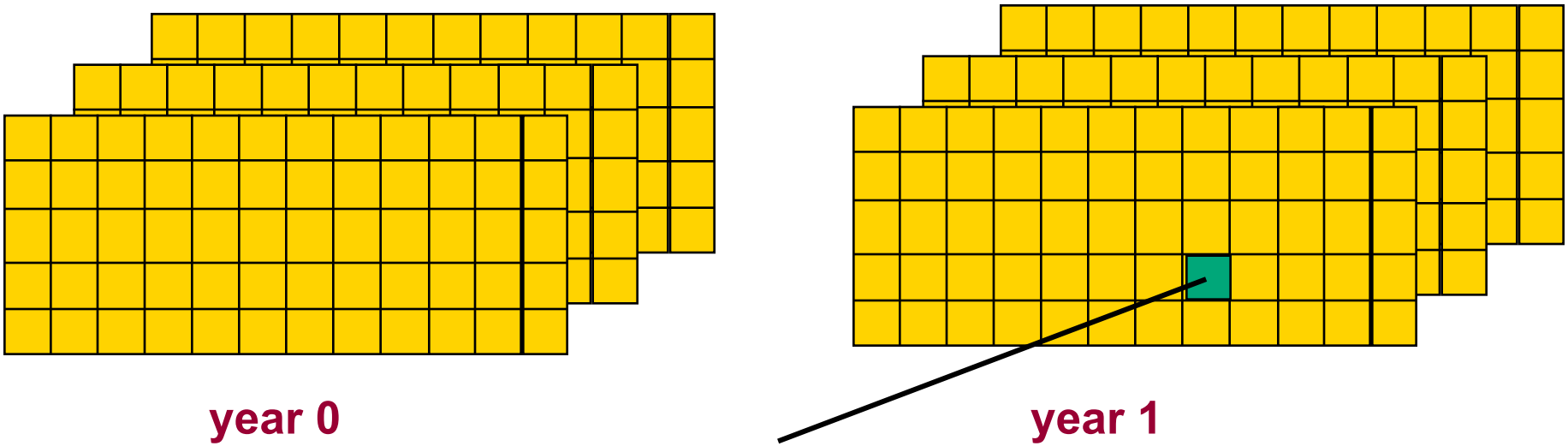
```
// mens, womens, childrens, electronics,
// furniture
const  NUM_DEPTS  = 5;
const  NUM_MONTHS  = 12;
// White Marsh,  Owings Mills, Towson
const  NUM_STORES  = 3;
int  monthlySales[NUM_DEPTS][NUM_MONTHS][NUM_STORES];
       .  .  .  .
```

```
for  (month = 0; month < NUM_MONTHS;  month++)
{
    cout  <<  "COMBINED  SALES  FOR  ";
    // Function call to write the name of month
    WriteOut(month);
    cout  << "DEPT # DEPT NAME SALES $" << endl;
```

```
  for  (dept = 0; dept < NUM_DEPTS; dept++)
  {
    totalSales = 0;
    for  (store = 0; store < NUM_STORES; store++)
      totalSales = totalSales +
                   monthlySales[dept][month][store];

      WriteDeptNameAndSales(dept, totalSales);
  }
}
```

Adding a Fourth Dimension . . .

```
const NUM_DEPT = 5;    // mens, womens, childrens …
const NUM_MONTHS = 12;
const NUM_STORES = 3; // White Marsh,  Owings Mills, Towson
const NUM_YEARS  = 2;
int  moreSales[NUM_DEPTS][NUM_MONTHS][NUM_STORES][NUM_YEARS];
```



for electronics, August, White Marsh, one year after starting year

C-Style Strings

- We have already been introduced to the C++ **string** data type.
- Because C++ is a superset of C it inherited C's primitive mechanism for representing strings.

Strings as Arrays

- C represents strings as arrays of char:

```
char mystring[4];
mystring[0] = 'd';
mystring[1] = 'o';
mystring[2] = 'g';
mystring[3] = 's';
```

C-String Literal Initialization

- A character array can also be initialized with a string literal:

```
char mystring[] = “dogs”;
```

- The compiler will automatically create an array of the proper length and generate the assignments we saw on the previous slide.

C-String Literal Initialization

- A character array can also be initialized with a string literal:

```
char mystring[] = "dogs";
```

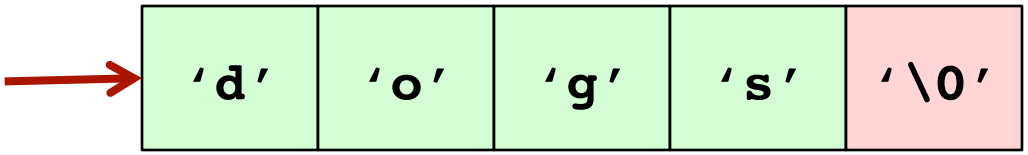
- The compiler will automatically create an array of the proper length and generate the assignments we saw on the previous slide.

However, the resulting array contents are not exactly the same…

C-String Initialization Differences

```
char mystring[4];
mystring[0] = 'd';
mystring[1] = 'o';
mystring[2] = 'g';
mystring[3] = 's';
```

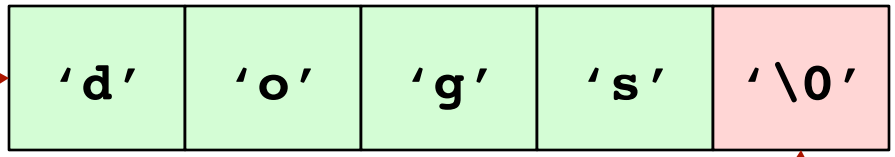
This construction creates an array of characters as you would expect.



C-String Initialization Differences

```
char mystring[4];
mystring[0] = 'd';
mystring[1] = 'o';
mystring[2] = 'g';
mystring[3] = 's';
```

This construction creates an array of characters as you would expect.



The string literal initialization automatically adds the *null* character '\0' to the end of the array.

```
char mystring[] = "dogs";
```

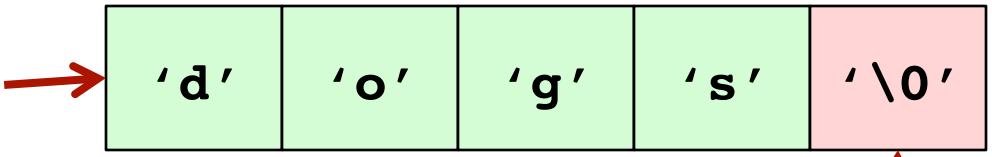
This array has length 5.

C-String Initialization Differences

We can achieve the same effect manually

```
char mystring[5];
mystring[0] = 'd';
mystring[1] = 'o';
mystring[2] = 'g';
mystring[3] = 's';
mystring[4] = '\0';
```

This construction creates an array of characters as you would expect.



The string literal initialization automatically adds the *null* character '\0' to the end of the array.

```
char mystring[] = "dogs";
```

This array has length 5.

Pointers to C-Strings

- Because an array variable is just a pointer to the first element of an array, we can declare a string as a pointer to char:

```
char *mystring = "dogs";
```

These are the same!

```
char mystring[] = "dogs";
```

String Termination & Length

- The '\0' sentinel character is used to indicate the end of a C-style string.
- We can use this to determine the length:

```
int length(char* str) {
  int len = 0;
  char* ch = str;
  while (*ch != '\0') {
    len++;
  }
  return len;
}
```

String Termination & Length

- The '\0' sentinel character is used to indicate the end of a C-style string.
- We can use this to determine the length:

```
int length(char* str) {
  int len = 0;
  char* ch = str;
  while (*ch != '\0') {
    len++;
  }
  return len;
}
```

Loop until you reach the '\0' character (end of string)

C-Strings Mutability

- C-Strings are **mutable**.
  - The contents of a string can be modified.
  - Assigning a character to any location in the C-string will overwrite the existing character with the one specified.

```
char ch[] = "hello";

ch[2] = 'Z';
```

This changes the first 'l' character in "hello" to a 'Z'.

Useful C-String Functions

- Include the `string.h` header file.

Function	Description
`size_t strlen(const char *s);`	Computes the length of the string **s**.
`int strcmp(const char *s1, const char *s2);`	Lexicographically compare strings **s1** and **s2** and returns an integer greater than, equal to, or less than 0.
`char* strncat(char *s1, const char *s2, size_t n);`	Appends a copy of **n** characters from the string **s2** to the end of the string **s1**. The string **s1** must be sufficiently long enough to hold the result. Returns a pointer to the new string.
`char* strncpy(char *s1, const char *s2, size_t n);`	Copies **n** characters from the string **s2** to the string **s1**. Returns a pointer to the new string.
`char* strchr(const char *s, int c);`	Locates the first occurrence of the character **c** in the string **s**. Returns a pointer to the location of the character in the string.
`char* strstr(const char *s1, const char *s2);`	Locates the first occurrence of the string **s2** in the string **s1** and returns a pointer to the start of that string.

Converting to C++ Strings

- **Converting To C++ String**

```
char *cdog = "dogs";
string cppdog(cdog);
```

- **Converter From C++ String**

```
char *cdog = cppdog.c_str();
```