C++ Programming Essentials

Unit 3: Basic Abstract Data Types

CHAPTER 11: ARRAYS

DR. ERIC CHOU

IEEE SENIOR MEMBER

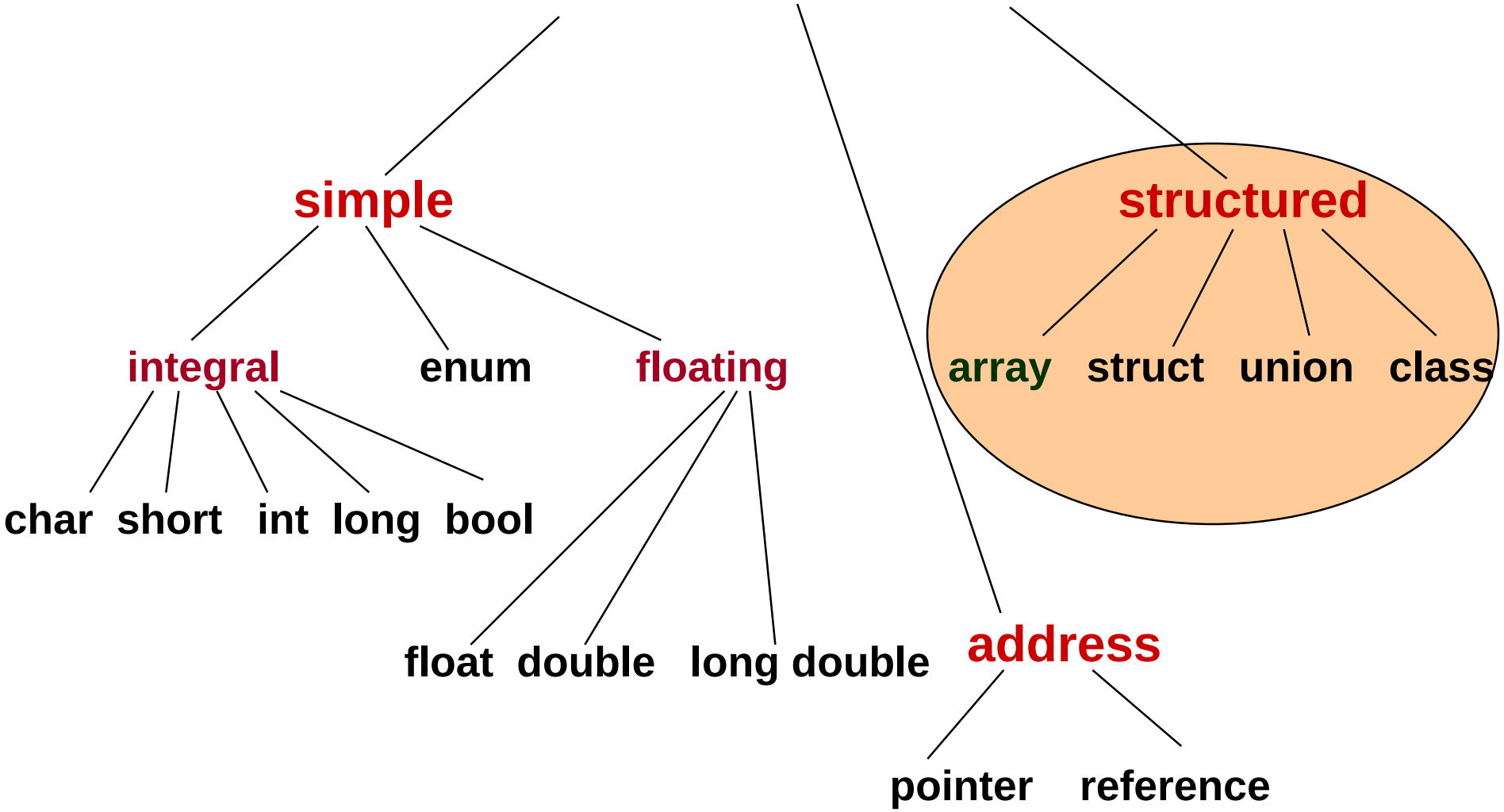
LECTURE 1

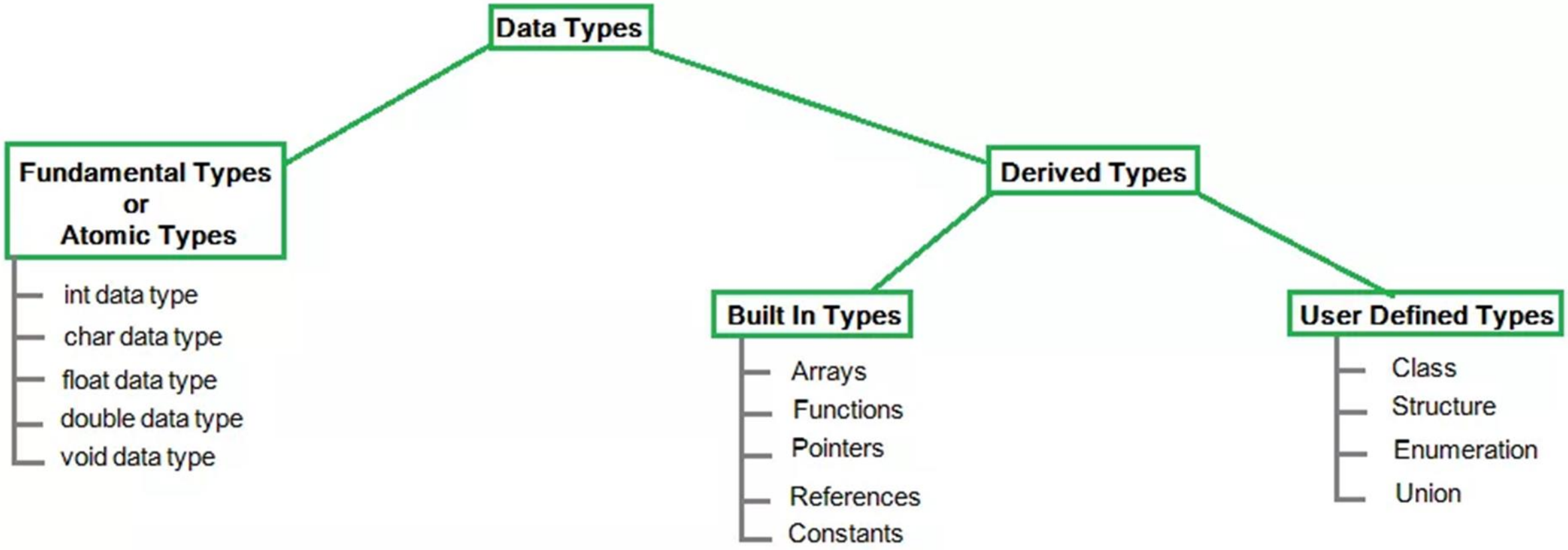
One-Dimensional Array

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
- 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


simple
integral
enum
floating
char
short
int
long
bool
float
double
long double
structured
array
struct
union
class
address
pointer
reference


Data Types
Fundamental Types or Atomic Types
int data type
char data type
float data type
double data type
void data type
Derived Types
Built In Types
Arrays
Functions
Pointers
References
Constants
User Defined Types
Class
Structure
Enumeration
Union

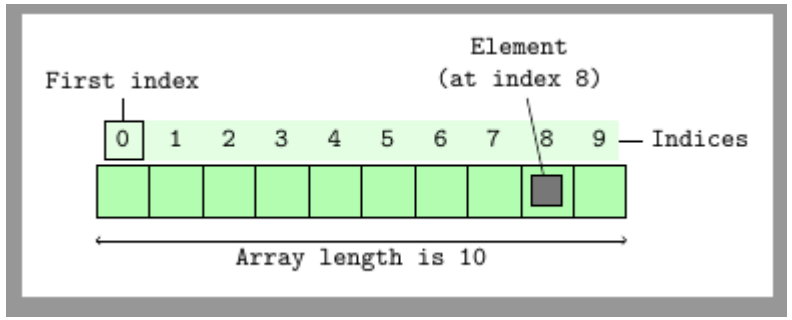
Structured Data Type

Composite Data Type

A structured data type is a type that

1. stores a collection of individual components with one variable name
2. and allows individual components to be stored and retrieved

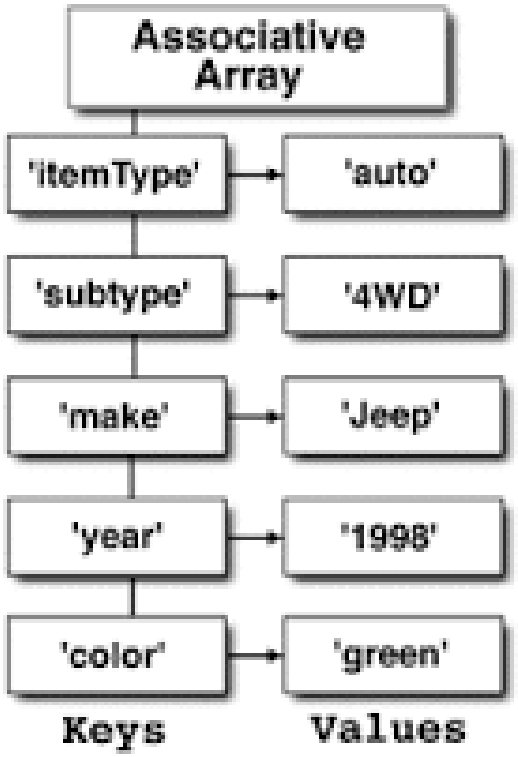
Arrays



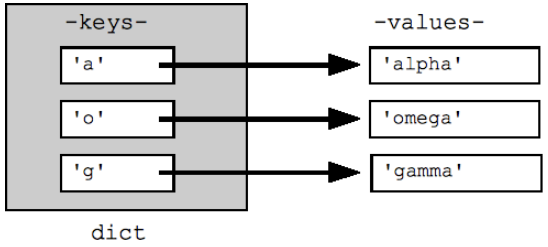
- Arrays are the most common and important composite data types
- Unlike records, which group related fields of disparate types, arrays are usually homogeneous
- Semantically, they can be thought of as a mapping from an **index type** to a **component** or **element type**
- A **slice or section** is a rectangular portion of an array.

Associative Arrays

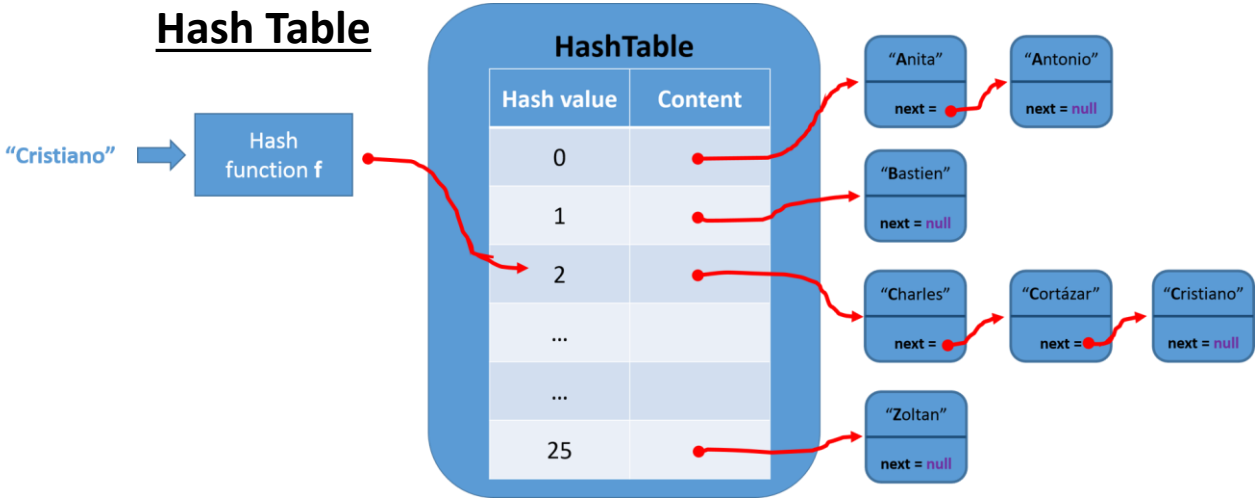
Associative Memory



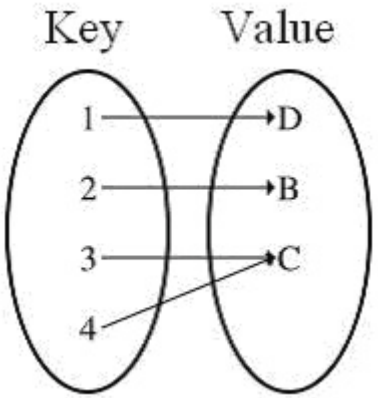
Python Dictionary



Hash Table



Java HashMap



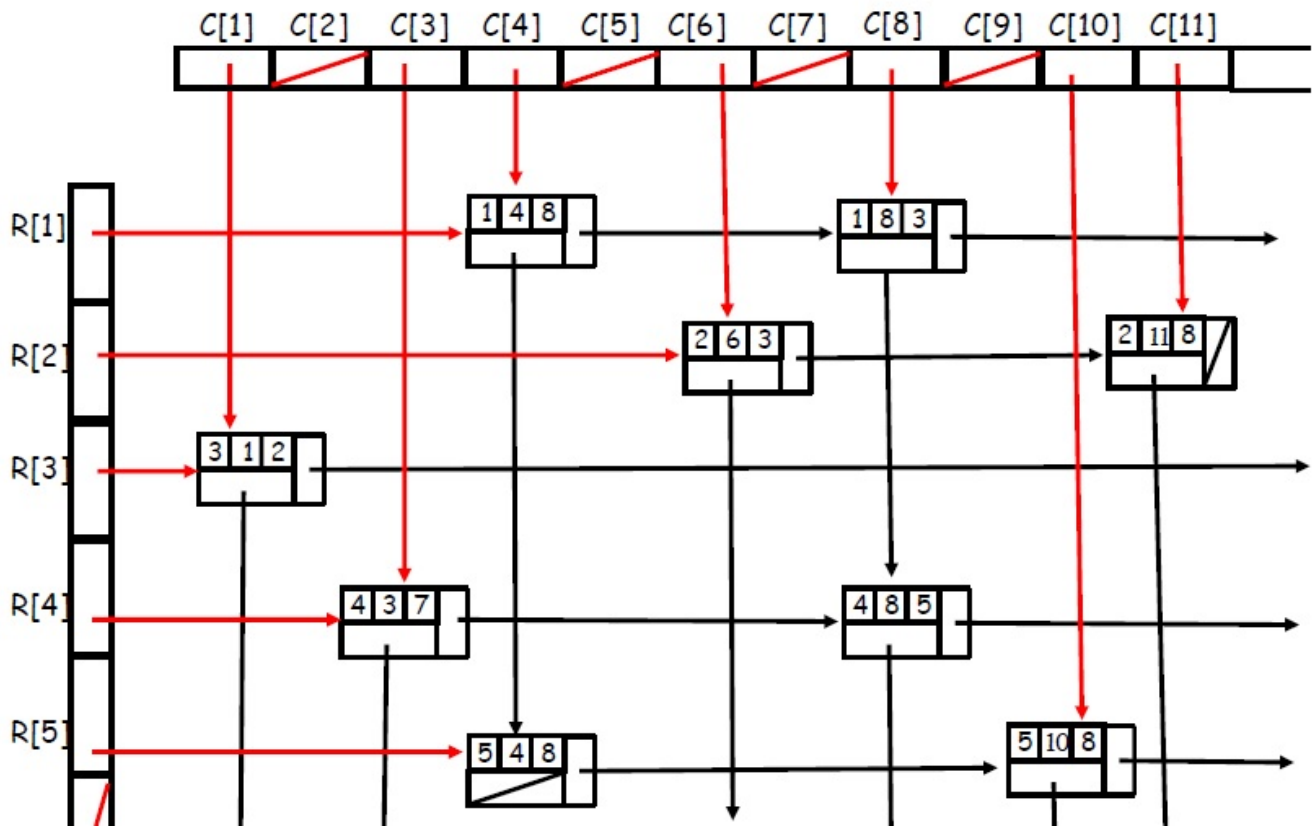
SQL Table (QUERY)

Instructor

Seminar

Student

Attribute	Length	Type	Rules
Name	40	Alpha	At least 2 words
Email Address	50	Mixed	Must contain @
Phone #	10	Numeric	Reject all "555"
Address	30	Mixed	Format - ### alpha
City	20	Alpha	none
State	2	Alpha	Must be a valid state

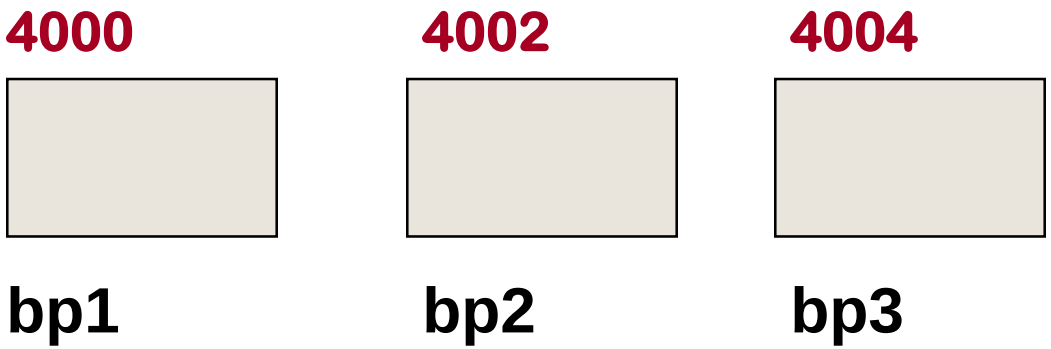


Sparse Matrix

C/C++ Constructed by Array and Nodes

Declare variables to store and total 3 blood pressures

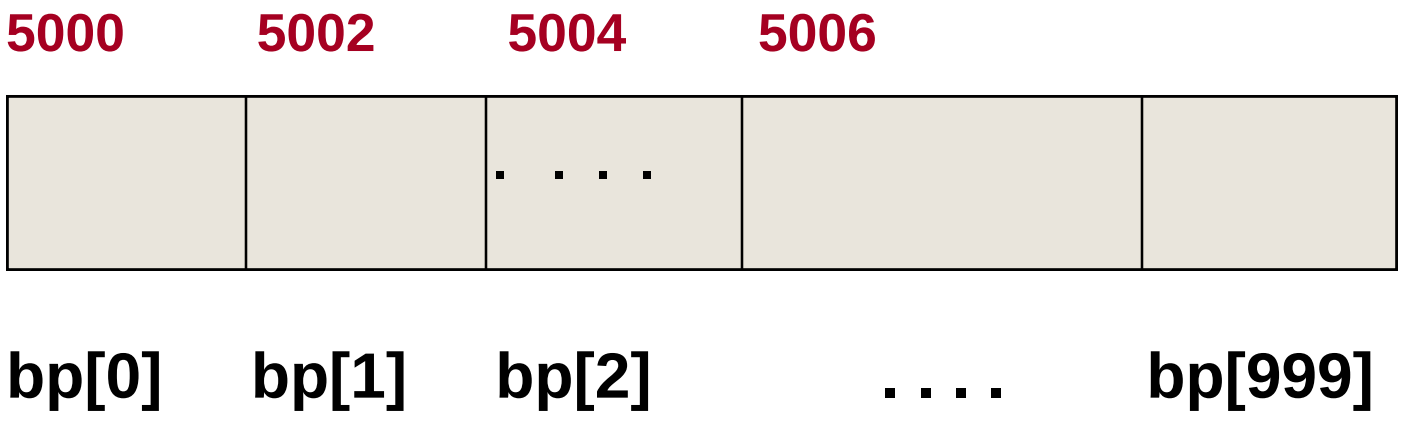
```
int bp1, bp2, bp3;
int total;
```




```
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
  - same data type
  - given a single name
- stored in adjacent memory locations.
- The individual components are accessed by using the array name together with an integral valued index in square brackets.

  Ex.) StudentID[5]
- 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.

number of elements in the array

```
float  temps[5];      // declaration allocates memory
```

Base Address

7000	7004	7008	7012	7016
temps[0]	temps[1]	temps[2]	temps[3]	temps[4]

indexes or subscripts

LECTURE 2

Array Declaration and Initialization

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.

number of elements in the array

```
char  name[10];        // declaration allocates memory
```

Base Address

6000	6001	6002	6003	6004	6005	6006	6007	6008	6009
name[0]	name[1]	name[2]	name[3]	name[4]		. . . . .			name[9]

Assigning Values to Individual Array Elements

```
float  temps[ 5 ] ;                    // allocates memory for array
int     m = 4 ;
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?
```

7000	7004	7008	7012	7016
99.4	?	98.6	101.2	50.6
temps[0]	temps[1]	temps[2]	temps[3]	temps[4]

What values are assigned?

```
float  temps[ 5 ] ;            // allocates memory for array
int     m ;

for (m = 0; m < 5; m++)        // will run 4 times
{
    temps[ m ] = 100.0 + m;
}
```

7000	7004	7008	7012	7016
?	?	?	?	?
temps[0]	temps[1]	temps[2]	temps[3]	temps[4]

Now what values are printed?

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

7000	7004	7008	7012	7016
100.0	101.0	102.0	103.0	104.0
temps[0]	temps[1]	temps[2]	temps[3]	temps[4]

Variable Subscripts

```
float  temps[ 5 ] ;          // allocates memory for array
int     m = 3 ;
 . . . . . .
```

What is temps[m + 1] ?

What is temps[m] + 1 ?

7000	7004	7008	7012	7016
100.0	101.0	102.0	103.0	104.0
temps[0]	temps[1]	temps[2]	temps[3]	temps[4]

A Closer Look at the Compiler

```
float  temps[5];   // this declaration allocates memory
```

To the compiler, the value of the identifier **`temps`** alone 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.

7000	7004	7008	7012	7016
100.0	101.0	102.0	103.0	104.0
temps[0]	temps[1]	temps[2]	temps[3]	temps[4]

Initializing in a Declaration

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


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

6000	6002	6004	6006	6008
40	13	20	19	36
ages[0]	ages[1]	ages[2]	ages[3]	ages[4]

LECTURE 3

Passing Arrays as Arguments

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) with any type of component elements 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 2 items of information as arguments:
- the beginning memory address of the array (base address)
- the number of elements to process in the array (you will want to send this as another parameter)

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 ;

int main (   )
{
    int    temp[31] ;   // array to hold up to 31 temperatures
    int    numDays ;
    int    average ;
    int    hottest ;
    // continued next page
```

Example with Array Parameters

```
Obtain(temp, numDays);
    Print(temp, numDays);
    cout << endl;

    FindAverage(temp, numDays, average);
      FindWarmest(temp, numDays, hottest);
      cout << "Average Temp=" << average
             << "  Hottest Temp=" << hottest
             << endl;
    return 0;
}
```

In, Out, InOut and const variables

void f(int **a**, int &**b**, int &**c**, const int **d**){ … }

- **a**: input variable (modifiable)
- **b**: in/out variable (modifiable), may be used as output variable
- **c**: in/out variable (modifiable), can be used as in/out variable (no difference from **b**)
- **d**: input constant variable, can never be modified.

- Note: C has no output variable. C's pointer is not very straight-forward.

LECTURE 4

Memory Allocation for Arrays

Memory Allocated for Array

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

Base Address

6000

50	65	70	62	68	. . . . .		
temp[0]	temp[1]	temp[2]	temp[3]	temp[4]	. . . . .		temp[30]

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

// Has user enter number temperature values 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 ]  have been printed 5 to a line
{
     int  m;
     cout  <<   "You entered: " ;
     for ( m = 0 ; m < number;  m++ )
      {
         if  ( m % 5 == 0 )
             cout  <<  endl ;
      cout  <<  setw(7) << temp [m] ;
      }
}
```

LECTURE 5

const Data Type (final)

Use of `const`

- because the identifier of an array holds the base address of the array, an & 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 heading and prototype

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

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

```
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 == arithmetic average of temp[0 . . number-1]
{
      int  m;
      int  total = 0;
      for ( m = 0 ; m < number;  m++ )
        {
           total = total + temp [m] ;
        }
      avg = int (float (total) / float (number) + .5) ;
} // avg = total/number
```

```
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]
{
     int  m;
     largest = temp[0] ;      // initialize largest to first element
                              // then compare with other elements
     for ( m = 0 ; m < number;  m++ )
      {
        if ( temp [m]  > largest )
            largest  =  temp[m] ;
      }
}
```

Demo Program

temp.cpp

Go Dev C++!!!

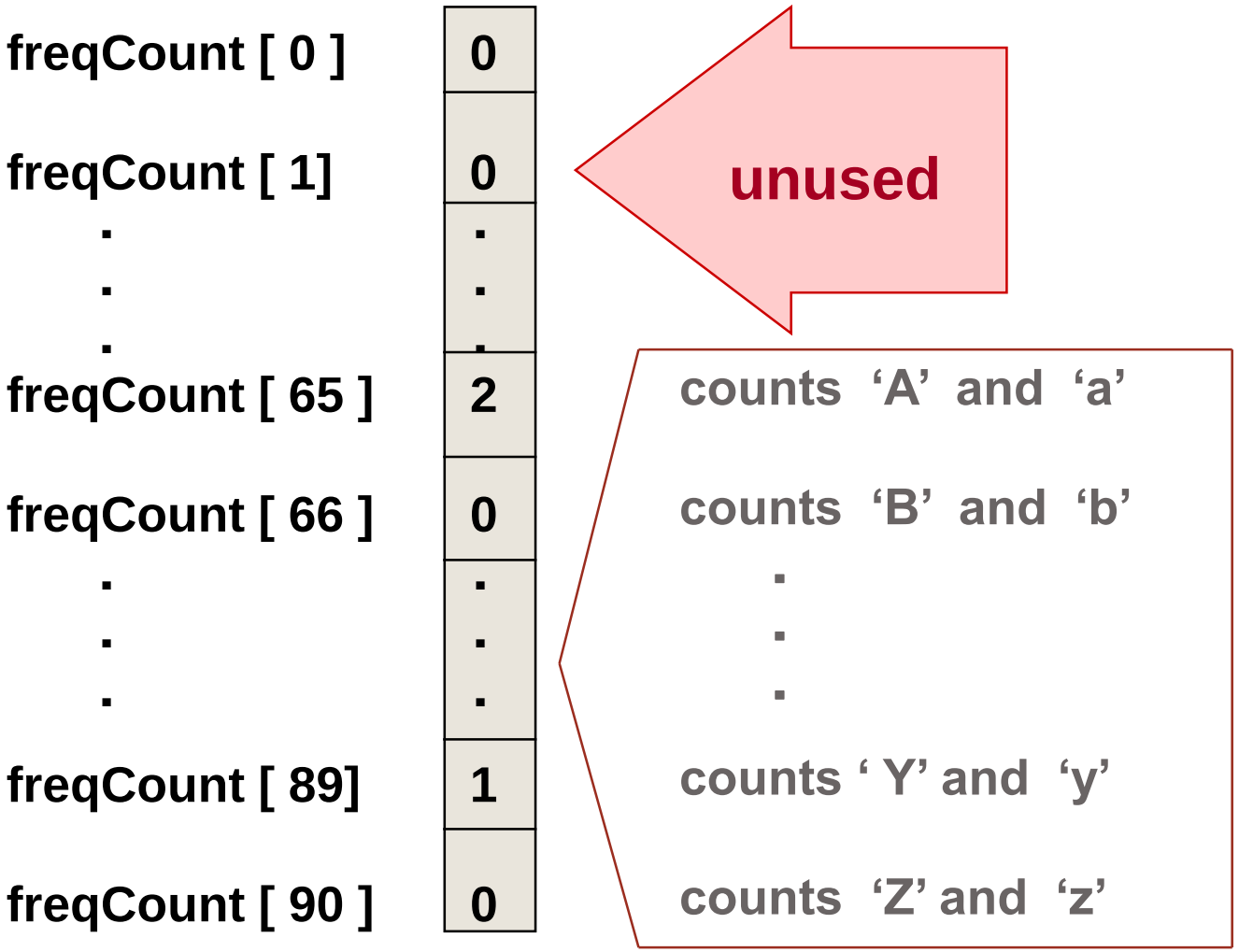
LECTURE 6

Use Array as Occurrence Counter

Using arrays for Counters

Write a program to count the number of each alphabet letter in a text file.

```
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

using  namespace  std ;
```

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

    dataFile.open ( "A:\\my.dat" ) ;            // open and verify success
    if  ( ! dataFile )
    {
          cout  <<  " CAN'T OPEN INPUT FILE ! " << endl;
          return  1;
    }

    for (  int  m = 0; m < SIZE;  m++ )          // zero out the array

          freqCount [ m ]  =  0 ;
```

```
                                        // read file one character at a time

    dataFile.get ( ch ) ;               // priming read
    while ( dataFile )                  // while last read was successful
    {
            if (isalpha ( ch ) )
            {
                    if ( islower ( ch ) )
                            ch  =  toupper ( ch ) ;

                    freqCount [ ch ]  =  freqCount [ ch ] + 1 ;
            }
            dataFile. get ( ch ) ;              // get next character
    }
    PrintOccurrences ( freqCount ) ;
    return  0;
}
```

```
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
{     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 ;
      }
}
```

Demo Program:

occurrence.cpp

Go Dev C++!!!

C:\Eric_Chou\Cpp Course\

```
A       484
B        95
C       188
D       261
E       876
F       184
G       131
H       352
I       457
J        17
K        14
L       231
M       146
N       494
O       522
P       140
Q         6
R       429
S       481
T       647
U       210
V        74
W        97
X         9
Y        82
Z         4
```

Letter count of US Declaration of Indenpendence

LECTURE 7

enum Variable as index

More about Array Index

- array index can be any integral type. This includes `char` and `enum` types
- it is programmer's responsibility to make sure that an array index does not go out of bounds. The index must be within the range 0 through the declared array size minus one
- using an index value outside this range causes the program to access memory locations outside the array. The index value determines which memory location is used

Array with enum Index Type

DECLARATION

```
enum Department { WOMENS,  MENS,  CHILDRENS,
                  LINENS, HOUSEWARES, ELECTRONICS };

float            salesAmt [ 6 ] ;

Department   which;   //  (enum is New!)
```

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])**	

LECTURE 8

Parallel Arrays

Parallel Arrays

DEFINITION

Parallel arrays are 2 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

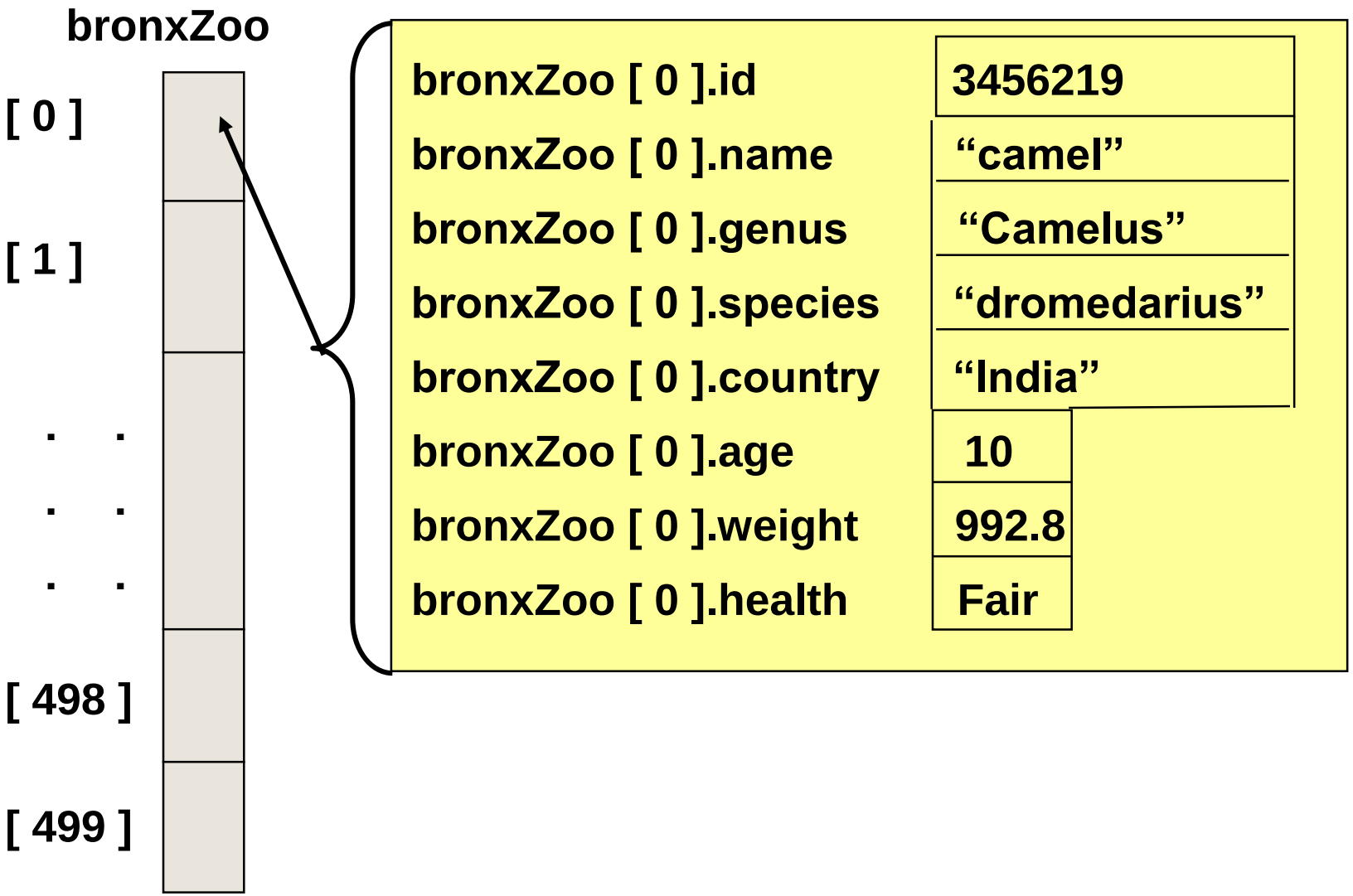
```
const  int  SIZE  50 ;
int      idNumber [ SIZE ] ;        // parallel arrays hold
float    hourlyWage [ SIZE ] ;      // related information
```

idNumber [0]	**4562**	**hourlyWage [0]**	**9.68**
idNumber [1]	**1235**	**hourlyWage [1]**	**45.75**
idNumber [2]	**6278**	**hourlyWage [2]**	**12.71**
. . .	. . .	. . .	. . .
idNumber [48]	**8754**	**hourlyWage [48]**	**67.96**
idNumber [49]	**2460**	**hourlyWage [49]**	**8.97**

Array of Structures

```
const  int  MAX_SIZE = 500 ;

enum  HealthType  { Poor,  Fair,  Good,  Excellent } ;

struct  AnimalType                                  // declares struct data type
{
 long          id ;
 string        name ;
 string        genus ;
 string        species ;
 string        country ;                      //  8 struct members
 int           age ;
 float         weight ;
 HealthType    health ;
} ;

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

```
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 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 ;
```

Parallel of Arrays or Array of struct (objects)

This largely depends on how you intend to implement the solution. If you want to take advantage of data parallel features of the CPU or GPU then you might well be better off implementing this as a struct of arrays than an array of structs.

- Array of struct may contain more memory holes.
- struct of arrays can have faster index advancing calculation

```
typedef struct {
  unsigned int* rowIdxs;
  unsigned int* colIdxs;
  unsigned int* dataValues;
} entity, *spMat;
```

Parallel of Arrays or Array of struct (objects)

- This will make it easier to write code that either the CPU compiler's vectorizor or the GPU's compiler can use efficiently. So in this case I would probably use an struct of arrays first and optimize for data parallel(ness).
- That being said it will largely depend on how good your implementation is. it would be possible to write a poorly performing implementation with either approach.

LECTURE 9

Array of Objects and 2-D Arrays

Array of Class Objects

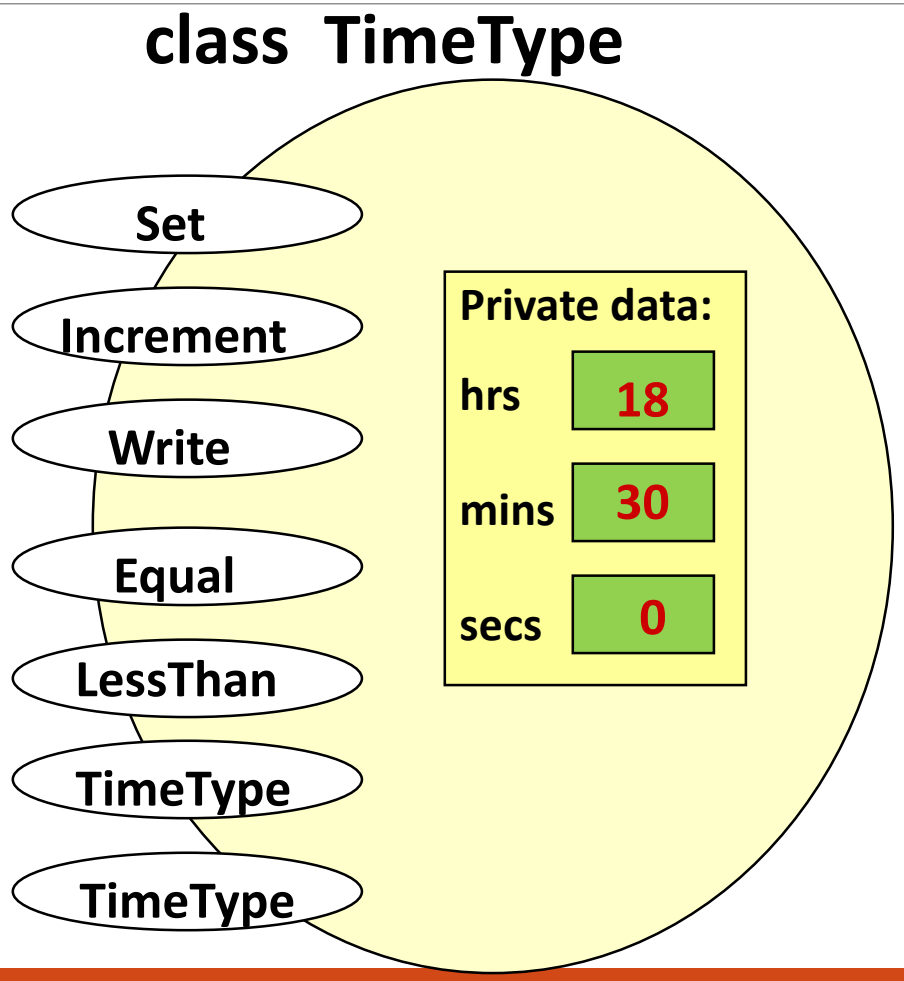
```
const  int  MAX_SIZE = 50 ;
                                        // declare array of class objects
TimeType   trainSchedule[ MAX_SIZE ] ;
```

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

Specification of `TimeType`

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

 TimeType ( int  initHrs ,  int  initMins ,  int  initSecs ) ;  // constructor
 TimeType ( ) ;                                                 // default constructor

private :                                                //  3 data members
 int          hrs ;
 int          mins ;
 int          secs ;
} ;
```

`TimeType` Class Instance Diagram



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.

SYNTAX FOR ARRAY DECLARATION

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

EXAMPLE -- To keep monthly high temperatures for all 50 states in one array.

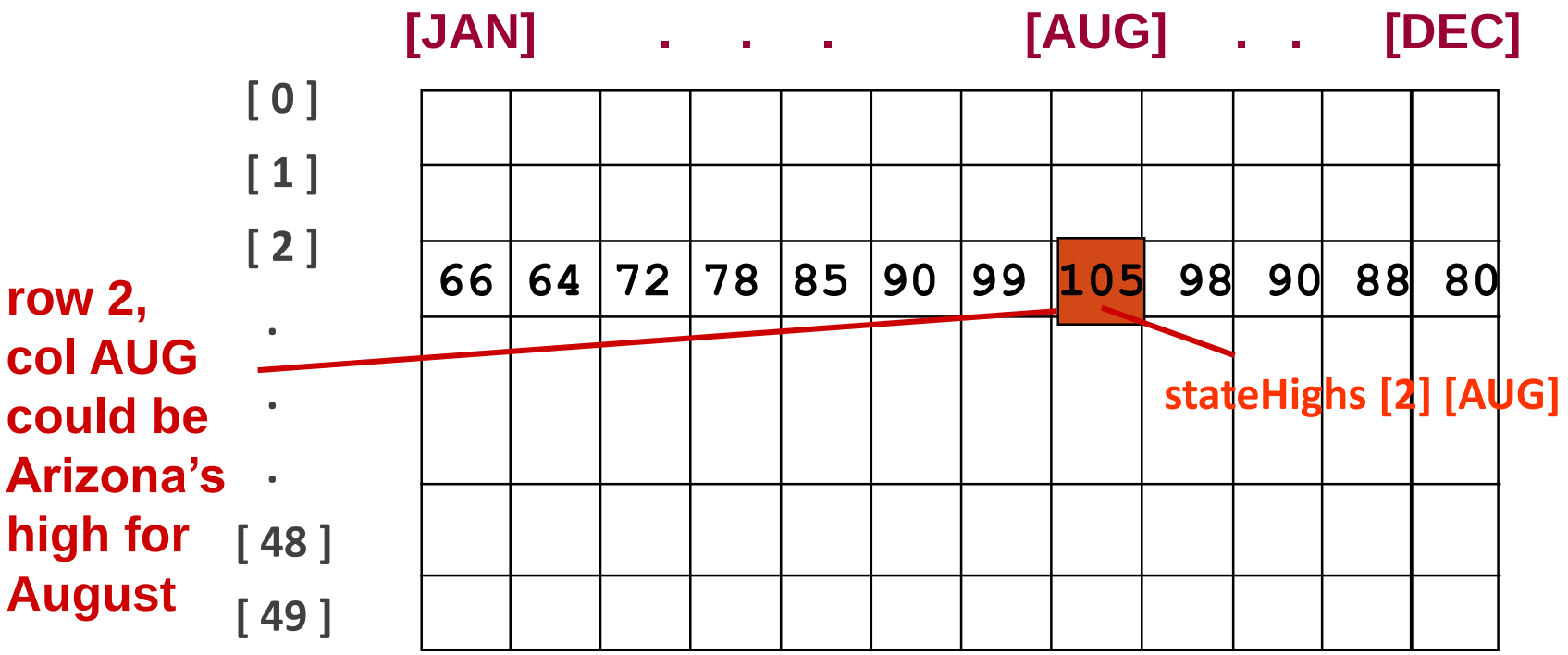
```
const int NUM_STATES   = 50 ;
const int NUM_MONTHS = 12 ;
int stateHighs [ NUM_STATES ] [ NUM_MONTHS ] ;
```

	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]	[11]
[0]												
[1]												
[2]	66	64	72	78	85	90	99	105	98	90	88	80
.												
.												
.												
[48]												
[49]												

row 2, col 7 might be Arizona's high for August

stateHighs [2] [7]

```
enum MonthType { 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 ] ;
```



```
enum StateType  {  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  MonthType   {  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 Arizona's high for August

	[JAN]	.	.	.				[AUG]	.	.	[DEC]
[AL]											
[AK]											
[AZ]	66	64	72	78	85	90	99	105	98	90	88	80
.											
.											
.											
[WI]											
[WY]											

stateHighs [AZ] [AUG]

Array of Arrays in C/C++, Java, C#

Figure: Showing jagged array.

```
int[][] jagArray = new int[5][];
```

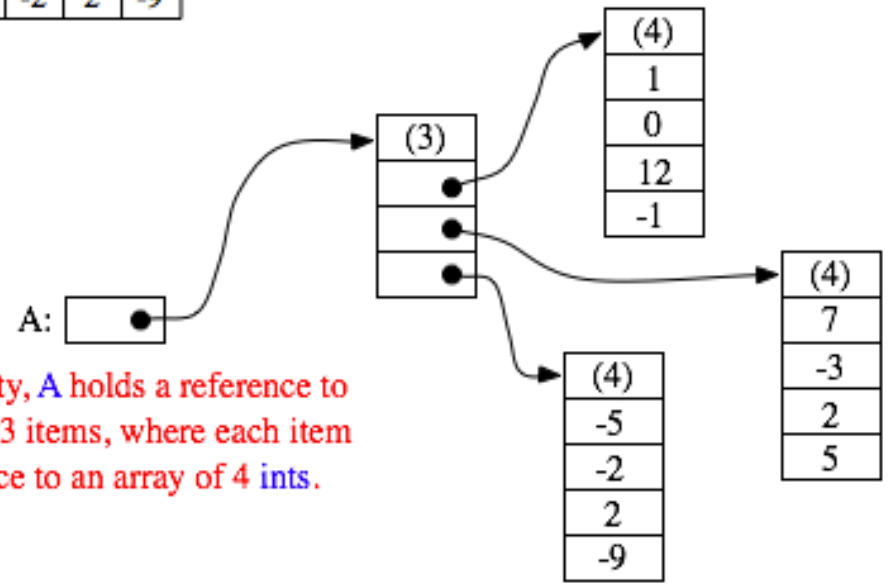
0	int[]
1	int[]
2	int[]
3	int[]
4	int[]

On each index of jagged array another array reference is stored.

If you create an array A = new int[3][4], you should think of it as a "matrix" with 3 rows and 4 columns.

A:

1	0	12	-1
7	-3	2	5
-5	-2	2	-9



But in reality, A holds a reference to an array of 3 items, where each item is a reference to an array of 4 ints.

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

```
int            total  = 0 ;
MonthType  month ;                    //   WITH  ENUM  TYPES DEFINED
int            average ;
for ( month = JAN ; month <= DEC ; month = MonthType( month + 1) )
     total = total + stateHighs [ AZ] [ month ] ;
average  = int ( total / 12.0  + 0.5 ) ;
```

average

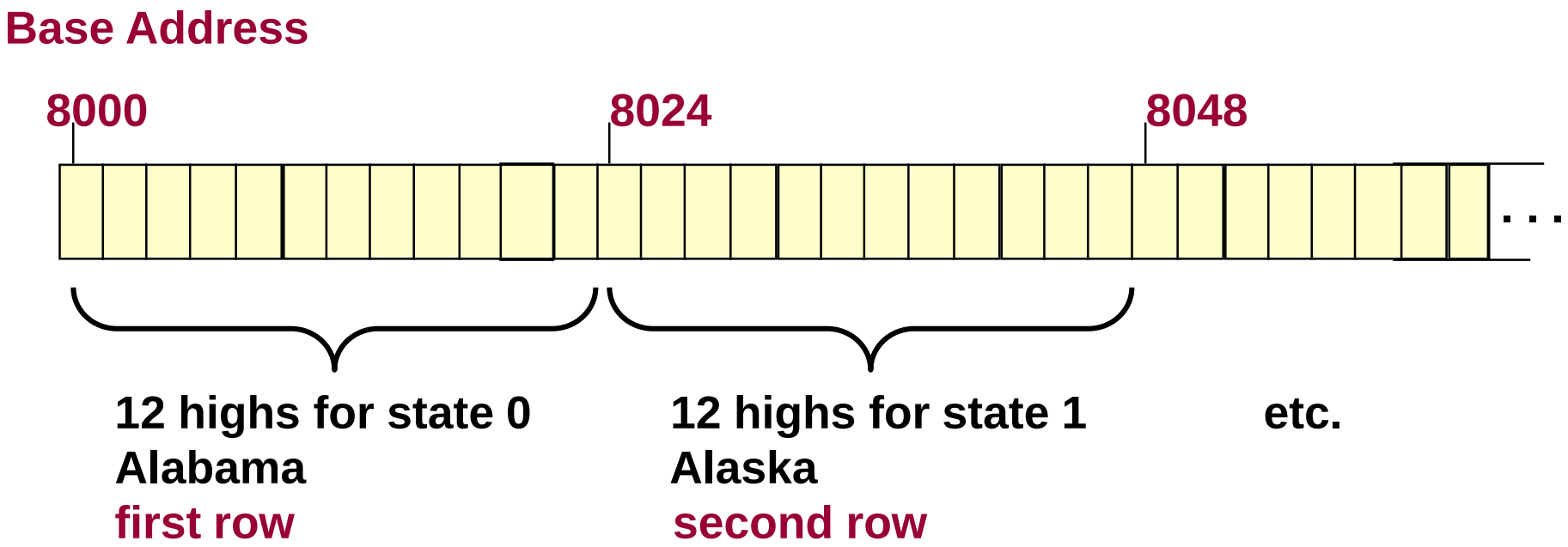
85

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

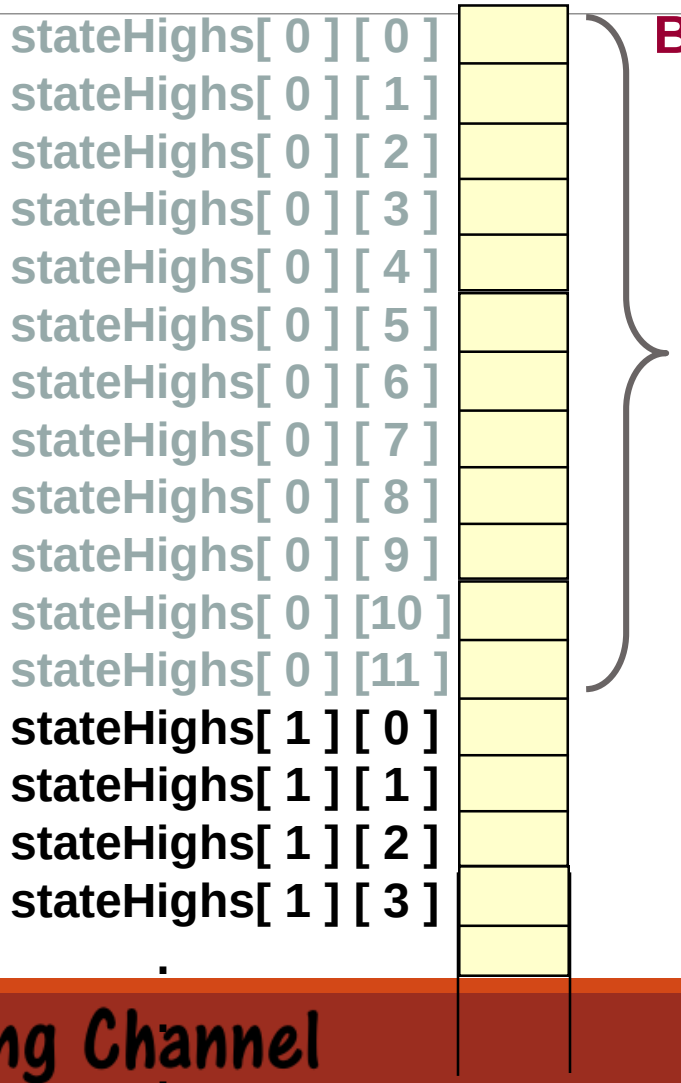
rows columns

STORAGE

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



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.

LECTURE 10

Array of Objects

Arrays as Parameters

- just 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 and prototype
- the sizes of those dimensions in the function's parameter list must be exactly the same as 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]
		[0]												
Alaska	62	[1]	43	42	50	55	60	78	80	85	81	72	63	40
Arizona	85	[2]	66	64	72	78	85	90	99	105	98	90	88	80
		.												
		.												
		.												
		[48]												
		[49]												

```
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 average

 //                                        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

- helps eliminate the chances of size mismatches between function arguments and parameters.
- **FOR EXAMPLE,**

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

typedef  int   StateAveragesType [ NUM_STATES ] ;

void  FindAverages( /* in */  const  StateHighsType       stateHighs ,
                    /* out */          StateAveragesType  stateAverages  ){

        .

        .

        .

}
```

LECTURE 11

Multi-Dimensional Arrays

Declaring Multidimensional Arrays

EXAMPLE OF THREE-DIMENSIONAL ARRAY

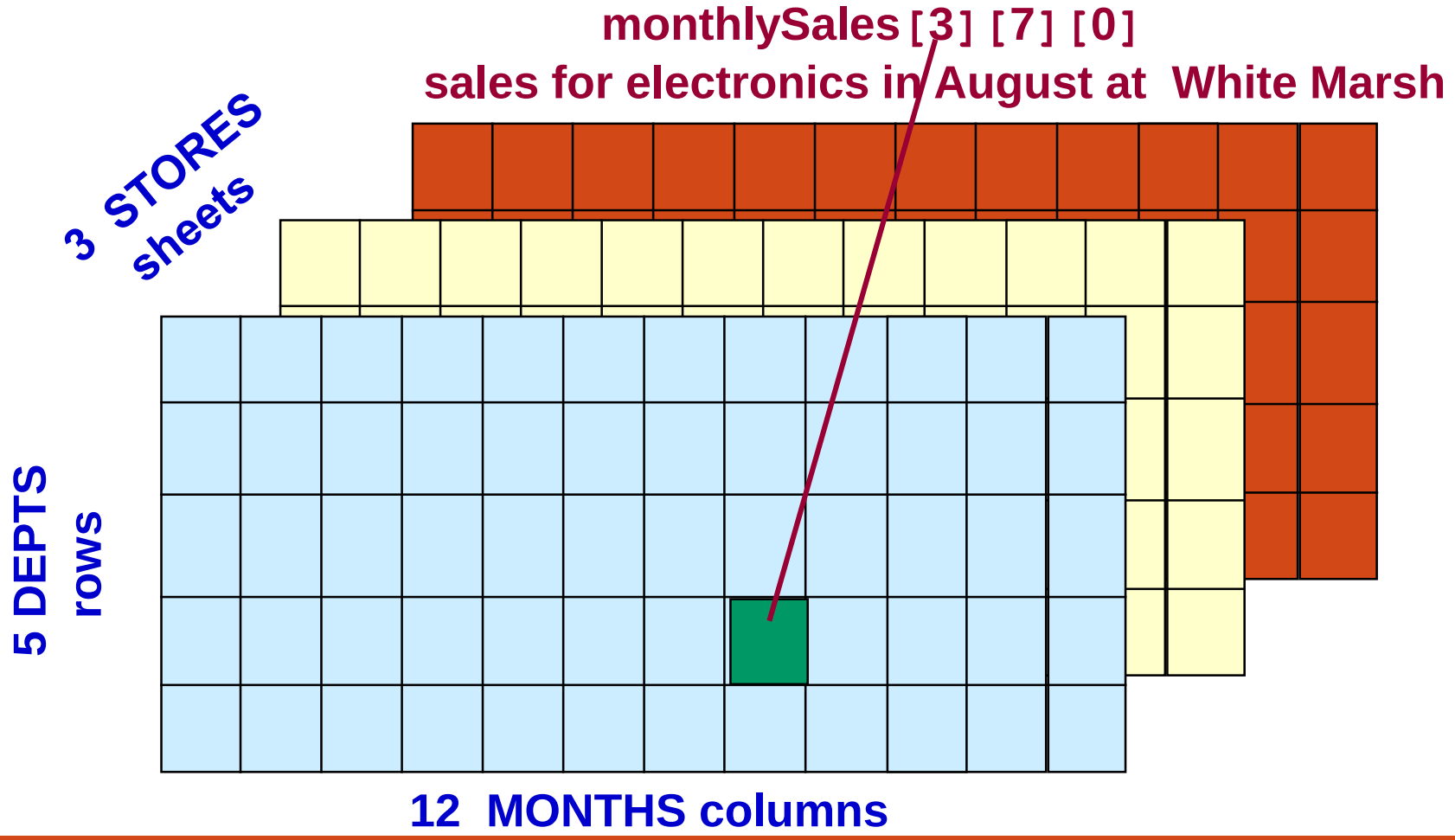
```
const  NUM_DEPTS     =  5 ;     // mens, womens, childrens, electronics, furniture
const  NUM_MONTHS  =  12 ;
const  NUM_STORES   =   3 ;     // White Marsh,  Owings Mills, Towson
int  monthlySales [ NUM_DEPTS ] [ NUM_MONTHS ] [ NUM_STORES ] ;
```

rows columns sheets

OR USING TYPEDEF

```
typedef  int  MonthlySalesType [NUM_DEPTS] [NUM_MONTHS] [NUM_STORES] ;
MonthlySalesType   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 each month 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
.	.	
.	.	
.	.	

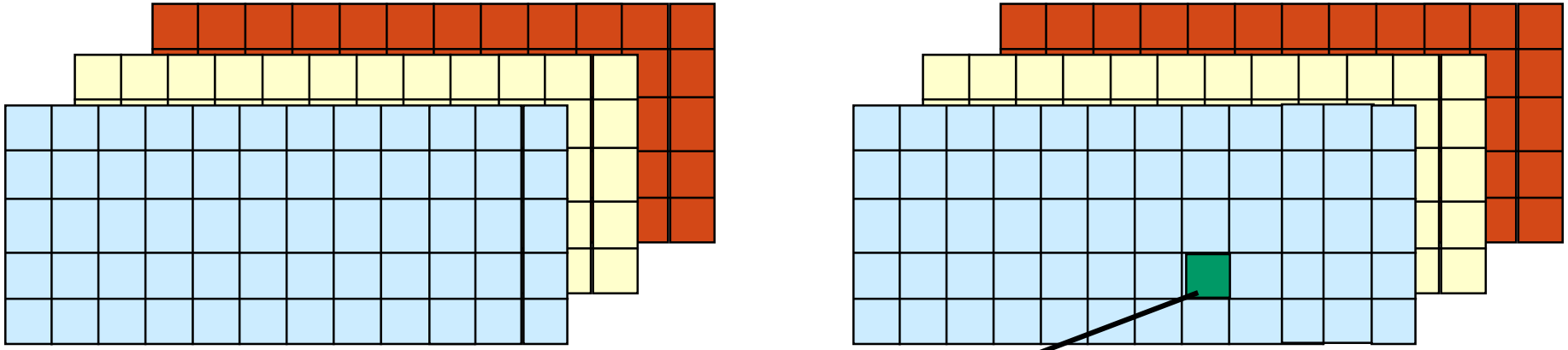
COMBINED SALES FOR December

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

```
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] ;

      . . . .

for  ( month = 0 ; month < NUM_MONTHS ;  month++ )
{
       cout  <<  "COMBINED  SALES  FOR  "  ;
       WriteOut(month) ;                  // function call to write the name of month

       cout  << "DEPT #        DEPT NAME           SALES $" << endl;

       for  (dept = 0 ; dept < NUM_DEPTS ; dept++ )
       {       totalSales = 0;                                  // sum over all stores
        for  (store = 0 ; store < NUM_STORES ; store++ )
              totalSales = totalSales + monthlySales [dept] [month] [store] ;

              WriteDeptNameAndSales(dept, totalSales ) ;        // function call
       }
}
```

Adding a Fourth Dimension . . .

```
const  NUM_DEPTS     =   5 ;     // mens, womens, childrens, electronics, furniture
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] ;
```



year 0 year 1

moreSales[3] [7] [0] [1]

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

LECTURE 12

C++ array class

C++ Provide array class for array containers

Introduction

- Arrays are sequence container of fixed size. Container is a objects that holds data of same type. Sequence containers store elements strictly in linear sequence.
- The container class uses implicit constructor to allocate required memory statically. Memory is allocated at the compile time, hence array size cannot shrink or expand at runtime. All elements inside array are located at contiguous memory locations.

Definition

Below is definition of std::array from <array> header file.

```
template < class T, size_t N >
class array;
```

C++ Provide array class for array containers

Parameters

T – Type of the element contained.

T may be substituted by any other data type including user-defined type.

N – Size of the array.

Zero sized arrays are also valid. In that case array.begin() and array.end() points to same location. But behavior of calling front() or back() is undefined.

Demo Program:

array1.cpp

Go Dev C++!!!

C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch12\array1\array1.exe

```
     0            3 type: i
     1            3 type: i
----------------------------
     0          3.4 type: f
     1          2.8 type: f
     2 2.02085e-039 type: f
----------------------------
     0            H type: c
     1            E type: c
     2            L type: c
     3            L type: c
     4            O type: c
----------------------------


--------------------------------
Process exited after 0.02 seconds with return value 0
Press any key to continue . . .
```

```
#include <iostream>    // array.h
using std::cout;
using std::endl;
#include <iomanip>
using std::setw;
#include <typeinfo>
```

```
#ifndef ARRAY_H_
#define ARRAY_H_
template< typename T > class array {
private:
        int size;
        T *myarray;
public:
        array (int s) {
        size = s;
        myarray = new T [size];
        }
void setArray ( int elem, T val) {
        myarray[elem] = val;
        }
void getArray () {
    for ( int j = 0; j < size; j++ ) {
            cout << setw( 7 ) << j << setw( 13 ) << myarray[ j ]
            << " type: " << typeid(myarray[ j ]).name() << endl;
        }
        cout << "----------------------------" << endl;
        }
};
#endif
```

Note:
C++ STL provide an array template class which can be used to create array of any data type.

```
#include "array.h"
int main()
{
// instantiate int_array object of class array<int> with size 2
array< int > int_array(2);
// set value to a first element
// call to array class member function to set array elements
int_array.setArray(0,3);
// set value to a second element
// NOTE: any attempt to set float to an int array will be translated to int value
int_array.setArray(1,3.4);

// call to array class member function to display array elements
int_array.getArray();

// instantiate float_array object of class array<float> with size 3
array< float > float_array(3);

// set value to a first element
// call to array class member function to set array elements
float_array.setArray(0,3.4);
// set value to a second element
float_array.setArray(1,2.8);

// call to array class member function to display array elements
float_array.getArray();

// instantiate float_array object of class array<char> with size 5
array< char > char_array(5);

// set value to a first element
// call to array class member function to set array elements
char_array.setArray(0,'H');
// set value to a other array elements
char_array.setArray(1,'E');
char_array.setArray(2,'L');
char_array.setArray(3,'L');
char_array.setArray(4,'O');

char_array.getArray();

return 0;
}
```

C++ Array Class #include <array>

Type Definition	Description
array::const_iterator	The type of a constant iterator for the controlled sequence.
array::const_pointer	The type of a constant pointer to an element.
array::const_reference	The type of a constant reference to an element.
array::const_reverse_iterator	The type of a constant reverse iterator for the controlled sequence.
array::difference_type	The type of a signed distance between two elements.
array::iterator	The type of an iterator for the controlled sequence.
array::pointer	The type of a pointer to an element.
array::reference	The type of a reference to an element.
array::reverse_iterator	The type of a reverse iterator for the controlled sequence.
array::size_type	The type of an unsigned distance between two elements.
array::value_type	The type of an element.

Operator	Description
array::operator=	Replaces the controlled sequence.
array::operator[]	Accesses an element at a specified position.

Member Function	Description
array::array	Constructs an array object.
array::assign	Replaces all elements.
array::at	Accesses an element at a specified position.
array::back	Accesses the last element.
array::begin	Designates the beginning of the controlled sequence.
array::cbegin	Returns a random-access const iterator to the first element in the array.
array::cend	Returns a random-access const iterator that points just beyond the end of the array.
array::crbegin	Returns a const iterator to the first element in a reversed array.
array::crend	Returns a const iterator to the end of a reversed array.
array::data	Gets the address of the first element.
array::empty	Tests whether elements are present.
array::end	Designates the end of the controlled sequence.
array::fill	Replaces all elements with a specified value.
array::front	Accesses the first element.
array::max_size	Counts the number of elements.
array::rbegin	Designates the beginning of the reversed controlled sequence.
array::rend	Designates the end of the reversed controlled sequence.
array::size	Counts the number of elements.

LECTURE 12

Array Processing (class)

Array Processing (I)

- Accessor
  1. at() :- This function is used to access the elements of array.
  2. get() :- This function is also used to access the elements of array. This function is not the member of array class but overloaded function from class tuple.
  3. operator[] :- This is similar to C-style arrays. This method is also used to access array
- Traversal
  4. front() :- This returns the first element of array.
  5. back() :- This returns the last element of array.

Array Processing (II)

- Size Detection

  6. size() :- It returns the number of elements in array. This is a property that C-style arrays lack.

  7. max_size() :- It returns the maximum number of elements array can hold i.e, the size with which array is declared. The size() and max_size() return the same value.

- Swap

  8. swap() :- The swap() swaps all elements of one array with other.

- Filling

  9. empty() :- This function returns true when the array size is zero else returns false.

  10. fill() :- This function is used to fill the entire array with a particular value.

Demo Program:

arrayclass/accessor package

Go Code::Block!!!

```
#include<iostream>
#include<array> // for array, at()
#include<tuple> // for get()
using namespace std;
int main()
{
   // Initializing the array elements
   array<int,6> ar = {1, 2, 3, 4, 5, 6};

   // Printing array elements using at()
   cout << "The array elemets are (using at()) : ";
   for ( int i=0; i<6; i++)
   cout << ar.at(i) << " ";
   cout << endl;

   // Printing array elements using get()
   cout << "The array elemets are (using get()) : ";
   cout << get<0>(ar) << " " << get<1>(ar) << " ";
   cout << get<2>(ar) << " " << get<3>(ar) << " ";
   cout << get<4>(ar) << " " << get<5>(ar) << " ";
   cout << endl;

   // Printing array elements using operator[]
   cout << "The array elements are (using operator[]) : "
   for ( int i=0; i<6; i++)
   cout << ar[i] << " ";
   cout << endl;

   return 0;

}
```

Note;
Widows Key + R Then Browse through the directory.
Use build.bat script or Codeblock. (Dev C++ does not support C++11)

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch11\arraypackage\accessor>accessor
The array elemets are (using at()) : 1 2 3 4 5 6
The array elemets are (using get()) : 1 2 3 4 5 6
The array elements are (using operator[]) : 1 2 3 4 5 6
```

Demo Program:

arrayclass/traverse package

Go Code::Block!!!

```
// C++ code to demonstrate working of
// front() and back()
#include<iostream>
#include<array> // for front() and back()
using namespace std;
int main()
{
    // Initializing the array elements
    array<int,6> ar = {1, 2, 3, 4, 5, 6};
    // Printing first element of array
    cout << "First element of array is : ";
    cout << ar.front() << endl;
    // Printing last element of array
    cout << "Last element of array is : ";
    cout << ar.back() << endl;
    return 0;
}
```

Note;
Widows Key + R Then Browse through the directory.
Use build.bat script or Codeblock. (Dev C++ does not support C++11)

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch11\arraypackage\traverse>traverse
First element of array is : 1
Last element of array is : 6
```

Demo Program:

arrayclass/size package

Go Code::Block!!!

```cpp
// C++ code to demonstrate working of
// size() and max_size()
#include<iostream>
#include<array> // for size() and max_size()
using namespace std;
int main(){
    // Initializing the array elements
    array<int,6> ar = {1, 2, 3, 4, 5, 6};
    // Printing number of array elements
    cout << "The number of array elements is : ";
    cout << ar.size() << endl;
    // Printing maximum elements array can hold
    cout << "Maximum elements array can hold is : ";
    cout << ar.max_size() << endl;
    return 0;
}
```

Note;
Widows Key + R Then Browse through the directory.
Use build.bat script or Codeblock. (Dev C++ does not support C++11)

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch11\arraypackage\size>size
The number of array elements is : 6
Maximum elements array can hold is : 6
```

Demo Program:

arrayclass/swap package

Go Code::Block!!!

Note;
Widows Key + R Then Browse through the directory.
Use build.bat script or Codeblock. (Dev C++ does not support C++11)

```
C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\ch11\arraypackage\swap>swap
The first array elements before swapping are : 1 2 3 4 5 6
The second array elements before swapping are : 7 8 9 10 11 12
The first array elements after swapping are : 7 8 9 10 11 12
The second array elements after swapping are : 1 2 3 4 5 6
```

```cpp
// C++ code to demonstrate working of swap()
#include<iostream>
#include<array> // for swap() and array
using namespace std;
int main(){
    // Initializing 1st array
    array<int,6> ar = {1, 2, 3, 4, 5, 6};
    // Initializing 2nd array
    array<int,6> ar1 = {7, 8, 9, 10, 11, 12};
    // Printing 1st and 2nd array before swapping
    cout << "The first array elements before swapping are : ";
    for (int i=0; i<6; i++)
    cout << ar[i] << " ";
    cout << endl;
    cout << "The second array elements before swapping are : ";
    for (int i=0; i<6; i++)
    cout << ar1[i] << " ";
    cout << endl;
    // Swapping ar1 values with ar
    ar.swap(ar1);
    // Printing 1st and 2nd array after swapping
    cout << "The first array elements after swapping are : ";
    for (int i=0; i<6; i++)
    cout << ar[i] << " ";
    cout << endl;
    cout << "The second array elements after swapping are : ";
    for (int i=0; i<6; i++)
    cout << ar1[i] << " ";
    cout << endl;
    return 0;
}
```

Demo Program:

arrayclass/filling package

Go Code::Block!!!

```cpp
// C++ code to demonstrate working of empty()
// and fill()
#include<iostream>
#include<array> // for fill() and empty()
using namespace std;
int main(){
    // Declaring 1st array
    array<int,6> ar;
    // Declaring 2nd array
    array<int,0> ar1;
    // Checking size of array if it is empty
    ar1.empty()? cout << "Array empty":
        cout << "Array not empty";
    cout << endl;
    // Filling array with 0
    ar.fill(0);
    // Displaying array after filling
    cout << "Array after filling operation is : ";
    for ( int i=0; i<6; i++)
        cout << ar[i] << " ";
    return 0;
}
```

Note;

Widows Key + R Then Browse through the directory.

Use build.bat script or Codeblock. (Dev C++ does not support C++11)

"C:\Eric_Chou\Cpp Course\C++ Programming Essentials\CppDev\c...

```
Array empty
Array after filling operation is : 0 0 0 0 0 0
Process returned 0 (0x0)   execution time : 0.018 s
Press any key to continue.
```