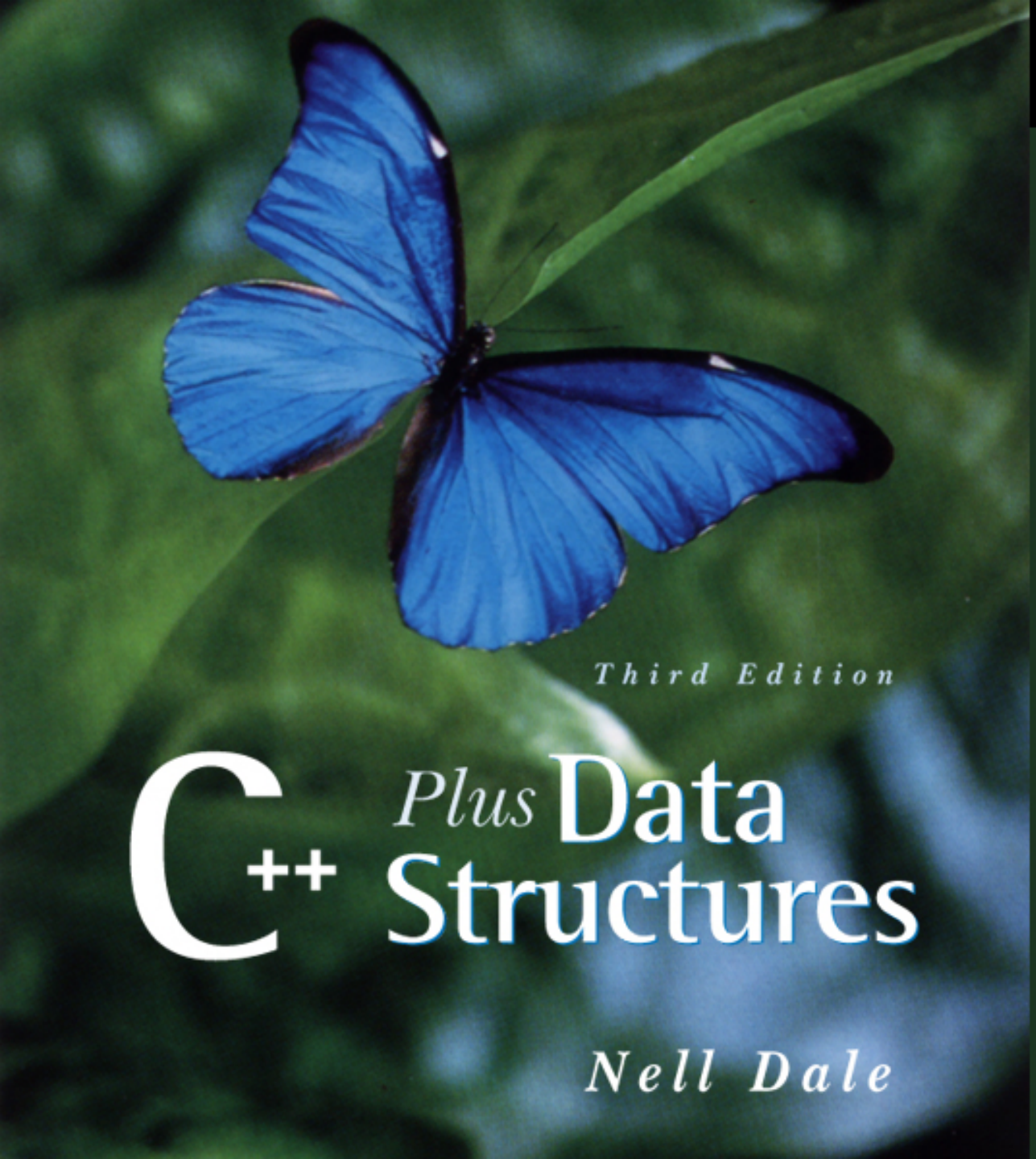


Chapter

2

Data Design and Implementation

Data

- The representation of information in a manner suitable for communication or analysis by humans or machines
- Data are the nouns of the programming world:
  - The objects that are manipulated
  - The information that is processed

Data Abstraction

- **Separation of a data type's logical properties from its implementation.**

LOGICAL PROPERTIES	IMPLEMENTATION
What are the possible values?	How can this be done in C++?
What operations will be needed?	How can data types be used?

Data Encapsulation

- **is the separation of the representation of data from the applications that use the data at a logical level; a programming language feature that enforces information hiding.**

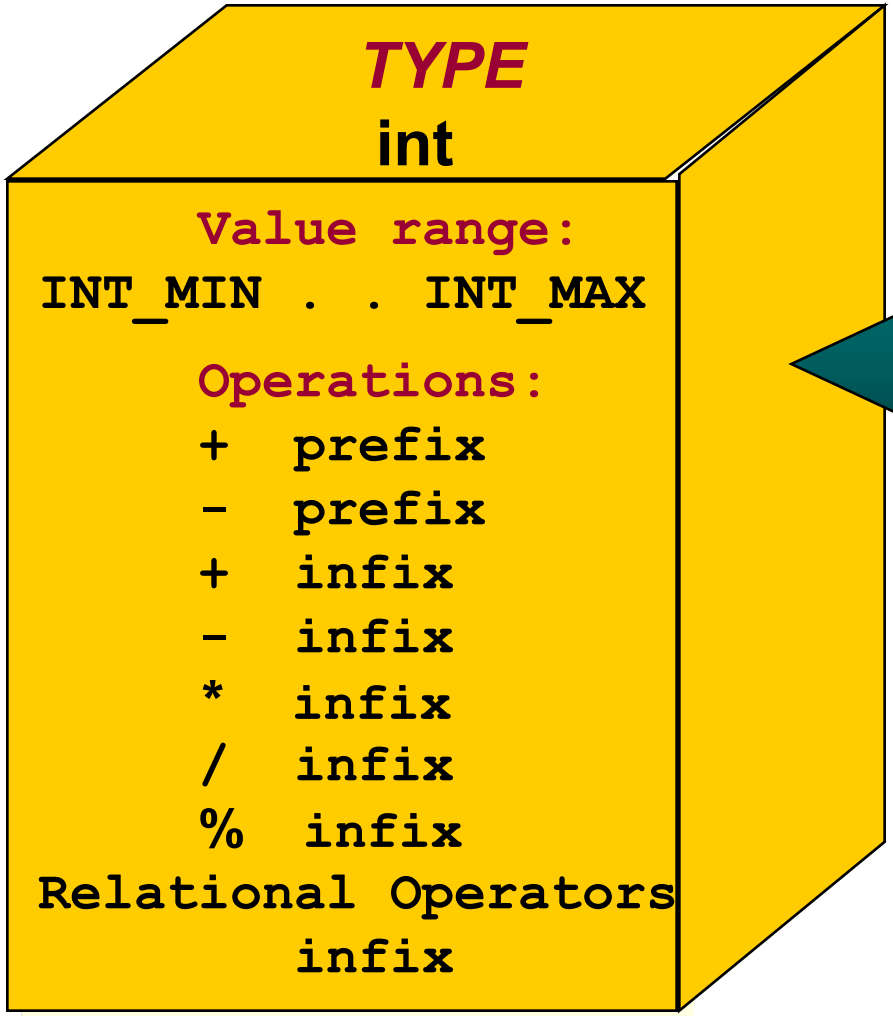
APPLICATION

```
int  y;

y = 25;
```

REPRESENTATION

```
0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1
```

Encapsulated C++ Data Type int



(inside)

Representation of

int

as 16 bits two's complement

+

Implementation of Operations

Abstract Data Type (ADT)

- **A data type whose properties (domain and operations) are specified independently of any particular implementation.**

Data Structure

- A collection of data elements whose organization is characterized by accessing operations that are used to store and retrieve the individual data elements
- The implementation of the composite data members in an abstract data type

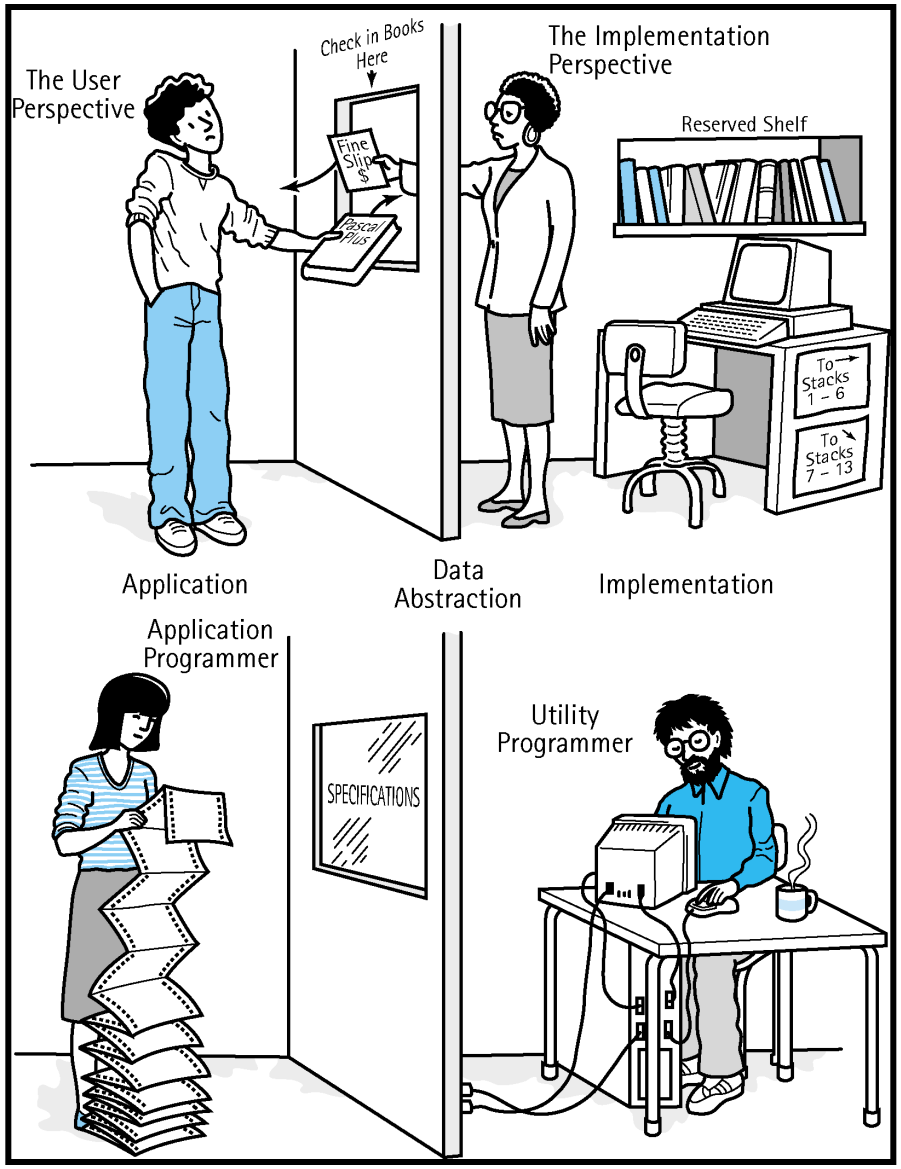
Features of Data Structures

- They are decomposed into their component elements
- The arrangement of the elements affects how each element is accessed
- Both the arrangement of the elements and the way they are accessed can be encapsulated

Data from 3 different levels

- ***Application (or user) level:*** **modeling real-life data in a specific context.**
- ***Logical (or ADT) level:*** **abstract view of the domain and operations.** **WHAT**
- ***Implementation level:*** **specific representation of the structure to hold the data items, and the coding for operations.** **HOW**

Communication between the Application Level and Implementation Level



Viewing a library from 3 different levels

- *Application (or user) level:* **Library of Congress, or Baltimore County Public Library.**
- *Logical (or ADT) level:* **domain is a collection of books; operations include: check book out, check book in, pay fine, reserve a book.**
- *Implementation level:* **representation of the structure to hold the "books", and the coding for operations.**

4 Basic Kinds of ADT Operations

- **Constructor -- creates a new instance (object) of an ADT.**
- **Transformer -- changes the state of one or more of the data values of an instance.**
- **Observer -- allows us to observe the state of one or more of the data values without changing them.**
- **Iterator -- allows us to process all the components in a data structure sequentially.**

Abstraction and Built-In Types

- **A built-in simple type such as int or float is an abstraction whose underlying implementation is defined in terms of machine-level operation**
  - **Remind the previous example of the int type**
- **The same perspective applies to built-in composite data type provided in programming languages to build data objects**

Composite Data Type

A composite data type is a type which

- **stores a collection of individual data components under one variable name,**
- **and allows the individual data components to be accessed.**

Two Forms of Composite Data Types

UNSTRUCTURED

Components are not organized with respect to one another.

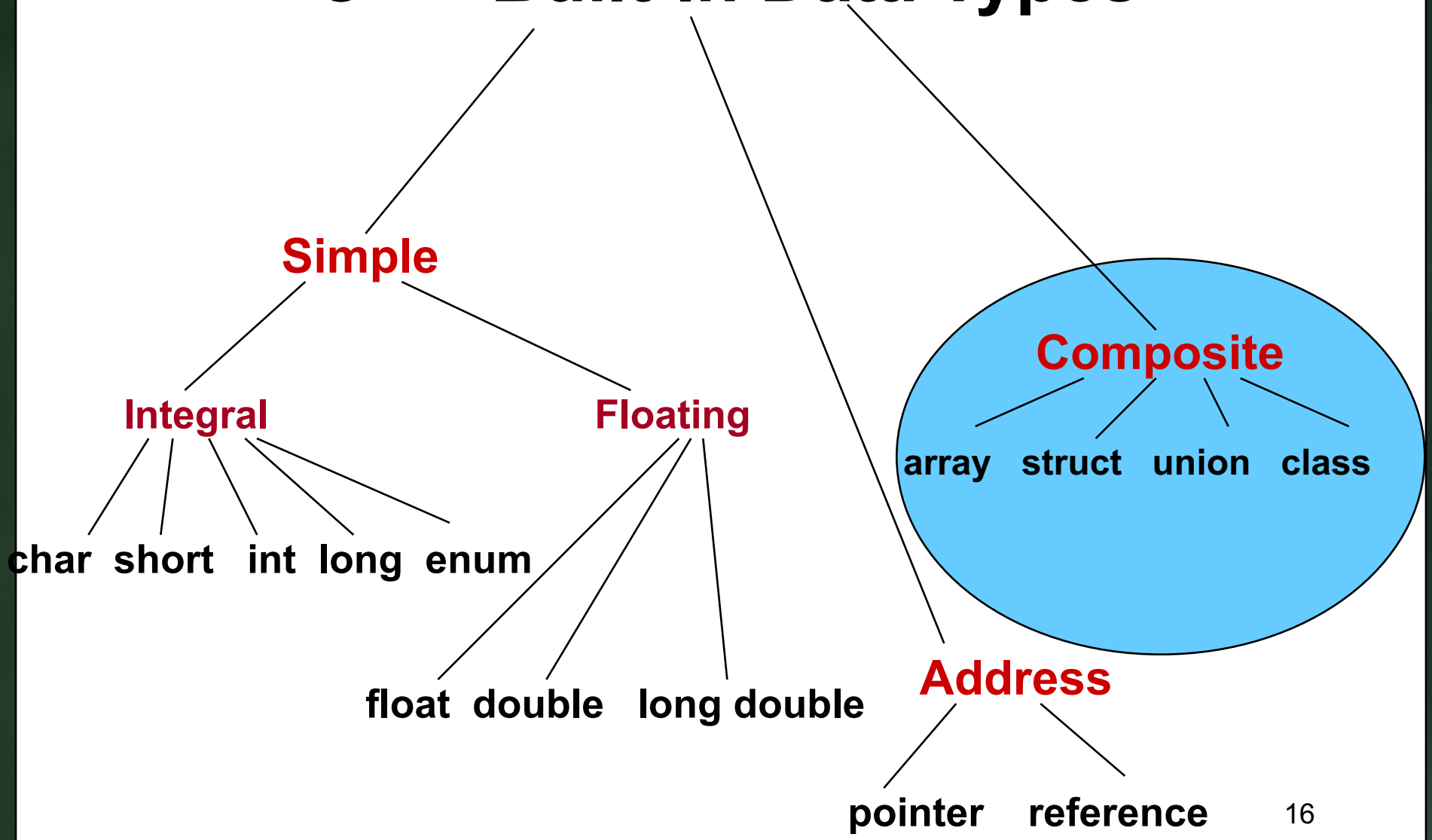
EXAMPLES:
classes and structs

STRUCTURED

The organization determines method used to access individual data components.

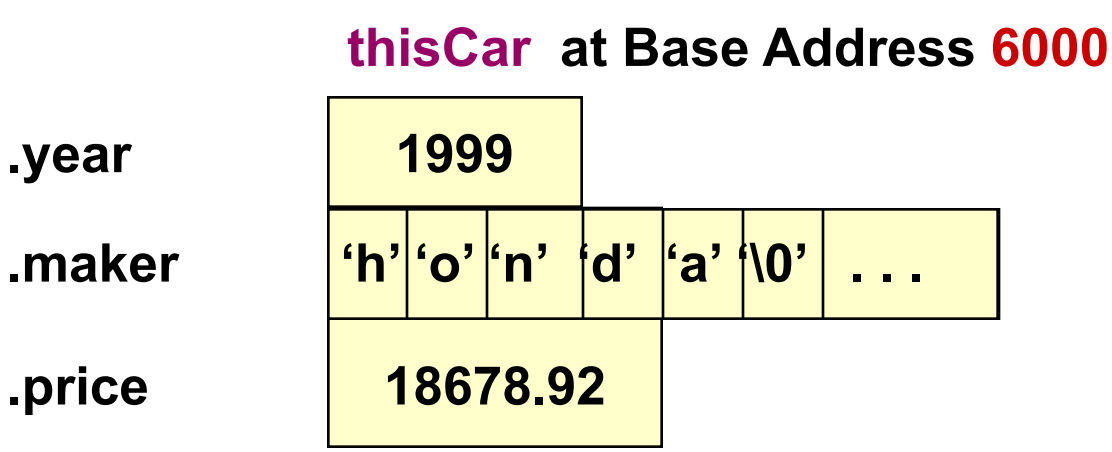
EXAMPLES: arrays

C++ Built-In Data Types



Records at the Logical Level

A record is a composite data type made up of a finite collection of not necessarily homogeneous elements called *members* or *fields*. For example . . .



struct CarType

```
struct  CarType
{
    int      year ;
    char     maker[10];
    float    price ;
} ;

CarType  thisCar;     //CarType variables
CarType  myCar;
```

Accessing struct members

The member selection operator (period .) is used between the variable name and the member identifier to access individual members of a record (struct or class) type variable.

EXAMPLES

```
myCar.year
thisCar.maker[4]
```

Valid struct operations

- Operations valid on an entire struct type variable:

  assignment to another struct variable of same type,

  pass as a parameter to a function
  (either by value or by reference),

  return as the value of a function.

Records at the Application Level

- Useful for modeling objects that have a number of characteristics
  - collect various types of data about an object
  - refer to the whole object by a single name
  - Refer to the different members of the object by name
- Also useful for defining other programmer-defined data structures

Records at the Implementation Level

- In order to implement
  - Memory cells must be reserved for the data
  - The accessing functions must be determined
- Base address
  - the location in memory of the first cell in the record
- Member-length-offset table
  - A table containing the number of memory locations needed for each member of the record

One-Dimensional Array at the Logical Level

A one-dimensional array is a structured composite data type made up of a finite, fixed size (*known at compile time*) collection of homogeneous (*all of the same data type*) elements having relative positions and to which there is direct access (*any element can be accessed immediately*).

Array operations (*creation, storing a value, retrieving a value*) are performed using a declaration and indexes.

One-Dimensional Array at Application Level

- **The natural structure for the storage of lists of like data elements**
- **Examples**
  - Grocery lists
  - Price lists
  - Lists of phone numbers
  - Lists of student records
  - Lists of characters (a string)

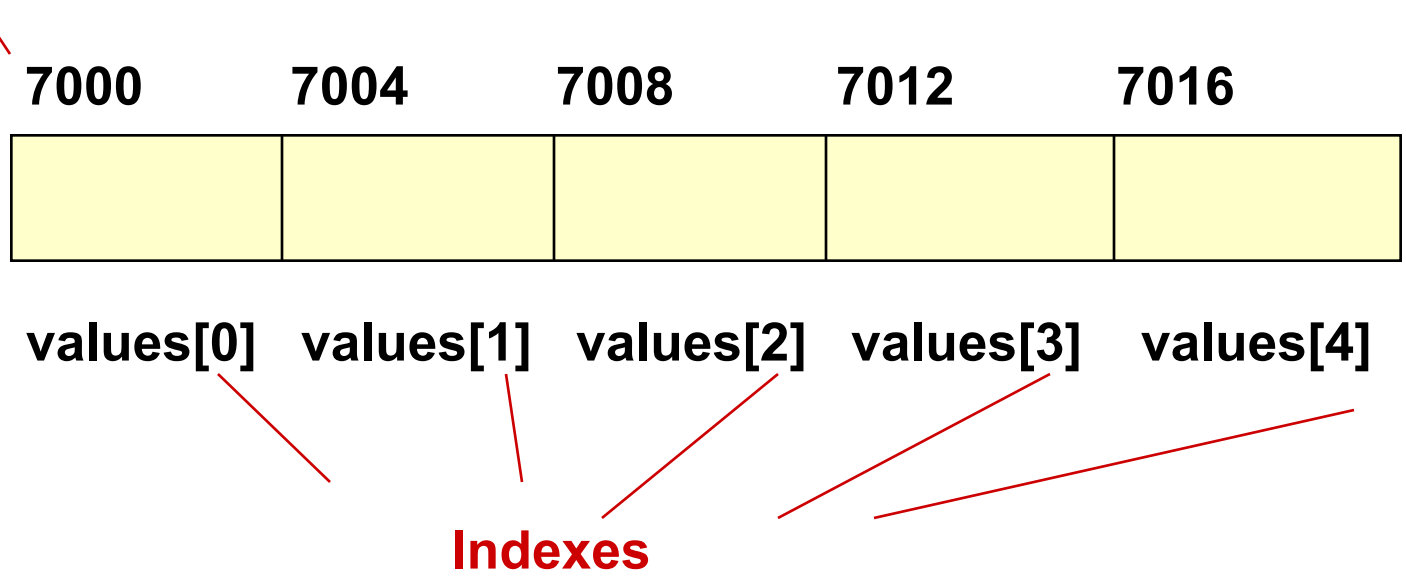
Implementation Example

This ACCESSING FUNCTION gives position of values[Index]

Address(Index) = BaseAddress + Index * SizeOfElement

float values[5]; ***// assume element size is 4 bytes***

Base Address



One-Dimensional Arrays in C++

- The index must be of an integral type (char, short, int, long, or enum).
- The index range is always 0 through the array size minus 1.
- Arrays cannot be assigned one to another, and cannot be the return type of a function.

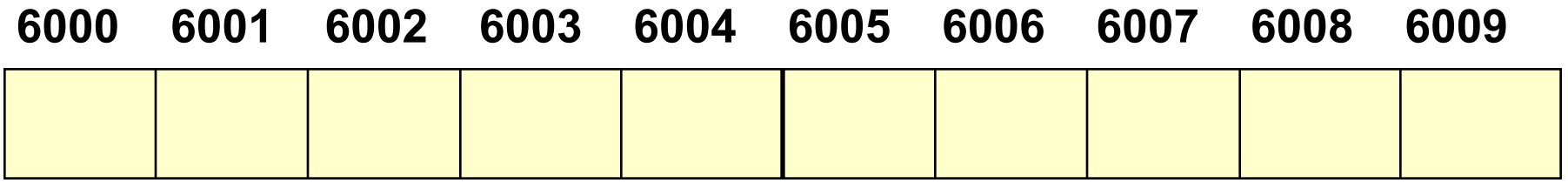
Another Example

This ACCESSING FUNCTION gives position of name[Index]

Address(Index) = BaseAddress + Index * SizeOfElement

char name[10]; ***// assume element size is 1 byte***

Base Address



Two-Dimensional Array at the Logical Level

A two-dimensional array is a structured composite data type made up of a finite, fixed size collection of homogeneous elements ordered in two dimensions having relative positions and to which there is direct access.

Array operations (*creation, storing a value, retrieving a value*) are performed using a declaration and a pair of indexes (*called row and column*) representing the component's position in each dimension.

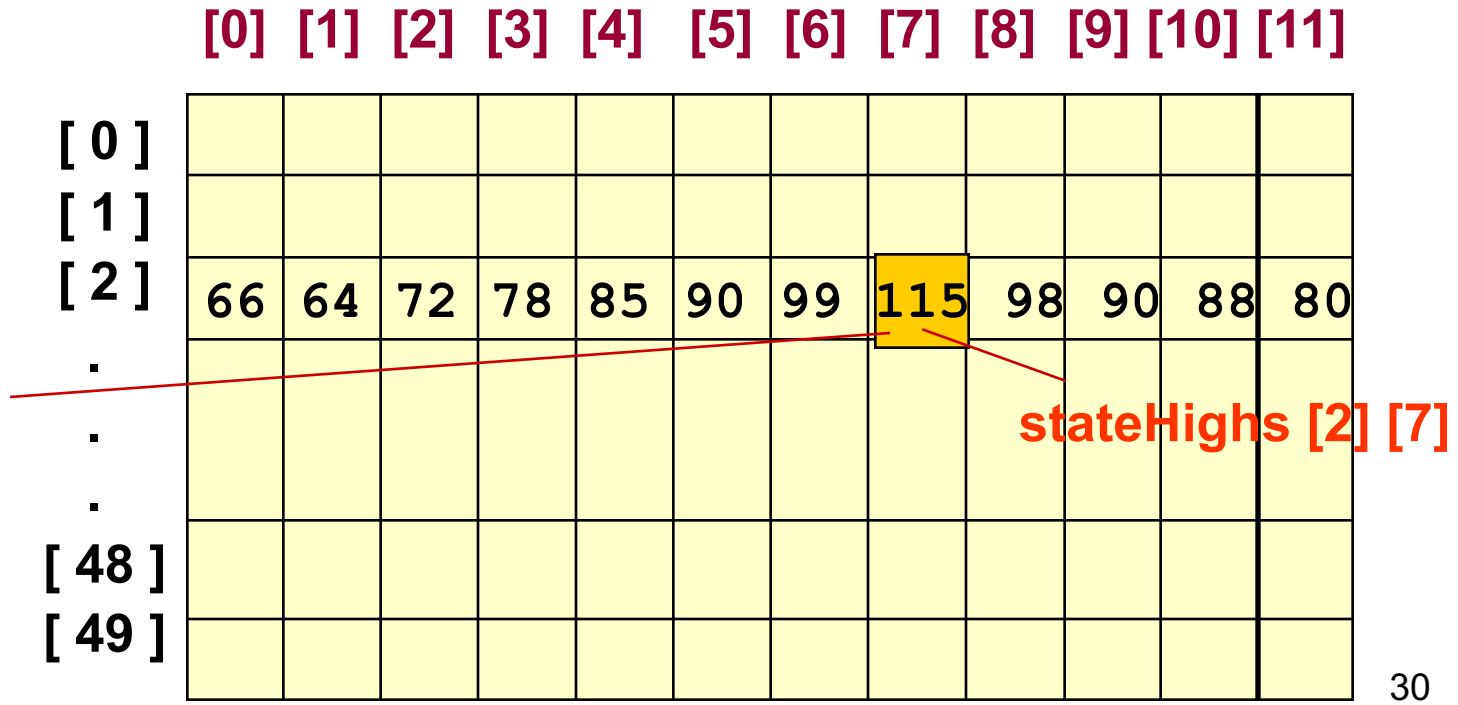
Two-Dimensional Array at Application Level

- **The ideal data structure for modeling data that are logically structured as a table with rows and columns**
  - The first dimension: rows
  - The second dimension: columns

Implementation Example

EXAMPLE -- To keep monthly high temperatures for 50 states in a two-dimensional array.

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



Finding the average high temperature for Arizona

```
int   total   = 0 ;
int   month ;
int   average ;

for ( month = 0 ; month < NUM_MONTHS ; month ++ )
       total = total + stateHighs [ 2 ] [ month ] ;

average   = int ( total / 12.0   + 0.5 ) ;
```

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

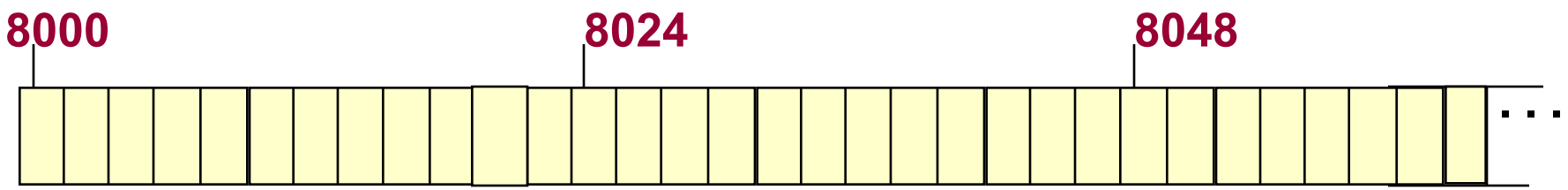
rows

columns

STORAGE

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

Base Address

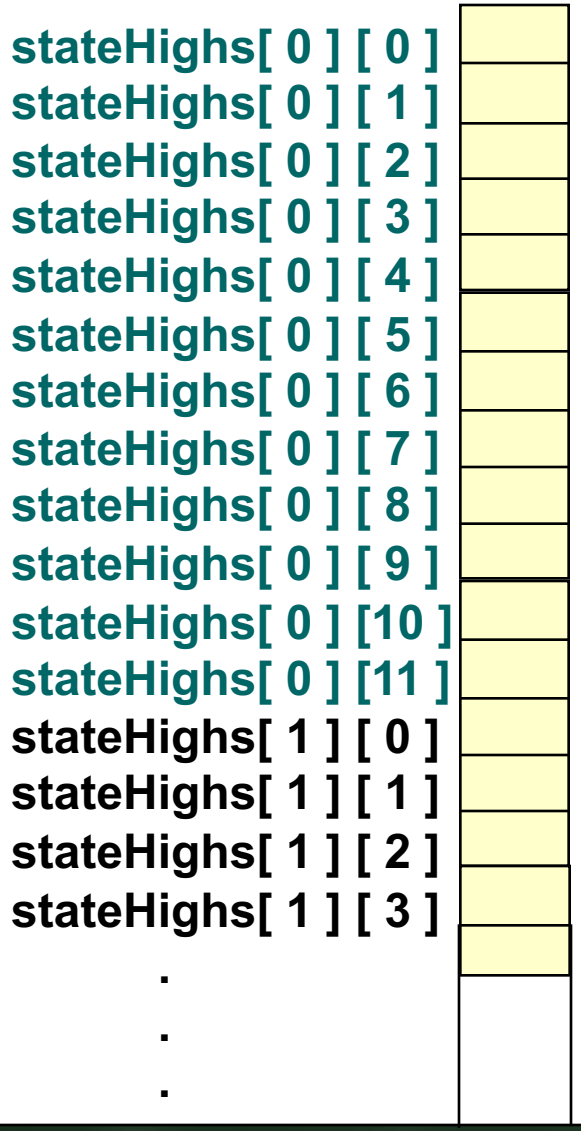


12 highs for state 0
Alabama
first row

12 highs for state 1
Alaska
second row

etc.

Implementation Level View



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.

C++ class data type

- A class is an unstructured type that encapsulates a fixed number of data components (data members) with the functions (called member functions) that manipulate them.
- The predefined operations on an instance of a class are whole assignment and component access.

class DateType Specification

```
//  SPECIFICATION FILE      ( datetype.h )

class  DateType                 // declares a  class data type
{

public :                                //  4 public member functions

   void Initialize (int newMonth, int newDay, int newYear ) ;
   int  YearIs( )    const ;       //  returns year
   int  MonthIs( )   const ;       // returns month
   int  DayIs( )     const ;       // returns day

private :                               //  3 private data members

   int  year ;
   int  month ;
   int  day ;
} ;
```

Use of C++ data type class

- Variables of a class type are called objects (or instances) of that particular class.
- Software that declares and uses objects of the class is called a client.
- Client code uses public member functions (called methods in OOP) to handle its class objects.
- Sending a message means calling a public member function.

Client Code Using DateType

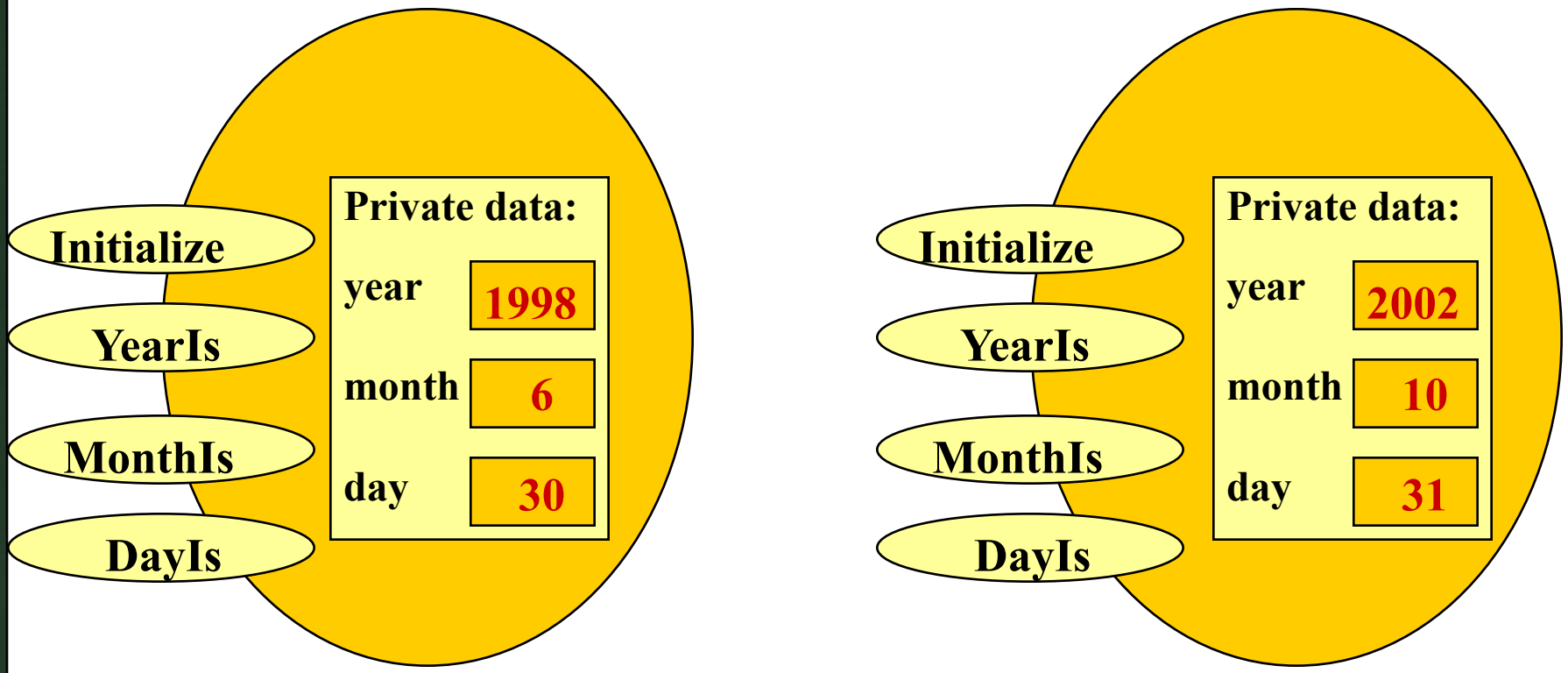
```
#include   "datetype"  // includes specification of the class
#include   "bool"
using namespace std;
int  main ( void )
{
    DateType     startDate ; // declares 2 objects of DateType
    DateType     endDate ;
    bool        retired  =  false ;
    startDate.Initialize ( 6, 30, 1998 ) ;
    endDate.Initialize ( 10, 31, 2002 ) ;
     cout <<  startDate.MonthIs( ) << "/" << startDate.DayIs( )
          << "/" <<  startDate.YearIs( ) << endl;
    while  ( ! retired )
    {    finishSomeTask( ) ;
          .  .  .
    }
}
```

2 separate files generally used for class type

```
//    SPECIFICATION FILE          ( datetype .h )
//    Specifies the data and function members.
class DateType
  {
     public:
          . . .

     private:
               . . .
   } ;
```

```
//    IMPLEMENTATION FILE       ( datetype.cpp )


//    Implements the DateType member functions.

       . . .
```

DateType Class Instance Diagrams



Implementation of DateType member functions

```
// IMPLEMENTATION FILE                          (datetype.cpp)
#include "datetype.h"    // also must appear in client code

void DateType :: Initialize ( int  newMonth, int  newDay,
                              int  newYear )
//  Post:  year is set to newYear.
//  month is set to newMonth.
//  day is set to newDay.
{
   year    =  newYear ;
   month =  newMonth ;
   day      =  newDay ;
}
```

```
int DateType :: MonthIs ( ) const
//  Accessor function for data member month
{
    return  month ;
}

int DateType :: YearIs ( ) const
//  Accessor function for data member year
{
   return  year ;
}

int DateType :: DayIs ( ) const
//  Accessor function for data member day
{
   return  day ;
}
```

Familiar Class Instances and Member Functions

- The member selection operator (.) selects either data members or member functions.
- Header files iostream and fstream declare the istream, ostream,and ifstream, ofstream I/O classes.
- Both cin and cout are class objects and get and ignore are member functions.

```
cin.get (someChar) ;
cin.ignore (100, '\n') ;
```

- These statements declare myInfile as an instance of class ifstream and invoke member function open.

```
ifstream  myInfile ;
myInfile.open ( "A:\\mydata.dat" ) ;
```

Scope Resolution Operator (::)

- C++ programs typically use several class types.
- Different classes can have member functions with the same identifier, like Write( ).
- Member selection operator is used to determine the class whose member function Write( ) is invoked.

```
currentDate .Write( ) ;          // class DateType
numberZ .Write( ) ;              // class ComplexNumberType
```

- In the implementation file, the scope resolution operator is used in the heading before the member function's name to specify its class.

```
void  DateType :: Write ( )   const
{       . . .

}
```

A Short Review of Object-Oriented Programming

- Three inter-related constructs: classes, objects, and inheritance
- Objects are the basic run-time entities in an object-oriented system.
- A class defines the structure of its objects.
- Classes are organized in an “is-a” hierarchy defined by inheritance.

Inheritance

1. Allows programmers to create a new class that is a specialization of an existing class.
2. The new class is called a derived class of the existing class; the existing class is the base class of the new class.

Inheritance & Polymorphism

- **Inheritance** fosters reuse by allowing an application to take an already-tested class and derive a class from it that inherits the properties the application needs

- **Polymorphism:** the ability of a language to have duplicate method names in an inheritance hierarchy and to apply the method that is appropriate for the object to which the method is applied

Inheritance & Polymorphism

Inheritance and polymorphism combined allow the programmer to build useful hierarchies of classes that can be reused in different applications