CS 2133
Data Abstraction & ADTs
(Abstract Data Types)

Abstraction

- **Abstraction is the separation of the essential qualities of an object from the details of how it works or is composed**
  - **Focuses on what, not how**
  - **Is necessary for managing large, complex software projects**

Data Abstraction

- **Data abstraction separates the logical properties of a data type 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?

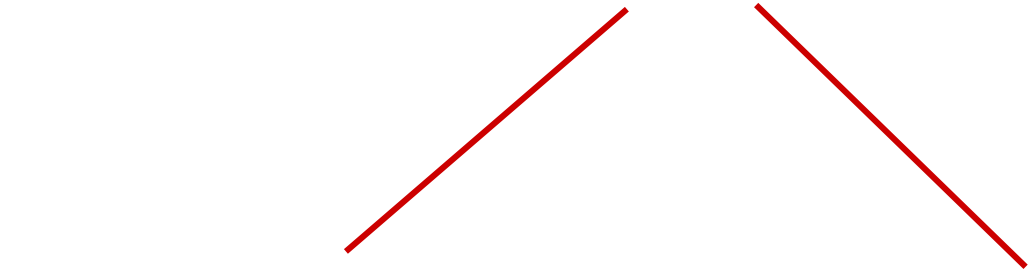
Data Abstraction

- With data abstraction we think about what operations can be performed on a particular type of data and not how it does
- Data abstraction is used as a tool to increase the modularity of a program
- We use it to build **new abstract data types** .

Data Abstraction

- Data Abstraction is one of the most powerful programming paradigms
- It allows us to create our own user defined data types (using the class construct) and define variables (i.e., objects) of those new data types.

Data Type



set of values
(domain)

allowable operations
on those values

FOR EXAMPLE, data type `int` has

domain	operations
-32768 . . . 32767	+, -, *, /, %, >>, <<

Abstract Data Type (ADT)

- An abstract data type is a data type whose properties (domain and operations) are specified (*what*) independently of any particular implementation (*how)*

ADT Specification Example

TYPE

Time

DOMAIN

Each Time value is a time in hours, minutes, and seconds.

OPERATIONS

Set the time

Print the time

Increment by one second

Compare 2 times for equality

Determine if one time is “less than” another

Another ADT Specification

TYPE

ComplexNumber

DOMAIN

Each value is an ordered pair of real numbers (a, b) representing a + b*i*

OPERATIONS

Initialize the complex number

Write the complex number

Add

Subtract

Multiply

Divide

Determine the absolute value of a complex number

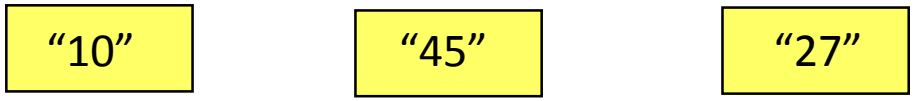
ADT Implementation

- **ADT implementation**
  - **Choose a specific data representation for the abstract data using data types that already exist (built-in or programmer-defined)**
  - **Write functions for each allowable operation**

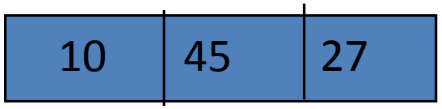
Several Possible Representations of ADT `Time`

3 int variables

3 strings



3-element int array

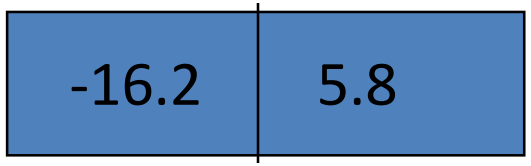


Choice of representation depends on time, space, and algorithms needed to implement operations

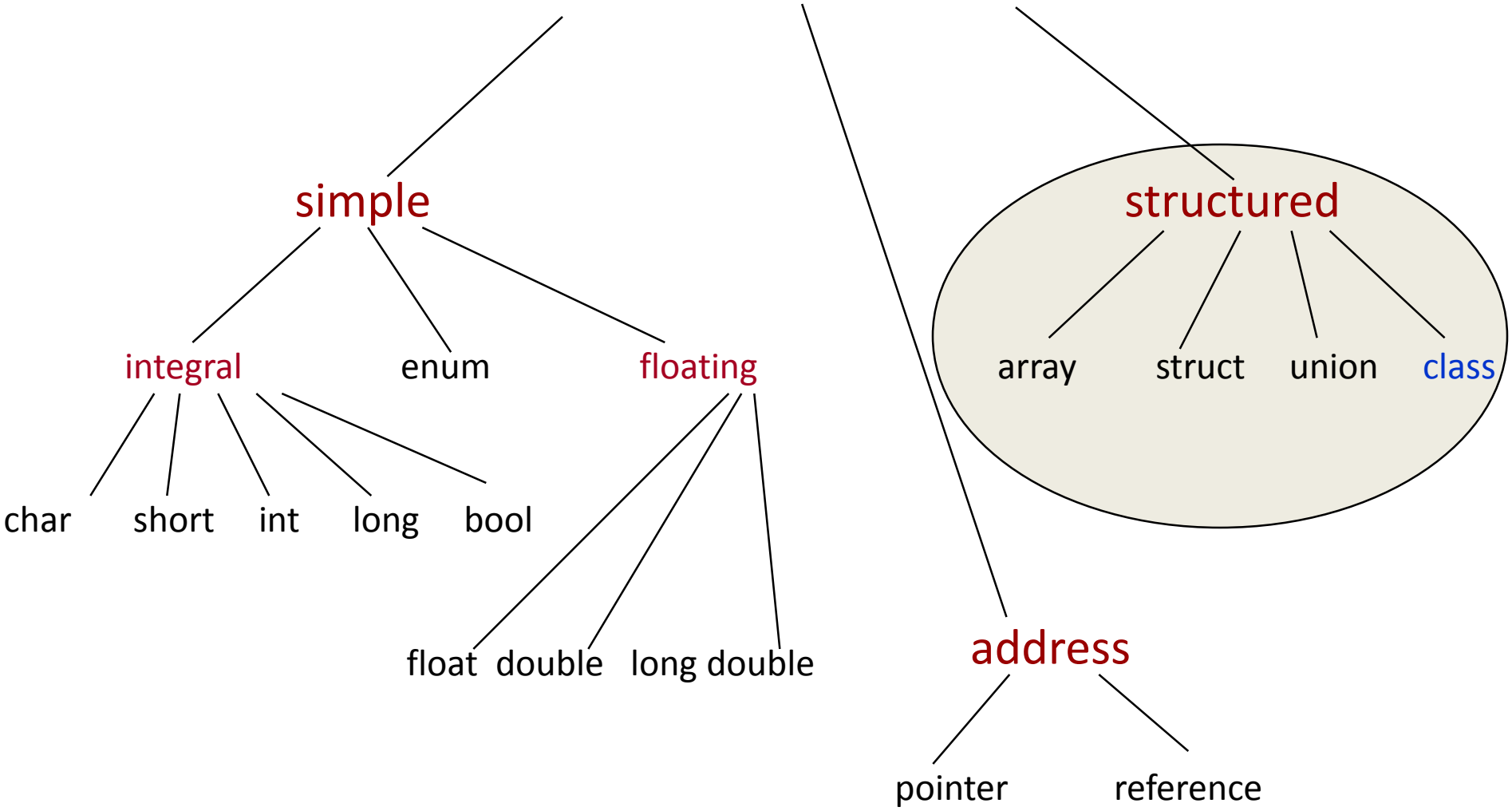
Some Possible Representations of ADT `ComplexNumber`

struct with 2 float members

2-element float 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

Abstract Data Types in C++

- Based on C **`struct`** type and classes
- The *class* is the encapsulation device
  - All of the class instances of a class share a single copy of the member functions
  - Variables of the class type are called *class objects* or *class instances*
  - Each instance has own copy of class data members
  - Instances can be static, stack dynamic, heap dynamic
- Information hiding
  - *Private* clause for hidden entities
  - *Public* clause for interface entities
  - *Protected* clause for inheritance

C++ `class`

- Facilitates re-use of C++ code for an ADT
- Software that uses the class is called a client
- Client code uses class's public member functions to manipulate class objects

`class Time` Specification

```
//  Specification file (Time.h)
class  Time          // Declares a  class data type
{                    //  does not allocate memory
public :             // Five public function members
    void  Set (int  hours , int  mins , int  secs);
    void  Increment ();
    void  Write ()  const;
    bool  Equal (Time   otherTime)  const;
    bool  LessThan (Time   otherTime)  const;
private :            // Three private data members
    int  hrs;
    int  mins;
    int  secs;
};
```

C++ (cont.)

- Constructors:
  - Functions to initialize the data members of instances (they do not create the objects)
  - May also allocate storage if part of the object is heap-dynamic
  - Can include parameters to provide parameterization of the objects
  - Implicitly called when an instance is created
  - Can be explicitly called
  - Name is the same as the class name

Language Examples: C++ (cont.)

- Destructors
  - Functions to clean up after an instance is destroyed; usually just to reclaim heap storage
  - Implicitly called when the object's lifetime ends
  - Can be explicitly called
  - Name is the class name, preceded by a tilde (~)

Member Functions Defined in Class

```
class Stack {
  private:
     int *stackPtr, maxLen, topPtr;
  public:
     Stack() { // a constructor
          stackPtr = new int [100];
          maxLen = 99;   topPtr = -1;   };
     ~Stack () {delete [] stackPtr;};
     void push (int num) {…};
     void pop () {…};
     int top () {…};
     int empty () {…};
}
```

Implicitly inlined → code placed in caller's code

Client Code Using `Time`

```
#include   "time.h"   // Includes specification of the class
using  namespace  std;

int  main ()
{
    Time  currentTime; // Declares two objects of Time
    Time  endTime;
    bool  done  =  false;

    currentTime.Set (5, 30, 0);
    endTime.Set (18, 30, 0);
    while  (! done)
    {  .  .  .

        currentTime.Increment ();
        if  (currentTime.Equal (endTime))
            done  =  true;
    };
}
```

User-defined Data Type

- For example, you might want to add a new type called: list
  - which maintains a list of data
  - the data structure might be an array of structures
  - operations might be to add to, remove, display all, display some items in the list

Example

- For a list of videos, we might start with a struct defining what a video is:

```
struct video {
    char title[100];
    char category[5];
    int quantity;
};
```

Recall: Data Structure

- Data structure usually refers to an organization for data in main memory.

Example

- For a list of videos data type:

```
class list {
     public:
          list();
          int add (const video &);
          int remove (char title[]);
          int display_all();
     private:
          video my_list[CONST_SIZE];
          int num_of_videos;
};
```

Example

- For a client to create a list object:

```
main() {
        list home_videos; //has an array of 100 videos
        list kids_shows; //another 100 videos here...
        …
        video out_of_site;
        scanf("%s",out_of_site.title);
        …
        home_videos.add(out_of_site); //use operation
```