CSC230

Intro to C++ Lecture 15

Outline

- □ Review of the OOP principles
- □ What is Data Abstraction? What is ADT?
- □ Complex Number ADT Example

Object-Oriented Programming: A Review

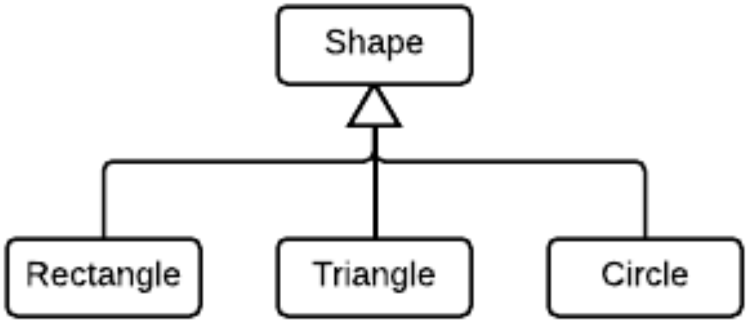
- What is C++?
  - It’s an object-oriented programming language.
  - What are the principles for OOP language.
    - Inheritance
    - Polymorphism
    - Encapsulation
    - Abstraction

Object-Oriented Programming: A Review

- Inheritance
  - Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application.
  - class derived-class : access-specifier base-class
  - Example

Object-Oriented Programming: A Review

- Polymorphism
  - Polymorphism means having multiple forms of one thing.

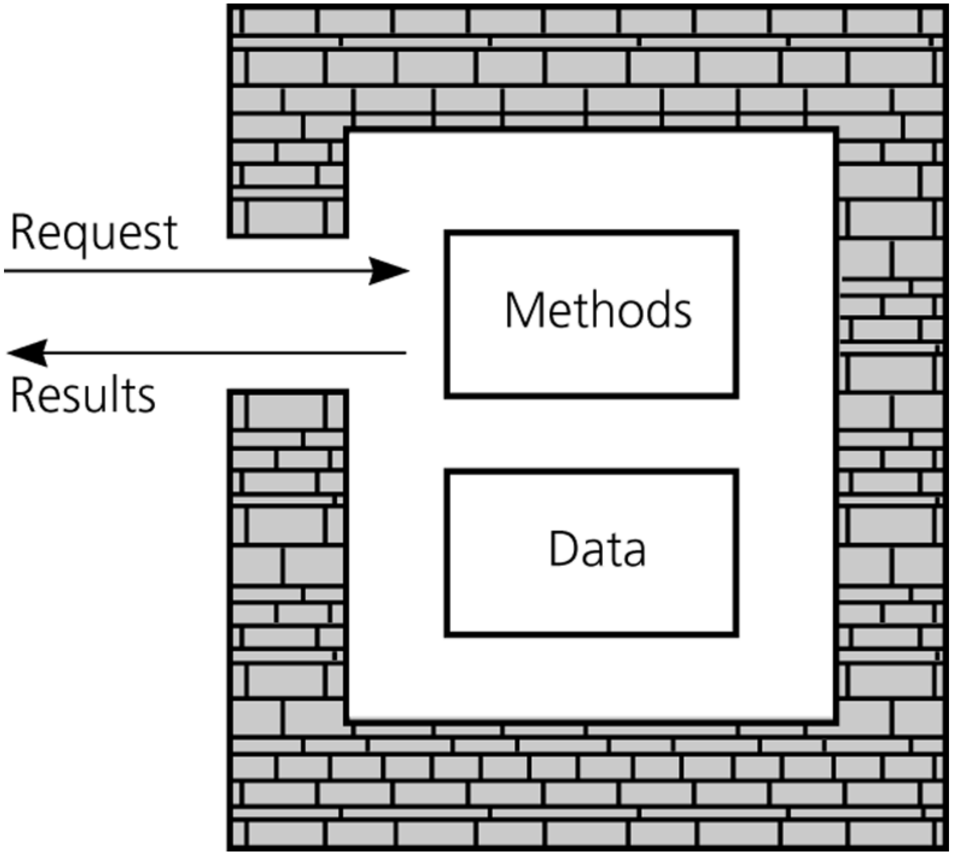


  - How to calculate the area of each type?
    - R = length * width
    - T = base * height
    - C = π * Radius^2
    - Examples, poly.cpp ; poly-1.cpp

Object-Oriented Programming: A Review

- Encapsulation
  - Encapsulation is basically the approach of hiding specific details inside a class.
  - Private
  - Example

C++ Classes



An object’s data and methods

are encapsulated

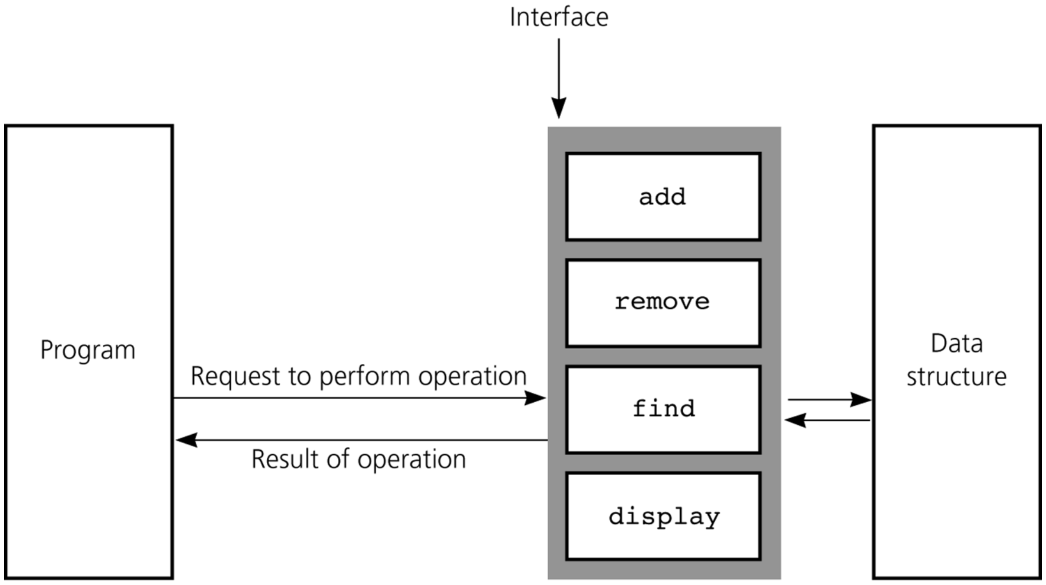
What is Data Abstraction?

- Concept of "**Abstraction**"
  - Allows us to consider the high-level characteristics of something without getting bogged down in the details
  - For example: Shape class, getArea.

- **Data Abstraction**
  - We know what a data type and/or operation can do
  - How it is done is hidden

Abstract Data Types

- Data abstraction
  - Asks you to think *what you can do* to a collection of data independently of *how you do it*
  - Allows you to develop each data structure in relative isolation from the rest of the solution



Wall of ADT operations

What is an Abstract Data Type (ADT)

An Abstract Data Type, or ADT (from a user point):

A type (collection of data together with operations on them), where:

- We state in some fashion what the operations do
- We may give constraints on the operations, such as how much they cost (how much time or space they must take)
- It provides equal attention to data and operations

- Common examples of ADTs:
  - Built-in types:
    - boolean, integer, array, etc.
  - User-defined types:
    - list, stack, queue, tree

Built-in ADTs

Boolean

- Values:
  - true and false
- Operations:
  - and, or, not

integer

- Values:
  - Whole numbers between MIN and MAX values
- Operations:
  - add, subtract, multiply, divide

arrays

- Values:
  - Homogeneous elements, i.e., array of X
- Operations:
  - initialize, store, retrieve, copy

User-defined ADTs

List

- Values:
  - a sequence of values(int, float, char, etc)
- Operations:
  - append, insert, remove, empty, size, etc.

queue

- Values:
  - Values: Queue elements, i.e., queue of X
- Operations:
  - Operations: create, destroy/dispose, enqueue, dequeue, is_empty, is_full

Benefits

- Manufacturer Benefits:
  - easy to modify, maintain
  - reusable
- Client Benefits:
  - simple to use, understand
  - component–based

ADT Example: List

List:

- An ordered sequence of values
- The same value may occur more than once

Operation	Description	Input(s)
append	Add a new value to the end of the list	Item
insert	Add a new value at a particular location shifting others back	Index, item
remove	Remove value at the given location	Index
get	Get value at given location	Index
empty	Returns true if there are no values in the list	
size	Returns the number of values in the list	
find	Return the location of a given value	item

ADT Example: Set

Set:

- A group of values with the same type (such as int)
- The values must be **different**

Operation	Description	Input(s)
set	Create a new set, which is empty	
size	Returns the number of values in the list	
add	Add an item to the set	item
delete	Delete an item from the set	item
isIn	Check whether one item is in the set	item

ADT Example: Stack

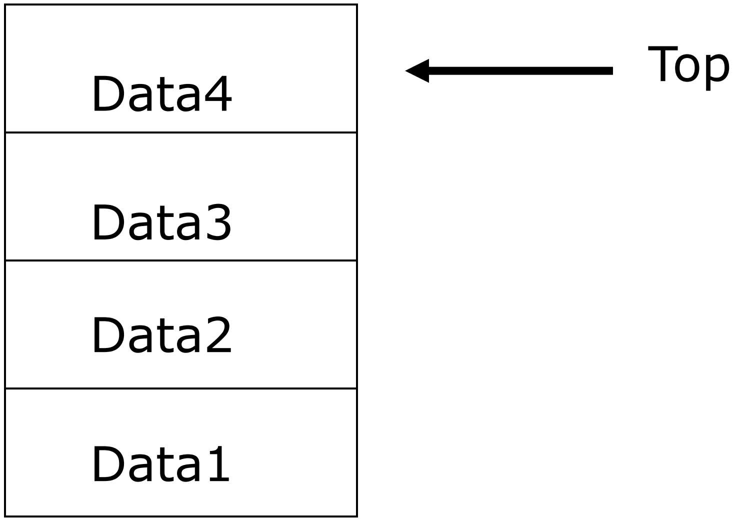
Stack:

- A group of elements
- The elements follow the rule of LIFO (last in, first out)

Operation	**Description**	**Input(s)**
push	Adds one element to the stack	item
pop	Removes (also returns) the last element that was added	
peek	Returns (without removal) the last element that was added	
size	Returns the size of the stack	
isEmpty	Return whether the stack is empty or not	

Stacks

- Collection with access only to the last element inserted
  - Last in first out
  - insert/push
  - remove/pop
  - top
  - make empty



Data4

Data3

Data2

Data1

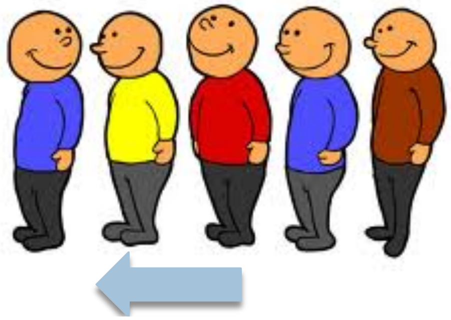
Top

ADT Example: Queue

Queue:

- A group of elements
- The elements follow the rule of FIFO (First in, first out)

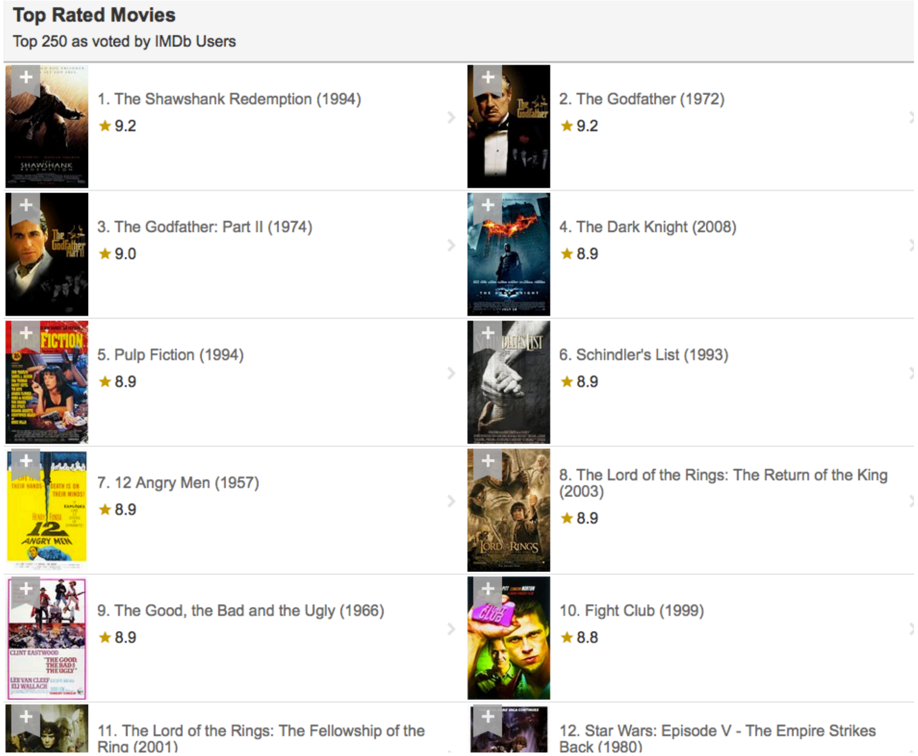
Operation	Description	Input(s)
enQueue	Adds one element to the queue	item
deQueue	Removes (also returns) the first element that was added	
peek	Returns (without removal) the first element that was added	
size	Returns the size of the queue	
isEmpty	Return whether the queue is empty or not	



ADT Example: Map/Dictionary

Map/Dictionary:

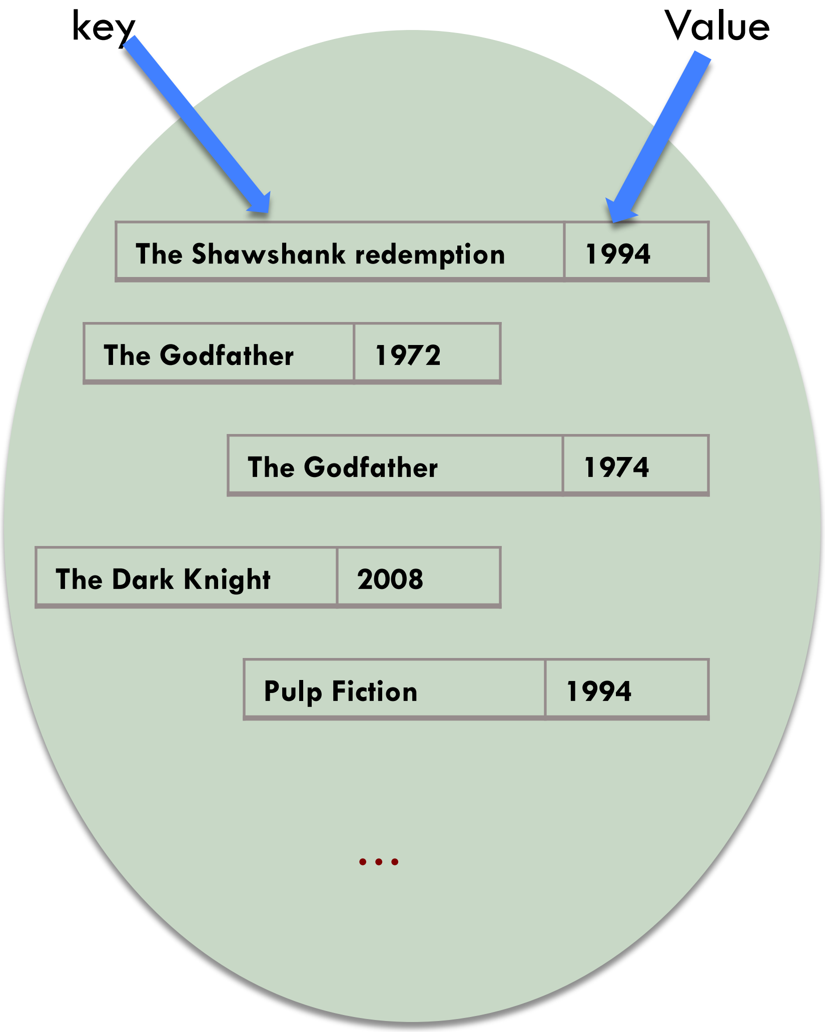
- A group of key and value **pairs**
- The key value must be **unique**



Top Rated Movies

Top 250 as voted by IMDb Users

1. The Shawshank Redemption (1994) ★9.2
2. The Godfather (1972) ★9.2
3. The Godfather: Part II (1974) ★9.0
4. The Dark Knight (2008) ★8.9
5. Pulp Fiction (1994) ★8.9
6. Schindler's List (1993) ★8.9
7. 12 Angry Men (1957) ★8.9
8. The Lord of the Rings: The Return of the King (2003) ★8.9
9. The Good, the Bad and the Ugly (1966) ★8.9
10. Fight Club (1999) ★8.8
11. The Lord of the Rings: The Fellowship of the Ring (2001)
12. Star Wars: Episode V - The Empire Strikes Back (1980)



key	Value
The Shawshank redemption	1994
The Godfather	1972
The Godfather	1974
The Dark Knight	2008
Pulp Fiction	1994

…

ADT Example: Map/Dictionary

Map/Dictionary:

- A group of key and value **pairs**
- The key value must be **unique**

Operation	Description	Input(s)
Add/insert	Adds a pair of key and value to the map	Key, value
Remove	Removes the pair with the given key	key
Lookup/get	Lookup the value associated with the given key, OR indicate the pair does not exist	Key
size	Returns the size of the map	
isEmpty	Return whether the map is empty or not	

Which ADT should be used

Problem	ADT
Words in a book	List
Course roster	List or Set
Google queries within one hour	List
Your TCNJ username and password	Map
Movie and its release date	Map
Facebook friends	Set
Top Baby names 2016	Set

Implementation: List

If use array to implement list

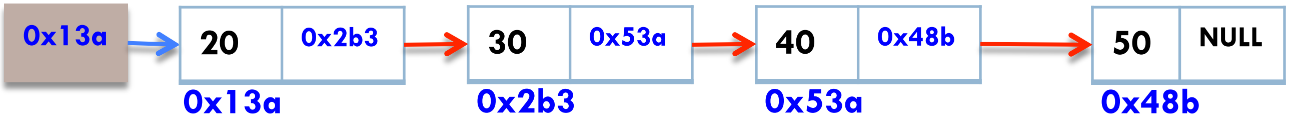
5	**20**	**23**	**2**	**6**					

- Must specify array size at creation
- Need a variable to contain the number of elements
- Insert, delete require moving elements
- Must copy array to a larger array when it gets full

When list gets full, create a new array of twice the size, copy values into it, and use the new array

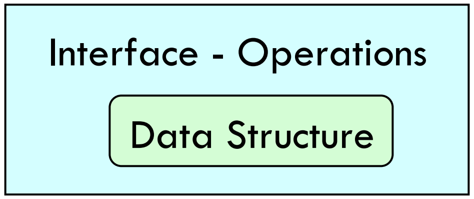
Implementation: List

If use Linked List to implement list

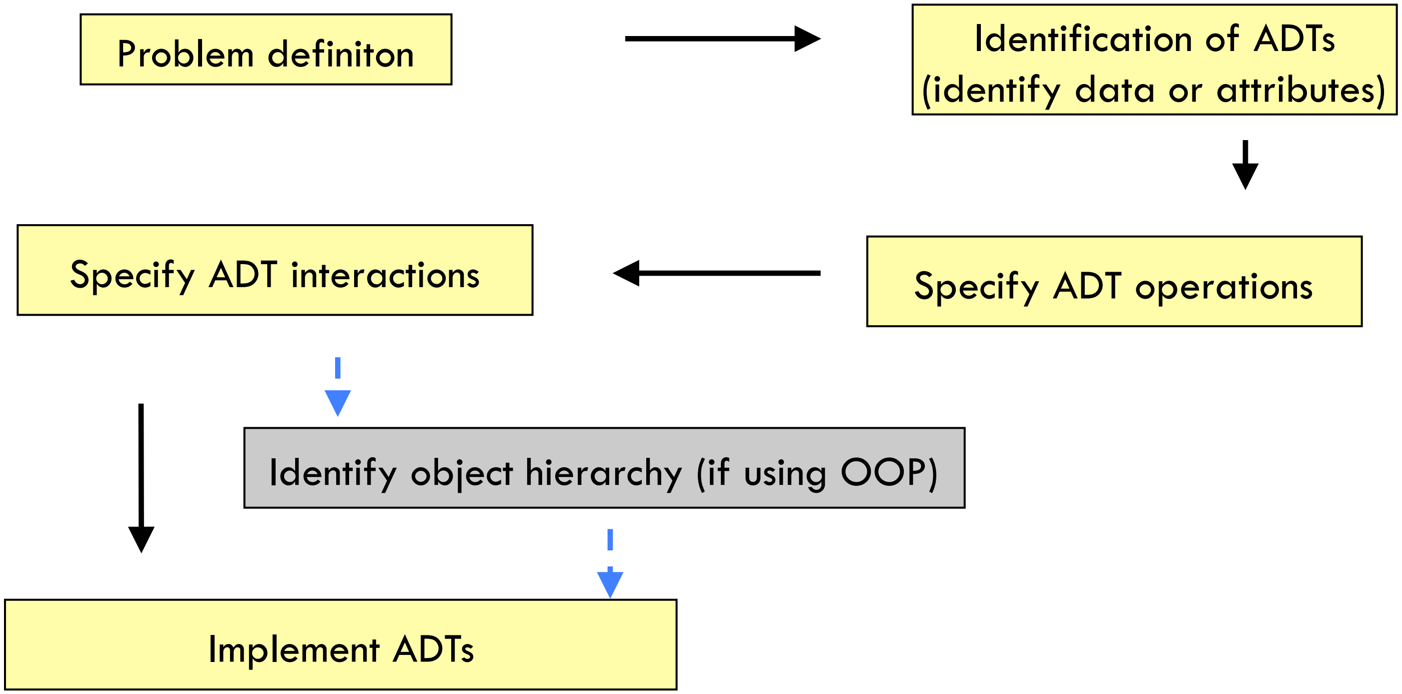


- Do not guess the data size
- Visit one given node needs traversal of multiple nodes
- Insert, delete does not require moving elements

Model for an ADT



System design with ADTs


Problem definiton
Identification of ADTs (identify data or attributes)
Specify ADT operations
Specify ADT interactions
Identify object hierarchy (if using OOP)
Implement ADTs

Complex Number ADT Example

- What is a complex number
  - a real part
  - an imaginary part e.g.: **2+4i**
- operations ?
  - create a complex number
  - add, subtract, multiply, divide
  - print a complex number
  - test to see if something is complex
  - etc.

Declare a complex number

- □ Interface:

```
Complex c1, c2, c3;
```

- □ Possible Implementation (using struct):

```
struct complex {
    double real;
    double imag;
};
typedef struct complex Complex;
```

Create a complex number

- Interface:

```
c1 = create_complex(2, 3);

/* conceptually, c1 = 2+3i */
```

Implementation

```
 Complex create_complex(
                        double real,
                       double imag)
{    Complex c;
      c.real = real;
      c.imag = imag;
      return c;
}
```

Add two complex numbers

- Interface:

```
c3 = add_complex(c1, c2);

/* conceptually, c3 = c1 + c2 */
```

Implementation

```
Complex add_complx(Complex c1,
                   Complex c2)
{  Complex csum;

   csum.real = c1.real + c2.real;
   csum.imag = c1.imag + c2.imag;

   return csum;
}
```

Using the Complex Number ADT

```
 #include <stdio.h>
 /* type implementation */
 struct complex {
  double real,imag;
typedef struct complex Complex;

 /* operation interface */
 Complex create_complex(double,double);
 Complex add_complex(Complex, Complex);
/* other Complex prototypes
 print_complex() . . .
*/
```

Using the Complex Number ADT

```
int main ( )
{ Complex c1, c2, c3;

  c1 = create_complex(2,-3);
  c2 = create_complex(2,-3);
  c3 = add_complex(c1,c2);

  print_complex(c3);

  return 0;
}


/*Implementation of Complex functions */
```

ADT vs Object-Oriented Programming (OOP)

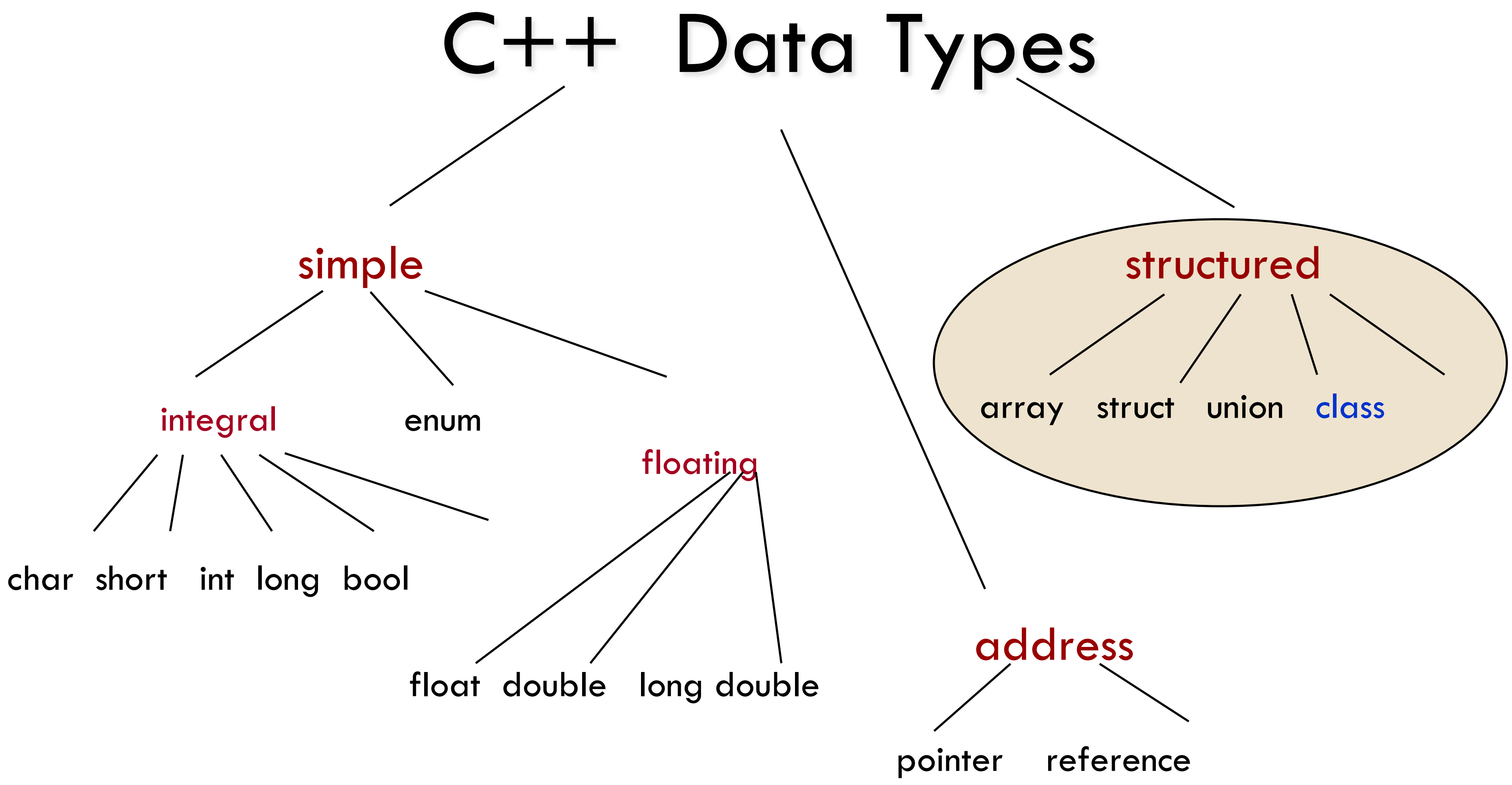
- ADTs are not a part of a particular programming language
- Rather they are implemented by a programmer to solve a particular problem or some class of problems

- In OOP, an ADT can be easily modeled as a class
  - An instance as an object
  - Data of ADT as properties or fields of a class
  - Operations as methods
- ADT ≠ OOP
- Classes in OOP offers more features than ADTs : Inheritance (Superclass-Subclass), Polymorphisms, etc.

C++ Classes

- Each class definition is placed in a header file
  - *Classname*.h
- The implementation of a class's member functions are placed in an implementation file
  - *Classname*.cpp

Data Types in C++


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