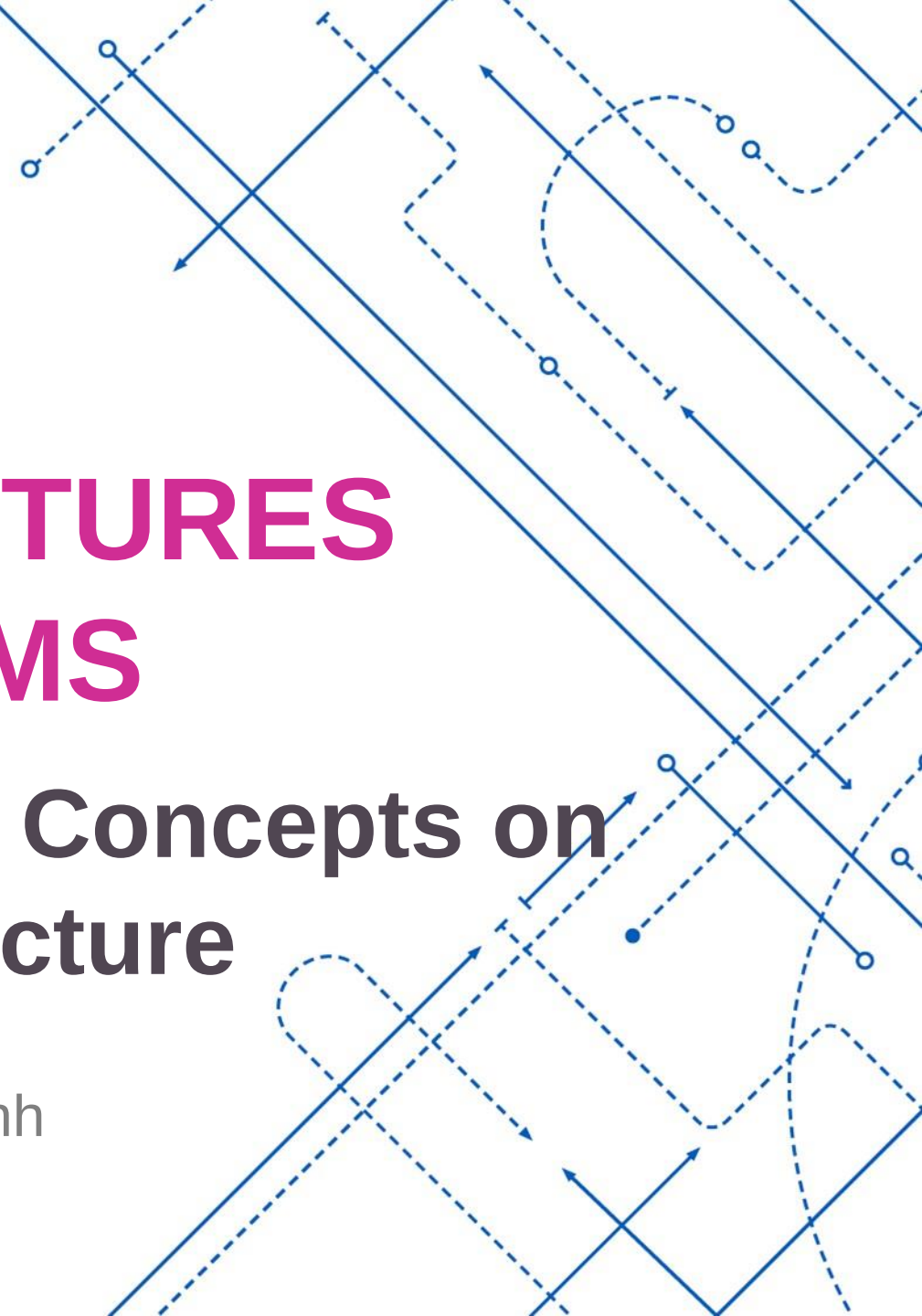
fit@hcmus

DATA STRUCTURES & ALGORITHMS

Lecture 4: Basic Concepts on ADT & Data Structure

Lecturer: Dr. Nguyen Hai Minh

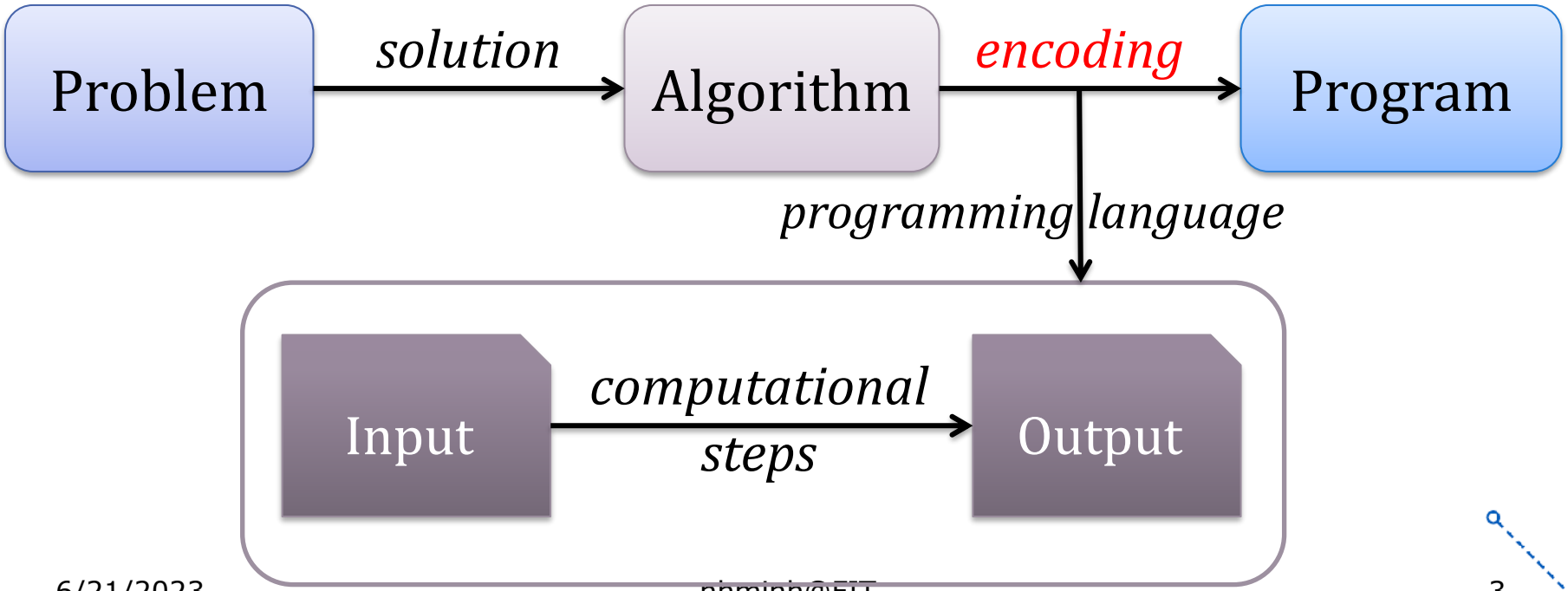


CONTENT

1. Data Type
2. Standard Data Type
3. Structured Data Type
4. Abstract Data Type
5. Data Structure
6. Data Structure Analysis

Introduction

- Programming = encoding an algorithm into a notation (programming language) → it can be executed by a computer



Introduction

- In order to provide a notational way to represent both the process and data, programming languages provide control ***constructs*** and ***data types***.
  - Constructs: for, if, else, while, …
  - Data types?

Data Type

- All data items in the computer are represented as strings of binary digits.
- In order to give these strings meaning, we need to have data types.
  - Provide an interpretation for this binary data
  - We can think about the data in terms that make sense
- Give an example of some data types that you know.
  - Briefly describe the characteristics of each data type.

Data Type

- Example:
  - Integer types: int, short, long, …
  - Character types: char, signed char, unsigned char
  - Real number types: float, double, …
  - Boolean type: bool
- Definition of "**Data Type**" :

$$T = <V, O>$$

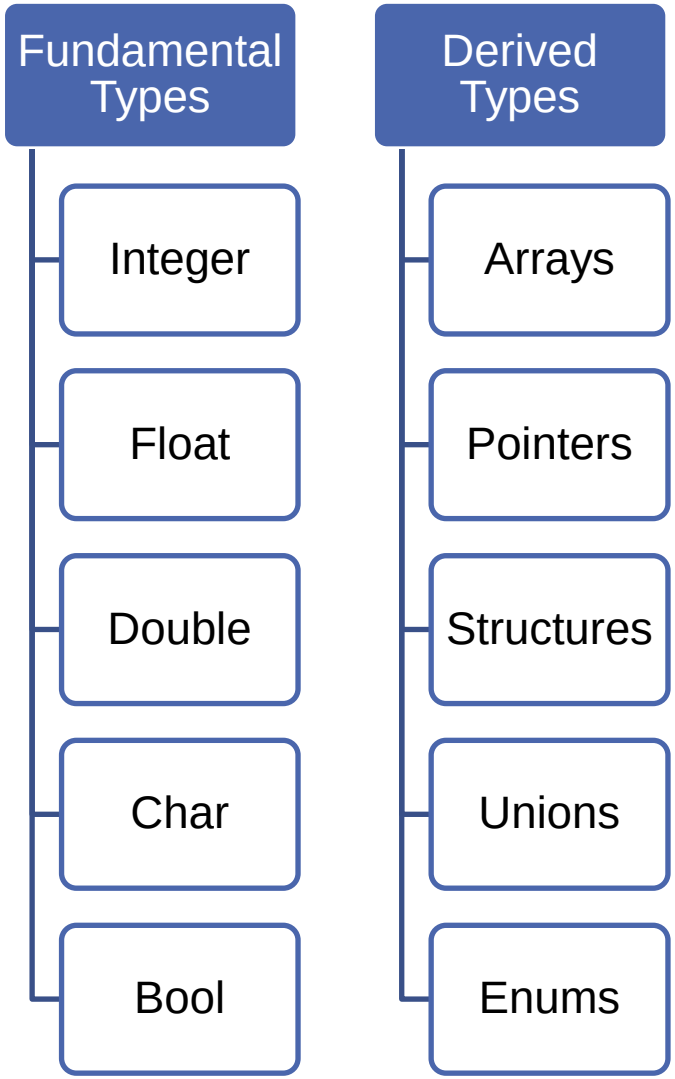
  - *V* (values – data range): collection of the values that *T* can handle.
  - *O* (operators): collection of basic operations that are operated in *V*

Data Type

- Example
  - T = short int (2 bytes)
    - V = {-32,768 .. +32,767}
    - O = {+, -, *, div, mod, >, >=, <, <=, ==, !=, <<, >>}
  - T = int (4 bytes)
    - V = {-2,147,483,648 .. 2,147,483,647}
    - O = {+, -, *, div, mod, >, >=, <, <=, ==, !=, <<, >>}
  - T = unsigned char (1 bytes)
    - V = {0 .. 255}
    - O = {+, -, *, div, mod, >, >=, <, <=, ==, !=, <<, >>}

Size of data types is compiler dependend

C/++ Data Types


Fundamental Types
Integer
Float
Double
Char
Bool
Derived Types
Arrays
Pointers
Structures
Unions
Enums

Fundamental Data Type

- A built-in-type for which the programming language provides built-in support.
- Also called primitive data type.
- List of fundamental data types:
  - Integer: short int, int, long
  - Logic: bool
  - Real number: float, double
  - Character: char

Fundamental Data Type in C/C++

Data type	Size	Values
bool	1 byte	?
char, unsigned char	1 byte	?
short, unsigned short	2 bytes	?
int, unsigned int	4 bytes	?
long, unsigned long	4 (8) bytes	?
long long, unsigned long long	8 bytes	?
float	4 bytes	?
double	8 bytes	?

Derived Data Type

- Programmers can build their own data type by combining standard data types to form a new structured data type/derived data type:
  - array
  - pointer
  - struct
  - enum
  - union
- Structured data types can contain any of the standard data types including pointers and other structs or arrays.

Derived Data Type

- **array:**
  - `int NumList[100];` // array including 100 integers. Size = ?
  - `char Name[30];` // array including 30 characters. Size = ?
- **struct:**
  -
```
struct DATE {
unsigned short int Year, Month, Day;
}; // Size = ?
```
  -
```
struct PERSON {
char CardID[9];
char Name[30];
struct DATE Birthday;
float Weight;
}; // Size = ?
```

Derived Data Type

- **enum:**

```
enum BOOLEAN
{
    Ffalse,
    Ttrue
};
enum BOOLEAN isCorrect = Ttrue;
enum WEEKDAYS                  // days of a week
{
     sunday,                   // sunday=0, monday=1, tuesday=2, …
     monday,
     tuesday,
     wednesday,
     thursday,
     friday,
     saturday
};
enum WEEKDAYS today = thursday;
```

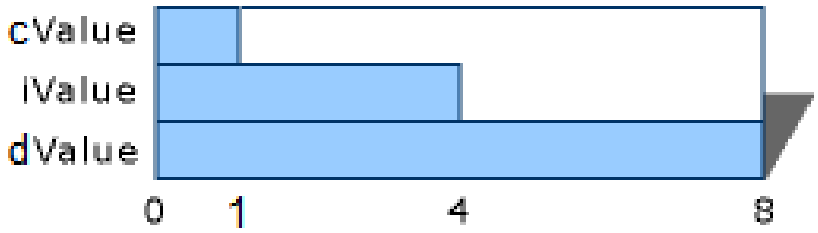
Derived Data Type

- **union:**

```
// using_a_union.cpp
#include <iostream>

union NumericType
{
    char            cValue;
    int             iValue;
    double          dValue;
};  // Size = 8 bytes

int main()
{
            union NumericType Values;
    Values.iValue = 1000;
            cout << Values.iValue << endl;
            Values.dValue = 3.1416;
            cout << Values.dValue << endl;
}
```



Abstract Data Type - ADT

- Solutions for some problems are very complex.
  - Difficult to use simple, language-provided constructs and data types to work through the problem-solving process.
  - Need ways to control the complexity and assist with the creation of solutions.

→ *ABSTRACTIONS*

Focus on the "**big picture**" without getting lost in the details.

Procedural Abstraction

- Procedural (Functional) Abstraction:
  - Allows us to view the problem and solution in such a way as to separate the so-called **logical** and **physical perspectives**.
  - It is essential to team projects. When working in team, you have to use modules written by others, frequently without knowledge of their algorithms.

Procedural Abstraction – Example

- Users:
  - make/receive calls
  - take photos
  - send messages
  - surf the internet
  - check email, facebook
  - play music …

→ Logical perspective

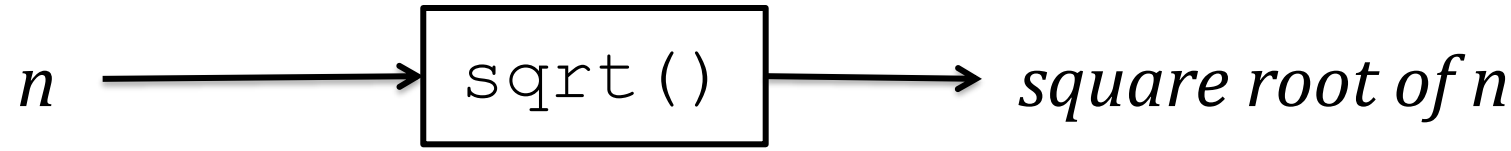
- Programmers/designer:
  - how the phone works,
  - how the message sent,
  - how operating system works
  - how to code apps

→ Physical perspective

User interface

Procedural Abstraction – Example

```
#include <cmath>
double x = sqrt(16);
```



- We do not necessarily know how the square root is being calculated, but we know how to use it:
  - name of the function
  - what is needed (parameters)
  - what will be returned

→ The details are hidden inside (black box)

interface

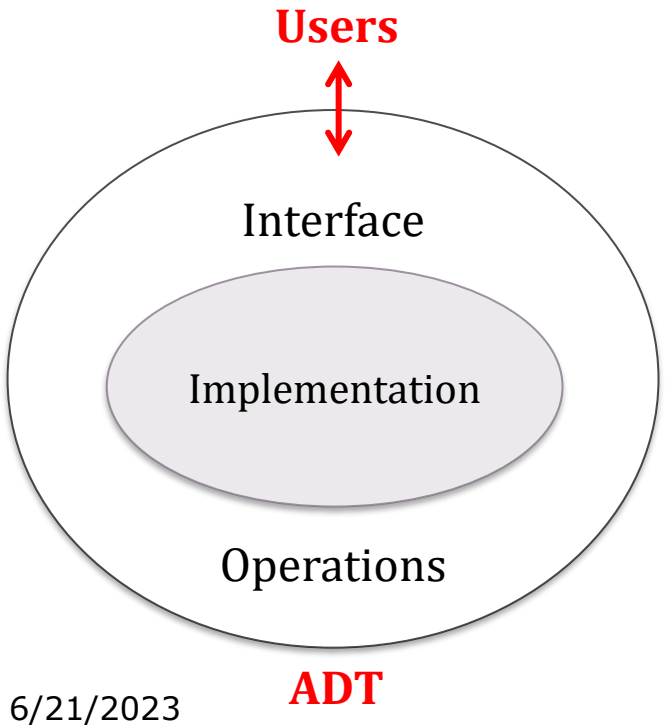
Data Abstraction

- Consider now a ***collection of data*** and a set of ***operations*** on the data.
  - The operations might include ones that add new data to the collection, remove data from the collection, or search for some data.
- **Data Abstraction** focuses on what the operations do with the collection of data, instead of how you implement them.

Abstract Data Type – ADT

Definition:

- A logical description of how we view the data and the operations that are allowed without regard to how they will be implemented.



Explain:

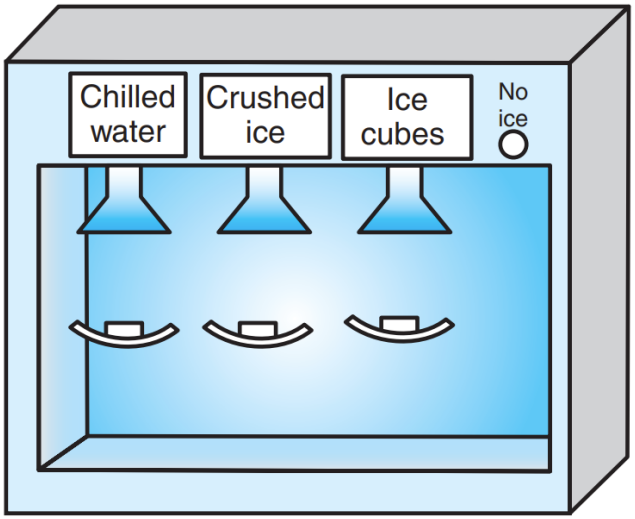
- Concern only with the data is representing, not how it is constructed.
- Creating an encapsulation around the data → hiding them from the user's view.

Abstract Data Type – ADT

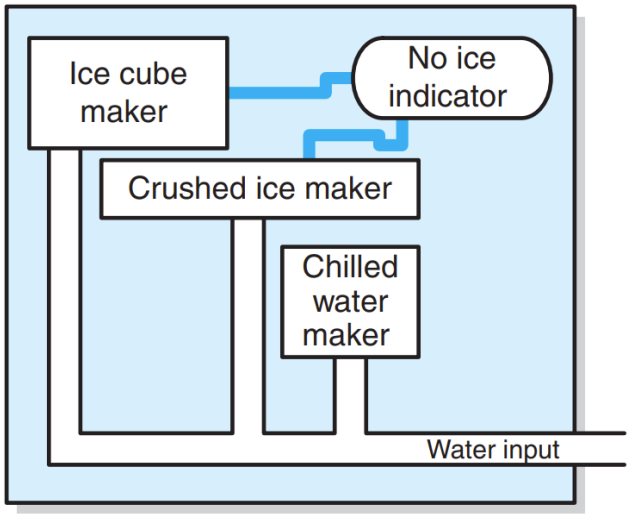
- Someone (perhaps you) will implement the ADT by using a **data structure**, which is a construct that you can define to store a collection of data.
- Example:
  - You need an ADT to store a collection of names in a manner that allows you to search rapidly for a given name.
    - The definition of your ADT should not specify whether to store the data in consecutive memory locations or in disjoint memory locations

ADTs vs Data Structures

- An **ADT** is a specification for a group of values and the operations on those values.
- A **Data Structure** is an implementation of an ADT within a programming language.
- Example: A dispenser of chilled water, crushed ice, and ice cubes.



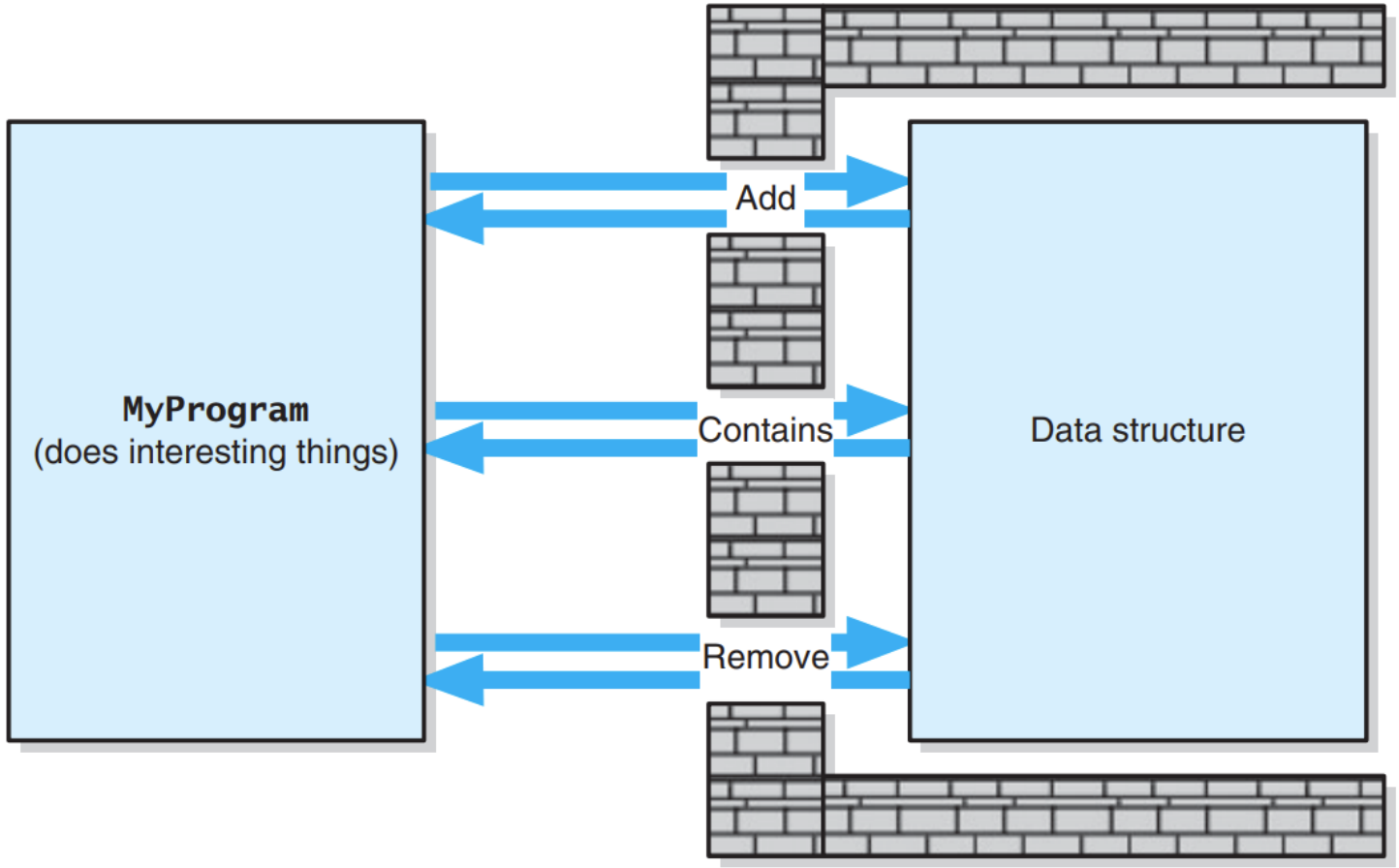
User view from specifications



Technician view

ADTs vs Data Structures

- A wall of ADT operations isolates a data structure from the program that uses it



Abstract Data Type – ADT

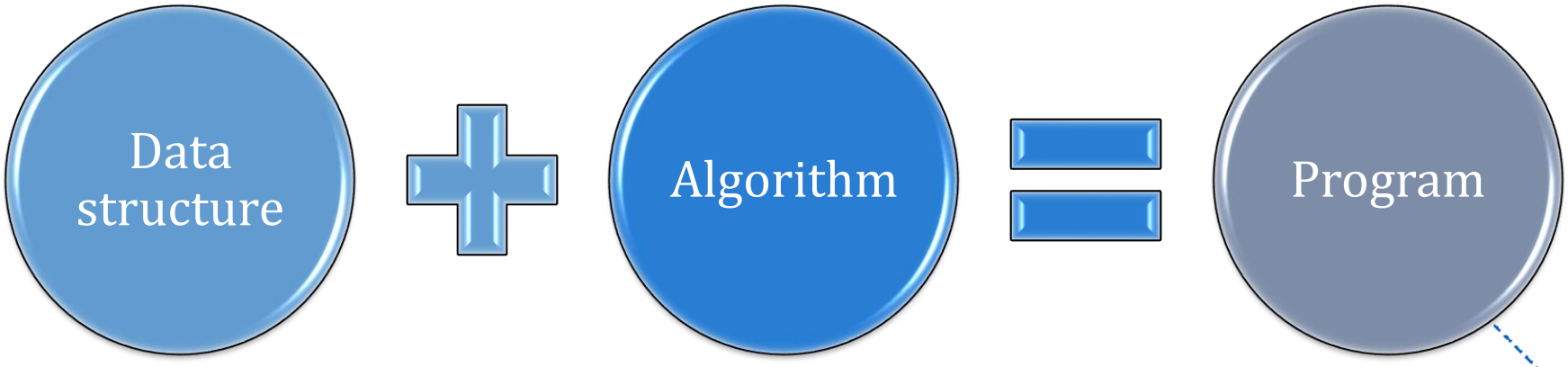
- Example:
  - Stack ADT:
    - Data
    - Operations: push, pop, peek
    - Implement: using array or linked list
  - Queue ADT:
    - Data
    - Operations: enqueue, dequeue, front
    - Implement: array/linked list

Data Structure

- Data structure is the implementation of the abstract data type.
  - → Provides an implementation-independent view of data.
  - → The user of data remain focused on how to interacts with it.
- Example:
  - Array
  - Linked list
  - Tree
  - Hash Table
  - Heap
  - Graph
  - …

Data Structure

- Each data structure is suitable for a specific problem.
- Example:
  - B-Tree: database
  - Hash Table: searching, dictionary
  - Queue: finding shortest path



Data Structure Analysis

- A data structure is called suitable for an application (*A*) if it satisfies the following criteria:
  1. Storing the data of *A* correctly and completely.
  2. Easy to access and manipulate.
  3. Saving the memory.

Data Structure Analysis

- Completeness and correctness:
  - **Ex1. data of GPA**

```
int GPA;
char GPA;
float GPA;
```

  - **Ex2. data of day [1-31]**

```
int Day,
short int Day;
unsigned short int Day;
float Day;
```

  - **Ex3. data of year [0-2015]**

```
unsigned char Year;
unsigned int Year;
unsigned short int Year;
```

Data Structure Analysis

- Easy to access:
  - Example: data of date of birth

```
char DOB[8];    // ddmmyyyy
char DOB[8];    // yyyymmdd
struct DATE DOB;
```

- Saving memory
  - Example

What's next?

- After today:
  - Read textbook 2 – Chapter 1 (page 28~)


Q&A