CSCI 2520
Data Structures and Applications

Course Instructor

Laiwan Chan 陳麗雲 Lauren

- Room 1016, Ho Sin Hang Engineering Building
- Tel : 3943 8434
- Email : lwchan AT cse.cuhk.edu.hk
- http://www.cse.cuhk.edu.hk/~lwchan

- Lectures :
  - Mon 2:30 pm - 4:15 pm
    Leung Kau Kui Bldg 101
  - Thu 10:30 am - 11:15 am
    William M W Mong Eng Bldg 407

- Tutorial :
  - Thu 5:30 pm - 6:15 pm
    William M W Mong Eng Bldg 404

tutors

- ZHANG Kai (kzhang AT cse.cuhk.edu.hk)
- ZHENG Chenguang (cgzheng AT cse.cuhk.edu.hk)

Course Information

- blackboard : https://elearn.cuhk.edu.hk/

CSCI 2520
Data Structures and Applications

- This course formally examines the relationship between abstract data types and data structures. The implementation of abstract data types using various data structures will be discussed. Abstract data types including list, stack, queue, symbol table, tree and graph will be introduced. Introductory complexity analysis and big-O notation will be illustrated with simple algorithms such as searching and sorting.

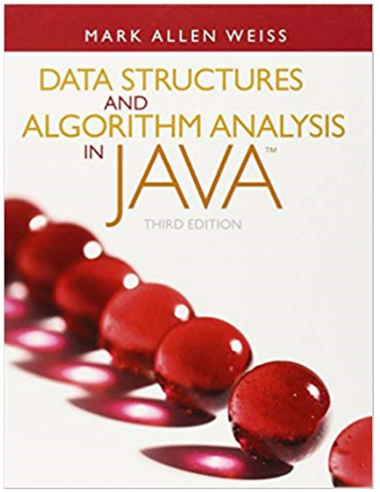
Prerequisite

- CSCI1110 or 1120 or 1130 or 1510 or 1520 or 1530 or 1540 or ENGG1110 or ESTR1002 or 1100 or 1102 or (MATH2210 and 2220) or PHYS2351 or its equivalent.

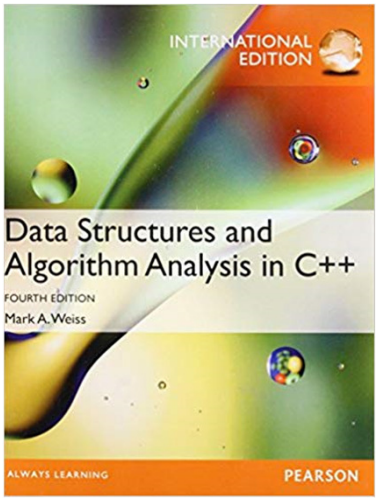
- CSCI1110 Introduction to Computing Using C
- CSCI1120 Introduction to Computing Using C++
- CSCI1130 Introduction to Computing Using Java
- CSCI1510 Computer Principles and C Programming
- CSCI1520 Computer Principles and C++ Programming
- CSCI1530 Computer Principles and Java Programming
- CSCI1540 Fundamental Computing with C++
- ENGG1110 Problem Solving By Programming
- ESTR1002 Problem Solving By Programming
- ESTR1100 Introduction to Computing Using C++
- ESTR1102 Introduction to Computing Using Java
- MATH2210 Mathematics Laboratory I
- MATH2220 Mathematics Laboratory II
- PHYS2351 Basic Computational Physics
- or its equivalent.

textbook

- **Data Structures and Algorithm Analysis in Java,** Mark Allen Weiss, third edition, Pearson, ISBN-10: 0132576279



- **Data Structures and Algorithm Analysis in C++,** Mark Allen Weiss, ISBN-10: 0273769383



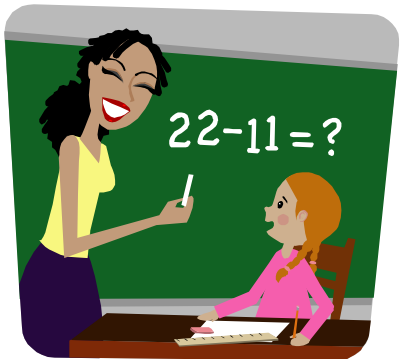
Tentative schedule

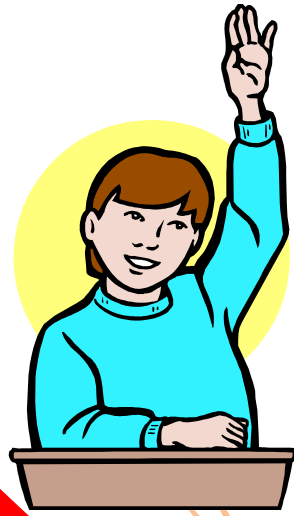
Week	Topics	
1	Abstract Data Types	
2	Stack and queue	
3	Hashing	
4	List and recursion	Assignment 1
5, 6	Sorting and complexity	
7, 8, 9	Trees and searching	Assignment 2
10, 11, 12	Graphs	Assignment 3
13, 14	Heap	Assignment 4

Assessment

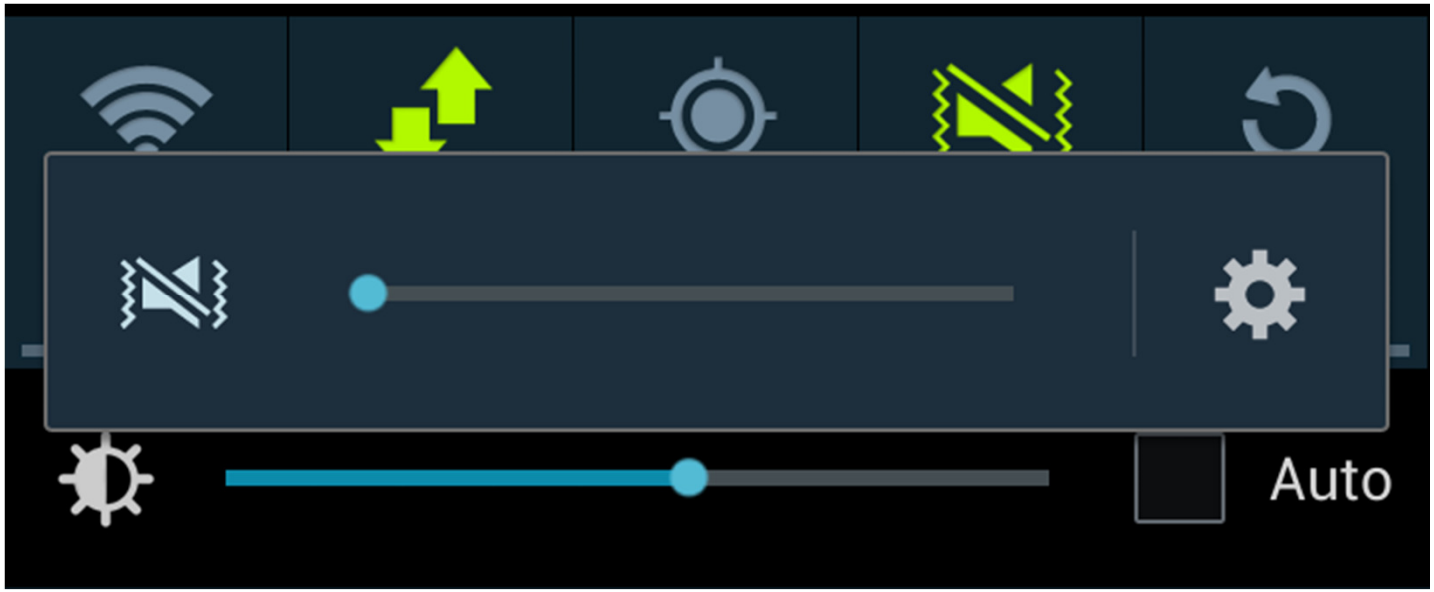
- 50 % Final Exam (closed book) and with a min passing score of 40/100
- 50 % Assignments and quizzes

CLASSROOM MANAGEMENT





PLEASE TURN OFF YOUR PHONES/BEEPERS


Auto

PLEASE DO NOT RECORD

Honesty in Academic Works

•Attention is drawn to University policy and regulations on honesty in academic work, and to the disciplinary guidelines and procedures applicable to breaches of such policy and regulations. Details may be found at http://www.cuhk.edu.hk/policy/academichonesty/ .

•With each assignment, students will be required to submit a signed declaration that they are aware of these policies, regulations, guidelines and procedures. For group projects, all students of the same group should be asked to sign on the declaration.

•For assignments in the form of a computer-generated document that is principally text-based and submitted via VeriGuide, the statement, in the form of a receipt, will be issued by the system upon students' uploading of the soft copy of the assignment. Assignments without the receipt will not be graded by teachers. Only the final version of the assignment should be submitted via VeriGuide.

CSCI 2520
Data Structures and Applications

- This course formally examines the relationship between abstract data types and data structures. The implementation of abstract data types using various data structures will be discussed. Abstract data types including list, stack, queue, symbol table, tree and graph will be introduced. Introductory complexity analysis and big-O notation will be illustrated with simple algorithms such as searching and sorting.

Algorithm

- An **algorithm** is a clearly specified set of simple instructions to be followed to solve a problem.
- **Algorithm analysis**
  - To determine how much resources (time or space) is required.

Example 1

- Given *n* numbers in ascending order, find whether the number *x* is in the sequence.
- Find if x = 10 is in 2, 3 ,5, 7, 10, 22, 30
- **Solution 1** : go through the numbers one after another one.
- **Solution 2** : compare 10 with 7 (the middle one)

2, 3 ,5, 7, 10, 22, 30

~~2, 3 ,5, 7,~~ 10, 22, 30

compare 10 with 22 (the middle one)

10, 22, 30

- 2, 3 ,5, 7, 10, 22, 30
- Solution 1 : 5 comparisons
- Solution 2 : 3 comparisons

- The number of comparisons depends on
  - *n*, the length of the sequence
  - the location of the target *x*

Complexity analysis

- A measure of the efficiency of the algorithm
  - Best case
    - not informative, only if you are lucky
  - Average case
    - give you the idea of the average case
    - may be difficult to analyse
  - Worst case
    - guarantee the performance

Example 2

- Given : N numbers 16, 23, 4, 53, 29, 17, ….
- Problem : Find the k^{th} largest

- **Solution 1** :
  - read the data into an array
  - Sort the array in decreasing order
  - Return the element in position k

Example 2

- Given : N numbers 16, 23, 4, 53, 29, 17, ….
- Problem : Find the k^{th} largest

- **Solution 2** :
  - read the first k numbers into an array
  - Sort the array in decreasing order
  - Get a new number
    - Ignore it if it is smaller than the kth one
    - Otherwise, place it in the correct spot in the array and bump the kth one out of the array
  - Return the number in position k

Role of data structure

- **Data structure** is a data organization, management, and storage format that enables efficient access and modification.
- It supports various operations with the data
  - Example :
    - insertion
    - deletion
    - breaking the sequence into half

      2, 3 ,5, 7, 10, 22, 30 → 10, 22, 30

What is a Data Type?

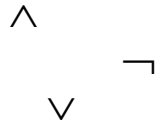
- A data type is
  - a set of ***values***, associated with
  - a set of ***operations*** applied on these values.

- Integers, reals, and booleans are data types.
- They have operations associated with them,

- Example: **integer**
  - ***Values***: 1, 3, 6, 99, -67, -245, 0, …
  - ***Operations***: addition, subtraction, multiplication, division, modulo, …

Data Type: Examples

- Example: **Boolean**
  - ***Values***: true, false
  - ***Operations***: conjunction, disjunction, negation



- Example: **string**
  - ***Values***: “hello”, “csci2520”, “123”, “-6”, “”, …
  - ***Operations***: string copy, string concatenation, string length, …

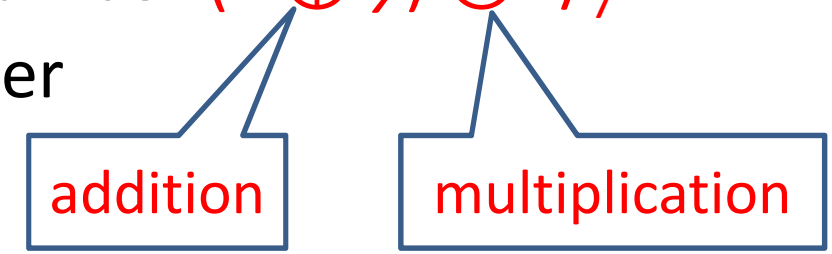
Data Type

- A programming language directly supports only *some* data types.
  - E.g., C directly supports int, long, float, double, char, …
- When a data type is not directly supported, it is not trivial to use it.

Example: Rational Number

- A rational number is a number that can be expressed as a fraction p / q,
  - where p and q are integers and $q \neq 0$.

- Suppose you need to write a program that:
  - reads two rational numbers x and y
  - computes the rational number $(x \oplus y) \otimes {}^{6}/_{7}$
  - prints the rational number



Rational Number in C

- There is no such data type “rational number” in C.
- In this case, we need an ***abstract data type*** (ADT) in order to express “rational number” and support the operations on “rational number”.

$(x \oplus y) \otimes {}^{6}/_{7}$

- rationalADT x, y, temp, ans // define the variables as rational numbers
- x = read_rational() // read x
- y = read_rational() // read y
- temp = create_rational(6,7)
- ans = add_rational(x,y)
- ans = mult_rational(ans, temp)
- print_rational(ans)

mult_rational(x,y)

- $x = p_x / q_x$
- $y = p_x / q_y$

Answer = create_rational($p_x p_y$, $q_x q_y$)

Return(Answer)

add_rational(x,y)

- $x = p_x / q_x$
- $y = p_y / q_y$

Answer = create_rational($p_x q_y + p_y q_x$, $q_x q_y$)

Return(Answer)

Using an ADT: calculate.c

```
#include "rational.h"

int main() {
   rationalADT x, y, temp, ans;
   x = read_rational();   // read rational number x
   y = read_rational();   // read rational number y
   temp = creat_rational(6, 7);   // 6/7
   ans = add_rational(x, y);   // x ⊕ y
   ans = mult_rational(ans,temp); // (x ⊕ y) ⊗ 6/7
   print_rational(ans);
   return 0;
}
```

ADT Interface: rational.h

```
typedef struct rationalCDT *rationalADT;

rationalADT read_rational();
rationalADT add_rational(rationalADT x, rationalADT y);   // ⊕
rationalADT minus_rational(rationalADT x, rationalADT y); // ⊖
rationalADT mult_rational(rationalADT x, rationalADT y);  // ⊗
rationalADT div_rational(rationalADT x, rationalADT y);   // ⊘
rationalADT create_rational(int p, int q);   // p/q
void print_rational(rationalADT x);

rationalADT simplify_rational(rationalADT x);
```

Abstract data type (ADT)
Usage vs Implementation

- An ADT is implemented in an ***abstract*** way. Users do ***not*** (and should not) need to know how it is implemented.

- Usage of an ADT should be ***independent*** of its implementation.

Actual Implementation of an ADT

```
typedef struct rationalCDT *rationalADT;
```

- There can be different ways to implement rationalADT, as long as the implementations agree to the specifications in rational.h.

- Let's consider two implementations.
  - Implementation 1: using structure
  - Implementation 2: using array

rationalADT: Implementation 1

rational.c (Implementation 1)

```
#include "rational.h"

struct rationalCDT {    // Implementation 1
   int p, q;    // rational number = p/q
};

rationalADT create_rational(int a, int b) {
   rationalADT x;
   x = (rationalADT)malloc(sizeof(*x));
   x->p = a;
   x->q = b;
   return x;
}
```

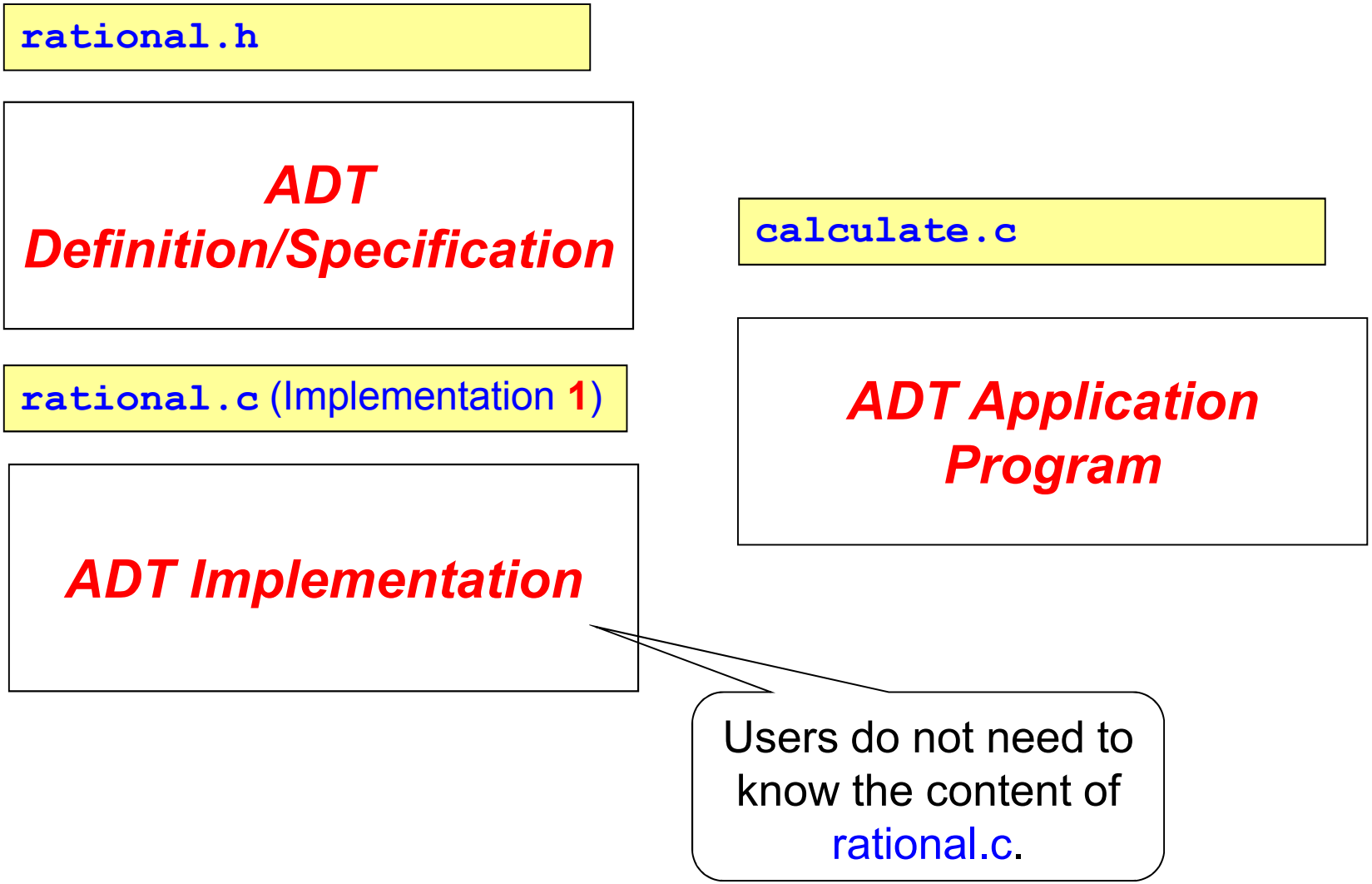
rationalADT: Implementation 1

rational.c (Implementation 1)

```
rationalADT add_rational(rationalADT x, rationalADT y)
{
   rationalADT z;
   int a, b;

   a = x->p * y->q + x->q * y->p;
   b = x->q * y->q;

   z = create_rational(a, b);
   z = simplify_rational(z);

   return z;
}

…… // similarly for r_minus(), r_mult(), …
```

$p_x q_y + p_y q_x \,/\, q_x q_y$

The Whole Program

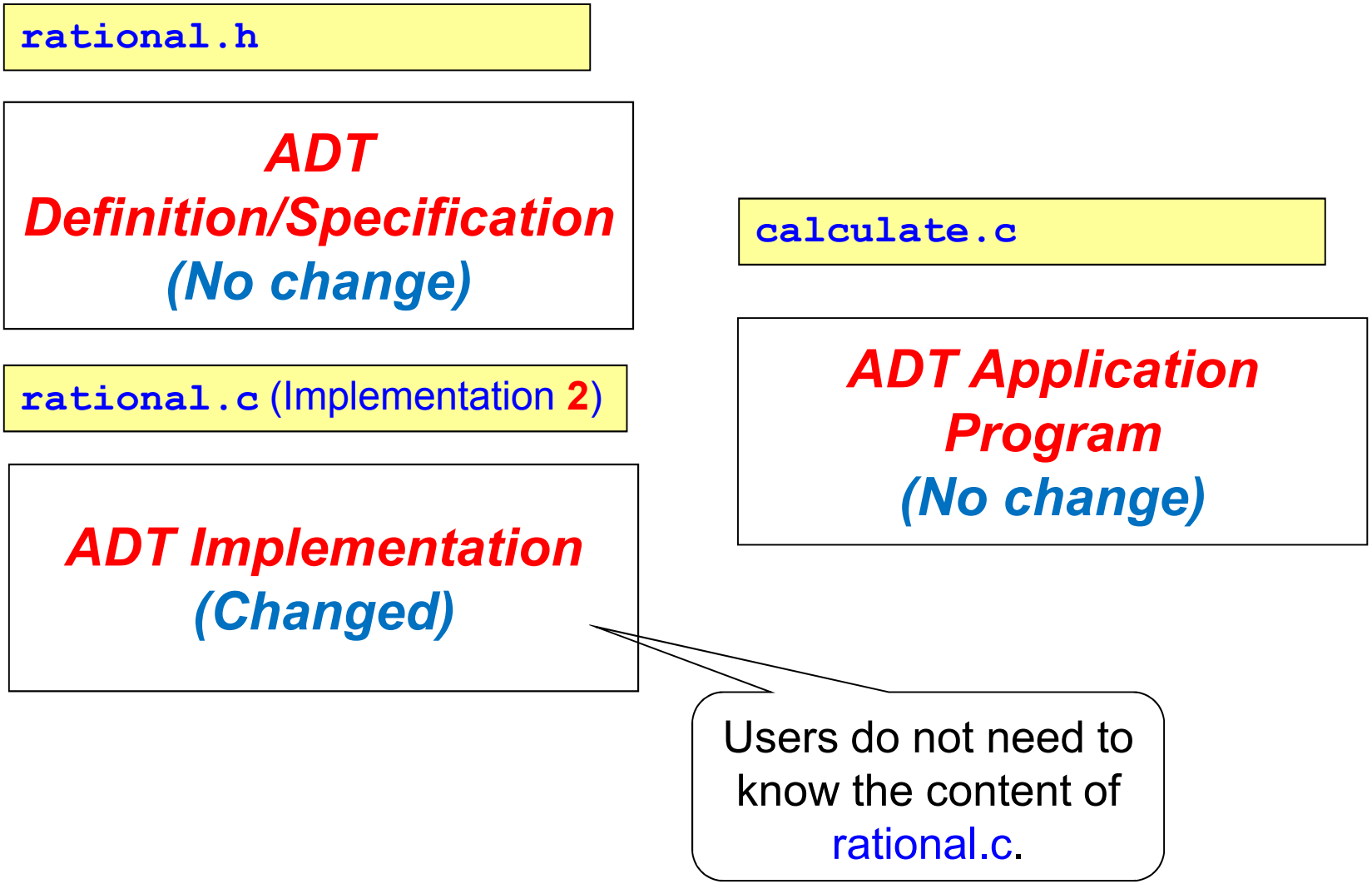


rationalADT: Implementation 2

rational.c (Implementation 2)

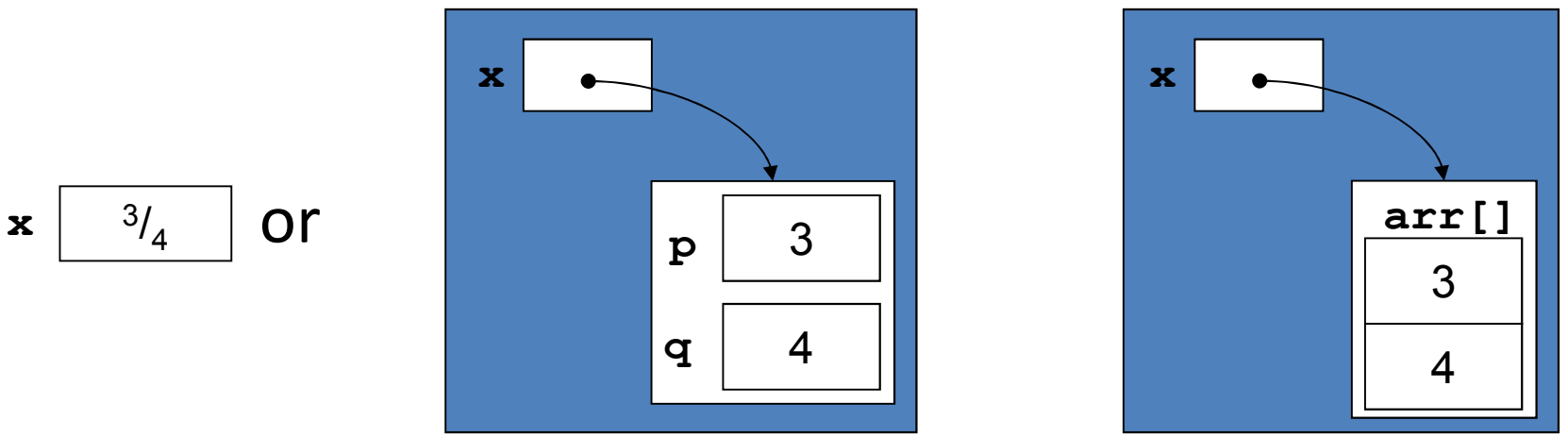
```
#include "rational.h"

struct rationalCDT {    // Implementation 2
   int arr[2];    // rational number = arr[0]/arr[1]
};

rationalADT create_rational(int a, int b) {
   rationalADT x;
   x = (rationalADT)malloc(sizeof(*x));
   x->arr[0] = a;
   x->arr[1] = b;
   return x;
}

.....    // other functions
```

The Whole Program



ADT: Usage, Interface, and Implementation

- ***Hiding*** the implementation of an ADT from the user means there are fewer details for the user to understand.
  - As an ADT *user*, what do you prefer?



ADT: Usage, Interface, and Implementation

- A programmer who implements an ADT is ***flexible*** to change its underlying representation.
  - It is always possible to have Implementation 3, Implementation 4, Implementation 5, etc. for rationalADT, as long as the interface remains the same.
  - Any change of representation does not affect the user programs.