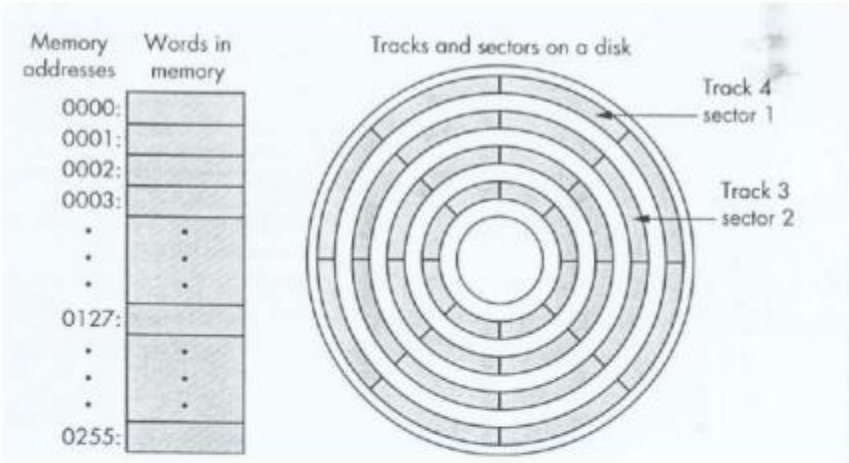
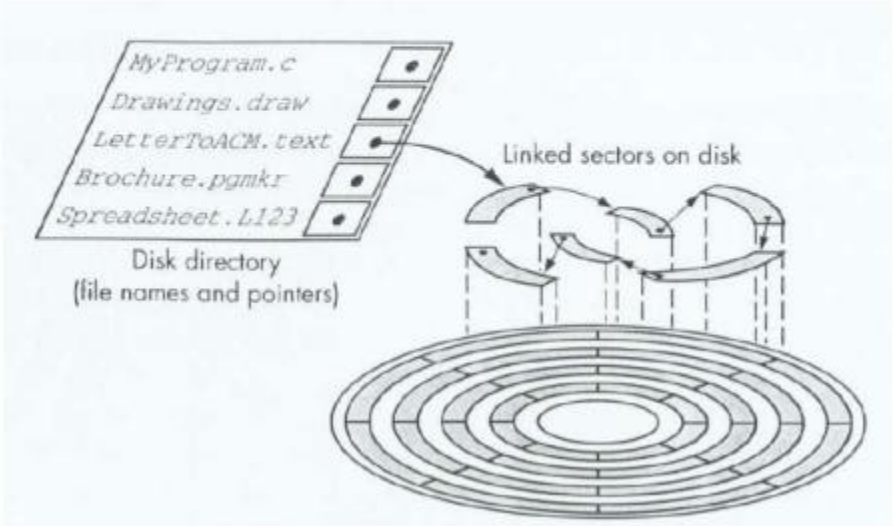
Fakultas Informatika
School of Computing
Telkom University

CDK2AAB4 - Struktur Data
Pointer dan Abstract Data Type (ADT)

Pointer

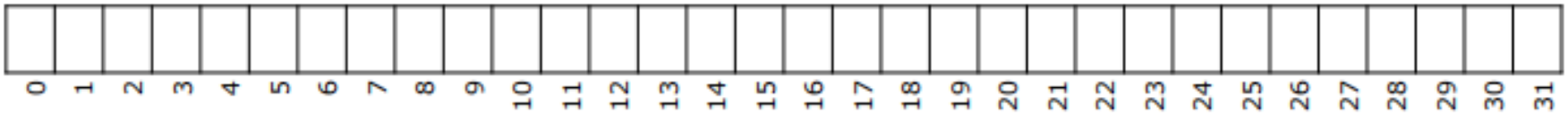
Representation of a Media Storage





Data and Memory

- Data of a variable is stored in memory
- Picture it as a 1-dimension array
- Each cell has a unique “index”, we call it **address**



Data and Memory

- While program runs, OS will allocate the memory space for each variable

Dictionary

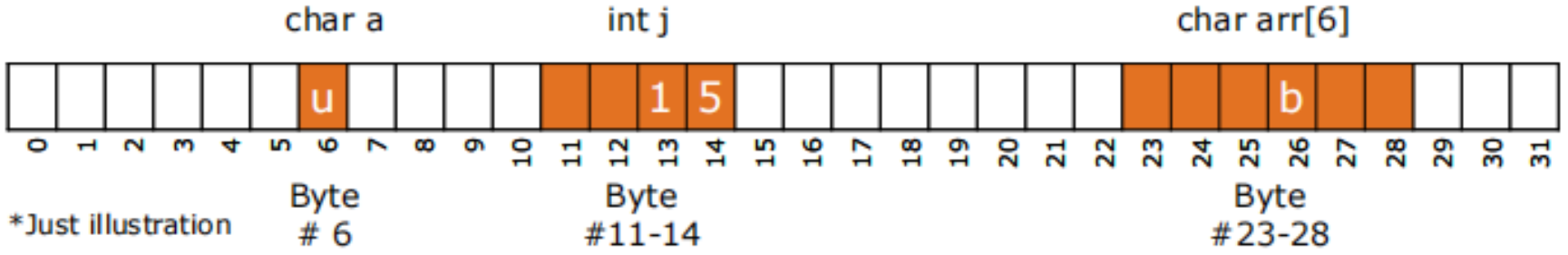
a : char
j : integer
arr : array [1..6] of char

Data and Memory

- We can call or change the value of a variable by calling the address where it's stored

Algorithm

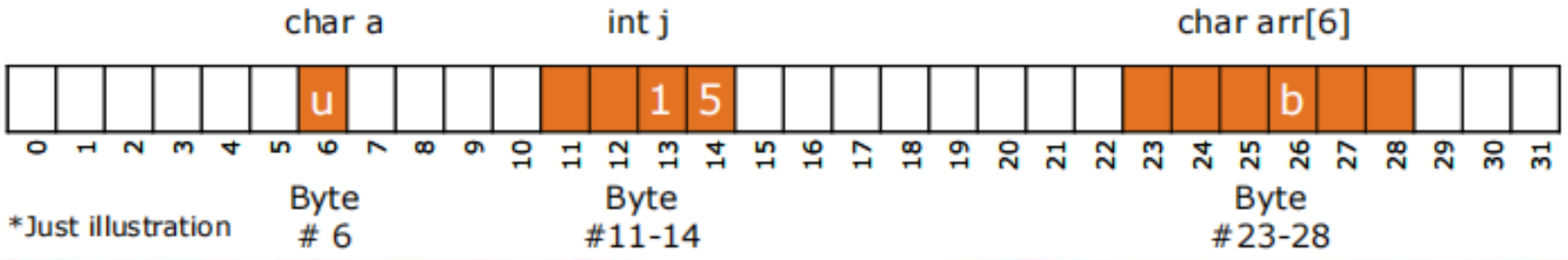
arr[3] ← 'b'
a ← 'u'
j ← 15



Data and Memory

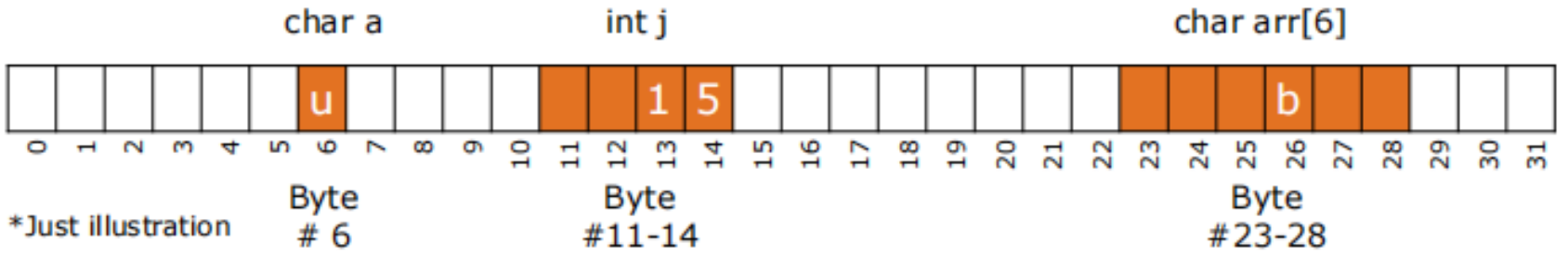
- Specific for C/Cpp-family programming language, we can access the address of a variable using keyword '&'

Algorithm	Output
output(a)	u
output(&a)	0x6
output(&arr[3])	0x26



Pointer

- Basic variable type
- Store an address of a variable in hexadecimal
- Size of an integer (4byte)



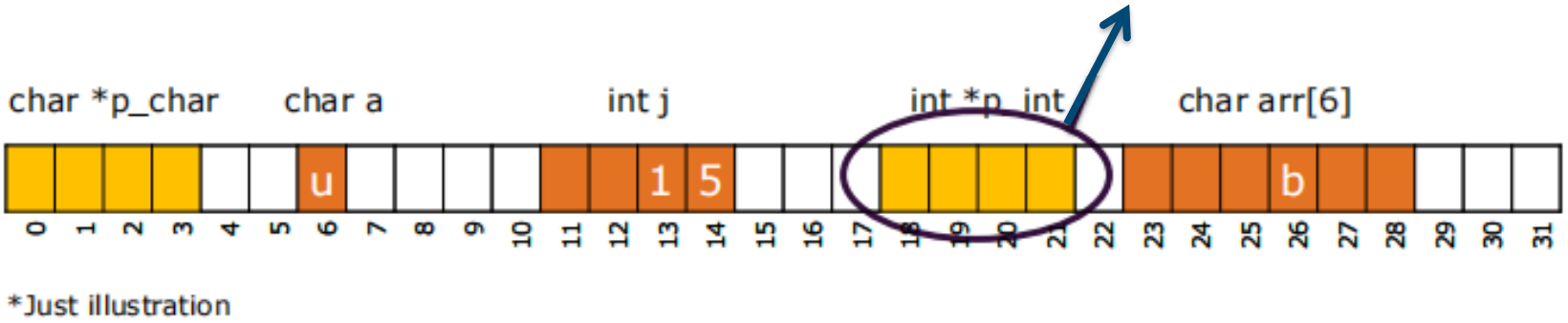
Pointer

- Pointer also has a variable type
- Can only points to variables of the same type

Dictionary

a : char
j : integer
arr : array [1..6] of char

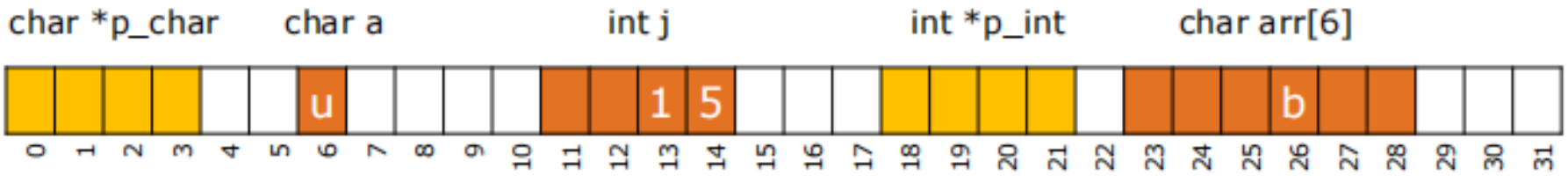
Pointer also use
Space memory



*Just illustration

Pointer (in pseudocode)

- For a pointer to refer onto a variable, just assign the variable into pointer
- Use keyword * to assign the value of a variable pointed by pointer



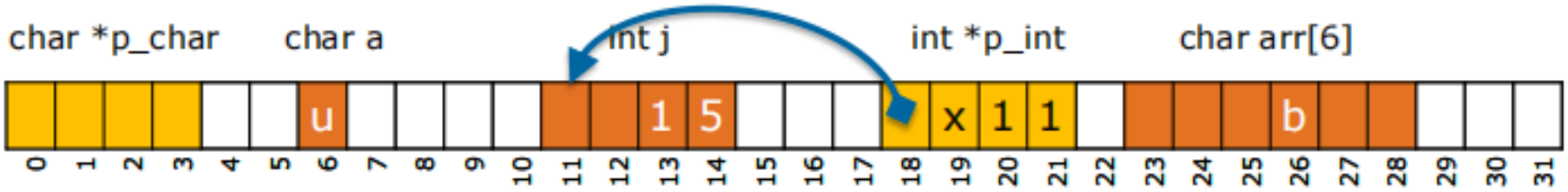
Operation using Pointer

Algorithm

```
p_int ← &j
output(j)
output(p_int)
output(*p_int)
```

Output

```
15
0x11
15
```



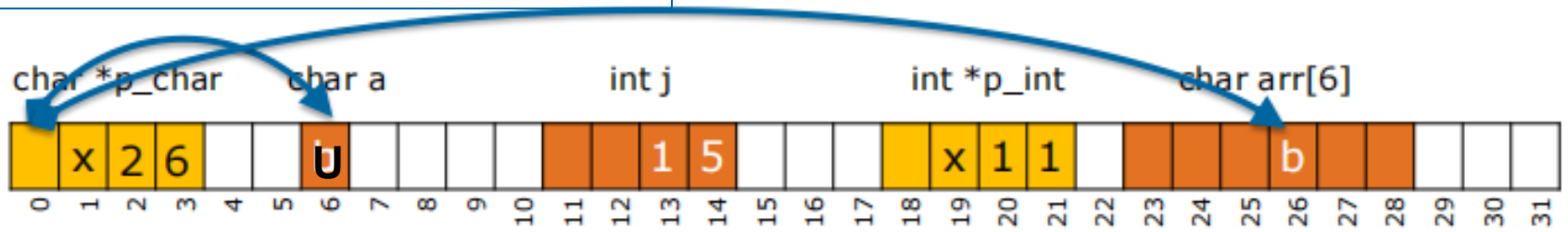
Operation using Pointer

Algorithm

```
p_char ← &a
output( *p_char )
p_char ← &arr[3]
output( *p_char )
a ← *p_char
output( a )
```

Output

‘U’ // pointing to a
‘b’ // pointing to arr[6]
‘b’



Data and Memory

- On Algorithm, pointer is about the value of the variable pointed
- Here we don’t talk about how to manually set a pointer to refer some address
- Program wise, it’s also not good to manually set a pointer into some memory address

Don’t be confused

Dictionary

a, b : char

p1, p2 : pointer to char

Algorithm

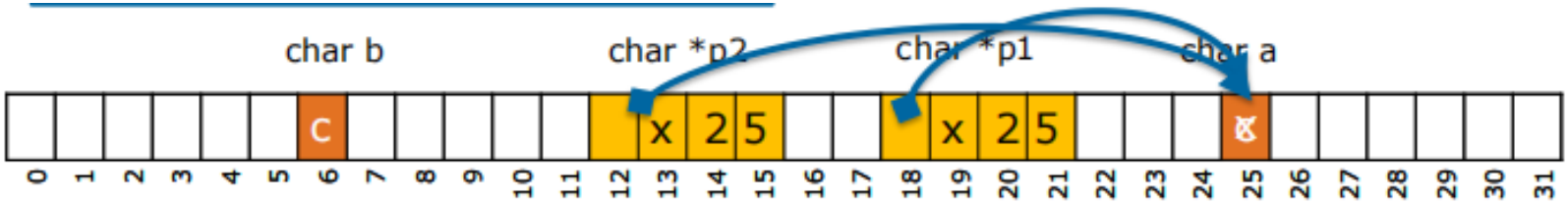
a ← ‘c’

p1 ← &a

p2 ← p1

b ← *p1

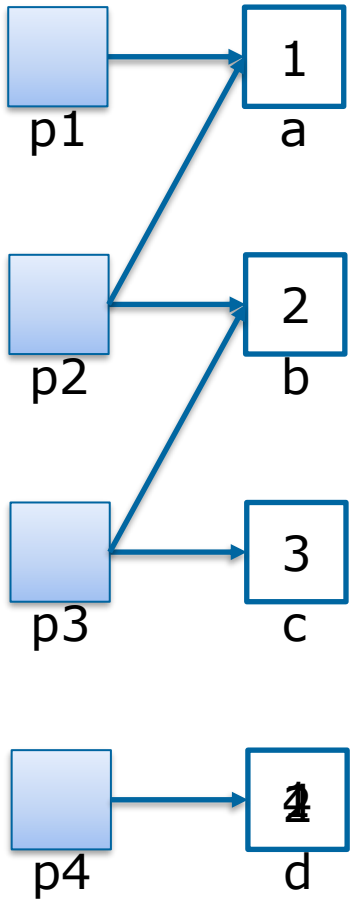
*p2 ← ‘x’



Don’t be confused

Dictionary
    a, b, c, d : integer
    p1, p2, p3, p4 : pointer to integer
Algorithm
    a = 1; b = 2
    c = 3; d = 4
    p1 = &a; p2 = &b
    p3 = &c; p4 = &d

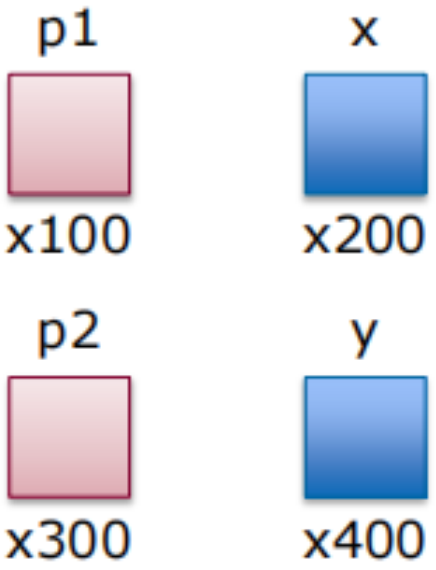
    p2 = p1
    *p4 = *p1
    p3 = &b
    *p4 = b



Exercise - Draw the pointers

Dictionary
    x, y : integer
    p1, p2 : pointer to integer
Algorithm
    x ← 5
    y ← 10

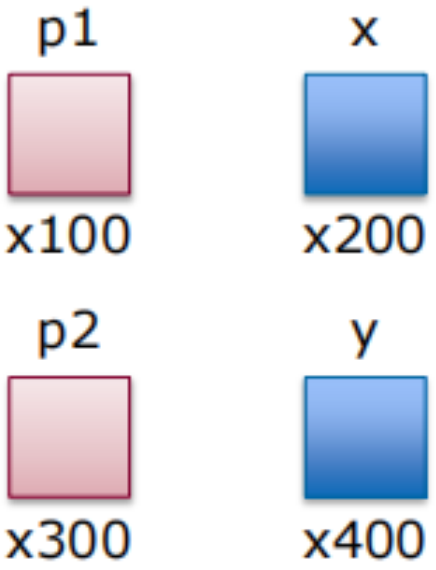
1	p1 ← &x *p1 ← 7
2	p2 ← &y x ← *p2
3	x ← y p1 ← &y p2 ← &x
4	p2 ← &x p1 ← p2 *p2 ← 6



Exercise - Draw the pointers

Dictionary
    x, y : integer
    p1, p2 : pointer to integer
Algorithm
    x ← 5
    y ← 10

1	p1 ← &y p2 ← &x *p1 ← *p2
2	p2 ← &x *p2 ← 7 p1 ← p2
3	p1 ← &x *p1 ← y



Exercise - write the value inside each variable and pointer

Dictionary

a, b, c : integer

p1, p2, p3 : pointer to integer

Algorithm

a ← 10
b ← 15
p1 ← &b
p2 ← p1
c ← 27
p1 ← &c
a ← *p1
p3 ← &b
*p2 ← 8

What is the output?					
0x78 a	0x150 b	0x100 c	0x215 p1	0x111 p2	p3

Exercise - write the value inside each variable and pointer

Dictionary

a, b, c : integer

p1, p2, p3 : pointer to integer

Algorithm

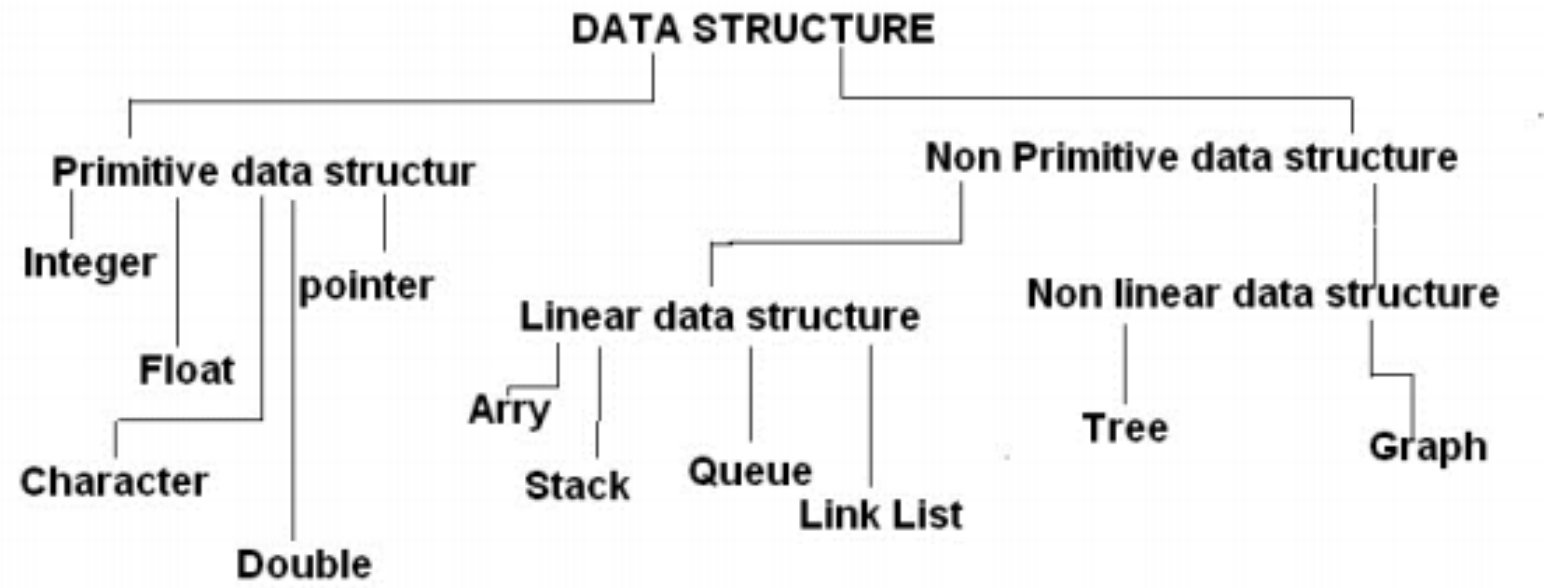
a ← 10
b ← 15
c ← 27
p1 ← &a
p2 ← &b
*p1 ← c
a ← *p2
b ← 6
p3 ← &b
p3 ← &c
*p1 ← *p3

What is the output?

0x86 a	0x150 b	0x100 c	0x215 p1	0x111 p2	0x78 p3

Abstract Data Type

Understanding Data Structure and Algorithm



Classification of data structure

Organizing Data

- Data structure is meant to be
  - an organization for a collection of data items.
  - a way of organizing input data and operations which can be performed on this data
- Organized data must be able to be searched, processed in any order, or modified
- The choice of data structure and algorithm can make the difference between a program running in a few seconds or many days.

Why Data Structure

- In computer science, often the question is not how to solve a problem, but how to solve a problem well
- In this case, it means Efficiency
- Data structure is one of fundamental items to develop a good computer systems
  - You'll learn Computer efficiency more deeply at other subject (Design and Analysis of Algorithms)

Space-Time Tradeoffs and Eficiency

- Trade-off between speed and memory
- To make a more powerful computers → more complex applications.
- More complex applications demand more calculations

Selecting Data Structure

- Analyze the problem to determine the resource constraints a solution must meet.
- Determine the basic operations that must be supported. Quantify the resource constraints for each operation.
- Select the data structure that best meets these requirements

Data Structure Classification

- Linear data structure
  - Linked List
  - Stack
  - Queue
- Non-linear data structure
  - Tree
  - Graph

Question?

Modularity

- Describes a program organized into loosely coupled, highly cohesive modules”
  - Carrano and Prichard, p. 7
- “A technique that keeps the complexity of a large program manageable by systematically controlling the interaction of its components”
  - Carrano and Prichard, p. 106

In Short

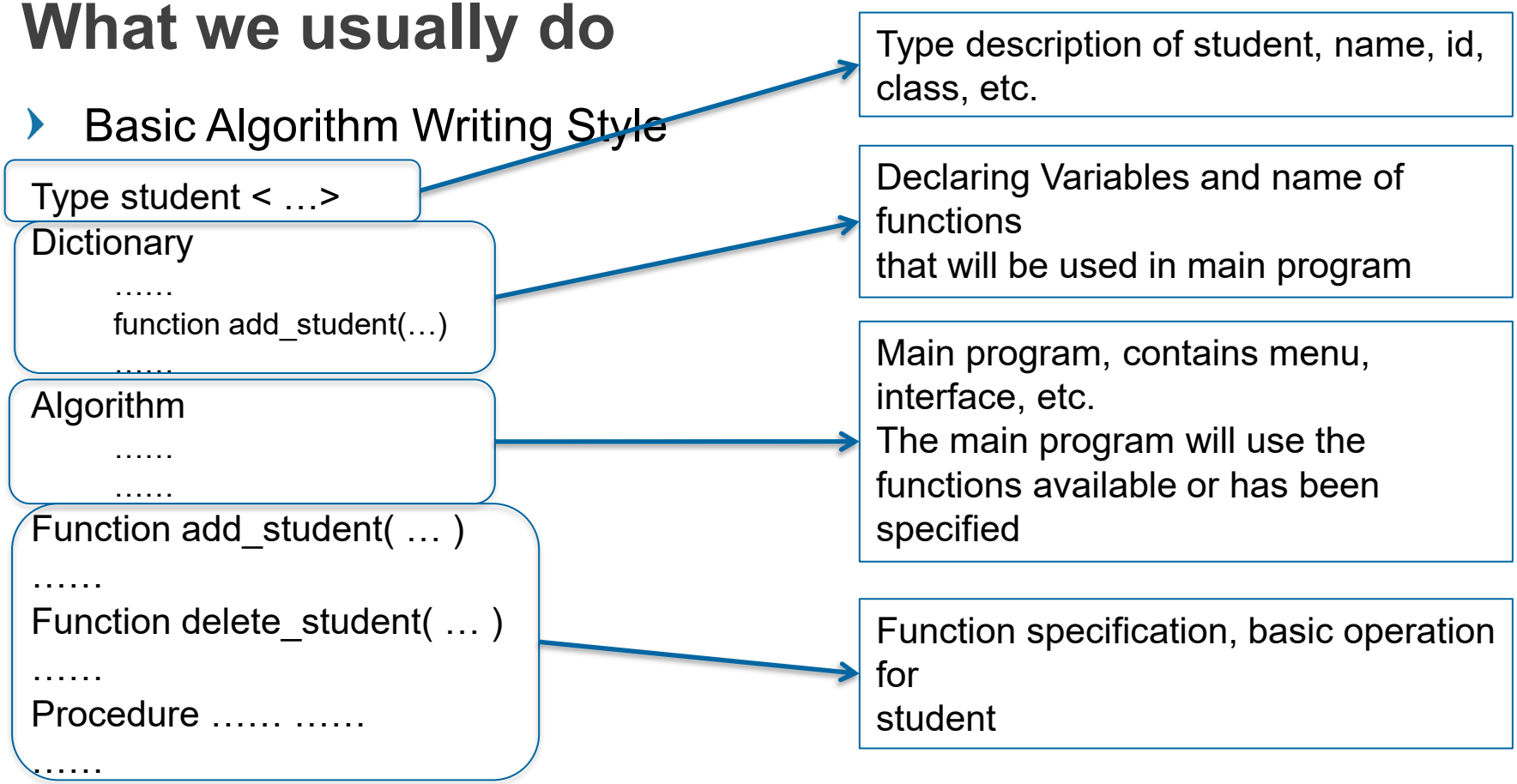
- Modularity is the degree to which a system's components may be separated and recombined
- Here we will know about **Abstract Data Type (ADT)** to understand better about ADT,
  - let's see the illustration

Student Information System

- Suppose we will make an information System to store students record
- There are 4 menus
  - Add new data
  - Delete data
  - Edit data
  - View stored data

What we usually do

- Basic Algorithm Writing Style



All in one file

Here we actually have

- 3 parts of program or modules

Type student < …>
Dictionary
……
function add_student(…)
……

Algorithm
……
……

Function add_student(…)
……
Function delete_student(…)
……
Procedure …… ……
……

→ Type description and Primitive declaration [Specification]

→ Body of main program
[Program implementation]

→ Primitive Implementation (basic operation)

Abstract Data Type

- The Goal of ADT is to **separates** between **specification** and **implementation**
- ADT itself is the Specification part that contains **TYPE** declaration and **PRIMITIVE** specification

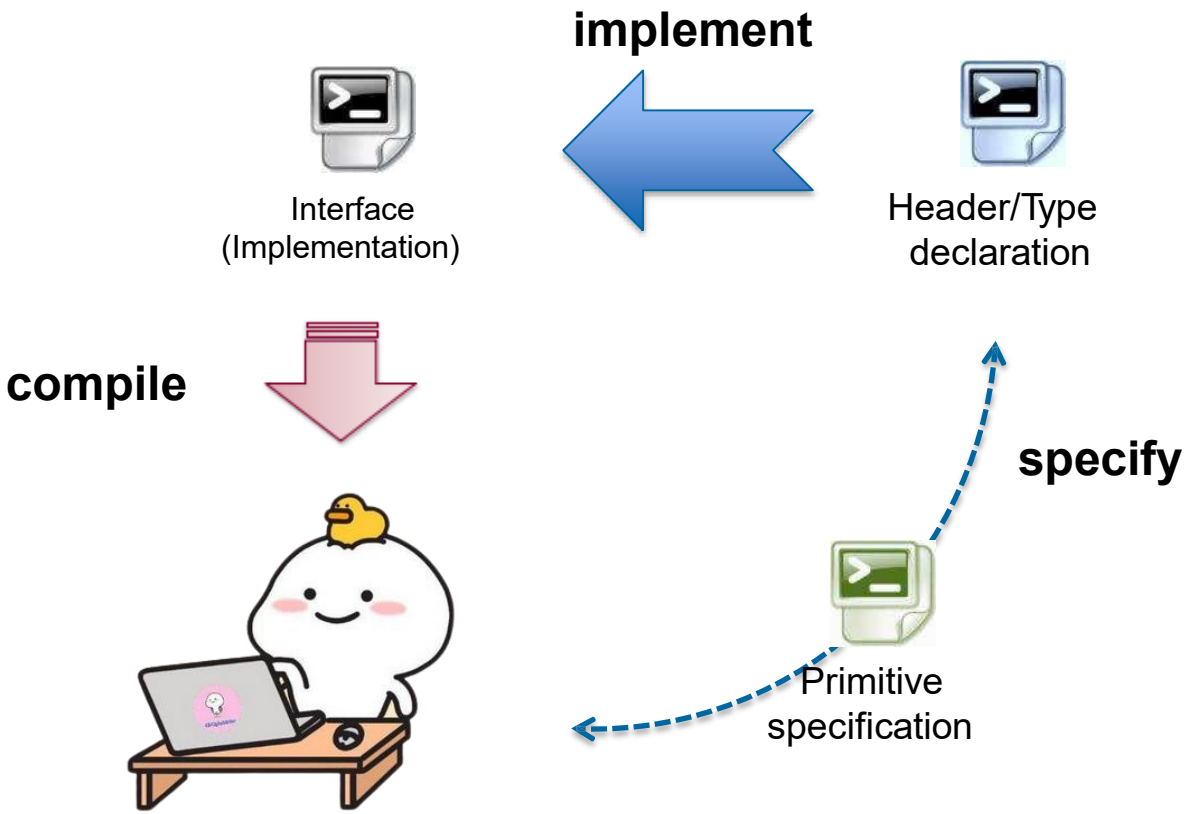
Abstract Data Type

- ADT Style Algorithm

Basic Writing Stye : 1 file
Type student < …> Dictionary …… function add_student(…) …… Algorithm …… …… function add_student(…) …… function delete_student(…) …… procedure …… ……

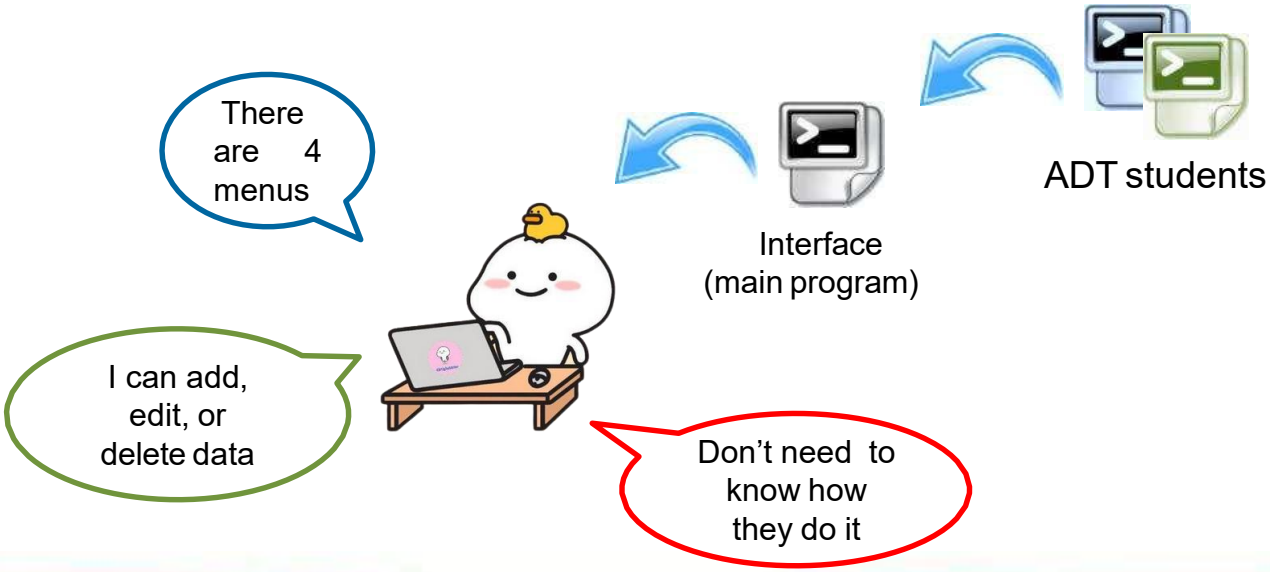
ADT Style : At least 3 files
Header / type declaration
Type student < …> function add_student(…) function delete_student(…)
Primitive specification
function add_student(…) …… function delete_student(…) ……
Main / body program
Algorithm …… ……

Connectivity Between Module



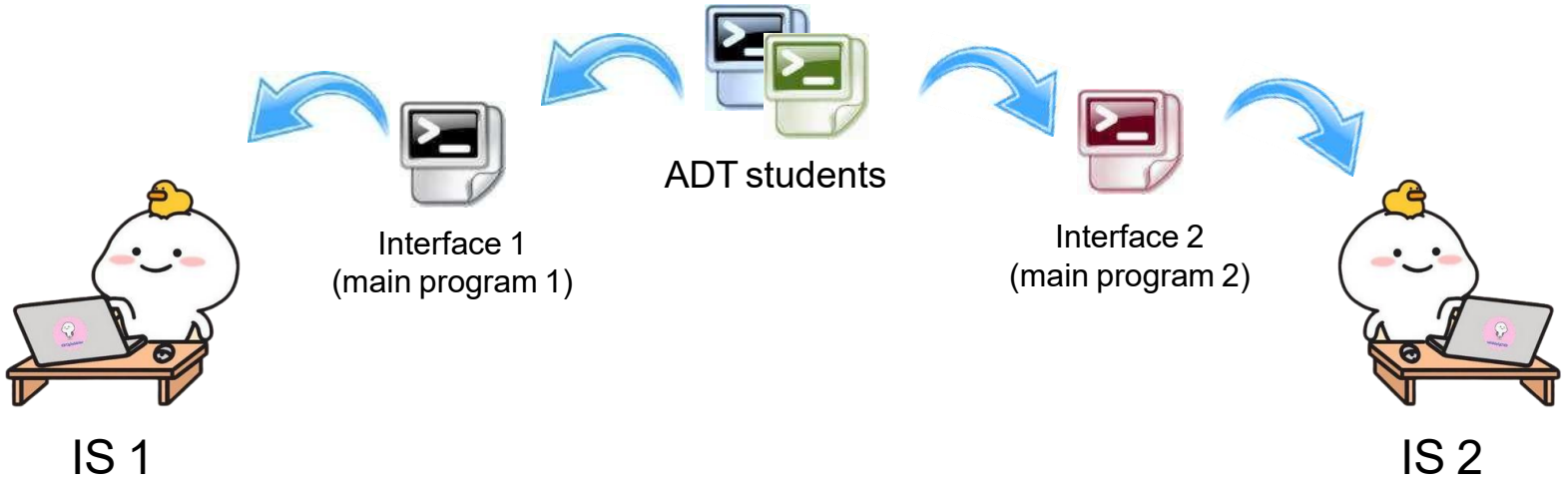
Why we use ADT

- Security
  - user only needs to know the specifications of features without the need to be given detailed implementation of these features.



Why we use ADT

- Reusability
  - Suppose we're going to build another information system that happens also use students record (add, edit, delete)
  - We don't need to code the ADT again, we can use what we already have



DRY Principles

- Don’t Repeat Yourself
- “Every piece of knowledge must have a single, unambiguous, authoritative representation within a system”
  - When the DRY principle is applied successfully, a modification of any single element of a system does not require a change in other logically unrelated elements

Question?

Task/Exercise : Clock ADT

- Create a Clock ADT (Clock.h) to store time (hour, minute, and second)

```
Type Hour : integer [0..23]
Type Minute : integer [0..59]
Type Second : integer [0..59]
Type Clock : < HH : Hour, MM: Minute, SS : Second>
```

Task/Exercise : Clock ADT

- Primitive for Clock.h
- Validator
  - function isValid(HH,MM,SS: integer) → boolean
  - { return true if 0≤HH≤23, and 0≤MM≤59, and 0≤SS≤59 }
- Constructor
  - function makeClock(HH, MM, SS: integer) → Clock
  - { return Clock created from input }

Task/Exercise : Clock ADT

- Selector
  - function getHour(c : Clock) → Hour
  - function getMinute(c : Clock) → Minute
  - function getSecond(c : Clock) → Second
- Value changer
  - procedure setHour(In/Out c : Clock, newHH: integer)
  - procedure setMinute(In/Out c : Clock, newMM: integer)
  - procedure setSecond(In/Out c : Clock, newSS: integer)

Task/Exercise : Clock ADT

- Relational Operation
  - Function isEqual (c1 : Clock, c2 : Clock) → Boolean
  - c1=c2
- Arithmetic Operation
  - function addClock (c1 : Clock, c2 : Clock) → Clock
- Output Process
  - procedure printClock (c : Clock);

Task/Exercise : Clock ADT

- Create the Implementation of Clock ADT (Clock.cpp)
- Create the Driver application to try the implementation (Main.cpp)
  - Example :
  - c1 ← makeClock(2,30,4)
  - c2 ← makeClock(6,0,0)
  - c3 ← isEqual (c1, c2)
  - output(getHour(c1))

Task/Exercise : Clock ADT

- Create the Implementation of Clock ADT (Clock.cpp)
- Create the Driver application to try the implementation (Main.cpp)
  - Example :
  - c1 ← makeClock(2,30,4)
  - c2 ← makeClock(6,0,0)
  - c3 ← isEqual (c1, c2)
  - output(getHour(c1))

Train Your Brain

- Divide class into groups (3-4 students/group)
- Implement all ADT Clock's functions and procedures in pseudocode (discuss with your group, handwritten)
- No gadget involved!

Train Your Brain

1. function **isValid**(hh, mm, ss: integer) → Boolean
2. function **makeClock**(hh, mm, ss: integer) → Clock
3. function **getHour**(c : Clock) → integer
4. function **getMinute**(c : Clock) → integer
5. function **getSecond**(c : Clock) → integer
6. procedure **setHour**(in/out c : Clock, in newHH: integer)
7. procedure **setMinute**(in/out c : Clock, in newMM: integer)
8. procedure **setSecond**(in/out c : Clock, in newSS: integer)
9. function **isEqual** (c1 : Clock, c2 : Clock) → Boolean
10. function **addClock** (c1 : Clock, c2 : Clock) → Clock
11. procedure **printClock** (in c : Clock)

Home Task

- Learn more about pointer in Cpp
- Create a project to try the previous exercise in Cpp
- Read more about Dynamic Memory Allocation

Fakultas Informatika
School of Computing
Telkom University

THANK YOU