Fakultas Informatika
School of Computing
Telkom University

CDK2AAB4
STRUKTUR DATA

Pointer & Abstract Data Type (ADT)

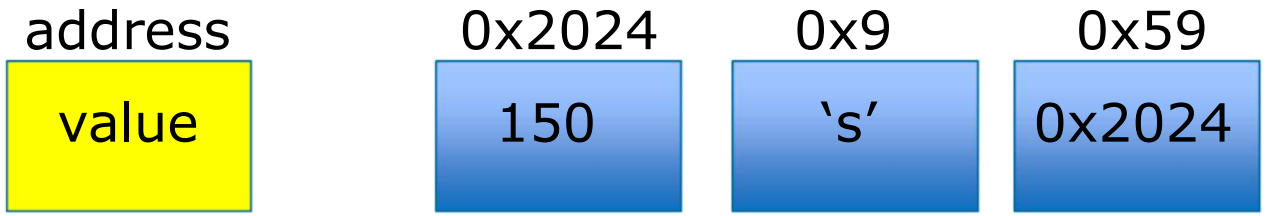
Fakultas Informatika
School of Computing
Telkom University

CDK2AAB4
STRUKTUR DATA

Pointer

Intro

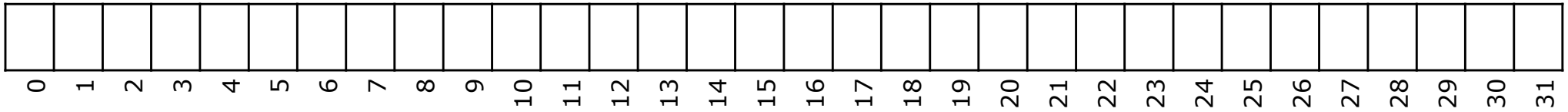
- Variables
  - something for holding data, placeholders for representing data



- Data Types
  - **primitive** data types: int, float, char, double, bool, etc.
  - **user-defined** data types: struct (or type in pseudocode)
- A Pointer Variables
  - a variable that stores the address where another object resides.

Data and Memory

- Data of a variable is stored in memory
- Picture it as a 1-dimension array

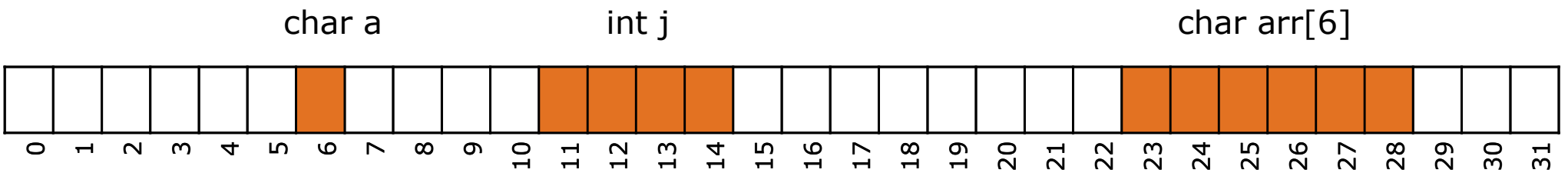


- Each cell has a unique “index”, we call it **address**

Data and Memory

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

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

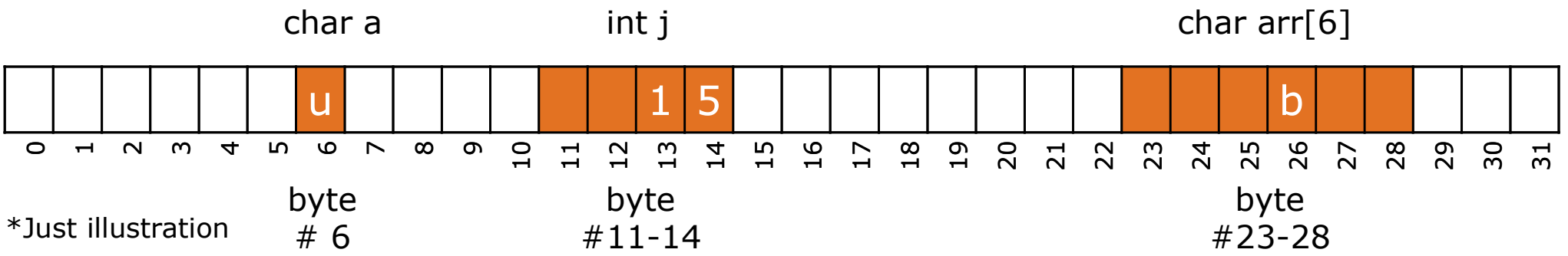


*Just illustration

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
```

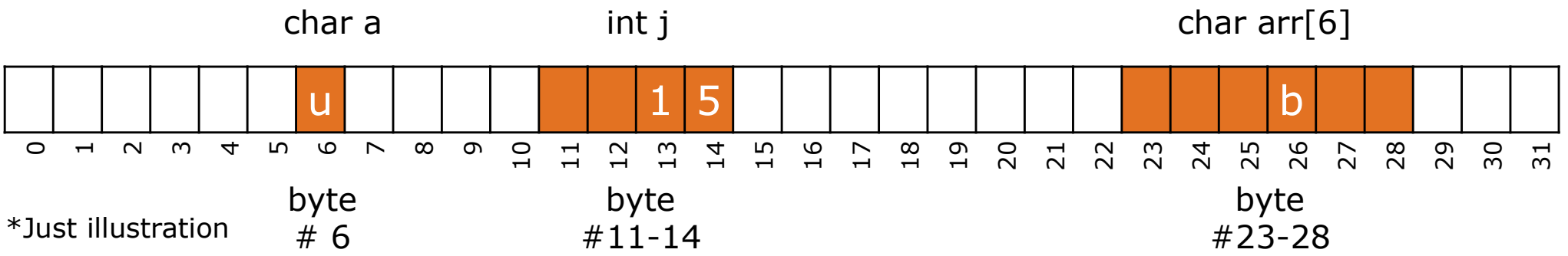


Data and Memory

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

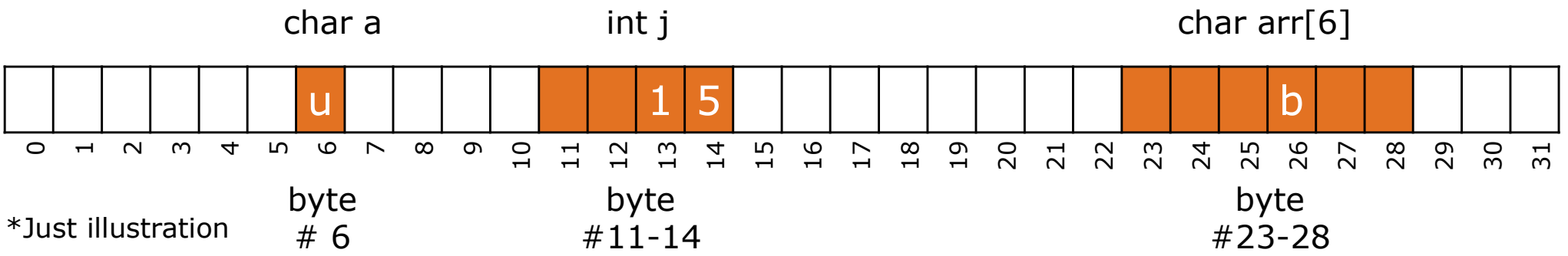
```
Algorithm
    output( a )
    output( &a )
    output( &arr[3] )
```

```
output
u
0x6
0x26
```



Pointer

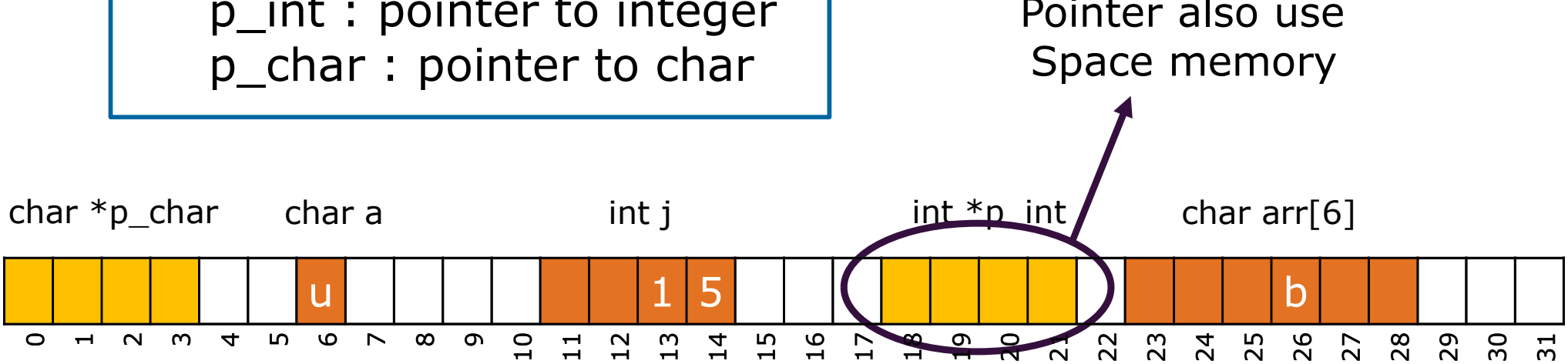
- Basic variable type
- Store an address of a variable in hexadecimal
- Size of an integer (4 byte)



Pointer

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

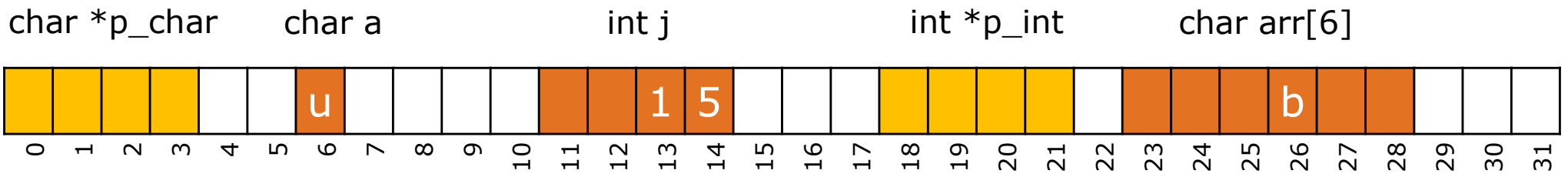
Dictionary
p_int : pointer to integer
p_char : pointer to char



*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



*Just illustration

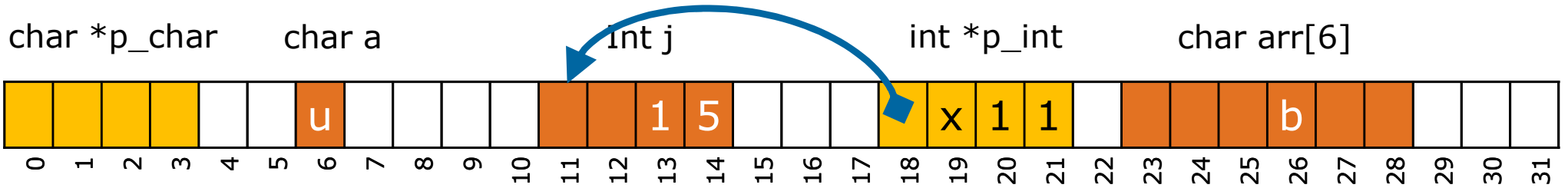
Operation using Pointer

Algorithm

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

Output

```
15
0x11
15
```



Operation using Pointer

Algorithm

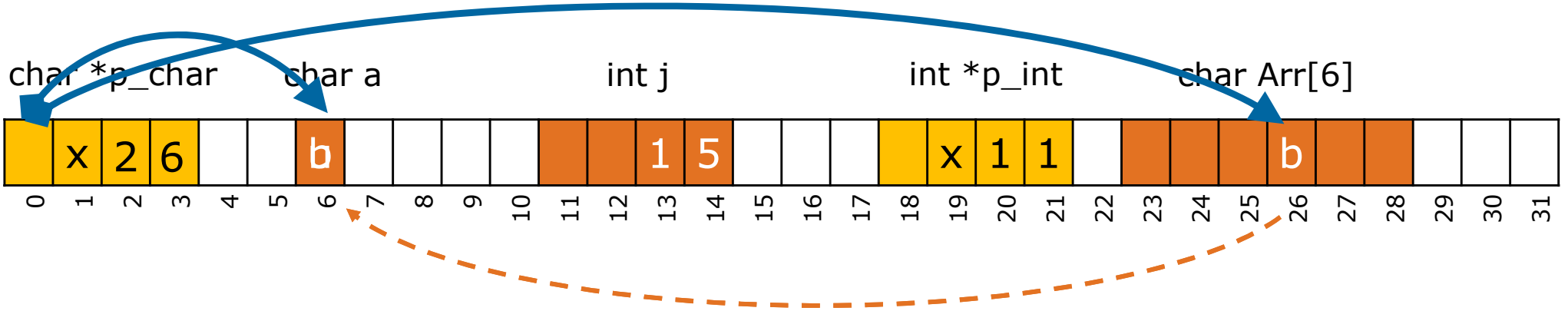
```
p_char ← &a
output( *p_char )
p_char ← &arr[3]
output( *p_char )
a ← *p_char
output( a )
```

Output

```
'u'  // pointing to a

'b'  // pointing to arr[6]

'b'
```

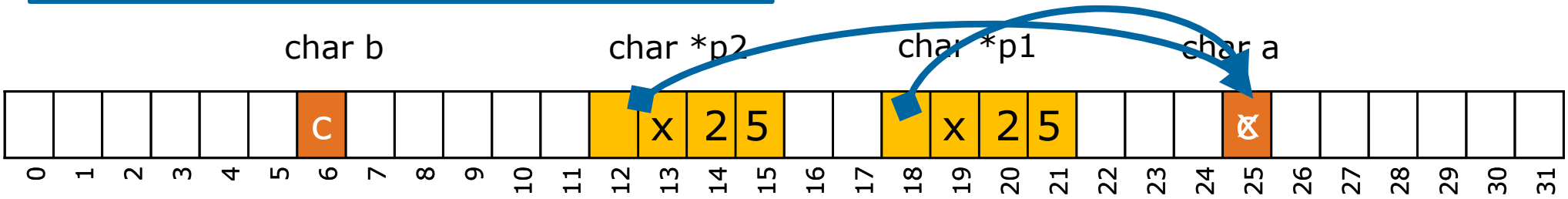


Pointers

- 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

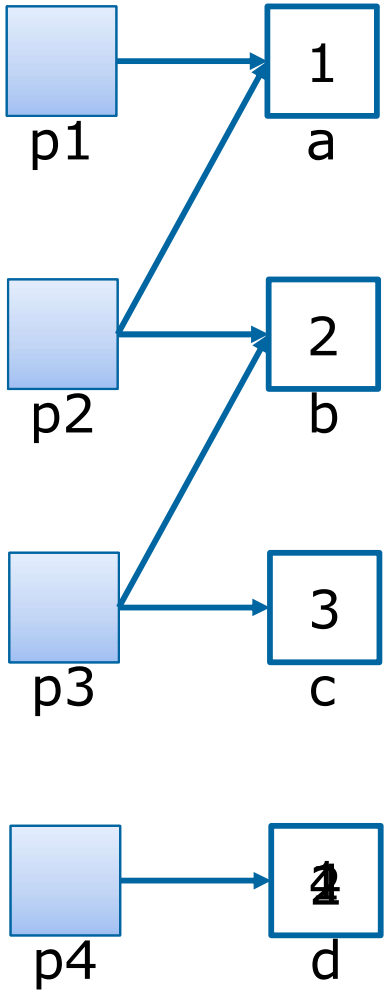
Dictionary
    a, b : char
    p1, p2 : pointer to char
Algorithm
    a ← ‘c’
    p1 ← &a
    p2 ← p1
    b ← *p1
    *p2 ← ‘x’



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

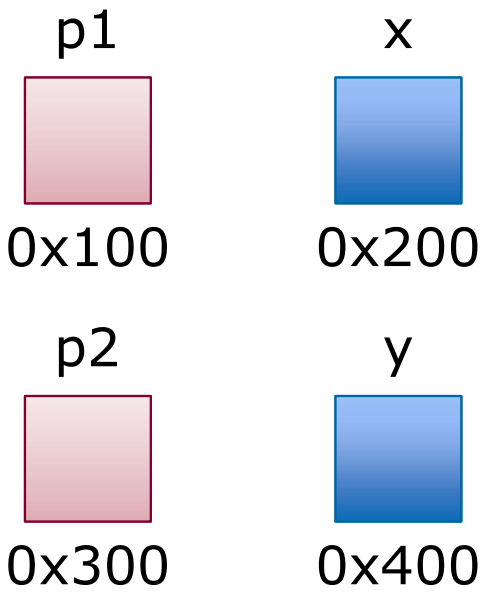


Question?

Exercise – draw the pointers

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

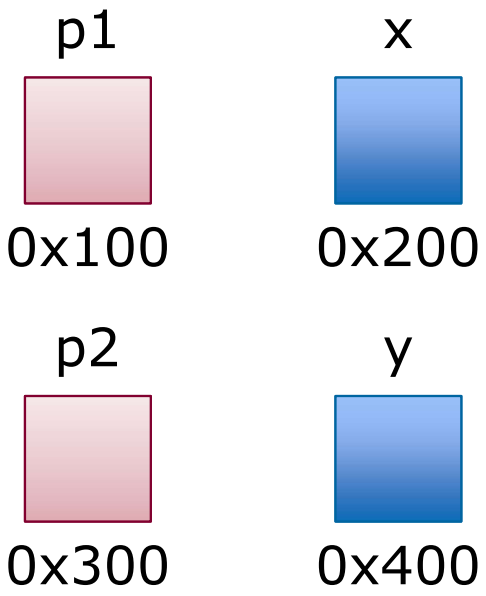
1	p1 ← &x *p1 ← 7
2	p2 ← &y x ← *p2
3	x ← y p1 ← &y p2 ← &x
4	p2 ← &x p1 ← p2 *p2 ← 6



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

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

Question?

Fakultas Informatika
School of Computing
Telkom University

CDK2AAB4
STRUKTUR DATA

Abstract Data Type (ADT)

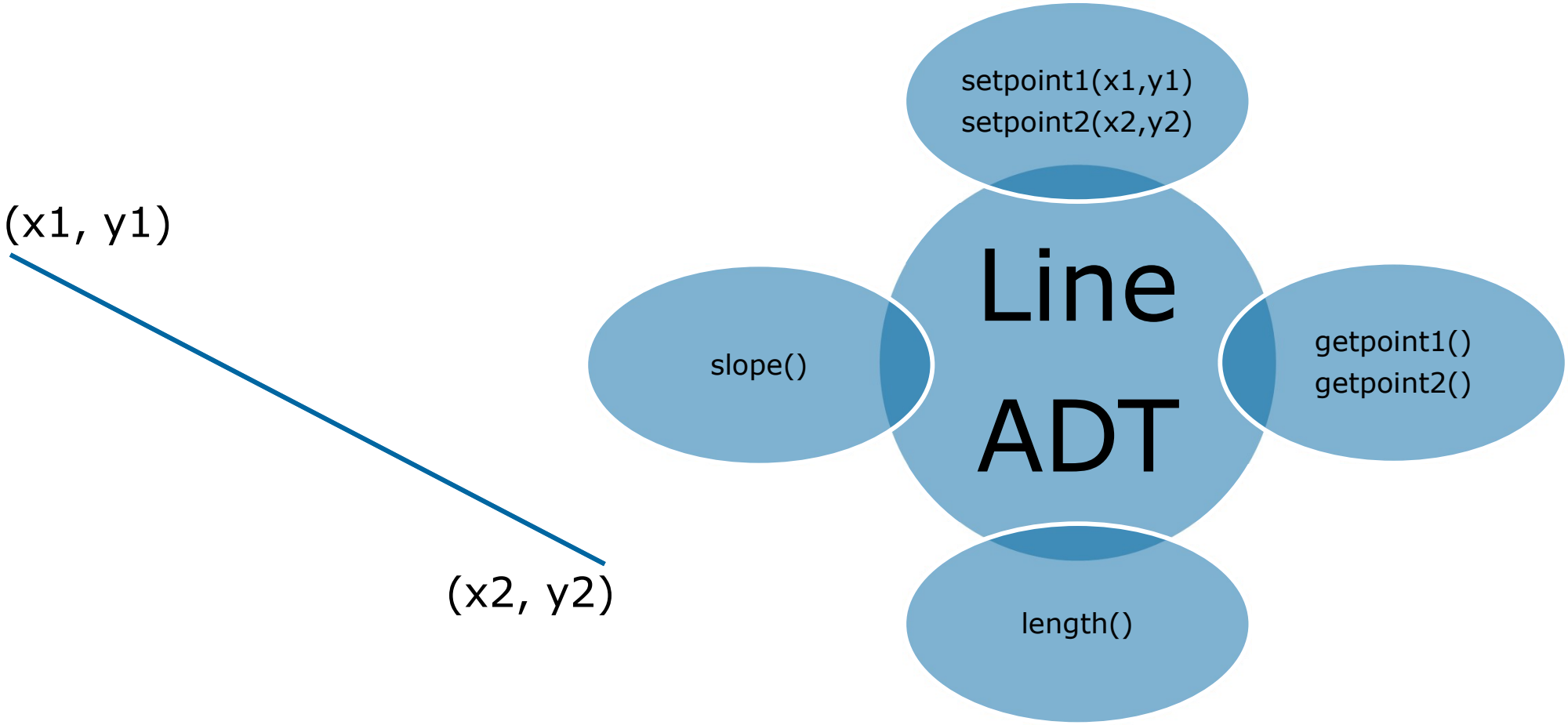
Abstract Data Types (ADTs)

- **Conceptual Description**

  what **operations** are performed, **NOT** how they are implemented.

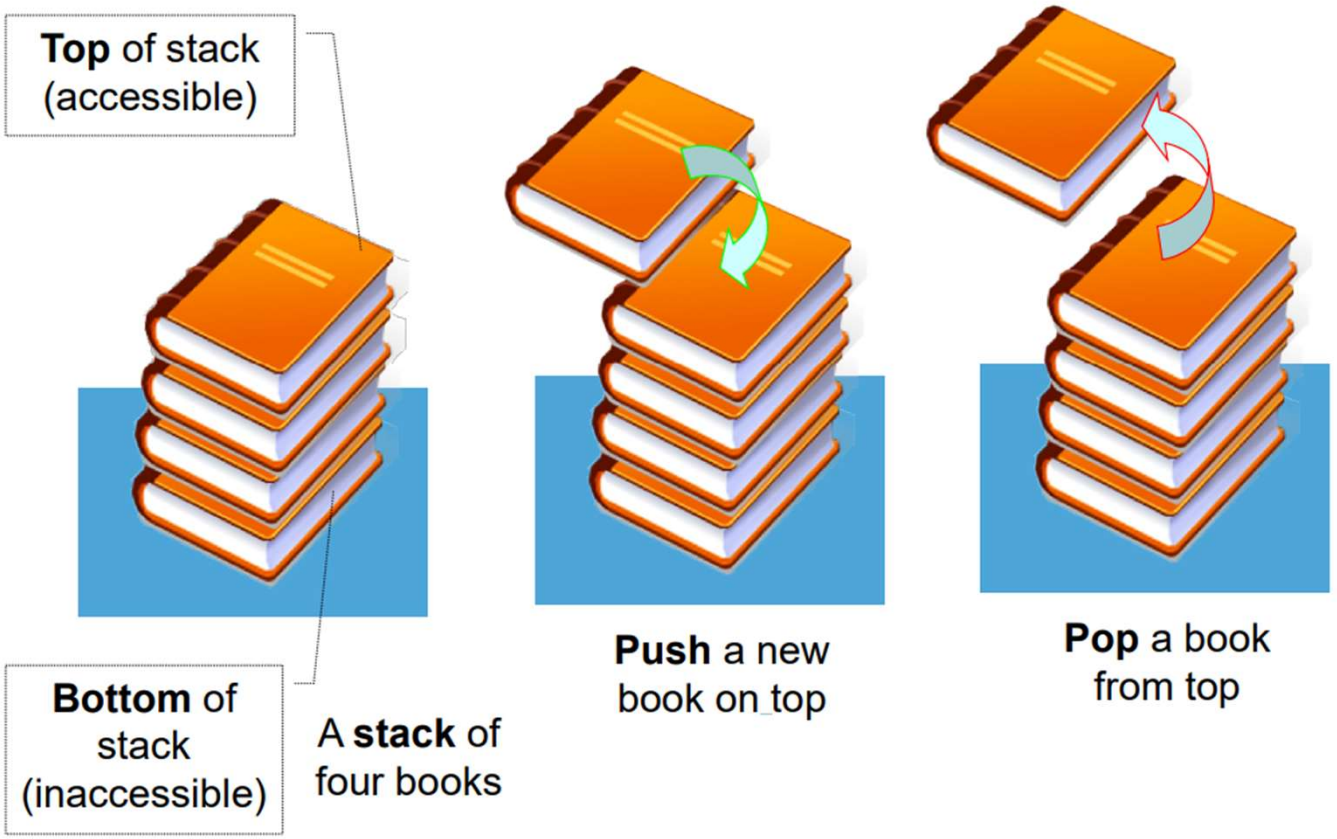
  - declaration of **data**
  - declaration of **operations**

Example: Line ADT

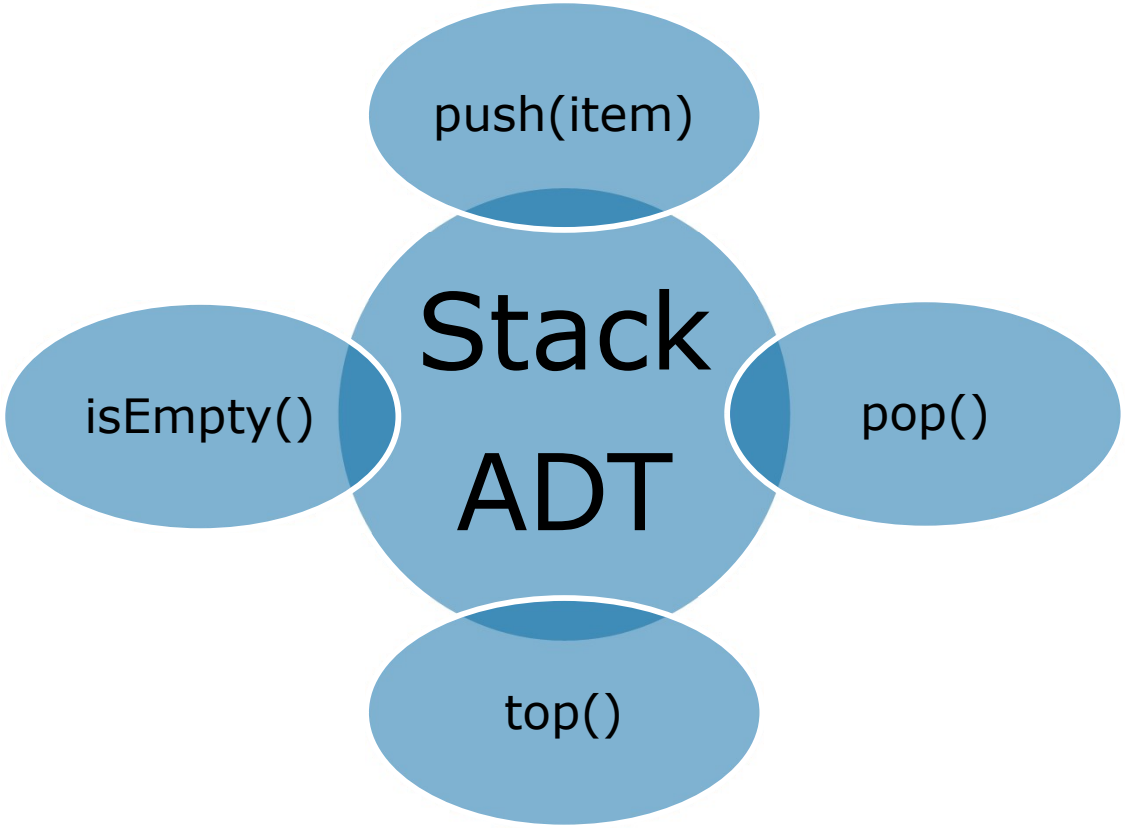


Example 2: Stack ADT

Stack: Illustration



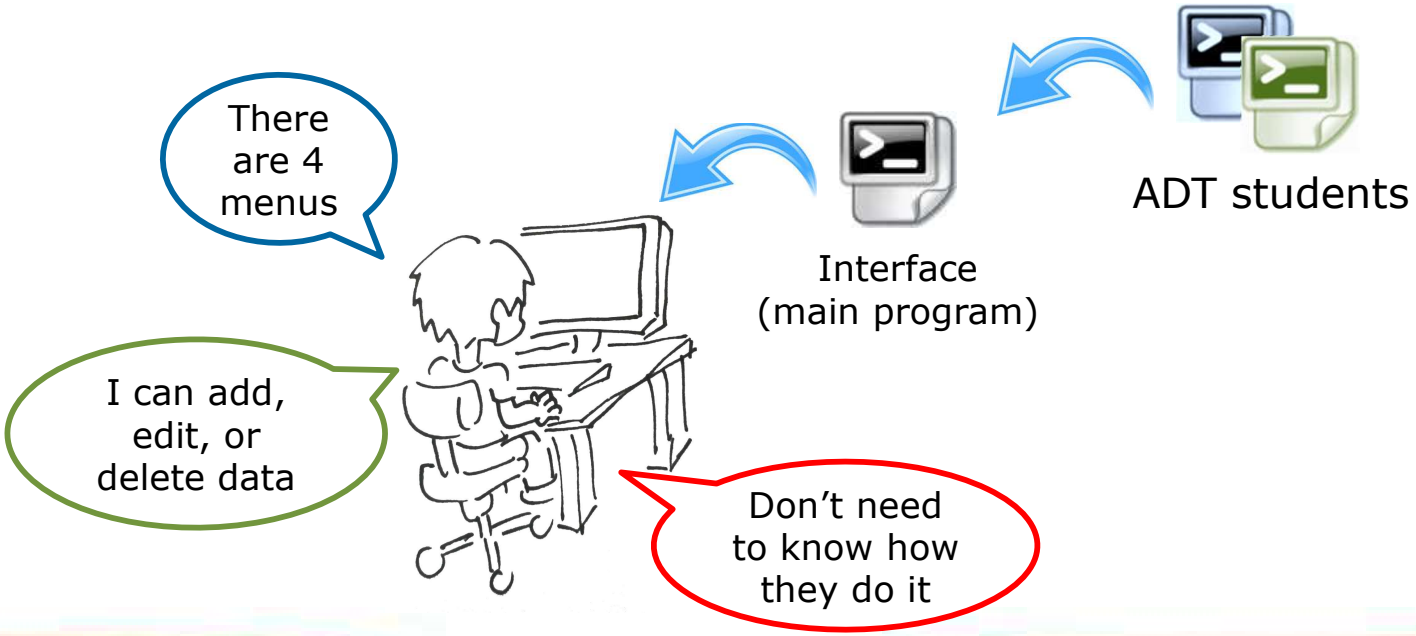
Example 2: Stack ADT (cont.)



https://en.cppreference.com/w/cpp/container/stack

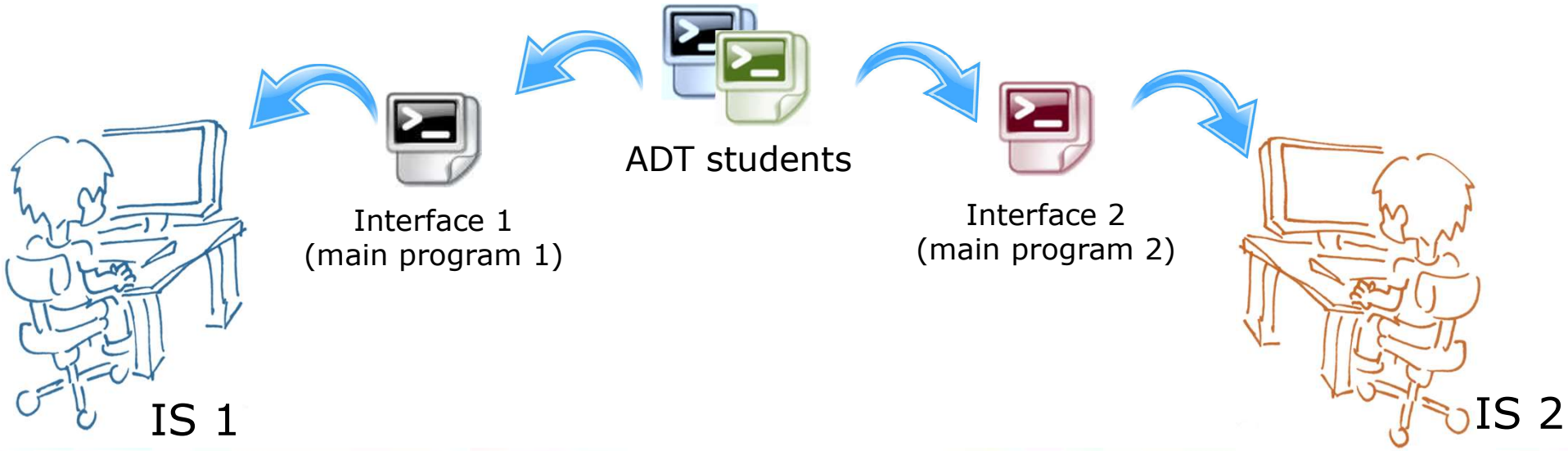
Why we use ADT?

- 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



Data Structures

- **Concrete Implementation:** A data structure is a specific implementation of an ADT for organizing and storing data.
  - how data is structured in memory and the algorithms used to **implement** the operations defined by an ADT.
- Data Structure: Stack
  - Array-based Stack
  - List-based Stack

Question?

Task/Exercise : Clock ADT

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

```
type Clock :
<
    hh      : integer,
    mm      : integer,
    ss      : integer
>
```

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≤MM≤59 }
- Constructor
  - function **makeClock**(hh, mm, ss: integer) → Clock
    { return valid clock created from input; if invalid, return 0:0:0 }

Task/Exercise : Clock ADT

- Selector
  - function getHour(c : Clock) → integer
  - function getMinute(c : Clock) → integer
  - function getSecond(c : Clock) → integer

- Value changer
  - procedure setHour(in/out c : Clock, in newHH: integer)
  - procedure setMinute(in/out c : Clock, in newMM: integer)
  - procedure setSecond(in/out c : Clock, in newSS: integer)

  { value will be unchanged if hour/minute/second is invalid }

Task/Exercise : Clock ADT

- Relational Operation
  - function isEqual (c1 : Clock, c2 : Clock) → Boolean
- Arithmetic Operation
  - function addClock (c1 : Clock, c2 : Clock) → Clock

  { return valid clock; if invalid, return 0:0:0 }
- Output Process
  - procedure printClock (in c : Clock)

Task/Exercise : Clock ADT

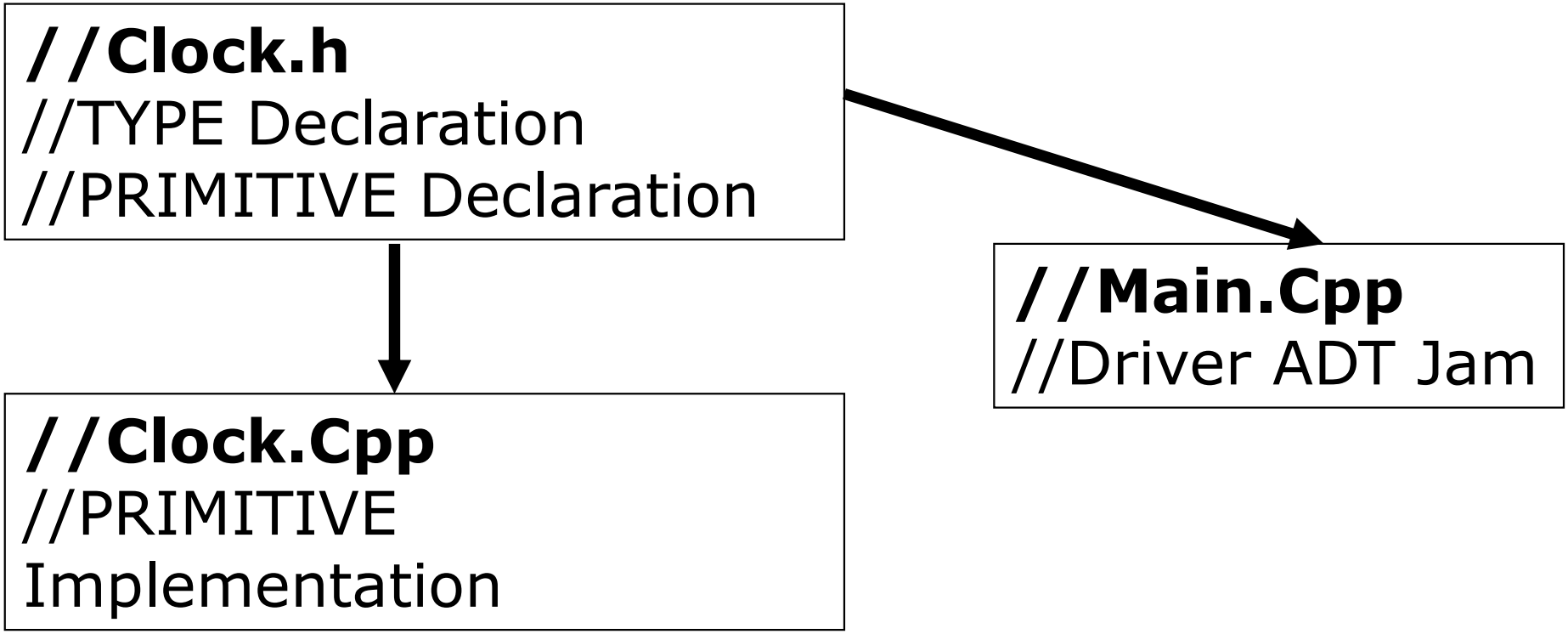
- Create the Implementation of Clock ADT (Clock.cpp)
- Create the Driver application to try the implementation (Main.cpp)

Example :

```
c1, c2, c3: Clock
c1 ← makeClock( 2, 30, 4 )
c2 ← makeClock( 6, 0, 0 )
c3 ← addClock ( c1, c2 )
printClock ( c3 )                    { print 8:30:4 }
```

Clock ADT

- Implementation Diagram of Clock ADT




TRAIN YOUR BRAIN

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

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)


```
function isValid( hh, mm, ss: integer ) → Boolean


endfunction
```


```
function makeClock( hh, mm, ss: integer ) → Clock
.
:
endfunction
```


```
function getHour( c : Clock ) → integer
:

endfunction

function getMinute( c : Clock ) → integer
:

endfunction

function getSecond( c : Clock ) → integer


endfunction
```


```
procedure setHour( in/out c : Clock, in newHH: integer )
;



endprocedure

procedure setMinute( in/out c : Clock, in newMM: integer )



endprocedure
```


```
procedure setSecond( in/out c : Clock, in newSS: integer )
;



endprocedure
```


```
function addClock ( c1 : Clock, c2 : Clock ) → Clock
:
:
:
endfunction
```


```
procedure printClock ( in c : Clock )
;

endprocedure
```

Referensi

- Karumanchi, N. (2017). Data Structures And Algorithms Made Easy (5th ed.). CareerMonk Pub.
- Weiss, M. A. (2014). Data Structures and Algorithm Analysis in C++ (4th ed.). Addison-Wesley Pub.
- Drozdek, A. (2013). Data Structures and Algorithms in C++ (4th ed.). Cengage Learning.
- NUS Computing. Stack ADT. Lecture 7a.

Fakultas Informatika
School of Computing
Telkom University

THANK YOU