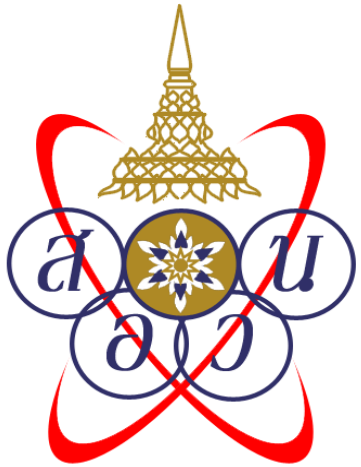


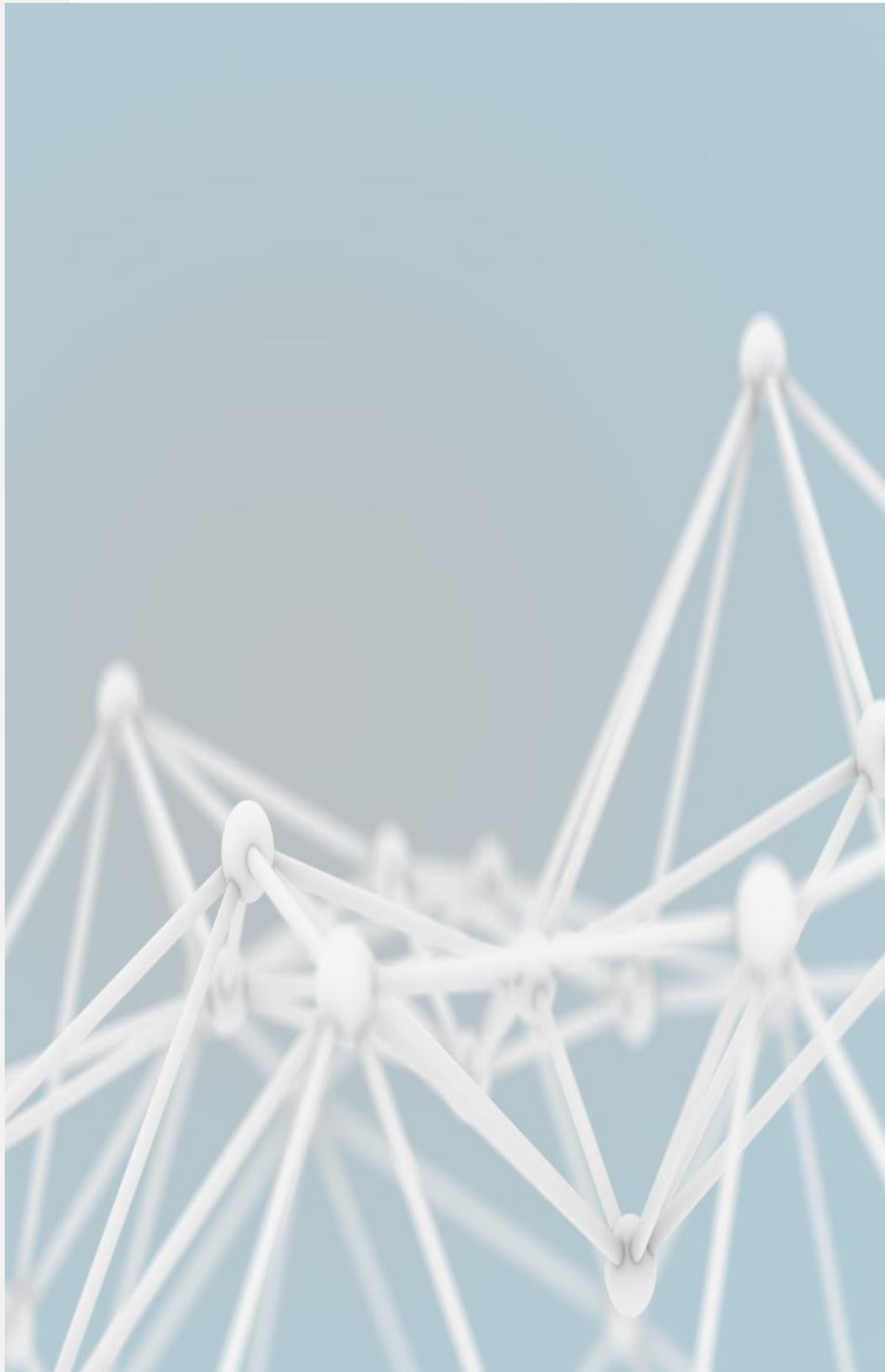
เอกสารประกอบการอบรม
ส่วนที่ 1 วิชาโครงสร้างข้อมูล

ค่ายคอมพิวเตอร์โอลิมปิก สอวน. ค่าย 2 2/2566
ศูนย์โรงเรียนสามเสนวิทยาลัย - มหาวิทยาลัยธรรมศาสตร์
ระหว่างวันที่ 18 มีนาคม – 3 เมษายน 2567

โดย
สาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์และเทคโนโลยี
มหาวิทยาลัยธรรมศาสตร์

ส่วนที่ 1: โครงสร้างข้อมูล

ผศ.ดร.วิรัตน์ จารีวงศ์ไพบูลย์
อาจารย์ นุชชากร งามเสาวรส



เค้าโครงการอบรม

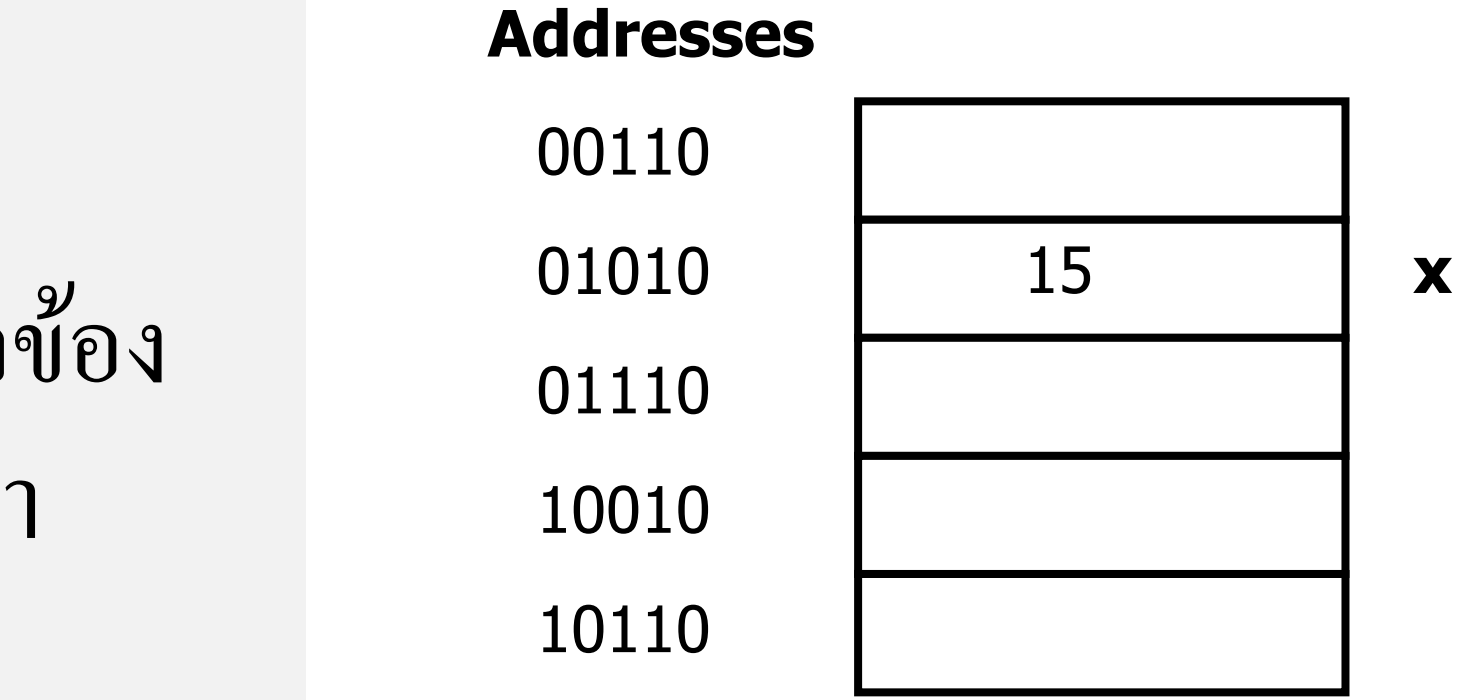
- ทบทวนการใช้ pointer และ array ในภาษา C++
- ข้อมูลนามธรรม
- Linear Data Structure
  - List, Stack, Queue
- Recursion
- Non-Linear Data Structure
  - Tree, Binary Tree, Binary Search Tree, Binary Heaps Tries, Graph, Hashing

ทบทวนการใช้ Pointer และ Array

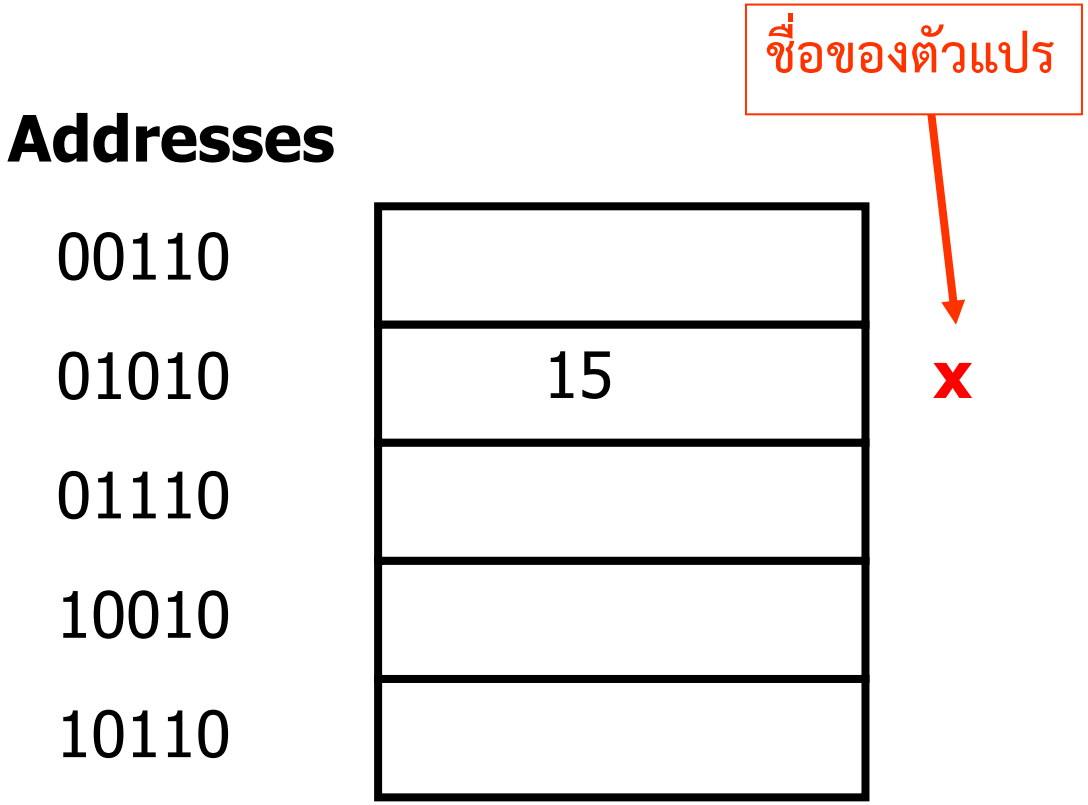
Terms ที่เกี่ยวข้องกับหน่วยความจำ

- variable name
- variable
- value
- address – เลขฐานสองที่ระบบปฏิบัติการใช้เพื่อระบุตำแหน่งของหน่วยความจำใน RAM

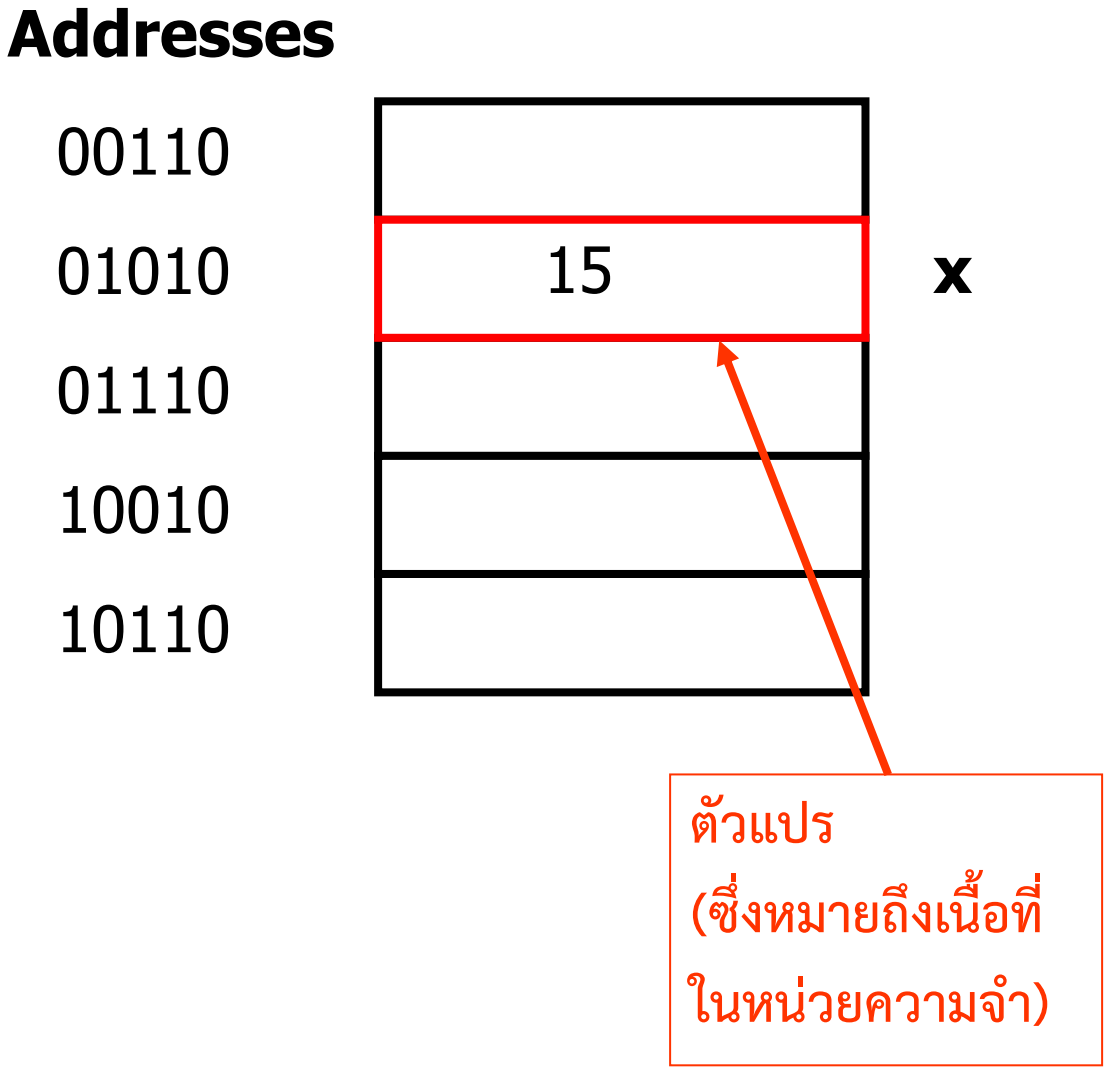
Terms ที่เกี่ยวข้องกับหน่วยความจำ



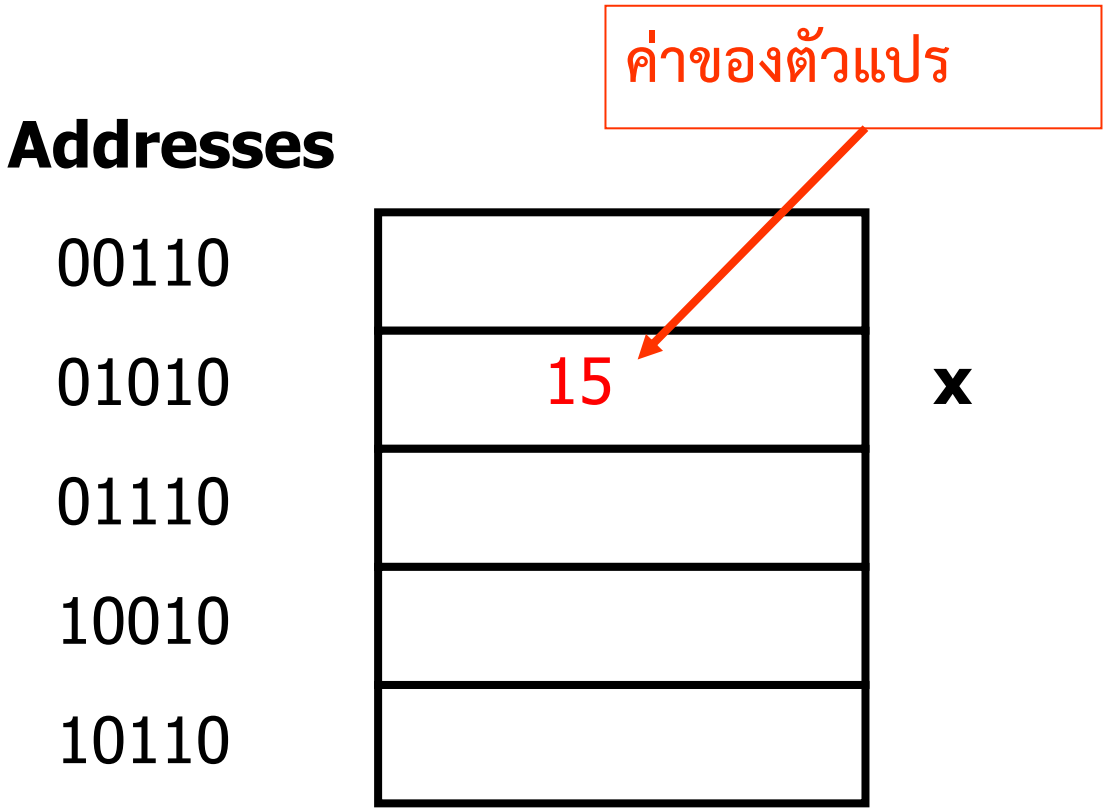
Terms ที่เกี่ยวข้องกับหน่วยความจำ (Variable name)



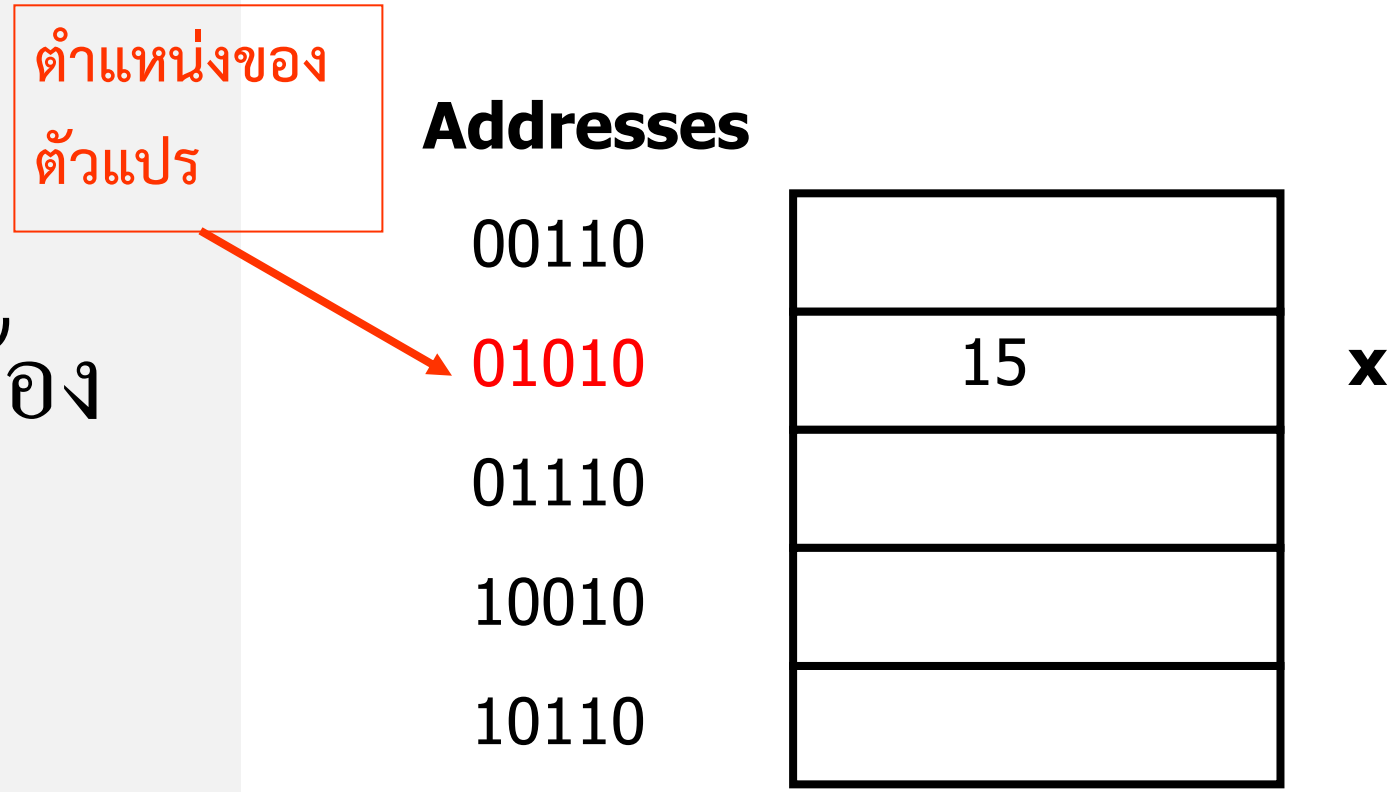
Terms ที่เกี่ยวข้องกับหน่วยความจำ (Variable)



Terms ที่เกี่ยวข้องกับหน่วยความจำ (Value)

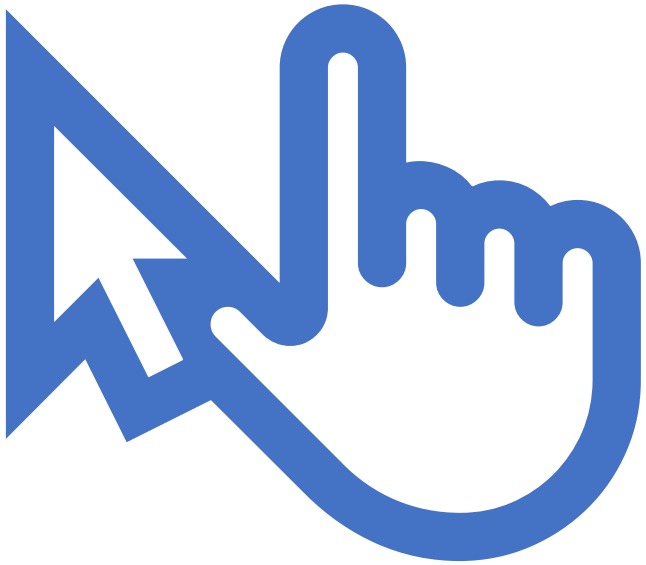


Terms ที่เกี่ยวข้องกับหน่วยความจำ (Address)



ลักษณะของตัวแปร

- ตัวแปร (variable) เป็นตำแหน่งในหน่วยความจำ
- เมื่อตัวแปรถูกใช้ในโปรแกรม จะมีความหมายต่างกันขึ้นกับว่าตัวแปรนั้นอยู่ด้านซ้ายหรือด้านขวาของ assignment statement
- ถ้าตัวแปรอยู่ทางด้านซ้ายของ assignment statement ตำแหน่งของตัวแปรจะถูกใช้
- ถ้าตัวแปรไม่ได้อยู่ทางด้านซ้ายของ assignment statement ค่าที่เก็บในตำแหน่งนั้นจะถูกใช้
- ตัวอย่าง a = b; /* ค่าที่เก็บในตำแหน่ง b จะถูกนำไปใส่ที่ตำแหน่ง
  ของตัวแปร a */



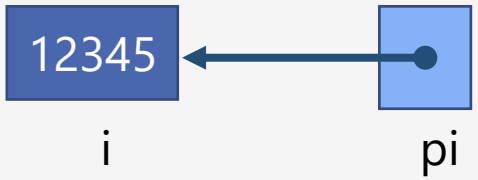
Pointers

- A ***pointer*** หมายถึงตัวแปรที่เก็บค่า address หรือตำแหน่งในหน่วยความจำ
- การประกาศตัวแปร pointer ต้องกำหนดชนิดของข้อมูลที่จะเก็บในตำแหน่งที่ pointer เก็บอยู่ เช่น

```
int   *pi;     /* a pointer to an integer */
float  *pf;    /* a pointer to a float number */
char   *pc;    /* a pointer to a character */
```

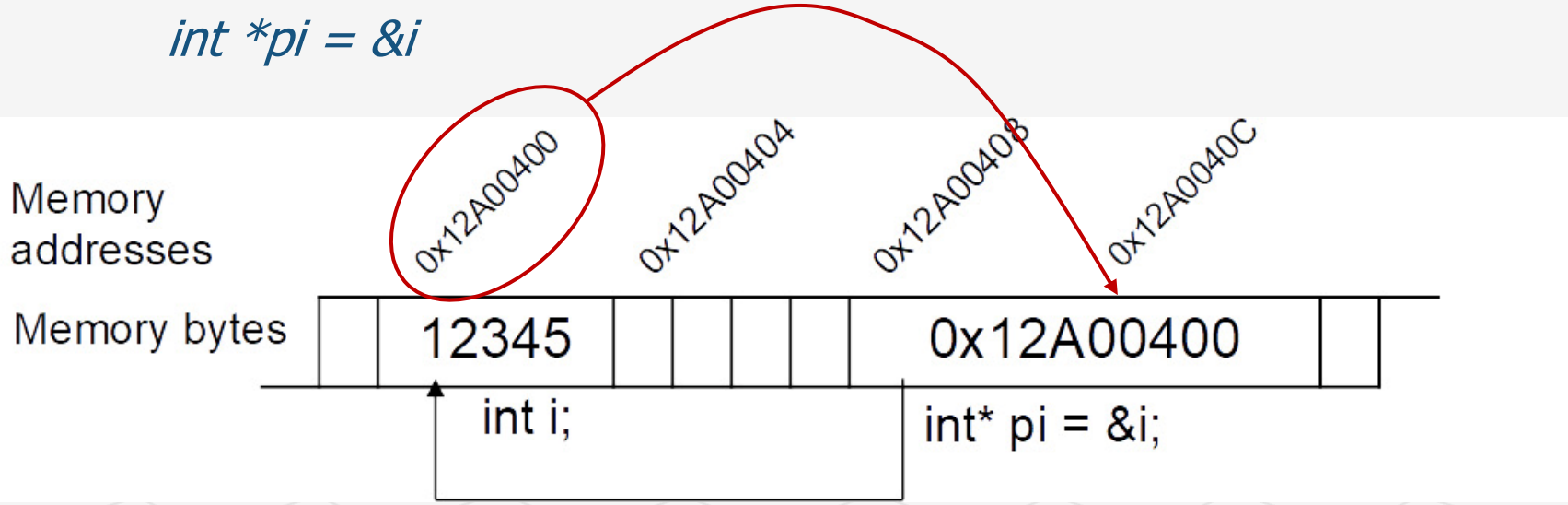
การกำหนดค่าเริ่มต้นของ pointer

- ในการประกาศตัวแปร pointer เราไม่ควรปล่อยให้ตัวแปร pointer มีค่าที่อาจทำให้ผิดพลาด
- เราจึงควรกำหนดค่า address เริ่มต้นให้กับตัวแปร pointer ด้วยการใช้ ***address-of*** operator (&)



int i = 12345;

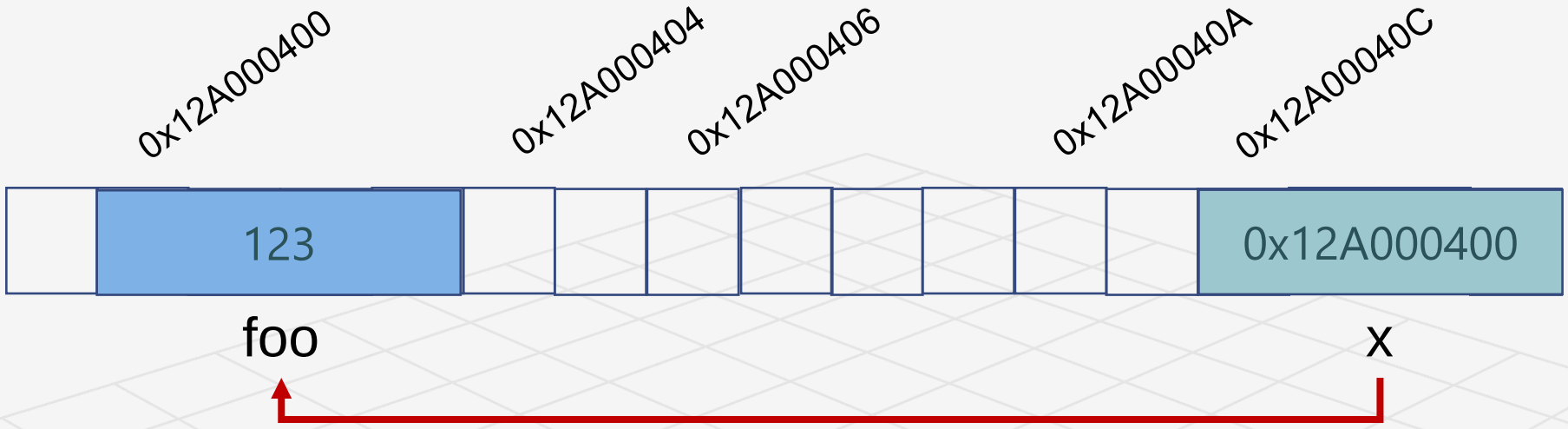
*int *pi = &i*



The address-of(&)operator

ใน C เราสามารถเอา *address* ของตัวแปรเพื่อนำมาใช้งานโดยใช้ "&" operator.

```
int *x;
int foo = 123;
x = &foo;          //&foo means "the address of foo"
```

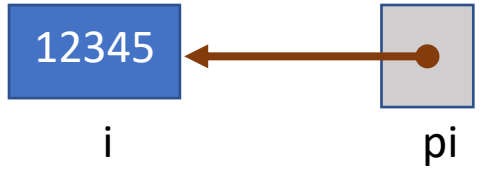


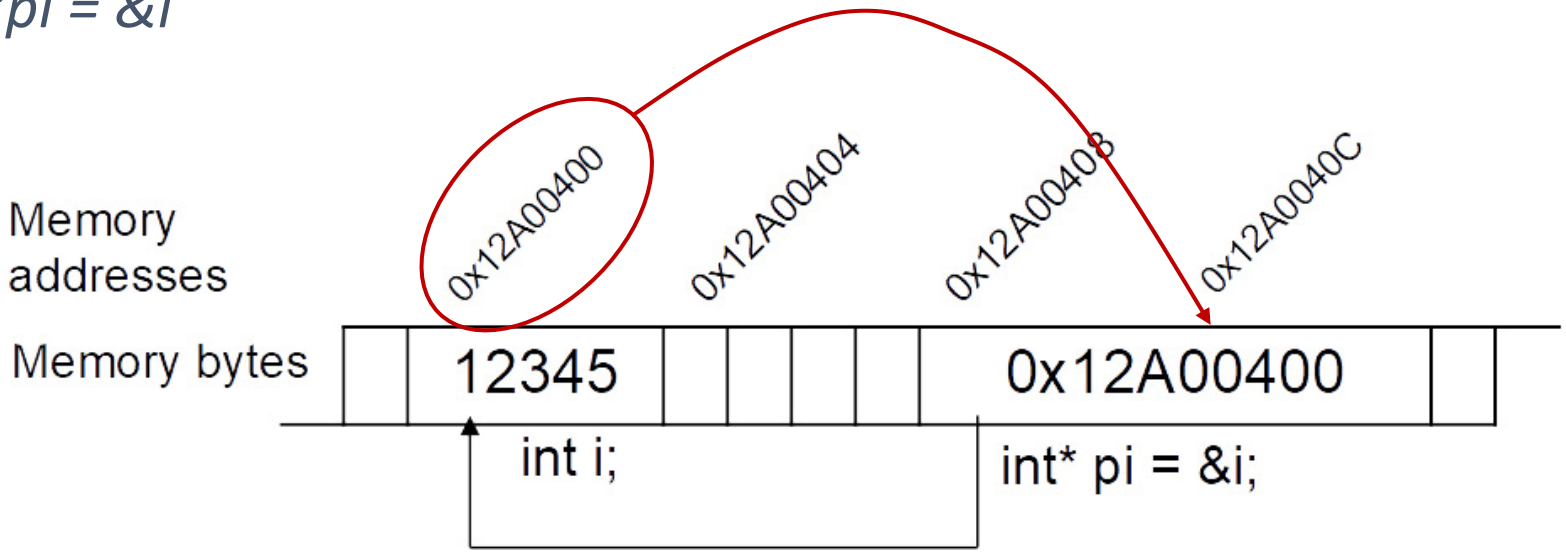
การกำหนดค่าเริ่มต้นของ pointer

- เราสามารถประกาศตัวแปร pointer พร้อมใส่ค่า address ในตัวแปร pointer ด้วยการใช้ ***address-of*** operator (&)

int i = 12345;

*int *pi = &i*





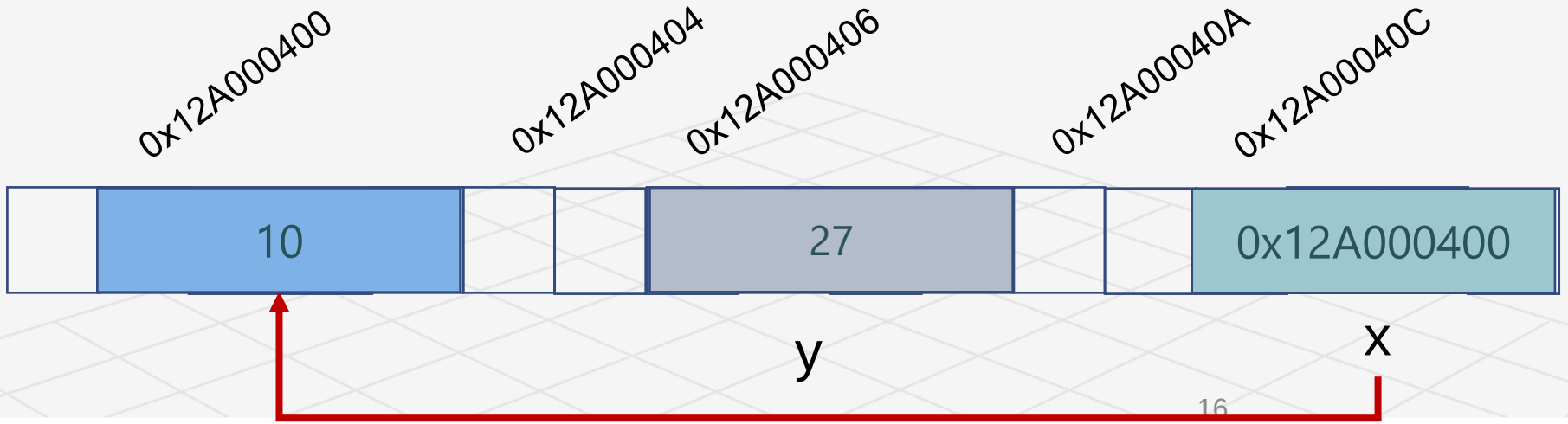
The * (dereference) operator

x is a pointer to an integer.

```
int *x, y;
```

You can use the integer kept in a memory address which x points to in a C++ expression like this:

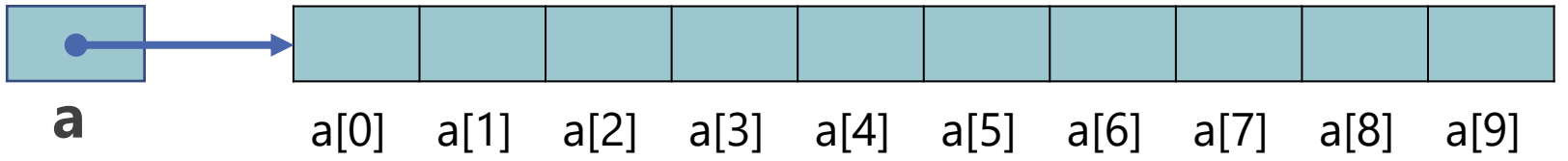
```
*x = 10;
 y = *x + 17;
```



Arrays

- address ของอะเรย์ก็คือ address ของข้อมูลตัวแรกในอะเรย์
- ชื่ออะเรย์ (array name) ที่ไม่มี index กำกับจะบอกตำแหน่งของอะเรย์
- address ของอะเรย์สามารถ assign ให้กับ pointer โดยใช้ชื่อของอะเรย์
- ชื่ออะเรย์ (array name) ไม่เป็น pointer เพราะ address ของอะเรย์ไม่สามารถเปลี่ยนแปลงได้ (address ที่เก็บใน pointer สามารถเปลี่ยนได้).

```
int a[10];
```

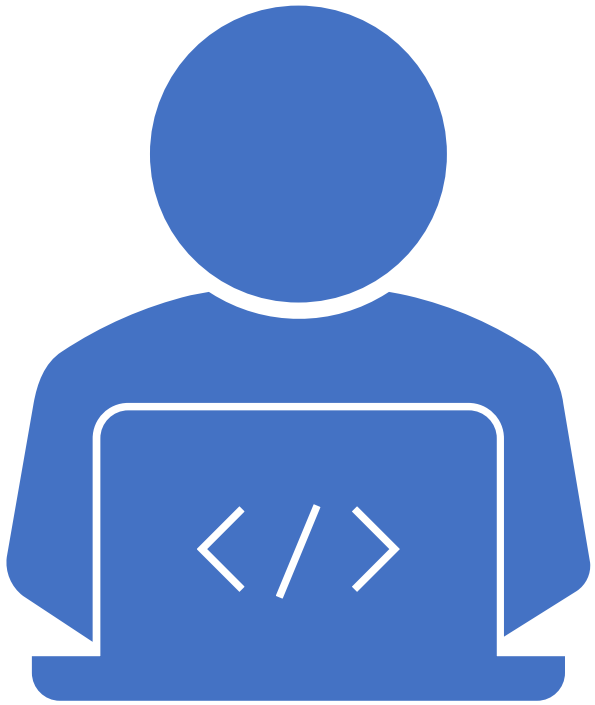


The [ ] Operator

- [ ] ใช้ระบุช่องของข้อมูลในอะเรย์ เช่น

  nums[3] ให้ค่าเช่นเดียวกับ *(nums + 3).

- การคำนวณหา address จาก nums + 3 ทำได้โดย
  - หาชนิดของข้อมูลที่ nums เก็บอยู่
  - นำขนาด (bytes) ของชนิดข้อมูลนั้นคูณด้วย 3
  - นำผลคูณที่ได้บวกกับ address ที่เก็บอยู่ใน nums
- เมื่อได้ address ที่เป็นผลลัพธ์ ก็จะเข้าไปหาข้อมูลที่ address นี้ได้



หน่วยความจำพลวัต (Dynamic Memory)

การจัดสรรแบบพลวัต (dynamic allocation) ในภาษา C/C++ หมายถึง การจองหรือขอใช้เนื้อที่หน่วยความจำโดยโปรแกรมเมอร์เป็นผู้ร้องขอเมื่อต้องการใช้เนื้อที่

The Heap

- The ***heap*** เป็นส่วนหนึ่งของ RAM ที่ถูกกันไว้สำหรับการใช้งานของโปรแกรม
- เมื่อโปรแกรมมีการใช้หน่วยความจำในส่วนของ heap เราเรียกส่วนของ heap ที่ถูกใช้ว่า ***dynamically-allocated memory***.

การขอเนื้อที่ใน Heap

- เป็นการขอเนื้อที่ระหว่างที่โปรแกรมทำงานอยู่
- มักจะใช้กับการขอเนื้อให้ strings arrays หรือ structures
- ฟังก์ชันที่ใช้ในการขอ / คืน เนื้อที่แบบ dynamic ใน heap คือ
  - **new** - ขอเนื้อที่ในหน่วยความจำ แต่ไม่ได้มีการเซตค่าเริ่มต้น
  - **delete** - คืนเนื้อที่ให้กับหน่วยความจำ

new operator

new **operator** เป็นการขอจองเนื้อที่ใน **heap**

ถ้าเนื้อที่ว่างเพียงพอก็จะจองเนื้อที่ให้ และส่งตำแหน่ง (address) ของหน่วยความจำไปให้กับตัวแปรพอยเตอร์

Syntax to use new operator

pointer-variable = **new** *data-type*;

Where :

pointer-variable is the pointer of type data-type.

data-type could be any built-in data type including array or any user defined data types including structure and class.

Examples

// Pointer initialized with NULL

// Then request memory for the variable

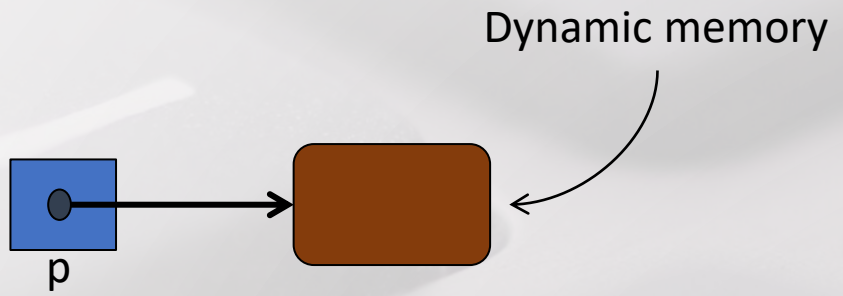
```
int *p = NULL;
p = new int;
```

OR

// Combine declaration of pointer

// and their assignment

```
int *p = new int;
```



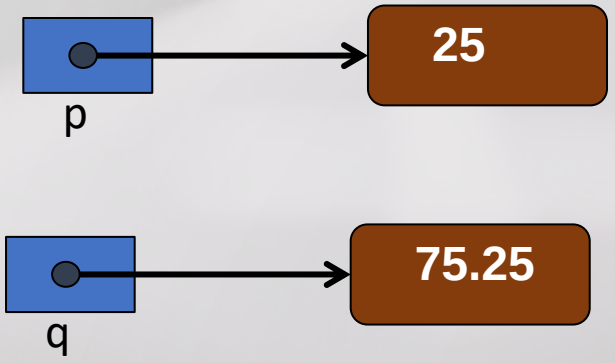
Initialize memory

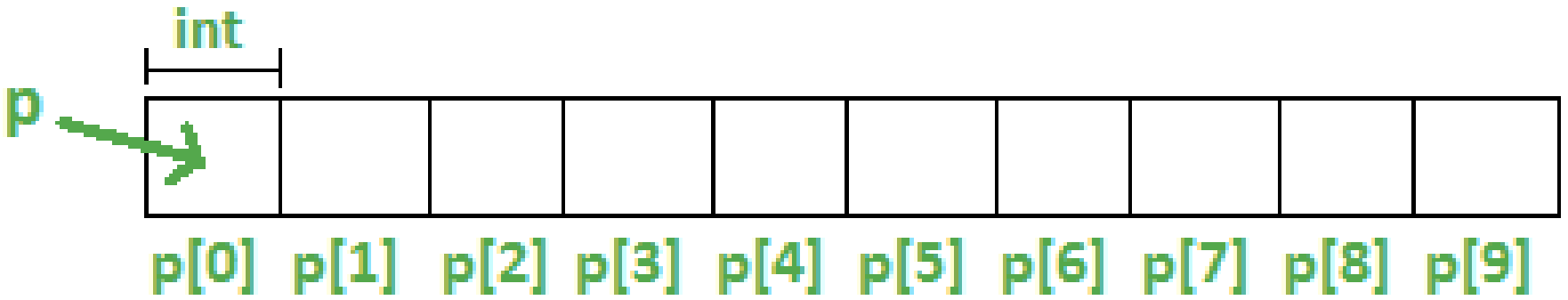
- ในการใช้ **new** operator เราสามารถกำหนดค่าเริ่มต้นให้กับตัวแปรได้ด้วยคำสั่ง

  pointer-variable = **new** data-type(value);

- **Example:**

```
int *p = new int(25);

float *q = new float(75.25);
```





การขอเนื้อที่สำหรับ Dynamic Arrays

- pointer-variable = **new** data-type[size];
- Example: int *p = new int[10]

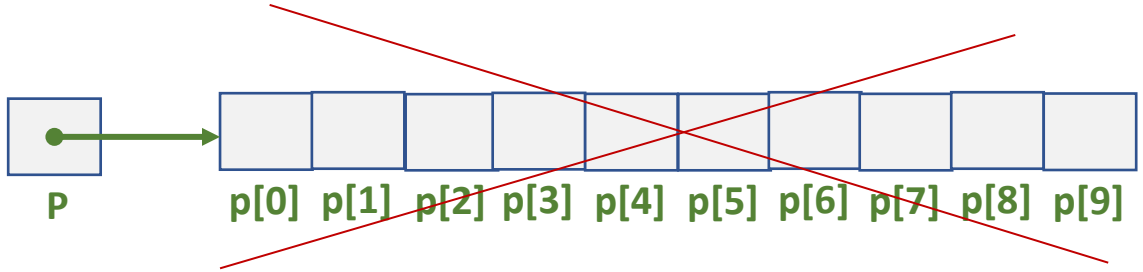
การประกาศตัวแปร array VS **new** operator ?

- Example

```
int  p[10];
int *p = new int[10];
```

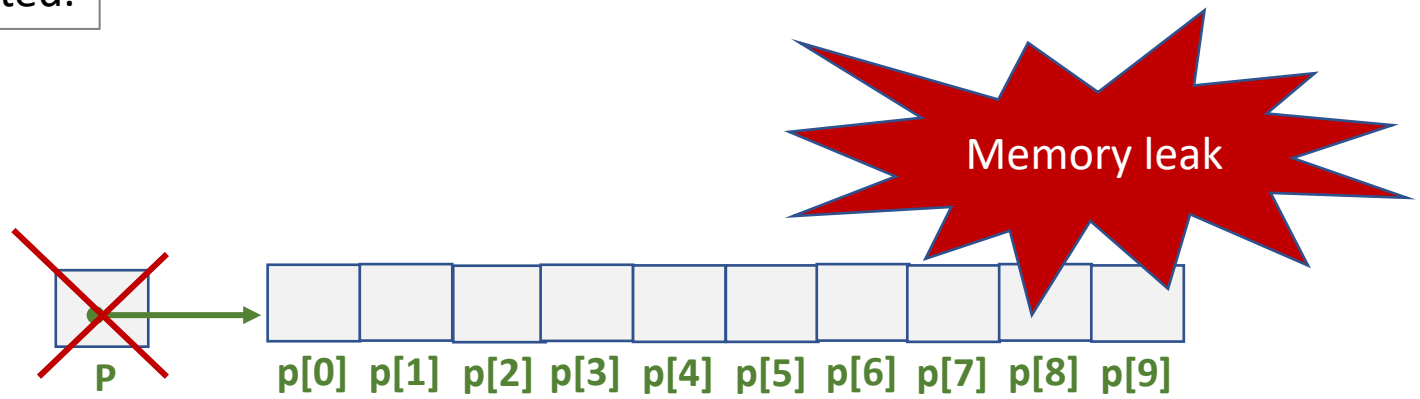
Example

```
{ int  p[10];
   :
   :
}
```



Array a is deallocated.

```
{int *p = new int[10];
   :
   :
}
```



Array p is not deallocated.

Memory Leak

- เมื่อฟังก์ชันทำงานจบ เนื้อที่ของตัวแปรทุกตัวในฟังก์ชัน และค่าที่ตัวแปรเก็บอยู่จะถูกคืนไป (p)
- address ของ dynamic array ก็หายไปด้วย ทั้งๆ ที่ dynamic array ยังไม่ถูกคืนเนื้อที่ และเนื้อที่ส่วนนี้จะไม่สามารถนำกลับมาใช้ได้อีก
- ปัญหานี้เรียกว่า ***memory leak (***จะเกิดขึ้นขณะที่โปรแกรมทำงาน และหายเมื่อโปรแกรมทำงานจบ)
- ถ้าโปรแกรมปล่อยให้เกิด memory leaks อาจทำให้พื้นที่ในส่วนของ heap memory ถูกใช้งานจนหมดและโปรแกรมไม่สามารถทำงานต่อได้
- ปัญหา Memory leak สามารถป้องกันได้

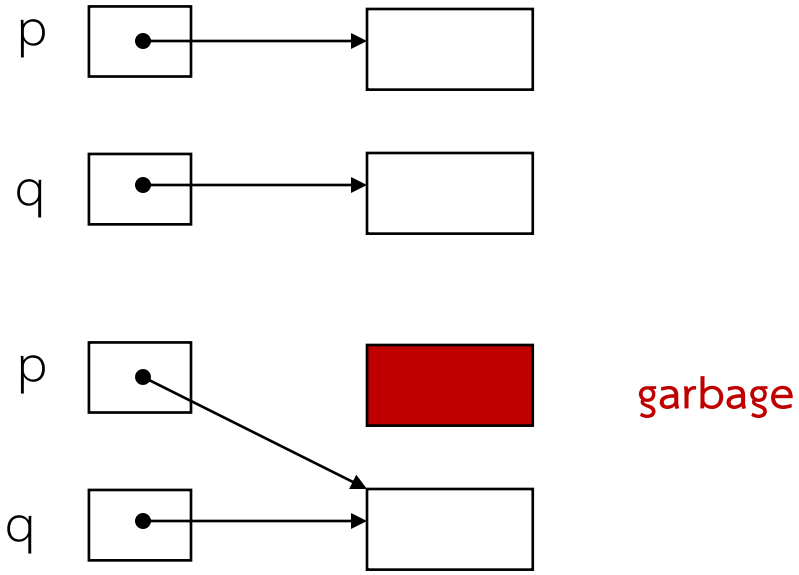
การคืนเนื้อที่ใน heap

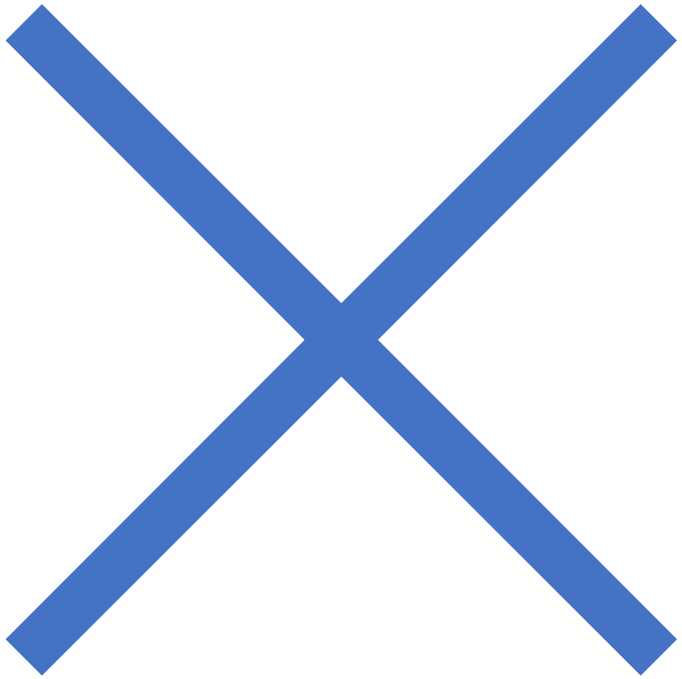
เพื่อป้องกันการเกิด *memory leak* เราควรคืนเนื้อที่ให้กับระบบเมื่อไม่ใช้งานเนื้อที่นั้นแล้ว

p = **new int**;

q = **new int**;

p = q;





delete operator

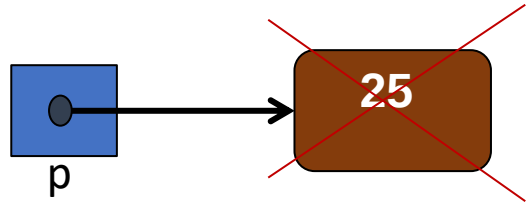
- **Syntax:**

  delete operator จะคืนเนื้อที่หน่วยความจำตำแหน่งที่ตัวแปร p ชี้อยู่ เนื้อที่ที่คืนไปนั้น ระบบจะสามารถนำกลับมาใช้ได้อีก

- Examples:

  delete p;

  delete q;



การคืนเนื้อที่สำหรับ dynamic array

// Release block of memory

// pointed by pointer-variable

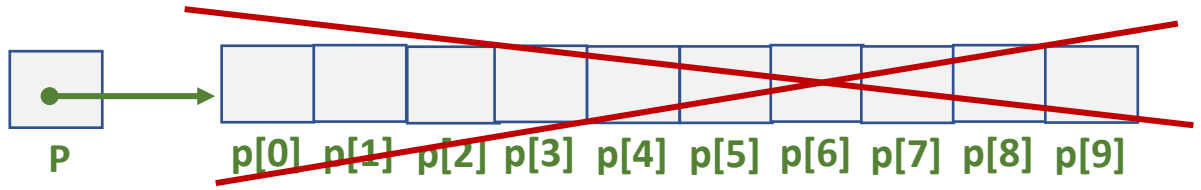
```
delete[] pointer-variable;
```

Example:

// It will free the entire array

// pointed by p.

```
Delete [] p;
```



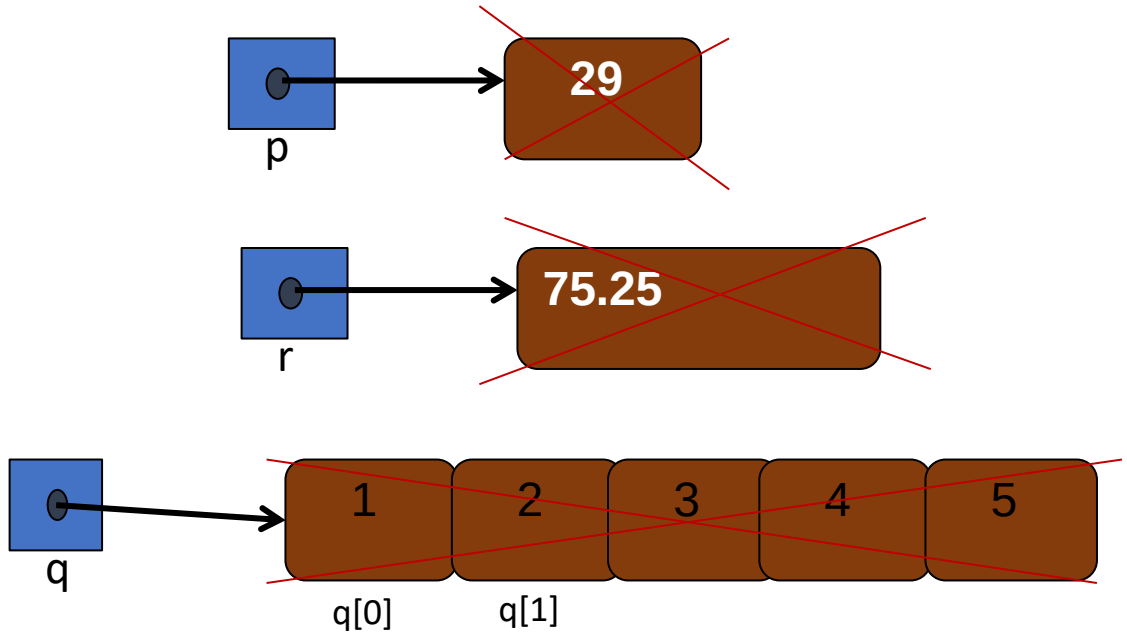
Example

```
int main () {
        // Pointer initialization to null
        int* p = NULL;

        // Request memory for the variable using new operator
        p = new(nothrow) int;
        if (!p)
                cout << "allocation of memory failed\n";
        else
        {       // Store value at allocated address
                *p = 29;
                cout << "Value of p: " << *p << endl;
        }
```

```
// Request block of memory
// using new operator
float *r = new float(75.25);

cout << "Value of r: " << *r << endl;

// Request block of memory of size n
int n = 5;
int *q = new int[n];
```

```
        for (int i = 0; i < n; i++) q[i] = i+1;

        cout << "Value store in block of memory: ";

        for (int i = 0; i < n; i++)
                cout << q[i] << " ";

        // freed the allocated memory
        delete p;
        delete r;

        // freed the block of allocated memory
        delete[] q;

        return 0;
}
```

int q[5]



การหลีกเลี่ยงปัญหา Memory Leak

เมื่อต้องการเปลี่ยน address ที่เก็บอยู่ใน pointer ให้คิดก่อนว่า address ที่เก็บอยู่ถูกใช้สำหรับ dynamically-allocated memory หรือไม่

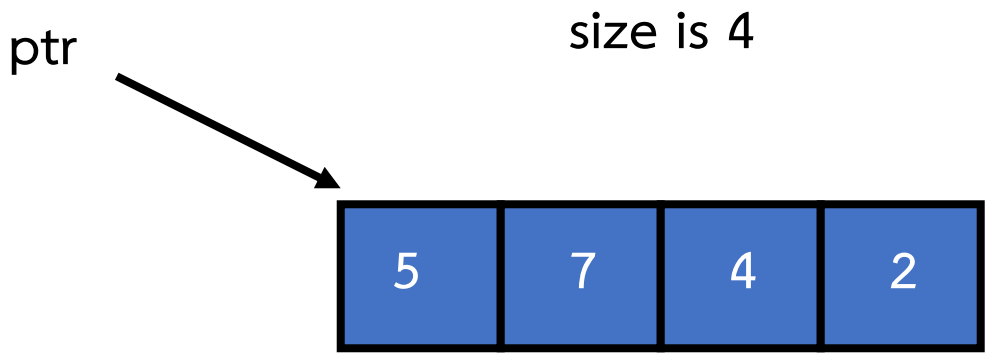
ถ้าใช่ (และเนื้อที่นั้นไม่ถูกใช้งานแล้ว) ใช้คำสั่ง delete คืนเนื้อที่ก่อนที่จะเปลี่ยนค่า address ใน pointer

ขนาดของอะเรย์ ?

ขนาดของอะเรย์ควรเป็นเท่าไร?

- ถ้ากำหนดขนาดเล็กเกินไป ก็อาจจะไม่พอ
- ถ้ากำหนดขนาดใหญ่เกินไป และเนื้อที่ถูกใช้เพียงเล็กน้อย ก็จะเสียเนื้อที่โดยเปล่าประโยชน์
- วิธีที่ดีที่สุดคือ เริ่มอะเรย์จากขนาดเล็กๆ และขยายขนาดเมื่อมีข้อมูลมากขึ้นเรื่อยๆ
- วิธีข้างต้นไม่สามารถทำได้โดยใช้อะเรย์ที่กำหนดขนาดล่วงหน้า แต่สามารถทำได้โดยใช้ pointers และการขอเนื้อที่แบบ dynamic

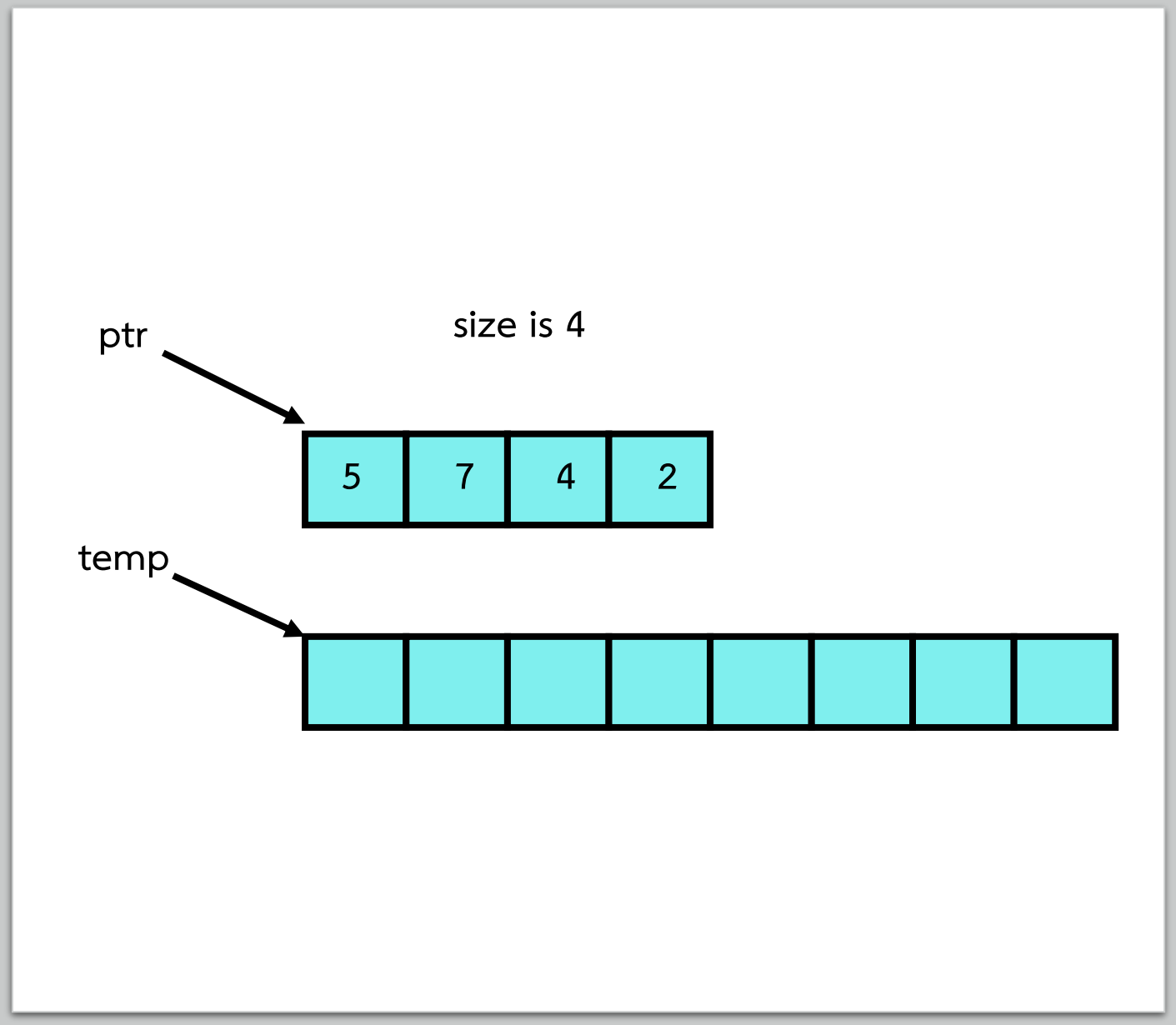
การขยายขนาดของอะเรย์



สมมติว่า array ถูกใส่ข้อมูลจนเต็ม และจะมีข้อมูลที่มาใส่อีก จึงต้องขยายขนาดของอะเรย์

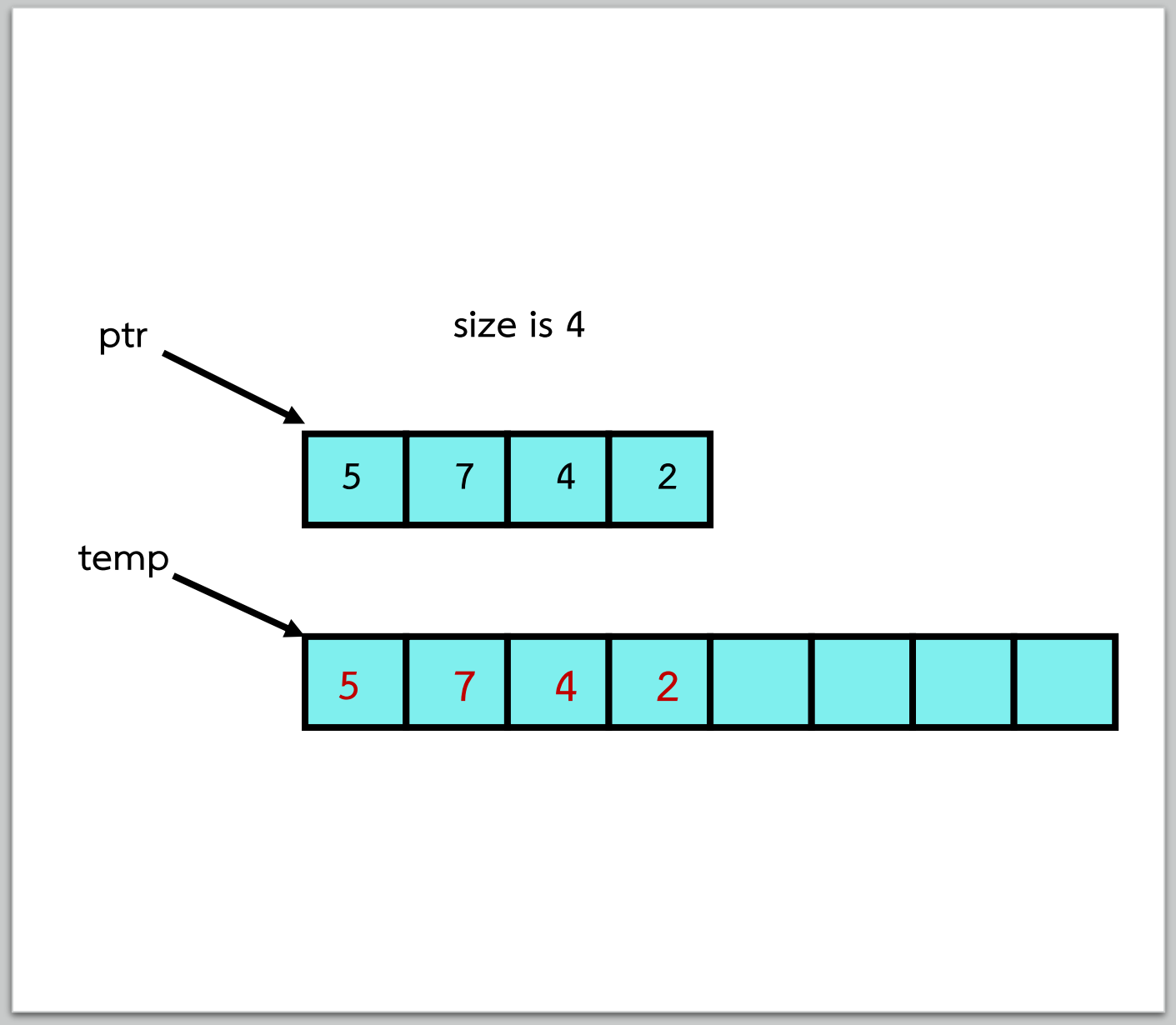
ขั้นที่ 1 : ขอเนื้อที่ใหม่ที่มีพื้นที่เป็น 2 เท่าของพื้นที่เดิม

```
int *temp = malloc(size
* 2 *sizeof(int));
```



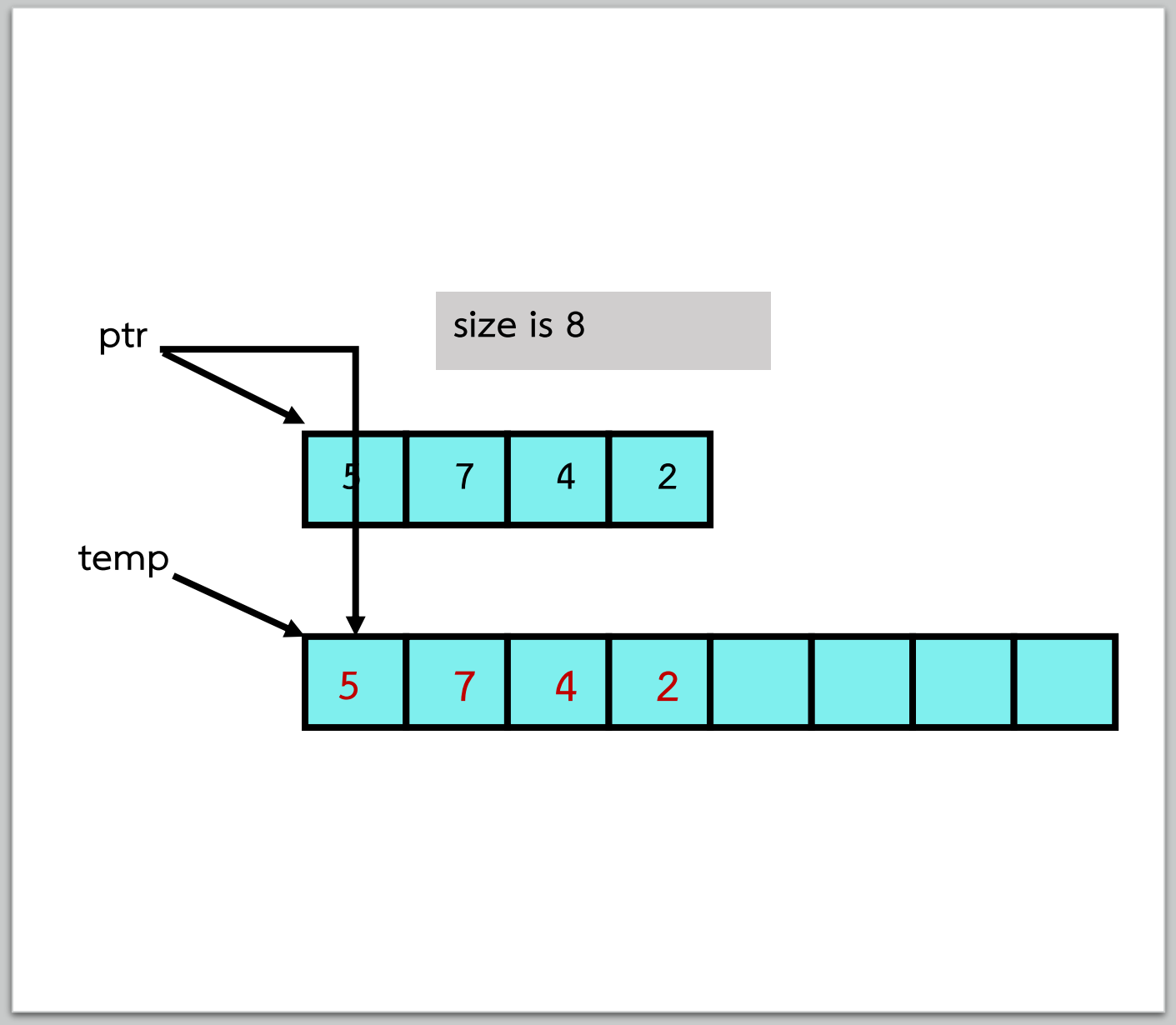
ขั้นที่ 2 : คัดลอกข้อมูลเดิมไปยังเนื้อที่ใหม่

```
for ( int i = 0; i < size; i++ )
    temp[ i ] = ptr[ i ];
```



ขั้นที่ 3 : คืนเนื้อที่ของพื้นที่เดิมให้กับ heap และเปลี่ยนขนาดของ size ใหม่

```
free(ptr);

ptr = temp;

size = size *2;
```



ข้อมูลนามธรรม (Data Abstraction)

- **Data Type**
- **Data Structure**
- **Abstract Data Type**

ชนิดของข้อมูล (Data Type)

ชนิดของข้อมูล คือ ช่วงของค่าที่ตัวแปรสามารถใช้ได้

- นอกจากชนิดของข้อมูลจะบอกถึงช่วงของข้อมูลแล้วยังบอกถึงสิ่งที่สามารถกระทำได้กับข้อมูลนั้น ๆ (operations)
- ชนิดของข้อมูลโดยทั่วไปแล้วจะแตกต่างกันในแต่ละภาษาโปรแกรม
- ตัวอย่างของชนิดของข้อมูลได้แก่

integer ชนิดของข้อมูลที่มีค่าเป็นตัวเลขจำนวนเต็ม มักจะมีอยู่ในทุกภาษาโปรแกรม

boolean ชนิดของข้อมูลที่เป็นจริงหรือเท็จมีในบางภาษาเท่านั้น

string ชนิดของข้อมูลที่เก็บข้อความหรือตัวอักษรตั้งแต่ 1ตัวขึ้นไป

โครงสร้างข้อมูล (Data Structures)

โครงสร้างข้อมูลเป็นที่เก็บข้อมูลหลาย ๆ ตัวไว้ด้วยกัน โดยที่ข้อมูลเหล่านั้นอาจจะเป็นข้อมูลที่มีชนิดเดียวกันหรือต่างชนิดกันก็ได้ มักมีอยู่ในทุกภาษาโปรแกรม ตัวอย่างเช่น

array เป็นโครงสร้างข้อมูลที่เก็บข้อมูลชนิดเดียวกันไว้ด้วยกัน

record เป็นโครงสร้างข้อมูลที่เก็บข้อมูลต่างชนิดกันหรือชนิดเดียวกันไว้ด้วยกัน

ข้อมูลนามธรรม (Abstract Data Type)

ข้อมูลนามธรรม เป็นชนิดของข้อมูลหรือโครงสร้างข้อมูลที่ไม่มีอยู่ในภาษาโปรแกรม ผู้เขียนโปรแกรมสามารถสร้างขึ้นมาเพื่อแก้ปัญหาโดยยังไม่ต้องคำนึงว่าจะเขียนเป็นโปรแกรมอย่างไร เมื่อมีการสร้างหรือกำหนดขึ้นมาผู้สร้างจำเป็นต้องกำหนดสิ่งที่สามารถจะกระทำได้ (operations) กับข้อมูลที่สร้างใหม่นี้ด้วย

Example : Set

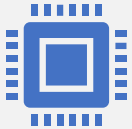
สมมติให้ A, B และ C มีชนิดเป็นเซตของตัวเลขจำนวนเต็ม

EMPTYSET(A) ทำให้เซต A กลายเป็นเซตว่าง

UNION(A,B,C) ทำการ union เซต A กับ เซต B ให้ผลลัพธ์อยู่ในเซต C

SIZE (A) ส่งกลับขนาดหรือจำนวนสมาชิกของเซต A

ทำไมต้องสร้างข้อมูลนามธรรม



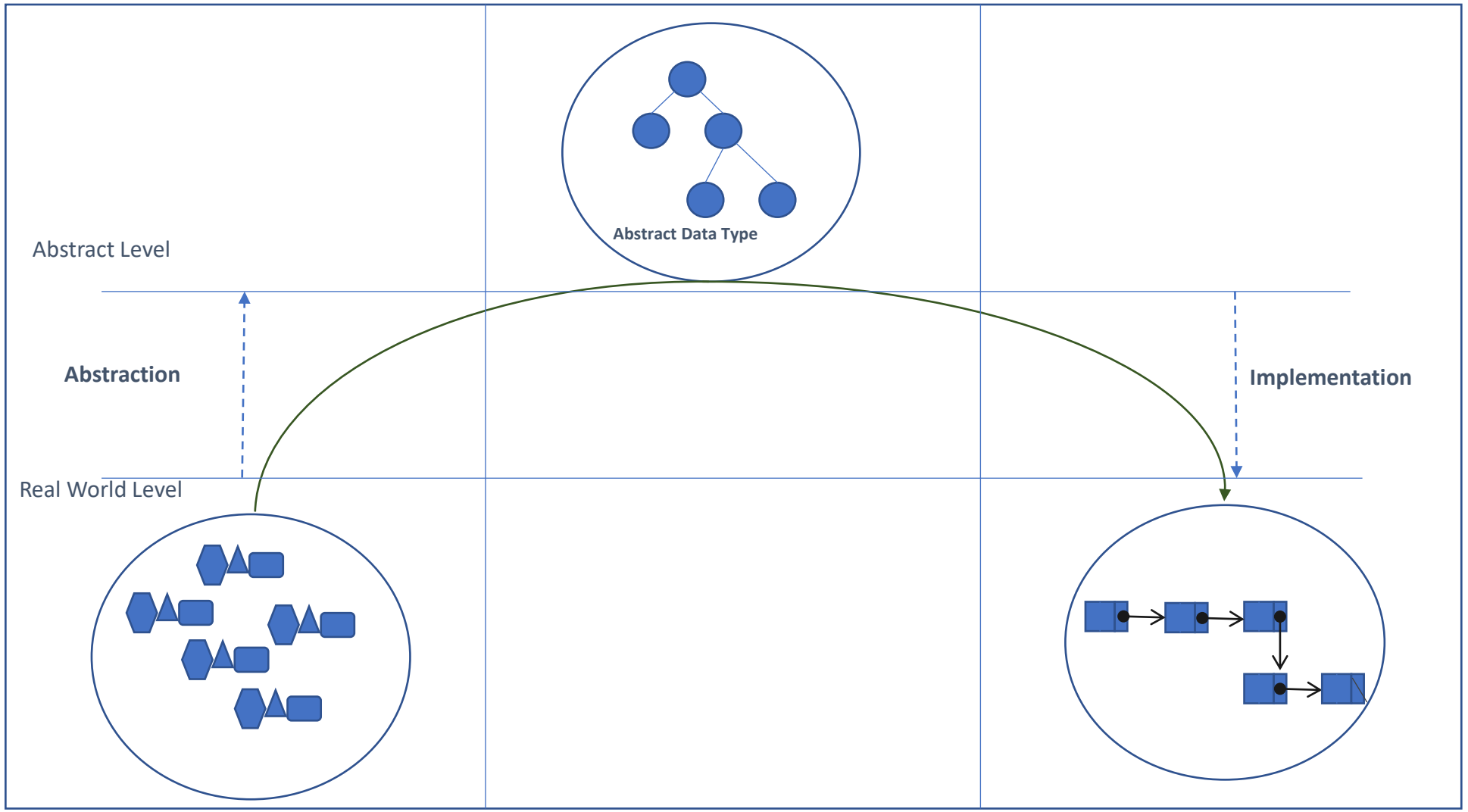
เราไม่สามารถที่จะออกแบบภาษาโปรแกรมที่ประกอบไปด้วย ชนิดของข้อมูลที่สามารถแก้ไขปัญหาได้ทุกปัญหาในโลก

ในแต่ละภาษาโปรแกรมมีเพียงชนิดของข้อมูลหรือโครงสร้างของข้อมูลหลักๆ ซึ่งบางครั้งไม่เพียงพอในการแก้ปัญหาที่ซับซ้อน

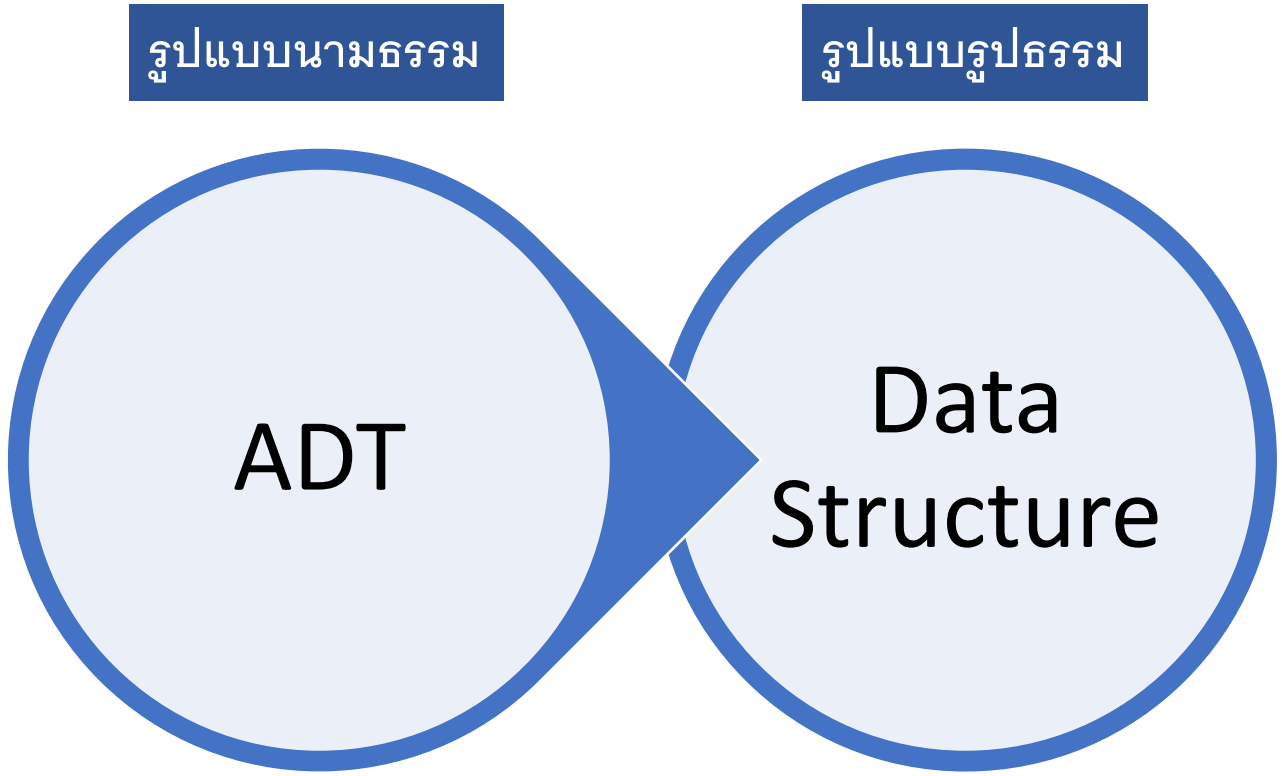
ผู้พัฒนาโปรแกรมต้องสร้างชนิดของข้อมูลพิเศษขึ้นมาใช้เอง ชนิดของข้อมูลที่สร้างขึ้นมานี้ก็คือ ข้อมูลนามธรรม (Abstract Data Type) นั่นเอง

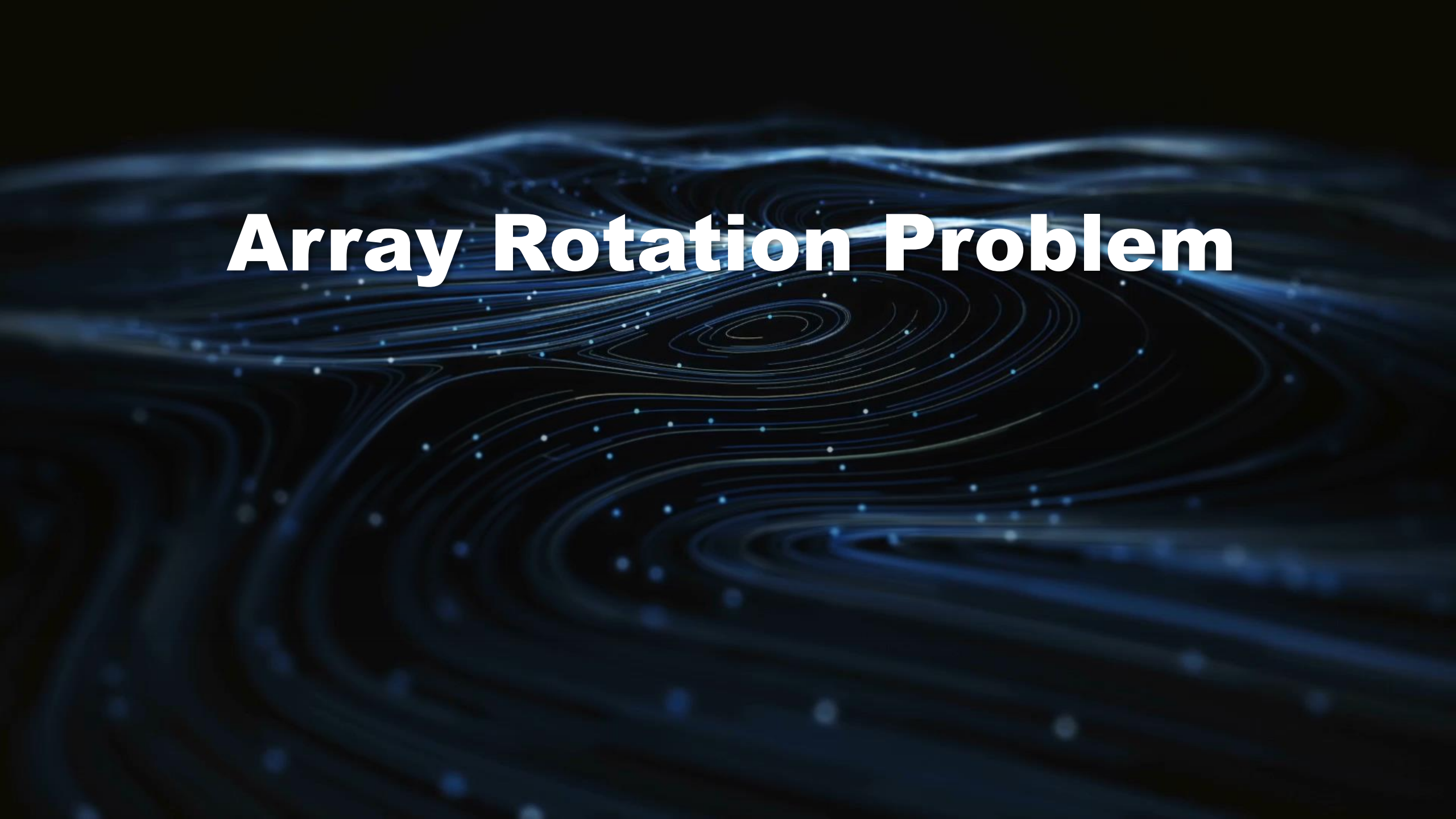
เมื่อสร้างข้อมูลนามธรรมขึ้นมาใหม่ ผู้สร้างก็ต้องกำหนด operations ที่สามารถทำงานได้กับข้อมูลชนิดนี้ขึ้นมาด้วย เช่นเดียวกันกับการสร้างชนิดข้อมูล SET ตามตัวอย่าง

Abstraction VS Implementation



ความสัมพันธ์ของ ADT และโครงสร้างข้อมูล




Array Rotation Problem

Exercise : Left Rotation

- A *left rotation* operation on an array of size ***n*** shifts each of the array's elements ***d*** unit to the left. For example, if 2 left rotations are performed on array [1, 2, 3, 4, 5, 6, 7], then the array would become [3, 4, 5, 6, 7, 1, 2].
- Given an array of ***n*** integers and a number, ***d***, perform ***d*** left rotations on the array. Then print the updated array as a single line of space-separated integers.

METHOD 1: Using temp array

Input arr[] = [1, 2, 3, 4, 5, 6, 7], d = 2, n =7

1) Store the first d elements in a temp array

temp[] = [1, 2]

2) Shift rest of the arr[]

arr[] = [3, 4, 5, 6, 7, 6, 7]

3) Store back the d elements

arr[] = [3, 4, 5, 6, 7, 1, 2]

Time complexity : O(n)

Auxiliary Space : O(d)

METHOD 2: Rotate one by one

```
leftRotate(arr[], d, n)
start
  For i = 0 to i < d
    Left rotate all elements of arr[] by one
end
```

- Let us take the same example arr[] = [1, 2, 3, 4, 5, 6, 7], d = 2
- Rotate arr[] by one 2 times
- We get [2, 3, 4, 5, 6, 7, 1] after first rotation
- and [3, 4, 5, 6, 7, 1, 2] after second rotation.

- **Time complexity :** O(n * d)
- **Auxiliary Space :** O(1)

METHOD 3: A Juggling Algorithm

Instead of moving one by one, divide the array in different sets

where number of sets is equal to **GCD** of ***n*** and ***d*** and move the elements within sets.

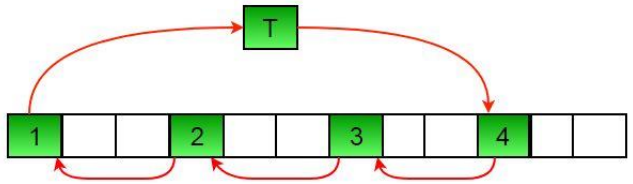
If GCD is 1 as is for the previous example array (*n* = 7 and *d* =2), then elements will be moved within one set only, we just start with temp = arr[0] and keep moving arr[I+d] to arr[I] and finally store temp at the right place.

Example: Juggling Algorithm

n =12 and d = 3. GCD is 3 and

Let arr[] be {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12}

a) Elements are first moved in first set



arr[] after this step --> {4 2 3 7 5 6 10 8 9 1 11 12}

b) Then in second set.

arr[] after this step --> {4 5 3 7 8 6 10 11 9 1 2 12}

c) Finally in third set.

arr[] after this step --> {4 5 6 7 8 9 10 11 12 1 2 3}

Time complexity : O(n)
Auxiliary Space : O(1)