บทที่ 1 แนวคิดพื้นฐาน

บทเรียนย่อย

1.1 Pseudocode

1.2 The Abstract Data type

1.3 Model for an Abstract Data Type

1.4 Abstract Data type Implementations

1.5 Document and Format

วัตถุประสงค์

- มีความรู้ ความเข้าใจเกี่ยวกับการนำรหัสเทียมมาใช้ในการพัฒนาอัลกอริทึม
- มีความรู้ ความเข้าใจเกี่ยวกับประเภทของข้อมูล รูปแบบของข้อมูล และโครงสร้างของข้อมูล
- มีความรู้ ความเข้าใจเกี่ยวกับองค์ประกอบในการดำเนินการ และการใช้งานโครงสร้างข้อมูล
- มีความรู้ ความเข้าใจเกี่ยวกับรูปแบบ และรายละเอียดของเอกสารสำหรับประกอบการดำเนินงาน
- ทบทวนคำสั่งพื้นฐานต่าง ๆ ในการเขียนโปรแกรมเกี่ยวกับโครงสร้างข้อมูล

บทที่ 1 แนวคิดพื้นฐาน

บทเรียนย่อย

1.1 Pseudocode (รหัสเทียม)

Pseudocode คือ รหัสลำลองที่ใช้เป็นตัวแทนของอัลกอริทึม โดยใช้ถ้อยคำหรือประโยคคำสั่งที่เขียนอยู่ในรูปแบบของภาษาอังกฤษหรือภาษาไทย ที่ไม่ขึ้นกับภาษาคอมพิวเตอร์ภาษาใดภาษาหนึ่ง เพื่อใช้แสดงรายละเอียดและขั้นตอนวิธีการของอัลกอริทึม โดยมีองค์ประกอบที่สำคัญ ดังนี้

- Algorithm Header
  - Name of algorithm, Purpose, Condition, and Return
- Statement Numbers
- Variables
- Statement Constructs
- Algorithm Analysis

Keyword of Algorithm Header

Keyword

- Algorithm : ชื่อของอัลกอริทึม
- Pre : ความต้องการของพารามิเตอร์ที่จะรับเข้ามาทำงาน
- Post : กิจกรรมที่เกิดขึ้น และสถานะของพารามิเตอร์
- Return : คือเงื่อนไขที่มีการส่งกลับไปหลังจากทำงาน การจัดระบบความสัมพันธ์ระหว่างข้อมูลในแบบต่าง ๆ กัน

การเขียนส่วน Statement Constructs

- การรับข้อมูล

รูปแบบ	read var1, var2, var3, ...
ตัวอย่าง	read id

- การแสดงข้อมูล

รูปแบบ	print var1, var2, var3, ...
ตัวอย่าง	print 'Name',name

การเขียนส่วน Statement Constructs [2]

- การกำหนดค่า

รูปแบบ set variable to expression/constant

เช่น

set count to 0

set average to total/2

การเขียนส่วน Statement Constructs [3]

- การคำนวณ

รูปแบบ

compute variable = expression / constant หรือ

compute add variable to expression / constant

ตัวอย่าง

compute total = num1 + num2

การเขียนส่วน Statement Constructs [4]

- การเปรียบเทียบ

  กรณีที่ 1 เปรียบเทียบระหว่างค่า 2 ค่า

รูปแบบ

```
if (condition)   then
     true statement(s)
else
     false statement(s)
end if
```

ตัวอย่าง

```
if  x > 0    then
     read  x
else
     compute   sum =  x + y
end if
```

การเขียนส่วน Statement Constructs [5]

- การเปรียบเทียบ

  กรณีที่ 2 เปรียบเทียบทางเลือกหลายทาง

รูปแบบ

```
case variable of
    a : a - statement(s)
    b : b - statement(s)
    c : c - statement(s)
end case
```

ตัวอย่าง

```
case grade of
    4 : print 'A'
    3 : print 'B'
    2 : print 'C'
end case
```

การเขียนส่วน Statement Constructs [6]

- การทำงานแบบวนซ้ำ

รูปแบบ

```
loop  (เงื่อนไข)
    action
end loop
```

ตัวอย่าง

```
loop  (number <= 10)
    number = number + 1
end loop
```

การเขียนส่วน Statement Constructs [7]

- การทำงานแบบวนซ้ำ ที่มีการทำงานคำสั่งภายในก่อนตรวจสอบเงื่อนไข

รูปแบบ

```
repeat
    statement(s)
until (condition)
```

ตัวอย่าง

```
n = 0
repeat
    read    id,name
    print   id,name
    compute   n = n + 1
until (n=5)
```

แนวทางการเขียนรหัสเทียม

- ใช้ถ้อยคำที่อ่านเข้าใจง่าย ใช้เป็นภาษาอังกฤษหรือภาษาไทยก็ได้
- ไม่ควรใช้ข้อความสั่งที่เป็นไปตามไวยากรณ์ของภาษาที่ใช้เขียนโปรแกรม
- ในหนึ่งบรรทัด ให้มีเพียงหนึ่งกิจกรรม
- ให้ใช้ย่อหน้าในการแสดงการควบคุมอย่างเป็นสัดส่วน
- แต่ละประโยคคำสั่งให้เขียนอย่างเป็นลำดับจากบนลงล่าง
- แต่ละบรรทัดควรมีหมายเลขกำกับเรียงลำดับจากน้อยไปมาก
- กรณีที่เป็นกิจกรรมย่อยให้ใช้เลขลำดับ ที่สามารถอ้างอิงถึงกิจกรรมที่เป็นข้อหลักได้
- กรณีที่มีการเรียกใช้กิจกรรมที่เป็นชุด ให้มีการเรียกใช้เป็นฟังก์ชัน

ตัวอย่าง Pseudocode การสั่งพิมพ์รายงาน

```
Algorithm sample (pageNumber)
This algorithm reads a file and prints a report.
   Pre    pageNumber passed by reference
   Post   Report Printed
          pageNumber contains number of pages in report
   Return Number of lines printed
1  loop (not end of file)
   1   read file
   2   if (full page)
       1   increment page number
       2   write page heading
   3   end if
   4   write report line
   5   increment line count
2  end loop
3  return line count
end sample
```

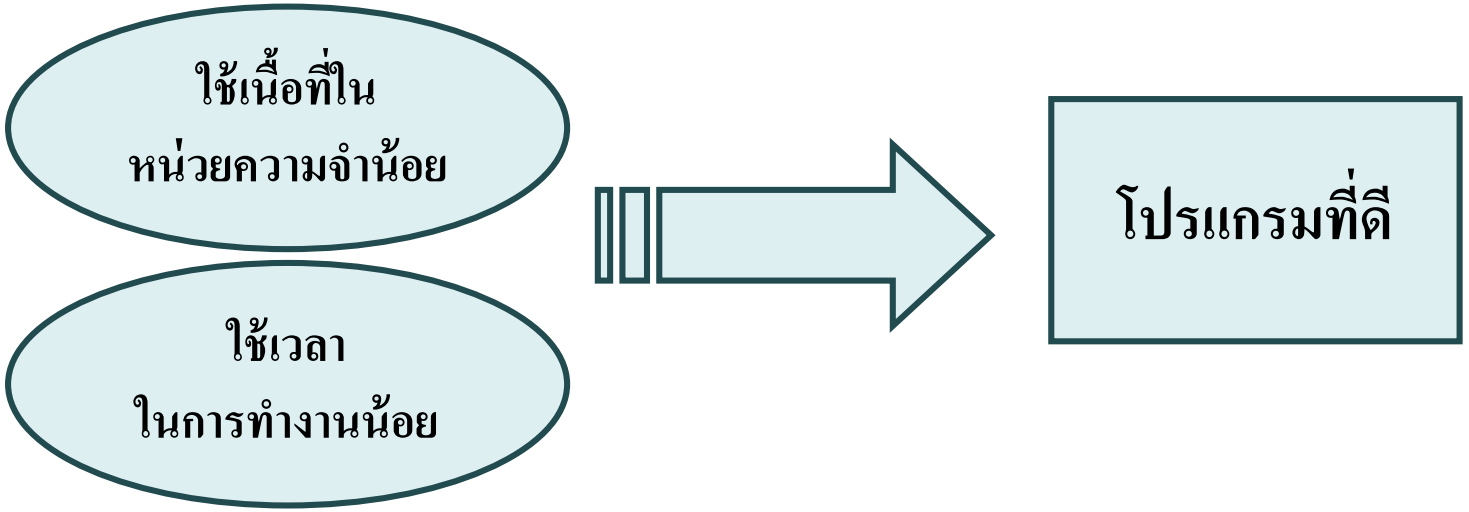
ตัวอย่าง Pseudocode การหาค่าเบี่ยงเบนในการพิมพ์รายงาน

```
Algorithm deviation
   Pre    nothing
   Post   average and numbers with their deviation printed
1  loop (not end of file)
   1   read number into array
   2   add number to total
   3   increment count
2  end loop
3  set average to total / count
4  print average
5  loop (not end of array)
   1   set devFromAve to array element - average
   2   print array element and devFromAve
6  end loop
end deviation
```

Algorithm Analysis (การประเมินประสิทธิภาพของอัลกอริทึม)

- Space/Memory : การใช้เนื้อที่ในหน่วยความจำ ซึ่งขึ้นอยู่กับการเลือกโครงสร้างข้อมูล และการแทนข้อมูล

- Time : ระยะเวลาที่ใช้ในการทำงาน ซึ่งขึ้นอยู่กับอัลกอริธึมที่เลือกใช้



บทที่ 1 แนวคิดพื้นฐาน

บทเรียนย่อย

1.1 Pseudocode

1.2 The Abstract Data type

1.3 Model for an Abstract Data Type

1.4 Abstract Data type Implementations

1.5 Document and Format

1.2 The Abstract Data type : ADT

Abstract Data type (ชนิดของข้อมูลแบบคัดย่อ) คือ การจัดการ หรือการกำหนดชนิดข้อมูลขึ้น จากการรวบรวมข้อมูลชนิดต่างๆ ข้อมูลที่มีโครงสร้าง และการดำเนินการต่างๆ ไว้ร่วมกัน ด้วยหลักการห่อหุ้มข้อมูลและซ่อนข้อมูล เพื่อให้เกิดความยืดหยุ่นในการนำไปใช้งาน และรองรับการจัดเก็บข้อมูลที่หลากหลายได้ โดยมีองค์ประกอบที่สำคัญ ดังนี้

- Atomic and Composite Data
- Data Type
- Data Structure

Atomic and Composite Data

Atomic Data (ข้อมูลเชิงเดี่ยว) คือ ข้อมูลที่ประกอบด้วยค่าเดียวที่ไม่สามารถแบ่งส่วนข้อมูลนี้ออกไปสื่อความหมายได้อีก เช่น เลขจำนวนเต็ม อายุ เพศ เป็นต้น

ตัวอย่าง เลขจำนวนเต็ม

..., -2, -1, 0, 1, 2, ...

Composite Data (ข้อมูลประกอบ) คือ ข้อมูลที่สามารถแตกออกเป็นข้อมูลย่อยได้ โดยข้อมูลที่แตกย่อยออกมาสามารถสื่อความหมายได้ เช่น เบอร์โทรศัพท์ รหัสนักศึกษา บ้านเลขที่ หมายเลขครุภัณฑ์ เป็นต้น

ตัวอย่าง รหัสนักศึกษา

57920642

Data Type

Data Type (ประเภทของข้อมูล) สามารถแบ่งออกเป็น 2 ส่วน ดังนี้

- A set of values (ชุดข้อมูล)
- A set of operations on values (ชุดของตัวดำเนินการ และเครื่องหมายที่ใช้ร่วมกับชุดข้อมูล)

ตัวอย่าง ชุดข้อมูลของประเภทข้อมูล

Type	Values	Operations
integer	-∞, ... , -2, -1, 0, 1, 2,..., ∞	*, +, -, %, /, ++, --, ...
floating point	-∞, ... , 0.0, ..., ∞	*, +, -, /, ...
character	\0, ..., 'A', 'B', ..., 'a', 'b', ..., ~	<, >, ...

Data Structure (โครงสร้างข้อมูล)

ความหมายของ Data Structure

- เป็นการจัดระบบความสัมพันธ์ระหว่างข้อมูลในแบบต่าง ๆ กัน เพื่อให้เหมาะสมกับการดำเนินการกับข้อมูลในระบบงานนั้น ๆ และเพื่อให้การใช้เนื้อที่ในหน่วยความจำหลักเกิดประโยชน์สูงสุด
- การรวมกันของข้อมูลเชิงเดี่ยวและข้อมูลเชิงประกอบเข้าด้วยกันเป็นกลุ่มพร้อมกับการกำหนดความสัมพันธ์

ตัวอย่างโครงสร้างข้อมูล

- แถวลำดับ (Array)
- เรคคอร์ด (Record)
- ชุดข้อความ (String)

ตัวอย่างการกำหนดโครงสร้างข้อมูลแบบอาร์เรย์

การกำหนดโครงสร้างข้อมูลแบบอาร์เรย์ (Array)

- ชุดข้อมูลต้องมีค่าชนิดใดชนิดหนึ่ง
- ชนิดของข้อมูลจะต้องอธิบายลักษณะข้อมูลได้โดยตรง และลำดับข้อมูลต้องมีความสัมพันธ์กับข้อมูลที่เก็บ เช่น อาร์เรย์ที่ใช้เก็บข้อมูลของเดือน โดยลำดับที่ 0 เก็บ มกราคม, ลำดับที่ 1 เก็บ กุมภาพันธ์ เป็นต้น

ประเภทของโครงสร้างข้อมูล

- Primitive data structure เป็นข้อมูลที่มีค่าเฉพาะประเภทใดประเภทหนึ่ง เช่น แบบเลขจำนวนเต็ม (integer) แบบตรรกะ (boolean) แบบตัวอักษร (character) เป็นต้น

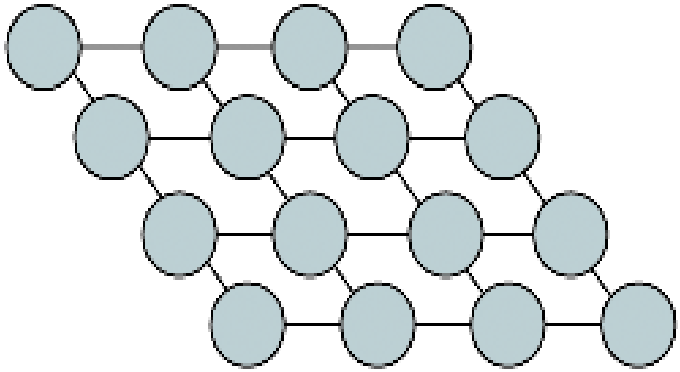
- Simple data structure เกิดจากการนำเอาข้อมูลโครงสร้างพื้นฐานประกอบขึ้นมาเป็นชุดของข้อมูลที่มีความสัมพันธ์กันในลักษะใดลักษณะหนึ่ง เช่น ข้อมูลแบบอาร์เรย์

- Compound data structure เกิดจากการนำเอาข้อมูลองค์ประกอบอย่างง่ายประกอบขึ้นเป็นข้อมูลที่มีโครงสร้างซับซ้อนขึ้น แบ่งเป็นแบบ linear structure และ nonlinear structure

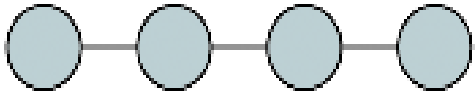
โครงสร้างข้อมูลจำแนกตามประเภท

Primitive data structure	Simple data structure	Compound data structure	
		linear	nonlinear
Integer Boolean Character	Array String Record	Stack Queue Linked - List	Graph General Tree Binary Tree

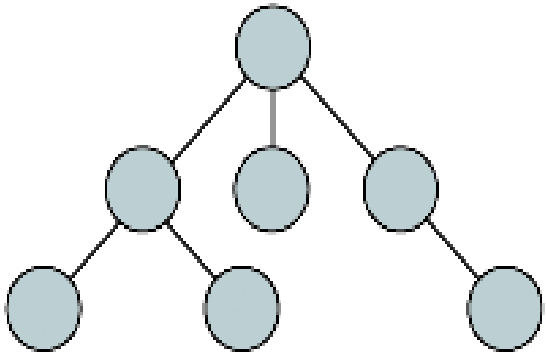
ลักษณะของโครงสร้างข้อมูล



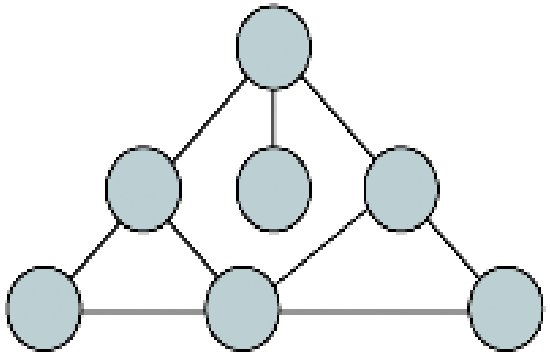
(a) Matrix



(b) Linear list



(c) Tree



(d) Graph

การดำเนินการกับข้อมูลในโครงสร้างข้อมูล

- การเพิ่มข้อมูล
- การลบข้อมูล
- การเปลี่ยนแปลงแก้ไขข้อมูล
- การค้นหาข้อมูล
- การแสดงข้อมูล
- การเรียงลำดับข้อมูล

การเลือกใช้โครงสร้างข้อมูล

- การพัฒนาโปรแกรมเพื่อให้ได้โปรแกรมที่มีประสิทธิภาพในการทำงานสูงสุด ต้องคำนึงถึงโครงสร้างข้อมูลที่ใช้

- กรณีที่โปรแกรมไม่ยุ่งยากซับซ้อนมาก ผู้เขียนโปรแกรมไม่จำเป็นต้องคำนึงถึงโครงสร้างข้อมูลมากนัก เพราะโครงสร้างข้อมูลอาจจะไม่มีผลกับประสิทธิภาพในการทำงานของโปรแกรม

- ในระบบงานใหญ่ที่มีความสลับซับซ้อนมาก ๆ ต้องคำนึงถึงโครงสร้างข้อมูลที่เลือกใช้ด้วย

ความสำคัญของโครงสร้างข้อมูล

- ลดเวลาในการประมวลผลของคอมพิวเตอร์ เพราะข้อมูลที่ถูกจัดเก็บอย่างเป็นระเบียบ และมีขั้นตอนการเข้าถึงอย่างมีระบบจะทำให้สะดวกในการจัดการกับข้อมูล

- ใช้งานสะดวก ยืดหยุ่น ในการจัดการข้อมูล และสามารถรองรับจัดการข้อมูลที่หลากหลายได้

- ช่วยลดการใช้เนื้อที่และเพิ่มประสิทธิภาพในการใช้หน่วยความจำหลัก

บทที่ 1 แนวคิดพื้นฐาน

บทเรียนย่อย

1.1 Pseudocode

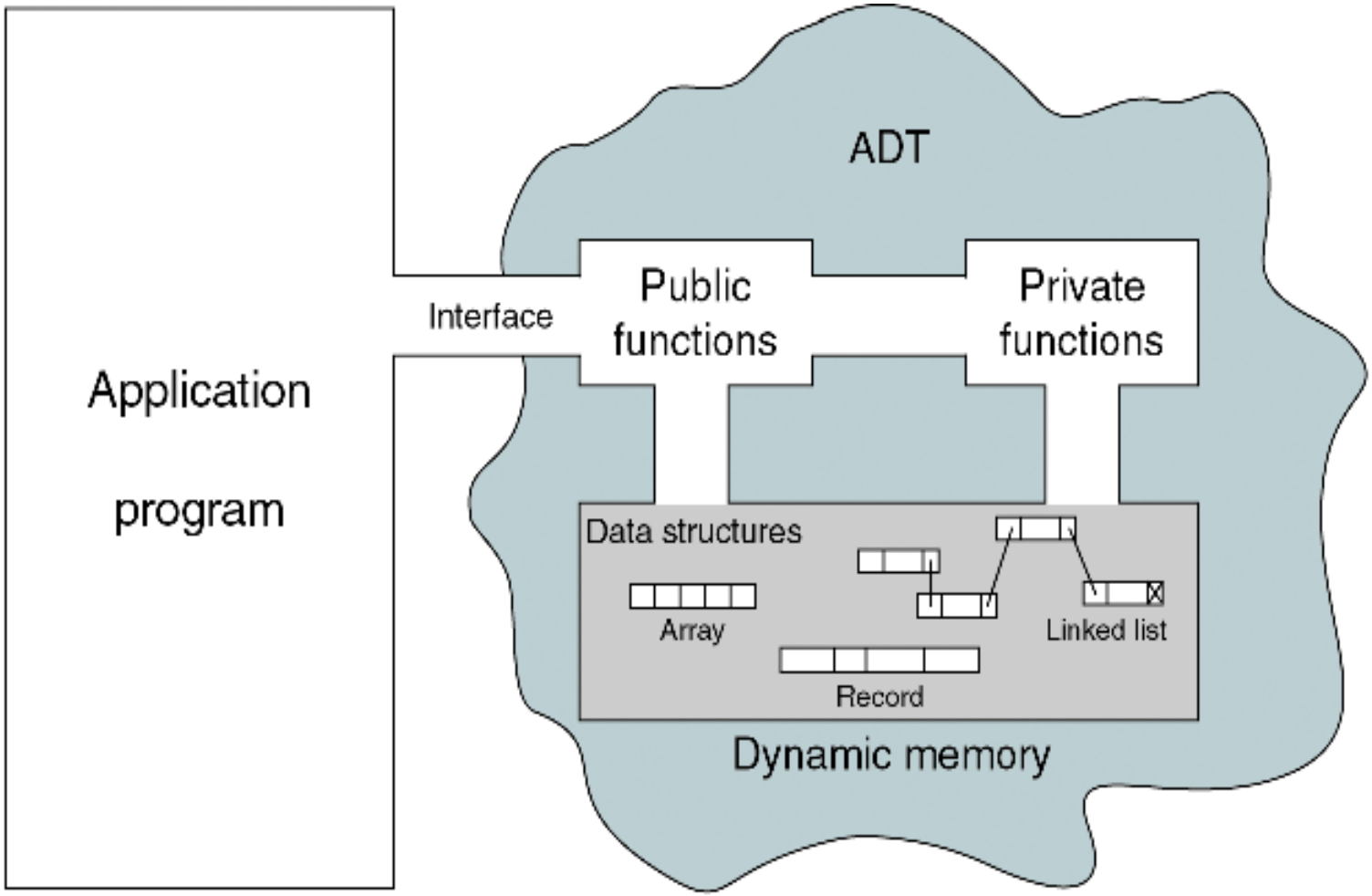
1.2 The Abstract Data type

1.3 Model for an Abstract Data Type

1.4 Abstract Data type Implementations

1.5 Document and Format

1.3 Model for an Abstract Data Type



บทที่ 1 แนวคิดพื้นฐาน

บทเรียนย่อย

1.1 Pseudocode

1.2 The Abstract Data type

1.3 Model for an Abstract Data Type

1.4 Abstract Data type Implementations

1.5 Document and Format

1.4 Abstract Data type Implementations

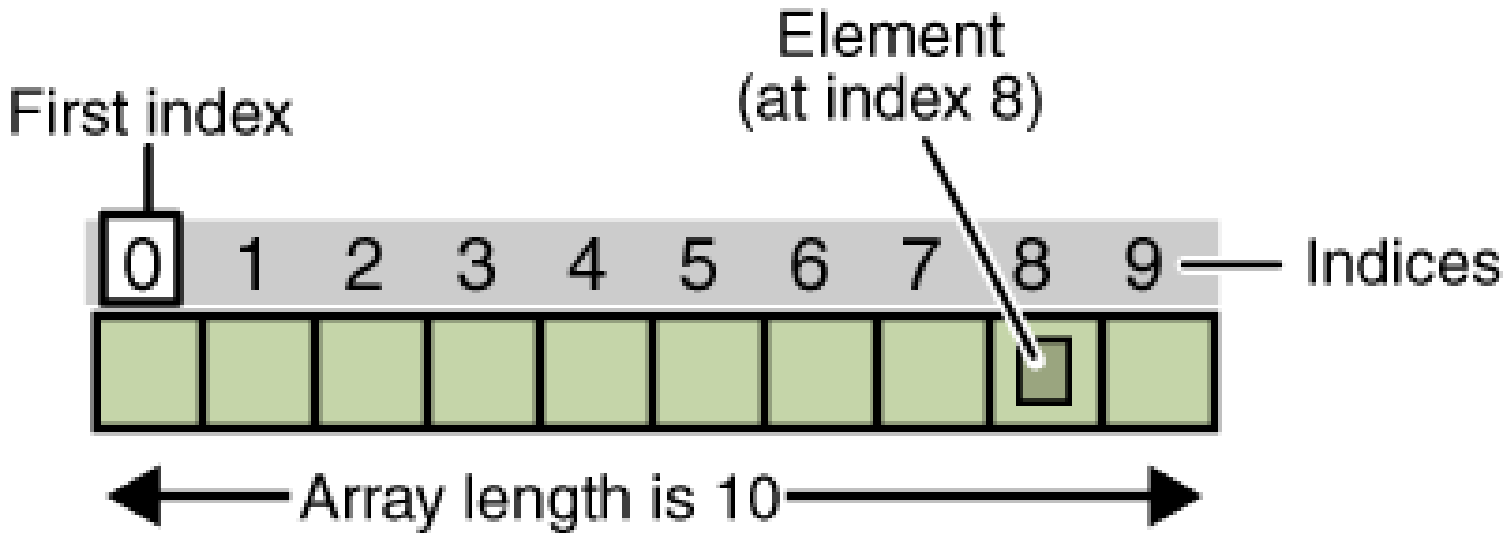
ADT Implementations (การดำเนินการกับชนิดของข้อมูลแบบคัดย่อ)

ในการดำเนินการจะใช้โครงสร้างข้อมูลพื้นฐาน 2 ชนิด ดังนี้

- Arrays
- Linked Lists

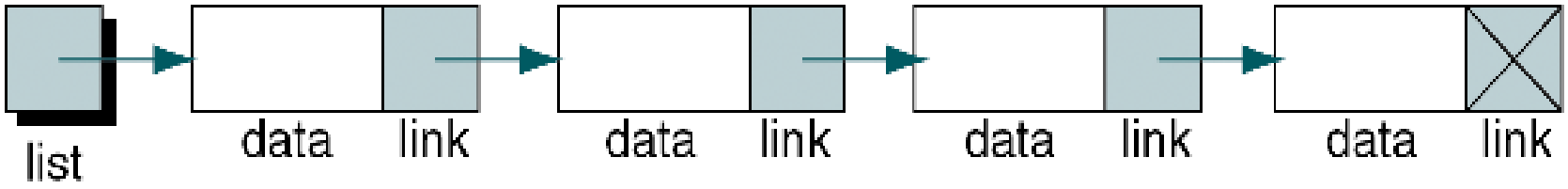
ลักษณะของ Array

Array มีการเก็บข้อมูลประเภทเดียวกันแบบเป็นลำดับได้ โดยข้อมูลนั้นจะอยู่ในตัวแปรตัวเดียวกัน

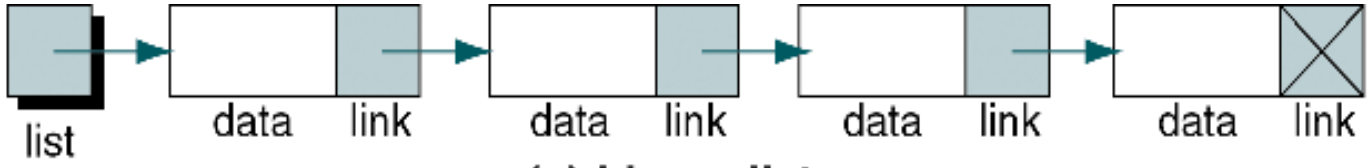


ลักษณะของ Linked List

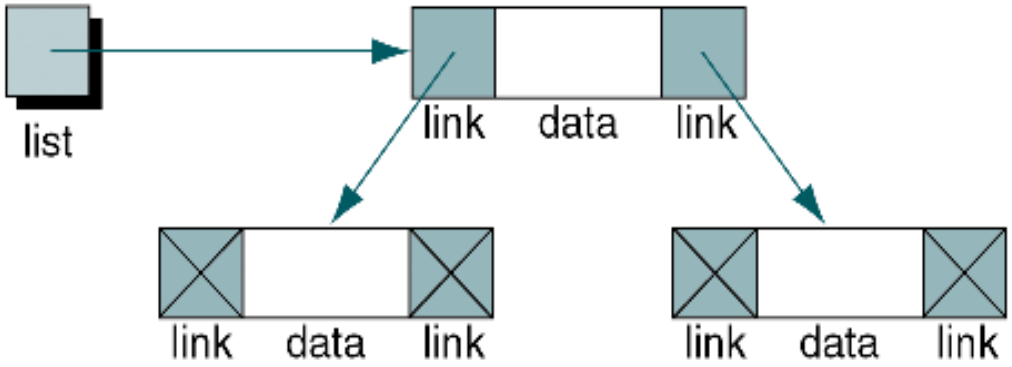
Linked List เป็นโครงสร้างข้อมูลแบบไดนามิก โดยที่ขนาดของมันสามารถเปลี่ยนแปลงได้ โดยสมาชิกแต่ละตัวของลิงค์ลิสต์จะถูกเรียกว่าโหนด (Node) ซึ่งประกอบด้วย 2 ส่วนคือ ส่วนของข้อมูล (Data) และส่วนที่เป็นตำแหน่งที่อยู่ของโหนดตัวต่อไปในลิงค์ลิสต์ (Next Address) หรืออาจเรียกว่า พอยเตอร์ (Pointer)



ตัวอย่าง Linked List แบบต่างๆ



(a) Linear list



(b) Non-linear list

(c) Empty list

บทที่ 1 แนวคิดพื้นฐาน

บทเรียนย่อย

1.1 Pseudocode

1.2 The Abstract Data type

1.3 Model for an Abstract Data Type

1.4 Abstract Data type Implementations

1.5 Document and Format

1.5 Document and Format

การเรียนการสอนในรายวิชานี้ ตลอดภาคการศึกษา จะมีการฝึกทำภาคปฏิบัติ และการส่งโครงงานย่อย ซึ่งมีข้อกำหนดที่ต้องดำเนินการ ดังนี้

- ทำเอกสารประกอบการดำเนินโครงงาน
  - เอกสารการวิเคราะห์และออกแบบ
  - เอกสารแสดงรายละเอียดผลลัพธ์
  - เอกสารการวิเคราะห์และประเมินผลลัพธ์
- เขียนและส่งไฟล์โปรแกรม
  - เขียน Comment
  - จัดสัดส่วนของ Source Code ให้เรียบร้อยและสวยงาม

PART II

Review Programming

บทที่ 1 แนวคิดพื้นฐาน

บทเรียนย่อย

1.6 Loops Review

1.7 Array Review

1.8 Recursion Review

1.6 Loops Review

Loop (ลูป) คือ เป็นคำสั่งสำหรับการเขียนโปรแกรมเพื่อให้เกิดการทำงานแบบวนซ้ำตามจำนวนรอบที่ต้องการ ซึ่งเป็นกระบวนการหนึ่งที่สำคัญในการออกแบบอัลกอริทึม โดยสามารถแบ่งออกเป็น 2 ประเภท ตามลักษณะการทำงานดังนี้

1. Pre Test (ตรวจสอบเงื่อนไขก่อนดำเนินการ)
   - คำสั่ง for loop
   - คำสั่ง while loop
2. Post Test (ดำเนินการก่อนตรวจสอบเงื่อนไข)
   - คำสั่ง do – while loop

ตัวอย่างคำสั่ง for loop

รูปแบบ

for(ระบุค่าเริ่มต้น ; ระบุเงื่อนไข ; ปรับปรุงค่าตัวควบคุม)

ตัวอย่าง

```
#include <iostream>
using namespace std;

int main () {
   // for loop execution
   for( int i = 10; i < 20; i = i + 1 ) {
      cout << "value of i: " << i << endl;
   }

   return 0;
}
```

ผลลัพธ์

```
value of i: 10
value of i: 11
value of i: 12
value of i: 13
value of i: 14
value of i: 15
value of i: 16
value of i: 17
value of i: 18
value of i: 19
```

ตัวอย่างคำสั่ง while loop

รูปแบบ

```
ระบุค่าเริ่มต้น ;
while( ระบุเงื่อนไข )
{
        statement ...
        ปรับปรุงค่าตัวควบคุม ;
}
```

ตัวอย่างคำสั่ง while loop [2]

ตัวอย่าง

```
#include <iostream>
using namespace std;

int main () {
    int i = 10;
    // while loop execution
    while( i < 20 ) {
        cout << "value of i: " << i << endl;
        i = i + 1;
    }
    return 0;
}
```

ผลลัพธ์

```
value of i: 10
value of i: 11
value of i: 12
value of i: 13
value of i: 14
value of i: 15
value of i: 16
value of i: 17
value of i: 18
value of i: 19
```

ตัวอย่างคำสั่ง do - while loop

รูปแบบ

```
ระบุค่าเริ่มต้น ;
do
{
        statement ...
        ปรับปรุงค่าตัวควบคุม ;

} while( ระบุเงื่อนไข )
```

ตัวอย่างคำสั่ง do - while loop [2]

ตัวอย่าง

```
#include <iostream>
using namespace std;

int main () {
    int i = 10;
    // do - while loop execution
    do
    {
        cout << "value of i: " << i << endl;
        i = i + 1;
    }while( i < 20 );
    return 0;
}
```

ผลลัพธ์

```
value of i: 10
value of i: 11
value of i: 12
value of i: 13
value of i: 14
value of i: 15
value of i: 16
value of i: 17
value of i: 18
value of i: 19
```

บทที่ 1 แนวคิดพื้นฐาน

บทเรียนย่อย

1.6 Loops Review

1.7 Array Review

1.8 Recursion Review

1.7 Array Review

Array (อาเรย์) คือ ประเภทของข้อมูลที่สามารถเก็บข้อมูลประเภทเดียวกันแบบเป็นลำดับได้ โดยข้อมูลนั้นจะอยู่ในตัวแปรตัวเดียวกันที่เรียกว่าตัวแปรอาเรย์ โดยใช้ index ในการเข้าถึงข้อมูล ซึ่งถือว่าเป็นข้อมูลที่มีโครงสร้างชนิดหนึ่ง

ตัวอย่าง Array 1 มิติ

```
int a[10] = {4, 21, 36, 14, 62, 91, 8, 22, 7, 81};
```

a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
4	21	36	14	62	91	8	22	7	81

1.7 Array Review [2]

ตัวอย่าง Array 2 มิติ

array [row][column] : int a[3][4] ;

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

ตัวอย่างคำสั่งรับค่าและแสดงผลของ Array 1 มิติ

```
#include <iostream>
using namespace std;

int main () {
   int a[ 10 ]; // a is an array of 10 integers

   // initialize elements of array n to 0 and input values
   for ( int i = 0; i < 10; i++ ) {
      cout << "Input value of array " << i << " : ";
      cin >> a[i];
   }

   // display values
   for ( int i = 0; i < 10; i++ ) {
      cout << "Value of array " << i << " : " << a[i];
   }

   return 0;
}
```

ตัวอย่างคำสั่งรับค่าและแสดงผลของ Array 2 มิติ

```
#include <iostream>
using namespace std;

int main () {
   int a[3][4]; // a is an array of integers 3 row 4 column

   // input values
   for ( int i = 0; i < 3; i++ ) {
        for(int j = 0; i < 4; i++ ){
            cout << "Input value of array " << i << "-" << j << " : ";
            cin >> a[i][j];
        }
   }

   // display values
   for ( int i = 0; i < 3; i++ ) {
        for(int j = 0; i < 4; i++ ){
            cout << "Value of array " << i  << "-" << j << " : " << a[i][j];
        }
   }

   return 0;
}
```

บทที่ 1 แนวคิดพื้นฐาน

บทเรียนย่อย

1.6 Loops Review

1.7 Array Review

1.8 Recursion Review

1.8 Recursion Review

Recursion (การเวียนเกิด) คือการใช้คำสั่งแบบการทำซ้ำโดยการเรียกฟังก์ชันตัวเอง โดยมีขั้นตอนที่สำคัญดังนี้

- กำหนด Base case ซึ่งเป็นส่วนสำคัญของการเขียนโปรแกรมแบบ recursion
- การเรียกใช้งานฟังก์ชันทุกครั้งจะต้องนำไปสู่ Base case ไม่เช่นนั้นจะเกิดการเรียกตัวเองโดยไม่หยุด

ตัวอย่างการเขียนฟังก์ชันหาค่า Factorial แบบ Recursion

Base Case : Factorial(1) = 1 และ Factorial(0) = 1

```
int factorial(int number) {
    int temp;

    if(number <= 1) return 1;

    temp = number * factorial(number - 1);
    return temp;
}
```

ตัวอย่างการคำนวณค่า Factorial แบบ Recursion

เป็นการคำนวณค่า Factorial(3) โดยรูปทางฝั่งซ้ายแสดงให้เห็นถึงการลดค่าเลขจำนวนเต็มจาก 3 ลงทีละหนึ่ง จนกระทั่งถึงค่า 0 ซึ่งจะมีการคืนค่าเป็น “1” แล้วจึงนำไปแทนค่าทีละขั้นในรูปทางฝั่งขวามือ จนได้ค่าสุดท้ายของ Factorial(3) = 6