บทที่ 11

โครงสร้างข้อมูล

วัตถุประสงค์

หลังจากเรียนจบบทที่ 11 แล้ว นักศึกษาต้องสามารถ:

- อธิบายความหมายและประโยชน์ของโครงสร้างข้อมูลแบบ array
- อธิบายความแตกต่างระหว่าง record และ array
- เข้าใจแนวคิดของ linked list และความแตกต่างระหว่าง array กับ linked l
- เข้าใจว่าเมื่อไรจึงจะใช้ array และเมื่อไรจึงจะใช้ linked-list

โครงสร้างข้อมูล

- ในบทต้นๆเราได้ใช้ตัวแปร (variables) เก็บข้อมูล 1 ค่า ถึงแม้ว่าตัวแปรจะมีการใช้ในการเขียนโปรแกรมอย่างกว้างขวาง แต่การใช้ตัวแปร 1 ตัวแทนค่าข้อมูลเพียง 1 ค่าไม่สามารถใช้แก้ปัญหาที่มีความซับซ้อนสูงๆได้อย่างมีประสิทธิภาพ
- ในบทนี้เป็นการแนะนำโครงสร้างข้อมูลที่ใช้เก็บข้อมูลที่ซับซ้อน โดยโครงสร้างข้อมูลจะใช้กลุ่มของตัวแปรที่มีความสัมพันธ์กัน สารถเข้าถึงครั้งละตัว หรือเข้าถึงครั้งละทั้งหมด กล่าวอีกอย่างหนึ่งคือโครงสร้างข้อมูลเป็นโครงสร้างที่แทนกลุ่มของข้อมูลที่มีความสัมพันธ์กันตามวัตถุประสงค์ของการใช้

โครงสร้างข้อมูล

- เราจะศึกษาโครงสร้างข้อมูล 3 แบบคือ

  1. อะเรย์ (Array)

  2. ระเบียน (Record)

  3. ลิงค์ลิสท์

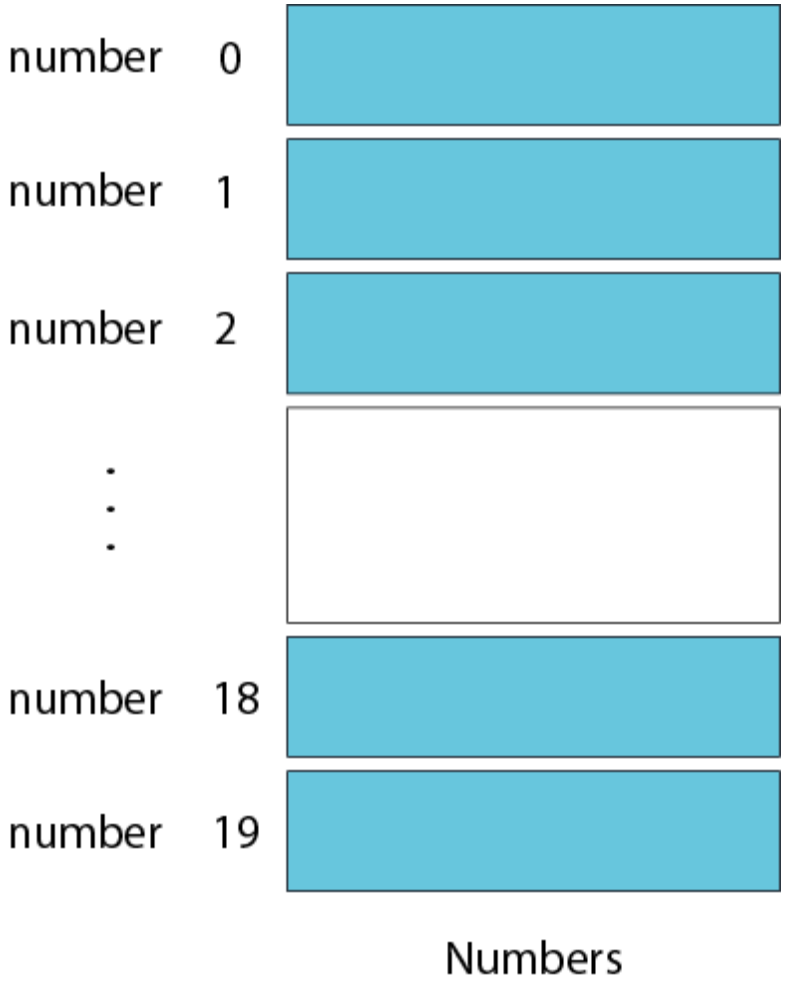
  โปรแกรมคอมพิวเตอร์ส่วนมากจะใช้โครงสร้างข้อมูล 2 ประเภทแรก ส่วนลิงค์ลิสท์จะทำการจำลองแบบโดยใช้ ตัวชี้ (pointers) และ ระเบียน

11.1

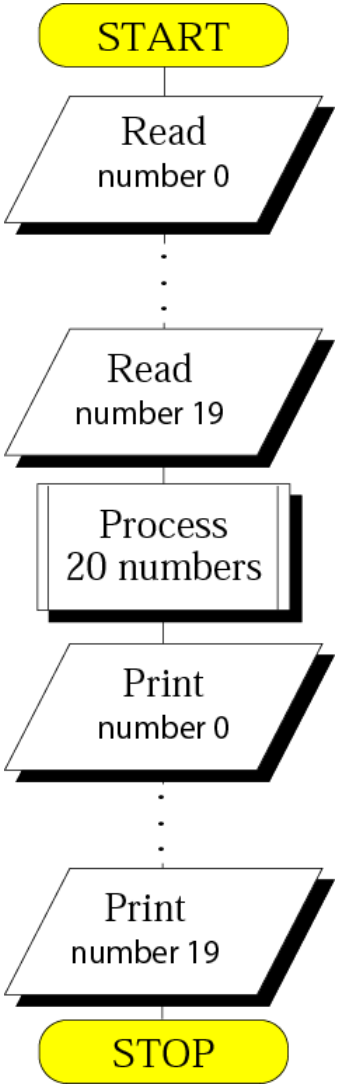
ARRAYS

อะเรย์

- สมมติว่าท่านมีโจทย์ปัญหาที่จำเป็นต้องประมวลผลตัวเลข 20 จำนวน ท่านต้องอ่านเลขทั้ง 20 จำนวน ประมวลผลทั้ง 20 จำนวน และพิมพ์ผลลัพธ์ ยิ่งกว่านั้น ท่านจะต้องเก็บเลขทั้ง 20 จำนวนไว้ในหน่วยความจำตลอดช่วงเวลาการประมวลผลของโปรแกรม ท่านอาจกำหนดตัวแปร 20 ตัวเพื่อเก็บตัวละค่า แต่ละตัวมีชื่อที่แตกต่างกัน ดังรูปที่ 11.1
- การตั้งชื่อตัวแปร 20 ตัวที่แตกต่างกันมีความลำบากและก่อให้เกิดปัญหาที่ตามาอีกอย่างหนึ่งคือ เราจะอ่านข้อมูลจากคีย์บอร์ดและนำเข้าไปเก็บอย่างไร? การอ่านเลขจำนวน 20 จำนวนต้องใช้การอ้างอิงรวม 20 ครั้ง ตัวแปรละ 1 ครั้ง นอกจากนี้การพิมพ์ก็ต้องมีการอ้างอิงอีก 20 ครั้ง ดังรูปที่ 11.2



รูปที่ 11-1 ตัวแปรที่แตกต่างกัน 20 ตัว



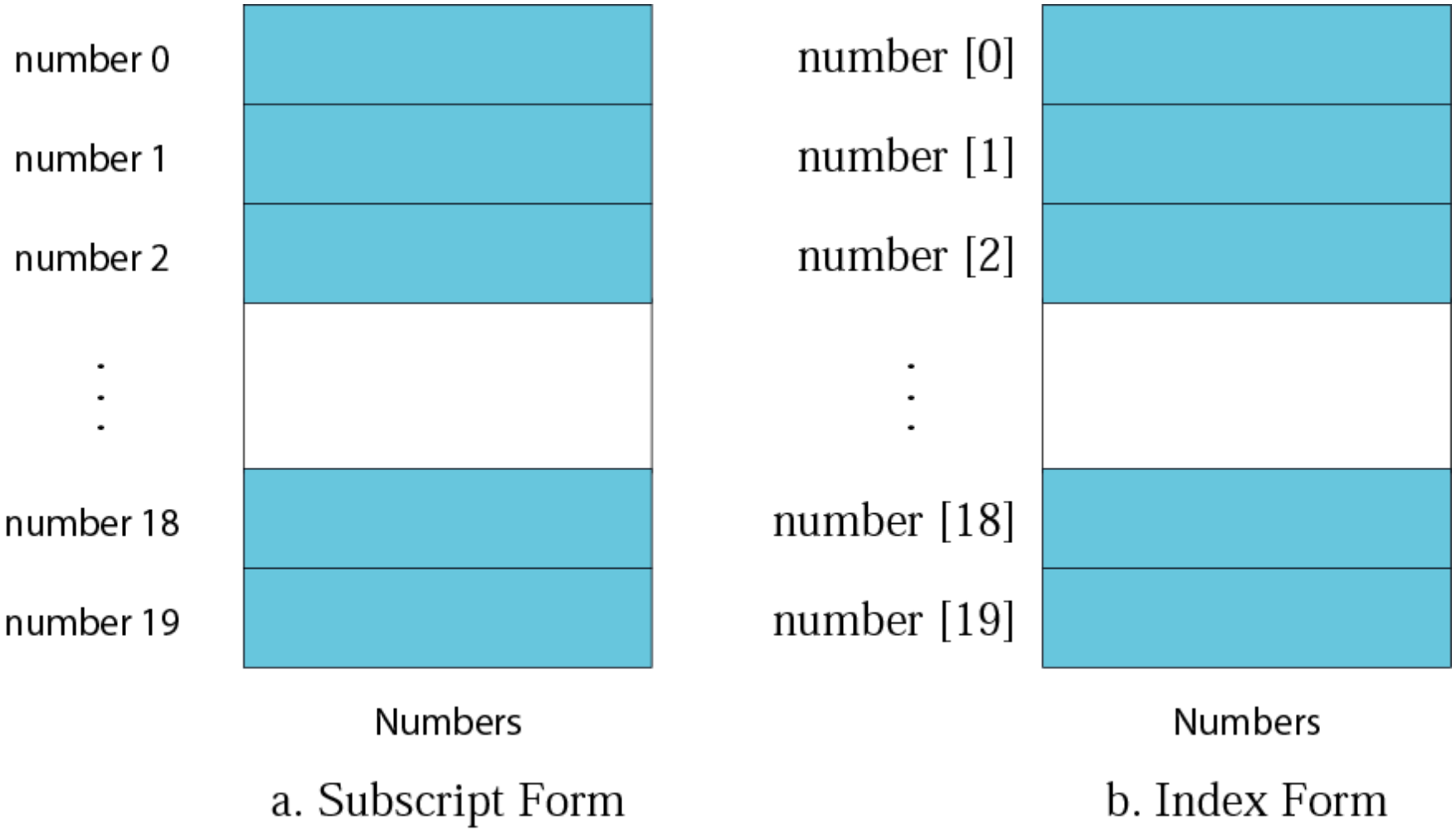
รูปที่ 11-2 การประมวลผลตัวแปรทีละตัว

อะเรย์

- จากตัวอย่างที่กล่าวมา อาจยอมรับได้สำหรับข้อมูลที่มี 20 จำนวน แต่ถ้าโจทย์ปัญหาของเราต้องใช้ข้อมูลมากๆเช่น 200 หรือ 2,000 หรือ 20,000 จำนวนก็คงดำเนินการเช่นที่ผ่านมาอย่างมีประสิทธิภาพไม่ได้
- การประมวลผลกับข้อมูลที่มีจำนวนมากๆ ในทางคอมพิวเตอร์จะต้องใช้โครงสร้างข้อมูลที่มีประสิทธิภาพทั้งในเรื่องการตั้งชื่อ การอ่าน การประมวลผล และการพิมพ์ โครงสร้างข้อมูลดังกล่าวคือ อะเรย์ (array)
- อะเรย์เป็นโครงสร้างข้อมูลที่มี ขนาดคงที่ เป็นข้อมูลประเภทเดียวกัน และเรียงกันอยู่ตามลำดับของหน่วยความจำที่ใช้เก็บ การอ้างถึงข้อมูลในอะเรย์สามารถอ้างถึงตัวที่หนึ่ง ตัวที่สอง และตัวต่อๆไปจนหมด

อะเรย์

- ถ้าเราต้องการเก็บเลข 20 จำนวนไว้ในอะเรย์ เราสามารถเก็บข้อมูลตัวแรกอยู่ที่ $number_0$ ดังรูปที่ 11.1 ในทำนองเดียวกันข้อมูลตัวที่สองเก็บอยู่ที่ $number_1$ และเป็นเช่นนี้ไปจนข้อมูลตัวสุดท้ายจะเก็บไว้ที่ $number_{19}$ โดยตัวเลขที่ห้อย (subscript) จะระบุเลขที่ลำดับของข้อมูลที่นับลำดับจากตำแหน่งเริ่มต้นของอะเรย์
- ในรูปที่ 11.3 จะเห็นว่าข้อมูลแต่ละตัวในอะเรย์จะมี ตำแหน่งที่อยู่ (address) เป็นของตัวเองโดยเฉพาะซึ่งกำหนดโดยตัวเลขที่ห้อย โครงสร้างอะเรย์โดยรวมแล้วมีชื่อเดียวคือ number และข้อมูลแต่ละตัวในอะเรย์สามารถเข้าถึงได้อย่างเป็นอิสระต่อตัวอื่นโดยใช้ตัวเลขที่ห้อย



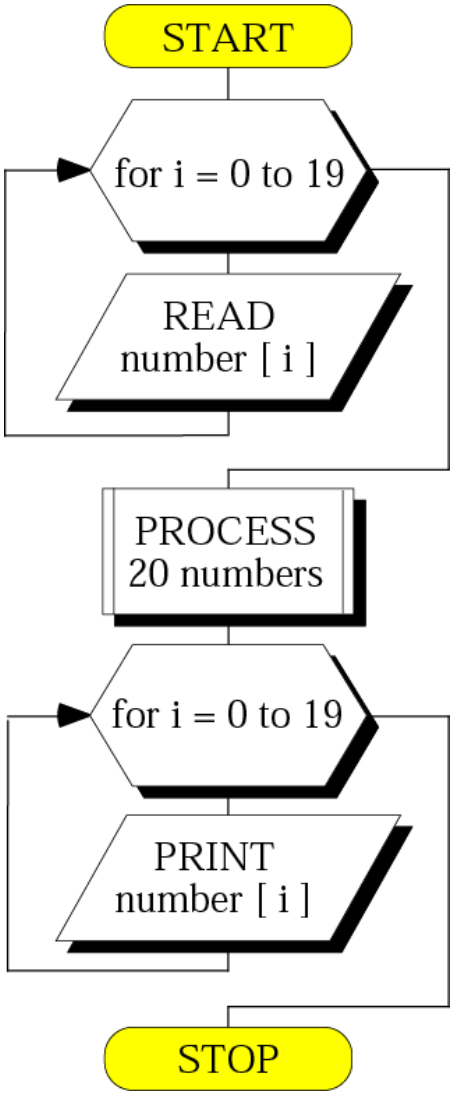
รูปที่ 11-3 Arrays with subscripts and indexes

อะเรย์

- ข้อดีอย่างหนึ่งของโครงสร้างข้อมูลอะเรย์คือมีโครงสร้าง (constructs) ของภาษาคอมพิวเตอร์ที่ช่วยให้การประมวลผลอะเรย์ได้อย่างมีประสิทธิภาพมากขึ้น โครงสร้างดังกล่าวคือการทำซ้ำๆ (loops) ซึ่งทำให้การประมวลผลอะเรย์ง่ายขึ้น
- เราสามารถใช้การทำซ้ำๆเพื่ออ่าน เขียน บวก ลบ คูณ หรือ หารสมาชิกของอะเรย์ นอกจากนี้เรายังสามารถใช้การทำซ้ำๆเพื่อการระมวลผลที่ซับซ้อนมากขึ้นเช่นการคำนวณหาค่าเฉลี่ยของค่าทั้งหมดในอะเรย์ ไม่ว่าจะมีข้อมูล 2, 20, 200, 2000 หรือ 20,000 จำนวนก็ตาม การทำซ้ำๆทำให้การประมวลผลโครงสร้างอะเรย์สะดวกและรวดเร็วยิ่งขึ้น

อะเรย์

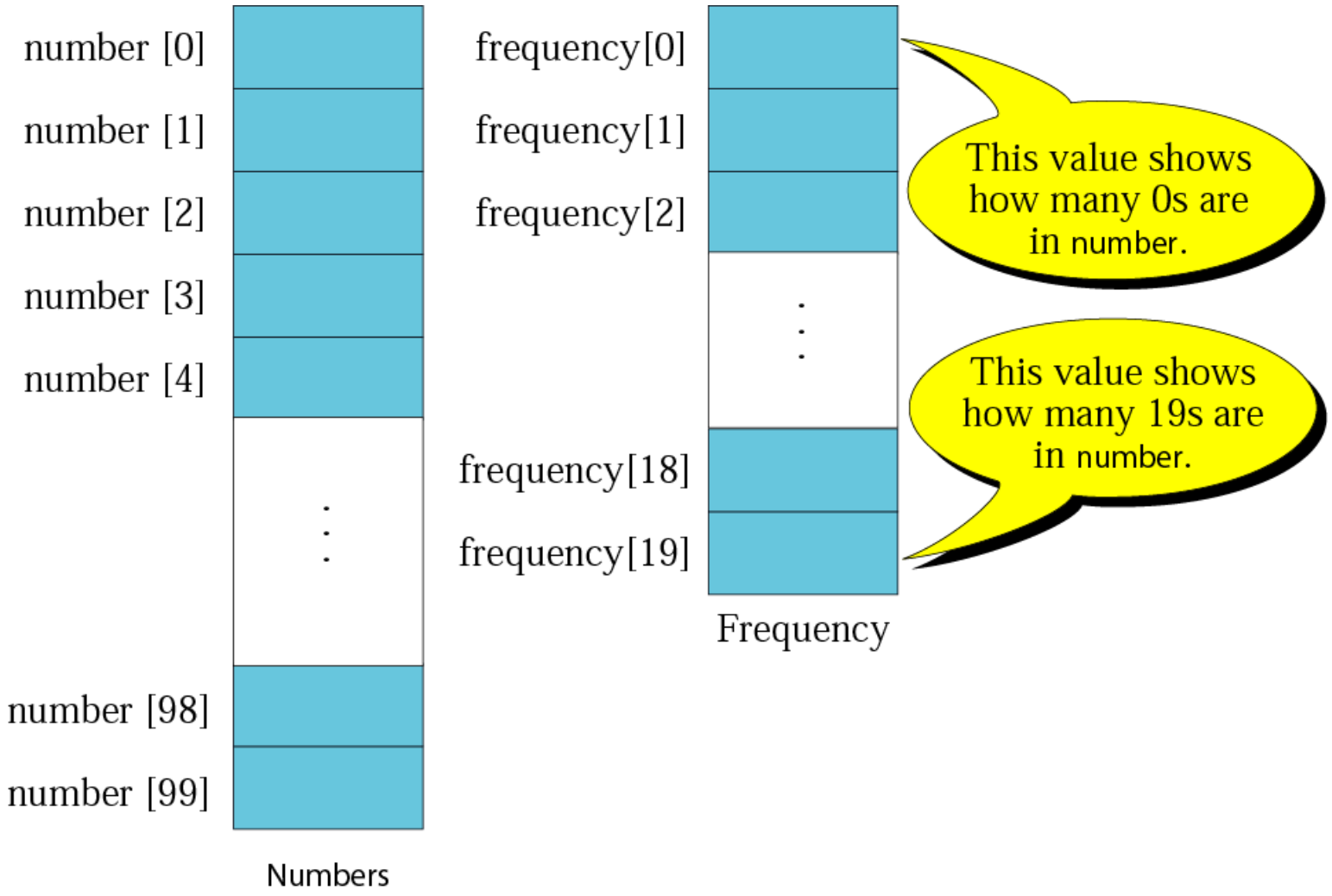
- ยังมีปัญหาที่จะต้องคิดต่อคือเราจะเขียนคำสั่งเพื่อเข้าถึงสมาชิกตัวแรกและตัวต่อๆไปอย่างไร? แนวทางคือใช้ตัวเลขที่ห้อยให้เกิดประโยชน์ โดยการเขียนตัวเลขที่ห้อยไว้ในวงเล็บใหญ่ [ ] เช่น $number_0$ จะเขียนแทนด้วย number[0] และ $number_1$ แทนด้วย number[1] เป็นต้น การแทนด้วยวิธีการดังกล่าวรู้จักกันในชื่อดัชนี (indexing) การอ้างถึงข้อมูลในอะเรย์จะใช้ชื่อตัวแปรที่เป็นชื่อของอะเรย์
- ตัวอย่างผังงานการประมวลผลข้อมูลตัวเลข 20 จำนวนโดยใช้อะเรย์และการทำซ้ำๆแสดงในรูปที่ 11.4



รูปที่ 11-4 การประมวลผล array

การประยุกต์ใช้อะเรย์

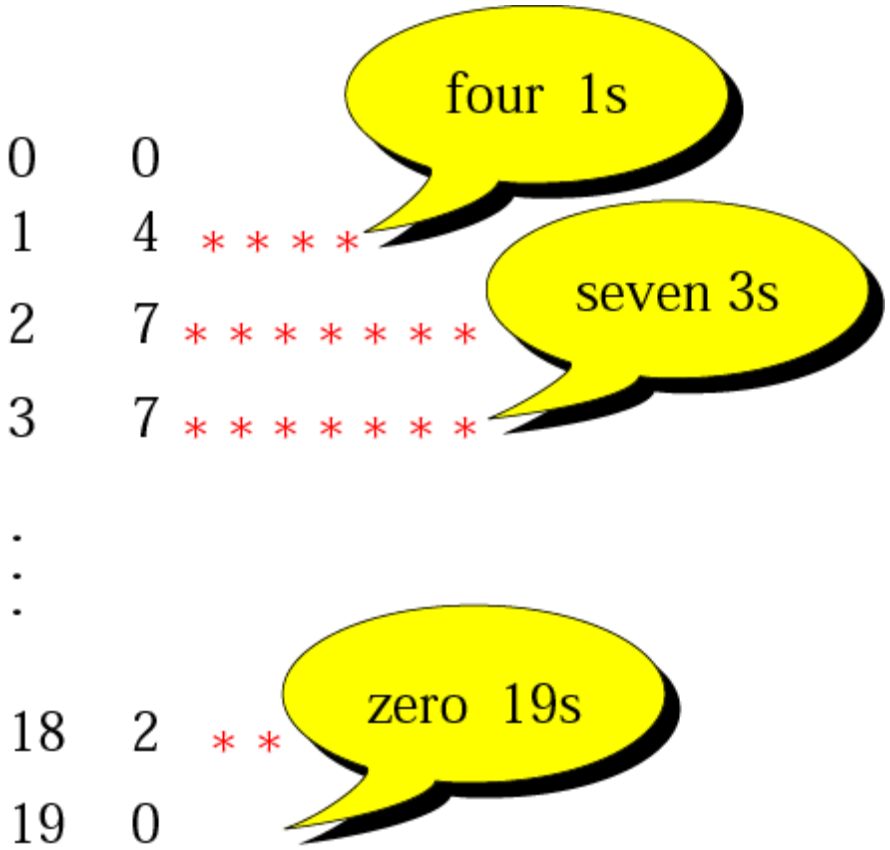
- ในหัวข้อนี้จะยกตัวอย่างการประยุกต์ใช้อะเรย์ในการนับความถี่และการแทนความถี่ด้วยกราฟ อะเรย์ความถี่จะแสดงจำนวนข้อมูลที่เป็นตัวเลขแสดงถึงว่ามีข้อมูลนั้นอยู่กี่จำนวน สมมติว่ามีข้อมูลที่มีค่าตั้งแต่ 0 ถึง 19 อยู่ 100 จำนวนหากเราต้องการที่จะทราบว่ามี 0 กี่ตัว มี 1 กี่ตัว มี 2 กี่ตัว และต่อไปเรื่อยๆ จนกระทั่งมี 19 อยู่กี่ตัว การแก้ปัญหานี้เราสามารถอ่านค่าของข้อมูลเข้าไปเก็บในอะเรย์ชื่อ **number** จากนั้นทำการสร้างอะเรย์ที่มีสมาชิก 20 ตัวซึ่งจะเก็บความถี่หรือจำนวนครั้งที่ข้อมูลแต่ละตัวปรากฏ ดังรูปที่ 11.5



รูปที่ 11-5 Frequency array

ฮิสโตแกรม

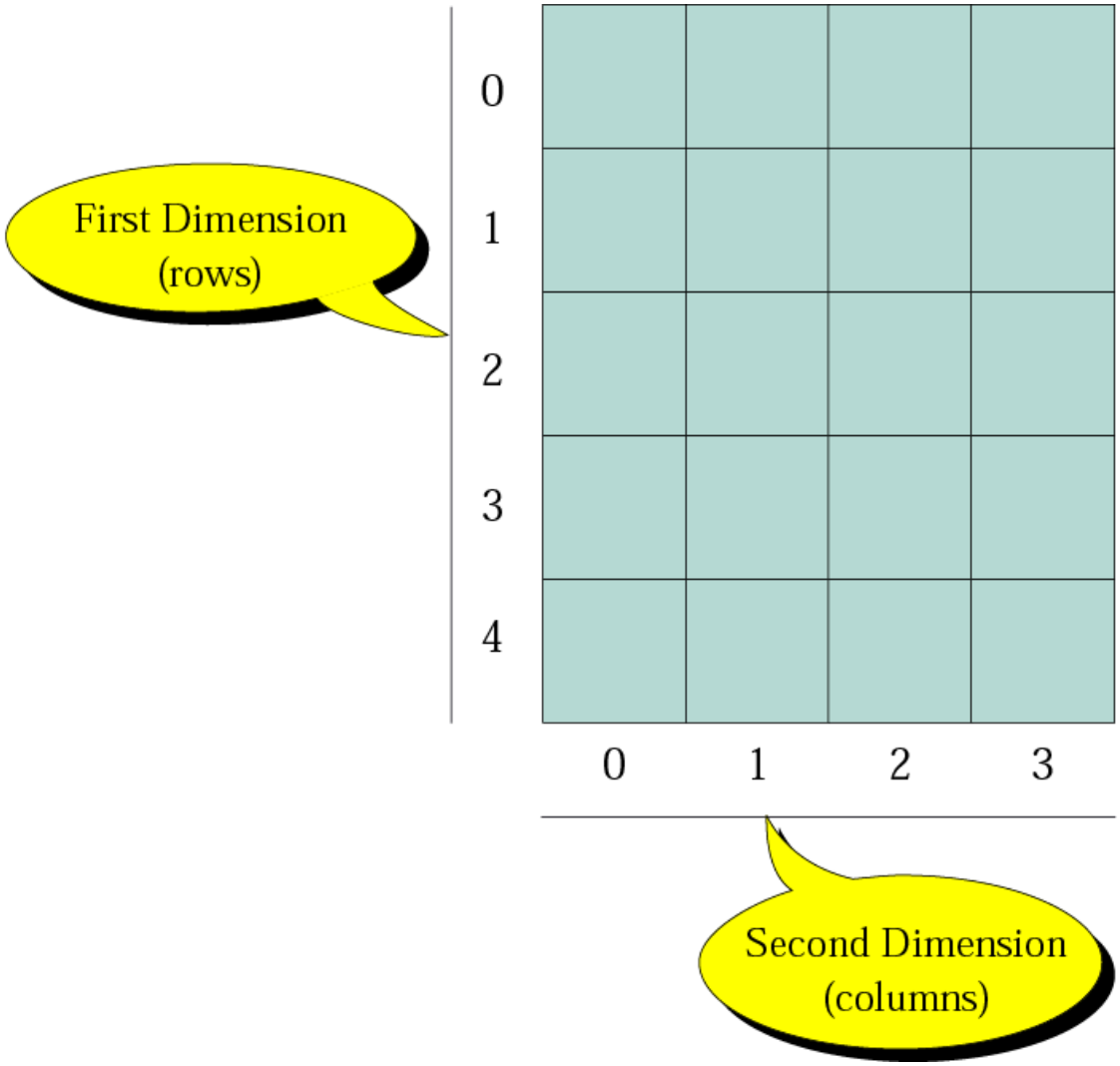
- ฮิสโตแกรมคือรูปภาพที่ใช้แทนความถี่ในอะเรย์ แทนที่จะเขียนความถี่เป็นตัวเลข เราสามารถแทนความถี่ด้วยการพิมพ์เป็นรูป bar chart ตัวอย่างเช่นในรูปที่ 11.6 เป็นฮิสโตแกรมสำหรับข้อมูลที่มีค่าตั้งแต่ 0 ถึง 19 โดยใช้สัญลักษณ์ * แทน bar chart โดยกำหนดว่า * แทนความถี่ 1 ครั้งของข้อมูลนั้นๆ



รูปที่ 11-6 Histogram

อะเรย์ 2-มิติ

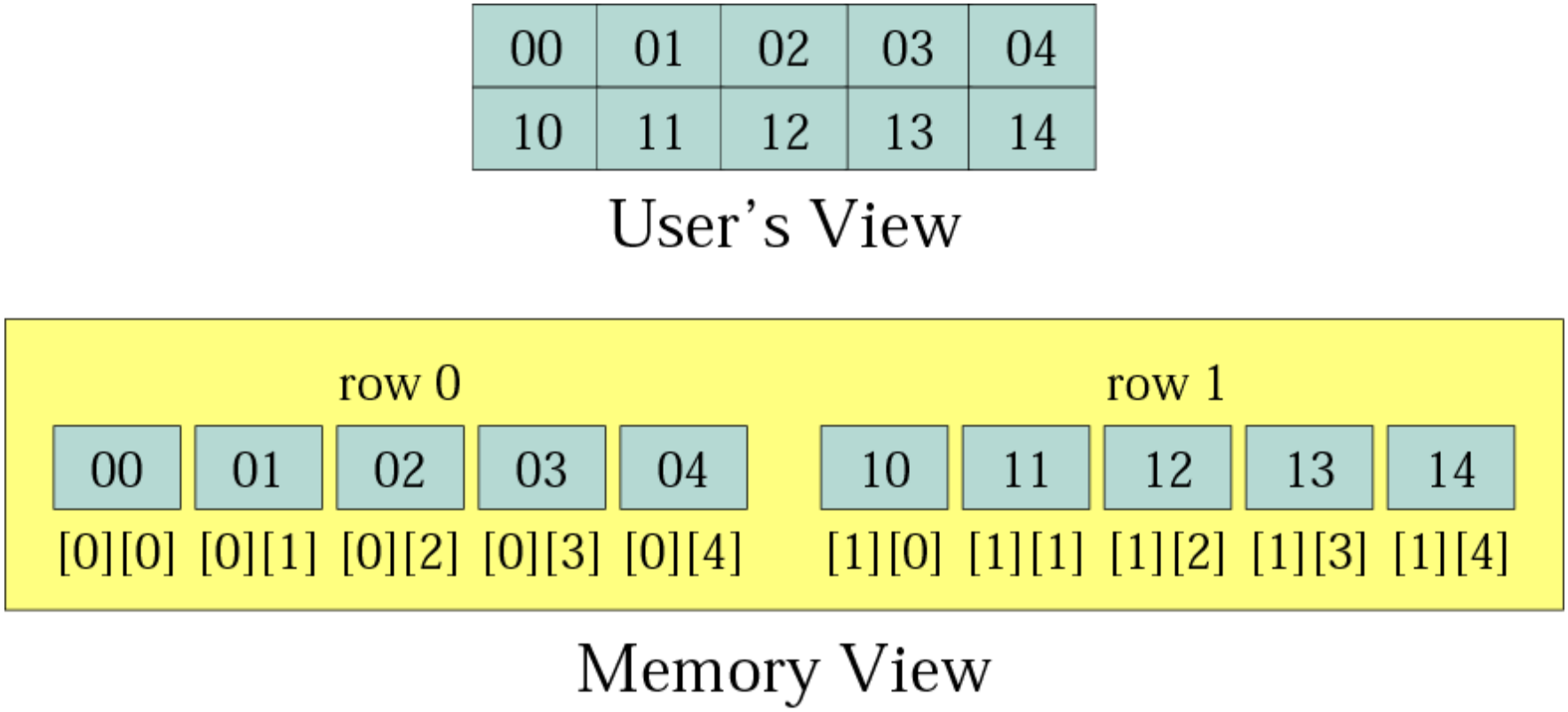
- โครงสร้างข้อมูลอะเรย์ที่กล่าวมาแล้วเรียกว่าอะเรย์ 1-มิติ ที่เรียกเช่นนี้เพราะว่าข้อมูลจัดเรียงตามแนวไปในทิศทางเดียว แต่ในการประยุกต์ใช้จริงยังมีความต้องการในการใช้ข้อมูลที่มีมากกว่า 1-มิติ ที่เราพบเสมอคือข้อมูลที่อยู่ในรูปเมตริกซ์ซึ่งมี 2 แถวในแนวนอน (row) และ 2 แถวในแนวตั้ง (column) รูปที่ 11.7 แสดงตารางซึ่งปกติจะเรียกว่าอะเรย์ 2-มิติ จากรูปนี้เราสามารถกำหนดอะเรย์ 3-มิติ อะเรย์ 4-มิติ และต่อๆไปได้ อะเรย์ที่มีมิติที่สูงกว่า 2 เราเรียกว่าอะเรย์หลายมิติ (multidimensional array) ซึ่งจะไม่กล่าวถึง ณ ที่นี้ เอาไว้เรียนในวิชาที่สูงขึ้น



รูปที่ 11-7- ตอนที่ I อะเรย์ 2-มิติ

รูปแบบของหน่วยความจำที่แทนอะเรย์

- ดัชนีของอะเรย์ 2-มิติแทน row และ column ซึ่งรูปแบบนี้จะแทนรูปแบบการเก็บข้อมูลในหน่วยความจำ ถ้าเราพิจารณาหน่วยความจำประกอบด้วยแถวของไบท์โดยที่ตำแหน่งที่อยู่ (address) ที่น้อยที่สุดอยู่ทางซ้ายมือสุด และตำแหน่งที่อยู่ (address) ที่มากที่สุดอยู่ทางขวามือสุด ดังนั้น อะเรย์จะเก็บค่าข้อมูลตัวแรกไว้ที่หน่วยความจำซ้ายมือสุด และค่าข้อมูลตัวสุดท้ายจะเก็บไว้ที่หน่วยความจำขวามือสุด
- ในกรณีที่เป็นอะเรย์ 2-มิติ มิติแรกคือแถวของข้อมูลที่เก็บอยู่ด้านซ้าย วิธีการเก็บแบบนี้เรียกว่า “row-major” storage ดังแสดงในรูปที่ 11.8



รูปที่ 11-8 Memory layout

11.2

ระเบียน

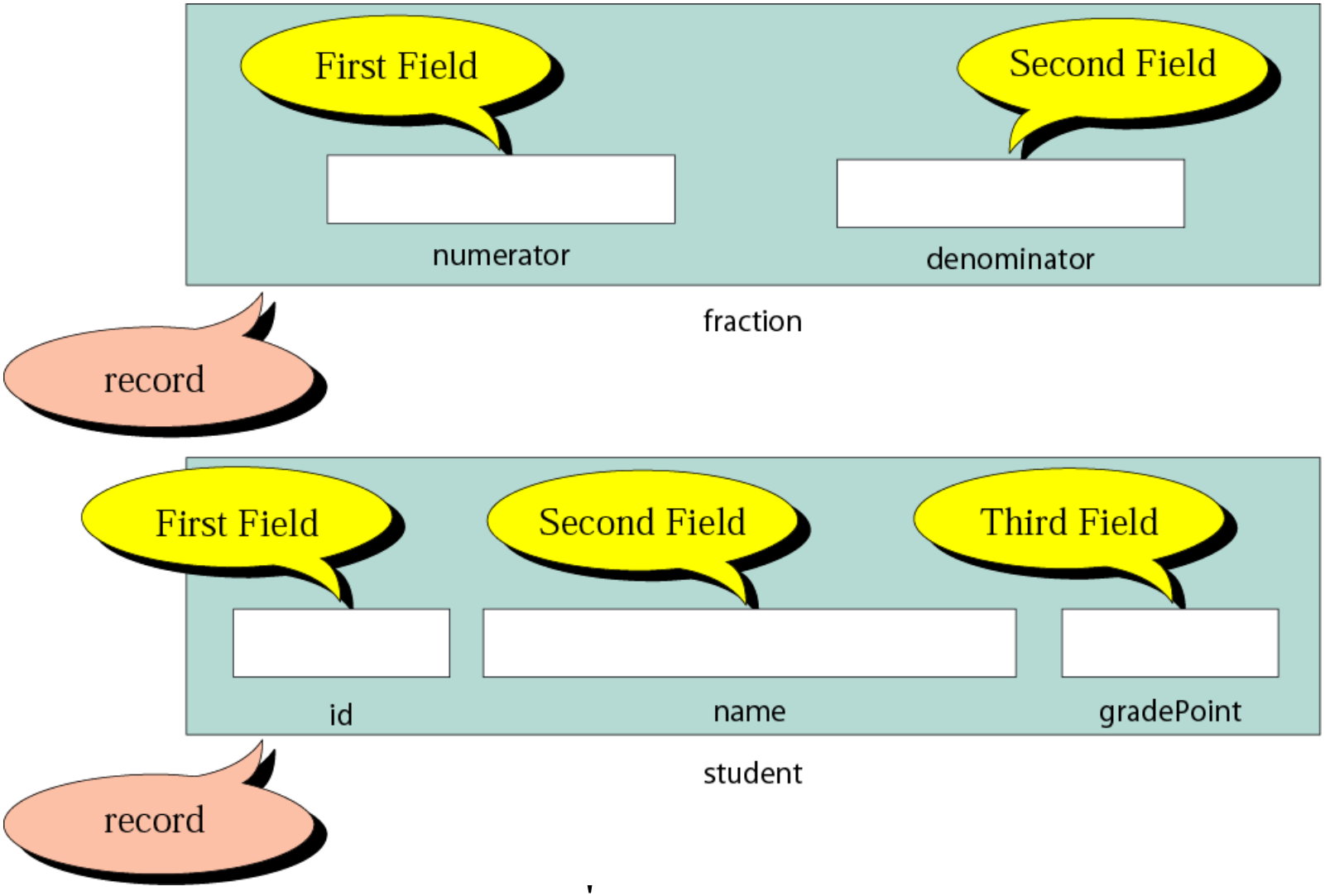
RECORDS

ระเบียน

- ระเบียนเป็นโครงสร้างข้อมูลที่ประกอบด้วยกลุ่มของข้อมูลที่สัมพันธ์กัน ซึ่งข้อมูลอาจต่างประเภทกันและมีชื่อเรียกเพียงชื่อเดียว แต่ละส่วนประกอบที่เป็นส่วนย่อยที่เล็กที่สุดในระเบียนเรียกว่าฟิลด์ (field) โดยที่ฟิลด์เป็นชื่อของข้อมูลที่มีความหมาย ฟิลด์ต้องระบุประเภทของข้อมูลและลักษณะการเก็บในหน่วยความจำ ฟิลด์สามารถกำหนดให้เก็บข้อมูลและสามารถเข้าถึงได้ ฟิลด์แตกต่างจากตัวแปรคือฟิลด์เป็นส่วนหนึ่งของระเบียนเท่านั้น คุณสมบัติอื่นๆจะเหมือนกับตัวแปรทั่วไป

ระเบียน

- ความแตกต่างระหว่างอะเรย์กับระเบียนคือทุกข้อมูลในอะเรย์จะต้องเป็นประเภทเดียวกัน แต่ข้อมูลในระเบียนอาจเหมือนหรือต่างประเภทกันได้ ตัวอย่างในรูปที่ 11.9 ประกอบด้วยระเบียน 2 ระเบียน โดยระเบียนแรกชื่อ fraction ประกอบด้วย 2 ฟิลด์ แต่ละฟิลด์เป็นประเภทจำนวนเต็ม (integer) ในระเบียนที่สองที่ชื่อ student มี 3 ฟิลด์ โดยฟิลด์แรกชื่อ id อาจมีประเภทของข้อมูลเป็นสตริงหรือจำนวนเต็มก็ได้ ส่วนฟิลด์ name มีประ เภทข้อมูลเป็นสตริง ส่วนฟิลด์ gradePoint มีปรเภทข้อมูลเป็นจำนวนจริง เป็นต้น



รูปที่ 11-9 ระเบียน

ข้อสังเกต:

ข้อมูลในระเบียนอาจเป็นประเภทเดียวกันหรือต่างประเภทกันก็ได้ แต่ข้อมูลทั้งหมดภายในระเบียนเดียวกันจะต้องสัมพันธ์กัน

การเข้าถึงระเบียน

- ข้อพึงระวัง: ข้อมูลแต่ละฟิลด์ในระเบียนจะต้องสัมพันธ์ซึ่งกันและกัน ระเบียนแรกในรูปที่ 11.9 ประเภทของข้อมูลทั้งสองฟิลด์เป็นจำนวนเต็ม ของเศษส่วนจำนวนเดียวกัน ส่วนข้อมูลในระเบียนที่สองของทั้งสามฟิลด์มีความสัมพันธ์กันทั้งหมดเพราะเป็นข้อมูลของนักเรียนคนเดียวกัน
- การเข้าถึงข้อมูลในแต่ละฟิลด์ของระเบียนสามารถกระทำได้โดยการเขียนเป็นคำสั่งที่ประกอบด้วยตัวกระทำ (operator) การกระทำใดก็ตามที่เราสามารถทำได้กับตัวแปรปกติ เราสามารถทำได้กับฟิลด์ในระเบียนเช่นเดียวกัน แต่จะต้องระบุเฉพาะฟิลด์ที่ต้องการเท่านั้น

การเข้าถึงระเบียน (ต่อ)

- เนื่องจากแต่ละฟิลด์ในระเบียนมีชื่อเฉพาะที่ไม่ซ้ำกัน การเข้าถึงทำได้โดยการระบุชื่อของฟิลด์เท่านั้น แต่ถ้าต้องการเปรียบเทียบค่าข้อมูลในฟิลด์ที่มีชื่อเดียวกันแต่อยู่ในระเบียนที่มีชื่อต่างกัน เราทำได้โดยระบุชื่อระเบียนตามด้วย . (จุด) แล้วตามด้วยชื่อฟิลด์ เช่นถ้าเรามี 2 ระเบียน ชื่อ student1 กับ student2 ซึ่งเป็นระเบียนประเภทเดียวกัน เราสามารถเข้าถึงระเบียนทั้งสองได้ดังนี้

  student1.id, student1.name, และ student1.gradePoint

  student2.id, student2.name, และ student2.gradePoint

  เราสามารถอ่านและเขียนข้อมูลจากระเบียนได้เช่นเดียวกับตัวแปร

11.3

ลิงค์ลิสต์

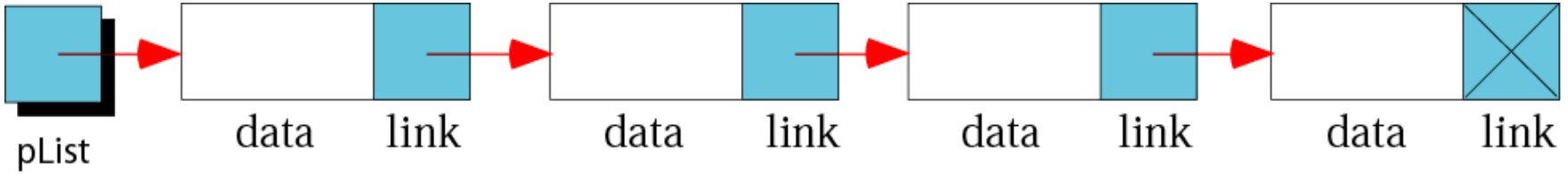
LINKED LISTS

ลิงค์ลิสต์

- ลิงค์ลิสต์เป็นโครงสร้างข้อมูลที่ประกอบด้วยกลุ่มของข้อมูลที่จัดเรียงตามลำดับที่ข้อมูลแต่ละตัวจะมีตำแหน่งที่อยู่ (location/address) ของข้อมูลในลำดับถัดไป นั่นคือข้อมูลแต่ละตัวประกอบด้วย 2 ส่วนคือส่วนที่เป็นข้อมูล (data) และส่วนที่เป็นตัวเชื่อมโยง (link) ส่วนข้อมูลจะมีข้อมูลตามที่กำหนดให้ใช้ในการประมวลผล ส่วนตัวเชื่อมโยงเป็นตัวชี้ (pointer) ที่เก็บตำแหน่งที่อยู่ของข้อมูลตัวถัดไปที่อยู่ในลิสต์นั้น จุดเริ่มต้นของลิงค์ลิสต์เป็นตัวแปรประเภทตัวชี้ที่ชี้ไปยังสมาชิกตัวแรกของลิสต์ ชื่อของลิสต์จะเป็นชื่อเดียวกับชื่อตัวแปรที่เป็นตัวชี้ ลิงค์ลิสต์ที่จะอธิบายต่อไปนี้เป็นลิสต์ธรรมดาที่เรียกว่า **singly linked list** เพราะว่าเป็นลิสต์ที่มีเพียงลิงค์เดียวที่ชี้ไปยังข้อมูลตัวถัดไป

ลิงค์ลิสต์

- ลิงค์ลิสต์ไม่ใช่โครงสร้างข้อมูลโดยตัวของมันเอง แต่เกิดจากการจำลองแบบโครงสร้างด้วยโครงสร้างของภาษาคอมพิวเตอร์ที่ใช้ รูปที่ 11.10 แสดงลิงค์ลิสต์ชื่อ pList (แทนตัวชี้ที่ชี้ไปยังข้อมูลตัวแรก หรือ head ของลิสต์) ที่ประกอบด้วยข้อมูล 4 ชุด ลิงค์ของข้อมูลแต่ละตัวยกเว้นตัวสุดท้ายชี้ไปยังข้อมูลตัวถัดไป ส่วนลิงค์ของข้อมูลตัวสุดท้ายไม่ได้ชี้ไปที่ใด เรียกตัวชี้ตัวสุดท้ายนี้ว่า **null pointer** เป็นตัวบ่งบอกว่าจบลิงค์ลิสต์
- เรานิยามลิงค์ลิสต์ที่ไม่มีสมาชิก หรือลิสต์ว่าง (empty linked list) ว่าคือลิสต์ที่มี head pointer เป็น null รูปที่ 11.10 ด้านล่างแสดงลิงค์ลิสต์ชื่อ pList แต่เป็นลิสต์ว่าง



A linked list with a head pointer pList

An empty linked list

รูปที่ 11-10 ลิงค์ลิสต์

โหนด (nodes)

- โดยปกติ สมาชิกในลิงค์ลิสต์มักจะเรียกว่าโหนด (node) โดยแต่ละโหนดเปรียบได้กับหนึ่งระเบียนที่ประกอบด้วยอย่างน้อย 2 ฟิลด์คือฟิลด์แรกเก็บข้อมูล และฟิลด์ที่สองเก็บตำแหน่งที่อยู่ของโหนดถัดไปในลิสต์ แสดงดังรูปที่ 11.11 โหนดในลิงค์ลิสต์มักเรียกว่าระเบียนที่อ้างอิงตัวเอง (self-referential records) ในระเบียนประเภทนี้ แต่ละกรณีของระเบียน (instance) จะมี pointer ที่ชี้ไปยังกรณีของระเบียนตัวต่อไปที่มีโครงสร้างเดียวกัน

รูปที่ 11-11

Node

ตัวชี้ไปยังลิงค์ลิสต์

- ลิงค์ลิสต์จะต้องมี **head pointer** เสมอ ส่วน pointer ที่อยู่ในแต่ละโหนดนั้นอาจมีมากกว่าหนึ่งก็ได้ ขึ้นอยู่กับวัตถุประสงค์ในการใช้งาน เช่นถ้าเราต้องการค้นหาข้อมูลในลิงค์ลิสต์ เราจะต้องมี pointer (pLoc) ที่ชี้ไปยังตำแหน่งที่อยู่ของข้อมูลที่เราต้องการค้นหา ในกรณีที่มีระเบียนจำนวนมากๆ การประมวลผลจะมีประสิทธิภาพมากขึ้นถ้ามีการใช้ pointer 2 ตัว คือตัวหนึ่งชี้ไปยัง head ของลิงค์ลิสต์ อีกตัวหนึ่งชี้ไปที่โหนดสุดท้ายของลิสต์ pointer ที่ชี้ไปยังโหนดตัวสุดท้ายนี้มักจะเรียกว่า pLast หรือ pRear การตั้งชื่อเช่นนี้ก็เพื่อให้สอดคล้องกับความหมายที่ pointer นี้แทนเท่านั้น เราอาจตั้งชื่อเป็นอย่างอื่นก็ได้

การดำเนินการกับลิงค์ลิสต์

- เรานิยามการดำเนินการกับลิงค์ลิสต์ 5 รูปแบบซึ่งเพียงพอสำหรับการแก้ปัญหาที่เกี่ยวกับลิสต์แบบลำดับ (sequential list) แต่ถ้าการประยุกต์กับปัญหาใดที่ต้องการการดำเนินการเพิ่มเติม ก็สามารถเพิ่มได้โดยง่าย การดำเนินการทั้ง 5 มีดังนี้

  1. การเพิ่มโหนด (Inserting a Node): ดำเนินการตาม 3 ขั้นตอนต่อไปนี้

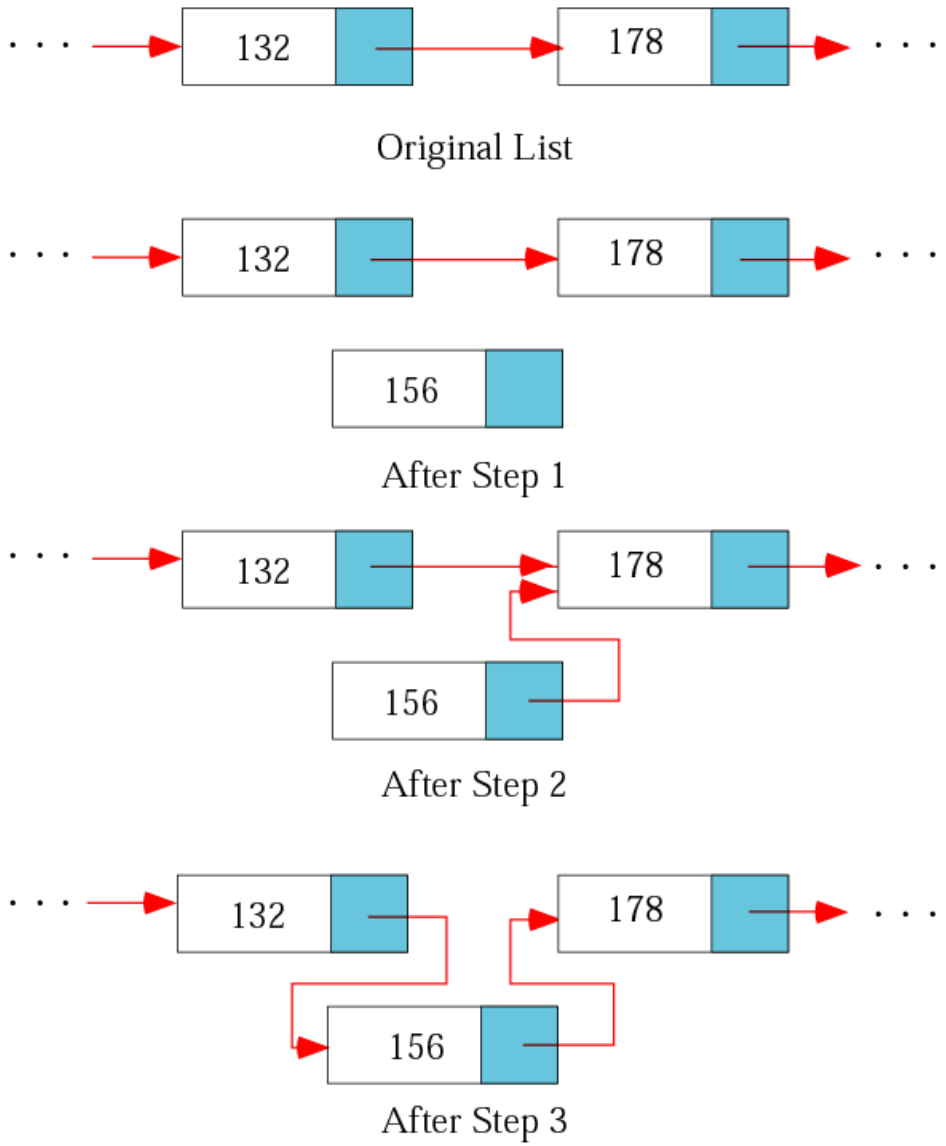
     (1) จัดหาหน่วยความจำสำหรับโหนดใหม่พร้อมทั้งใส่ข้อมูล

     (2) กำหนดค่าให้ลิงค์ฟิลด์ของโหนดใหม่ชี้ไปยังโหนดถัดไป

     (3) กำหนดให้ลิงค์ฟิลด์ของโหนดก่อนหน้า ชี้ไปยังโหนดใหม่

รูปที่ 11-12

การเพิ่มโหนด

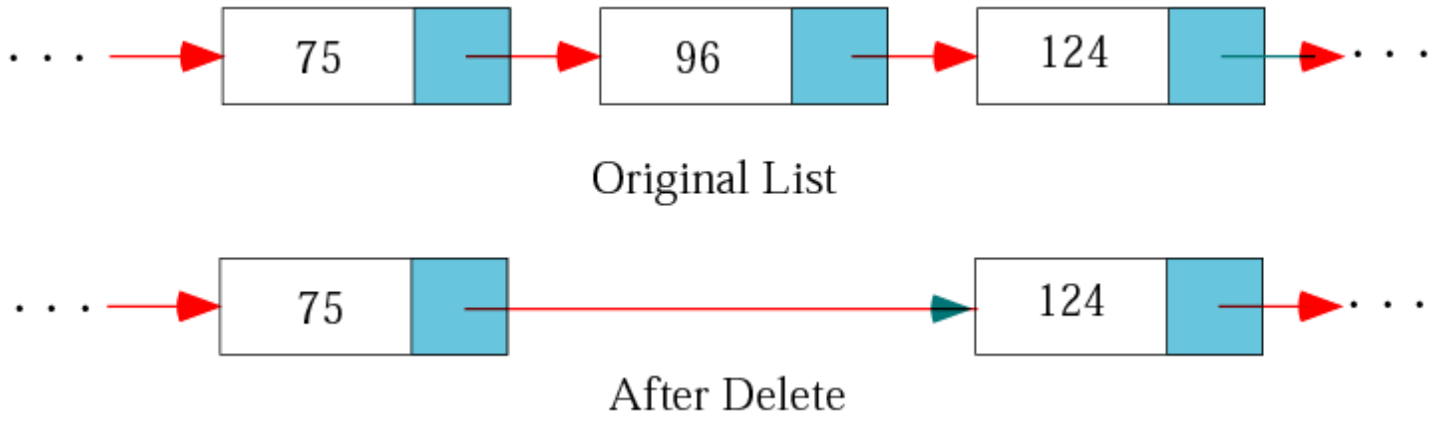


การดำเนินการกับลิงค์ลิสต์ (ต่อ)

2. การตัดโหนด (Deleting a node): การตัดโหนดในลิงค์ลิสต์ ก่อนอื่นต้องค้นหาโหนดที่ต้องการให้พบก่อน เมื่อพบแล้วก็เพียงทำการเปลี่ยนลิงค์ฟิลด์ของโหนดก่อนหน้าให้ชี้ไปยังโหนดที่ถัดไปจากโหนดที่ต้องการตัด (ดังรูปที่ 11.13) การตัดโหนดต้องระวังกรณีที่ในลิงค์ลิสต์มีเพียงโหนดเดียว เพราะผลลัพธ์คือลิสต์ว่าง เช่นเดียวกัน การตัดโหนดแรกก็ต้องใช้ความระมัดระวังเช่นเดียวกัน เพราะขั้นตอนที่อธิบายข้างต้นจะต้องมีการปรับเปลี่ยนบ้าง มิฉะนั้นอาจก่อให้เกิดผลที่ผิดพลาดได้

รูปที่ 11-13

การตัดโหนด



การดำเนินการกับลิงค์ลิสต์ (ต่อ)

- **3. การค้นหาโหนด ในลิสต์ (Searching a List): การค้นหาโหนดในลิสต์เป็นการดำเนินการที่ถูกเรียกใช้โดยการดำเนินการอื่น เช่นการเพิ่มโหนดเข้าไปในลิสต์ เราต้องทราบตำแหน่งของโหนดที่อยู่ก่อนหน้าโหนดที่เราต้องการเพิ่ม และถ้าต้องการตัดโหนดที่ไม่ต้องการออก เราก็จะต้องค้นหาโหนดนั้นให้พบก่อน ในทำนองเดียวกัน ถ้าเราต้องการดึงข้อมูลจากโหนดที่ต้องการ ก็ต้องค้นหาโหนดนั้นให้พบก่อนเช่นกัน**
- **การค้นหาโหนดในลิสต์โดยใช้คีย์ แต่ละโหนดในลิสต์จะต้องมีฟิลด์ที่ใช้เป็นคีย์ อย่างไรก็ตาม ถ้าการค้นหาใช้ข้อมูลเป็นตัวค้น ฟิลด์ที่เป็นคีย์จะเป็นฟิลด์เดียวกับฟิลด์ข้อมูลก็ได้**

การกระทำกับลิงค์ลิสต์ (ต่อ)

- ในกรณีที่โหนดเป็นระเบียนที่มีความซับซ้อน ฟิลด์ที่เป็นคีย์โดยเฉพาะจำเป็นจะต้องมี เมื่อกำหนดคีย์ที่ต้องการให้ การค้นหาโหนดที่ต้องการในลิงค์ลิสต์คือพยายามหาตำแหน่งที่อยู่ของโหนดที่มีค่าข้อมูลในฟิลด์ที่เป็นคีย์เท่ากับค่าคีย์ที่กำหนด ถ้ามีโหนดซึ่งมีค่าคีย์ตรงตามที่กำหนด การค้นหาก็ประสบความสำเร็จด้วยการส่งผลไปยังผู้เรียกใช้ (อาจเป็นการดำเนินการอื่น)ว่า “true” หรือ “success” แต่ถ้าไม่มีโหนดที่มีค่าคีย์ตรงตามที่กำหนด การค้นหาก็ประสบกับความล้มเหลว (failure) พร้อมทั้งส่งผลไปยังผู้ใช้ว่า “false” หรือ “failure”

การกระทำกับลิงค์ลิสต์ (ต่อ)

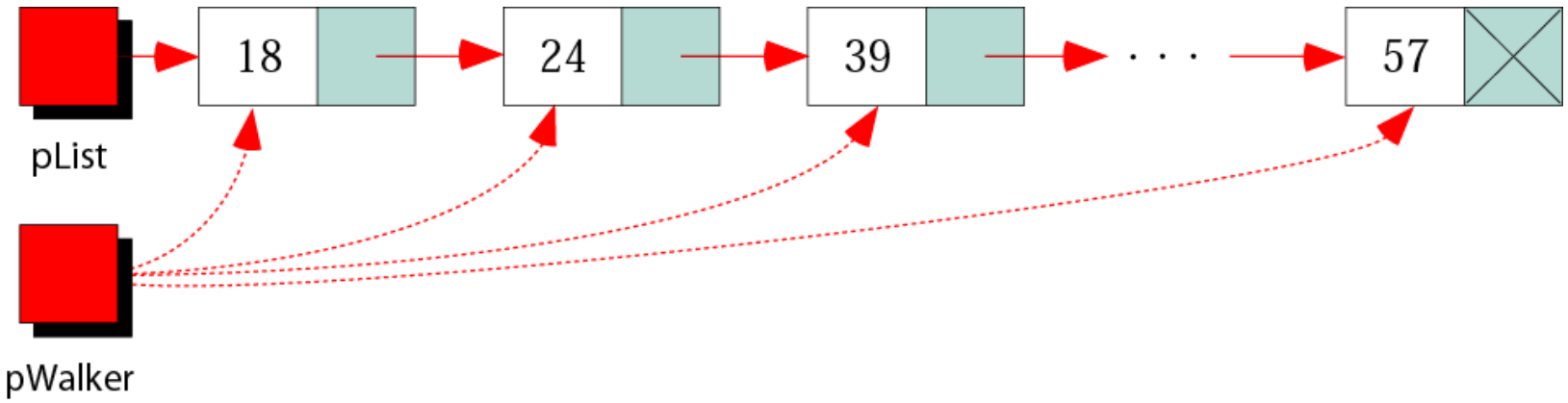
- **4. การดึงหรือการค้นคืนข้อมูลจากโหนด (Retrieving a Node): การค้นคืนข้อมูลจากลิงค์ลิสต์ทำได้โดยการค้นหาโหนดที่ต้องการก่อน ถ้าพบก็สามารถดึงข้อมูลออกมาได้ แต่ถ้าไม่พบ (ไม่มีข้อมูลที่ต้องการ) ก็ไม่ต้องทำอะไร**
- **5. การท่องลิสต์ (Traversing a List): อัลกอริธึมสำหรับการท่องลิสต์เริ่มต้นจากโหนดแรกในลิสต์ ทำการตรวจสอบและประมวลผลแต่ละโหนดตามลำดับจนถึงโหนดสุดท้าย การท่องลิสต์จำเป็นต้องใช้เพื่อเป็นส่วนหนึ่งของหลายๆอัลกอริธึมเช่น การเปลี่ยนค่าในแต่ละโหนด การพิมพ์ค่าในลิสต์ การหาผลรวมของข้อมูลในลิสต์ การหาค่าเฉลี่ย เป็นต้น**

การกระทำกับลิงค์ลิสต์ (ต่อ)

- การท่องลิสต์จำเป็นต้องใช้ตัวชี้พิเศษ 1 ตัวคือ “walking pointer” ซึ่งเป็นตัวชี้ที่เคลื่อนที่จากโหนดหนึ่งไปยังอีกโหนดหนึ่งหลังจากที่แต่ละโหนดได้รับการประมวลผลแล้ว เริ่มต้นด้วยการกำหนดให้ walking pointer ชี้ไปยังโหนดแรกในลิสต์ จากนั้นใช้การทำซ้ำๆ (loop) จนกระทั่งข้อมูลในทุกโหนดของลิสต์ได้รับการประมวลผลทั้งหมด ในแต่ละวงรอบจะมีการประมวลผลข้อมูล ส่งผลลัพธ์ไปที่ที่ต้องการ แล้วทำการเลื่อน walking pointer ไปยังโหนดถัดไป เมื่อโหนดสุดท้ายถูกประมวลผลแล้วเสร็จ ค่าของ walking pointer จะเป็น null และการทำซ้ำก็จบลง (ดังรูปที่ 11.14)

รูปที่ 11-14

การท่องลิสต์



คำสำคัญ

- Data Structures
- Array
- Pointer