



ค่ายโอลิมปิกวิชาการ วิชาคอมพิวเตอร์

VSCODE

ผศ.ดร.เนียบวุฒิ รัตนวิไลสกุล chaibwoot.r@sci.kmutnb.ac.th

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าฯ พระนครเหนือ



VSCODE



Virtual Code



VS CODE

The screenshot shows the Visual Studio Code (VS Code) interface. The main area displays a C++ file named "code.cpp" with the following content:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int a = 0;
    cin>>a;
    cout<<"Hello CS "<<a<<endl;
    return 0;
}
```

The status bar at the bottom indicates the code is in C++ mode, using Win32 encoding, and was last saved 28/2/2568 at 10:44. The terminal window shows the command used to run the code in the Microsoft MI Engine.

Annotations in red circles highlight the following elements:

- A red circle around the terminal tab in the bottom-left corner, labeled "เลือก terminal".
- A red circle around the "RUN" button in the top-right corner of the editor, labeled "กด RUN".



VS CODE

The screenshot shows the Visual Studio Code (VS Code) interface. At the top, there's a navigation bar with File, Edit, Selection, View, Go, Run, Terminal, Help, and a search bar containing 'CPP'. Below the navigation bar is a code editor window titled 'code.cpp'. The code is:

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int a = 0;
    cin>>a;
    cout<<"Hello CS "<<a<<endl;
    return 0;
}
```

A red annotation in the code editor highlights the line 'cout<<"Hello CS "<<a<<endl;' with the text 'สามารถแก้ไข CODE ตรงนี้' (You can edit the code here). The status bar at the bottom shows the file path 'C:\Users\studentcs\Documents\CPP>', the current file 'code.cpp - CPP - Vi...', and other system information like CPU usage, memory, and disk space.

In the bottom right corner of the code editor, there's a terminal window showing the output of the program:

```
PS C:\Users\studentcs\Documents\CPP> & 'c:\Users\studentcs\.vscode\extensions\ms-vscode.cpptools-1.23.6-win32-x64\debug\adapters\bin\WindowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-z13c3vhs.ltx' '--stdout=Microsoft-MIEngine-Out-j1cronzd.am5' '--stderr=Microsoft-MIEngine-Error-rnmibbzvy.hum' '--pid=Microsoft-MIEngine-Pid-14t021zr.145' '--dbgExe=C:\msys64\ucrt64\bin\gdb.exe' '--interpreter
250                                     กรอก Input และแสดงผลต่อหน้า
Hello CS 250
PS C:\Users\studentcs\Documents\CPP>
```

A red circle highlights the number '250' in the terminal output. The status bar at the bottom of the terminal window also displays file paths and system metrics.



ค่ายโอลิมปิกวิชาการ วิชาคอมพิวเตอร์

GRADER

ผศ.ดร.เนียบวุฒิ รัตนวิไลสกุล chaibwoot.r@sci.kmutnb.ac.th

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ



GRADER

IP: <http://202.44.40.145>

A screenshot of a web browser window. The address bar shows the URL "202.44.40.145". The search bar contains the text "olympic". The main content area displays a login form titled "Welcome" with the sub-instruction "Please log in". It has two input fields: "Username" containing "b" and "Password" containing a masked character. Below the fields are "Login" and "Reset" buttons. To the right of the browser window, the text "Input username" is displayed. In the bottom right corner of the slide, the number "6" is visible.



GRADER

The screenshot shows a web browser window with the URL 202.44.40.145/tasks/test_lab/description. The page title is "olympic_2568_1". The top navigation bar includes links for various applications like Facebook, Gmail, Mail, PDF, YouTube, and Google Translate, along with system icons for battery, signal, and network.

Server time: 01:07:18
Time left: 42389:52:41

TEST LAB

- Overview
- Communication
- Statement** (highlighted with a red oval)
- Submissions
- Documentation

Contest Management System is released under the [GNU Affero General Public License](#).

test_lab (test_lab) description

Statement

no statement available

Some details

Type	Batch				
Time limit	10.000 seconds				
Memory limit	32.0 MiB				
Compilation commands	<table border="1"><tr><td>C++11 / g++</td><td>/usr/bin/g++ -DEVAL -std=gnu++11 -O2 -pipe -static -s -o test_lab test_lab.cpp</td></tr><tr><td>C11 / gcc</td><td>/usr/bin/gcc -DEVAL -std=gnu11 -O2 -pipe -static -s -o test_lab test_lab.c -lm</td></tr></table>	C++11 / g++	/usr/bin/g++ -DEVAL -std=gnu++11 -O2 -pipe -static -s -o test_lab test_lab.cpp	C11 / gcc	/usr/bin/gcc -DEVAL -std=gnu11 -O2 -pipe -static -s -o test_lab test_lab.c -lm
C++11 / g++	/usr/bin/g++ -DEVAL -std=gnu++11 -O2 -pipe -static -s -o test_lab test_lab.cpp				
C11 / gcc	/usr/bin/gcc -DEVAL -std=gnu11 -O2 -pipe -static -s -o test_lab test_lab.c -lm				

Attachments

[test_lab.pdf](#) 19.8 KB

Download เอกสาร



GRADER

A screenshot of a PDF viewer window titled "file:///C:/Users/b/Downloads/test_lab.pdf". The window shows a document with the following content:

โจทย์ Day 1

ข้อ 2 จงเขียนโปรแกรมคอมพิวเตอร์ รับตัวเลขจำนวนเต็ม 1 ตัว และยกกำลัง 2

ข้อมูลนำเข้า บรรทัดที่ 1 ตัวเลขจำนวนเต็ม 1 ค่า มีค่าอยู่ในช่วง $-10000 \leq I \leq 10000$

ข้อมูลส่งออก บรรทัดที่ 1 แสดงผลลัพธ์การยกกำลัง 2 ของตัวเลขที่รับเข้ามา

ตัวอย่างข้อมูลนำเข้า	ตัวอย่างข้อมูลส่งออก
4	16

ตัวอย่างข้อมูลนำเข้า	ตัวอย่างข้อมูลส่งออก
5	25



GRADER

The screenshot shows a web browser window displaying a contest management system. The URL in the address bar is 202.44.40.145/tasks/test_lab/submissions?submission_id=PrPrdwDDdd-pZ3CzdhQngA1EZhKiPEk2Wc2dBGxefo. The page title is "olympic_2568_1". The top navigation bar includes links for various applications like Facebook, Gmail, Mail, PDF, YD2, M, Instagram, etc., and a "Logout" button. A sidebar on the left lists "Overview", "Communication", "TEST_LAB", "Statement", "Submissions" (which is selected), and "Documentation". The main content area shows "test_lab (test_lab) submissions". A green bar indicates a "Score: 30 / 30". Below it is a "Submit a solution" form with a red oval highlighting the file input field ("test_lab: Browse... No file selected.") and the dropdown menu ("C++11 / g++"). Buttons for "Submit" and "Reset" are also present. To the right, a message box says "Submission received" with the note "Your submission has been received and is currently being evaluated." A red arrow points from the CIS logo towards the "GRADER" section. A red circle highlights the "Submit a solution" form. Red text "ส่งไฟล์" is overlaid near the submission form.

Submission received
Your submission has been received and is currently being evaluated.

Server time: 01:36:28
Time left: 42389:23:31

test_lab (test_lab) submissions

Score:
30 / 30

Submit a solution

test_lab: No file selected.

Submit Reset

Previous submissions

Time	Status	Score
1:33:25 AM	Evaluated	details 30 / 30
1:32:34 AM	Evaluated	details 30 / 30
1:28:27 AM	Evaluated	details 10 / 30
1:24:42 AM	Evaluated	details 10 / 30
1:22:09 AM	Evaluated	details 10 / 30
1:20:28 AM	Evaluated	details 10 / 30
1:19:14 AM	Evaluated	details 1 / 30
1:13:48 AM	Evaluated	details 1 / 30
1:12:47 AM	Evaluated	details 0 / 30



ค่ายโอลิมปิกวิชาการ วิชาคอมพิวเตอร์

POINTER

ผศ.ดร.เนียบวุฒิ รัตนวิไลสกุล chaibwoot.r@sci.kmutnb.ac.th

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าฯ พระนครเหนือ

พอยน์เตอร์

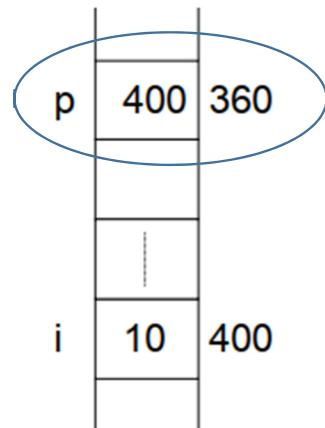
- เป็นชนิดข้อมูลชนิดหนึ่งของภาษา C (int, float, char)
- **มีความเร็วในการทำงานสูง**
- ช่วยประหยัดเนื้อที่ในหน่วยความจำหลักขณะประมวลผล เมื่อเทียบกับอาร์เรย์ (array)
- ใช้ตัวซีร่วมกับฟังก์ชันเพื่อเพิ่มประสิทธิภาพการเขียนโปรแกรม
- ตัวแปรชนิดพอยน์เตอร์จะเก็บค่าที่อยู่ของหน่วยความจำหลัก ซึ่งต่างกับตัวแปรปกติที่เก็บค่าที่แท้จริงของข้อมูล
- การใช้ตัวแปรชนิดพอยน์เตอร์จะเป็นการเข้าถึงข้อมูลหรือเป็นการอ้าง ถึงตำแหน่งที่เก็บข้อมูล

การประกาศตัวแปรประเภทพอยน์เตอร์

`int *ip;` เป็นการประกาศตัวแปร `ip` ให้เป็นตัวแปรพอยน์เตอร์ที่ซึ่งไปยังตัวแปรประเภท `int`
`double *dp;` เป็นตัวแปรพอยน์เตอร์ที่ซึ่งไปยังตัวแปรประเภท `double`

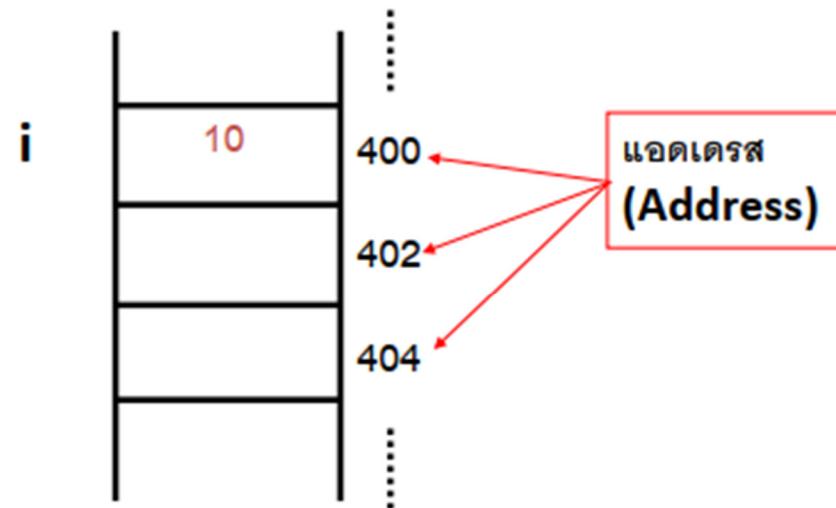
ภาพจำลองการแทนข้อมูลในหน่วยความจำแบบปกติ

```
int i;
i = 10;
```



```
int i = 10;
int *p = &i;
```

```
cout<<i<<endl;
cout<<p<<endl;
cout<<&i<<endl;
cout<<*p<<endl;
```



ตัวแปร pointer

`int *p = &i;` คือ p เก็บค่าของตำแหน่ง คือ 400
และในช่อง 400 หรือ i เก็บ ค่าจริงๆ คือ 10



ตัวชี้และอาร์กิวเม้นท์ของฟังก์ชัน

- เนื่องจากภาษาซีมีการส่งอ้างอิงแบบ By Value และฟังก์ชันสามารถคืนค่า (return) ค่าได้เพียงหนึ่งค่า หากต้องการให้ฟังก์ชันมีการเปลี่ยนแปลงค่าและคืนค่ากลับมา�ัง ฟังก์ชันที่เรียกว่ามากกว่าหนึ่งค่าจะต้องนำพอยน์เตอร์เข้ามาช่วย
- อาร์กิวเม้นท์ที่เป็นประเภทพอยน์เตอร์จะช่วยให้ฟังก์ชันสามารถเปลี่ยนค่าให้กับตัวแปรที่ส่งเข้ามาได้ เนื่องจากอาร์กิวเม้นท์นั้นจะเก็บแอดเดรสของตัวแปรที่ส่ง เข้ามา เมื่อมีการเปลี่ยนแปลงค่าของอาร์กิวเม้นท์ผ่าน reference ค่าของตัวแปรที่ส่งเข้ามาจะถูกเปลี่ยนค่าพร้อมกันในทันที
- จากตัวอย่างนี้ เราไม่ต้อง return แต่เราสามารถกำหนดค่าให้ตัวแปรได้โดย คือตัวแปร x กับ y โดยอาศัยการแก้ค่าโดย pass by reference

```
void swap (int *px, int *py)
{
    int temp;
    temp = *px;
    *px = *py;
    *py = temp;
}

int main ( )
{
    int x = 5, y = 10;
    printf("Before swap : x = %d", x, "y = %d\n", y);
    swap ( &x, &y );
    printf("After swap : x = %d", x, "y = %d\n", y);
}
```



ตัวชี้กับอาร์เรย์

- อาร์เรย์เป็นประเภทข้อมูลที่เก็บชุดของข้อมูลประเภทเดียวกัน มักใช้กับการทำงานที่ต้องทำงานกับตัวแปรชนิดเดียวกันหลายตัวที่มี การทำงานเหมือนกัน เช่น คะแนนของนักศึกษาภายในห้อง 20 คน เป็นต้น อาร์เรย์ในภาษาซีจะนำหลักการของพอยน์เตอร์เข้ามาใช้
- การทำงานใด ๆ ของอาร์เรย์สามารถใช้พอยน์เตอร์เข้ามาแทนที่
- อาร์เรย์ = pointer

int v[4]; char s1[2]; float f[10];

v[0] = 10; Start at 0

v[3] = 15; END

v[4] = 5; Error ค่าที่ผิดไป การอ้างข้อมูลต้องอยู่ในขอบเขต



ให้อ่านค่าของจำนวนเต็ม 5 จำนวนจากคีย์บอร์ด และแสดงผลในลำดับที่กลับกัน

```
# include <stdio.h>
# define SIZE 5
main ( )
{
    int table[SIZE];
    for (int k = 0; k < SIZE; k++)
        scanf ("%d", &table[k]);
    for (int k = SIZE-1; k >= 0; k--)
        printf ("%d\n", table[k]);
}
```



การใช้ตัวชี้กับอาร์เรย์

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

```
int *pa;
```

กำหนดให้พอยน์เตอร์ pa ซึ่งไปยังอาร์เรย์ a ด้วยคำสั่ง

```
pa = &a[0];      หรือ pa = a;
```

pa จะเก็บค่าแอดเดรสเริ่มต้นของอาร์เรย์ a

```
pa+1;            การอ้างถึงแอดเดรสของ a[1]
```

```
*(pa+1);        ค่าของ a[1]
```

```
char *s;
```

```
char s[];
```

f (&a[2]) หรือ f (a+2) เป็นการส่ง แอดเดรสของสมาชิก a[2] ให้กับฟังก์ชัน f สามารถทำได้โดยการประกาศ

```
f (int arr[]){ ..... }
```

```
f (int *arr){ ..... }
```



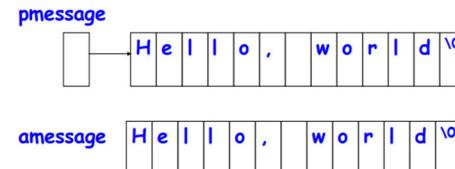
ตัวชี้ตัวอักขรและฟังก์ชัน

```
int strlen (char *s)
{
    char *p = s;
    while (*p != '\0')
        p++;
    return p-s;
}
```

เนื่องจาก s ชี้อยู่ที่ตำแหน่งแรกเริ่มต้น โดยมี p ชี้ไปที่ s เช่นเดียวกัน แต่จะมีการเลื่อน p ไปทีละหนึ่งตำแหน่ง จนกว่าค่าที่ตำแหน่งที่ p ชี้อยู่จะเท่ากับ ' $\0$ ' เมื่อนำ p ค่าสุดท้ายมาลบกับ s ที่ตำแหน่งเริ่มต้นก็จะได้ความยาวของข้อมูลที่ส่งเข้ามา

ตัวชี้ตัวอักขรและฟังก์ชัน

- การทำงานกับข้อความหรือที่เรียกว่า **สตริง (String)** หรืออาร์เรย์ของข้อมูลประเภท **char** หรืออาจจะใช้พอยน์เตอร์ซึ่งไปยังข้อมูลประเภท **char** การทำงานกับค่าคงที่ สตริง (String Constant) สามารถเขียนภาษาในเครื่อง “ ” เช่น “I am a string”
- เมื่อมีการใช้ค่าคงที่สตริงจะมีการพื้นที่ในหน่วยความจำเท่ากับความยาวของค่าคงที่ สตริงบวกด้วย 1 เนื่องจากลักษณะการเก็บข้อมูลประเภทข้อความในหน่วยความจำจะมีการປะตัวอักษร null หรือ ‘\0’ ต่อท้ายเสมอเพื่อให้รู้ว่าเป็นจุดสิ้นสุดของข้อมูล
- การจองพื้นที่ดังกล่าวจะเหมือนการจองพื้นที่ของข้อมูลประเภทอาร์เรย์ เป็นอาร์เรย์ของ **char**
- ค่าคงที่สตริงที่พบเห็นได้เสมอได้แก่ข้อความที่ใช้ในฟังก์ชัน **printf()** เช่น **printf(“Hello, world\n”);**
- ฟังก์ชัน **printf()** จะรับพารามิเตอร์เป็นพอยน์เตอร์ซึ่งไปยังแอดเดรสของข้อมูลที่ตำแหน่งเริ่มต้นของอาร์เรย์ และนำข้อความนั้นแสดงออกทางอุปกรณ์แสดงข้อมูลมาตราฐาน
- ในการเขียนโปรแกรมจะสามารถใช้พอยน์เตอร์ซึ่งไปค่าคงที่สตริงได ๆ ก็ได เช่น **char *pmassage = “Hello, world”;**
- **pmassage** จะเป็นพอยน์เตอร์ประเภท **char** ซึ่ไปที่อาร์เรย์ของตัวอักขร จะแตกต่างจากการใช้อาร์เรย์ทั่วไป เช่น **char amessage[] = “Hello, world”;**





การประกาศตัวแปรชี้ (pointer) ซึ่งไปยัง STRUCT

- กรณี การส่งอ้างอิงmenที่เป็นตัวแปร struct จะไม่เหมาะกับ struct ที่มีขนาดใหญ่น่องจากทุกรังที่ส่งตัวแปร struct จะเป็นการสำเนาตัวแปรตัวใหม่ขึ้นมาในฟังก์ชันซึ่งจะทำให้ซ้ำและเปลี่ยนพื้นที่หน่วยความจำ เราจะใช้พอยน์เตอร์เข้ามาช่วยแก้ปัญหานี้
- โดยสิ่งแวดล้อมเดรสของตัวแปร struct anyak พังก์ชันซึ่งรับอ้างอิงmenที่ เป็นพอยน์เตอร์อ้างอิงmenที่จะ ซึ่งไปยัง แอดเดรสเริ่มต้นของตัวแปร struct จะช่วยให้การทำงานเร็วขึ้นและเปลี่ยนหน่วยความจำน้อยลง แต่ สิ่งที่ต้องระวังคือ หากมีการเปลี่ยนแปลงค่าที่อ้างอิงmenที่พอยน์เตอร์ซื้อยู่ ค่าในตัวแปร struct ที่ส่งมายังฟังก์ชันจะเปลี่ยนตามโดยอัตโนมัติ



การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง STRUCT

```
struct point origin;  
  
point *pp;  
  
pp = &origin;  
  
printf ( "origin is (%d, %d)\n", (*pp).x, (*pp).y );  
  
printf ( "origin is (%d, %d)\n", pp->x, pp->y );
```

(*pp).x จะไม่เหมือนกับ *pp.x เนื่องจากเครื่องหมาย . จะมีคำตั้งความสำคัญสูงกว่า *

- การอ้างถึงสมาชิกโครงสร้างโดยใช้เครื่องหมาย ->

```
ptrdate->day = 7;  
  
scanf ( "%d", &ptrdate->year );
```

- การอ้างถึงสมาชิกของโครงสร้างผ่านตัวแปรอยู่ในเทอร์

```
(*ptrdate).day = 7;  
  
scanf ( "%d", &((*ptrdate).year) );
```

การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง STRUCT

```
typedef struct {  
    int day;  
    int month;  
    int year;  
} Date;
```

```
Date today;
```

```
Date *ptrdate;
```

แบบที่ 1

การประกาศแบบ
ข้อมูลโครงสร้าง

การประกาศตัวแปร
ข้อมูลแบบโครงสร้าง

การประกาศตัวแปร
pointer ชี้ไปยังโครงสร้าง

```
typedef struct {  
    int day;  
    int month;  
    int year;  
} Date;
```

```
typedef Date *PtrDate;  
  
PtrDate ptrdate;
```

แบบที่ 3

การประกาศแบบ
ข้อมูลโครงสร้าง

การประกาศประเภท
ตัวแปร pointer ชี้ไปยัง
โครงสร้าง

การประกาศตัวแปร
pointer ชี้ไปยัง โครงสร้าง

```
struct date {  
    int day;  
    int month;  
    int year;  
} *ptrdate;
```

แบบที่ 2



การประกาศตัวแปรชี้ (pointer) ชี้ไปยัง STRUCT

```
struct date
{
    int day;
    int month;
    int year;
};

main ( )
{
    struct date      today;
    struct date     *ptrdate;

    ptrdate = &today;
    ptrdate->day = 27;
    ptrdate->month = 9;
    ptrdate->year = 1985;
    printf ( "Today's date is %2d/%2d/%4d\n", ptrdate->day, ptrdate->month, ptrdate->year );
}
```



การใช้งาน Pointer

```
#include<stdio.h>
void main( )
{
    int N = 100;
    int *pN;           ประกาศ
    pN = &N;           เซ็ตค่า

    printf ( "%p\n", &N );
    printf ( "%p\n", pN );      ตัวมันเป็น address

    printf ( "%d\n", N );
    printf ( "%d\n", *pN );    ค่าของมัน

}
```



การใช้งาน Pointer of FUNCTION

```
#include<stdio.h>

void sum_value( int a, int b ) //pass by value
{
    a = 3;
    b = 4;
    printf( "%d %d\n", a, b );
}

void sum_ref( int *a, int *b ) //pass by reference
{
    *a = 5;
    *b = 6;
    printf( "%d %d\n", *a, *b );
}

void main( )
{
    int x = 1;
    int y = 2;
    printf( "%d %d\n", x, y );
    sum_value(x,y);
    printf( "%d %d\n", x, y );
    sum_ref(&x,&y);
    printf( "%d %d\n", x, y );
}
```

1 กับ 2
3 กับ 4
เหมือนเดิม 1 กับ 2
ส่ง address ให้ x กับ y คือ 5 กับ 6
x กับ y คือ 5 กับ 6



การใช้งาน Pointer of Array

```
#include<stdio.h>
void main( )
{
    int x[] = {1,2,3,4,5};
    printf( "%p\n", x);
```

```
int *y = x;
```

x คือ pointer และ x คือ Array

```
printf( "%p\n", y);
```

```
printf( "%d\n", x[0]);
```

อ้างอิงตำแหน่ง

```
printf( "%d\n", *(y+0) );
```

อ้างอิงตำแหน่ง

```
printf( "%d\n", x[1]);
```

อ้างอิงตำแหน่ง

```
printf( "%d\n", *(y+1) );
```

อ้างอิงตำแหน่ง

```
}
```

การใช้งาน Pointer of String

```
char *s1 = "my name is be1";
printf( "%s\n", s1);
printf( "%c\n", *(s1+1) );
char s2[] = "my name is be2";
printf( "%s\n", s2);
printf( "%c\n", s2[3] );
s1 = "I love U";
//s2 = "I love U";
//(s1+2) = 'x';
s2[2]      = 'x';
printf( "%s\n", s2);
char *s3[] = {"s1","s2","s3"};
printf( "%s\n", s3[0] );
s3[1] = "GOD";
printf( "%s\n", s3[1] );
char *s4 = s2;
s4[2] = 'k';
printf( "%s\n", s2);
printf( "%s\n", s4);
```

ข้อความเป็น pointer

แสดงทั้งหมด

เอกสารตัวเดียว

ข้อความเป็น []

เอกสารตัวเดียว

เปลี่ยนทั้งอันทำได้

//Error ถ้าเป็น Array ทำไม่ได้

//Error

ทำได้

ข้อความเป็น pointer + array

การ copy string ต้องระวัง เพราะถ้าเขียน char *s4 = s1;
ถ้าต้นฉบับเป็น Array สามารถแก้ไขค่าได้
s1 กับ s4 เชื่อมกัน ถ้าเปลี่ยน เปลี่ยนทั้งคู่



การใช้งาน Pointer of String

```
char *s1 = "test";  
char s2[4];  
char s3[4];  
char *s4 = s3;
```

printf("%s\n", s1); สามารถทำได้

scanf("%s",s1); ไม่สามารถทำได้ ถ้าเป็น pointer
printf("%s\n", s1); สามารถทำได้

scanf("%s",s2); สามารถทำได้ เป็น Array สามารถทำได้
printf("%s\n", s2); สามารถทำได้

scanf("%s",s4); สามารถทำได้ ถ้ากรณีเป็น Array มาก่อน
printf("%s\n", s4); สามารถทำได้

```
char name[100][100];  
scanf("%s",name[i]);
```

```
int num[100];  
scanf("%d",&num[i]);
```

กรณีใช้ String Array ควรใช้ Array 2 มิติ
อ้างอิง Array 2 มิติ

หรืออาจจะใช้ pointer ก็ได้



การใช้งาน Pointer of STRUCT

```
#include<stdio.h>
#include<string.h>
struct Books
{
    char title[50];
    int book_id;
};

void printBook( struct Books *book )
{
    printf( "Book title : %s\n", book->title);
    printf( "Book book_id : %d\n", book->book_id);
}

void main( )
{
    struct Books Book1;
    strcpy( Book1.title, "C Programming");
    Book1.book_id = 6495407;

    printf( "Book 1 title : %s\n", Book1.title);
    printf( "Book 1 book_id : %d\n", Book1.book_id);

    printBook( &Book1 );
}
```

ควร import string เพราะต้องใช้สำหรับการใส่ค่า struct ให้ function ข้างในไม่ได้ จึงต้อง pass parameter โดย pointer

รับค่าเป็น pointer

ถ้ากรณีอ้างอิงผ่าน pointer ใช้ ->

สร้าง structure

พยายามใช้ฟังก์ชันสำเร็จรูป

ถ้ากรณีปกติใช้ “.”



สรุป Pointer

#include<bits/stdc++.h> → ใช้ตัวนี้ตัวเดียวสามารถใช้งานได้หมด

using namespace std; → ใช้ตัวนี้ตัวเดียวสามารถใช้งานได้หมด

```
void f_val( int a)           { a = 2; }                                //Declare
void f_ref( int *a)          { *a = 2; }                               //value, ref
struct A                      { int n; char s[]; };
void f_s(struct A *a)         { cout<<a->n<<" "<<a->s<<endl; }      //Array
int main()
{
    int i = 1;      int *p = &i;  cout<<i<<" "<<p<<" "<<*p<<endl;           //String
    f_val(i); cout<<i<<" "; f_ref(p); cout<<i<<endl;
    int arr[4] = {1,2,3,4};      int *pt = arr;             cout<<arr[2]<<" "<<*(pt+2)<<endl;   //Error input
    char *s1 = "message1";      char s2[] = "message2";  cout<<s1<<" "<<s2<<endl;           //Can input
    s1 = "Can do";
    s2[0] = 'T';
    //s2 = "Error size";

    //cin>>s1;
    cin>>s2;
    cout<<s2<<endl;

    char str1[20] = "C1"; char str2[20]; strcpy(str2, str1); cout<<strcmp(str1, str2)<<endl;      //function string
    struct A a;   strcpy( a.s, "msg");      a.n = 4;       cout<<a.n<<" "<<a.s<<endl;           f_s( &a );
    return 0;
}
```



ค่ายโอลิมปิกวิชาการ วิชาคอมพิวเตอร์

โครงสร้างข้อมูลแบบเชิงเส้น (Linear data structures)

ปรับปรุงจาก เอกสารของ ผศ.ดร. ลีอ้อน พิพานเมฆาภรณ์ และ ผศ.ดร. สรร รัตนสัญญา

ผศ.ดร.เนียบวุฒิ รัตนวิไลสกุล chaibwoot.r@sci.kmutnb.ac.th

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าฯ พระนครเหนือ

- อาร์เรย์ (array)
- สแตก (Stack)
- คิว (Queue)
 - คิวแบบง่าย (simple queue)
 - คิวงกลม (circular queue)
 - คิวลำดับความสำคัญ (priority queue)

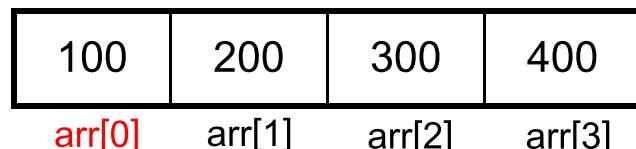


โครงสร้างข้อมูลเชิงเส้น (Linear data structure)

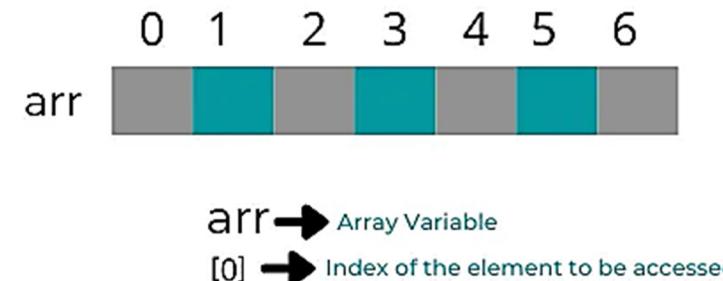
- โครงสร้างข้อมูล (Data Structure) เป็นเครื่องมือในการจัดเก็บข้อมูลในรูปแบบที่มีโครงสร้างไว้ในคอมพิวเตอร์ เพื่อช่วยให้สามารถประมวลผลข้อมูลได้อย่างมีประสิทธิภาพ แบ่งออกเป็น 2 ประเภท ได้แก่
- โครงสร้างข้อมูลแบบเชิงเส้น (Linear data structure) เป็นโครงสร้างข้อมูลที่จัดเก็บข้อมูลในลักษณะต่อเนื่องกัน เช่น อาร์เรย์ สแตก และคิว เป็นต้น
- โครงสร้างข้อมูลแบบไม่เชิงเส้น (Non-linear data structure) โครงสร้างที่ไม่มีคุณสมบัติของเชิงเส้น สามารถใช้แสดงความสัมพันธ์ของข้อมูลที่ซับซ้อนได้มากกว่า โครงสร้างข้อมูลแบบเชิงเส้น เช่น ทรี (tree) กราฟ (graph)

อาร์เรย์ (Array)

- อาร์เรย์ (array) เป็นรายการข้อมูลที่จัดเก็บไว้ในหน่วยความจำ โดยที่ข้อมูลแต่ละตัวจะถูกเก็บไว้ในตำแหน่งที่ติดกัน ซึ่งจะสามารถเข้าถึงข้อมูลแต่ละตัวได้โดยการระบุเลขดัชนี (index)
 - ในภาษาซี เลขดัชนีจะเริ่มต้นที่ตำแหน่ง 0 เป็นอ (สมाचิกตัวแรก)
- ในการใช้งานอาร์เรย์จะต้องมีการกำหนดขนาด (size) ของอาร์เรย์ที่แน่นอน



อาร์เรย์ (Array)



ตัวอย่างการเตรียมใช้งานอาร์เรย์

- การระบุประเภท (type of array) และขนาดของอาร์เรย์

- int arr[10]; // arr[0], arr[1], , arr[9]

- การระบุประเภทพร้อมกำหนดค่าข้อมูลเริ่มต้น

- int arr[] = {10, 20, 30, 50, 100, -100}; // arr[0]=10, arr[1]=20,...,arr[5] = -100



Dynamic Memory allocation using malloc()

- โดยทั่วไปการกำหนดขนาดของอาร์เรย์จะมีลักษณะเป็น static กล่าวคือจะต้องกำหนดขนาดไว้ล่วงหน้า และไม่สามารถเปลี่ยนแปลงขนาดได้ขณะรันโปรแกรม
- อย่างไรก็ตาม ในภาษาซีมีวิธีการกำหนดขนาดของอาร์เรย์**ในขณะรันโปรแกรม**ได้ โดยใช้ฟังก์ชัน **malloc()** `ptr = (type *)malloc(N * sizeof(type));`

CODE

```
int *a;  
int n = 0;  
cin>>n;  
a = (int*)malloc(n * sizeof(int));  
for(int i=0;i<100;i++){a[i] = i;}  
for(int i=0;i<100;i++){cout<<a[i]<<endl;}
```

ปัญหาทั่วไปในการใช้อาร์เรย์

กำหนดให้อาร์เรย์มีข้อมูล n จำนวน

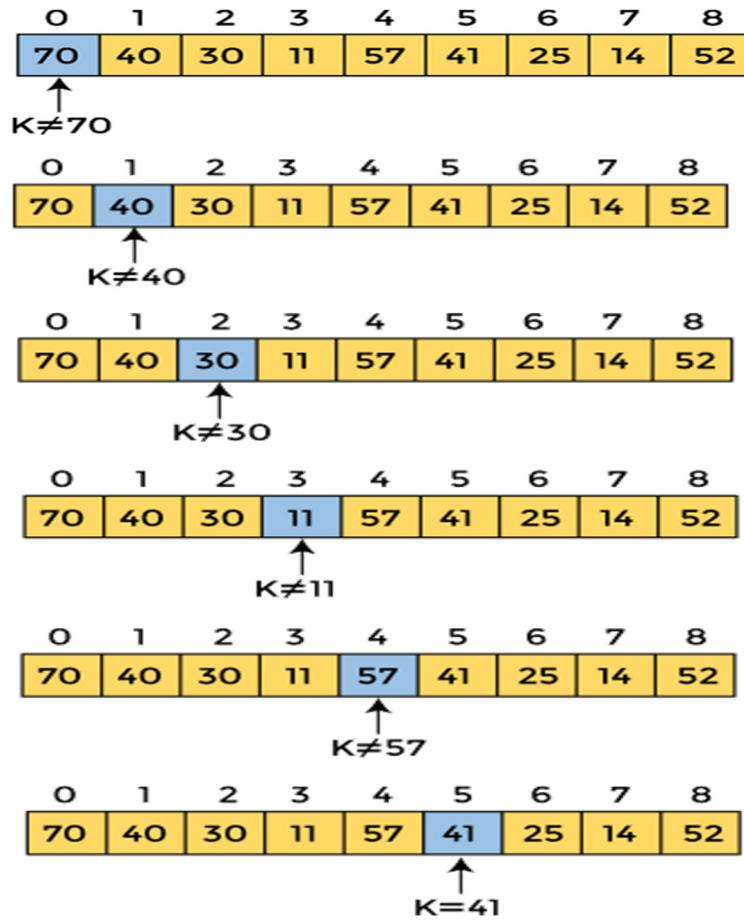
■ การค้นหาข้อมูล (Searching) ในอาร์เรย์

- กำหนดให้ k เป็นอินพุตที่ต้องการค้นหา
- ค้นหาตำแหน่งสماชิกในอาร์เรย์ที่มีค่าเท่ากับ k
- อัลกอริทึมค้นหาข้อมูล เช่น linear search (sequential search), binary search

■ การเรียงลำดับข้อมูล (Sorting) ในอาร์เรย์

- ระบุรูปแบบในการเรียงข้อมูล: น้อยไปมาก (ascending order) มากไปน้อย (descending order)
- เรียงข้อมูลในอาร์เรย์ตามรูปแบบที่กำหนด
- อัลกอริทึมเรียงข้อมูล เช่น Bubble sort, Insertion sort, Selection sort

Linear Search



Binary Search

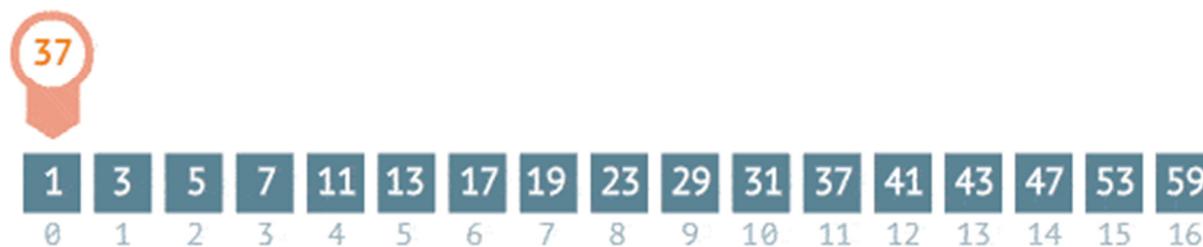
Binary search

steps: 0

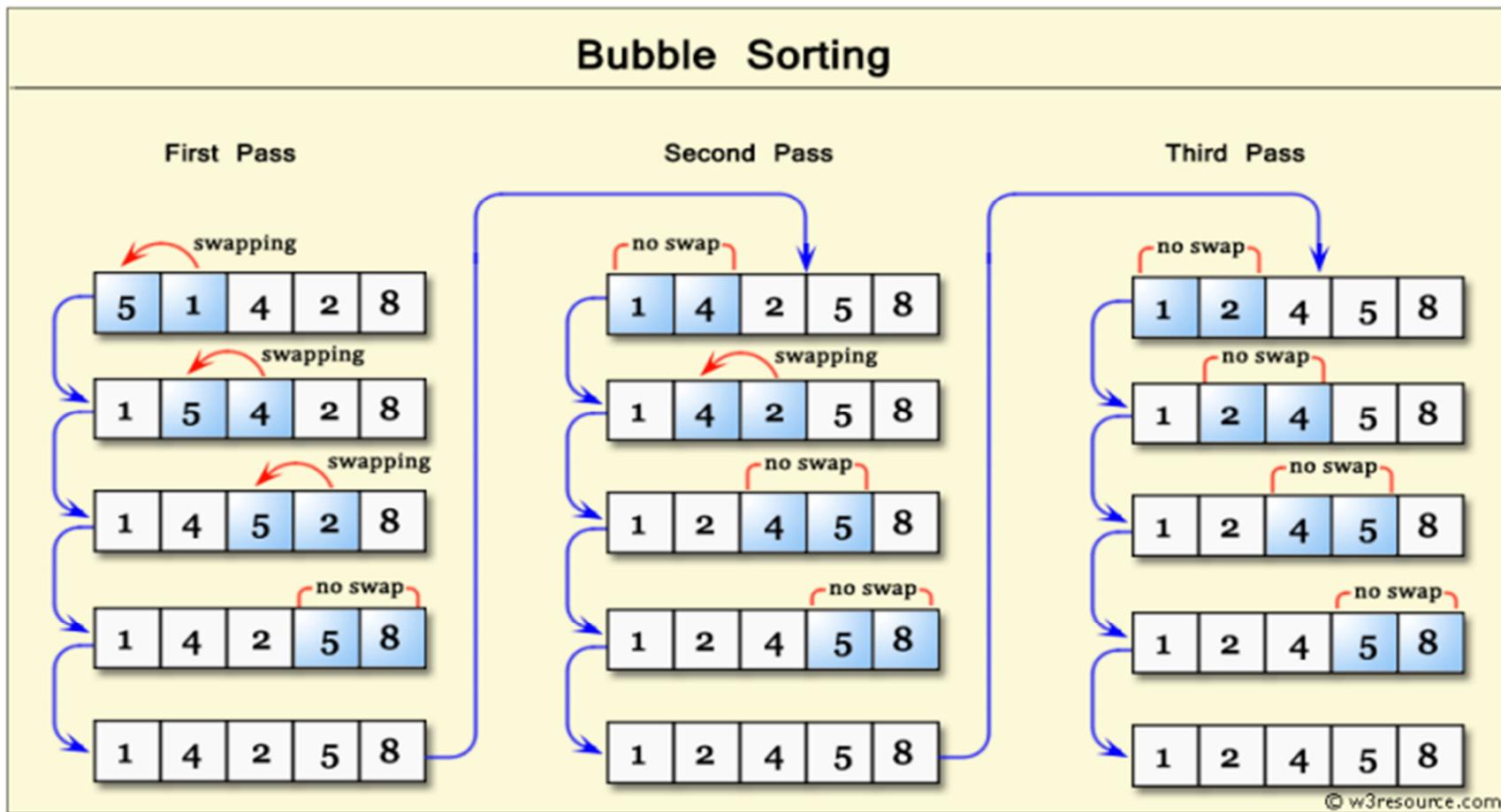


Sequential search

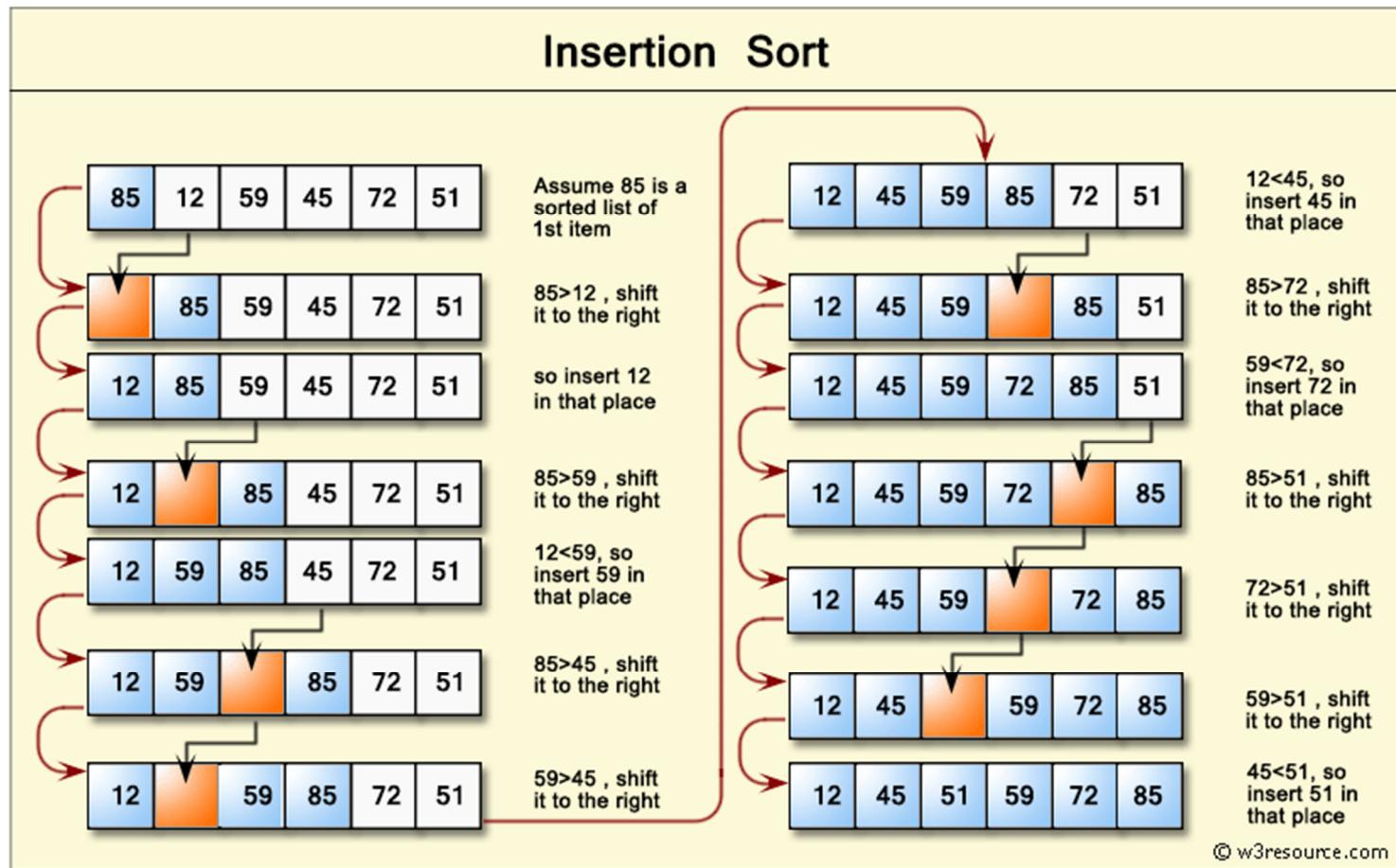
steps: 0



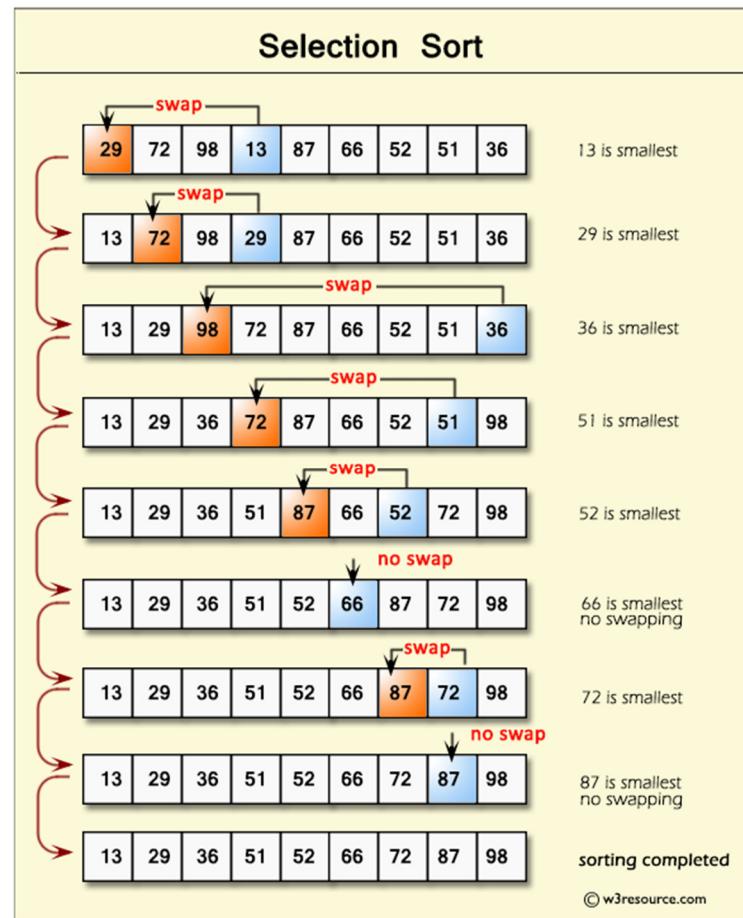
Bubble Sort



Insertion sort



Selection sort



Time complexity

- ตัวชี้วัดความซับซ้อนในการทำงานของอัลกอริทึมในเชิงเวลา (time) ซึ่งมีสมมติฐานว่า อัลกอริทึมจะทำงานแตกต่างกันไปตามขนาดข้อมูล (size of data) ที่ป้อนให้
- ในการประเมินเวลาทำงานของอัลกอริทึมจะใช้วิธีนับจำนวนรอบการทำงานของอัลกอริทึมเทียบกับขนาดข้อมูลอินพุต (input size) มากกว่าการจับเวลาจากเครื่องคอมพิวเตอร์ที่รันซึ่งมักขึ้นอยู่กับฮาร์ดแวร์ (hardware) ระบบปฏิบัติการ และภาษาที่ใช้ในการเขียนโปรแกรม
- โดยทั่วไปการประมาณเวลาแบบนับจำนวนรอบมักจะประมาณแบบกว้างๆ (ไม่ต้องการจำนวนรอบแน่นอน แต่มักต้องการคำตอบเป็นจำนวนรอบสูงสุดที่เป็นไปได้) เรียกว่า **worse-case time complexity**
- **worse-case time complexity** มักเขียนในรูปของ $O(f(n))$ อ่านว่า big-oh ของฟังก์ชัน $f(n)$ เมื่อ $f(n)$ เป็นจำนวนรอบทำงานซึ่งสัมพันธ์กับขนาดอินพุต n

Time complexity ของ Linear search

```
1 #include<stdio.h>
2
3 int main() {
4     int arr[] = {70,40,30,11,57,41,25,14,52};
5     int n = sizeof(arr)/sizeof(int);
6     int k = 41, i=0;
7
8     while( (arr[i]!= k) && (i < n))
9         i++;
10
11    if(i<n)
12        printf("found item %d at index %d in arr", k, i);
13    else
14        printf("no %d in arr", k);
15
16    return 0;
17 }
```

อัลกอริทึม linear search จะวนลูปสัมพันธ์กับขนาดของอินพุต ก (ขนาดของอาร์เรย์)

- เมื่อ $n = 10$, ลูป while รัน 10 รอบ
- $n = 100$, ลูป while รัน 100 รอบ

จะเห็นได้ว่าจำนวนรอบสูงสุดของลูป while จะไม่เกินค่าของ n ดังนั้น time complexity ของอัลกอริทึม linear search สามารถเขียนในรูปของ $O(n)$

หรือกล่าวอีกนัยว่า “การค้นหาข้อมูลใน linear search จะทำงานไม่เกิน n รอบ”

Time complexity ของ Binary Search

```

1 #include <iostream>
2 using namespace std;
3
4 int binarySearch(int array[], int low, int high, int key) {
5
6     while (low <= high) {
7         int mid = low + (high - low) / 2;
8
9         if (key == array[mid])
10            return mid;
11         if (key > array[mid])
12            low = mid + 1;
13         if (key < array[mid])
14            high = mid - 1;
15     }
16
17     return -1;
18 }
```

อัลกอริทึม binary search จะตรวจสอบค่าของ key กับตำแหน่งต่างๆ ของข้อมูลใน array และทำการลดขนาดของ array ลงในทุกรอบของการทำงาน

- เมื่อ $n = 8$, ลูป while รันมากสุด 3 รอบ
- $n = 16$, ลูป while รันมากสุด 4 รอบ

จะเห็นได้ว่าจำนวนรอบสูงสุดของลูป while จะไม่เกินค่าของ $\log_2 n$ ดังนั้น time complexity ของอัลกอริทึม linear search สามารถเขียนในรูปของ $O(\log n)$

หรือกล่าวอีกนัยว่า “การค้นหาข้อมูลใน binary search จะทำงานไม่เกิน $\log n$ รอบ”

Time complexity ของ Bubble Sort

```
1 #include<stdio.h>
2
3 int main() {
4     int arr[] = {5,1,4,2,8}, temp;
5     int n = sizeof(arr)/sizeof(int);
6     for(int i=0; i<n-1; i++)
7         for(int j = i+1; j<n; j++)
8             if(arr[i] > arr[j])
9                 { temp = arr[i];
10                   arr[i] = arr[j];
11                   arr[j] = temp;
12               }
13
14     for(int i=0; i<n; i++)
15         printf("%d ", arr[i]);
16     return 0;
17 }
```

อัลกอริทึม Bubble sort มีลูปซ้อนกัน 2 ลูป ได้แก่ ลูป i ทำหน้าที่ควบคุมจำนวนการเรียงข้อมูลในแต่ละรอบ และลูป j ทำหน้าที่เปรียบเทียบสมาชิกทุกตัวที่อยู่ติดกัน

- เมื่อ $i=0 \rightarrow$ ลูป j ทำงาน $n-1$ รอบ
- เมื่อ $i=1 \rightarrow$ ลูป j ทำงาน $n-2$ รอบ
-
- เมื่อ $i=n-2 \rightarrow$ ลูป j ทำงาน 1 รอบ

จำนวนรอบทำงานของอัลกอริทึม Bubble sort จะเกิดจากผลรวมของลูป j
 $= 1+2+3+4+\dots+(n-2)+(n-1) \cong O(n^2)$

หรือกล่าวอีกนัยว่า “การเรียงลำดับข้อมูลใน bubble sort จะทำงานไม่เกิน n^2 รอบ”

Time complexity ของ Insertion sort

```

1 void insertion_sort(int array[], int n){
2     int i, j, tmp;
3
4     for(i=1; i<n; i++){
5         tmp = array[i];
6         j = i-1;
7         while(j>=0 && array[j] > tmp){
8             array[j+1] = array[j];
9             j--;
10        }
11        array[j+1] = tmp;
12    }
13 }
```

อัลกอริทึม Insertion sort มีลูปซ้อนกัน 2 ลูป ได้แก่ ลูป i ทำหน้าที่แทรกสมาชิกในตำแหน่งที่ถูกต้องในแต่ละรอบ และลูป j ทำหน้าที่ตรวจสอบตำแหน่งที่ถูกต้องของสมาชิกแต่ละตัว

- เมื่อ $i=1 \rightarrow$ ลูป j ทำงานมากสุด 2 รอบ
- เมื่อ $i=2 \rightarrow$ ลูป j ทำงานมากสุด 3 รอบ
-
- เมื่อ $i=n \rightarrow$ ลูป j ทำงานมากสุด $n-1$ รอบ

จำนวนรอบการทำงานของอัลกอริทึม Insertion sort จะเกิดจากผลรวมของ ลูป j $= 1+2+3+4+\dots+(n-2)+(n-1) \cong O(n^2)$

หรือกล่าวอีกนัยว่า “การเรียงลำดับข้อมูลใน insertion sort จะทำงานไม่เกิน n^2 รอบ”

Time complexity ของ Selection sort

```

1 void selection_sort(int array[], int n){
2     int i,j,min_index,tmp;
3
4     for(i=0;i<n-1;i++){
5         min_index = i;
6         for(j=i+1;j<n;j++){
7             if(array[j] < array[min_index]){
8                 min_index = j;
9             }
10        }
11        tmp = array[min_index];
12        array[min_index] = array[i];
13        array[i] = tmp;
14    }
15 }
```

อัลกอริทึม Selection sort มีลูปซ้อนกัน 2 ลูป ได้แก่ ลูป i ทำหน้าที่ควบคุมการสลับตำแหน่งข้อมูลในแต่ละรอบ และลูป j ทำหน้าที่ตรวจสอบตำแหน่งที่ถูกต้องของสมาชิกแต่ละตัว

- เมื่อ $i=0 \rightarrow$ ลูป j ทำงาน $n-1$ รอบ
- เมื่อ $i=1 \rightarrow$ ลูป j ทำงาน $n-2$ รอบ
-
- เมื่อ $i=n-1 \rightarrow$ ลูป j ทำงาน 1 รอบ

จำนวนรอบการทำงานของอัลกอริทึม Selection sort จะเกิดจากผลรวมของ ลูป $j = 1+2+3+4+\dots+(n-2)+(n-1) \cong O(n^2)$

หรือกล่าวอีกนัยว่า “การเรียงลำดับข้อมูลใน selection sort จะทำงานไม่เกิน n^2 รอบ”



Array in C++ STL

- #include <array>
 - [], at()
 - front(), back()
 - empty(), fill()
 - swap() 
- #include <tuple>
 - get()

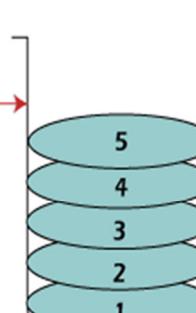
```
int a[10] = {1,2,3,4,5,6,7,8,9,10};  
for(int i=0;i<10;i++){cout<<a[i]<<" ";} cout<<endl;  
swap(a[0],a[5]);  
for(int i=0;i<10;i++){cout<<a[i]<<" ";}
```

stack

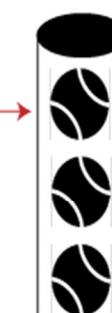
- เป็นโครงสร้างข้อมูลเชิงเส้นประเภทหนึ่งที่สามารถพัฒนาโดยใช้อาร์เรย์
- สแตก (stack) มีการจัดการข้อมูลแบบ **เข้าก่อนออกทีหลัง** (last in first out: LIFO) กล่าวคือ เมื่อมีการนำข้อมูลมาเก็บไว้ในสแตกจะไม่สามารถนำข้อมูลนี้ออกจากสแตกได้ จนกว่าข้อมูลที่เข้ามาทีหลังจะออกจากสแตกไปก่อนจนหมด
- สแตกเป็น abstract datatype ที่จะต้องพัฒนาໂອเปเรชันขึ้นใช้งานเอง ได้แก่
 - ✓ **push(x)**
 - นำเข้าข้อมูล x ไปเก็บไว้ในสแตก
 - ✓ **y = pop()**
 - นำออกข้อมูลล่าสุดที่เข้ามาในสแตก
- การควบคุมลำดับเข้าออกของข้อมูลจะใช้ ตัวแปร **top** เพื่อชี้จุดเข้าและออกในสแตก



Stack of Coins



Stack of Plates

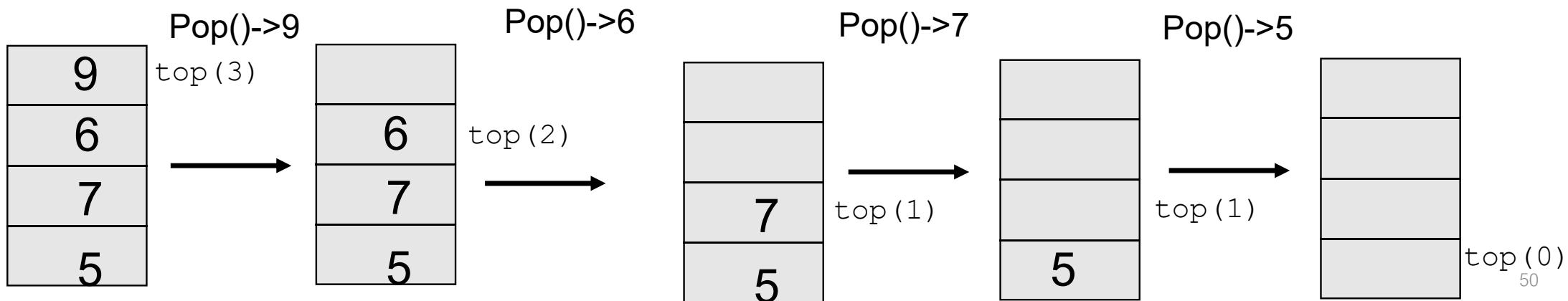
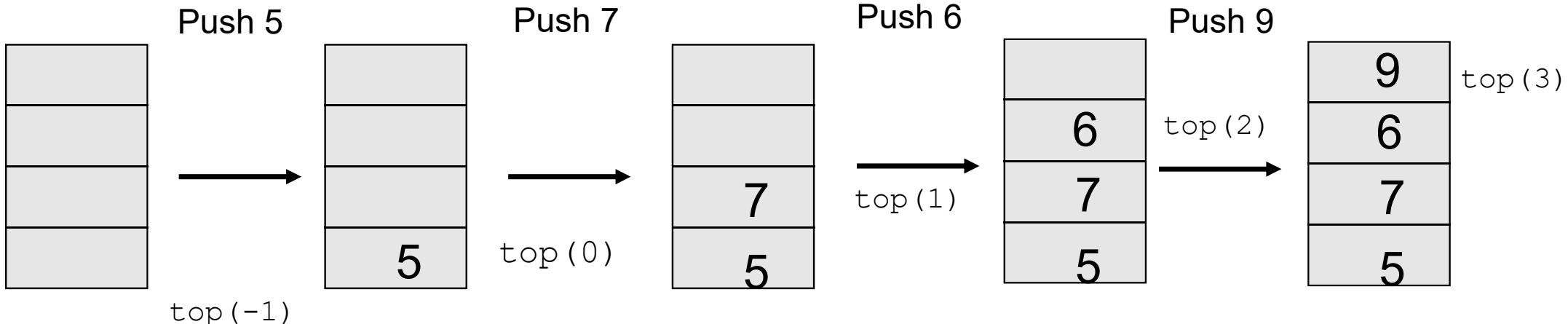


Can of Tennis Balls



Stack of Books

การทำงานของ push และ pop





Stack in C++ STL

- ใน C++ แสตกถูกพัฒนาเป็น container สำหรับร่วมให้ใช้งาน

```
stack<int> s;  
s.push(5);  
s.push(3);  
s.push(9);  
cout<<s.top()<<endl;  
s.pop();  
cout<<s.top()<<endl;
```

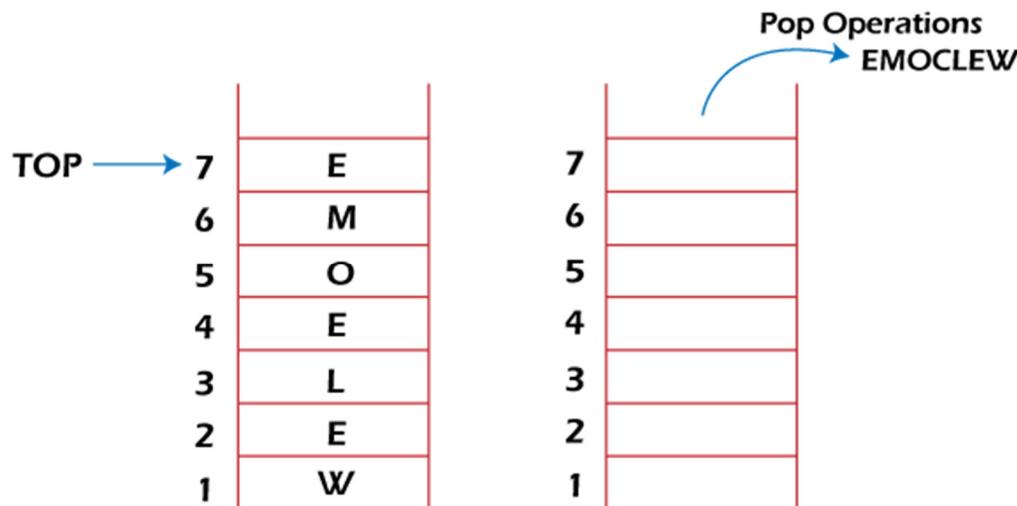


แอพพลิเคชันของสแตก

- การกลับลำดับสตริง (string reversal)
- การเช็ควงเล็บ (delimiter checking)
- การหาค่านิพจน์ (Evaluation of arithmetic expression)

string reversal

- เนื่องจาก stack มีการจัดลำดับข้อมูลที่เข้าและออกจาก stack แบบ LIFO
- ดังนั้นหากต้องการเรียงลำดับของข้อมูลใหม่จากหน้าไปหลัง ให้ push ข้อมูลที่จะตัวลงใน stack จนครบ และ pop ออกมายัง stack ว่าง
- ตัวอย่างเช่น WELEOME -> EMOCLEW



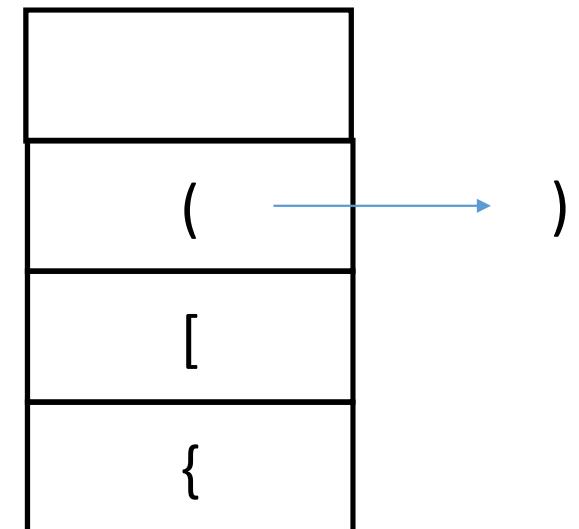
Delimiter checking

- นิพจน์ทางคณิตศาสตร์ส่วนมากจะใช้งเล็บ (parenthesis) เพื่อสร้างลำดับการคำนวณ ซึ่งมีหลายรูปแบบ เช่น
 - () { } []
- ในการตรวจสอบความถูกต้องของการใช้งเล็บในนิพจน์จะต้องตรวจสอบ 2 ข้อ
 1. จำนวนงเล็บเปิดและปิดจะต้องมีจำนวนเท่ากัน
 - { [()] } **OK**
 - { [() } **NOT OK**
 2. งเล็บเปิดและปิดที่จะจับคู่กันได้จะต้องเป็นประเภทเดียวกัน
 - { ([)] } **NOT OK**

Delimiter checking

- สามารถใช้ stack มาช่วยในการตรวจสอบความถูกต้องได้ เช่น { [()] }

1. อ่านข้อมูลจากซ้ายไปขวาทีละตัว
2. push วงเล็บเปิดลงใน stack
3. pop วงเล็บเปิดออกจาก stack ทุกครั้งที่อ่านเจองานปิด
4. หากวงเล็บเปิดและปิดเป็นคนละประเภท ให้จบการทำงาน
5. วนซ้ำจนกระทั่ง stack ว่าง



การคำนวณนิพจน์

- ในการหาผลลัพธ์ของนิพจน์ทางคณิตศาสตร์ โดยทั่วไปมักเขียนนิพจน์ในรูปของ infix notation
- โดยทั่วไปจะใช้ *infix expressions* ในรูปของ

operand1 OPERATOR operand2

- เช่น:
 - $1 + 2$
 - $4 * 3$
- ในการหาผลลัพธ์ของ infix expressions จะเป็นต้องคำนวณตามลำดับความสำคัญของเครื่องหมาย:
 - $1 + 2 * 3$
 - $10 - 4 - 3$
 - $2^3 \cdot 3^3$
 - $1 - 2 - 4^5 \cdot 3^6 / 7^2 \cdot 2^2$
- และต้องใช้วงเล็บเป็นตัวจัดลำดับการหาผลลัพธ์ $(1-2)-(((4^5)*3)*6)/(7^(2^2))$

การคำนวณนิพจน์

- ตัวแปลภาษาสมัยใหม่จะไม่สามารถแปลงนิพจน์ infix ไปเป็นภาษาเครื่องโดยตรงเนื่องจากค่อนข้างซับซ้อน โดยเฉพาะอย่างยิ่งนิพจน์ที่ซับซ้อน
- อย่างไรก็ตาม หากเขียนนิพจน์ infix ให้อยู่ในรูปนิพจน์แบบ postfix หรือ prefix ก่อนก็จะทำให้ง่ายขึ้น
- รูปแบบของ postfix expression

operand1 operand2 operator

- เช่น: $1+2 \rightarrow 1\ 2\ +$ หรือ $4*3 \rightarrow 4\ 3\ *$
- โดยการแสดงนิพจน์ในรูปของ postfix จึงไม่จำเป็นต้องใช้วงเล็บเพื่อกำหนดลำดับความสำคัญ
 - Infix expression:

$1 - 2 - 4 ^ 5 * 3 * 6 / 7 ^ 2 ^ 2$ หรือ $(1-2)-(((4^5)*3)*6)/(7^(2^2))$

- Postfix expression:

$1\ 2\ -\ 4\ 5\ ^\ 3\ *\ 6\ *\ 7\ 2\ 2\ ^\ ^\ /\ -$



การแปลง infix ไปเป็น postfix

1. จานนิพจน์ infix ให้ส่วนเล็บให้กับทุกนิพจน์ย่อยโดยการคำนึงถึงลำดับการคำนวณ (precedence order)
2. เริ่มจากนิพจน์ย่อยในสุดก่อน โดยให้บ้าย operator ไปไว้ตรงกับตำแหน่งของวงเล็บปิดของนิพจน์นั้น
3. ลดวงเล็บออกทั้งหมดก็จะได้นิพจน์แบบ postfix

เช่น $A + B * C$

$(A + (B * C))$

$(A + (B C *))$

$(A (B C *)+)$

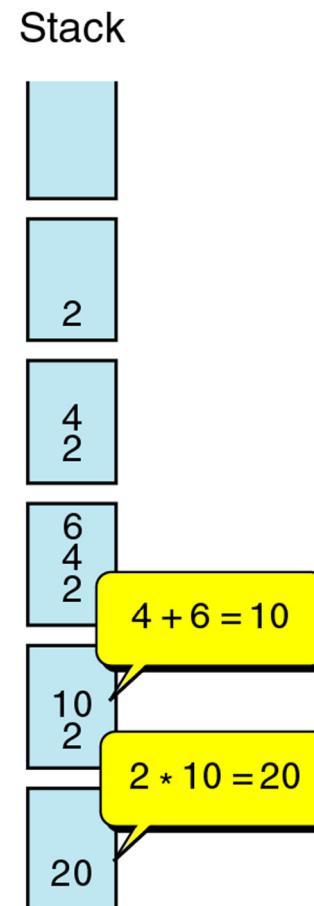
$A B C * +$

การแปลง infix ไปเป็น postfix โดยใช้ stack

Infix	Stack	Postfix
(a) A + B * C - D / E		
(b) + B * C - D / E	A	
(c) B * C - D / E	A	
(d) * C - D / E	AB	
(e) C - D / E	AB	
(f) - D / E	AB	
(g) D / E	ABC	
(h) / E	ABC	
(i) E	ABC	
(j)	ABC	
(k)	ABC	

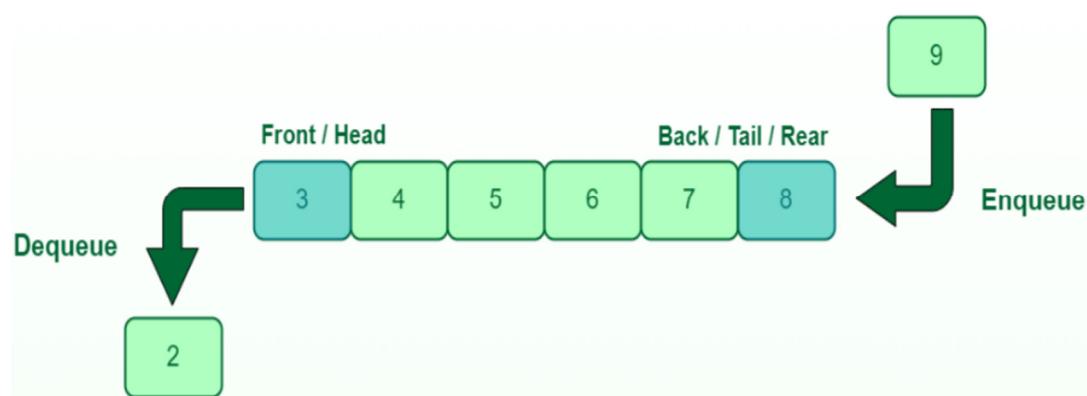
การหาผลลัพธ์นิพจน์ postfix โดยใช้ stack

- Postfix
- (a) 2 4 6 + *
 - (b) 4 6 + *
 - (c) 6 + *
 - (d) + *
 - (e) *
 - (f)



คิว (Queue)

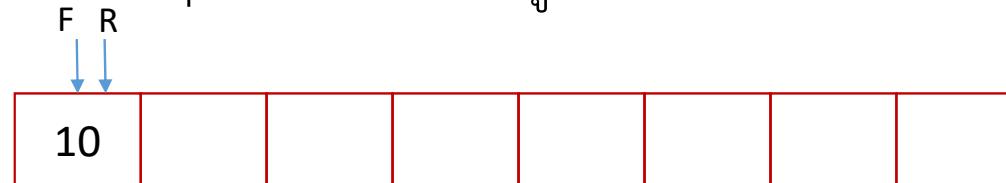
- คิว (queue) เป็นโครงสร้างข้อมูลเชิงเส้นที่มีลำดับการนำเข้าและส่งออกข้อมูลที่แตกต่างจาก stack กล่าวคือ คิวจะมีลำดับการจัดการข้อมูลแบบ ข้าก่อนออกก่อน (First in First out: FIFO)
- โอเปอเรชันในคิว
 - enqueue(x) โอเปอเรชันสำหรับการนำข้อมูล x ไปเก็บไว้ในคิว
 - dequeue() โอเปอเรชันนำข้อมูลออกจากคิว
- การจัดการข้อมูลในคิวจะใช้ตัวแปร 2 ตัว ได้แก่
 - **Front** : ชี้ตำแหน่งที่จะนำข้อมูลออกจากคิว
 - **Rear** : ชี้ตำแหน่งที่จะนำข้อมูลมาเก็บในคิว



คิวแบบง่าย (Simple queue)

- ต้องมีการกำหนดขนาดของคิว (queue size) ให้เพียงพอ กับข้อมูลที่จะนำเข้า
- เมื่อต้องการนำข้อมูลเข้าคิว (enqueue) จะเพิ่มข้อมูลลงในตำแหน่ง rear และเพิ่มค่า rear ขึ้นไป 1
- เมื่อต้องการนำข้อมูลออกจากคิว (dequeue) จะนำข้อมูลในตำแหน่ง front และเพิ่มค่า front ขึ้นไป 1

enqueue(10)



enqueue(20)

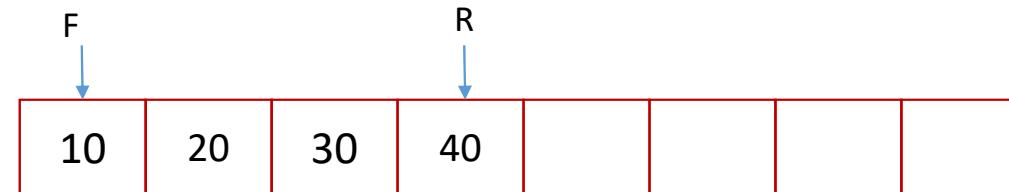


enqueue(30)

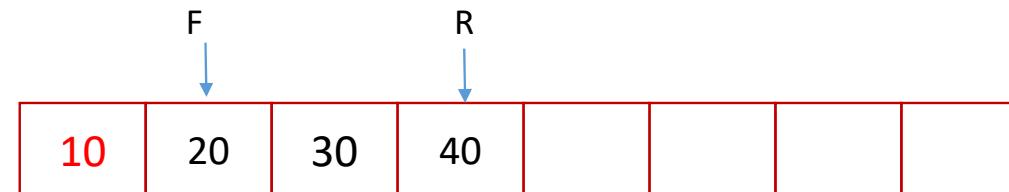


คิวแบบง่าย (Simple queue)

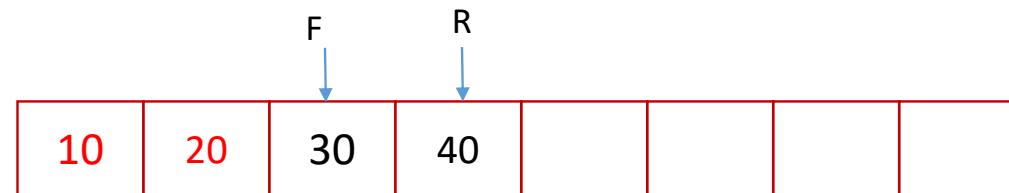
enqueue(40)



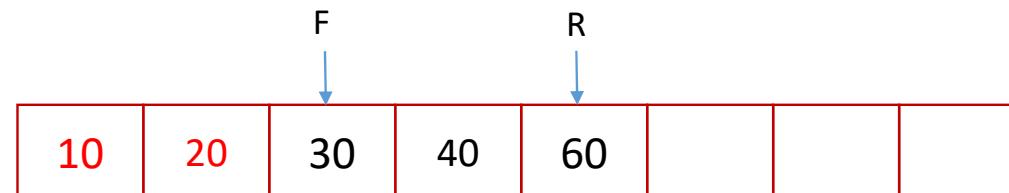
dequeue()



dequeue()



enqueue(60)



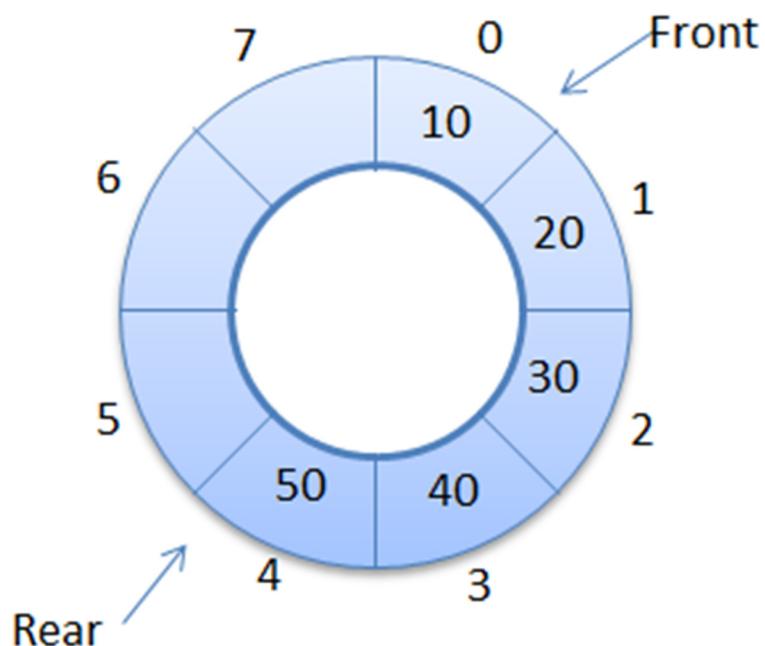


Simple Queue in C++ STL

```
queue<int> q;  
q.push(5);  
q.push(3);  
q.push(9);  
cout<<q.size()<<endl;  
cout<<q.front()<<endl;  
cout<<q.back()<<endl;  
q.pop();  
cout<<q.front()<<endl;  
cout<<q.back()<<endl;
```

คิววงกลม (Circular queue)

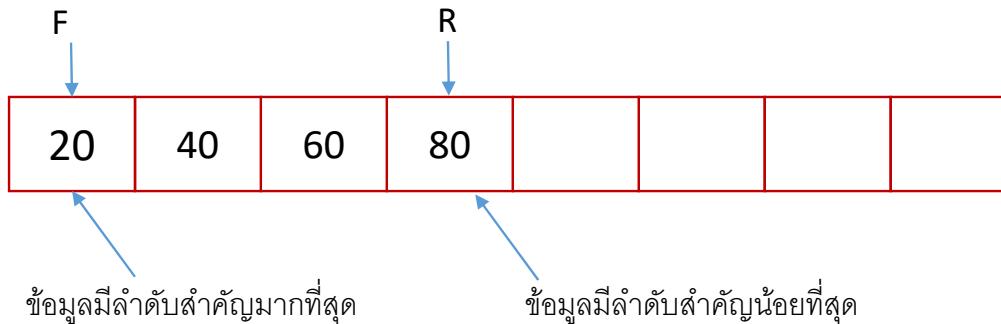
- คิววงกลมถูกออกแบบมาเพื่อแก้ไขข้อจำกัดของคิวแบบง่ายเมื่อ rear (R) ถูกเพิ่มค่าไปจนถึงตำแหน่งสุดท้ายของคิว ก็จะไม่สามารถนำเข้าข้อมูลต่อไปได้ถึงแม้ว่าคิวจะยังมีตำแหน่งว่างอยู่ก็ตาม



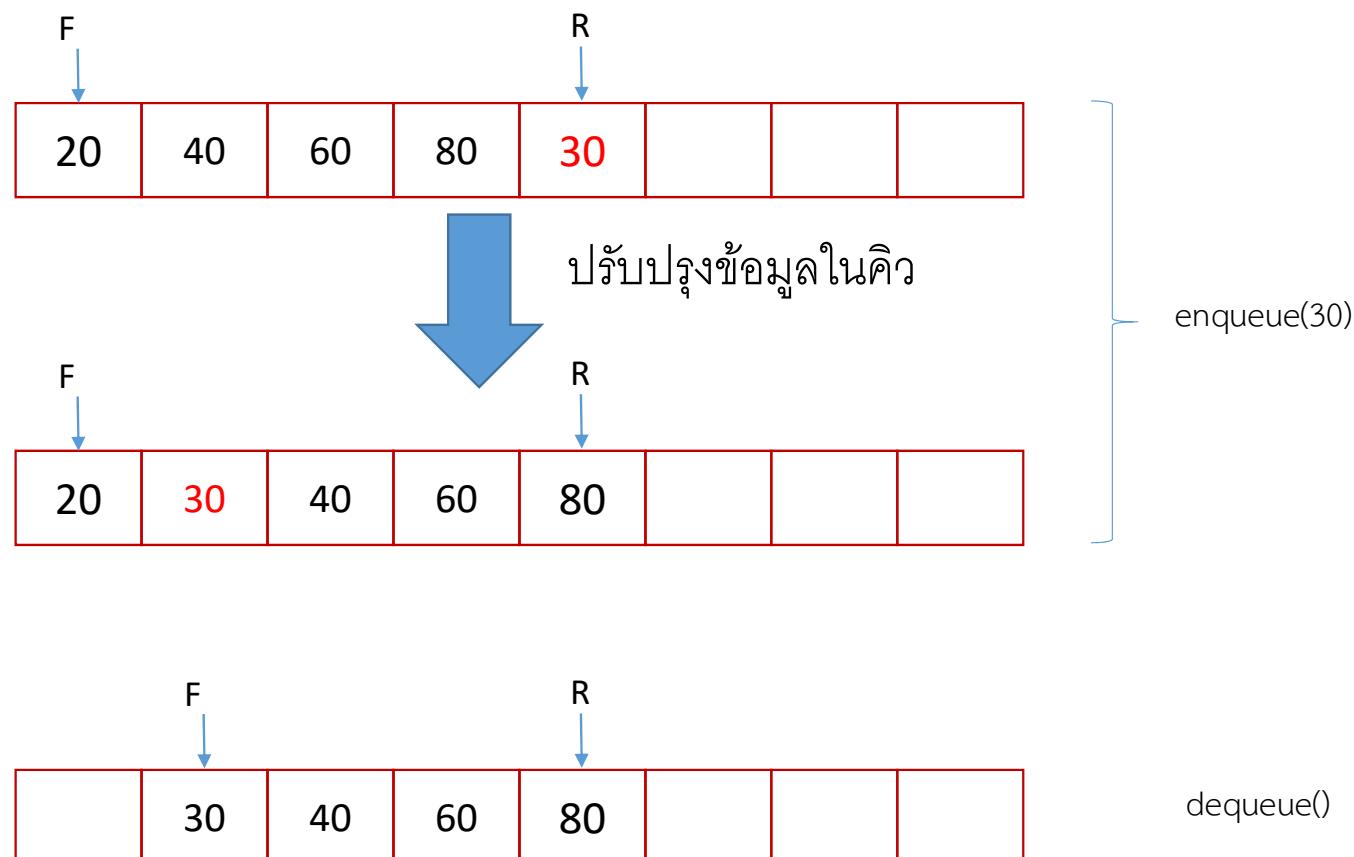
- คิวในตำแหน่งแรก (0) และตำแหน่งสุดท้าย (7) จะถูกยึดโยงเข้าด้วยกัน เพื่อที่ว่า Rear จะสามารถกลับมาที่ตำแหน่งแรกได้หากพบว่ามีว่างอยู่
- ข้อดีของ circular queue คือสามารถใช้พื้นที่ในคิวได้อย่างมีประสิทธิภาพ

คิวลำดับความสำคัญ (Priority queue)

- จัดเก็บข้อมูลตามลำดับความสำคัญ (Priority) จากมากไปน้อย หรือน้อยไปมากก็ได้
- ข้อมูลที่มีลำดับสำคัญที่สุดจะอยู่ที่หัวคิว (Front) และข้อมูลลำดับสำคัญน้อยที่สุดจะอยู่ท้ายคิว (Rear)
- ทุกครั้งที่มีการเพิ่มข้อมูลใหม่ จะต้องมีการปรับปรุงข้อมูลในคิว



คิวลำดับความสำคัญ (Priority queue)





Priority Queue in C++ STL

```
priority_queue<int> q;                                //มากไปหน้า雍
//priority_queue< int, vector<int>, greater<int> > q;    //น้อยไปมาก
q.push(5); q.push(3); q.push(9); q.push(2); q.push(10);
int n = q.size();
for(int i=0 ; i< n; i++ )
{
    cout<< q.top()<<" "; q.pop();
}
```



ค่ายโอลิมปิกวิชาการ วิชาคอมพิวเตอร์

Linked List

ผศ.ดร.เนียบวุฒิ รัตนวิไลสกุล chaibwoot.r@sci.kmutnb.ac.th

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ

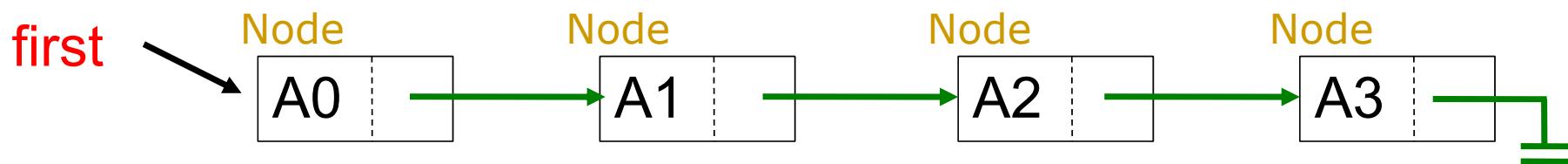


Outline

- Linked list nodes
- Linked list operations
 - Insertion
 - Append
 - Deletion
- Linked list representation & implementation
- Other types of linked lists
 - Sorted
 - Doubly-linked
 - Circular

Linked Lists

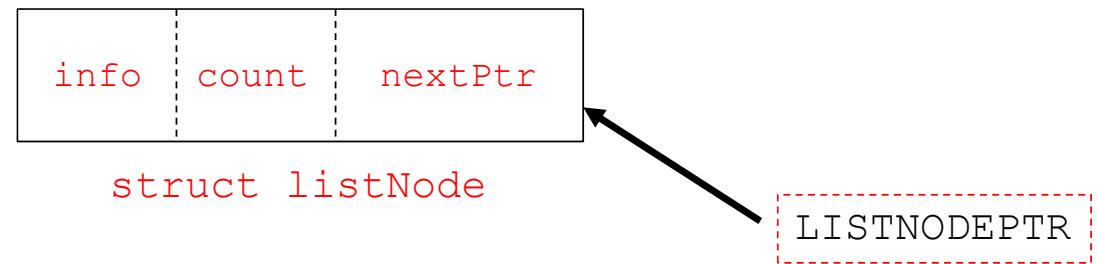
- รายการโยง ประกอบด้วยกลุ่มของโหนดซึ่งแต่ละโหนดประกอบด้วยเขตข้อมูลอย่างน้อยสองเขต(field)
 - ข้อมูลในโหนด (data)
 - ตัวชี้หรือรายการโยง บอกถึงที่อยู่ของโหนดถัดไปในรายการ (reference to the next Node in the list)
- การเพิ่ม(addition) หรือ การลบ(deletion) ข้อมูลที่อยู่ในรายการใช้เวลา $O(n)$.



STRUCT LINKED LISTS

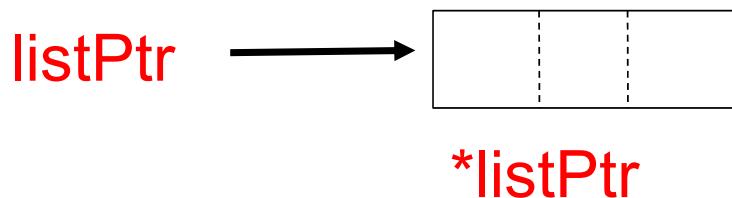
```
typedef struct listNode
{
    char info;
    int count;
    struct listNode *nextPtr;
} my_listNode;

main()
{
    my_listNode LISTNODE;
    my_listNode *LISTNODEPTR;
```



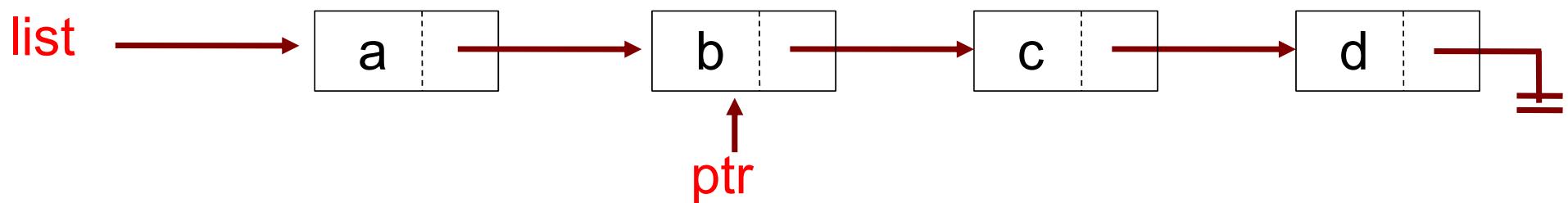
การนำตัวชี้มาใช้งาน

- การเขียนสัญลักษณ์ * ตามด้วยชื่อตัวชี้ จะแทน唬นดของข้อมูลซึ่งตัวชี้ซึ่งอยู่ เช่น จากการประกาศนิດตัวชี้ของ ListNode การเขียน nextPtr อ้างถึงที่อยู่ของ唬นดหนึ่ง ในลิงค์ลิสต์ ที่ตัวชี้ nextPtr ซึ่งไปถึง listPtr แตกต่างจาก *listPtr เพราะ listPtr คือตัวชี้ที่อ้างถึงเลขที่อยู่ แต่ *listPtr คือข้อมูลที่ถูกซึ้ง
- สามารถซื้อไปที่เขตต่าง ๆ ของฟิลด์ได้ โดย ชื่อตัวชี้->ชื่อฟิลด์



การจัดการกับโหนดชนิดตัวชี้

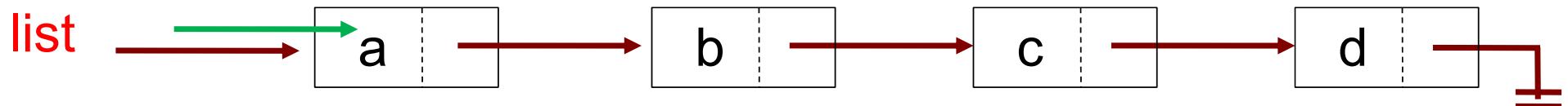
- กำหนดให้ `ptr` และ `ptr->nextPtr` เป็นตัวชี้ (Pointer) ซึ่งไปที่โครงสร้างข้อมูลที่เป็น `listNode`



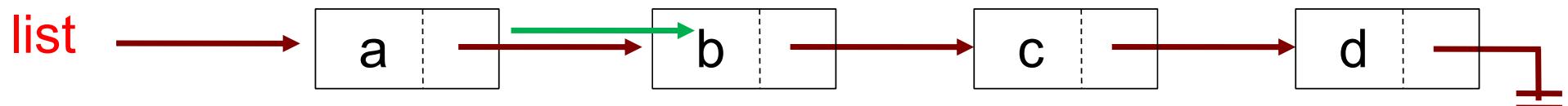
- `list` และ `ptr` เป็นตัวชี้โดย `list` ซึ่งไปที่โหนดแรก และ `ptr` ซึ่งไปที่โหนดที่สอง

การเรียกใช้ตัวชี้ที่ต่ำแห่งต่าง ๆ ในลิงค์สิสต์

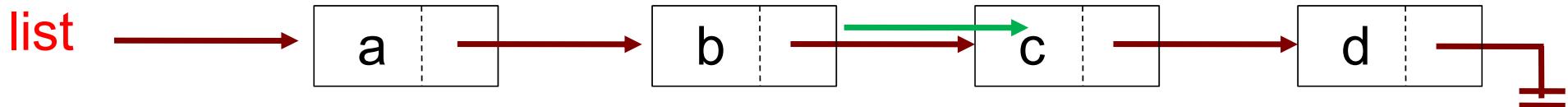
- `list` ชี้ไปที่หนดแรกของรายการ



- `list->next` ชี้ไปที่หนดที่ 2 ของรายการ



- `list->next->next` ชี้ไปที่หนดที่ 3 ของรายการ

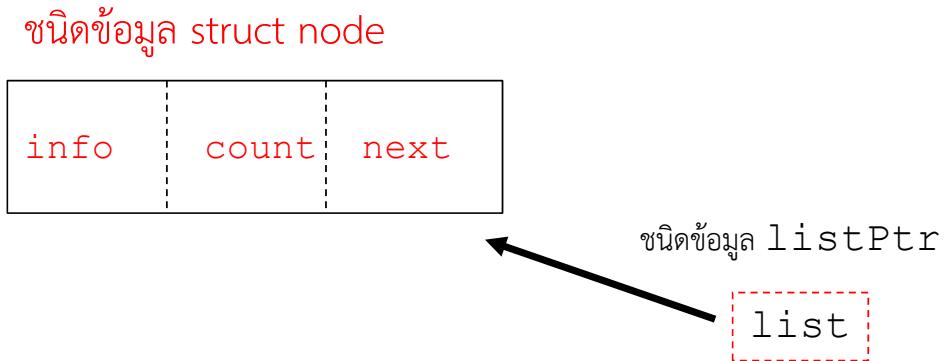


การทำงานของลิงค์ลิสต์

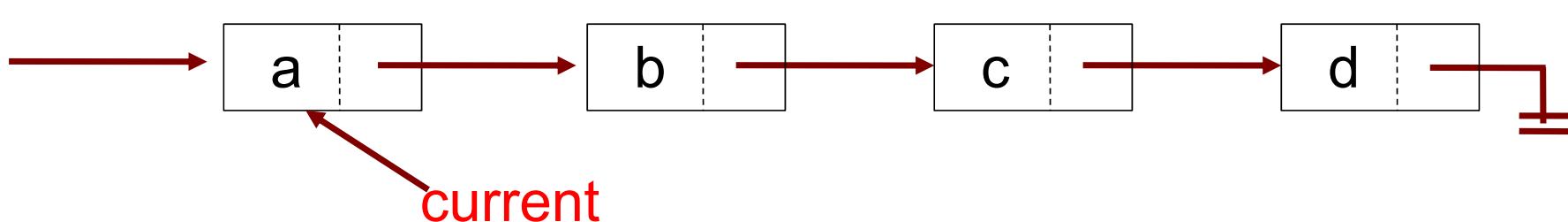
- ในการค้นหา อ่าน และการพิมพ์ข้อมูลจากลิงค์ลิสต์สามารถทำได้โดยการท่องไปในลิงค์ลิสต์ โดยเริ่มต้นจากกำหนด ptr ที่จุดเริ่มต้นของการด้วยคำสั่ง ptr=list
- ต่อมาสามารถเลื่อนตัวชี้ไปที่หนึ่งถัดไปโดย ptr=ptr->next

```
typedef struct node
{
    int info;
    int count;
    struct node *next;
} my_node;

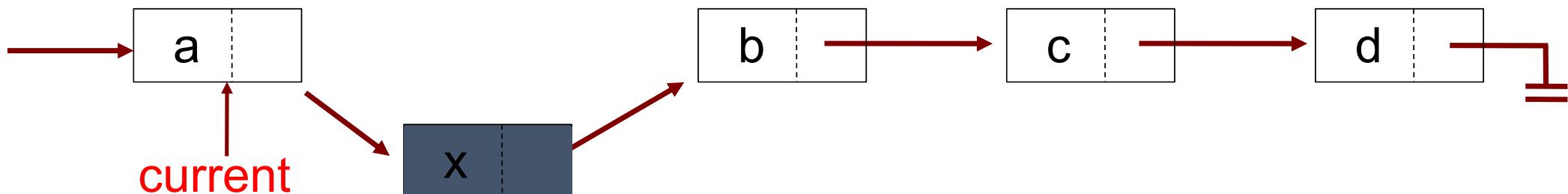
main()
{
    my_node *ptrNode;
```



การเพิ่มโนดในลิงค์ลิสต์

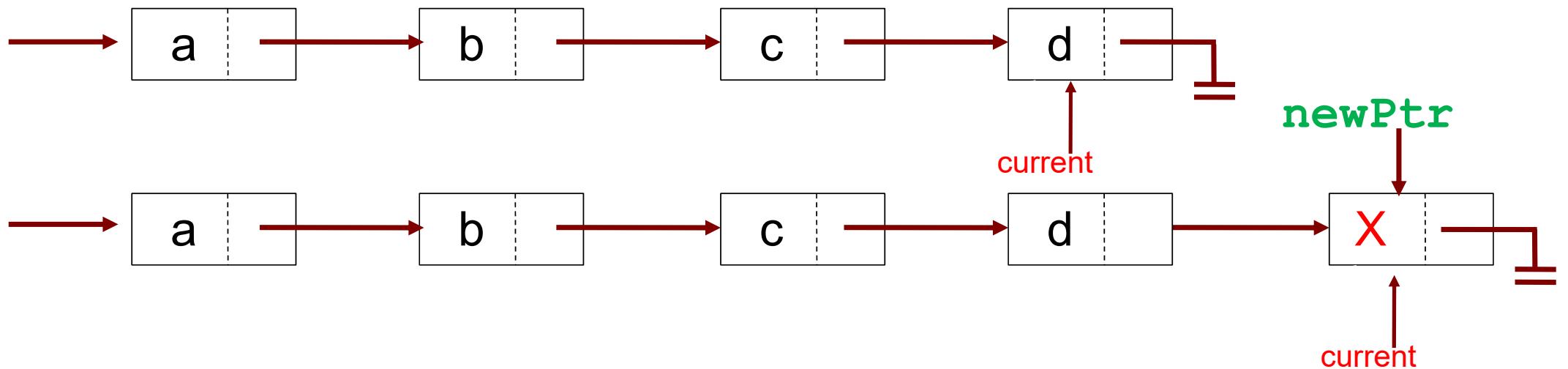


- เพิ่มโนด X หลังตำแหน่งที่ current ชี้



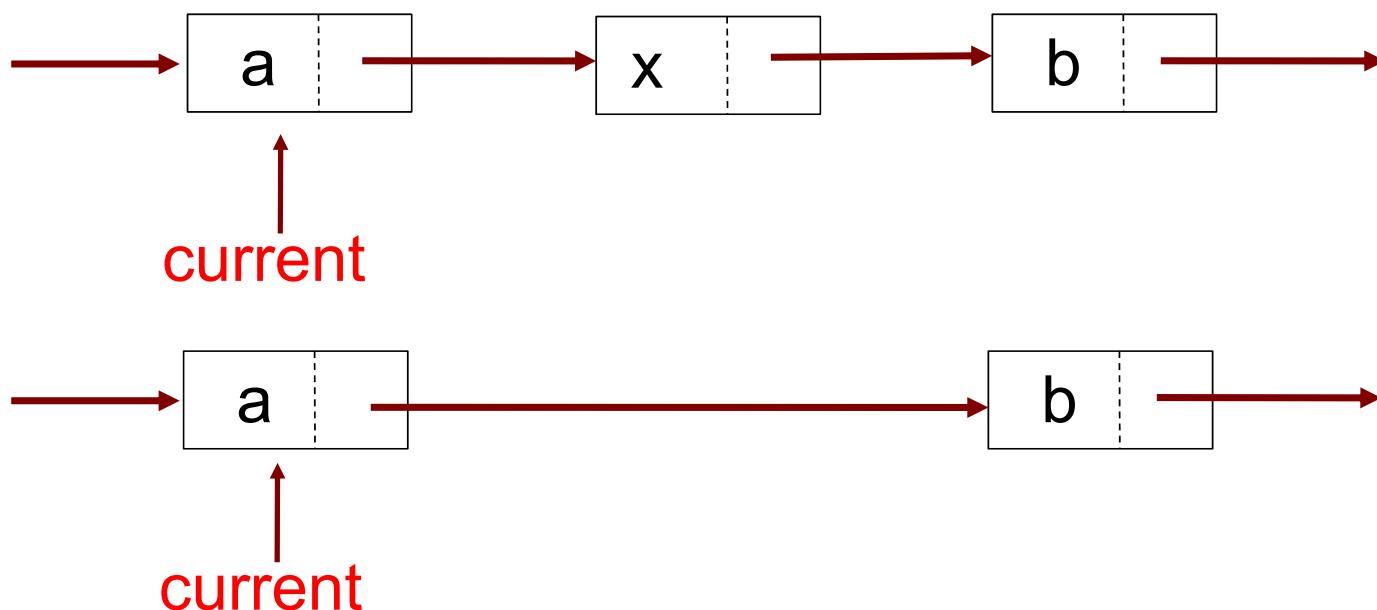
การเพิ่มโนนดในลิงค์ลิสต์ (Append)

- เพิ่มโนนด X ที่ตำแหน่งสุดท้ายของลิงค์ลิสต์



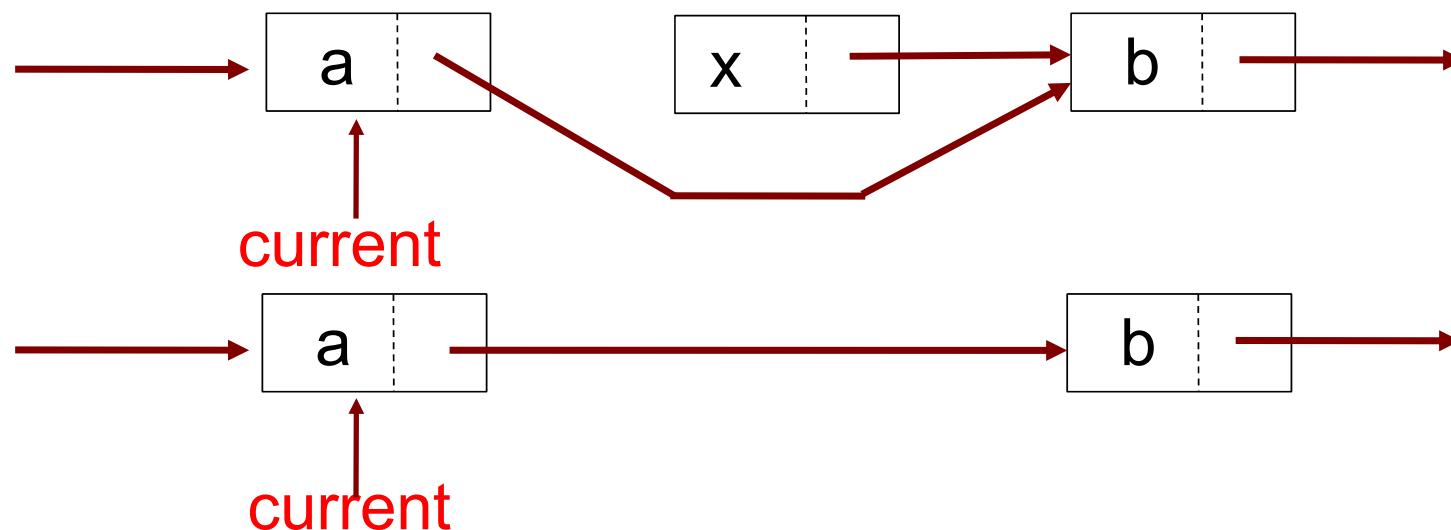
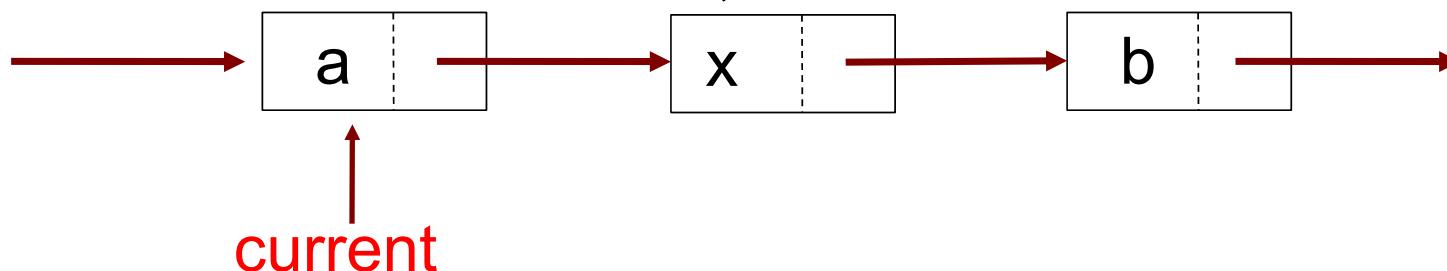
การลบโนนดในลิงค์ลิสต์

- ลบโนนด X หลังตำแหน่งที่ current ชี้



การลบโหนดในลิงค์ลิสต์

```
current.next = current.next.next;
```



การท่องไปในลิงค์ลิสต์

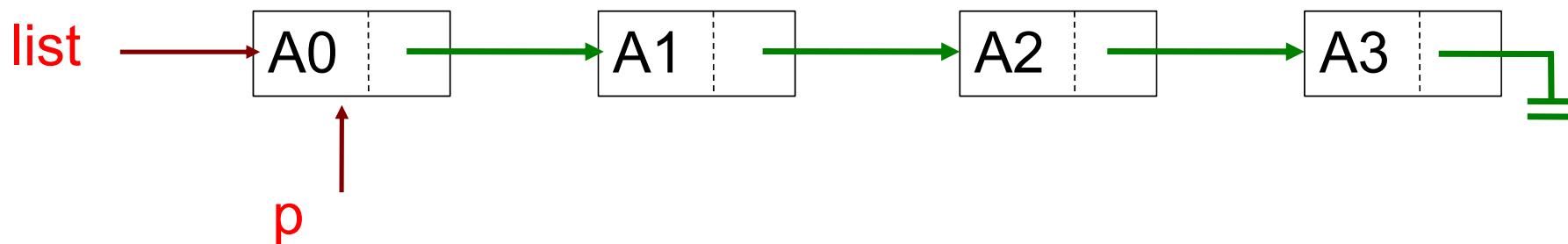
- ถ้าข้อมูลอยู่ในอาร์เรย์:

```
for (index = 0; index < length; index++)  
    printf (a[index]);
```

- ถ้าข้อมูลอยู่ในลิงค์ลิสต์

position p

```
for( p=list; p!=null; p=p.next)  
    printf (p->info);
```



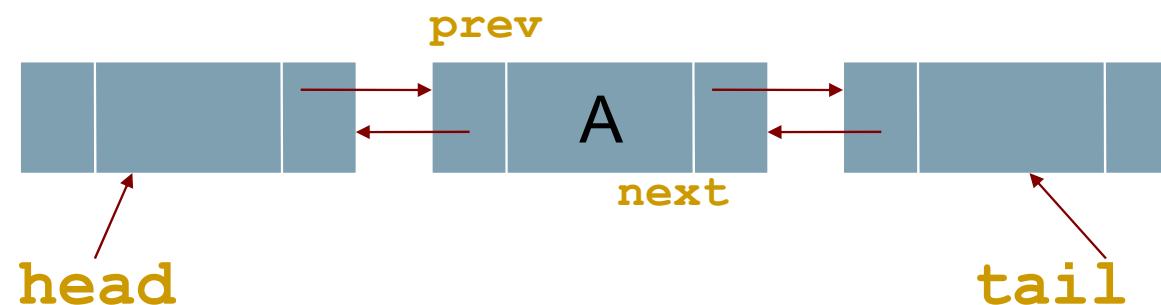


คุณสมบัติของ Linked List

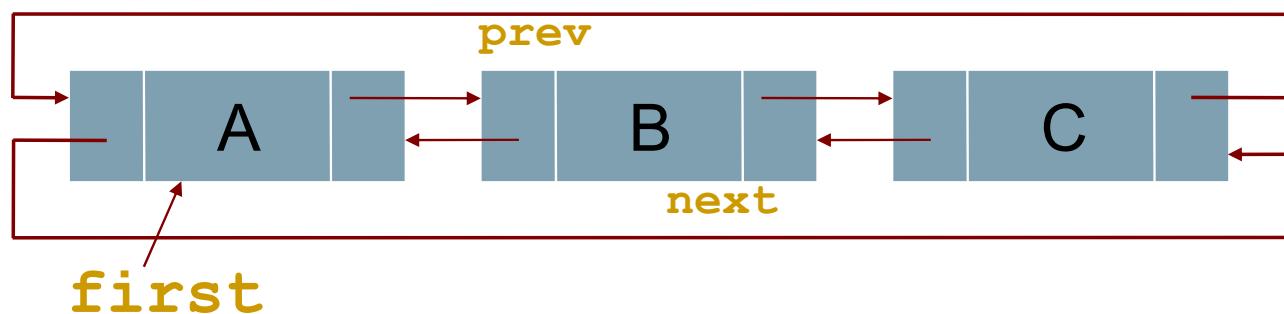
- การวิเคราะห์เวลา Running Time
 - การเพิ่มโหนดต่อจากโหนดก่อนหน้า หรือโหนดสุดท้ายใช้เวลา $O(1)$
 - การลบโหนดต่อจากโหนดก่อนหน้า หรือโหนดแรกใช้เวลา $O(1)$
 - การค้นหาข้อมูลใช้เวลา $O(n)$
 - การดึงข้อมูล ณ ตำแหน่งปัจจุบันใช้เวลา $O(1)$
- ประโยชน์
 - สามารถเพิ่มโหนดได้เรื่อยๆ
 - ง่ายต่อการอ่าน ลบ เพิ่ม ข้อมูลของโหนดแรกหรือโหนดสุดท้าย
- ข้อเสีย
 - เสียเวลาในการเรียก `malloc`
 - เสียพื้นที่สำหรับเก็บตัวชี้ที่จะซึ่งไปที่ตำแหน่งของโหนดต่อไป

ลิงค์ลิสต์ประเภทอื่น ๆ

- *Doubly-linked lists*: ทุก ๆ โหนดใน list จะเก็บ pointer ซึ่งไปที่โหนดก่อนหน้า และโหนดถัดไป.



- *Circular-linked lists*: โหนดสุดท้ายจะซึ่งไปที่โหนดแรกโดยจะใช้ Header ด้วยกันได้





Array versus Linked Lists

- Arrays are suitable for:
 - Inserting/deleting an element at the end.
 - Randomly accessing any element.
 - Searching the list for a particular value.
- Linked lists are suitable for:
 - Inserting an element.
 - Deleting an element.
 - Applications where sequential access is required.
 - In situations where the number of elements cannot be predicted beforehand.



ค่ายโอลิมปิกวิชาการ วิชาคอมพิวเตอร์

Library สำหรับการแข่งขัน

ปรับปรุงจาก เอกสารของ ผศ.ดร.อภิสิทธิ์ รัตนารานุรักษ์

ผศ.ดร.เนียบวุฒิ รัตนวิไลสกุล chaibwoot.r@sci.kmutnb.ac.th

ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าฯ พระนครเหนือ



STL – Standard Template Library

- STL มี 3 components
 - Containers - class templates สำหรับเก็บข้อมูล
 - Algorithms - function templates สำหรับจัดการ Containers
 - Iterators - smart pointers สามารถใช้งานกับ containers สร้างขึ้นเพื่อจัดการ STL Containers



STL – Container

- Sequence Container
 - Vector
 - Deque
 - List
- Adapter Containers
 - Stack
 - Queue
 - Priority queue
- Associative Container
 - Set, multiset
 - Map, multimap



STL – Standard Template Library

- STL มีการออกแบบที่มีประสิทธิภาพในเรื่อง Time complexity
- STL containers สามารถเพิ่มลดขนาดได้อัตโนมัติ
- STL มี built-in algorithms ในการจัดการ containers
- STL มี iterators ที่สามารถทำให้ containers และ Algorithms ทำงานได้อย่างยืดหยุ่นและมีประสิทธิภาพ
- STL สามารถสร้าง container และ Algorithms ได้
 - STL algorithms สามารถ process STL containers โดยใช้ user-defined containers ได้
 - User defined algorithms สามารถ process STL containers ได้



Containers

Data structures ที่สามารถเก็บข้อมูล

- ❑ list: doubly linked list
- ❑ vector: เมื่อเทียบกับ C array แต่ปรับค่าได้
- ❑ map: set ของคู่ key/value
- ❑ Set: set ของ keys



Algorithms

generic functions ที่ใช้ในงานทั่วไป ไม่ว่าจะเป็นการค้นหา การจัดเรียงข้อมูล การเปรียบเทียบ

- find
- merge
- reverse
- sort
- and more: count, random shuffle, remove, Nth-element, rotate.



Vector

- มีโครงสร้างเช่นเดียวกันกับ array
- Vector สามารถเพิ่มขนาดลดขนาดได้
- สามารถแทรกและลบออกได้
- สามารถใช้แทน array ได้



Defining Vector

Syntax: `vector<of what>`

เช่น:

`vector<int>` - vector of integers.

`vector<string>` - vector of strings.

`vector<int * >` - vector of pointers to integers.

`vector<Shape>` - vector of Shape objects. Shape เป็น class ที่สร้างขึ้นมาเอง.



สรุป VECTOR

```
vector <int> v;  
v.push_back(11); v.push_back(2); v.push_back(50); v.push_back(8);  
cout<<v[2] << " " << v.at(3) << " " << v.front() << " " << v.back() << " " << v.size() << " " << v.empty() << endl;  
for(int i = 0; i < v.size(); i++){cout << v[i] << " ";}cout << endl;  
  
v.erase( v.begin() + 1 );  
v.insert( v.begin() + 1 , 100);  
for(int i = 0; i < v.size(); i++){cout << v[i] << " ";}cout << endl;  
  
sort(v.begin(), v.end());  
for(int i = 0; i < v.size(); i++){cout << v[i] << " ";}cout << endl;  
  
reverse(v.begin(), v.end());  
for(int i = 0; i < v.size(); i++){cout << v[i] << " ";}cout << endl;  
  
vector<int>::iterator it = find( v.begin(), v.end(), 50 ); cout << *it << " "  
it = find( v.begin(), v.end(), 10 ); cout << *it << " "
```



list

- list เป็น sequence container จาก header list
- Class template list มีรูปแบบของ doubly linked list – ทุก Node มี pointer จาก previous node และ next node
- Member function
 - push_front, pop_front, ...
 - remove
 - unique
 - merge
 - reverse
 - sort
 - ...



ស្នូល LIST = LINK LIST

```
list<int> l; l.push_front(10); l.push_front(25); l.push_front(5); l.push_back(80); l.push_back(16);
for(list<int>::iterator it = l.begin() ; it != l.end() ; it++){ cout << *it << " "; }cout<<endl;
```

```
l.sort(); for(list<int>::iterator it = l.begin() ; it != l.end() ; it++){ cout << *it << " "; }cout<<endl;
l.reverse(); for(list<int>::iterator it = l.begin() ; it != l.end() ; it++){ cout << *it << " "; }cout<<endl;
```

```
list<int> l; l.push_front(1); l.push_front(2); l.push_front(3); l.push_back(4);
list<int> d; d.push_front(7); d.push_front(2); d.push_front(1); d.push_back(5);
l.merge(d);
for(list<int>::iterator it = l.begin() ; it != l.end() ; it++){ cout << *it << " "; }cout<<endl;
```



ស្នូល LIST = LINK LIST

```
list<int> l;  
l.push_back(1);    l.push_back(2);    l.push_back(3);    l.push_back(4);  
list<int>::iterator it = l.begin();  
advance(it, 2);  
l.insert(it, 5);  
for(list<int>::iterator it = l.begin() ; it != l.end() ; it++){ cout << *it << " "; }cout<<endl;  
  
list<int> l;  
l.push_back(1);    l.push_back(2);    l.push_back(3);    l.push_back(4);  
list<int>::iterator it = l.begin();  
advance(it, 2);  
l.erase(it);  
for(list<int>::iterator it = l.begin() ; it != l.end() ; it++){ cout << *it << " "; }cout<<endl;
```



deque

- Double ended queue – สามารถ Push/Pop ได้ทั้ง head และ tail
- #include<deque>
- Member function
 - push_front,
 - push_back,
 - pop_front,
 - pop_back



สรุป DEQUE

```
deque<int> e;      e.push_front(3);      e.push_front(4);      e.push_front(5);      e.push_back(6); e.push_back(7);  
for ( int i=0 ; i < e.size() ; i++){cout<<e[i]<<" ";      }          cout<<endl;  
  
cout<<e.front()<<endl;  
e.pop_front();  
  
cout<<e.back()<<endl;  
e.pop_back();  
  
for ( int i=0 ; i < e.size() ; i++){cout<<e[i]<<" ";      }          cout<<endl;
```



Next...

- Map
- pair
- copy algorithm



Class Employee

```
class Employee
{
public:
    Employee () {}

    Employee (const string& name) : _name(name) {}

    void set_salary (int salary) { _salary = salary; }

    int salary() const { return _salary; }

    void set_name (const string& name) { _name = name; }

    const string& name() const { return _name; }

    int _salary;

    string _name;
};
```



Locating an Employee

เก็บข้อมูล employee ใน vector

หากต้องการหาข้อมูลให้ หากคนใน vector

Bad solution - not efficient! ความเร็วเป็น n

เพราะจะໄล' check ที่ละตัว ตั้งแต่แรกถึงตัวสุดท้าย ความเร็วเป็น n



CODE Vector + Class

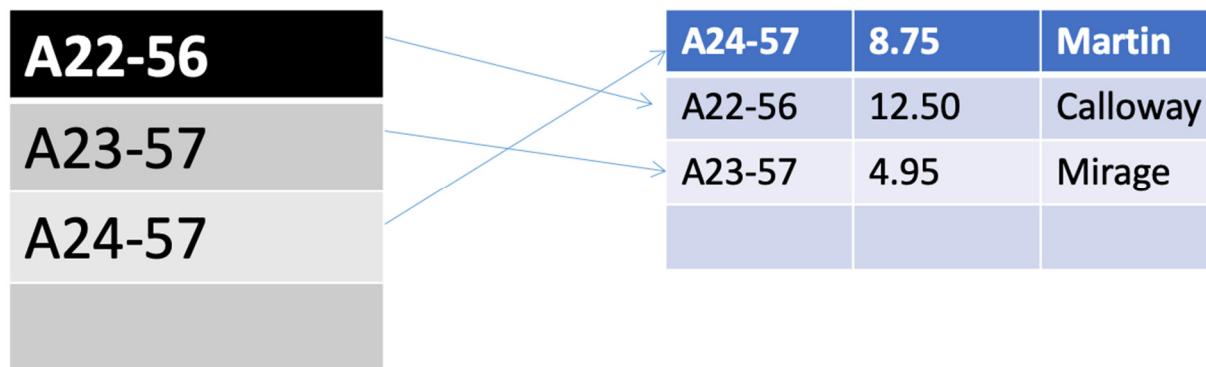
```
class Employee
{
    public:
    char s[50];
    int i;
    Employee (char x[], int y)
    {
        strcpy(s, x);
        i = y;
    }
    void show()
    {
        cout<<s<<" "<<i<<endl;
    }
};

int main()
{
vector<Employee> v;
v.push_back( Employee("a",1) );
v.push_back( Employee("b",2) );
v.push_back( Employee("c",3) );

v[0].show(); v[1].show(); v[2].show();    การค้นหาจะໄล่ເອາ ຕັ້ງແຕ່ຕົວແຮກຄືນຕົວສຸດທ້າຍ ຄໍາໃຊ້ search ແບບ vector ແຕ່ຄໍາໃຊ້ map ຈະເປັນກາຣະໂດດ
return 0;
}
```

Solution: Map - Associative Array

- ใช้ key/value pair มี index และ data เก็บข้อมูล
- `#include<map>`
- Insertion/find operation - $O(\log n)$





สรุป CODE MAP : SEARCH

```
map<int, string> student;
```

```
student[101] = "Jacqueline";  
student[2] = "Blake";
```

เข้าข้อมูลใส่
เข้าข้อมูลใส่

```
student.insert( make_pair(300, "Denise") );  
student.insert( make_pair(4, "Blake") );
```

เข้าข้อมูลใส่
เข้าข้อมูลใส่

```
student[5] = "Timothy";  
student[5] = "Aaron";
```

เข้าข้อมูลใส่
เข้าข้อมูลใส่

```
cout << student[2] << endl;  
cout << student[101] << endl;
```



สรุป CODE MAP : SEARCH

```
class Employee
{
public:
    char s[50];
    int i;
    Employee (char x[], int y) { strcpy(s, x); i = y; }
    void show(){cout<<s<<" "<<i<<endl;}
};

map<int, Employee*> s1; //must use pointer only
//map<int, Employee> s2; //no pointer error

s1.insert( make_pair(1, new Employee("a1",200) ) );
s1.insert( make_pair(2, new Employee("b1",300) ) );
s1.insert( make_pair(3, new Employee("c1",320) ) );

s2.insert( make_pair(1, Employee("a1",200)) );
s2.insert( make_pair(2, Employee("b1",300)) );
s2.insert( make_pair(3, Employee("c1",320)) );

s1[2]->show();                                //can run
s1[1]->show();                                //can run
//s2[1].show();                                //Error
```



ស្នូល CODE MAP : SEARCH

```
map<int, Employee*>::iterator it;
for( it = s1.begin(); it != s1.end(); ++it)
{
    cout << it->first << " " << it->second->s << " " << it->second->i << endl;
}

it = s1.find(2);                                //search key only
cout << it->first << " :" << it->second->s << " , " << it->second->i << endl;

it = s1.find(20);                               //search key no have = end() only
cout << it->first << " :" << it->second->s << " , " << it->second->i << endl;
```



สรุป CODE PAIR

```
vector< pair<int, Employee*> > v;
```

```
pair<int, Employee*> p;
```

```
p.first = 2; p.second = new Employee("a1",100);      v.push_back(p);
p.first = 3; p.second = new Employee("b1",200);      v.push_back(p);
p.first = 1; p.second = new Employee("c1",300);      v.push_back(p);
```

```
cout << v[0].first << " "; v[0].second->show();
cout << v[1].first << " "; v[1].second->show();
cout << v[2].first << " "; v[2].second->show();
```

```
sort(v.begin(), v.end() );
```

```
cout << v[0].first << " "; v[0].second->show();
cout << v[1].first << " "; v[1].second->show();
cout << v[2].first << " "; v[2].second->show();
```



Copy

copy(Iterator first, Iterator last, Iterator where);

Copy ตั้งแต่ 'first' ถึง 'last' ไปยัง 'where'.

```
int ia[] = { 0, 1, 1, 2, 3, 5, 5, 8 };
vector<int> ivec1(ia, ia + 8 ), ivec2;

copy(ivec1.begin(), ivec1.end(), back_inserter(ivec2));
```



Algorithms

- #include<algorithm>
- Sort
- Search
- Count



sort

```
int main()
{
    vector<int> x = {20,50,80,90,100,15};

    sort( x.begin(), x.end() );

    for(auto it=x.cbegin();it!=x.cend();it++)
    {
        cout<<*it<<" ";
    }
    return 0;
}
```



find

```
#include<iterator>
int main()
{
    vector<int> x = {20,50,80,90,100,15};
vector<int>::iterator it = find(x.begin(),x.end(),90);
    if(it!=x.end())
        cout<<"Found at "<<it-x.begin()<<" "<<endl;;
    else
        cout<<"Not found"<<endl;
    return 0;
}
```



find_if

```
bool gt10(int value)
{
    return value>10;
}

int main()
{
    vector<int> x = {20,50,80,90,100,15};
    vector<int>::iterator it;
it = find_if(x.begin(),x.end(),gt10);
    if(it!=x.end())
        cout<<"Found greater than 10 at "<<it-x.begin()<<" "<<endl;;
    else
        cout<<"Not found"<<endl;
    return 0;
}
```



all_of

```
bool gt10(int value){  
    return value>10;  
}  
  
int main() {  
    vector<int> x = {20,50,80,90,100,15};  
    bool it = all_of(x.begin(),x.end(),gt10);  
    if(it)  
        cout<<"All elements are greater than 10"<<endl;  
  
    return 0;  
}
```



any_of

```
bool gt10(int value){  
    return value>10;  
}  
  
int main() {  
    vector<int> x = {20,50,80,90,100,15};  
    bool it = any_of(x.begin(),x.end(),gt10);  
    if(it)  
        cout<<"Some elements are greater than 10"<<endl;  
  
    return 0;  
}
```



none_of

```
bool gt10(int value){  
    return value>10;  
}  
  
int main() {  
    vector<int> x = {20,50,80,90,100,15};  
    bool it = none_of(x.begin(),x.end(),gt10);  
    if(it)  
        cout<<"None elements are greater than 10"<<endl;  
  
    return 0;  
}
```



count

```
int main()
{
    vector<int> x = {20,50,80,90,100,15,100,100};
int res = count(x.begin(),x.end(),100);
    cout<<res<<endl;
    return 0;
}
```



count

```
bool gt10(int value)
{
    return value>10;
}

int main()
{
    vector<int> x = {20,50,80,90,100,15,100,100};
int res = count_if(x.begin(),x.end(),gt10);
    cout<<res<<endl;
    return 0;
}
```



min_element & max_element

```
int main()
{
    vector<int> x = {20,50,80,90,100,15,100,100};
    vector<int>::iterator it;
    it = min_element(x.begin(),x.end());      cout<<it-x.begin()<<endl;
    it = max_element(x.begin(),x.end());      cout<<it-x.begin()<<endl;
    return 0;
}
```



accumulate

```
#include<numeric> //*****
int main()
{
    vector<int> x = {1,2,3,4,5,6,7,8,9,10};
    cout<<accumulate(x.cbegin(),x.cend(),0);
    return 0;
}
```



for_each

```
void calsquare(int val)
{
    cout<<val*val<<" ";
}

int main()
{
    vector<int> x = {20,50,80,90,100,15,100,100};
    for_each(x.cbegin(),x.cend(),calsquare);
    return 0;
}
```



reverse

```
int main() {  
    vector<int> x = {1,3,5,6,7,9};  
  
    reverse(x.begin(),x.end());  
  
    for(auto it=x.cbegin();it!=x.cend();it++){  
        cout<<*it<<" ";  
    }  
    return 0;  
}
```



partition

```
bool func(int x)
{
    if(x<5)
        return true;
    else
        return false;
}

int main()
{
    vector<int> x = {3,4,5,6,7,3,4,5};
    vector<int>::iterator bound = partition(x.begin(),x.end(),func);
    for(auto it=x.cbegin();it!=bound;it++)
        cout<<*it<<" ";
    cout<<endl;
    for(auto it=bound;it!=x.cend();it++)
        cout<<*it<<" ";
    return 0;
}
```



MAKE_HEAP & SORT_HEAP

```
int main()
{
    vector<int> x = {1,3,5,7,9};
    vector<int>::iterator it;
    make_heap(x.begin(),x.end());
    for(it=x.begin();it!=x.end();it++)
        cout<<*it<<" ";
    cout<<endl;
    sort_heap(x.begin(),x.end());
    for(it=x.begin();it!=x.end();it++)
        cout<<*it<<" ";
    cout<<endl;
}
```



สรุป CODE

//COPY

```
vector <int> v;
v.push_back(11); v.push_back(2); v.push_back(50); v.push_back(8);
for(int i=0 ; i < v.size() ; i++) { cout<<v[i]<<" "; } cout<<endl;

vector <int> v2;
copy(v.begin(), v.end(), back_inserter(v2));
for(int i=0 ; i < v.size() ; i++) { cout<<v2[i]<<" "; } cout<<endl;
```

//Sort

```
sort( v.begin(), v.end() );
for(int i=0 ; i < v.size() ; i++) { cout<<v[i]<<" "; } cout<<endl;
```

//Find

```
vector<int>::iterator it = find(v.begin(),v.end(),11);
cout<< it-v.begin() <<endl;

it = find(v.begin(),v.end(),100);
cout<< it-v.begin() <<endl;
```



สรุป CODE

```
//count
v.push_back(8);
v.push_back(8);
for(int i=0 ; i < v.size() ; i++) { cout<<v[i]<<" "; } cout<<endl;
cout << count(v.begin(),v.end(),8) <<endl;

//min max
for(int i=0 ; i < v2.size() ; i++) { cout<<v2[i]<<" "; } cout<<endl;
it = min_element(v2.begin(),v2.end());      cout<<it-v2.begin()<<endl;
it = max_element(v2.begin(),v2.end());      cout<<it-v2.begin()<<endl;

//reverse
sort( v2.begin(), v2.end() ); reverse(v2.begin(),v2.end());
for(int i=0 ; i < v2.size() ; i++) { cout<<v2[i]<<" "; } cout<<endl;
```



สรุป CODE

```
//heap
vector <int> v3;
v3.push_back(11); v3.push_back(2); v3.push_back(50); v3.push_back(8); v3.push_back(4); v3.push_back(17); v3.push_back(9);
make_heap(v3.begin(),v3.end()); cout<<v3[0]<<" "; v3.erase( v3.begin() ); for(int i=0 ; i < v3.size() ; i++) { cout<<v3[i]<<" "; } cout<<endl;
make_heap(v3.begin(),v3.end()); cout<<v3[0]<<" "; v3.erase( v3.begin() ); for(int i=0 ; i < v3.size() ; i++) { cout<<v3[i]<<" "; } cout<<endl;

vector <int> v4;
v4.push_back(12); v4.push_back(2); v4.push_back(11); v4.push_back(8); v4.push_back(4); v4.push_back(17); v4.push_back(9);
sort_heap(v4.begin(),v4.end()); for(int i=0 ; i < v4.size() ; i++) { cout<<v4[i]<<" "; } cout<<endl;

//merge
vector <int> v5;
v5.push_back(1); v5.push_back(2); v5.push_back(3); v5.push_back(4); v5.push_back(5); v5.push_back(6); v5.push_back(7);
vector <int> v6;
v6.push_back(12); v6.push_back(2); v6.push_back(11); v6.push_back(8); v6.push_back(4); v6.push_back(17); v6.push_back(9);
vector<int> z;
merge( v5.begin(),v5.end(), v6.begin(),v6.end(), back_inserter(z) );
for(int i=0 ; i < z.size() ; i++) { cout<<z[i]<<" "; } cout<<endl;
```



สรุป

- Container แบ่งออกเป็น Sequence, Adapter และ Associative Containers (แนะนำให้ใช้ Sequence และ Associative)
- STL Library มี algorithm ให้เลือกใช้ ให้เหมาะสมกับโปรแกรมที่จะเขียน
- หากสามารถเรียก library ได้ ให้เลือกใช้ library ก่อนเพื่อประหยัดเวลาในการเขียนโปรแกรม
- มีอีกหลายฟังก์ชันของ STL ที่สามารถใช้งานได้ที่ไม่ได้แนะนำในวันนี้



เครื่องที่สอบในค่าย 3

- ทาง สจล. ในฐานะเจ้าภาพ TOI ปีนี้ ขออนุญาตแจ้ง Environment ที่ใช้ในการแข่ง TOI 21
 - ❑ OS : Ubuntu 22.04 LTS
 - Compiler : GCC 11.5
 - Flag :
 - [C++17 / g++) /usr/bin/g++-11 -DEVAL -std=c++17 -O2 -pipe -static -s -o outputFile inputFile.cpp
 - [C11 / gcc] /usr/bin/gcc-11 -DEVAL -std=c11 -O2 -pipe -static -s -o outputFile inputFile.c -lm

- ❑ Text Editors and IDEs :
 - Sublime Text 4192
 - Visual Studio Code 1.98 (with C/C++ extension v1.24)
 - **ไม่มี Code::Blocks**