1. Introduction

Desktop Applications Programming

Why Programming?

- มีความสุขและสนุกที่ได้ทำอะไรสักอย่าง และสิ่งที่ทำนั้น มีประโยชน์ต่อผู้อื่นด้วย
- ได้นำเอาสิ่งต่างๆ มารวมกัน (ข้อมูล+วิธีการ) แล้วทำให้คอมพิวเตอร์ทำงานเหมือนสิ่งมีชีวิต
- การได้เรียนรู้สิ่งใหม่อยู่ตลอดเวลา ได้ทำงานกับเรื่องราวใหม่ๆ อยู่ตลอดเวลา
- การได้ทำงานเป็นทีม
- ได้ใช้ประโยชน์จาก Internet ในการสร้างสังคมแห่งการเรียนรู้
- ได้ช่วยคนที่ยังเขียนโปรแกรมไม่เก่ง

เราจะสร้าง Program ได้อย่างไร

- นำปัญหาในชีวิตจริง มาสร้างเป็นแบบจำลอง (ทางคณิตศาสตร์)
- นำ Algorithm ทางคณิตศาสตร์ มาแก้ปัญหากับแบบจำลองนั้น
- เขียนเป็นโปรแกรมลงในคอมพิวเตอร์
- สร้าง User interfacing เพื่อติดต่อกับผู้ใช้โปรแกรม
- ทดสอบ / แก้ไข
- เผยแพร่โปรแกรมออกใช้งาน

Why Windows?

- Multi Tasking
  - non-preamptive
  - preemptive
- Event-Driven
  - ตอบสนองรวดเร็วทันใจ
  - ทำงานก็ต่อเมื่อมีงานให้ทำเท่านั้น (ประหยัดพลังงาน)
- นอกจากระบบปฏิบัติการ Windows แล้วยังมี
  - Linux
  - Mac
  - ฯลฯ

Event-Driven Programming.

- สนับสนุน GUI (Graphics User Interface)
- จะทำงานตอบสนองต่อเหตุการณ์ที่เกิดขึ้น
  - ผู้ใช้ คลิกเมาส์
  - ผู้ใช้ กดแป้นพิมพ์
  - มีข้อมูลเข้ามาทางพอร์ตต่างๆ
  - เกิดเหตุการณ์พิเศษ (exception)

GUI Operating System

- ไม่ต้องจดจำคำสั่งยาวๆ ยากๆ
- OS จะแสดงภาพของ controls บนหน้าจอ
  - ปุ่มกด ไอคอน สกรอลบาร์ ฯลฯ
- ผู้ใช้ควบคุมผ่าน keyboard หรือ pointing Devices
- หน้าต่างของ Application ต่างๆ สามารถวางซ้อนทับกันได้
  - สามารถดึงหน้าต่างล่าง ขึ้นมาไว้ข้างบนได้
  - ทำงานเหมือนกับการทำงานด้วยกระดาษบนโต๊ะทำงาน
- ภาพแสดงสิ่งต่างๆ ได้มากกว่าคำบรรยาย
- สร้างเอกสารแบบ WYSISYG

Writing Windows Programs

- WIN32 API Programming
  - ใช้ภาษา C ผ่าน Applicaion Programming Interface
- MFC Programming
  - ใช้ภาษา C++ บน MFC Framework
- Microsoft .NET framework
  - ใช้ทุกภาษาที่ .NET รองรับ (ในวิชานี้ใช้ C#)

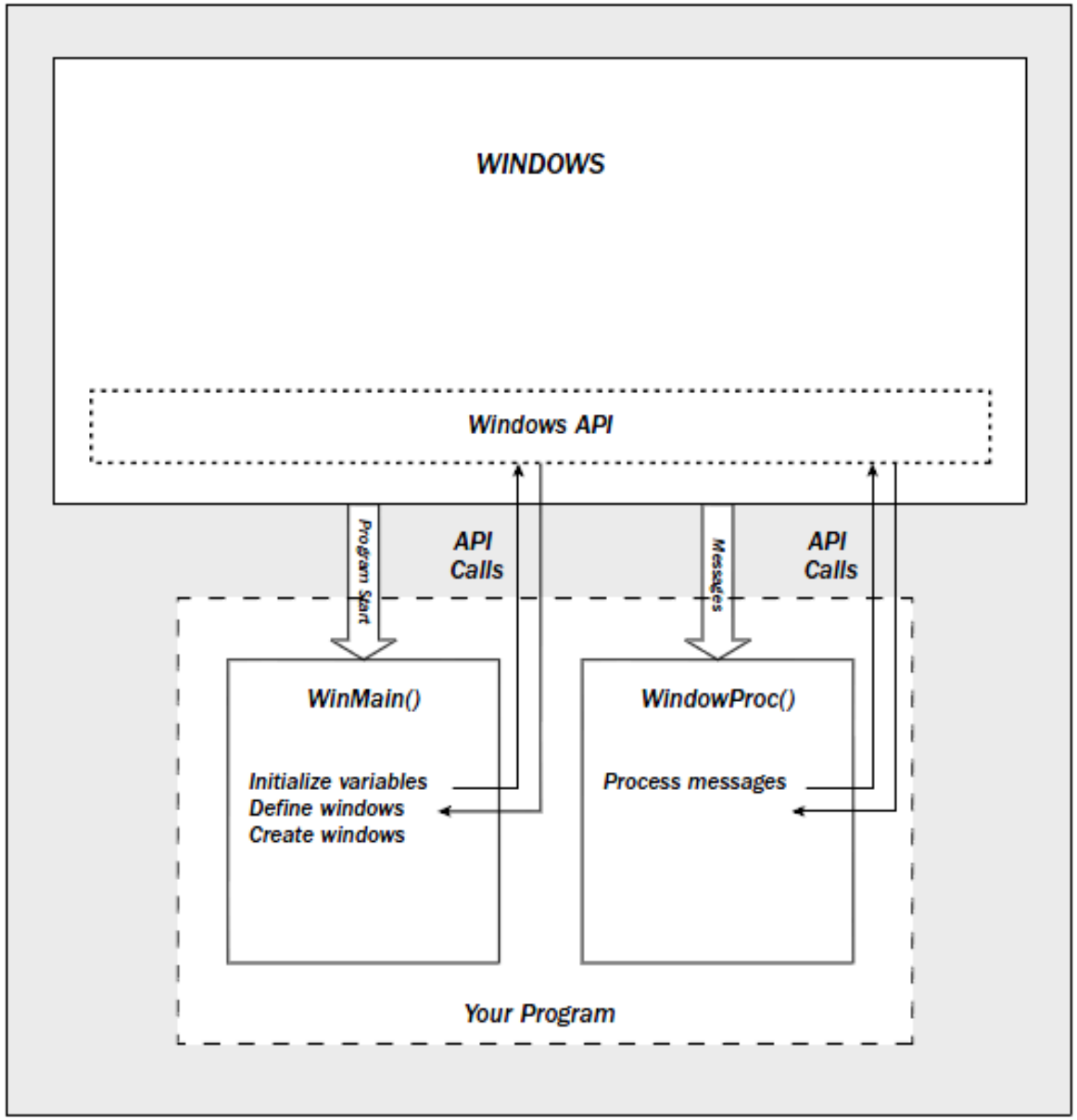
การพัฒนาโปรแกรมบนวินโดวส์ใน LAB นี้

- WIN32 API Programing
  - ใช้ภาษา C
- MFC Programming
  - ใช้ภาษา C++
- Microsoft .NET framework
  - ใช้ภาษา C#

ส่วนประกอบสำคัญ ในการพัฒนา Windows Apps

- สิ่งที่ทำให้ Windows Application ทำงานได้ ประกอบด้วย 3 ส่วนหลักๆ ได้แก่
  - Applications
  - Windows
  - Messages and Events

The Structure of a Windows Program



Applications

- Application คือ code ที่ทำงาน (execute) ได้
- Application สามารถเก็บในไฟล์ที่แยกจากกันได้
  - .exe (executable file)
  - .dll (dynamic Link Library)
  - Activex
  - Device Drivers

Application จะไม่เรียกใช้ Device driver โดยตรง แต่ใช้งานผ่าน operating system

Windows

- Windows ในที่นี้ไม่ได้หมายถึงระบบปฏิบัติการ
- เป็นกรอบหน้าต่างสี่เหลี่ยมที่ระบบปฏิบัติการสร้างขึ้น เพื่อ
  - แสดง controls เพื่อรวมรวมข้อมูลจากผู้ใช้
  - นำเสนอข้อมูลต่อผู้ใช้
- Windows จะเป็นส่วนหนึ่งของ Application
  - Application สามารถมีได้หลาย windows
- Operating system จะสื่อสารกับ windows ของ application ผ่าน message ของระบบ
- เมื่อ windows ถูกสร้างขึ้นมา จะได้รับหมายเลขประจำตัว เรียกว่า window handle

Messages and Events

- Message เป็น object เล็กๆ ในระบบวินโดสว์ ถูกส่งไปมาระหว่าง process ต่างๆ ใน message ประกอบด้วย
  - Time stamp (ใช้เป็นการภายใน OS เท่านั้น)
  - messages identifier
  - wParam และ lParam (เป็นข้อมูลที่ส่งมากับ message)
  - Window handle ของหน้าต่างผู้รับ message

message identifier

- เป็นค่าคงที่ ที่รู้จักโดยทุก application ในระบบปฏิบัติการ Windows
- *WM_LBUTTONDOWN.* เกิดขึ้นเมื่อมีการกดปุ่มซ้ายของเมาส์
  WM_KEYDOWN. เกิดขึ้นเมื่อ มีการกดปุ่มใดๆ บนคีย์บอร์ด
- *WM_CHAR.* เกิดต่อจาก WM_KEYDOWN ประกอบด้วย ASCII code ของอักษรที่กด
  - การกด function keys และ arrow keys จะเกิด WM_KEYDOWN แต่ไม่มีWM_CHAR.
- *WM_PAINT.* ถูกสร้างโดย operating system เมื่อต้องการ “refreshed” ส่วนต่างๆ ของหน้าต่าง เมื่อต้องการ resized หรือเมื่อ window ที่อยู่ด้านบนถูกลบออกไป
- และอีกหลายๆ Message

The application message queue

- เมื่อ application ถูกสร้างขึ้น มันจะสร้าง message queue ขึ้นมาเพื่อรองรับ การส่ง message มาจากระบบปฏิบัติการ
- เมื่อมีเหตุการณ์ต่างๆ เกิดขึ้น ระบบปฏิบัติการ Windows จะใส่ message ลงใน application message queue
- ระบบปฏิบัติการ Windows จะรู้จักที่อยู่ของแต่ละ Application message queue เนื่องจากตอนสร้าง application จะต้องมีการ register กับระบบ

The main message loop

- หลังจาก Windows ใส่ message ลงใน message queue ของ application แล้ว application จะต้องไปดึง message เหล่านั้นมาพิจารณา ทุก message ตามลำดับที่มีถึง
- ทุก Application จะต้องเริ่มที่ Winmain เสมอ
- ใน Winmain จะมีส่วนของโปรแกรมที่ทำหน้าที่จัดการกับ message ที่เข้ามา

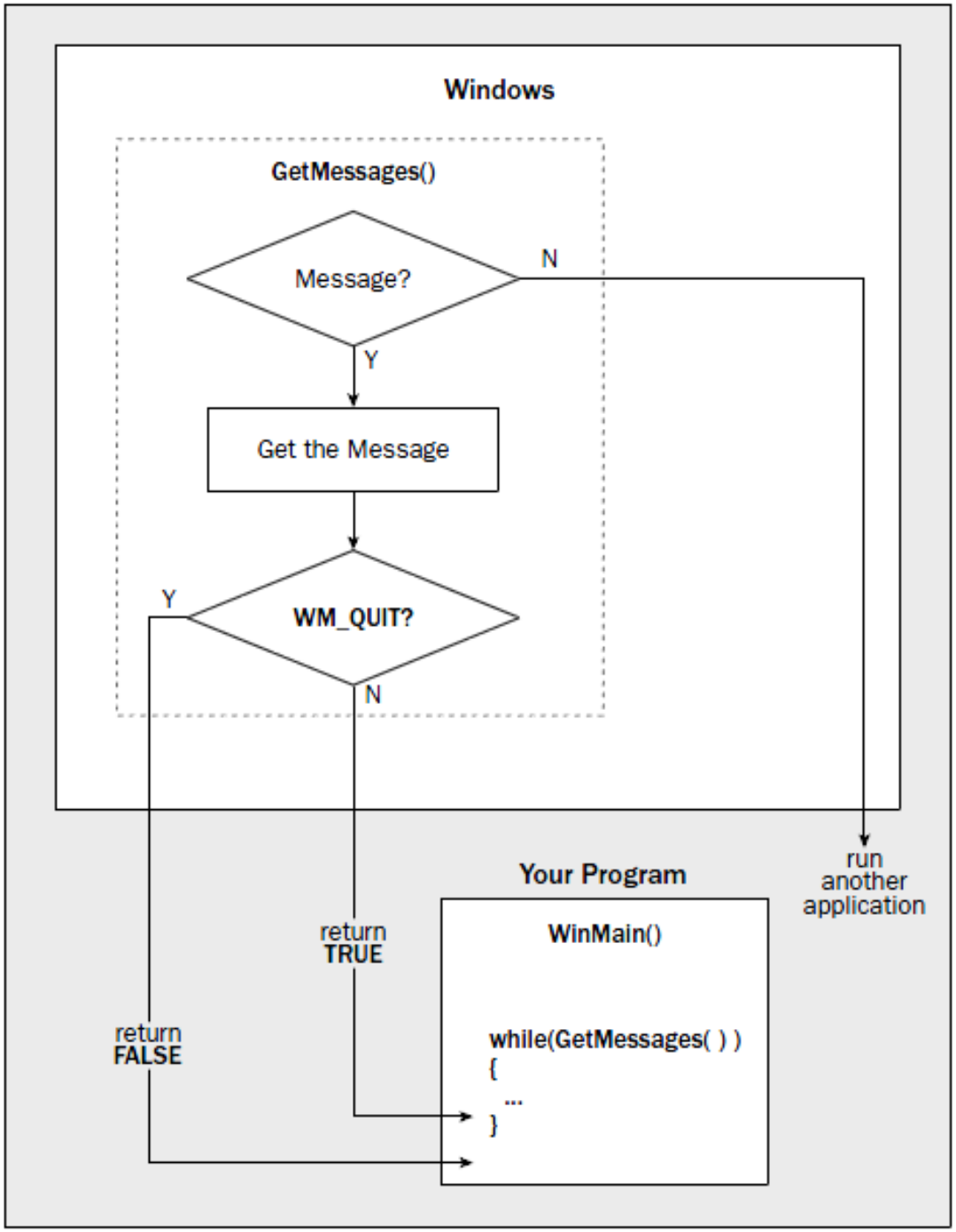
```
while (GetMessage(&msg))
    DispatchMessage(&msg)
```

  - GetMessage ดึง message ออกจาก Application message queue
  - DispatchMessage ส่ง message ไปยัง window ปลายทาง

MyWindowProc

- ในฟังก์ชัน MyWindowProc(hwnd, message_identifier, wParam, lParam) จะประกอบด้วย code ที่ทำหน้าที่จ่ายงานให้กับ window ตาม message ที่รับเข้ามาจาก OS

```
switch(message_identifier)
    { case WM_LBUTTONDOWN:
            /*  code to respond to this message */
            break;
      case WM_CHAR:
            /* code to respond to this message */
            break;
      case WM_PAINT:
            /* code to respond to this message */
            break;
       ...
    }
```


Windows
GetMessages()
Message?
N
Y
Get the Message
WM_QUIT?
Y
N
run another application
Your Program
WinMain()
return TRUE
return FALSE
while(GetMessages( ))
{
...
}

What about MFC and .NET?

- ในทุกๆ application จะมีส่วนของการทำ message loop ในทำนองคล้ายๆ กัน
- MFC และ .NET จะเขียน code เหล่านี้ไว้ให้แล้ว และเตรียม macro ไว้สร้างส่วนติดต่อกับ code เหล่านั้น
  - ใน MFC เก็บไว้ใน mfc.dll
  - ใน .NET เก็บไว้ใน dll ของ .NET
- แต่ Programmer ต้องเขียนส่วนของ event handler เอง

Question?