การพัฒนาไทม์แมชชีนด้วยภาษาไพธอน

DEVELOPMENT OF TIME MACHINE USING PYTHON

ภูมิ บัณฑิตย์จิรกุล

สุขุม บุตรดำ

สุพรรญา เพื่องฟู

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2553

ปริญญานิพนธ์ปีการศึกษา 2553

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาไทม์แมชชีนด้วยภาษาไพธอน

DEVELOPMENT OF TIME MACHINE USING PYTHON

ผู้จัดทำ

1. นายภูมิ	บัณฑิตย์จิรกุล	รหัสนักศึกษา 50011188
2. นายสุขุม	บุตรคำ	รหัสนักศึกษา 50011695
3. นางสาวสุพรรญา	เฟื่องฟู	รหัสนักศึกษา 50011734

_____ อาจารย์ที่ปรึกษา

(ผศ.ดร.วิศิษฏ์ หิรัญกิตติ)

การพัฒนาไทม์แมชชีนด้วยภาษาไพธอน

นายภูมิ	บัณฑิตย์จิรกุล	50011188
นายสุขุม	บุตรคำ	50011695
นางสาวสุพรรณา	เฟื่องฟู	50011734
ผู้ช่วยศาสตราจารย์ ดร.วิศิษฏ์	หิรัญกิตติ	อาจารย์ที่ปรึกษา

ปีการศึกษา 2553

บทคัดย่อ

ทุกๆกิจกรรมของมนุษย์ย่อมหลีกหนีไม่พ้นการติดต่อสื่อสาร ถ้าเรามีเครื่องมือที่สามารถบริหารการสื่อสารและสามารถทำงานตามกำหนดเวลาก็จะเป็นประโยชน์อย่างมาก ภาษา CL (ย่อมาจาก Communication Language) ซึ่งถูกออกแบบมาเพื่อรองรับเหตุผลดังกล่าว ภาษา CL คือภาษาคอมพิวเตอร์ที่สนับสนุนการทำงานตามเวลาและทำงานตอบสนองต่อเหตุการณ์ (Event) ซึ่งงานวิจัยที่ผ่านมาของ คุณสุพัณณดา โชติพันธ์ [2548] และคุณสุรชัย ล้อเจริญ [2550] ได้ทำการออกแบบส่วนภาษารวมทั้งโครงสร้างข้อมูลสำหรับจัดเก็บตารางเวลาการทำงานของคำสั่ง CL ในรูปของ Tree

แต่งานวิจัยทั้งสองยังขาดความสมบูรณ์ในส่วนของอินเตอร์พรีตเตอร์ที่ยังไม่รองรับการอินเตอร์พรีตฟังก์ชั่น คลาส และการทำงานร่วมกันของหลายๆโมดูล ดังนั้นโครงงานนี้จึงได้ปรับปรุงอินเตอร์พรีตเตอร์ภาษา CL เดิมในองค์ประกอบภายใน โดยการเพิ่มกลไกที่เรียกว่าเนมสเปซ (Namespace) เข้าไปในกระบวนการแปลความหมาย ชื่อตัวแปร ฟังก์ชั่น คลาส และ อ็อบเจ็ค ซึ่งจะช่วยทำให้อินเตอร์พรีตเตอร์ CL สามารถอ้างถึงอ็อบเจ็คที่ชื่อนั้นแทนอยู่ได้อย่างมีประสิทธิภาพ ทำให้อินเตอร์พรีตเตอร์สามารถดำเนินการนิยามฟังก์ชั่น นิยามคลาส และจัดการโมดูลได้

DEVELOPMENT OF TIME MACHINE USING PYTHON

Mr. Poom	Bunditjirakul	50011188
Mr. Sukhum	Butrkam	50011695
Miss. Supanya	Fuangfoo	50011734
Asst.Prof.Dr. Visit	Hirankitti	Advisor

Academic Year 2010

ABSTRACT

In every human activity it is unavoidable not to make use of some kind of communication. It would be very useful if we could have a software tool to help manage communication as well as execution of a work schedule. A CL (standing for Communication Language) language was designed for that purpose. CL is a computer language which can be used to program a computer to run instructions according to time and events. Two previous research works done by Choattiphan [2005] and Locharoen [2007] were a design of a CL language syntax, and a design of a tree-like internal data structure for storing a work schedule represented by CL statements, respectively.

There are some issues of the CL language that the two research works have not addressed, i.e. Pythonic namespaces that should be adopted in CL and object-oriented programming in CL. Therefore, in this thesis we have developed these two into CL. As a result, our improved CL interpreter can now apply different leveled namespaces to every interpretation of a CL statement, and especially these namespaces support the interpreter to handle user defined functions and classes as well as CL modules.

กิตติกรรมประกาศ

ปริญญานิพนธ์นี้สำเร็จลุล่วงไปได้ด้วยดี เพราะได้รับความร่วมมือ การดูแลเอาใจใส่และการสนับสนุนจากหลายๆฝ่ายช่วยกันผลักดันให้ปริญญานิพนธ์ครั้งนี้ดำเนินไปด้วยดี และประสบความสำเร็จลุล่วงในที่สุด

ขอขอบคุณสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง สาขาวิชาวิศวกรรมคอมพิวเตอร์ และสถานศึกษาในอดีตที่ผ่านมา สำหรับโอกาสดีๆทางการศึกษาที่เป็นแหล่งประสิทธิ์ประสาทวิชาทำให้ผู้จัดทำได้มีโอกาสในทุกวันนี้

ขอขอบพระคุณ ผศ. ดร.วิศิษฎ์ หิรัญกิตติ อาจารย์ที่ปรึกษา ในความความกรุณาอย่างดียิ่งที่ให้คำแนะนำต่างๆ ชี้แนะข้อบกพร่องต่างและแนวทางในการศึกษา ตั้งคำถาม เอาใจใส่ อธิบายข้อข้องใจต่างๆและช่วยเหลือเสมอมา

ขอขอบพระคุณ คณาจารย์ประจำสาขาวิชาวิศวกรรมคอมพิวเตอร์ทุกท่านที่กรุณาประสิทธิ์ประสาทวิชาความรู้และประสบการณ์อันมีค่ายิ่ง

ขอขอบคุณเพื่อน พี่น้อง นักศึกษาสาขาวิชาวิศวกรรมคอมพิวเตอร์ ทุกคนที่คอยเป็นกำลังใจและให้ความช่วยเหลือต่างๆตลอดมา

และสุดท้ายขอขอบพระคุณบุคคลที่สำคัญที่สุดในชีวิตซึ่งก็คือ บิดา มารดา และผู้มีพระคุณ อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูข้าพเจ้ามาเป็นอย่างดี พร้อมให้โอกาสในการศึกษาอย่างเต็มที่และยังให้กำลังใจเสมอมา ข้าพเจ้าระลึกในพระคุณอันสุดประมาณและขอกราบขอบพระคุณมา ณ ที่นี้

ภูมิ บัณฑิตย์จิรกุล
สุขุม บุตรคำ
สุพรรญา เฟื่องฟู

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ในปัจจุบันนี้ทุกๆกิจกรรมของมนุษย์ ไม่ว่าจะเป็นกิจวัตรประจำวันหรือการทำงาน ย่อมหลีกหนีไม่พ้นกิจกรรมของการติดต่อสื่อสารและคมนาคม ถ้าเราสามารถโปรแกรมให้คอมพิวเตอร์ ให้ทำกิจกรรมต่างๆตามกำหนดเวลาและเหตุการณ์นั้น จะเป็นประโยชน์อย่างมากเพราะจะเป็นการลดภาระของมนุษย์ในการควบคุมคอมพิวเตอร์ ให้เป็นหน้าที่ของเครื่องคอมพิวเตอร์เอง เพื่อที่มนุษย์จะได้มีเวลามากขึ้นในการทำงานที่ต้องอาศัยความคิดสร้างสรรค์ เมื่อพิจารณาดูภาษาคอมพิวเตอร์ที่ใช้ในปัจจุบัน เช่น ภาษา Java, C, C++ เป็นต้น จะพบว่าภาษาส่วนมากเหมาะสมสำหรับการสั่งงานที่เป็นชุดคำสั่งเท่านั้น ไม่เหมาะสำหรับการสั่งงานและการทำงานที่เกี่ยวข้องกับการสื่อสารซึ่งเป็นกิจกรรมที่มนุษย์ทำอยู่เป็นประจำ ข้อจำกัดดังกล่าวจะนำไปสู่ความไม่ยืดหยุ่นของการเขียนและการทำงานของโปรแกรม ซึ่งนับเป็นปัญหาที่น่าสนใจและควรแก่การแก้ไขปัญหาหากสามารถปรับปรุงรูปแบบของภาษาให้มีความยืดหยุ่นและสามารถทำงานได้คล้ายคลึงกับการทำงานโดยทั่วไปของคนมากขึ้นได้ ย่อมทำให้ผู้เขียนสามารถเขียนโปรแกรมและสื่อความหมายของโปรแกรมได้ง่าย และครอบคลุมมากยิ่งขึ้น

เพื่อแก้ไขปัญหาดังกล่าวข้างต้น โครงงานนี้จึงขอนำเสนอการออกแบบและพัฒนาภาษาคอมพิวเตอร์รูปแบบใหม่ ที่เรียกว่า "ภาษาเพื่อการสื่อสาร" (Communication Language) CL ซึ่งเป็นภาษาคอมพิวเตอร์ที่สนับสนุนการทำงานตามเวลา อีกทั้งยังสามารถโปรแกรมให้ทำงาน ตอบสนองต่อเหตุการณ์ (Event) แต่ยังมีความสามารถอย่างที่ปรากฏในภาษาคอมพิวเตอร์ทั่วไป ถ้าไม่มีการกำหนดเงื่อนไขของเวลาและเหตุการณ์ไว้ในโปรแกรม โครงงานนี้ได้ทำการปรับปรุงและพัฒนาอินเตอร์พรีตเตอร์ภาษา CL แล้วนำมาประยุกต์ใช้พัฒนาเครื่องมือบริหารการสื่อสารและการทำงานตามกำหนดเวลา โดยอาศัยความรู้ด้านต่างๆ ได้แก่ การสร้างอินเตอร์พรีตเตอร์สำหรับภาษาสคริปการเขียนโปรแกรม รวมทั้งได้พัฒนาอินเตอร์พรีตเตอร์เพื่อประมวลผลคำสั่งดังกล่าว

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

จุดมุ่งหมายของโครงงานนี้คือการพัฒนาอินเตอร์พรีตเตอร์ของภาษา CL ให้มีความสมบูรณ์มากยิ่งขึ้นโดยเพิ่มกลไกที่เรียกว่า "เนมสเปซ" (Namespace) เข้าไปในกระบวนการแปลความหมายเพื่อทำการเรียกใช้ ฟังก์ชั่น คลาส อ็อบเจ็ค และการทำงานร่วมกันของหลายๆโมดูลได้ ซึ่งถือได้ว่าโครงงานนี้เป็นการเพิ่มขีดความสามารถให้กับภาษา CL

1.3 สมมติฐานของการศึกษา

1) อาศัยการศึกษาธรรมชาติของการสื่อสารและการทำงานที่เกี่ยวข้องกับเวลาในชีวิตประจำวันแล้วพัฒนาเป็นแนวความคิดและหลักการ จากนั้นจึงนำเอาหลักการนี้ไปใช้พัฒนาเป็นภาษาคอมพิวเตอร์
2) ภาษาที่ใช้พัฒนาเป็นภาษา Python โดยภาษา CL ที่พัฒนาขึ้นจะเป็นส่วนของ Meta-Program บนภาษา Python อีกชั้นหนึ่ง แต่เราสามารถทำความเข้าใจกับ CL เสมือนภาษาคอมพิวเตอร์ภาษาหนึ่งที่ไม่ต้องพึ่งพาภาษา Python ได้
3) อินเตอร์พรีตเตอร์ CL จะต้องมี โครงสร้างข้อมูลภายในสำหรับเก็บลำดับการทำงานของโปรแกรม CL ซึ่งต้องสามารถเก็บได้ทั้งการทำงานบนช่วงเวลาและเหตุการณ์บนจุดเวลา รวมทั้งสามารถแทนเส้นแกนเวลาได้ทั้งหมด
4) อินเตอร์พรีตเตอร์ CL จะต้องมี คุณสมบัติต่างๆของ Class ให้ตรงตามหลักการของ Object Oriented Programming

1.4 ทฤษฎีหรือแนวคิดที่ใช้ในงานวิจัย

1.4.1 แนวคิดเกี่ยวกับเวลา

คนเรามักจดจำกำหนดการนัดหมายต่างๆ ในรูปของจุดเวลา ปี เดือน วัน ชั่วโมง นาที วินาที ลดหลั่นกันไปตามลำดับ และเป็นหน่วยเวลาสัมพันธ์กับการเปลี่ยนแปลงของดวงดาว เหตุที่เราใช้รูปแบบเวลาเช่นนี้เนื่องจากการทำงานของเรามีความสัมพันธ์ทั้งทางตรงและทางอ้อมกับการเปลี่ยนแปลงของดวงดาว ดังนั้นในการออกแบบโครงสร้างข้อมูลสำหรับเก็บลำดับการทำงานของโปรแกรม CL เราจึงเลือกเก็บจุดเวลาที่อยู่ในรูปแบบดังกล่าว

ในบางครั้งเราสังเกตว่า เรามักมีกำหนดการการทำงานที่เป็นค่าเป็นตัวแปร หรือการทำงานที่มีกำหนดเวลาเปลี่ยนแปลงเป็นคาบเวลา ซึ่งกำหนดการของการทำงานเหล่านั้นควรถูกจัดเก็บลงไปในโครงสร้างข้อมูลได้

ในโครงงานนี้ได้พัฒนาภาษา CL ที่มีความสามารถทำงานตามเวลา และเสนอความคิดเกี่ยวกับเวลาว่า เวลาเกิดจากการดำเนินไปของกระบวนการ (Process) อันหนึ่งอย่างต่อเนื่อง เมื่อไรก็ตามที่เกิดกระบวนการ ก็จะมีเวลาเกิดขึ้น ดังนั้น เราสามารถใช้เวลาเพื่อแสดงจุดกำเนิดของกระบวนการการทำงานได้

บทความที่เกี่ยวข้องกับเวลาของนักวิจัยหลายท่าน ได้อธิบายถึงรูปแบบของเวลา ความสัมพันธ์ของเวลา และการประมวลผลเวลาเชิงคณิตศาสตร์ รวมไปถึงความสัมพันธ์ระหว่างเวลาและการทำงาน อีกทั้งการวิจัยถึงความสัมพันธ์ของเวลาและเหตุการณ์อีกด้วย ซึ่งในโครงงานนี้

ได้นำแนวคิดดังกล่าวมาใช้และอธิบายเวลาให้อยู่ในรูปแบบของภาษาคอมพิวเตอร์และได้เพิ่มเติมรูปแบบและความสัมพันธ์ของเวลาตามความเหมาะสม

1.5 ขอบเขตของการวิจัย

1) สร้างกลไกที่เรียกว่า เนมสเปซ (Namespace) เข้าไปในกระบวนการแปลความหมาย ชื่อตัวแปร ฟังก์ชั่น คลาส และ อ็อบเจ็ค ซึ่งจะช่วยทำให้อินเตอร์พรีตเตอร์ CL สามารถอ้างถึงอ็อบเจ็คที่ชื่อนั้นแทนอยู่ได้อย่างมีประสิทธิภาพ
2) ผู้เขียนโปรแกรมสามารถกำหนดเวลาการทำงานให้กับโปรแกรมและสามารถโปรแกรมให้มีการทำงานตอบสนองต่อเหตุการณ์ที่เกิดขึ้นได้ ด้วยภาษา CL
3) สร้างโปรแกรมการจัดการส่วนบุคคล Personal organizer อย่างง่ายขึ้น เพื่อทำการแจ้งเตือนตามกำหนดเวลา โดยกำหนดให้มีการแจ้งเตือนแบบ Pop-up , SMS และ Email
4) สร้าง CL Interpreter ให้สามารถ Import module จาก File ต่างๆ และ Run Code ภาษา CL จาก File ได้
5) สร้าง CL Interpreter ให้สามารถรองรับคุณสมบัติต่างๆของ Class ให้ตรงตามหลักการของ Object Oriented Programming

1.6 ประโยชน์ที่เกิดขึ้นจากงานวิจัย

1) ได้ภาษาคอมพิวเตอร์รูปแบบใหม่ที่มีความสามารถประมวลผลการทำงานตามเวลาที่กำหนด
2) ได้อินเตอร์พรีตเตอร์ที่มีความสมบูรณ์มากขึ้น โดยได้เพิ่มความสามารถทางด้านการประมวลผลชุดคำสั่งให้ครอบคลุมมากกว่าเดิม อีกทั้งยังเพิ่มความสามารถของอินเตอร์พรีตเตอร์ให้รองรับการนิยามของฟังก์ชันและคลาสได้
3) ได้โครงสร้างข้อมูลสำหรับเก็บเนมสเปซ ซึ่งเนมสเปซประกอบไปด้วย ตัวแปร ค่าของตัวแปร นิยามฟังก์ชัน นิยามคลาส นอกจากนี้ยังได้โครงสร้างข้อมูลสำหรับเก็บชุดคำสั่งที่ทำงานตามกำหนดเวลา
4) ภาษา Python ได้รับการเพิ่มขีดความสามารถในการสร้างเอเจนต์สื่อสารและประมวลผลคำสั่งการทำงานตามเวลา

1.7 ขั้นตอนของการศึกษา

การทำงานแบ่งออกเป็น 3 ส่วน ดังนี้

1) กำหนดปัญหา

1.1) ศึกษาหลักการทำงานของอินเตอร์พรีตเตอร์และคอมไพล์เลอร์

1.2) ค้นหาและศึกษาโปรแกรมที่ช่วยในการสร้างอินเตอร์พรีตเตอร์

1.3) การเขียนโปรแกรมภาษา Python ที่นำมาใช้ในการสร้างอินเตอร์พรีตเตอร์ และเป็นต้นแบบในการพัฒนาภาษา CL

1.4) ศึกษาภาษา CL และโครงสร้างอินเตอร์พรีตเตอร์ CL จากงานวิจัยเดิมที่มีมาก่อน

1.5) ใช้วิธีคิดเชิงวิจารณ์ เพื่อหาข้อจำกัดใน CL เดิม

2) ตั้งสมมติฐาน

2.1) ศึกษาคุณลักษณะของเวลา จุดเวลา ช่วงเวลา และระยะเวลา

2.2) สำรวจโครงสร้างข้อมูลที่มีอยู่ในปัจจุบันว่ามีโครงสร้างข้อมูลใดที่เหมาะสมในการเก็บลำดับการทำงานของโปรแกรม CL

2.3) ออกแบบโครงสร้างข้อมูลสำหรับเก็บเนมสเปซ ขึ้นมาใหม่

2.4) ออกแบบโครงสร้างการจัดการฟังก์ชั่น คลาส และ โมดูล

3) ทดสอบสมมติฐาน

3.1) ทดสอบความถูกต้องของโครงสร้างข้อมูล

3.2) ทดสอบการประมวลผล Statement แต่ละแบบ

3.3) ทดสอบการประมวลผลชุดคำสั่งที่ทำงานตามกำหนดเวลา

3.4) ทดสอบการประมวลผลชุดคำสั่งที่เรียกใช้คลาส และ ฟังก์ชั่น

บทที่ 2

ภาษา Python

ในบทนี้จะกล่าวถึงภาษาที่ใช้สร้างภาษา CL ได้แก่ ภาษา Python โดยการโปรแกรมภาษา CL จะกระทำบนภาษา Python อีกที หรือเรียกว่า ภาษา CL เป็น Meta-Program บนภาษา Python เหตุที่เลือกใช้ภาษา Python นั้นเพราะภาษา Python เป็นภาษาที่มีการทำงานแบบอินเตอร์พรีตเตอร์ (Interpreter) เหมาะสำหรับแนวคิดที่ต้องมีการตอบสนองต่อเหตุการณ์ภายนอกของภาษา CL อีกทั้งภาษา Python ยังมีไวยากรณ์ที่เข้าใจได้ง่าย ทำให้การพัฒนาภาษา CL เป็นไปได้สะดวกยิ่งขึ้น

2.1 ชนิดข้อมูลในภาษา Python

ชนิดของข้อมูลในภาษา Python จะอยู่ในอ๊อบเจ็คที่ชื่อว่า Python Object การกำหนดชนิดของข้อมูลให้กับตัวแปรในภาษา Python จะถูกกระทำโดยอัตโนมัติ ผู้เขียนโปรแกรมสามารถเปลี่ยนแปลงตัวแปรหนึ่งให้เก็บค่าที่ต่างไปได้โดยไม่ต้องสนใจชนิดของตัวแปรที่เก็บอยู่เดิม ชนิดของข้อมูลทั่วไปได้แก่

1) ตัวเลข (Numeric) ข้อมูลชนิดตัวเลข ได้แก่
   1.1) เลขฐานสิบ (Interger) คือเลขที่ประกอบด้วยเลข 0-9 ไม่ว่าจะเป็นจำนวนบวก หรือ
   1.2) จำนวนลบ แต่ต้องไม่มีจุดทศนิยม ตัวอย่างเช่น 1, -3, 8784 ฯลฯ
   1.3) เลขฐานแปด คือ ตัวเลขที่ขึ้นต้นด้วย 0 เช่น 0715 ฯลฯ
   1.4) เลขฐานสิบหก คือ ตัวเลขที่ขึ้นต้นด้วย 0x เช่น 0x9FAC, 0x12A ฯลฯ
   1.5) เลขทศนิยม (Float number) เช่น 3.2, -45e12 ฯลฯ
   1.6) เลขจำนวนเต็มขนาดยาว (Long integer) คือตัวเลขที่ลงท้ายด้วย L หรือ l เช่น 4890549579L, 5l ฯลฯ
   1.7) เลขจำนวนเชิงซ้อน (Complex number) เช่น 9j, 5i-4j, 45i ฯลฯ
2) จำนวนทางตรรกะ (Boolean) โดยปกติจำนวนทางตรรกะจะมีค่าเท็จ หรือจริง สำหรับ ภาษา Python จะแทนค่าเท็จด้วย เลขศูนย์, โครงสร้างเปล่า หรือ None value เช่น 0, [], {}, (), None และแทนค่าจริงด้วย ค่าใดๆที่ไม่ใช่ศูนย์, โครงสร้างที่มีข้อมูลอยู่ เช่น 1, [5], (7,8,98,347), “xyz”
3) สายอักขระ (String) เป็นข้อมูลชนิดของตัวอักษร เช่น ‘This is a string.’, “This is another string” สายอักขระนี้จะอยู่ภายในสัญลักษณ์ Single quoted หรือ Double quoted ก็ได้ ในข้อมูลชนิดสายอักขระนั้น จะมีอักขระพิเศษอยู่ด้วย ได้แก่ \n = Newline \’ = Single

quoted\b = Backspace \t = Tab \" = Double quoted \f = Formfeed Backslash\a = Bell\r = Carriage return \v = Vertical tab

4) ลิสต์ (Lists) มีลักษณะคล้ายกับข้อมูลชนิดอาเรย์ (Array) แต่จะต่างกันตรงที่ข้อมูลที่เก็บในลิสต์ไม่จำเป็นต้องเก็บข้อมูลชนิดเดียวกัน ข้อมูลที่สามารถเก็บในลิสต์ ได้แก่ ข้อมูลชนิดต่างๆ ของ Python เช่น ตัวเลข, ตัวอักษร, สายอักขระ หรือแม้แต่จะเป็น ลิสต์ด้วยกันเองก็ได้ การเขียนอ้างอิงถึงข้อมูลชนิดลิสต์ทำได้ดังนี้ list2 = [1,"two",[3,4]] เป็นการสร้างลิสต์ที่ประกอบด้วยสมาชิก 3 ตัวโดยตัวแรก เป็น ตัวเลข, ตัวที่สองเป็นสายอักขระ, ตัวที่สามเป็นลิสต์ที่มีสมาชิกเป็นตัวเลขสองตัวการอ้างถึงสมาชิกในลิสต์ทำได้ดังนี้ list2[0] จะได้ผลลัพธ์เป็น 1 ด้วยคุณสมบัติและลักษณะข้างต้น จึงสามารถใช้ลิสต์ในการสร้างสแตค (Stack) และคิว (Queue) ได้ ซึ่งจะมีประโยชน์ต่อการสร้างอินเตอร์พรีตเตอร์ภาษา CL (CL-Interpreter) ต่อไป

5) ทัปเปิล (Tuples) ข้อมูลชนิดทัปเปิลมีลักษณะเหมือนข้อมูลชนิดลิสต์ แต่จะต่างกันตรงที่ไม่สามารถแก้ไขข้อมูลของสมาชิกภายในทัปเปิลได้ ทัปเปิลจึงเหมาะแก่การใช้เป็นข้อมูลอ้างอิง การกำหนดทัปเปิลมีวิธีการดังนี้ tuple2 = ("one",2,"three",4) การจัดการกับทัปเปิลนั้นมีวิธีการเช่นเดียวกับการจัดการกับลิสต์

6) ดิกชันนารี (Dictionary) ข้อมูลชนิดดิกชันนารีจะประกอบไปด้วยคีย์ (Keys) และค่าที่เก็บ (Values) ตัวอย่างของดิกชันนารีเป็นดังนี้ dict = {1: "one", 2 : "two", 3: "three"} เราสามารถอ้างอิงข้อมูลในดิกชันนารีทำได้โดย dict[key] จะให้ผลลัพธ์เป็นอ๊อบเจ็คที่ถูกเก็บโดยคีย์นั้น เช่น dict[1] จะให้ผลลัพธ์เป็น "one" เป็นต้นโดยปกติข้อมูลชนิดลิสต์จะสามารถเรียงลำดับข้อมูลได้ แต่ข้อมูลชนิดดิกชันนารีนั้นจะไม่มีการเรียงลำดับข้อมูล การอ้างอิงทำได้โดยผ่านทางคีย์เท่านั้น

7) ข้อมูลเปล่า (None) ข้อมูลชนิดนี้ใช้ในการระบุค่าเริ่มต้นของตัวแปร หรือแสดงว่าไม่มีข้อมูล ซึ่งในการเปรียบเทียบค่านั้น None จะมีค่าเท่ากับ None เท่านั้น

2.2 นิพจน์ (Expression)

นิพจน์ (expression) คือ การดำเนินการที่ประกอบด้วยตัวดำเนินการ (operator) และตัวถูกดำเนินการ (operand) ตัวดำเนินการ ในภาษา Python แบ่งออกเป็น 4 ประเภทคือ ตัวดำเนินการทางตรรกะ, ตัวดำเนินการทางการเปรียบเทียบ, ตัวดำเนินการทางบิตไวส์ และตัวดำเนินการทางคณิตศาสตร์

1) ตัวกระทำทางตรรกะ (Logical operators) ตัวกระทำทางตรรกะมีทั้งหมด 3 ตัวด้วยกัน คือ and, or, not

2) ตัวกระทำทางการเปรียบเทียบ (Comparison operators) ตัวกระทำที่ใช้เปรียบเทียบค่าต่างๆ ได้แก่ <, >, <=, >=, ==, <>, !=, in, not in, is, is not

3) ตัวกระทำทางบิตไวส์ (Bitwise operators) เป็นตัวกระทำที่ใช้คำนวณเลขฐานสอง ได้แก่ << (Shift left), >> (Shift right), &(and), | (or), ^ (xor), ~ (not)

4) ตัวกระทำทางคณิตศาสตร์ (Arithmatic-Style operators) ตัวกระทำทางคณิตศาสตร์นี้สามารถใช้งานไม่เพียงกับตัวเลขเท่านั้น แต่ยังสามารถใช้งานกับข้อมูลชนิดอื่นๆ เช่นสายอักขระได้ ตัวกระทำดังกล่าวได้แก่ ตัวกระทำ +, -, *, /, **, %

5) ลำดับของตัวกระทำ (Precedence) ลำดับของตัวกระทำเป็นดังเช่นโปรแกรมคอมพิวเตอร์ทั่วไปซึ่งมีลำดับดังนี้คือ or, and, not, (<, <=, ==, >=, >, !=, <>, is, in, not, not in), |, ^, &, (<<, >>), (+, -), (*, /, %), **, (unary+, unary-, unary~)

2.3 Python Statement

คำสั่งลูปเป็นคำสั่งที่ใช้ในการควบคุมลำดับการประมวลผลของคำสั่งต่างๆ โดยให้ประมวลผลวนรอบไปเรื่อยๆจนกว่าจะถึงเงื่อนไขที่กำหนด คำสั่งลูปต่างๆได้แก่ IF, WHILE, FOR

2.3.1คำสั่ง IF

การประมวลผลประโยค if นั้น จะทำการตรวจสอบเงื่อนไขซึ่งกำหนดโดย expression ถ้าเงื่อนไขถูกต้อง จึงทำ Statement ที่กำหนด รูปแบบของคำสั่ง IF เป็นดังนี้

โปรแกรม 2.1 รูปแบบของคำสั่ง IF

```
if <EXPRESSION>:
        <STATEMENT>
elif <EXPRESSION>:
        <STATEMENT>
else:
        <STATEMENT>
```

โปรแกรม 2.2 ตัวอย่างของการใช้คำสั่ง if

```
if  x == 0:
      print 'x equal 0'
elif x >0:
      print 'x is positive number'
else:
      print 'x is negative number'
```

2.3.2 คำสั่ง WHILE

ใช้สำหรับการประมวลผลแบบวนรอบ ซึ่งจะทำงานไปเรื่อยๆ จนกว่าเงื่อนไข หรือ expression จะเป็นเท็จ คำสั่ง WHILE นั้นมีการตรวจสอบเงื่อนไขก่อนการทำคำสั่งในลูปทุกครั้ง รูปแบบคำสั่ง while เป็นดังนี้

โปรแกรม 2.3 รูปแบบของคำสั่ง WHILE

```
while <EXPRESSION> :
    <STATEMENT>
```

โปรแกรม 2.4 ตัวอย่างของการใช้คำสั่ง while

```
while 1:
    print 'Infinite loop'
```

2.3.3 คำสั่ง FOR

เป็นประโยคที่ทำการประมวลผลแบบวนรอบจนครบตามจำนวนที่กำหนด คำสั่ง FOR เป็นคำสั่งที่จะวนรอบการทำงานจนครบจำนวนที่กำหนด รูปแบบของคำสั่ง for เป็นดังนี้

โปรแกรม 2.5 รูปแบบของคำสั่ง FOR

```
for <VARIABLE> in <RANGE>:
    <STATEMENT>
```

โปรแกรม 2.6 ตัวอย่างของการใช้คำสั่ง for

```
list1 = [1,'a',[b,c]]
for var1 in list1:
    print var1
```

โดยผลลัพธ์ที่ได้จากการรันคำสั่งเป็นดังนี้

```
1
'a'
[b,c]
```

โดย <RANGE> คือตัวแปรที่เป็นลิสต์ หรือคำสั่งเฉพาะบางคำสั่ง ส่วน <VARIABLE> คือตัวแปรที่เก็บค่าของสมาชิกทุกตัวของ <RANGE> การทำงานของคำสั่ง FOR จะทำงานวนรอบเป็นจำนวนเท่ากับจำนวนสมาชิกของ <RANGE>

2.4 การสร้างคำนิยามของฟังก์ชัน (Function Definition)

ผู้ใช้สามารถสร้างฟังก์ชันใหม่ขึ้นเองได้ เพื่อความสะดวกในการเขียนโปรแกรม ซึ่งการสร้างฟังก์ชันใหม่นั้น มีรูปแบบคำสั่ง ดังนี้

โปรแกรม 2.7 การสร้างฟังก์ชัน

```
def <Function name> (<Parameter>) :
    <Statement>
```

สำหรับ <Parameter> หมายถึง พารามิเตอร์ต่างๆ ของฟังก์ชันตามที่ผู้เขียนกำหนด อาจจะมากกว่า 1 ตัวก็ได้ โดยใช้เครื่องหมาย "," แยกระหว่างพารามิเตอร์ต่างๆ

2.5 โมดูล (Module)

โมดูล คือ ที่เก็บฟังก์ชั่นเพื่อให้เรียกใช้สะดวก สามารถใช้ประโยค import *module_name* ในการเรียกใช้ ส่วนชื่อโมดูลจะถูกเก็บไว้ในตัวแปรรวม (เฉพาะในโมดูล) ชื่อ __name__ สามารถสรุปคุณสมบัติเพิ่มเติมของโมดูลเป็นข้อๆ ได้ดังนี้

1) สามารถตั้งให้มีโค๊ดในการรันครั้งแรก (ครั้งเดียว) ที่ถูกอิมพอร์ตได้
2) เนื่องจากต้องอ้างอิงค่าในโมดูลผ่านชื่อโมดูล ในรูปของ modname.itername จึงไม่ต้องกังวลเรื่องชื่อตัวแปรในโมดูลจะซ้ำกับตัวแปรในโปรแกรมหลัก
3) โมดูลสามารถอิมพอร์ตโมดูลอื่นได้เขียนรูปแบบการอิมพอร์ตได้หลากหลาย คือ สามารถเลือกอิมพอร์ตบางฟังก์ชั่น หรือ อิมพอร์ตทุกฟังก์ชั่น ที่ไม่ใช่ฟังก์ชั่นท้องถิ่น (ฟังก์ชั่นท้องถิ่นจะถูกนำหน้าชื่อฟังก์ชั่นด้วยสัญลักษณ์ขีดเส้นใต้ '_') แบบละเลยชื่อโมดูล

2.5.1 การค้นหาพาธของโมดูล (The Module Search Path)

การค้นหาพาธของโมดูลจะหาที่ไดเรคทอรี่ปัจจุบันก่อนถ้าไม่พบจะไปหาจากค่าในตัวแปรแวดล้อม PYTHONPATH ตามด้วยพาธของ Python เอง

2.5.2 โมดูลมาตรฐาน (Standard Modules)

Python มีโมดูลมาตรฐานมากมายซึ่งดูได้จาก บรรณสารของ Python (Python Library Reference) โมดูลหลายตัวเป็นโมดูลของระบบ บางตัวอาจเรียกใช้ได้ในบางสถานะ ตัวอย่างเช่น โมดูล sys ซึ่งเป็นโมดูลระบบ

2.5.3 ฟังก์ชั่น dir()

ฟังก์ชั่น dir() เป็นฟังก์ชั่นที่ใช้ดูว่าโมดูลนั้นประกอบไปด้วยรายชื่อ (คือตัวแปร โมดูล ฟังก์ชั่น คลาส หรืออะไรก็ตามที่เราสร้างไว้) มีอะไรบ้างและเก็บค่าเป็นลิสต์

```
>>> import __builtin__
>>> dir(__builtin__)
['ArithmeticError', 'AssertionError', 'AttributeError', 'DeprecationWarning',
 'EOFError', 'Ellipsis', 'EnvironmentError', 'Exception', 'False',
 'FloatingPointError', 'FutureWarning', 'IOError', 'ImportError',
 'IndentationError', 'IndexError', 'KeyError', 'KeyboardInterrupt',
 'LookupError', 'MemoryError', 'NameError', 'None', 'NotImplemented',
 'NotImplementedError', 'OSError', 'OverflowError',
 'PendingDeprecationWarning', 'ReferenceError', 'RuntimeError',
 'RuntimeWarning', 'StandardError', 'StopIteration', 'SyntaxError',
 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'True',
 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError',
 'UnicodeEncodeError', 'UnicodeError', 'UnicodeTranslateError',
 'UserWarning', 'ValueError', 'Warning', 'WindowsError',
 'ZeroDivisionError', '_', '__debug__', '__doc__', '__import__',
 '__name__', 'abs', 'apply', 'basestring', 'bool', 'buffer',
 'callable', 'chr', 'classmethod', 'cmp', 'coerce', 'compile',
 'complex', 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod',
 'enumerate', 'eval', 'execfile', 'exit', 'file', 'filter', 'float',
 'frozenset', 'getattr', 'globals', 'hasattr', 'hash', 'help', 'hex',
 'id', 'input', 'int', 'intern', 'isinstance', 'issubclass', 'iter',
 'len', 'license', 'list', 'locals', 'long', 'map', 'max', 'memoryview',
 'min', 'object', 'oct', 'open', 'ord', 'pow', 'property', 'quit', 'range',
 'raw_input', 'reduce', 'reload', 'repr', 'reversed', 'round', 'set',
 'setattr', 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super',
 'tuple', 'type', 'unichr', 'unicode', 'vars', 'xrange', 'zip']
```

รูป 2.1 ตัวอย่างการเรียกใช้ฟังก์ชั่น dir( )

2.5.4 แพกเกจ (Packages)

เมื่อมีโมดูลที่สร้างเพิ่มขึ้นจะต้องทำการจัดกลุ่มให้ไปรวมในไดเรคทอรี่ต่างหาก ในภาษา Python เรียกวิธีจัดการกลุ่มโมดูลนี้ว่า "แพกเกจ" ซึ่งชื่อแพกเกจก็คือชื่อไดเรคทอรี่นั่นเอง

2.5.5 การอิมพอร์ต (Importing * From a Package)

เมื่อเรียกอิมพอร์ต Python ใช้ตัวแปร __all__ ในไฟล์ __init__.py สำหรับระบุว่าในแพกเกจนี้จะต้องเรียกใช้งานโมดูลไหนบ้าง (แทนการดูจากชื่อไฟล์ในไดเรคทอรี่ เพราะมีข้อจำกัดมากสำหรับระบบปฏิบัติการที่หลากลาย)ระหว่างการอิมพอร์ต ถ้าเกิดข้อผิดพลาดขึ้นมา Python จะแจ้ง ImportError อิมพอร์ตได้เฉพาะสิ่งที่มีตัวตนและสิ่งที่ยอมให้อิมพอร์ตได้เท่านั้น คือ แพกเกจหรือโมดูล ที่เหลือนอกจากนี้คือคลาส ฟังก์ชั่น หรือตัวแปร ไม่สามารถอิมพอร์ตได้

2.5.6 การอ้างถึงกันระหว่างโมดูลในแพกเกจ (Intra-package References)

การอ้างถึงกันระหว่างโมดูลในแพกเกจมีหลักอยู่ว่าถ้าอยู่ในระดับชั้นไดเรคทอรี่เดียวกันเรียกได้โดยตรง โดยไม่ต้องมีชื่อแพกเกจนำหน้าแต่ ถ้าอยู่ลึกลงไป สามารถเรียกผ่านจากระดับเดิมได้ทันที นอกเหนือจากนี้ ต้องอ้างอิงแบบเต็มเหมือนการเรียกจากโมดูลจากแพกเกจอื่น

2.6 เนมสเปซ (Namespace)

เนมสเปซ (Namespace) เปรียบเสมือนห้องส่วนตัวนับตั้งแต่เราเอาสิ่งที่เราเขียนไว้ หรือเอาฟังก์ชั่นที่มีอยู่แล้ว มาสร้างเป็นออปเจคต์ เป็นแบบนี้ในทุกระดับของออปเจคต์ ในทางปฏิบัติแล้ว Python เก็บไว้ในรูปดิกชันนารี ตัวอย่างของเนมสเปซคือ กลุ่มของบิลด์อินเนม (Buid-in Names =

buid-in function + buid-in exception) จะอยู่ภายใต้เนมสเปซเดียวกัน หรือ กลอบอลเนม (Global Names = global function + global exception + global variables) ต่าง ๆ ของมอดูล ก็จะอยู่ในเนมสเปซของมัน การมีเนมสเปซทำให้ออปเจคต์ต่างเนมสเปซกันไม่ตีกันถึงแม้จะชื่อเดียวกัน เวลาเราอ้างถึงเนมที่อยู่ลึกลงไป เราจึงต้องอ้างตามรายแอตทริบิวต์ เนมสเปซต่าง ๆ จะถูกสร้างขึ้นเมื่อออปเจคต์ถูกสร้าง และมีอายุตามออปเจคต์นั้น ๆ เช่น

1) เนมสเปซที่บรรจุบิลด์อินเนม (Build-in Name = buid-in function + buid-in exception) จะถูกสร้างขึ้นตั้งแต่เราเริ่มเรียกใช้โปรแกรมและคงอยู่จนกว่าโปรแกรมจะจบ
2) กลอบอลเนมสเปซของมอดูล (Module Global Namespace) จะถูกสร้างขึ้นตั้งแต่มอดูลถูกอิมพอร์ตจนกว่าจะจบโปรแกรมเช่นกัน
3) ประโยคทั้งหมดที่ถูกรันโดย Python ไม่ว่าจะเป็นการรันสคริปต์ไฟล์หรือพิมพ์สดในหมวดโต้ตอบก็ดี จะอยู่ภายใต้เนมสเปซของมอดูล __main__
4) บิลด์อินเนม (Buid-in Names = buid-in function + buid-in exception) จะอยู่ภายใต้เนมสเปซของมอดูล __buildin__
5) เวลาฟังก์ชั่นถูกเรียกใช้งาน มันจะสร้างเนมสเปซของมันขึ้นมา แล้วถูกลบทิ้งไปเมื่อทำงานเสร็จ แต่ถ้าเรียกลึกลงไปเรื่อย ๆ เนมสเปซที่ลึกลงไปก็จะถูกซ้อนอยู่ภายใต้เนมสเปซที่เป็นผู้เรียก

แอตทริบิวต์ (Attribute) คือ เนม (Name) ตามด้วยจุด (.) ตามด้วยชื่อแอตทรีบิวต์ เช่น z.real เราเรียก real ว่าเป็นแอตทริบิวต์ของออปเจคต์ z อะไรก็ตามที่อยู่ในระดับเดียวกับ real คืออ้างถึงด้วย z.XXX เราจะเรียกว่าอยู่ภายใต้เนมสเปซเดียวกัน แอตทรีบิวต์ อาจเป็นได้ทั้งอ่านอย่างเดียวและเขียนได้ด้วย ถ้าเป็นแบบเขียนได้ เราก็สามารถเปลี่ยนแปลงค่าได้ และใช้ประโยค del ในการลบแอตทริบิวต์นั้นได้

สโคป (Scope) หมายถึง ช่วงที่เนมสเปซสามารถเข้าถึงได้โดยตรง โดยไม่ต้องอ้างอิงไล่แอตทริบิวต์ไปถึงออปเจคต์ที่อยู่ในเนมสเปซอื่นในทุก ๆ ขณะของการทำงาน จะมีอย่างน้อยสามสโคปเสมอ คือ สโคปในสุด (เวลาค้นหาออปเจคต์ โปรแกรมจะค้นจากสโคปในสุดเสมอ) บรรจุตัวแปรในระดับเดียวกัน(และระดับลูก)และฟังก์ชั่นในระดับลูก สโคปชั้นกลาง บรรจุตัวแปรแบบกลอบอล และฟังก์ชั่นในระดับเดียวกัน สโคปชั้นนอกสุด บรรจุพวกบิลด์อินเนม (Buid-in Names = buid-in function + buid-in exception)

2.7 การสร้างคลาส (Class Definition)

ผู้ใช้สามารถสร้างคลาสขึ้นเองได้ด้วยรูปแบบคำสั่ง ดังนี้

โปรแกรม 2.8 การสร้างคลาส

```
class <Class name> :
        <Method>
```

เมื่อทำการนิยามคลาสแล้ว ผู้เขียนสามารถสร้างอินสแตนส์ (instance) ของคลาส และเรียกเมธอด (Method) จากอินสแตนส์นั้นได้ ตัวอย่างเช่น

โปรแกรม 2.9 ตัวอย่างการสร้างคลาส

```
class MyDataWithMethod :
    def printFoo(self) :
        print 'You invoked printFoo()!'
```

ผู้เขียนสามารถสร้างอินสแตนส์ของคลาสนี้ได้ด้วยการใช้คำสั่ง myObj = MyDataWith Method() เมื่อต้องการเรียกเมธอดในคลาสก็ทำได้โดย myObj.printFoo() หน้าจอก็จะแสดง You invoked printFoo()! ออกมาเป็นผลลัพธ์

2.8 Inheritance and Polymorphism

2.8.1 การสืบทอดคลาส (Inheritance)

คุณสมบัติในการสืบทอดเป็นจุดเด่นอย่างหนึ่งของ OOP โดยสามารถสร้างคลาสหลักขึ้นมา 1 คลาสแล้วกำหนดให้มีคุณสมบัติต่างๆ ที่จำเป็นต้องมี แล้วจึงสร้างคลาสอีกคลาสหนึ่งขึ้นมาเพื่อรับคุณสมบัติทั้งหมดจากคลาสหลัก และเพิ่มคุณสมบัติอื่นเข้าไปอีกในคลาสนี้ เราจะเรียกคลาสหลักว่า “Superclass” (คลาสแม่) และเรียกคลาสย่อยว่า “Subclass” (คลาสลูก)

หลักการการสืบทอดคลาส “Inheritance” คือการที่ “Class” ใดๆยอมให้การสืบทอด (Inherit) ทั้งคุณสมบัติ (Attributes) และการทำงาน (Method) ไปยัง “Class” อื่น ตัวอย่างเช่น สมมติว่าเรากำลังพิจารณาในเรื่องสัตว์เลี้ยงลูกด้วยนม (Mammal) โดยได้กำหนดคลาสขึ้นมา 2 คลาส คือ หมา (Dog) กับ แมว (Cat) และแต่ละคลาสเหล่านี้จะมี Attributes สีของตา (Eye Color) ซึ่งถ้ามองในแง่ของโปรแกรมแบบกระบวนความแล้ว จะพบว่าแต่ละ Code ทางตรงข้ามสำหรับการออกแบบเชิงวัตถุแล้ว Attributes นี้แยกออกเป็นของใครของมัน ในทางตรงข้ามสำหรับการออกแบบเชิงวัตถุแล้ว Attributes ของสีตาจะถูกย้ายไปเก็บไว้ที่ Class “สัตว์เลี้ยงลูกด้วยนม” เพียงที่เดียว เนื่องจากว่าเป็นสิ่งที่มีลักษณะที่สามารถใช้ได้ทั้งคลาส “หมา” และ “แมว” ในกรณีเช่นนี้ ทั้งคลาส “หมา”และ “แมว” จะได้รับการสืบทอดคุณสมบัติมาจากคลาสสัตว์เลี้ยงลูกด้วยนมการสืบทอดคลาส (Inheritance) มีรูปแบบโครงสร้างดังนี้คือ

```
class DerivedClassName(BaseClassName):
    <statement-1>
    .
    .
    .
    <statement-N>
```

รูป 2.2 โครงสร้างการสืบทอดคลาส

ถ้าคลาสหลักถูกกำหนดไว้ที่ Module อื่น รูปแบบจะเป็น

```
class DerivedClassName(modname.BaseClassName):
```

รูป 2.3 โครงสร้างการสืบทอดคลาสที่ถูกกำหนดไว้ที่มอดูลอื่น

คลาสใหม่ที่แตกออกมานี้ สามารถสร้างเมธอดเพื่อครอบงำเมธอดเดิมได้อย่างไม่มีข้อจำกัด โดยที่ถ้าสร้างขึ้นมาแล้วโค๊ดภายใต้เมธอดเดิมจะไม่ถูกเรียก แต่หากยังต้องการเรียกโค๊ดจากเมธอดเดิมอยู่ เราต้องเรียกใช้เองในรูปแบบ BaseClassName.methodname(self, arguments)

2.8.2 การสืบทอดหลายทาง (Multiple Inheritances)

การสืบทอดหลายทางมีรูปแบบโครงสร้างคือ

```
class DerivedClassName(Base1, Base2, Base3):
    <statement-1>
    .
    .
    .
    <statement-N>
```

รูป 2.4 โครงสร้างการสืบทอดหลายทาง (Multiple Inheritances)

ลำดับการทำงานก็คือ ถ้าพบคลาส Base1 ตรง ๆ ก็จะสืบทอดจาก Base1 เลย แต่ถ้าไม่พบก็จะควานหาคลาสฐานของ Base1 ลึกลงไปจนสุด และถ้ายังไม่พบจึงมาเริ่มต้นค้นจาก Base2 ต่อไปเรื่อย ๆ จนหมด

2.8.3 Abstract Class

Abstract Class คือ โครงร่างของคลาส ซึ่งภายใน Abstract Class มี attributes และ Method ที่ยังใช้งานไม่ได้ ต้องรอให้ คลาสอื่นสืบทอดคุณสมบัติไปสร้างให้สมบูรณ์ก่อนถึงจะเรียกใช้งานได้ เช่น คลาสของสัตว์เลี้ยงลูกด้วยนมจะเป็น Abstract Class ที่ไม่สามารถสร้างเป็นอ็อปเจ็คได้

2.8.4 Polymorphism

polymorphism มาจากคำในภาษากรีกสองคำ คือ poly (โพลี) แปลว่า หลาย และ morphism (มอร์ฟีส) แปลว่า รูปร่าง polymorphism จึงแปลว่า "หลายรูปทรง" หรือ "แปลงสภาพได้" Polymorphism คือสภาวะที่ Method มีหลายรูปแบบ เป็นวิธีการกำหนดรูปแบบการกระทำที่เหมือนกันแต่ได้ผลที่แตกต่างกัน สมมุติว่ามี คลาสแม่ (Superclass) ที่เป็นคลาสของสัตว์เลี้ยงลูกด้วยนม โดยจะต้องมีคลาสลูก (Subclass) คือ คลาสของคนกับสุนัข ซึ่งทั้งคนกับสุนัขมี Method ที่ใช้ในการเปล่งเสียงซึ่งการเปล่งเสียงระหว่างคนกับสุนัขนั้นไม่เหมือนกัน แต่มันก็คือการเปล่งเสียง ดังนั้นเมื่อได้รับ Message ที่บอกให้ทำการเปล่งเสียงทั้งคนและสุนัขก็สามารถเปล่งเสียงได้เพียงแต่มี เสียงที่แตกต่างกันเท่านั้นเอง เพื่อให้สามารถเข้าใจได้ง่ายเราก็จะสร้าง Method ที่ชื่อว่า เปล่งเสียง ( ) มาเพียงชื่อเดียวแล้วใช้หลักการ polymorphism นี้เพื่อให้สามารถเปล่งเสียงได้หลายๆ รูปแบบ

```
1  class Transportation(object):
2      """Abstract base class"""
3      def __init__( self, start, end, distance ):
4          if self.__class__ == Transportation:
5              raise NotImplementedError
6          self.start = start
7          self.end = end
8          self.distance = distance
9      def find_cost( self ):
10         """Abstract method; derived classes must override"""
11         raise NotImplementedError
12 class Walk( Transportation ):
13     def __init__( self, start, end, distance ):
14         Transportation.__init__( self, start, end, distance)
15     def find_cost( self ):
16         return 0
17 class Taxi( Transportation ):
18     def __init__( self, start, end, distance ):
19         Transportation.__init__( self, start, end, distance)
20     def find_cost( self ):
21         return self.distance * 40
22 class Train( Transportation ):
23     def __init__( self, start, end, distance, station_no ):
24         Transportation.__init__( self, start, end, distance)
25         self.station_no = station_no
26     def find_cost( self ):
27         return self.station_no * 5
28
```

รูป 2.5 ตัวอย่าง code ที่ใช้หลักการ polymorphism

จากรูปพบว่า คลาสแม่ (Superclass) เป็น Class transportation โดยมีคลาสลูก (Subclass) คือ Class Walk, Class Taxi และ Class Train ซึ่งมี Method find_cost ที่ใช้ในการคำนวณ ค่าการเดินทางแตกต่างออกไปตามแต่ละประเภท

บทที่ 3

ภาษา CL

เนื้อหาในบทนี้จะกล่าวถึงโครงสร้างและไวยากรณ์ทั้งหมดของภาษา CL โดยเริ่มจากการยกตัวอย่างโปรแกรมภาษา CLหัวข้อ 3.1 แล้วจึงแสดงไวยากรณ์ทั้งหมดของภาษา CL ในรูปแบบ BNFในหัวข้อที่ 3.2 หลังจากนั้นจึงกล่าวถึงองค์ประกอบต่างๆ ของภาษา CL โดยละเอียด เริ่มด้วยชนิดข้อมูลของภาษา CL ในหัวข้อที่ 3.3 ซึ่งส่วนมากจะเป็นชนิดข้อมูลเดิมที่ปรากฏในภาษา Python หลังจากนั้น ในหัวข้อที่ 3.4 – 3.5 จึงกล่าวถึงรายละเอียดของการเขียนโปรแกรมภาษา CL เช่น การกำหนดประโยคต่างๆ โอเปอเรเตอร์และเอ็กซ์เพรสชั่นของภาษา CL ใน และสุดท้ายในหัวข้อ 3.6 เป็นการโปรแกรมด้วยภาษา CL

3.1 ตัวอย่างโปรแกรมภาษา CL

ในหัวข้อนี้จะยกตัวอย่างโปรแกรมภาษา CL อย่างง่าย โดยตัวอย่างนี้จะเป็นโปรแกรมที่จะทำการแจ้งเตือนทุกๆหนึ่งนาที ด้วยข้อความว่า HelloWorld ซึ่งมีลักษณะดังนี้

โปรแกรม 3.1 แจ้งเตือนทุกๆ 1 นาที

```
X = [2011,2,12,0,31,0]
Y = [2011,2,12,0,32,2]
i=1
n=5
while(i<=n):
   time[X,Y]:
        PopUp('HelloWorld')
   Y = Y addtime [0,0,0,0,1,0]
   X = X addtime [0,0,0,0,1,0]
   i+=1
```

3.2 BNF ของภาษา CL

ไวยากรณ์ทั้งหมดของภาษา CL แสดงไว้ในรูปแบบ BNF (Backus Naur Form) ดังนี้

1) <cl-program> ::= <define-function> | <comm-script> | <action-statement>
2) <define-function> ::= def <id> ([<id> (, <id>)*]) : <action-statement>+ [return [<expression>]]
3) <comm-script> ::= commscript : <comm_grp>+
4) <comm_grp> ::= <comm_sequence> | <comm_unordered> | <comm_indp_total> |

<comm_one> | <comm_stmt>

5) <comm_sequence> ::= comm_sequence : <comm_stmt>+

6) <comm_unordered> ::= comm_unordered : <comm_stmt>+

7) <comm_indp_total> ::= comm_independent_total : <comm_stmt>+

8) <comm_one> ::= comm_one : <comm_stmt>+

9) <comm_stmt> ::= <comm_msg> → <action-statement>+

10) <comm_msg> ::= [<interval_time> :] <comm_expr>

11) <comm_expr> ::= exist(<comm_type>)

12) <comm_type> ::= <mode> (<msg>,<snd>,<recv>,<msg_id>,<msg_ret_id>)

13) <mode> ::= tell_msg | query_msg | request_msg | delete_msg | please_tell_msg | please_query_msg | please_request_msg | please_delete_msg | yes_msg | no_msg

14) <msg> ::= <string> | <id> | <retid> | _

15) <snd> ::= <string> | <id> | <retid> | _

16) <recv> ::= <string> | <id> | <retid> | _

17) <msg_id> ::= <number> | <id> | <retid> | _

18) <msg_ret_id> ::= <number> | <id> | <retid> | _

19) <action-statement> ::= <module_stmt> | <statement> | <comment>

20) <module_stmt> ::= import <id>

21) <statement> ::= [<interval_time> :] <statement_type>

22) <statement_type> ::= <assign_stmt> | <control_stmt> | <group_stmt>

23) <assign_stmt> ::= <left_value> (= | += | -= | *= | /= | %= | ^^= | <<= | >>= | &= | |= | ^=) <expression>

24) <left_value> ::= <id> | <access>

25) <control_stmt> ::= <if_stmt> | <while_stmt> | <for_stmt> | <repreat_stmt>

26) <if_stmt> ::= if <expression> : <action-statement>+ (elif <expression> : <action-statement>+)* [else : <action-statement>+]

27) <while_stmt> ::= while <expression> : <action-statement>+

28) <for_stmt> ::= for <id> in <expression> : <action-statement>+

29) <repreat_stmt> ::= repeat : <action-statement>+ until <expression>

30) <group_stmt> ::= (sequence | unordered | alternative | parallel) : <action-statement>+

31) <expression> ::= <unary_expr> | <binary_expr> | <bitwise_expr> | <shift_expr> | <extend_expr> | <shrink_expr> | <boolean_expr> | <logic_expr> |<group_expr> |

<term>

32) <unary_expr> ::= (- | + | ~) <expression>

33) <binary_expr> ::= <expression> (+ | - | * | / | ** | %) <expression>

34) <bitwise_expr> ::= <expression> (& | | | ^) <expression>

35) <shift_expr> ::= <expression> (<< | >>) <expression>

36) <extend_expr> ::= <expression> (<- | ->) <expression>

37) <shrink_expr> ::= <expression> (>- | -<) <expression>

38) <boolean_expr> ::= <expression> (< | > | <= | >= | == | != | mi | m | oi | o | di | d | si| s | fi | f) <expression>

39) <logic_expr> ::= <expression> (and | or) <expression>

40) <group_expr> ::= (<expression>)

41) <term> ::= <number> | <function> | <id> | <retid> | <string> | <meta> | <time> | <list> | <tuple> | <dictionary> | <access> | <boolean> | <none>

42) <number> ::= <int_number>|<float_number>|<long_number>|<complex_number>

43) <int_number> ::= (<digit>)+

44) <float_number> ::= (<digit>)+ . (<digit>)+

45) <long_number> ::= (<digit>)+ (l|L)

46) <complex_number> ::= (<digit>)+ [. <digit>+] (+|-) <digit>+ [. <digit>+] (j|J)

47) <function> ::= <id> ([<expression> (, <expression>)*])

48) <id> ::= (<upper>|<lower>) (<upper>|<lower>|<digit>|_)*

49) <retid> ::= ? (<upper>|<lower>) (<upper>|<lower>|<digit>|_)*

50) <string> ::= “(<upper>|<lower>|<digit>|<alpha>|<special>)* ”

51) <meta> ::= ‘’’ (<upper>|<lower>|<digit>|<alpha>|<special>)* ‘’’

52) <time> ::= <point> | <duration> | <interval>

53) <point>::=((0|1)(<digit>)|2(0…3)):(0…5)(<digit>):(0…5)(<digit>)|((0…2)<digit>)|3(0|1)) / ((1…9)|1(0…2)) / (<digit>)^4

54) <duration>::= <digit>+ [.<digit>+] (min|d|h|m|s)(, <digit>+ [.<digit>+] (min|d|h|m|s))*

55) <interval> ::= < (<point>|_) , (<point>|_) >

56) <list> ::= [[<expression> (, <expression>)*]]

57) <tuple> ::= ([<expression> (, <expression>)*])

58) <dictionary> ::= { [<expression> : <expression> (, <expression> : <expression>)*] }

59) <access> ::= <id> [<index> [: <index>]]

60) <index> ::= <int_number> | <id> | <string>

61) <boolean> ::= true | false

62) <none> ::= none

63) <upper> ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

64) <lower> ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

65) <digit> ::= 0|1|2|3|4|5|6|7|8|9

66) <alpha> ::= !|$|%|&|*|+|-|.|/|<|>|=|?|@|_|~

67) <special> ::= "|#|'|(|)|,|\|^

ไวยากรณ์ของภาษา CL แสดงด้วย BNF โดยสัญลักษณ์ที่แสดงถึง nonterminal จะถูกกำหนดภายใต้เครื่องหมาย < > ส่วน terminal จะแสดงด้วยอักษรตัวเข้ม นอกจากนี้ ยังมีสัญลักษณ์ที่ใช้กำหนดจำนวนการเกิดของส่วนต่างๆ อันได้แก่ การไม่เกิดหรือเกิดเพียง 1 ครั้ง (zero or one) การไม่เกิดหรือเกิดตั้งแต่ 1 ครั้งขึ้นไป (zero or more) การเกิดตั้งแต่ 1 ครั้งขึ้นไป (one or more) และการเกิดทั้งหมด n ครั้ง ซึ่งแสดงด้วยสัญลักษณ์ [<nonterminal>], <nonterminal>*, <nonterminal>+ และ <nonterminal>^n ตามลำดับ

ภาษา CL ประกอบด้วยส่วนประกอบหลักๆ 3 ส่วน ได้แก่ การนิยามฟังก์ชัน (<define-function>), สคริปต์ตอบสนองการสื่อสาร (<comm-script>) และส่วนการทำงานคำสั่ง (<action-statement>) ดังที่แสดงในบรรทัดที่ 1 โปรแกรมภาษา CL จะต้องประกอบด้วยส่วนประกอบนี้อย่างน้อย 1 ส่วน และต้องกำหนดส่วนประกอบตามลำดับ เริ่มจาก <define-function>, <comm-script> และ <action-statement>

การนิยามฟังก์ชัน (<define-function>) มีไวยากรณ์ตามบรรทัดที่ 2 โดยผู้เขียนสามารถกำหนดให้มีการคืนค่าจากฟังก์ชันหรือไม่ก็ได้ ตัวอย่างเช่น def add(x,y) : return (x+y) เป็นการนิยามฟังก์ชันที่ทำการบวกเลข 2 จำนวน และคืนค่าผลบวกนั้นออกมา

สำหรับส่วนสคริปต์ตอบสนองการสื่อสาร (<comm-script>) แสดงไว้ในบรรทัด 3-11 ซึ่งจะประกอบด้วยลิสต์ของการตอบสนองการสื่อสารในแบบต่างๆ ทั้งที่เป็นคำสั่งตอบสนอง (<comm_stmt>) และแบบบทสนทนา รูปแบบไวยากรณ์ของคำสั่งตอบสนอง แสดงไว้ในบรรทัดที่ 9 ซึ่งเป็นลักษณะ exist(comm_msg) → action นั่นคือ เมื่อได้รับข้อความสื่อสารแล้วให้ตอบสนองด้วยการทำคำสั่งต่างๆ ซึ่งข้อความสื่อสารจะแบ่งออกเป็นโหมด เช่น tell, query, request เป็นต้น และข้อความจะประกอบด้วยส่วนสาร (<msg>), ผู้ส่ง (<snd>), ผู้รับ (<recv>), หมายเลขสาร (msg_id), หมายเลขสารที่ตอบสนอง (msg_ret_id) ตามไวยากรณ์ที่แสดงในบรรทัดที่ 12-18 เช่น

exist(tell_msg(?msg,_,_,_,_)) → assert(msg) เป็นต้น สำหรับการตอบสนองแบบบทสนทนาในบรรทัดที่ 4-8 จะประกอบด้วยการตอบสนองแบบ sequence, unordered, independent repeatable and total, one ซึ่งจะประกอบด้วยลิสต์ของคำสั่งตอบสนอง (<comm_stmt>)

สำหรับส่วนคำสั่งการทำงาน (<action-statement>) หลักๆ แล้วจะประกอบด้วยประโยค (statement) ซึ่งผู้เขียนสามารถกำหนดช่วงเวลาสำหรับการทำคำสั่งได้ตามบรรทัดที่ 21 ตัวอย่างเช่น <10:00:00|21/12/2010, 11:00:00|21/12/2010> : play(movie) เป็นการกำหนดให้ทำการเปิดภาพยนตร์ movie ภายในเวลา 10.00-11.00 น. ของวันที่ 21 ธันวาคม 2010 เป็นต้น ประโยคในภาษา CL จะแบ่งเป็นประโยคกำหนดค่า(<assign_stmt>), ประโยคควบคุมลำดับการทำงาน (<control_stmt>), ประโยคคำสั่ง และประโยคชุดคำสั่ง (<group_stmt>)

ไวยากรณ์ของประโยคกำหนดค่าแสดงในบรรทัดที่ 23 ซึ่งเป็นการกำหนดค่าที่ได้จากเอ็กซเพรสชัน (<expression>) ให้กับตัวแปร เช่น x = 10 เป็นต้น ส่วนประโยคควบคุมลำดับการทำงานในโปรแกรมจะประกอบด้วย ประโยค if, for, while และ repeat ดังแสดงไว้ในบรรทัดที่ 25-29 ตัวอย่างเช่น repeat : print(x); x+=1; until x>5 กำหนดให้พิมพ์ค่า x แล้วเพิ่มค่า x ขึ้นทีละ 1 จนกว่า x จะมีค่ามากกว่า 5 เป็นต้น สำหรับส่วนประโยคชุดคำสั่ง (<group_stmt>) จะแบ่งเป็นแบบ sequence, unordered, alternative และ parallel ซึ่งมีไวยากรณ์ตามที่กำหนดในบรรทัดที่ 30

ในบรรทัดที่ 31-40 แสดงเอ็กซ์เพรสชันต่างๆ ใน CL ตามประเภทของโอเปอเรเตอร์อันได้แก่ โอเปอเรเตอร์ทางตรรกะ, โอเปอเรเตอร์ทางการเปรียบเทียบ, โอเปอเรเตอร์คำนวณเลขฐานสอง, โอเปอเรเตอร์ทางคณิตศาสตร์ และโอเปอเรเตอร์สำหรับเวลา ตัวอย่างเช่น 3+2 เป็นเอ็กซ์เพรสชันของการทำโอเปอเรเตอร์ทางคณิตศาสตร์ ซึ่งค่าที่ประเมินออกมาได้คือ 5 เป็นต้น

ในบรรทัดที่ 41-66 แสดงเทอมต่างๆ ใน CL ซึ่งประกอบด้วย ตัวเลข (<number>) เช่น 12 เป็นเลขจำนวนเต็ม (<int_number>), ฟังก์ชัน (<function>) เช่น add(2,3), ตัวแปร (<id>) เช่น id1 , ตัวแปรแบบคืนค่า (<retid>) เช่น ?id1, สายอักขระ (<string>) เช่น “Hi”, ข้อมูลเมต้า (<meta>) เช่น ‘‘‘12’‘‘ เป็นข้อมูลเมต้าของจำนวนเต็ม 12, จุดเวลา (<point>) เช่น 10:00:00|12/12/2010, ระยะเวลา (<duration>) เช่น 2d, ลิสต์ (<list>) เช่น [1,”john”], ทัปเปิล (<tuple>) เช่น (1,”john”), ดิกชันนารี (<dictionary>) เช่น {1:”john”,2:”mary”}, บูลีน (<boolean>) เช่น true และ ข้อมูลเปล่า (<none>) โดยขยายมาจากภาษา Python

3.3 ชนิดข้อมูลในภาษา CL

ภาษา CL เกิดจากการเพิ่มขยายภาษา Python ให้มีความสามารถในการประมวลผลเกี่ยวกับเวลาและการสื่อสาร อันเป็นแนวคิดใหม่ซึ่งไม่มีในภาษาคอมพิวเตอร์ปัจจุบัน ไวยากรณ์หลักๆ ในภาษา CL มีดังนี้

1) ประเภทข้อมูล (Data Type) ประเภทข้อมูลในภาษา CL ส่วนใหญ่จะเป็นประเภทข้อมูลเดิมที่มีในภาษา Python ได้แก่ Numbers, Boolean values, String, Lists, Tuples, Dictionary และ None ส่วนประเภทข้อมูลใหม่ที่เพิ่มเติม ได้แก่ Time
2) โอเปอเรเตอร์ (Operators) โอเปอเรเตอร์ที่ใช้ในภาษา CL แบ่งเป็น 4 ประเภทคือ โอเปอเรเตอร์ทางตรรกะ, โอเปอเรเตอร์ทางการเปรียบเทียบ, โอเปอเรเตอร์คำนวณเลขฐานสอง และโอเปเรเตอร์ทางคณิตศาสตร์ ซึ่งโอเปอเรเตอร์ทั้ง 4 มาจากภาษา Python แต่ก็ได้เพิ่มเติมโอเปอเรเตอร์ที่ประมวลผลเกี่ยวกับเวลาเข้ามาด้วย
3) เอ็กซ์เพรสชั่น (Expression) เป็นกลุ่มของโอเปอเรเตอร์และโอเปอแรนด์ ซึ่งเมื่อผ่านการประเมินค่า (Evaluation) จะให้ผลลัพธ์เป็นค่าออกมา ซึ่งค่าที่ได้สามารถนำไปใช้ในส่วนของประโยคได้ โดยค่าที่ได้จะเป็นข้อมูลประเภทใดนั้น ขึ้นกับประเภทโอเปอเรเตอร์และโอเปอแรนด์ที่ใช้ในเอ็กเพรสชั่นนั้นๆ
4) ประโยค (Statement) ในโปรแกรมภาษา CL มีความแตกต่างจากในภาษาคอมพิวเตอร์ทั่วไป คือ มีการกำหนดช่วงเวลาทำงานให้กับทุกประโยคในรูปแบบของ “ช่วงเวลา : ประโยค” ซึ่งหมายถึง ประโยคจะทำงานในช่วงเวลานี้ ถ้าผู้เขียนโปรแกรมไม่ต้องการระบุช่วงเวลานี้ก็จะได้ประโยคที่ใช้กันในภาษาคอมพิวเตอร์ทั่วไป ประโยคมีหลายแบบ คือ ประโยคสำหรับกำหนดค่าให้ตัวแปร, ประโยคคำสั่ง, ประโยคสำหรับควบคุมลำดับการทำงานในโปรแกรม และ ประโยคชุดคำสั่ง โดยอธิบายได้ดังนี้
   4.1) ประโยคสำหรับการกำหนดค่าให้ตัวแปร (Assignment Statement) การกำหนดค่าให้ตัวแปรใช้โอเปอเรเตอร์กำหนดค่า (‘=’) ด้วยรูปแบบ Var = Value
   4.2) ประโยคคำสั่ง เป็นคำสั่งที่ใช้สั่งงานในโปรแกรม สำหรับในภาษา Python ประโยคคำสั่งมี 3 รูปแบบคือ function(), module.function() และ class.function() ซึ่งภาษา CL ณ ปัจจุบัน สามารถรองรับรูปแบบประโยคคำสั่งในแบบหนึ่งและสองเท่านั้น ในอนาคตจะขยายให้ครอบคลุมแบบที่สาม
   4.3) ประโยคสำหรับควบคุมลำดับการทำงานในโปรแกรม (Control Statement) ใน CL มี

ประโยคสำหรับควบคุมลำดับการทำงานของโปรแกรมทั้งหมด 4 รูปแบบ คือ รูปแบบประโยค if, ประโยค while และประโยค for ซึ่งใช้ตามภาษา Python ทั้งหมด รวมทั้งประโยค repeat-until ที่ได้เพิ่มเติมเข้ามา

4.4) ประโยคชุดคำสั่ง เกิดจากการนำประโยคแบบต่างๆ มารวมกัน ประโยคชุดคำสั่งแบ่งเป็น 4 แบบ คือ แบบ Sequence, Unordered, Alternative, Parallel และ Nested

5) สคริปต์ตอบสนองต่อเหตุการณ์ (Event-action script) ใน CL ผู้ใช้สามารถกำหนดการตอบสนองต่อเหตุการณ์ที่ได้รับจากภายนอกด้วยสคริปต์ตอบสนองต่อเหตุการณ์ อันประกอบด้วย ประโยคการตอบสนองเหตุการณ์ และ ประโยคชุดของการตอบสนองต่อเหตุการณ์ ซึ่งการทำงานในส่วนนี้ยังไม่มีปรากฏในภาษา Python

5.1) ประโยคการตอบสนองเหตุการณ์ อยู่ในรูปแบบ เหตุการณ์ → การทำงาน หรือ Event → Action โดยที่ "เหตุการณ์" ในที่นี้ คือ Expression หรือ ประโยคคำสั่งที่ให้ค่าออกมาเป็นค่า Boolean ซึ่งในโครงงานนี้ได้อาศัยโครงสร้างจากงานวิจัยที่ผ่าน

5.2) ประโยคชุดของการตอบสนองเหตุการณ์ เกิดจากการนำประโยคการตอบสนองต่อเหตุการณ์มารวมกันเข้าเป็นชุด มีอยู่ 6 แบบ คือแบบ Independent Repeatable and Partial, Independent Repeatable and Total, Sequence, Unordered, One และ Nested

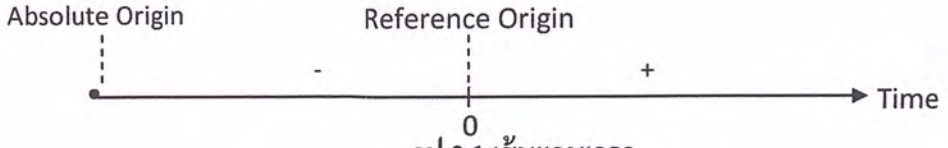
ดังที่กล่าวไว้ข้างต้นว่า ประเภทข้อมูลในภาษา CL นั้นประกอบด้วยประเภทข้อมูลเดิมที่มีในภาษา Python ได้แก่ Numbers, Boolean values, String, Lists, Tuples, Dictionary และ None รวมกับประเภทข้อมูลชนิดใหม่ที่กำหนดเพิ่มเติมเข้ามา ได้แก่ ข้อมูลประเภท Time (เวลา) โดยจะกล่าวถึงรายละเอียดในหัวข้อถัดไป สำหรับชนิดข้อมูลอื่น เช่น Numbers ได้อธิบายไปแล้วในบทที่ 2 เรื่องภาษา Python จะไม่ขอกล่าวถึงในบทนี้

3.3.1 ชนิดข้อมูลประเภทเวลา (Time)

เป็นข้อมูลที่ใช้แสดงเวลาในภาษา CL ซึ่งแบ่งได้ 2 ประเภท ได้แก่ จุดเวลา (Point) และช่วงเวลา (Interval) โดยก่อนที่จะศึกษารายละเอียดของข้อมูลประเภทเวลา ขอกล่าวถึงความหมายของเวลาก่อนเป็นดังนี้

ความหมายของเวลาในปริญญานิพนธ์นี้ได้อ้างอิงแนวคิดจากงานวิจัยที่ผ่านมาได้เสนอความคิดที่ว่า เวลาเกิดจากการดำเนินไปของกระบวนการ (Process) อันหนึ่งอย่างต่อเนื่อง เมื่อไรที่เกิดมีกระบวนการ ก็จะเกิดเวลาขึ้น แต่ถ้าไม่มี เวลาก็ไม่ปรากฏ เนื่องจากการดำเนินไปของกระบวนการมีความต่อเนื่อง จึงสมมุติให้เวลามีความต่อเนื่องด้วย โดยให้มีค่าเพิ่มขึ้นโดยต่อเนื่องอยู่ตลอดไม่ขาดตอน จึงอาจอธิบายได้โดยเส้นลูกศรในระบบ co-ordinate โดยให้เป็นแกนของเวลา

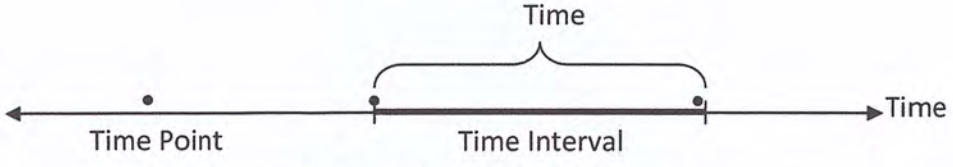
ดังรูป 3.1 ซึ่งถือว่าเป็นแกนที่เพิ่มเติมจากแกน xyz ในระบบเรขาคณิต ซึ่งอาจถือว่าเป็นแกนที่ 4 หรือมิติที่ 4



รูป 3.1 เส้นแกนเวลา

ดังนั้นเวลาจะมีค่าเพิ่มขึ้นเรื่อยๆ เมื่อนับจากจุดเริ่มต้นในอุดมคติของการเกิดเวลา (Absolute Origin) แต่ในความเป็นจริงนั้น เราไม่สามารถรู้ถึงจุดกำเนิดของเวลาที่แท้จริงได้ มนุษย์จึงได้กำหนดจุดเริ่มต้นสมมุติของเวลาขึ้นมาเพื่อง่ายต่อการอ้างอิงเวลาเท่านั้น และขอเรียกจุดเริ่มต้นเวลานี้ว่า "จุดเริ่มอ้างอิง" (Reference Origin) จุดต่างๆ บนเส้นแกนเวลานั้น แสดงการอ้างอิงเวลา ณ ขณะใดขณะหนึ่ง จุดนี้เรียกว่า จุดเวลา (Time Point) โดยจุดเวลาที่อยู่ขวามือของจุดเริ่มอ้างอิง (Reference Origin) มีค่าเป็นบวก และจุดที่อยู่ซ้ายมือของจุดเริ่มอ้างอิง มีค่าเป็นลบ เช่น จุดเวลาที่ 09:00:00 น. วันที่ 19 ต.ค. พ.ศ. 2553 เป็นจุดเวลาที่เป็นบวกเมื่อเทียบกับจุดเริ่มอ้างอิงตามระบบพุทธศักราช เป็นต้น

ในกรณีที่มีการลากเส้นเชื่อมต่อระหว่าง 2 จุดเวลา จากจุดเวลาที่น้อยกว่าซึ่งเรียกว่าจุดเริ่มต้น ไปยังจุดเวลาที่มีค่ามากกว่าซึ่งเรียกว่าจุดสิ้นสุด จะทำให้เกิด ช่วงเวลา (Time Interval) ซึ่งประกอบด้วยจุดเวลาหลายๆ จุด ที่เรียงต่อกันอย่างต่อเนื่องไม่ขาดสายจากจุดเริ่มไปจนถึงจุดสิ้นสุด โดยความยาวของช่วงเวลาหรือของเส้นตรงนี้ ถือว่าเป็น ระยะเวลา (Time Duration) จะสังเกตว่า ในกรณีที่จุดเริ่มต้นของช่วงเวลาอยู่ที่ตำแหน่งจุดเริ่มอ้างอิง (Reference Origin) ค่าของจุดสิ้นสุดของช่วงเวลาจะมีค่าเท่ากับระยะเวลาของช่วงเวลานั้น เช่น ณ เวลา 9:00:00 น. วันที่ 17 มิถุนายน พ.ศ. 2553 จะมองได้ว่าเป็น จุดเวลานี้ห่างจากจุดอ้างอิงเป็นระยะเวลา 2553 ปี 7 เดือน 17 วัน และ 9 ชั่วโมง ได้เช่นกัน ในทำนองเดียวกัน จะได้ว่าจุดเริ่มอ้างอิง หมายถึง จุดเวลาที่มีระยะเวลาเป็นศูนย์เมื่อเทียบกับตำแหน่งอ้างอิง หรือตัวมันเอง ดังแสดงในรูป 3.2



รูป 3.2 เวลาประเภทต่างๆ บนแกนเวลา

จากแนวคิดเรื่องเวลาที่ได้อธิบายไปแล้ว หัวข้อต่อไปจะกล่าวถึงรูปแบบของเวลาทั้ง 2 ประเภท คือ จุดเวลา และ ช่วงเวลา

1) รูปแบบจุดเวลา (Time Point) จุดเวลาสามารถกำหนดด้วยรูปแบบดังนี้

1.1) รูปแบบทางคณิตศาสตร์ของจุดเวลาในทางคณิตศาสตร์ จุดเวลาจะถูกแสดงให้อยู่ในรูปของหน่วยเวลาที่เล็กที่สุดเท่าที่นัยสำคัญจะมีผลกระทบต่อการทำงา (ในทางทฤษฎีควรจะมีค่าน้อยที่สุดเท่าที่จะทำได้) ซึ่งเริ่มต้นนับจำนวนหน่วยเวลานั้นตั้งแต่จุดเริ่มอ้างอิง ในที่นี้กำหนดให้เป็นหน่วยวินาที เพื่อง่ายต่อการสั่งงานคอมพิวเตอร์ เช่นจุดเวลา ณ วันที่ 1 มกราคม ค.ศ. 0001 เวลา 10.00 น. หมายถึง จุดเวลา ณ วินาทีที่ 36000 ในรูปแบบทางคณิตศาสตร์ (เวลาผ่านไปแล้ว 10 ชั่วโมง) เมื่อจุดอ้างอิงคือ วันที่ 1 มกราคม ค.ศ. 0001 เวลา 00.00 น. เป็นต้น จะสังเกตว่า การกำหนดจุดเวลาด้วยรูปแบบนี้ เป็นรูปแบบพื้นฐานที่เหมาะสำหรับการคำนวณโดยเฉพาะ ถ้าเป็นรูปของเดือน ปี การเพิ่มของเดือนปีจะไม่สม่ำเสมอเนื่องจากบางเดือนมี 28 วัน บางเดือนมี 30 วัน เป็นต้น

1.2) รูปแบบจุดเวลาที่ใช้ในชีวิตประจำวัน เป็นการกำหนดจุดเวลาในรูปแบบของ วัน เดือน ปี ชั่วโมง นาที และวินาที แทนที่จะกำหนดในหน่วยวินาทีเพียงหน่วยเดียว เช่น จุดเวลา ณ วินาทีที่ 360000 หมายถึงจุดเวลา ณ วันที่ 1 มกราคม ค.ศ. 0001 เวลา 10.00 น. เป็นต้น รูปแบบนี้จะเหมาะกับการแสดงผลหรือสื่อสารกับคน

1.3) รูปแบบจุดเวลาในภาษา CL เป็นการผสมผสานรูปแบบของจุดเวลาใน 2 แบบข้างต้น สามารถกำหนดจุดเวลาด้วยรูปแบบ ดังนี้ yyy : mm : dd | hh / minmin / ss yyyy โดยที่ h, min, s, d, m, y แทน ชั่วโมง, นาที, วินาที, วันที่, เดือน และปี ตามลำดับ โดยกำหนดให้ใช้จุดเริ่มอ้างอิง ณ ตำแหน่ง epoch หรือวันที่ 1 มกราคม ค.ศ. 1970 ดังที่กำหนดไว้ในภาษา Python เช่น 2002:08:28|11:00:00 หมายถึง ณ เวลา 11.00 น. ของวันที่ 28 สิงหาคม ค.ศ. 2002 ในรูปแบบจุดเวลาในชีวิตประจำวัน หรือหมายถึง จุดเวลา ณ วินาทีที่ 1030507200 เมื่อเทียบกับจุดเริ่มอ้างอิง (epoch) ในรูปแบบทางคณิตศาสตร์ เป็นต้น รูปแบบในภาษา CL ข้างต้นสอดคล้องกับรูปแบบจุดเวลาที่ใช้ในชีวิตประจำวัน แต่ในการประมวลผลจุดเวลาอินเตอร์พรีตเตอร์จะแปลงจุดเวลาในรูปแบบนี้ให้เป็นรูปแบบคณิตศาสตร์ก่อนแล้วจึงทำการคำนวณ จากนั้น ผลที่ได้ซึ่งอยู่ในรูปแบบทางคณิตศาสตร์จะถูกแปลงกลับให้อยู่ในรูปเดิมเพื่อการแสดงผล

2) รูปแบบช่วงเวลา (Time Interval) สามารถกำหนดเป็นรูปแบบต่างๆ ได้ดังนี้

2.1) รูปแบบทางคณิตศาสตร์ของช่วงเวลา ช่วงเวลา ประกอบด้วย จุดเวลาหลายๆ จุดที่เรียงต่อกันอย่างต่อเนื่องจากจุดเริ่มต้นจนถึงจุดสิ้นสุด ในทางคณิตศาสตร์จึงกำหนดให้ช่วงเวลาอยู่ในรูปแบบของเซตต่อเนื่องที่มีสมาชิกเป็นจุดเวลา คือ $[P_s, P_e]$ โดยที่ P_s คือ จุดเวลาเริ่มต้นซึ่งมีค่าน้อยกว่า P_e จุดเวลาสิ้นสุดหรือ $[P_s, P_s+d]$ โดย d คือระยะเวลาจาก P_s และ $d > 0$ หรือสามารถเขียนช่วงเวลาในรูปของเซตต่อเนื่อง $\{ t \mid P_s \leq t \leq P_e \}$ ได้ด้วย เช่น ช่วงเวลาตั้งแต่ เวลา 11.00 น. วันที่ 28 สิงหาคม ค.ศ. 2002 ถึงเวลา 12.00 น. วันที่ 28 สิงหาคม ค.ศ. 2002 หมายถึง [1030507200 , 1030510800] หรือ $\{ t \mid 1030507200 \leq t \leq 1030510800 \}$ ในรูปแบบทางคณิตศาสตร์

2.2) รูปแบบช่วงเวลาในภาษา CL ในภาษา CL รูปแบบช่วงเวลา ประกอบด้วยจุดเวลาเริ่มต้น และจุดเวลาสิ้นสุดของช่วงเวลา โดยแสดงไว้ภายในเครื่องหมาย […] ดังนี้ $[P_s, P_e]$ โดยที่ P_s และ P_e คือ ข้อมูลชนิดจุดเวลาแสดงจุดเวลาเริ่มต้นและสิ้นสุดตามลำดับ เช่น [2002:08:28|11:00:00, 2002:08:28|12:00:00] หมายถึงช่วงเวลาตั้งแต่ เวลา 11.00 – 12.00 น. ของวันที่ 28 สิงหาคม ค.ศ. 2002 สาเหตุที่ในภาษา CL เลือกใช้เครื่องหมาย [ ] ในกรณีที่โปรแกรมเมอร์กำหนดช่วงเวลา $[P_s, P_e]$ ที่ไม่เหมาะสมเข้าไปในโปรแกรมภาษา CL ด้วยการกำหนดให้ P_e เกิดก่อน P_s จะทำให้เกิดความผิดพลาดขึ้น แม้ว่าการกำหนดดังกล่าวจะถูกต้องที่ระดับไวยากรณ์ (Syntax) คือประกอบด้วยจุดเวลา 2 จุด แต่ไม่ถูกต้องที่ระดับความหมาย (Semantic) ที่กำหนดว่าจุดเวลาเริ่มต้น (P_s) ต้องเกิดขึ้นก่อนจุดเวลาสิ้นสุด (P_e)

3.4 เอ็กซเพรสชัน (Expression)

โอเปอเรเตอร์ที่ใช้ในภาษา CL แบ่งเป็น 4 ประเภทคือ โอเปอเรเตอร์ทางตรรกะ, โอเปอเรเตอร์ทางการเปรียบเทียบ, โอเปอเรเตอร์คำนวณเลขฐานสอง และโอเปเรเตอร์ทางคณิตศาสตร์ โดยส่วนหนึ่งนำมาจากภาษา Python ซึ่งได้กล่าวถึงไปแล้วในบทที่ 2 จึงไม่ขอกล่าวถึงในหัวข้อนี้อีก แต่จะอธิบายถึงโอเปอเรเตอร์ใหม่ที่เพิ่มเข้ามานอกเหนือจากภาษา Python ได้แก่ โอเปอเรเตอร์ที่ประมวลผลเกี่ยวกับเวลา โดยแบ่งได้เป็น โอเปอเรเตอร์ทางการเปรียบเทียบสำหรับเวลา (ซึ่งกล่าวในงานวิจัยการออกแบบและพัฒนาอินเตอร์พรีตเตอร์ภาษา CL) และโอเปอเรเตอร์ทางคณิตศาสตร์สำหรับเวลา ซึ่งมีรายละเอียดดังนี้

3.4.1 โอเปอเรเตอร์ทางคณิตศาสตร์สำหรับเวลา (Arithmetic operators of Time)

โอเปอเรเตอร์ทางคณิตศาสตร์สำหรับเวลาที่นำเสนอนี้ ผู้เขียนได้กำหนดขึ้นมาใหม่ทั้งหมดซึ่งเป็นรูปแบบโอเปอเรเตอร์อย่างง่ายที่พิจารณาจากการประมวลผลเวลาทั่วไปสำหรับกิจกรรมต่างๆ ในชีวิตประจำวัน ในกรณีที่เป็นข้อมูลเชิงปริมาณ โอเปอเรเตอร์ที่ใช้จะเป็นโอเปอเรเตอร์พื้นฐานทางคณิตศาสตร์ นั่นคือ บวก (+), ลบ (—) ในกรณีที่เป็นข้อมูลประเภทเซต โอเปอเรเตอร์ที่ใช้จะเป็นโอเปอเรเตอร์พื้นฐานสำหรับเซต ได้แก่ ยูเนียน (∪), อินเตอร์เซคชัน (∩), เซตผลต่าง (—) และคอมพลีเมนต์ (‘) และเป็นที่น่าสังเกตว่า ข้อมูลที่นำมาประมวลผลและผลลัพธ์จากการประมวลผลที่ได้ไม่จำเป็นต้องเป็นข้อมูลชนิดเดียวกัน ดังจะแสดงให้เห็นในลำดับต่อไป เพื่อความชัดเจนในการอธิบายในหัวข้อนี้ จึงกำหนดใช้สัญลักษณ์ดังต่อไปนี้P แทน ข้อมูลประเภทจุดเวลา, D แทน ข้อมูลประเภทระยะเวลา

โอเปอเรเตอร์ที่ใช้สำหรับจุดเวลานี้มีเพียง 2 ชนิด คือ โอเปอเรเตอร์บวก และ ลบ เท่านั้น จุดเวลาสามารถบวก หรือ ลบ กับข้อมูลประเภทระยะเวลาได้ อีกทั้งผลลัพธ์ที่ได้ก็สามารถเป็นได้ทั้งจุดเวลาและระยะเวลา ดังแสดงในตาราง 3.3

พีชคณิตที่ได้จะไม่ยุ่งยาก คือ ผู้ใช้สามารถหาจุดเวลาหนึ่งด้วยการลดหรือเพิ่มด้วยระยะเวลา และผู้ใช้สามารถหาระยะเวลาได้จากการลบกันของจุดเวลา 2 จุด

ตาราง 3.1 การคำนวณและโอเปอเรเตอร์สำหรับเวลา

โอเปอเรเตอร์	พีชคณิต	ตัวอย่าง
1.บวก (addtime)	$P_1 = P_2 + D$	[2002,8,30,11,0,0]= [2002,8,28,11,0,0] addtime [0,0,2,0,0,0]
2. ลบ (minustime)	$P_1 = P_2 - D$	[2002,8,28,11,0,0]= [2002,8,30,11,0,0] minustime [0,0,2,0,0,0]

3.4.2 ลำดับของโอเปอเรเตอร์ (Precedence)

ลำดับของตัวกระทำในภาษา CL ส่วนหนึ่งจะเหมือนกับลำดับของโอเปอเรเตอร์ในภาษา Python แต่เนื่องจากได้มีการเพิ่มเติมโอเปอเรเตอร์สำหรับเวลาเข้ามา ทำให้ลำดับของโอเปอเรเตอร์ในภาษา CL เป็นดังนี้ or, and, not, (<, <=, >, >=, ==, !=), |, ^, &, (<<, >>, <-, ->, >-, -<), (+, -), (*, /, %), **, (unary+, unary-), unary~

3.5 CL Statement

ประโยค (Statement) เป็นส่วนประกอบหลักของโปรแกรมภาษา CL โดยผู้เขียนจะใช้ประโยคเพื่อกำหนดการทำงานให้กับโปรแกรม ซึ่งในภาษา CL ได้มีการกำหนดการเขียนประโยค (statement) ไว้หลายประเภท ได้แก่ ประโยคสำหรับกำหนดค่าให้ตัวแปร, ประโยคคำสั่ง, ประโยคสำหรับควบคุมลำดับการทำงานในโปรแกรม และ ประโยคชุดคำสั่ง โดยอธิบายรายละเอียดได้ดังนี้

3.5.1 ประโยคสำหรับการกำหนดค่าให้ตัวแปร (Assignment Statement)

โปรแกรมเมอร์สามารถกำหนดค่าต่างๆ ให้กับตัวแปร ผ่านโอเปอเรเตอร์ "=" รูปแบบการใช้งานของประโยคประเภทนี้คือ <variable-identifier> = <expression> ตัวอย่างเช่น

โปรแกรม 3.2 การกำหนดค่าให้กับตัวแปร

```
anInt = 4
aString = "hello"
aFloat = -1.1111 * (5.0 **  2)
anotherString = "shop" + "ping"
aList = [123, "world", 5+4j]
```

ในภาษา CL ได้กำหนดให้สามารถนำโอเปอเรเตอร์ = เข้าไปใช้รวมกับโอเปอเรเตอร์ที่ใช้ในการคำนวณ (Arithmatic operation) ได้แก่ โอเปอเรเตอร์ +=, -=, *=, /=, **=, %= ซึ่งใช้ในกรณีที่มีการกำหนดค่าที่คำนวณได้ใหม่ให้กับตัวแปรเดิมที่มีอยู่ (reassign) ตัวอย่างเช่น a = a + 1 หรือสามารถเขียนได้อีกแบบว่า a += 1

3.5.2 ประโยคคำสั่ง

เป็นคำสั่งที่ใช้สั่งงานในโปรแกรม ซึ่งภาษา CL ณ ปัจจุบัน สามารถรองรับรูปแบบประโยคคำสั่งแบบ function() และ module.function() เท่านั้น และในอนาคตจะขยายให้ครอบคลุมคำสั่งแบบ class.function()

3.5.2.1 ประโยคคำสั่ง แบบ function()

เป็นประโยคคำสั่งที่ใช้เรียกฟังก์ชันให้ทำงานต่างๆ ตามที่กำหนดไว้ในฟังก์ชัน ฟังก์ชันในภาษา CL แบ่งเป็น 2 แบบ ได้แก่

1) Built-in function เป็นฟังก์ชันที่กำหนดไว้ภายในภาษา CL อยู่แล้ว เช่น ฟังก์ชัน abs,print เป็นต้น

2) Defined function เป็นฟังก์ชันที่ถูกกำหนดโดยโปรแกรมเมอร์รูปแบบของ

ประโยค คำสั่งเป็นดังนี้ <function-identifier>(<argument1>, <argumentN>) เช่น abs(-11) เป็นการเรียกฟังก์ชัน abs เพื่อหาค่าสัมบูรณ์ (absolute) ของอาร์กิวเมนต์ ซึ่งกำหนดให้เป็น–11เป็นต้น

3.5.2.2 ประโยคคำสั่ง แบบ module.function()

มีการทำงานเช่นเดียวกับแบบแรก แตกต่างกันเพียงฟังก์ชันที่ถูกเรียกนั้นไม่ได้ปรากฏในโปรแกรมปัจจุบัน หากแต่เป็นการเรียกฟังก์ชันจากโปรแกรมหรือโมดูลอื่นมาใช้แทนด้วยการใช้คำสั่ง import

3.5.3 ประโยคสำหรับควบคุมลำดับการทำงานในโปรแกรม (Control Statement)

ใน CL มีประโยคสำหรับควบคุมลำดับการทำงานของโปรแกรมทั้งหมด 4 รูปแบบ คือ รูปแบบประโยค if, ประโยค while, ประโยค for และประโยค repeat-until ซึ่งประโยคในสามแบบแรกใช้รูปแบบตามภาษา Python ทั้งหมด

1) ประโยค if (if statement) เป็นการควบคุมลำดับแบบเลือก (Selective) ของโปรแกรม หมายถึง ในโปรแกรมสามารถกำหนดให้มีการเลือกการทำงานอย่างใดอย่างหนึ่ง เมื่อเงื่อนไขที่กำหนดไว้ในโปรแกรมเป็นจริง รูปแบบของประโยค if เป็นดังนี้

โปรแกรม 3.3 ประโยค if (if statement)

```
if expression1 :
  expr1_true_statement
      [ elif expression2 :
  expr2_true_statement
      :
  elif expressionN :
     exprN_true_statement ]
       [ else :
  none_of_the_above_statement ]
```

ในการกำหนดประโยค if นั้น โปรแกรมเมอร์สามารถสร้างทางเลือกของการทำงานของโปรแกรมได้มากกว่า 2 ทางเลือก (เลือกที่จะทำประโยคในประโยค if หรือไม่ทำ) ด้วยประโยค elif และ else ซึ่งประโยค elif อาจมีได้มากกว่า 1 ประโยค ต่างจากประโยค else ที่มีได้เพียง 1 ประโยค แต่ถ้าต้องการสร้างเส้นทางการทำงานเพียง 2 เส้นทาง ก็ไม่จำเป็นต้องมีประโยค elif และ else

2) ประโยค while (while statement) เป็นการควบคุมลำดับแบบวนรอบ (Iterative) ของโปรแกรม ที่มีการตรวจสอบเงื่อนไข (expression) ก่อนประมวลผลประโยค (statement) ทุกครั้ง หรือเรียกว่า test-before loop รูปแบบของประโยค while เป็นดังนี้

โปรแกรม 3.4 ประโยค while (while statement)

```
while expression :
    statement
```

ประโยค while เป็นประโยคที่ใช้กำหนดให้มีการทำงานประโยค (statement) ซ้ำได้เมื่อเงื่อนไข (expression) ที่กำหนดไว้ในโปรแกรมเป็นจริง และจะหยุดการทำซ้ำเมื่อ เงื่อนไขเป็นเท็จ

3) ประโยค for (for statement) เป็นการควบคุมลำดับแบบวนรอบ (Iterative) ของโปรแกรม ที่มีการใช้อินเด็กซ์กำหนดจำนวนของการวนรอบ หรือเรียกว่า indexed loop รูปแบบของประโยค for เป็นดังนี้

โปรแกรม 3.5 ประโยค for (for statement)

```
for variable in sequence :
    statement
```

ประโยค for เป็นประโยคที่ใช้กำหนดให้มีการทำงานประโยค (statement) ซ้ำ ที่มีจำนวนการวนรอบที่แน่นอน

4) ประโยค repeat-until (repeat-until statement) เป็นการควบคุมลำดับแบบวนรอบ (Iterative) ของโปรแกรม ที่มีการตรวจสอบเงื่อนไข (expression) หลังการประมวลผลประโยค (statement) หรือเรียกว่า test-after loop รูปแบบของประโยค repeat-until เป็นดังนี้

โปรแกรม 3.6 ประโยค repeat-until (repeat-until statement)

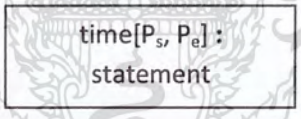
```
repeat :
    statement
until expression
```

ประโยค repeat-until เป็นประโยคที่มีการประมวลผลประโยค (statement) ก่อนที่จะตรวจสอบเงื่อนไข (expression) ที่กำหนด หากเงื่อนไขเป็นเท็จ จะทำการประมวลผลประโยคซ้ำ แต่ถ้าเงื่อนไขเป็นจริง ก็จะหยุดการวนรอบ นั่นคือ การ ประมวลผลประโยคจะดำเนินไปเรื่อยๆ จนกว่าเงื่อนไขจะเป็นจริง

3.5.4 คำสั่งการทำงานในรูป ช่วงเวลา: การทำงาน (Time interval: Action)

ในภาษา CL ผู้เขียนได้มีการเพิ่มความสามารถในการกำหนดช่วงเวลาให้กับการทำงาน ซึ่งยังไม่ปรากฏในภาษาคอมพิวเตอร์อื่นๆ จุดประสงค์เพื่อเพิ่มความยืดหยุ่นในการเขียนโปรแกรม ทั้งยังสนับสนุนให้โปรแกรมเมอร์พัฒนาโปรแกรมได้อย่างสอดคล้องกับการทำงานในชีวิตประจำวัน ทำให้โปรแกรมที่เขียนด้วยภาษา CL สามารถเขียนและเข้าใจได้ง่าย แม้ผู้อ่านไม่มีความรู้ด้านการเขียนโปรแกรมคอมพิวเตอร์มากนัก

จะสังเกตได้ว่า การทำงานหรือการทำกิจกรรมทุกอย่างของคนนั้น ล้วนแล้วแต่เกี่ยวข้องกับเวลาทั้งสิ้น ไม่ว่าจะเป็นจุดเวลาเริ่มต้นและจุดเวลาสิ้นสุดของการทำงานนั้น ๆ หรือกล่าวได้ว่า การทำงานคำสั่งย่อมทำในช่วงเวลาเสมอ ในภาษา CL จึงมีการกำหนดรูปแบบคำสั่งการทำงานหรือรูปแบบประโยคที่การทำงานกำหนดอยู่ภายใต้ช่วงเวลา คือมีการกำหนดช่วงเวลาทำงานให้กับทุกประโยค ดังนี้



รูป 3.3 รูปแบบการกำหนดจุดเวลา

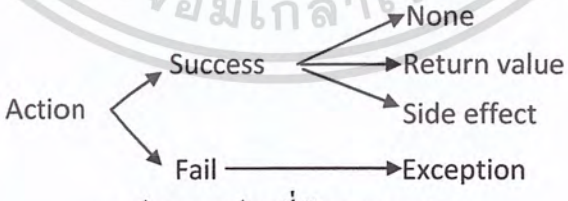
หมายถึง การทำ statement จะเริ่มต้น ณ จุดเวลา P_s และสิ้นสุด ณ จุดเวลา P_e โดย statement หมายถึงประโยคแบบต่างๆ ที่ได้อธิบายไปแล้ว และ [P_s , P_e] เป็นข้อมูลชนิดช่วงเวลาในภาษา CL จะใช้สัญลักษณ์ < > แทน [ ] ตัวอย่างเช่น <11:00:00|21/10/2010, 12:00:00|21/10/2010> แทนช่วงเวลาตั้งแต่เวลา 11.00 น. ของวันที่ 21 ตุลาคม 2010 ถึงเวลา 12.00 น. วันเดียวกัน นอกจากนี้ จุดเวลาเริ่มต้นและจุดเวลาสิ้นสุดสามารถใช้เป็น expression ใดๆ ที่ให้ค่าจุดเวลาออกมาได้ นอกเหนือจากการกำหนดโดยใช้ข้อมูลชนิดจุดเวลาโดยตรง เช่น ถ้ากำหนดให้ x = 11:00:00|21/10/2010 แล้ว ช่วงเวลาที่อธิบายไว้ข้างต้นสามารถเขียนได้เป็น < x, x+1h > ซึ่งแสดงถึงช่วงเวลาที่มีจุดเวลาเริ่มต้นและสิ้นสุดห่างกันเป็นระยะเวลา 1 ชั่วโมง นั่นเอง

ประเภทของคำสั่งการทำงาน คำสั่งการทำงานมีรูปแบบเดียวดังนั้นคำสั่งการทำงานแบบระบุเวลาเป็นคำสั่งที่ผู้เขียนโปรแกรมต้องการกำหนดเวลาที่แน่นอนในการเริ่มต้นและสิ้นสุดการทำคำสั่งนั้น เช่น ต้องการให้คอมพิวเตอร์เริ่มดาวน์โหลดไฟล์ ณ เวลา 10.00 น. และสิ้นสุดการดาวน์โหลด ณ เวลา 11.00 น. เป็นต้น

ตาราง 3.2 คำสั่งการทำงาน

ประเภทคำสั่ง	รูปแบบทั่วไป	อธิบาย	ตัวอย่างใน CL
แบบระบุเวลา	time[P_s, P_e] : statement	เริ่มทำงาน ณ เวลา P_s และสิ้นสุดการทำงาน ณ เวลา P_e	[2003:07:10|10:00:00, 2003:07:10|10:10:00]: Download('p.mp3')

การทำ Action เมื่อทำการประมวลผลคำสั่งนั้นแล้ว ผลที่เกิดขึ้นจะเป็นได้ใน 2 แบบ คือ ผลการทำ Action ที่ทำได้สำเร็จ (Success) ซึ่งหมายถึง การประมวลผลตามคำสั่งการทำงานได้อย่างครบถ้วน ซึ่งอาจมีการส่งผลกลับหรือไม่ก็ได้ คือ ถ้าส่งผลกลับจะเป็นลักษณะที่ผลที่ได้เป็นค่าข้อมูล (Value) หรือเป็นผลกระทบ (Side effect) เช่นการแสดงผลหน้าจอ เป็นต้น ส่วนถ้าไม่ส่งผลอะไรกลับให้ถือว่าสิ่งที่ส่งกลับคือ None ส่วนผลการทำ Action ที่ประสบความล้มเหลว (Failed) หมายถึง การประมวลผลตามคำสั่งนั้นได้ไม่สมบูรณ์ ซึ่งไม่ว่าจะด้วยสาเหตุใดก็ตาม ระบบจะจัดการกับผลลัพธ์นี้ในลักษณะของ Exception ซึ่งวิธีการจัดการกับ Exception กำหนดได้เองโดยผู้เขียนโปรแกรม ผลที่เกิดจากการทำ Action สามารถสรุปได้ดังรูป 3.4



รูป 3.4 สรุปผลที่เกิดจาก Action

ช่วงเวลารวมของชุดคำสั่งดังที่ได้อธิบายไปข้างต้นว่า โปรแกรมเมอร์สามารถกำหนดเวลาการทำประโยคใดๆ ได้ ประโยคชุดคำสั่งก็เป็นประโยคในรูปแบบหนึ่งของภาษา CL ทำให้เราสามารถกำหนดช่วงเวลาเริ่มต้นและสิ้นสุดของการทำงานของทั้งชุดคำสั่งได้ในรูปแบบ [P_s, P_e]: compound statement โดยหมายถึงว่า P_s คือจุดเวลาเริ่มต้นที่ชุดคำสั่งเริ่มทำงาน และ P_e คือ

จุดเวลาที่ชุดคำสั่งสิ้นสุดการทำงานชุดคำสั่ง นั่นหมายถึง ต้องทำงานประโยคทั้งหมดภายในชุดคำสั่งให้เสร็จภายในเวลา P_e ในกรณีที่สามารถทำงานประโยคชุดคำสั่งได้เสร็จก่อนเวลา P_e โปรแกรมก็ต้องรอจนกว่าถึงเวลาที่กำหนดจึงจะสามารถประมวลผลประโยคถัดไปได้ ในทางกลับกัน หากไม่สามารถประมวลผลประโยคทั้งหมดในประโยคชุดคำสั่งได้ภายในเวลา P_e จะเกิดความผิดพลาดและไม่ทำการประมวลผลประโยคใดๆ ต่อ

3.6 การโปรแกรมด้วยภาษา CL

ในหัวข้อนี้ จะกล่าวถึงรายละเอียดของการเขียนโปรแกรมด้วยภาษา CL ในส่วนต่างๆ เริ่มจากวิธีการเขียนโปรแกรม, รูปแบบการเขียนโปรแกรม, คำที่ใช้ในโปรแกรม จนถึงรายละเอียดของความหมายและวิธีการใช้ประโยค, การนิยามฟังก์ชัน และการนิยามคลาส ตามลำดับ

3.6.1 วิธีการเขียนโปรแกรมภาษา CL

กลุ่มอักขระ (Character set) ในภาษา CL ประกอบด้วย

1) ตัวอักษรภาษาอังกฤษ (Alphabetic characters) ทั้งขนาดเล็กและใหญ่รวม 52 ตัว ได้แก่ a, b, …, z, A, B, …, Z
2) ตัวเลข (numeric digits) 10 ตัว ได้แก่ 0, 1… 9
3) อักขระพิเศษ (special characters) ได้แก่ _ & | ^ > < = ~ + - * / % ( ) ! ; , : { } []

3.6.2 รูปแบบการเขียนโปรแกรม

รูปแบบการเขียนโปรแกรมในภาษา CL นั้น จะคล้ายคลึงกับรูปแบบการเขียนโปรแกรมของภาษา Python ซึ่งมีการใช้ย่อหน้า, การขึ้นบรรทัดใหม่ และการเขียนคอมเมนต์ ดังนี้

1) การใช้ย่อหน้า (indenting) ในภาษา CL ได้ทำการกำหนดสโคปของซอร์สโค้ดโดยใช้การย่อหน้าแทนสัญลักษณ์เช่นเดียวกับภาษา Python ซึ่งต่างจากภาษาอื่นๆ เช่น ภาษา C และ Java ใช้วงเล็บปีกกา และภาษา Pascal ใช้ begin end เป็นตัวกำหนดสโคปของซอร์สโค้ด
2) การขึ้นบรรทัดใหม่ ใช้สำหรับบอกตำแหน่งสิ้นสุดของการเขียนประโยค (statement) แต่ละประโยคเหมือนกับภาษา Python แทนที่จะลงท้ายเครื่องหมาย ";" ดังที่พบทั่วไปในภาษา Pascal, Java
3) คอมเมนต์ (Comment) ใช้สำหรับอธิบายโปรแกรมเพิ่มเติม โดยไม่นำไปประมวลผลรวมกับประโยคต่างๆ ในภาษา CL กำหนดให้สามารถเขียนคอมเมนต์ได้ ภายใต้เครื่อง "#" เช่นเดียวกับในภาษา Python

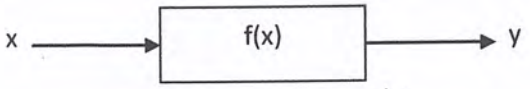
3.6.3 คำที่ใช้ในภาษา CL

คำที่ใช้ในโปรแกรม อาจจะเป็นคำที่กำหนดไว้ก่อนแล้วอย่างคีย์เวิร์ด (Language Keywords) ของภาษา หรือเป็นคำที่โปรแกรมเมอร์เป็นคนสร้างขึ้น (Identifier)

1) Identifierโปรแกรมเมอร์จะกำหนด identifier ให้กับตัวแปร(variable), ฟังก์ชัน (function) และตัวแปรแกรมเองได้ ชื่อ หรือ identifier อาจกำหนดด้วยอักษรภาษาอังกฤษทั้งหมด หรืออาจมีตัวเลขหรือเครื่องหมาย "_" เข้าไปด้วยก็ได้ แต่จะต้องเริ่มต้นด้วยอักษรภาษาอังกฤษเท่านั้น เช่น name, gender, student_name, name1 เป็นต้น สำหรับการกำหนดชื่อ หรือ identifier นั้นก็ควรมีการสื่อความหมายได้ดี เช่น การตั้งชื่อตัวแปรที่เก็บค่าที่บอกอายุว่า age เป็นต้น
2) คำสงวน (Reserved words) คำสงวนในภาษา CL แบ่งเป็น คีย์เวิร์ด (Keywords) และบิวท์อินฟังก์ชั่น(Built-in function)
   2.1) คีย์เวิร์ด (Keywords) คีย์เวิร์ดใน CL ส่วนหนึ่งนำมาจากภาษา Python นอกจากนี้ยังได้เพิ่มคีย์เวิร์ดที่ใช้สำหรับเทียบเวลา และกำหนดชุดคำสั่งเข้าไว้ด้วย คีย์เวิร์ดในภาษา CL มีดังนี้ 'as' , 'from' , 'def' , 'return' , 'if' , 'else' , 'elif' , 'true', 'false', 'and', 'or', 'time', 'none' , 'for' , 'in' , 'set' , 'class' , 'while' , 'import' , '__name__' , '__code__' , '__return__' , 'sequential' , 'parallel' , 'unordered' , 'alternative' , 'print' , 'pass' , 'break' , 'addtime' , 'minustime'
   2.2) บิวท์อินฟังก์ชั่น(Built-in function) เช่นเดียวกับคีย์เวิร์ด คือได้มีการเพิ่มเติมบิวท์อินฟังก์ชันที่ใช้สำหรับการสื่อสารนอกเหนือจากบิวท์อินฟังก์ชันใน Python บิวท์อินฟังก์ชันใน CL มีดังนี้ 'abs', 'divmod', 'round', 'pow', 'real', 'imag', 'len', 'min', 'max', 'append', 'extend', 'count', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort', 'clear', 'has_key', 'keys', 'update', 'values', 'open', 'read', 'readline', 'readlines', 'write', 'writelines', 'close', 'tell', 'query', 'request', 'delete', 'please_tell', 'please_query', 'please_request', 'please_delete', 'yes', 'no', 'deny', 'dontknow', 'ignore', 'find', 'send', 'assert', 'demo', 'agent', 'tell_msg', 'query_msg', 'request_msg', 'delete_msg', 'please_tell_msg', 'please_query_msg', 'please_request_msg', 'please_delete_msg' , 'show', 'Email' , 'SMS' , 'PopUp' , 'dirr'

3.6.4 การนิยามฟังก์ชันในภาษา CL

ฟังก์ชัน (Function) ฟังก์ชันเป็นการทำงานที่มีการคืนค่าผลลัพธ์ออกมา 1 ค่า ซึ่งเขียนอยู่ในรูปแบบทางคณิตศาสตร์ได้ว่า y = f(x) เมื่อ f คือ ชื่อฟังก์ชัน, x คือ อินพุทของฟังก์ชัน และ y คือผลลัพธ์ที่ได้จากการทำฟังก์ชัน ฟังก์ชันสามารถแสดงเป็นโมเดล ได้ดังนี้



รูป 3.5 การทำงานของฟังก์ชัน

ค่าหรือผลลัพธ์ต่างๆ ที่เกิดขึ้นจากการกำหนดของฟังก์ชัน เรียกโดยรวมว่า พารามิเตอร์ (Parameters) แบ่งได้เป็น 3 ประเภท ดังนี้

1) Input parameters หมายถึง ค่าที่ใช้สำหรับเป็นอินพุทของฟังก์ชัน
2) Output parameters หมายถึง ค่าที่ได้จากการทำฟังก์ชัน หรือผลลัพธ์นั่นเอง
3) Input/output parameters เป็นส่วนที่เป็นทั้งอินพุทของฟังก์ชันและเป็นผลลัพธ์ของการทำฟังก์ชันด้วย ตัวอย่างเช่น ให้ f(x) คือ ฟังก์ชันหาค่าสแควร์รูท (Square root) โดยที่ $f(x) = x^2$ จะได้ว่า f(2) = 4 นั่นคือ x = 2 และ y = 4 เป็นต้น

การนิยามฟังก์ชันโปรแกรมเมอร์สามารถกำหนดการทำงานในโปรแกรมภาษา CL ได้หลายวิธี หนึ่งในนั้นคือการเรียกฟังก์ชันที่กำหนดไว้ในและนอกโปรแกรมด้วยรูปแบบ function() และ module.function() ตามลำดับซึ่งฟังก์ชันที่ถูกเรียกใช้นั้นอาจเป็นฟังก์ชันที่เกิดจากการสร้างขึ้นของโปรแกรมเมอร์ (define function) หรือเป็นฟังก์ชันที่กำหนดไว้ในภาษา CL อยู่แล้ว (Built-in function) ก็ได้ ในหัวข้อนี้จึงขอกล่าวถึงการกำหนดหรือการสร้างนิยามฟังก์ชันในภาษา CL ซึ่งมีรูปแบบดังนี้ def function_name($param_1$, $param_2$, ...) : โดยกำหนดให้ function_name เป็นชื่อของฟังก์ชัน ซึ่งกำหนดได้ตามรูปแบบของ identifier param เป็นอินพุทพารามิเตอร์ของฟังก์ชัน ซึ่งสามารถกำหนดด้วยข้อมูลชนิดต่างๆ ในภาษา CL definition จะประกอบด้วยประโยครูปแบบต่างๆ ซึ่งเป็นส่วนที่ระบุการทำงานทั้งหมดของฟังก์ชัน

ในกรณีที่ Definition เป็น <P_s,P_e> : statement ซึ่งหมายถึงว่า function นี้เริ่มทำงาน ณ เวลา P_s และสิ้นสุดการทำงานที่เวลา P_e ดังนั้น การเรียกใช้งาน function นี้จะเป็นรูปแบบ <P_s,P_e> : function เช่นกัน โดยผู้เขียนโปรแกรมจะต้องกำหนด P_s และ P_e และเรียกใช้ function ให้สอดคล้องกับ P_s และ P_e ที่กำหนดใน definition

ขอบเขตของตัวแปร ในภาษา CL ได้มีการกำหนดขอบเขตของตัวแปร (Variable scope) ไว้ 2 ขอบเขต ได้แก่ ขอบเขตแบบหลัก (Global) และขอบเขตย่อย (Local) โดยตัวแปร หรือ

identifier ที่ถูกนิยามไว้ในขอบเขตดังกล่าว จะเรียกว่า ตัวแปรหลัก (Global variable) และ ตัวแปรย่อย (Local variable) ตามลำดับ

3.6.5 การนิยามคลาสในภาษา CL

คลาส (Class) คลาสคือต้นแบบที่กำหนดคุณสมบัติและพฤติกรรมการทำงานของอ็อบเจกต์ที่ถูกสร้างมาจากคลาสนั้น องค์ประกอบของคลาสมีสองส่วนหลักได้แก่

1) ข้อมูล (data) หรือ คุณสมบัติ (property) สำหรับเก็บข้อมูลในการทำงานหรือบอกสถานะปัจจุบันของอ็อบเจกต์ ตัวอย่างเช่น คลาสชนิดบัญชีธนาคารสามารถมีข้อมูลคือ เลขที่บัญชี, ชื่อเจ้าของบัญชี, และยอดเงินปัจจุบัน
2) พฤติกรรมการทำงาน (method หรือ function) เป็นส่วนรับคำสั่งและทำงานตามคำสั่งนั้น เช่นคลาสบัญชีธนาคารจากตัวอย่างข้างบนอาจจะมีฟังก์ชันสำหรับฝากและถอนเงินจากบัญชี เป็นต้น

นอกจากนี้คลาสยังสามารถมีฟังก์ชันพิเศษเรียกว่า Constructor ซึ่งจะถูกเรียกใช้เวลาที่กำลังสร้างอ็อบเจคจากคลาสเพื่อตั้งค่าเริ่มต้นให้กับข้อมูลและคุณสมบัติของอ็อบเจกต์นั้น

ความสัมพันธ์ระหว่างคลาสหลักกับคลาสย่อยเรียกว่า inheritance เช่นคลาสรถยนต์เป็นคลาสหลักสามารถวิ่งได้ มีรถแข่งและรถบรรทุกเป็นคลาสย่อย ทั้งรถแข่งและรถบรรทุกเป็นรถยนต์ชนิดหนึ่งจึงสามารถวิ่งได้เช่นกัน ความสัมพันธ์แบบคลาสหลักกับคลาสย่อยนี้ทำให้เกิดหลักสำคัญอีกอย่างคือ Polymorphism คือทั้งรถแข่งและรถบรรทุกสามารถเลี้ยวได้โดยการบังคับพวงมาลัย แต่รถทั้งสองชนิดจะมีการตอบสนองที่ต่างกัน คือรถแข่งจะเลี้ยวได้ไวกว่าในขณะที่รถบรรทุกเลี้ยวอย่างรวดเร็วไม่ได้เพราะจะทำให้พลิกคว่ำ

การนิยามคลาสโปรแกรมเมอร์สามารถกำหนดการทำงานในโปรแกรมภาษา CL ได้หลายวิธี หนึ่งในนั้นคือการสร้างคลาส ในหัวข้อนี้จึงขอกล่าวถึงการกำหนดหรือการสร้างนิยามคลาสในภาษา CL ซึ่งมีรูปแบบดังนี้

โปรแกรม 3.7 การสร้างนิยามคลาสในภาษา CL

```
class  class_name(parentclass1, parentclass2, …) :
     definition
```

โดยกำหนดให้

1) class_name เป็นชื่อของคลาส ซึ่งกำหนดได้ตามรูปแบบของ identifier

2) parentclass คือ class_name ที่ได้รับการสืบทอดมาจาก parentclass นี้ ถ้าไม่มี subclass ถือว่าเปงคลาสหลักไม่ได้รับการสืบทอดจากคลาสใดๆ

3) definition จะประกอบด้วยประโยครูปแบบต่างๆ ซึ่งเป็นส่วนที่ระบุการทำงานทั้งหมดของคลาส

เมื่อทำการเรียกใช้คลาสจำเป็นต้องสร้างอ็อบเจกต์ของคลาสนั้นเพื่อเรียกใช้medthod หรือ data ภายในคลาสนั้นๆเช่น

โปรแกรม 3.8 การสร้างคลาสและสร้างอ็อบเจกต์ของคลาส

```
class  A :
      a=1
class B(A):
      b=2
x=A()
y=B()
x.a  ---------------->  1
y.a  ---------------->  1
y.b  ---------------->  2
```

จะเห็นได้ว่า คลาส A เป็นคลาสหลัก คลาส B สืบทอดมาจากคลาส A ดังนั้น เมื่อสร้างอ็อบเจ็คของคลาส A จะเห็นได้ว่าจะเรียกใช้ data หรือ method ได้เฉพาะภายในคลาสเท่านั้น เนื่องจากไม่ได้สือทอดมาจากคลาสใดๆ แต่เมื่อสร้างอ็อบเจ็กต์ของคลาส B จะสามารถเรียกใช้ data หรือ method ได้ทุกคลาสที่ได้รับการสืบทอดมาและตัวของมันเอง

ขอบเขตของตัวแปรในภาษา CL ได้มีการกำหนดขอบเขตของตัวแปร (Variable scope) ไว้ 2 ขอบเขต ได้แก่ ขอบเขตแบบหลัก (Global) และขอบเขตย่อย (Local) โดยตัวแปร หรือ identifier ที่ถูกนิยามไว้ในขอบเขตดังกล่าว จะเรียกว่า ตัวแปรหลัก (Global variable) และ ตัวแปรย่อย (Local variable) ตามลำดับ

บทที่ 4
โครงสร้างข้อมูลสำหรับเก็บโปรแกรม CL

ในบทนี้เราขอนำเสนอโครงสร้างข้อมูลสำหรับจัดเก็บลำดับการทำงานของโปรแกรม CL โดยจะอธิบายการออกแบบโครงสร้างข้อมูลจากง่ายไปหายาก เราขอเริ่มจากประโยค CL ที่มีกำหนดเวลาแน่นอนก่อนในหัวข้อที่ 4.1 จากนั้นเราจะอธิบายถึงโครงสร้างข้อมูลสำหรับเก็บประโยค CL ที่มีกำหนดเวลาแน่นอนดังกล่าว ในหัวข้อที่ 4.2 ตามมาด้วย อัลกอริธึมบนโครงสร้างข้อมูล เช่น การเพิ่มจุดเวลาหรือช่วงเวลา การลบจุดเวลาหรือช่วงเวลา การค้นหาจุดเวลาหรือช่วงเวลา ในหัวข้อที่ 4.3 จากนั้นเราจะกล่าวถึงคุณสมบัติของโครงสร้างข้อมูลในหัวข้อที่ 4.4 ต่อมาเราจะเริ่มพิจารณาการบันทึกประโยค CL ในโครงสร้างข้อมูลในหัวข้อที่ 4.5 จากนั้นเราจะกล่าวถึงประโยค CL ที่มีกำหนดเวลาเปลี่ยนแปลงในหัวข้อที่ 4.6 จากนั้นเราจะอธิบายถึงโครงสร้างข้อมูลสำหรับเก็บประโยค CL ที่มีกำหนดเวลาเปลี่ยนแปลงในหัวข้อที่ 4.7

4.1 ประโยค CL ที่มีกำหนดเวลาแน่นอน

ประโยค CL ที่มีกำหนดเวลาเริ่มต้นและสิ้นสุดเป็นค่าแน่นอน ถูกใช้สำหรับโปรแกรมการทำงานที่เรามักพบเห็นได้ในชีวิตประจำวัน

ตัวอย่าง สมมติว่าเราต้องการสั่งงานให้คอมพิวเตอร์เริ่มเล่นเพลงในช่วงเวลา 20 ถึง 21 น. และปิดเครื่องตอน 2 น. ถึง 3 น. ของวันใหม่ ดังนั้นโปแกรมภาษา CL อาจเขียนได้ดังนี้

โปรแกรม 4.1 Play Music

```
time[2010:6:23|20:0:0, 2010:6:23|21:0:0]:
  play_music("song.mp3")
time[2010:6:24|2:0:0, 2010:6:24|3:0:0]:
  shutdown()
```

ประโยค CL ที่มีกำหนดเวลาแน่นอน ถือเป็นลักษณะการโปรแกรม CL ที่ง่ายที่สุด ดังนั้นเราจะขอเริ่มออกแบบโครงสร้างข้อมูลจากประโยครูปแบบนี้

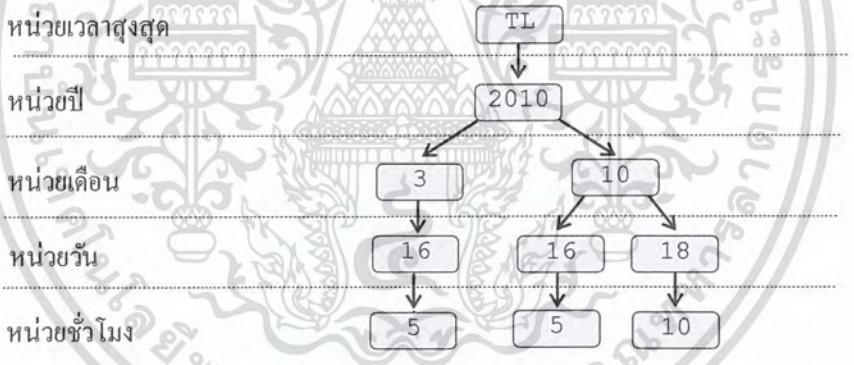
4.2 โครงสร้างข้อมูลสำหรับเก็บประโยค CL ที่มีกำหนดเวลาแน่นอน

จุดเวลาถูกระบุโดยลำดับการนับหน่วยเวลาหลายๆหน่วย โดยนำลำดับหน่วยต่างๆ เหล่านั้นมาเรียงจากใหญ่ไปหาเล็ก เช่น จุดเวลา 2010:3:16 | 5:30:10 หมายถึง ปีที่ 2010 เดือนที่ 3 วันที่ 16

ชั่วโมงที่ 5 นาทีที่ 30 วินาทีที่ 10 ในที่นี้เราได้ใช้โครงสร้างข้อมูลแบบ Tree [4] ในการเก็บจุดเวลาต่างๆ โดยวิธีดังนี้

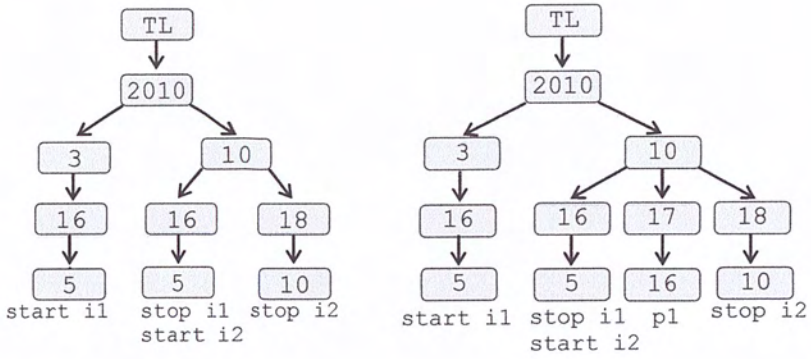
1) ให้ระดับของโหนดแทนหน่วยเวลา โดยระดับบนสุดเป็นหน่วยสูงสุด ระดับรองลงมาเป็นหน่วยย่อยที่แบ่งย่อยจากหน่วยก่อนหน้ามันลดหลั่นกันไปตามลำดับ
2) โหนดแต่ละโหนดแทนลำดับของหน่วยที่จะใช้อ้างในจุดเวลา ยกเว้น root ที่อยู่ที่ระดับหน่วยสูงสุด มีเพียงลำดับเดียว จึงขอแทนด้วย TL เพื่อให้สื่อถึงหน่วยเวลาที่ใหญ่ที่สุด คือ Time Line ซึ่งครอบคลุมเวลาทั้งหมด ทั้งอดีตปัจจุบันและอนาคตกลุ่มโหนดที่ระดับเดียวกันจะต้องเรียงลำดับ โดยโหนดค่าน้อยอยู่ทางซ้าย และโหนดค่ามากอยู่ทางขวา
3) การไล่เรียงโหนดจากระดับบนมายังโหนดลูกตามลำดับหน่วยที่ระบุในจุดเวลานั้นๆ เป็นการระบุถึงจุดเวลาที่สมบูรณ์ที่ถูกบันทึกไว้ในโครงสร้างข้อมูล

สรุปได้ว่าจุดเวลาแต่ละจุดจะแทนด้วย path จาก root ไปถึง leaf ดังนั้นเพื่อความสะดวกเราจะให้ leaf มีความหมายแทน path ที่มาสิ้นสุดที่ leaf นั้นทำให้ leaf มีความหมายแทนจุดเวลาที่ระบุโดย path ที่มาสิ้นสุดที่ leaf นั้นสมมติว่าเราจะเก็บจุดเวลา 3 จุด 2010/3/16 เวลา 5 น., 2010/10/16 เวลา 5 น., 2010/10/18 เวลา 10 น. ในโครงสร้างข้อมูล Tree ที่ได้จะเป็นดังรูป



รูป 4.1 Tree ที่เก็บ 3 จุดเวลา

สำหรับการบันทึกช่วงเวลาใน Tree นั้น ทำได้โดยบันทึกทั้งจุดเวลาเริ่มต้นและจุดเวลาสิ้นสุดลงใน Tree พร้อมระบุด้วยว่าจุดใดเป็นจุดเริ่มต้นและจุดใดเป็นจุดสิ้นสุดของช่วงเวลานั้น ดังตัวอย่าง การบันทึกช่วงเวลา i1=[2010/3/16 เวลา 5 น., 2010/10/16 เวลา 5 น.] และ i2=[2010/10/16 เวลา 5 น., 2010/10/18 เวลา 10 น.] ใน Tree จะได้ Tree ดังรูป (ก)



ก) ข)

รูป 4.2 Tree เก็บจุดเวลาและช่วงเวลา

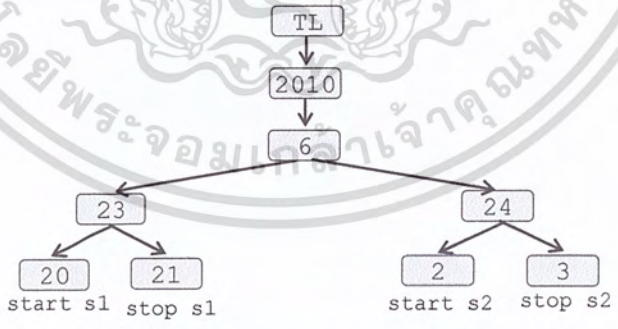
ก) เก็บช่วงเวลา

ข) เก็บจุดและช่วงเวลา

รูป (ข) เป็นตัวอย่าง Tree ที่เก็บทั้งจุดและช่วงเวลาไว้ด้วยกัน คือ จุดเวลา p1 ที่ 17/10/2010 เวลา 16 น. และช่วงเวลา i1 = [2010/3/16 เวลา 5 น., 2010/10/16 เวลา 5 น.] และ i2 = [2010/10/16 เวลา 5 น., 2010/10/18 เวลา 10 น.]

ดังนั้นประโยค CL ที่อยู่ในรูปแบบ time[p_s, p_e]:statement เมื่อถูกเก็บไว้ใน Tree จะมีลักษณะเช่นเดียวกับการเก็บบันทึกช่วงเวลา โดยเราบันทึกข้อมูลว่า start <statement> และ stops <statement> ไว้ที่จุดเวลาเริ่มต้นและสิ้นสุด ตามลำดับ

ตัวอย่างโปรแกรม CL ในตัวอย่างก่อนหน้านี้ ถูกเก็บใน Tree ดังรูป



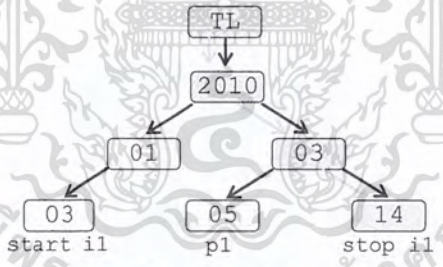
รูป 4.3 การจัดเก็บโปรแกรม CL ใน Tree

4.3 อัลกอริทึมสำหรับโครงสร้างข้อมูล

4.3.1 การค้นหาจุดเวลาที่เก็บในโครงสร้างข้อมูลโดยใช้ค่าจุดเวลาเป็นคีย์

การค้นหานี้ อาจเป็นได้ใน 2 ลักษณะคือ 1) การนำเอาค่าจุดเวลาไปค้นหาว่ามีจุดเวลานี้เก็บไว้ในโครงสร้างข้อมูลหรือไม่ 2) การนำเอาค่าจุดเวลาไปค้นหาว่าจุดเวลานี้เป็นจุดเริ่มต้นหรือจุดสิ้นสุดของช่วงเวลาใดที่เก็บในโครงสร้างข้อมูลหรือไม่ ต่อไปเป็นการนำเสนออัลกอริธึมเพื่อการค้นหาดังกล่าวอัลกอริธึมนี้รับจุดเวลาที่ระบุด้วยหน่วยเวลาหลายหน่วยเรียงลำดับตามหน่วยใหญ่หน่วยเล็ก การค้นหาจุดเวลานี้เริ่มจากนำค่าตัวเลขของหน่วยปี (หน่วยใหญ่สุด) เปรียบเทียบกับโหนด ลูกของ root (TL) ว่าพบค่าหน่วยปีที่โหนดลูกหรือไม่ ถ้าไม่พบก็แสดงว่าไม่มีจุดเวลานั้นเก็บใน Tree และยุติการค้นหา แต่ถ้าพบก็ให้นำค่าตัวเลขหน่วยถัดไปของจุดเวลาไปเปรียบเทียบกับโหนดลูกของโหนดที่ค้นพบ และทำซ้ำเดิมอีกเรื่อยๆ จนกระทั่งหมดหน่วยเวลาที่ใช้ค้นหา แล้วให้ผลลัพธ์เป็นชื่อจุดเวลาที่บันทึกไว้ที่โหนดหรือ leaf สุดท้ายที่ค้นพบค่าหน่วยเวลาสุดท้ายออกมา

ต่อไปเป็นรูปแสดงตัวอย่างการค้นหาจุดเวลาตามอัลกอริธึมนี้เมื่อต้องการค้นหาจุดเวลาด้วยจุดเวลาปี 2010 เดือน 3 วันที่ 14 เริ่มด้วยการเปรียบเทียบหน่วยปีที่ 2010 กับโหนดลูกของ root ซึ่งพบว่ามีโหนดปีที่ 2010 อยู่จึงนำเอาค่าหน่วยเดือนมาเปรียบเทียบกับโหนดลูกของโหนด 2010 พบว่ามีโหนดเดือนที่ 3 เช่นกัน จึงมีการตรวจสอบต่อไปอีกในหน่วยวัน แล้วพบว่าวันที่ 14 เป็นค่าหน่วยเวลาสุดท้ายแล้ว จึงให้ผลลัพธ์เป็นชื่อจุดเวลาที่บันทึกไว้ที่โหนดวันที่ 14 ออกไป

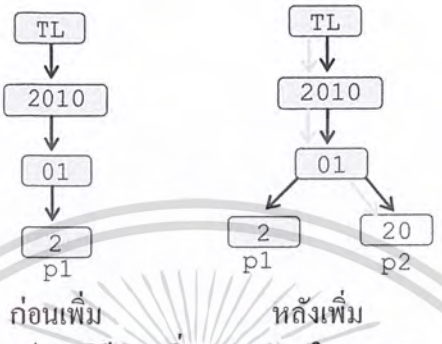


รูป 4.4 ขั้นตอนการค้นหาจุดเวลา

4.3.2 การเพิ่มจุดเวลาและช่วงเวลาลงในโครงสร้างข้อมูล

การเพิ่มจุดเวลาใดๆ กระทำได้โดยการสร้าง path เพื่อแทนจุดเวลาที่ต้องการเพิ่มเข้าไปโดยการเปรียบเทียบหน่วยจุดเวลาใหญ่สุดกับโหนดลูกของ root ลงไปทีละชั้นที่โหนดลูกลำดับต่อมาตามลำดับจนถึงหน่วยสุดท้าย และจะบันทึกชื่อจุดเวลานั้นไว้ที่โหนดหรือ leaf ที่หน่วยย่อยที่สุดไปสิ้นสุดอัลกอริธึมนี้รับอินพุทเป็น list ของเวลาเริ่มต้นกับสิ้นสุด และชื่อจุดเวลา symbol โดยจะอ่านค่าหน่วยเวลาออกมาที่ละหน่วยเวลา ซึ่งค่าที่ได้ถูกนำไปเทียบกับสมาชิกในลิสต์ของโหนดลูกทั้งหมดของ root ถ้าไม่พบก็จะสร้างโหนดนั้นขึ้นมา แล้วเรียกใช้อัลกอริธึมเดิมอีกครั้ง โดยส่ง subtree ที่มี root เป็นโหนดที่ค้นพบ ลิสต์ชุดค่าหน่วยเวลาที่เหลือ tail(tp) และชื่อจุดเวลาที่จะบันทึก

ลงใน Tree ไปเป็นพารามิเตอร์ ซึ่งจะกระทำเช่นนี้ไปเรื่อยๆ จนกระทั่งหมดค่าหน่วยเวลาของจุดเวลาที่ต้องการบันทึก แล้วจึงบันทึกชื่อจุดเวลานั้นไว้ที่โหนดสุดท้าย (ซึ่งอาจจะเป็น leaf ก็ได้) ที่หน่วยเวลาย่อยสุดท้ายไปสิ้นสุดต่อไปเป็นตัวอย่างการเพิ่มจุดเวลา p2 = 2010/1/20 ลงใน Tree ด้านซ้าย ตามอัลกอริธึม ซึ่งจะได้ Tree ด้านขวา ดังรูป



รูป 4.5 วิธีการเพิ่มจุดเวลาลงใน Tree

ส่วนการเพิ่มช่วงเวลา กระทำได้โดยการเพิ่มจุดเริ่มต้น และจุดสิ้นสุดของช่วงเวลาเข้าไปใน Tree โดยข้อมูลที่บันทึกไว้ที่ leaf เป็นชื่อจุดเริ่มต้น และชื่อจุดสิ้นสุดของช่วงเวลาตามลำดับ

4.3.3 การลบจุดเวลาและช่วงเวลาในโครงสร้างข้อมูล

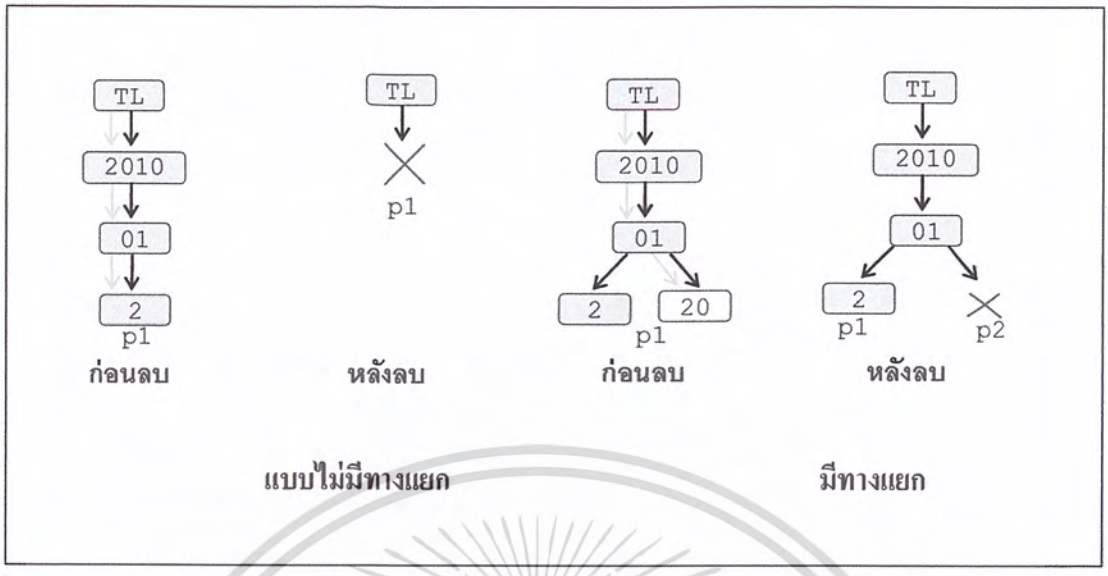
การลบจุดเวลาและช่วงเวลาใดๆ กระทำได้โดยการลบเวลาออกจาก Tree ที่บันทึกจุดเวลานั้นอยู่ โดยจะลบก็ต่อเมื่อได้ทำการประมวณผลคำสั่งนั้นๆแล้ว จึงลบเวลานั้นๆออกจาก treeซึ่งจะลบทั้งจุดเวลาเริ่มต้นและจุดเวลาสิ้นสุดของคำสั่ง โดยอัลกอริธึมดังกล่าว โดยเริ่มที่ delete_stmt() จะทำการเซตค่าเริ่มต้นให้ตัวแปลต่างๆแล้วเรียกไปยัง delete_tp() มีการทำงานโดยเริ่มจากการ traverse ลงไปใน tree แล้วเปรียบเทียบค่าทีละโหนด กับค่าเวลาปัจจุบัน จนเมื่อมีค่าไม่เท่ากัน ให้เก็บค่าโหนดนั้นไว้ในตัวแปร cantdelete หมายถึงห้ามลบโหนดนี้ ซึ่งหมายความว่าใน tree มีทางแยกให้เก็บ parent ไว้ในตัวแปร nodedelete เพื่อจะทำการลบลบโหนดลูกที่จุดนนี้ และทำการ traverse ต่อไปจนไปยังจุดสิ้นสุดของ tree ซึ่งเราจะได้ตัวแปรมาสองตัวคือ nodedelete และ cantdelete แล้วจะส่งไปยัง delete () ให้ทำงานเพื่อทำการลบจุดเวลาโดยจะนำตัวแปรทั้งสองที่ได้มามาตรวจสอบว่าควรจะลบลูกของ nodedelete ที่เส้นทางใดโดยเปรียบเทียบกับ ข้อมูลใน cantdelete ที่เก็บค่าเส้นทางที่ห้ามลบ โดยถ้าลูกของ nodedeleteไม่อยู่ใน cantdelete ให้ลบลูกนั้น แต่ถ้า cantdelete ไม่มีค่าแสดงว่า เส้นทางแรกในการค้นหาเป็นเส้นทางที่จะต้องลบ ซึ่งจะมีการลบสองกรณีคือ tree มีทางแยกกับไม่มีทางแยก ถ้าไม่มีทางแยกคือที่โหนดใดๆไม่มีลูกเกินหนึ่ง ให้ลบที่ root tree แต่ถ้ามีทางแยกให้ลบตรงที่มีทางแยกและลบลูก children[0]

```
def delete_stmt(roottimetable, timepoint):
    cantdelete=[]
    nodedelete = roottimetable
    nodedelete1 = roottimetable
    roottimetable = delete_tp(roottimetable,timepoint,nodedelete,nodedelete1,cantdelete)


def delete_tp(roottimetable,timepoint,nodedelete,nodedelete1,cantdelete):
    if roottimetable.children == []:
        cantdelete+=[None]
        delete(nodedelete1,nodedelete,cantdelete)
        return
    else :
        for childnode in roottimetable.children :
            if int(childnode.name) == int(timepoint[0]):
                return delete_tp(childnode,timepoint[1:],nodedelete,nodedelete1,cantdelete)
            else:
                nodedelete=roottimetable
                cantdelete += [[int(childnode.name)]]


def delete (nodedelete1,nodedelete,cantdelete):
    havechild=0
    count=0
    if nodedelete1.children == []:
        del nodedelete.children[0]
        return
    if cantdelete == [None] :
        for childnode in nodedelete1.children:
            havechild+=1
        if (havechild==1):
            return delete (childnode,nodedelete,cantdelete)
        else :
            del nodedelete1.children[0]
            return
    if nodedelete.children==[]:
        return 'no delete time tree'
    else:
        for child in nodedelete.children:
            if [child.name] != cantdelete[count]:
                del nodedelete.children[count]
            count+=1
```

รูป 4.6 อัลกอริธึมสำหรับการลบจุดเวลา



รูป 4.7 การลบจุดเวลา

4.4 คุณสมบัติของโครงสร้างข้อมูล

โครงสร้างข้อมูลที่ออกแบบขึ้นมามีคุณสมบัติดังนี้

1) ลำดับของการเพิ่มจุดเวลาไม่มีผลต่อโครงสร้างข้อมูล
2) พื้นที่เก็บข้อมูลสามารถแบ่งได้เป็น 2 ส่วน คือ โหนดที่เป็นโครงสร้างหลัก และชื่อจุด เวลาที่บันทึกไว้ที่โหนดหรือ leaf โดยโครงสร้างหลักใช้พื้นที่ขึ้นอยู่กับลักษณะจุดเวลาที่จัดเก็บ เช่น ถ้าโครงสร้างข้อมูลจัดเก็บจุดเวลาที่มีลำดับที่ของปีและเดือนต่างกันมากจะทำให้โหนด ปี และโหนดเดือน ที่ไม่มีการใช้ร่วมกัน ทำให้มีจำนวนโหนดมาก

4.5 การบันทึกประโยค CL ในโครงสร้างข้อมูล

ประโยค time[p_s, p_e]:statement มีการจัดเก็บในโครงสร้างข้อมูลตามรูปแบบของประโยค ดังนี้

1) ประโยคสำหรับการกำหนดค่าให้ตัวแปร อยู่ในรูปแบบ time[p_s, p_e] : <id> = <expression> เราจะจัดเก็บสัญญลักษณ์ start <id> = <expression> ไว้ยังโหนดที่แทนเวลาเริ่มต้น p_s และจัดเก็บสัญญลักษณ์ stop <id> = <expression> ไว้ยังโหนดที่แทนเวลาสิ้นสุด p_e โดยที่ สัญลักษณ์ = code
2) ประโยคเรียกใช้ฟังก์ชั่น อยู่ในรูปแบบ time[p_s, p_e] : <function-name>(<argument>…) เราจะจัดเก็บ สัญญลักษณ์ start <function-name>(<argument>…) ไว้ยังโหนดที่แทนเวลาเริ่มต้น p_s และจัดเก็บสัญญลักษณ์ stop <function-name>(<argument>…) ไว้ยังโหนดที่แทนเวลาสิ้นสุด p_e โดยที่ สัญลักษณ์ = code

3) ประโยคชุดคำสั่ง

3.1) Sequential Statement ประโยคย่อยๆ ภายใน Sequential Statement จะถูกจัดเก็บลงใน Tree ต้นเดียวกัน ทั้งหมด โดยวิธีการจัดเก็บของประโยคย่อยๆ จะใช้วิธีการที่กล่าวไปแล้ว

3.2) Parallel Statement ประโยคย่อยๆ ภายใน Sequential Statement จะถูกจัดเก็บลงใน Tree ต้นเดียวกันทั้งหมด เนื่องจากโครงสร้างข้อมูลที่นำเสนอ สามารถเรียงลำดับข้อมูล จากการเพิ่มข้อมูลจุดเวลา ที่ไม่เป็นลำดับได้ ทำให้วิธีการจัดเก็บ Parallel Statement ใช้วิธีเดียวกับ Sequential Statement ได้

3.3) Unordered Statement เนื่องจากประโยคย่อยๆ ภายใน Unordered Statement ไม่มีการระบุลำดับการประมวลผลที่แน่ชัด ดังนั้นอินเตอร์พรีตเตอร์จะต้องกำหนดลำดับการประมวลผล ก่อนที่จะจัดเก็บ โดยหลังจากที่อินเตอร์พรีตเตอร์กำหนดลำดับให้กับประโยคย่อยแล้ว จุดเวลาที่เป็น '_' จะตีความหมายเช่นเดียวกับ จุดเวลา '_' ใน Sequential Statement และจัดเก็บลงในโครงสร้างข้อมูล

4.6 ประโยค CL ที่มีกำหนดเวลาเปลี่ยนแปลง

จากที่ผ่านมา เรากำหนดว่า time[p_s,p_e] ที่ใช้ในภาษา CL เป็นค่าคงที่เท่านั้น มีผลทำให้ง่ายต่อการออกแบบตัวอินเตอร์พรีตเตอร์ แต่กรณีที่จุดเวลาเริ่มต้นและจุดสิ้นสุดใน time[p_s,p_e]:statement ถูกกำหนดโดยตัวแปรที่มีค่าแปรเปลี่ยนตามคาบเวลา เช่น ทุกๆเดือน ทุกๆ วัน ตามประโยคทำซ้ำ เช่น for-statement รวมทั้งในกรณีที่ค่าของตัวแปร p_s หรือ p_e ถูกกำหนดค่าโดยการทำงานของ statement ก่อนๆ แล้ว อินเตอร์พรีตเตอร์นี้จะไม่สามารถรองรับการประมวลผลได้

ในหัวข้อต่อไป เราจึงขอนำเสนอการปรับปรุงโครงสร้าง Tree ที่ใช้เก็บตารางการทำงานของโปรแกรม CL แบบเดิม เพื่อให้รองรับการจัดเก็บประโยค CL ที่มีกำหนดเวลาเป็นตัวแปร

4.7 โครงสร้างข้อมูลสำหรับเก็บประโยค CL ที่มีกำหนดเวลาเปลี่ยนแปลง

มีการปรับปรุงจาก Tree เดิมใน 3 ลักษณะดังนี้

4.7.1 โครงสร้างข้อมูลสำหรับเก็บประโยค CL ที่มีการทำงานเป็นคาบๆ

ในชีวิตประจำวันจะพบว่ามีการทำงานที่เกิดขึ้นเป็นคาบๆ เป็นประจำ เช่น การจ่ายค่าบิล ณ วันที่ 1 ของทุกๆเดือน ซึ่งการทำงานในลักษณะนี้สามารถเขียนเป็นโปรแกรม CL ได้ดังนี้

โปรแกรม 4.2 การจ่ายบิล

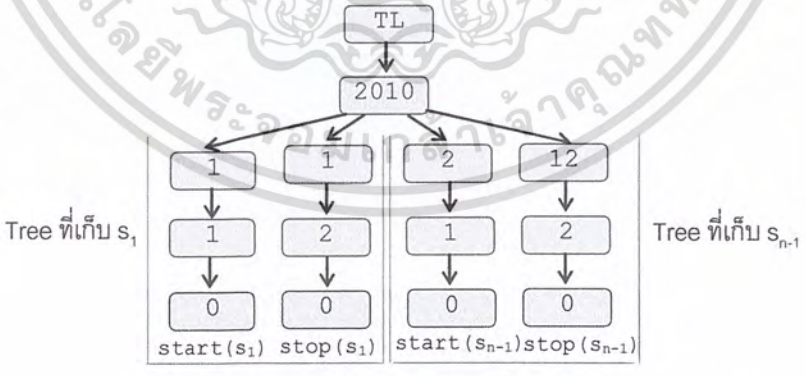
```
time[2010:1:1|0:0:0, 2010:12:2|0:0:0]:          (sn)
  for month in (1,12):
     time[2010:month:1|0:0:0,2010:month:2|0:0:0]:
          transfer_money("1000")
```

ซึ่งเป็นการสั่งให้คอมพิวเตอร์ โอนเงินเพื่อชำระบิล 1,000 บาท ณ วันที่ 1 และให้เสร็จภายในวันที่ 2 ของทุกๆเดือน ตลอดปี 2010 สังเกตว่าจะมีการนำเอาตัวแปรควบคุม month มาใช้ ใน for-statement เพื่อแปรเปลี่ยนค่าเดือน ในแต่ละคาบการทำงานทีละหนึ่ง ซึ่งแตกต่างจากภาษา CL เดิมที่ไม่สามารถใช้ตัวแปรกับกำหนดเวลาการทำงานได้

เราพบว่าตารางการทำงานซ้ำๆ ที่เป็น n คาบ สามารถแบ่งออกโดยเรียงลำดับเป็นตารางการทำงานคาบที่ 1 ก่อน แล้วตามด้วยตารางการทำงานสำหรับอีก n-1 คาบที่เหลือ โดยวิธีนี้ทำให้สามารถออกแบบโครงสร้าง Tree ที่ใช้จัดเก็บตารางการทำงานซ้ำๆ ที่เป็น n คาบได้ โดยการแบ่งเป็น Tree ที่เก็บตารางการทำงานของคาบที่ 1 แล้วตามด้วย Tree ที่เก็บตารางการทำงานของ n-1 คาบที่เหลือ ดังนั้น Tree ที่เก็บ s_n ดังรูป ประกอบด้วย Tree ที่เก็บ s_1 คือ

โปรแกรม 4.3 การจ่ายบิล โดยออกแบบโครงสร้าง Tree

```
time[2010:1:1|0:0:0,2010:1:2|0:0:0]transfer_money("1000")
      for month in (2,12):
        time[2010:month:1|0:0:0,2010:month:2|0:0:0]:
          transfer_money("1000")
```



รูป 4.8 Tree สำหรับเก็บตารางเวลาที่ทำงานเป็นคาบ

บทที่ 5
การออกแบบและพัฒนาอินเตอร์พรีตเตอร์ภาษา CL

ในบทนี้เราจะกล่าวถึงการขั้นตอนการประมวลผลประโยคในภาษา CL และอธิบายหน้าที่แต่ส่วนประกอบต่างๆภายในอิเตอร์พรีตเตอร์ภาษา CL เนื้อหาของบทนี้เริ่มจากการอธิบายภาพรวมของโครงสร้างอินเตอร์พรีตเตอร์ภาษา CL ในหัวข้อที่ 5.1 จากนั้นจะอธิบายขั้นตอนการทำงานของอินเตอร์พรีตเตอร์ตั้งแต่ขั้นตอนการตีความในกลุ่มของประโยค การจัดเก็บประโยค CL ในโครงสร้างข้อมูลจนกระทั่งประโยคถูกแปลความหมายเป็นการทำงาน ในหัวข้อที่ 5.2 จากนั้นเราจะอธิบายการทำงานของส่วนประกอบที่สำคัญ ได้แก่ ส่วนการรับ Source Code และการทำงานของ CL Parser ในหัวข้อที่ 5.3 Namespace หัวข้อที่ 5.4 การทำงานของส่วน Time Responderหัวข้อที่ 5.5การทำงานส่วน Code Executorหัวข้อที่ 5.6กระบวนการ Import file หัวข้อที่ 5.7 คลาสในภาษา CL หัวข้อที่ 5.8 การติดต่อสื่อสารระหว่าง Client กับ Server หัวข้อที่ 5.9 และสรุป ในหัวข้อที่ 5.10

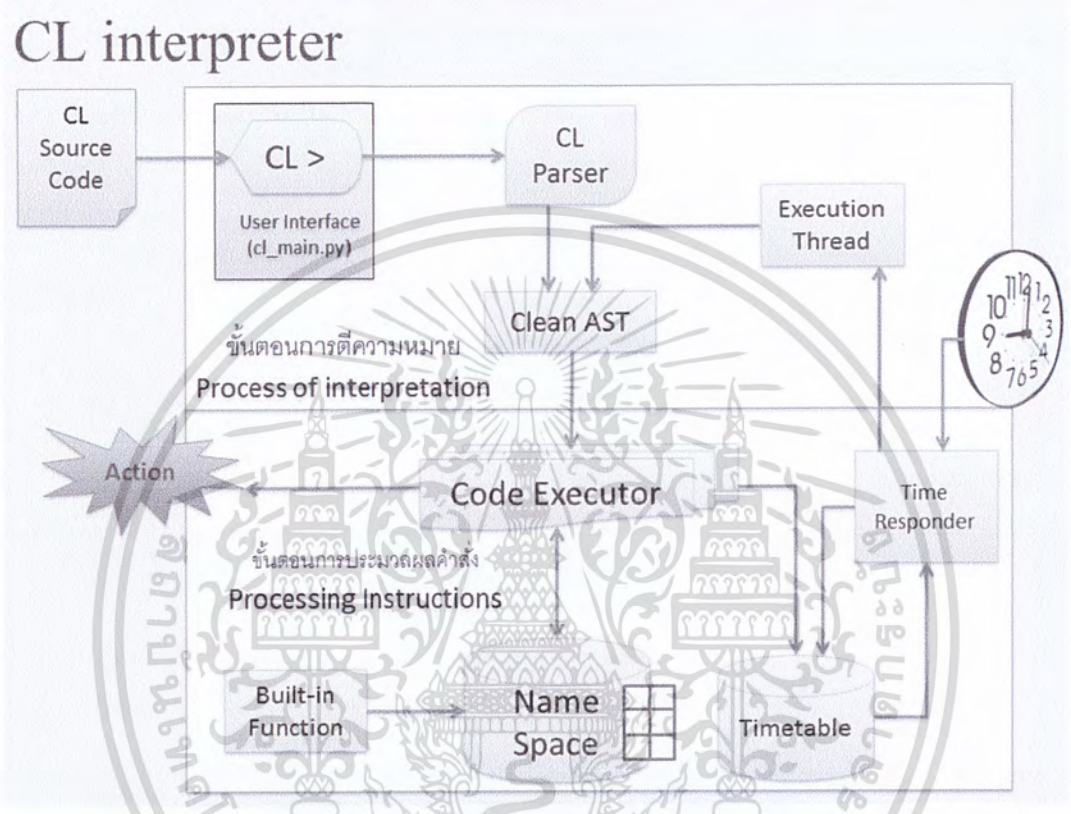
5.1 โครงสร้างอินเตอร์พรีตเตอร์CL

Source Code ในโปรแกรม CL จะถูกนำมาประมวลผล โดยอินเตอร์พรีตเตอร์CL ที่มีโครงสร้างหลักประกอบด้วย

1) Clock เป็นนาฬิกาของอินเตอร์พรีตเตอร์
2) Timetable เป็นตารางเวลาของอินเตอร์พรีตเตอร์ที่ทำการจัดเก็บ ประโยคคำสั่งตามเวลา
3) CL Parser เป็นรูปแบบของภาษาที่เอาไว้ตีความในแต่ละประโยคของภาษา CL
4) User Interface เป็นเธรดที่คอยรับ Source Codeจาก Console ฝั่ง server กับ User Interface ในฝั่ง Clientและ คอยแสดงสถานะของโปรแกรม
5) Time Responder เป็นเธรดที่คอยอ่านเวลาปัจจุบันมาจาก Clock เพื่อตรวจสอบกับกำหนดการทำคำสั่งที่เก็บอยู่ในตารางเวลาของอินเตอร์พรีตเตอร์
6) Built-in Function เป็นไลบรารีที่เก็บฟังก์ชันที่รวมอยู่ในอินเตอร์พรีตเตอร์เป็นฟังก์ชันการทำงานพื้นฐานของอินเตอร์พรีตเตอร์ เช่นการพิมพ์ข้อความออกมาทางจอ
7) Namespace เป็นที่เก็บการนิยามของฟังก์ชัน การนิยามของคลาส การนิยามของตัวแปร
8) Clean AST เป็นฟังก์ชันที่ทำหน้าที่ปรับปรุงโครงสร้างของ AST

9) Code Execution เป็นเธรดที่ทำงานตามคำสั่งภาษา CL โดยจะทำหน้าที่ ประสานงานกับ Namespace และ Timetable

10) Code Interpreter เป็นเธรดที่ทำงานตามคำสั่ง CL



รูป 5.1 ขั้นตอนการประมวลผลประโยคภาษา CL

5.2 ขั้นตอนการประมวลผลประโยคภาษา CL

ขั้นตอนการประมวลผลประโยคภาษา CL นั้นประกอบไปด้วยสองส่วน คือ กระบวนการตีความหมาย และ กระบวนการประมวลผลคำสั่ง

5.2.1 กระบวนการตีความหมาย

การทำงานเริ่มต้นจากส่วนของ User Interface ในส่วนของการรับ Source Code จาก Command line หรือ Text Fileหรือ Source Code จากฝั่ง client เมื่อรับ Source Code มาแล้วนั้นจะนำไปตีความหมายและตรวจสอบข้อผิดพลาด ในส่วนของ CL Parser ตามรูปแบบของภาษา CL ที่ได้กำหนดไว้จะได้AST (Abstract Syntax Tree) จะถูกนำไปปรับปรุงโครงสร้าง AST เพื่อเตรียมพร้อมให้ในส่วนของการตอบสนองคำสั่ง

5.2.2 กระบวนการตอบสนองคำสั่ง

ประกอบด้วยการทำงานส่วนหลักอยู่สองส่วนคือ Code Executor และ ส่วนของ Time responder

หลังจากส่วนกระบวนการตีความหมายปรับปรุงโครงสร้าง AST แล้ว AST จะถูกนำมาตีความหมายเพื่อทำการตอบสนองคำสั่งในส่วนของ Code Executor ซึ่งจะทำงานควบคู่ไปกับ Namespace และ Time table หากพบว่าคำสั่งชุดใดถูกกำหนดให้ทำงานตามเวลา ในส่วนของ AST นั้นจะถูกนำมาจัดเก็บลงใน Timetable หากมีการเรียกใช้ตัวแปร ใช้ฟังก์ชัน คลาส หรือ ออปเจค Code Executor จะทำการเรียกใช้ในส่วนของ Namespace และ Builtins function ต่อไป

การทำงานในส่วนของ Time Responder คอยทำหน้าที่อ่านเวลาปัจจุบันมาจาก Clock เพื่อตรวจสอบกับกำหนดการทำคำสั่งใน Tree (Timetable)เมื่อถึงเวลาจะทำการสร้างเธรดเพื่อประมวลผล ผ่านส่วนของ Code Executor และ เมื่อถึงเวลาสิ้นสุดจะทำลายเธรดที่กำลังประมวลผลพร้อมกับลบ Node ใน Tree ที่เก็บ Statement นั้นๆออกจาก Tree

5.3 การรับ Source Code และการทำงานของ CL Parser

เป็นส่วนที่ทำงานติดต่อกับผู้ใช้ ซึ่งจะทำหน้าที่รับ Source Code จาก Command line หรือ Text File หรือ Source Code จากฝั่ง clientถ้ารับจาก Command line จะส่ง Source Code ไปยังฟังก์ชัน command_line_input ในไฟล์ cl_parser.py เพื่อทำการตรวจสอบว่า Source Code ที่ผู้ใช้ป้อนเข้าไปนั้นผู้ใช้ต้องการป้อนเป็นชุดคำสั่ง(เป็นการนิยามของฟังก์ชัน การนิยามของคลาส หรือ กำหนดชุดคำสั่งเป็นเวลา)หรือไม่โดยทำการตรวจสอบเครื่องหมาย “ : ” ถ้าพบเครื่องหมายนี้ จะทำการรับ Source Code ในบรรทัดต่อไป โดยยังไม่ส่งไปยังส่วนของ CL Parser และในขณะเดียวกันนั้นจะทำการเช็คย่อหน้าว่าผู้ใช้ได้ทำการป้อนถูกต้องตามหลักของภาษา CL หรือไม่ หากไม่ถูกต้องจะแสดงข้อความออกมาว่าไม่ถูกต้อง เมื่อผู้ใช้ทำการป้อน Source Code ที่เป็นชุดคำสั่งเสร็จ(โดยการกด Enter โดยไม่ป้อนข้อมูลอะไร) จะทำการส่งชุดคำสั่งทั้งหมดไปยังส่วน CL Parser ต่อไป ผ่านฟังก์ชัน tokenize_parseหากรับ Source Code จาก Text file จะทำการส่งไปยังฟังก์ชัน tokenize_parseเช่นกัน

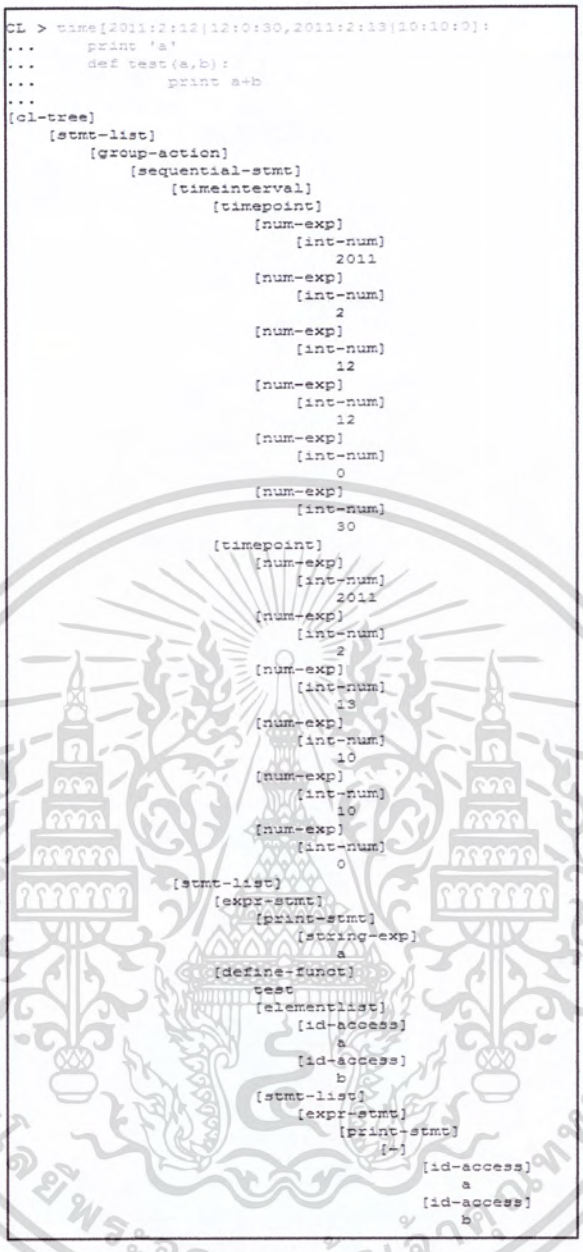
```
CL > time[2005:12:10|20:31:40,2005:12:11|12:31:40]:
...      a = 5
...      b = 6
...      c = taro(a,b)
...      a = taro(c,b)
...      b = a + b
...
CL >
CL > def taro(a,b):
...       c = a + b
...       return c
...
CL >
```

รูป 5.2 ตัวอย่างการรับ Source Code จาก Command line

การทำงานของฟังก์ชันของ tokenize_parse มีอยู่สองส่วนคือ ส่วนของ tokenize คือส่วนที่ดักจับString เป็นคำๆ ตามที่เรากำหนดไว้เช่น ตัวเลข ตัวแปร หรือ คำต่างๆเป็นต้น และส่วน parsing ทำหน้าที่ตีความหมายหลังจาก tokenize ของแต่ละประโยค ตามรูปแบบที่เราได้กำหนดไว้ เมื่อทำการตีความหมายของชุดประโยคแล้ว จะทำการสร้าง AST (Abstract Syntax Tree) เพื่อง่ายต่อการประมวลผลต่อไป

```
def cl_parser(source):
    source = filter_comment(source)
    if check_Indent(source) == 'correct':
        tokenize(source)
        ast = parse(source)
        return ast
    else:
        print "Indent error"
        return None
```

รูป 5.3 โค้ดในส่วน CL Parser



รูป 5.4 ตัวอย่างAST (Abstract Syntax Tree)

5.4 เนมสเปช Namespace

5.4.1 ความหมายของ Namespace

เนมสเปซ (namespace) เป็นชื่อเรียกทางนามธรรมของกลุ่มสิ่งที่บรรจุชื่อความหมายหรือคำศัพท์ของสิ่งหนึ่งๆ หรือคนใดคนหนึ่ง

ตามกฎของเนมสเปซชื่อใดๆที่อยู่ในแต่ละเนมสเปซสามารถมีความหมายได้เพียงหนึ่งความหมายโดยสิ่งที่แตกต่างกันไม่สามารถใช้ชื่อเดียวกันได้ภายในเนมสเปซเดียวกันถ้าชื่อเดียวกันอยู่คนละเนมสเปซสามารถหมายถึงของสองสิ่งที่แตกต่างกันได้

ภาษาแต่ละภาษาในโลกเปรียบเสมือนเป็นเนมสเปซหนึ่งๆซึ่งคำหนึ่งๆในแต่ละภาษาจะหมายถึงสิ่งของสิ่งๆหนึ่งซึ่งเมื่อคำเดียวกันในภาษาที่แตกต่างกันสามารถหมายถึงสิ่งของคนละสิ่งได้ภาษาที่แตกต่างกันของเนมสเปซนี้รวมไปถึงภาษาถิ่น และภาษาโปรแกรมต่างๆ ตัวอย่างเช่นคำว่า "ไอ" ในภาษาไทยหมายถึง อาการไอ ขณะที่ "ไอ" ในภาษาอังกฤษหมายถึง เรา (ตัวผู้พูด) และ "ไอ" ในภาษาญี่ปุ่นหมายถึง ความรัก หรือ คำว่า "ซาว" ในภาษาไทยกลาง หมายถึงการซาวข้าว ขณะที่ภาษาไทยเหนือ หมายถึง ยี่สิบ

ในภาษาทางคอมพิวเตอร์ภาษาหนึ่ง จะประกอบไปด้วยหลายๆเนมสเปซ ชื่อของ ตัวแปร ฟังก์ชัน หรือ คลาส ใดๆที่อยู่ในแต่ละเนมสเปซสามารถมีนิยามหรือค่าได้เพียงหนึ่ง โดยสิ่งเดียวกันไม่สามารถใช้ชื่อเดียวกันได้ภายในเนมสเปซเดียวกัน ถ้าชื่อเดียวกัน อยู่คนละเนมสเปซสามารถหมายถึงนิยามหรือค่า ที่แตกต่างกัน

5.4.2 โครงสร้างของ Namespace ในภาษา CL

เนมสเปซ (namespace) ในภาษา CL นั้นจะถูกสร้างขึ้นจากโครงสร้างข้อมูลแบบ Dictionary ของภาษา Python

Dictionary จะประกอบไปด้วย Key กับ Value ซึ่ง Key จะทำหน้าที่เก็บ Namespace ย่อย ของ Namespace นั้นๆ Key เปรียบได้ดั่งเป็น ชื่อตัวแปร ชื่อฟังก์ชัน หรือ ชื่อส่วน Value จะเก็บค่าต่างๆของตัวแปร นิยามของฟังก์ชัน นิยามของคลาส เป็นต้น

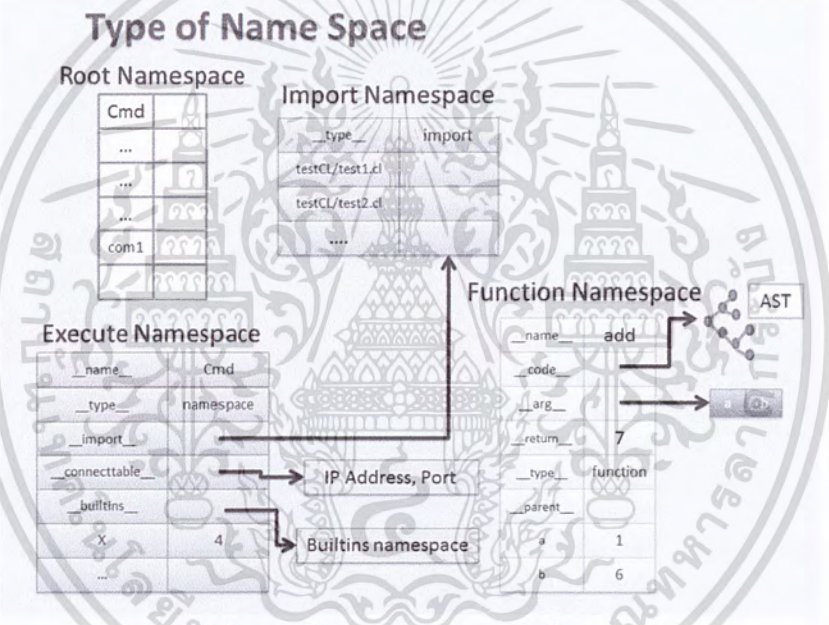
5.4.3 ประเภทของ Namespace ในภาษา CL

Namespace ในภาษา CL มีทั้งหมด 7 ประเภท ดังนี้

1) Root Namespace เป็น namespace ที่รวม execute namespace ทุก namespace ไว้เพื่อจัดการ execute namespace ให้เป็นระเบียบ key ต่างๆ ใน namespace นี้จะเป็นชื่อ execute namespace ทั้งสิ้น เพื่อจัดการไม่ให้โปรแกรมแต่ละโปรแกรมเกิดความสับสนในการเรียกใช้ข้อมูลต่างๆใน Execute namepsace
2) Execute Namespace เป็น namespace พื้นฐานในการประมวลผลคำสั่งในภาษา CL ซึ่งจะ 1Execute namespace จะหมายถึงโปรแกรม 1 โปรแกรมเท่านั้นตัวแปร ที่เก็บอยู่ใน Execute namespace นั้นจะเป็นตัวแปร Global นอกจากนี้ยังมี key หลักต่างๆ ดังนี้ __name__ จะเก็บชื่อของ namespace __type__ จะเป็น key ที่บอกว่าเป็น namespace ชนิดใด __import__ เป็น key ที่เก็บ Reference ของ Import namespace ของ execute namespace นั้นๆ __connecttable__ จะเก็บ IP Address และ Port number ของ Client หากคำสั่งภาษา CL นั้น ถูกส่งมาประมวลผลจากฝั่ง Client และสุดท้าย __builtins__ จะเก็บ Reference ของ Builtins namespace ไว้
3) Builtins Namespace เป็น namespace ที่เก็บ Reference ของ Builtins function ในภาษา Python และ ในภาษา CL

4) Import Namepsace เป็น namespace ทำหน้าที่เก็บ Reference ของ Execute namespace ของไฟล์นั้นที่ทำการ Import เข้ามาในโปรแกรม โดยใช้ key เป็นชื่อและที่อยู่ของไฟล์

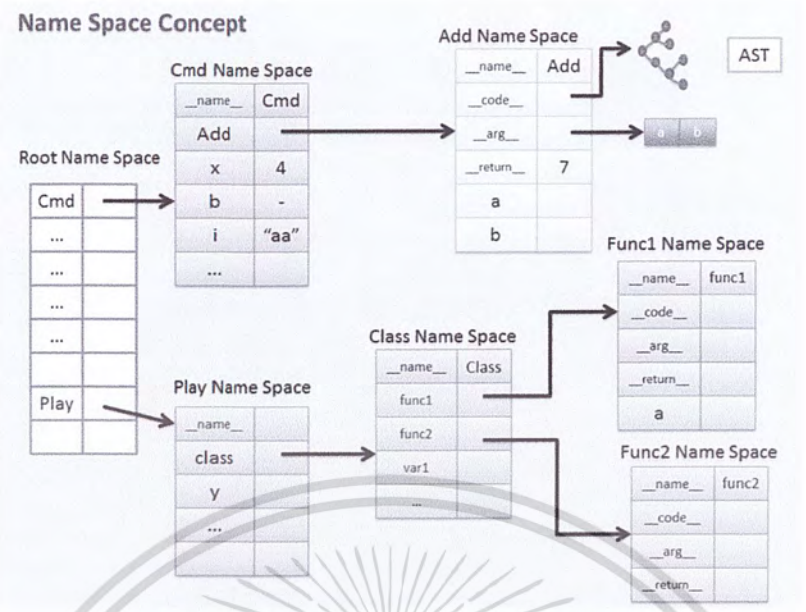
5) Function Namespace เป็น namespace ของฟังก์ชันในภาษา CL ซึ่งมีส่วนประกอบคือ __name__ เก็บชื่อของฟังก์ชัน __code__ เก็บ AST ที่เป็น Statement ภายในฟังก์ชัน __arg__ เก็บ list ของ Argument ของฟังก์ชัน __return__ เอาไว้ใช้สำหรับการคืนค่าหลังจากการเรียกใช้งานเสร็จสิ้นแล้ว __parent__ เป็น Reference ของ Namespace ที่ ฟังก์ชัน namespace อยู่ ซึ่งเอาไว้ใช้สำหรับการเรียกใช้ ตัวแปร Global



รูป 5.5 ประเภทต่างๆของ Namespace ในภาษา CL

6) Class Namespace เป็น Namespace ที่เก็บส่วนต่างของ Class ไว้ ตั้งแต่ Reference ที่ชี้ไปยังฟังก์ชัน ตัวแปร เป็นต้น โดยจะเก็บ key หลักได้แก่ __name__ , __type__ , __inheritance__ ซึ่ง key นี้จะเก็บ dictionary ที่เก็บ Reference ไปยัง Class แม่โดยใช้ Key เป็นชื่อของ Class แม่

7) Object Namespace เป็น Namespace ที่มีลักษณะคล้ายกับ Namespace ของ Class ซึ่ง Namespace นี้เอาไว้ใช้กับ Object นั้นๆ



รูป 5.6 ตัวอย่างการเก็บNamespace ในภาษา CL

5.5 การทำงานของส่วน Time Responder

เป็นเธรดที่คอยตรวจสอบว่า ณ เวลานั้นมีคำสั่งที่กำหนดเวลาทำงานอะไรบ้างใน Timetable โดยจะทำการเรียกฟังก์ชัน executeAtTimeทุกๆ 0.01 วินาที ซึ่งฟังก์ชันนี้จะทำหน้าตรวจสอบการทำงานกังกล่าว เมื่อพบแล้ว จะนำชุดของคำสั่งนั้นซึ่งเก็บอยู่ Timetable ในรูปแบบ AST มาประมวลผลโดยสร้างเธรดขึ้นมาเพื่อรับผิดชอบงานนั้นๆ เมื่อนำ AST มาได้แล้วนั้นจะลบชุดคำสั่งที่เก็บไว้ พร้อมกับลบ Node Startใน Treeของ Timetable และทำงานผ่านฟังก์ชัน execute เพื่อตอบสนองคำสั่งแต่ละคำสั่งนั้นๆ เมื่อทำงานสิ้นสุดลงจะลบNode Stopใน Tree ของ Timetable และทำลายเธรดที่สร้างขึ้นออกไป

```
class TimeResponder(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.running = True
        self.pause = False
    def run(self):
        while(self.running):
            if self.pause:
                time.sleep(1)
                continue
            timepoint = getCurrentTime()
            executeAtTime(namespace,timetable.rootScope, timepoint)
            time.sleep(0.01)
```

รูป 5.7 Class TimeResponder

```
def executeAtTime(scope,roottimetable, timepoint):
    roottimetable_stmt_list = timetable.find_stmt(roottimetable, roottimetable,timepoint)
    for (roottimetable, meta_stmt) in  roottimetable_stmt_list:
        for (prefix,name_scope,ast) in meta_stmt:
            if prefix == 'start':
                exp=[]
                for stmt in ast.children[0].children[0].children:
                    exp+= [stmt.type]
                t = Interpreter(name_scope,ast.children[0])
                t.run()
                runningThread[ast] = t
                del t
                timetable.delete_stmt(roottimetable,timepoint)
            else :
                timetable.delete_stmt(roottimetable,timepoint)
```

รูป 5.8 ฟังก์ชัน executeAtTime

5.6 การทำงานของส่วน Code Executor

เธรดที่ถูกสร้างขึ้นมาประมวลผลชุดคำสั่ง ชื่อ Interpreter เธรดนี้จะทำงานในส่วนที่เรียกว่า Code Executor โดยเธรดนี้จะทำการเรียกฟังก์ชัน execute ซึ่งจะรับพารามิเตอร์อยู่สี่ตัวคือ Roottime table(Reference ที่ชี้ไปยัง Root time table), Scope (เนมสเปซ ในส่วนที่เราสนใจ), Root Namespace และ AST (ชุดคำสั่งที่ประมวลผลอยู่ในรูป AST)

```
61 class Interpreter(threading.Thread):
62     def __init__(self, name_namespace,ast):
63         threading.Thread.__init__(self)
64         self.name_namespace = name_namespace
65         self.ast = ast
66         self.running = True
67
68     def run(self):
69         interpreter.execute(timetable.rootScope,self.name_namespace,self.name_namespace,self.ast)
70         pass
71
```

รูป 5.9 คลาส Interpreter

ฟังก์ชัน execute จะทำหน้าที่ตอบสนองคำสั่งของ CL Statement โดยตรวจสอบชนิดของ Node ใน AST และจะทำงานตามชนิดของ Node ดังตัวอย่างต่อไปนี้

```
def execute(roottimetable, scope, root_namespace, ast):
    if not isinstance(ast, parser.INode):
        return

    if ast.type == 'cl-tree' :
        for stmt in ast.children:
            execute(roottimetable, scope, root_namespace, stmt)

    elif ast.type == 'stmt-list' :
        for stmt in ast.children:
            execute(roottimetable, scope, root_namespace, stmt)

    elif ast.type == 'define-funct':
        create_namespace(roottimetable, scope, root_namespace , ast)

    elif ast.type == 'import-as-stmt' or ast.type == 'list-import-stmt':
        create_namespace_import(roottimetable, scope, root_namespace , ast)

    elif ast.type == 'Class' :
        filter_class_stmt(roottimetable, scope, root_namespace , ast)

    elif ast.type == 'group-action':
        for stmt in ast.children:
            filter_insert_group_stmt(scope, roottimetable, stmt)

    elif ast.type == 'sequential-stmt' :
        for stmt in ast.children:
            execute(roottimetable, scope, root_namespace, stmt)

    elif ast.type == 'timeinterval' :
        for stmt in ast.children:
            execute(roottimetable, scope, root_namespace, stmt)

    elif ast.type == 'expr-stmt':
        exp = execute(roottimetable, scope, root_namespace, ast.children[0])
        try:
            if '__type__' in exp:
                print "<" + exp['__name__'] + " is " + exp['__type__'] + ">"
                return exp
        except:
            pass
```

รูป 5.10 ตัวอย่างฟังก์ชัน execute

5.6.1 การทำงานร่วมกับ Namespace

หลังจากที่ AST ถูกประมวลผลในส่วนของ Code Executor ซึ่ง Code Executor นี้จะมี reference ที่ชี้ไปยัง namespace ต่างๆ ผ่าน function access_scopeเพื่อเข้าถึงข้อมูลใน namespace ต่างและนำมาบันทึกหรือใช้งานซึ่งฟังก์ชัน access_scopeนี้จะทำงานยึดตามหลัก คือ LEGB หมายความว่า ลำดับในการเข้าถึงข้อมูลนั้นจากทำการค้นหาข้อมูลที่สนใจใน Local scope เป็นอันดับแรกหาไม่พบจะ ทำการค้นหาใน Enclose นั้น หากไม่พบจากหาในส่วนที่เป็น Global หรือใน Execute namespace นั่นเองและหากไม่พบอีกจะทำการค้นหาในส่วนที่เป็น Builtins function และหากว่าไม่พบจากทำการฟ้อง Error ออกมาว่าไม่พบ

ในขณะที่ Code Executor ทำงานอยู่นั้นหากพบส่วนการนิยามคลาส การนิยามฟังก์ชันหรือการ Import File ส่วนของ Code Executor จะทำการเรียกฟังก์ชัน filter_class_stmt,

filter_define_namespace , create_namespace_importตามลำดับ เพื่อสร้าง Namespace แต่ละประเภทและบันทึกลงใน local scope ที่ทำการเรียกใช้คำสั่งนั้นๆ

5.6.2 การเรียกใช้ฟังก์ชัน

หากฟังก์ชัน execute ตรวจสอบชนิดของ Node เป็น funct-expฟังก์ชันจะทำการมองหาชื่อฟังก์ชันใน Namespace ของ Scope (Scope ในที่นี้หมายถึง Namespace ที่เราสนใจ หรือ Namespace ที่เป็น local) ก่อนโดยชื่อฟังก์ชันจะเป็น key ในการเรียกใช้ฟังก์ชัน หากไม่พบ จะไปมองหา ใน Root Namespace หากไม่พบอีกฟ้อง NameErrorออกมา

เมื่อพบฟังก์ชันใน Scope หรือ ใน Root Namespace แล้ว จะทำการเรียกฟังก์ชัน execute เข้าไปอีกชั้นเพื่อดูค่าของ Argument แต่ละตัวและส่งค่า Argument ให้แก่ Namespace ของฟังก์ชันโดยผ่าน Key '__argument__' หลังจากนั้นจะทำการเรียก ฟังก์ชัน call_namespaceโดยส่ง Namespace ของฟังก์ชัน เป็น Scope ที่ฟังก์ชันจะเรียกใช้ ส่ง Root Namespaceและส่ง AST ของฟังก์ชันที่อยู่ใน Namespace ของฟังก์ชันอยู่แล้วสุดท้ายการทำงานจะคืนค่าของฟังก์ชันโดยผ่าน Namespace ของฟังก์ชันใน Key '__return__'

```
elif ast.type == 'funct-exp':
    if  ast.children[0] in scope:
        pass_arg = []
        for arg in ast.children[1].children:
            pass_arg += [execute(scope,root_namespace,arg)]
        i = len(pass_arg)
        while i > 0:
            i = i - 1
            scope[ast.children[0]][scope[ast.children[0]]['__argument__'][i]] = pass_arg[i]
        call_namespace(scope[ast.children[0]],root_namespace,scope[ast.children[0]]['__code__'])
        return scope[ast.children[0]]['__return__']
    elif ast.children[0] in root_namespace:
        pass_arg = []
        for arg in ast.children[1].children:
            pass_arg += [execute(scope,root_namespace,arg)]
        i = len(pass_arg)
        while i > 0:
            i = i - 1
            root_namespace[ast.children[0]][root_namespace[ast.children[0]]['__argument__'][i]] = pass_arg[i]
        call_namespace(root_namespace[ast.children[0]],root_namespace,root_namespace[ast.children[0]]['__code__'])
        return root_namespace[ast.children[0]]['__return__']
    else:
        print "cannot find namespace "+ast.children[0]
```

รูป 5.11 การทำงานในกรณีเรียกใช้ฟังก์ชัน

การทำงานของฟังก์ชัน call_namespaceจะรับค่าสี่ตัวคือ Roottime table (Reference ที่ชี้ไปยัง Root time table), namepspace, Root Namespace และ astฟังก์ชันนี้จะทำหน้าที่เสมือน Stack ในการประมวลผลฟังก์ชันแบบ Recursive โดยใช้คุณสมบัติอินเตอร์พรีตเตอร์ภาษาไพธอนเข้าช่วย ในฟังก์ชันนี้จะ copy Namespace ของฟังก์ชันเก็บไว้เปรียบเสมือน Push Namespace ของฟังก์ชัน

ลงใน Stack หลังจากนั้นจะเรียกฟังก์ชัน execute และส่ง Namespace ของฟังก์ชันที่ copy แล้ว, ส่ง Root Namespace และส่ง AST ของฟังก์ชันลงไปอีกครั้ง หากในฟังก์ชันมีการเรียกตัวเองขึ้นอีกครั้ง ในฟังก์ชัน execute จะเรียกฟังก์ชัน call_namespaceอีกและทำการ copy Namespace ของฟังก์ชันอีกครั้งหนึ่งเท่ากับว่า Push Namespace ของฟังก์ชันลงไปใน Stack ทับ Namespace ของฟังก์ชัน ก่อนหน้านี้ เมื่อประมวลผลฟังก์ชันสิ้นสุดลงแล้ว ในฟังก์ชัน call_namespaceจะ copy ค่าของฟังก์ชันที่จะต้องคืนออกมาจาก scope จะถูก Copy คืนไปยัง Namespace ที่ถูก Copy มาเพื่อให้เมื่อจบฟังก์ชัน call_namespace แล้วจะไม่หายไปกับ Namespace ที่ถูก copy มา ซึ่งเปรียบเสมือน pop Namespace ของฟังก์ชันออกไป การเรียกใช้ฟังก์ชันของภาษา CL จะมีขั้นตอนการทำงานแบบนี้ไปเรื่อยๆจนการเรียกใช้ฟังก์ชันครั้งสุดท้ายสิ้นสุดลง

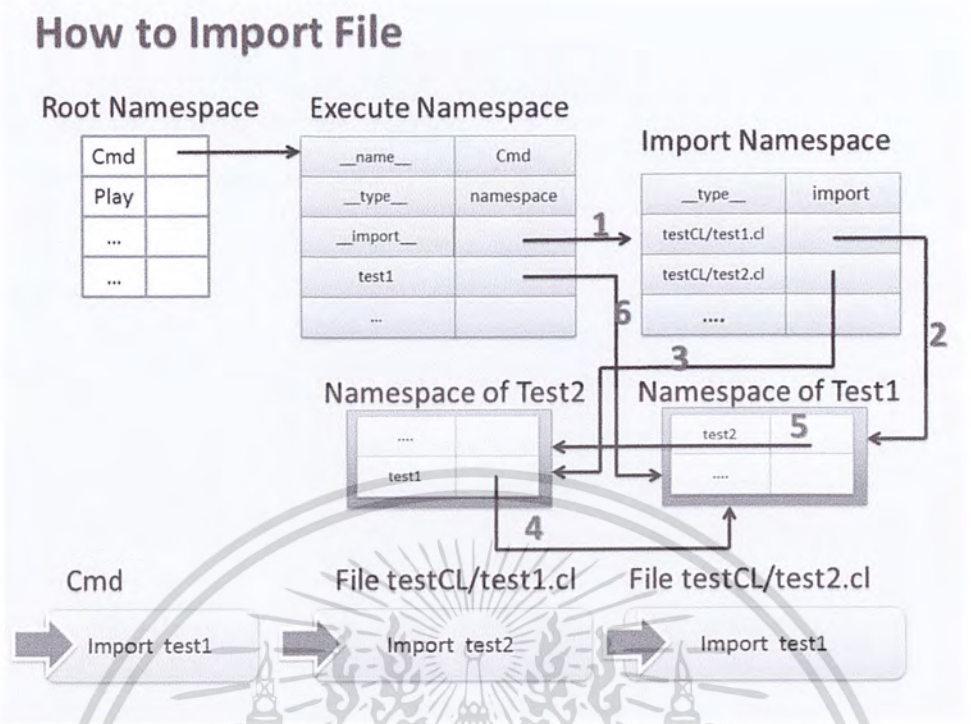
```
316
317  def call_namespace(roottimetable, namespace, root_namespace, ast):
318      scope = namespace.copy()
319      execute(roottimetable, scope, root_namespace, ast)
320      namespace['__return__'] = scope['__return__']
321
```

รูป 5.12 ฟังก์ชัน call_namespace

นอกจากตัวอย่างการทำงานของการประมวลผลตามคำสั่งของภาษา CL ข้างต้นนี้ยังมีกระบวนการประมวลผลอื่นๆอีก เช่น การกำหนดค่าตัวแปร (assign-stmt),การกระทำทางคณิตศาสตร์, Conditional operator(IF, FOR, WHILE, etc) เป็นต้น

5.7 กระบวนการ Import File

กระบวนการ Import File เป็นกระบวนการที่ทำให้โปรแกรมภาษา CL นั้นสามารถทำงานร่วมกันได้หลายไฟล์ด้วยกัน หากฟังก์ชัน execute ตรวจสอบชนิดของ Node เป็น import-as-stmt หรือ list-import-stmtจะทำการเรียกฟังก์ชัน create_namespace_importเพื่อ Import และทำการเชื่อต่อ Namespace ของไฟล์ที่ Import เข้ากับ local namespace ที่ ส่งคำสั่ง import file เข้ามา



รูป 5.13 ตัวอย่างกระบวนการ Import file

จากรูปเป็นตัวอย่างกระบวนการ import file จะพบว่าที่ Command line นั้นได้พิมพ์คำสั่ง import test1 ซึ่งเป็นการ import file จาก testCL/test1.cl ซึ่งในไฟล์นี้ ได้ import test1 ไว้ด้วยจะเห็นได้ว่าเป็นการ import file ไปมา เมื่อพิมพ์คำสั่ง import นั้น CL interpreter จะทำการตรวจสอบดูว่าใน import namespace ในExecute namespace ของโปรแกรมนั้นๆ ได้ทำการ Execute และสร้าง Namespace สำหรับไฟล์ที่ Import มาแล้วหรือไม่ หากพบว่าไฟล์ดังกล่าวไม่ได้ถูก execute ขึ้นมานั้น CL interpreter จะสร้าง Namespace เพื่อรองรับไฟล์นั้นและสร้าง Reference (หมายเลข 2)ไปยัง Namespace นั้นจากนั้นจึง Execute ไฟล์นั้นทั้งไฟล์ ซึ่งเมื่อ Execute ไฟล์ test1 แล้ว พบว่าที่ไฟล์ test1 ได้ทำการ import test2 CL interpreter จะเข้าไปทำการตรวจสอบใน Import namespace เช่นเดิมซึ่งพบว่าไม่ได้ทำการสร้าง namespace และ execute เหมือนไฟล์กับไฟล์ test1 และสร้าง Reference (หมายเลข 3) ในระหว่างการ Execute ของไฟล์ test2 นั้นจะพบว่ามีการ import test1 เข้ามาด้วย CL interpreter ก็ทำการตรวจสอบเช่นเดิมซึ่งพบว่าได้มีการสร้าง namespace ของไฟล์ test1 ไว้อยู่แล้ว Cl interpreter จึงสร้าง Reference หรือ link ไปยัง namespace test1 โดยใช้ key ชื่อว่า test1 (หมายเลข 4) จบจากไฟล์ test2 แล้วนั้น CL interpreter จะกลับมา execute ไฟล์ test1 ต่อ เมื่อจบในบรรทัด import test2 แล้วนั้น CL interpreter จะทำการสร้างReferenceหรือ link ไปยัง

namespace test2 โดยใช้ key ชื่อว่า test2(หมายเลข 5) หลังจากนั้นก็จะ Execute ไฟล์ test1 จบไฟล์ CL interpreter จึงสร้าง Reference ไปยัง test1 จาก namespace Cmd

ดังนั้นเมื่อมีการเรียกใช้ฟังก์ชันใน test1 จึงต้องอ้างถึง key ของไฟล์ test1 แล้วตามด้วยจุดก่อนชื่อฟังก์ชันในไฟล์ test1.xxxx() และหากต้องการเรียกใช้ข้อมูลในไฟล์ test2 จะต้องอ้างอิงจาก namespace ของไฟล์ test1 ก่อนแล้วตามด้วยไฟล์ test2 เป็นดังนี้ test1.test2.xxxxx() เป็นต้น

5.8 Object-Oriented Programming ในภาษา CL

5.8.1 ความหมายของ Class และ Object

คลาส (Class) คือ ประเภทของวัตถุเป็นการกำหนดว่าวัตถุ จะประกอบไปด้วย ข้อมูล (data) หรือคุณสมบัติ(property) และพฤติกรรม(behavior)หรือ การกระทำ(method) อะไรบ้าง ซึ่งคลาส (เช่นมนุษย์) เป็นโครงสร้างพื้นฐานของการเขียนโปรแกรมเชิงวัตถุ

วัตถุ (Object) โดยมากจะเรียกว่า อ็อบเจกต์ คือตัวตน(instance) ของ คลาส(เช่นนาย.ก, นาย.ข) ซึ่งจะเกิดขึ้นระหว่าง run-time โดยแต่ละ อ็อบเจกต์จะมีข้อมูลเฉพาะของตัวเอง ทำให้ อ็อบเจกต์แต่ละ อ็อบเจกต์ ของ คลาสซึ่งใช้ source code เดียวกันมีคุณลักษณะและคุณสมบัติที่แตกต่างกัน

5.8.2 Inheritance

คือการสืบทอดคุณสมบัติเป็นวิธีการสร้างคลาสย่อย ที่เรียกว่าซับคลาส (subclass) ซึ่งจะเป็นกำหนดประเภทของวัตถุให้จำเพาะเจาะจงขึ้น ซึ่ง ซับคลาสจะได้รับถ่ายทอดคุณสมบัติต่างๆ มาจากคลาสหลักด้วยไม่ว่าจะเป็น ตัวแปร หรือ ฟังก์ชัน

เมื่อเราสร้างคลาสย่อย(subclass) เราสามารถเพิ่มหรือเขียนทับ ตัวแปร หรือ ฟังก์ชันใดจากคลาสหลักได้ เราสามารถสร้างคลาสในภาษา CL ได้โดย

```
CL > class A:
...     pass
```

รูป 5.14 การสร้างคลาสแบบเริ่มต้น

```
CL > class B(A):
...     pass
```

รูป 5.15 การสร้างคลาสแบบโดย Single inheritance

```
CL > class D(B,C):
...     pass
```

รูป 5.16 การสร้างคลาสแบบโดย Multiple inheritance

```
class Transportation:
#   """Abstract base class"""

   def __init__( self, start, end, distance ):
#      if self.__class__ == Transportation:
#         raise NotImplementedError
      self.start = start
      self.end = end
      self.distance = distance

   def find_cost( self ):
      print ''
#     """Abstract method; derived classes must override"""
#     raise NotImplementedError


class Walk( Transportation ):

   def __init__( self, start, end, distance ):
      Transportation.__init__( self, start, end, distance)

   def find_cost( self ):
      return 0


class Taxi( Transportation ):

   def __init__( self, start, end, distance ):
      Transportation.__init__( self, start, end, distance)

   def find_cost( self ):
      return self.distance * 40


class Train( Transportation ):

   def __init__( self, start, end, distance, station_no ):
      Transportation.__init__( self, start, end, distance)
      self.station_no = station_no

   def find_cost( self ):
      return self.station_no * 5
```

รูป 5.17 ตัวอย่างการถ่ายทอดของ Class

สำหรับแนวความคิดในการสร้างคลาสให้ดูตามตัวอย่าง เช่น เรื่องเกี่ยวกับการเดินทาง จะมีคลาสหลัก (Base Class) คือ Transportation และมี คลาสย่อย (Sub class) คือ Walk, Taxi, Train เป็นต้น โดยที่ คลาสย่อย Walk, Taxi, Train นั้นถูกถ่ายทอด (inheritance) จากคลาสแม่ (base class) คือ Transportation หากเป็นการถ่ายทอดแบบ Multiple inheritance นั้นจะให้ลำดับความสำคัญต่อการถ่ายทอดด้วย เช่น

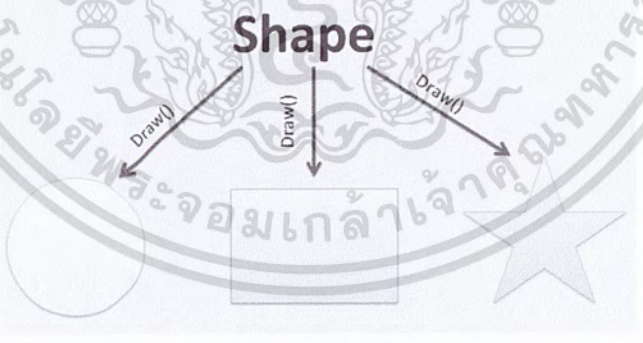
```
6class A:
7    a = 1
8    def __init__(self):
9        print 1
10
11class B:
12    a = 2
13
14class C(A,B):
15    c = 3
16    def __init__(self):
17        A.__init__()
18        print 2
19
20    def test(self):
21        return self.a
```

รูป 5.18 ตัวอย่างการถ่ายทอดแบบ Multiple inheritance

จากตัวอย่างคลาส C นั้นถูกถ่ายทอดมาจาก คลาสหลักคือ A,B ซึ่งพบว่ามีสมาชิกตัวเดียวกันคือ a แต่เนื่องจาก คลาส A มีลำดับความสำคัญมากกว่า B เมื่อมีการอ้างถึง a จากคลาส C นั้นจะได้ค่า a เท่ากับ 1

5.8.3 Polymorphism

Polymorphism หมายถึง การที่ Object มากกว่า 1 Object สามารถแสดงพฤติกรรมได้แตกต่างกันภายใต้ฟังก์ชัน (Method) เดียวกัน เรียกได้อีกอย่างหนึ่งว่าการเปลี่ยนรูป



รูป 5.19 หลักการของPolymorphism

จากรูปเป็นความสัมพันธ์ระหว่าง Class Shape ซึ่งแบ่งออกเป็น 3 ประเภท ได้แก่ Circle, Square และ Star แต่ละประเภทมีฟังก์ชัน Draw() ซึ่งได้รับการถ่ายทอดมาจากคลาสหลัก คือ Shape เหมือนกัน ดังนั้นเมื่อมีการเรียกใช้ฟังก์ชัน Draw() ส่งมาที่คลาสย่อย (Sub class) ทั้ง 3 ผลลัพธ์ที่ได้จะแตกต่างกัน ดังรูป

```
23class Food(object):
24    def __init__(self, name, calories):
25        self.name = name
26        self.calories = calories
27
28class HotDog(Food):
29    def tastesLike(self):
30        return "Extremely processed meat"
31
32class Hamburger(Food):
33    def tastesLike(self):
34        return "grilled goodness"
35
36class ChickenPatty(Food):
37    def tastesLike(self):
38        return "tastes like chicken"
39
```

รูป 5.20 ตัวอย่าง Code ภาษา CL ที่เป็นPolymorphism

5.8.4 Encapsulation

Encapsulation คือ การปิดบังข้อมูลเป็นวิธีการกำหนดสิทธิในการเข้าถึงข้อมูล หรือการกระทำกับอ็อบเจกต์ ของคลาสนั้นๆทำให้แน่ใจได้ว่าข้อมูลของอ็อบเจกต์นั้นจะถูกเปลี่ยนแปลงแก้ไขผ่านทาง methods หรือ properties ที่อนุญาตเท่านั้น (เช่นการกำหนดตำแหน่งทางการเมืองเป็น public method ที่ผู้อื่นสามารถกระทำได้ส่วนการลาออกจากตำแหน่ง เป็น private method ที่มีแต่อ็อบเจกต์ ของ คลาสเท่านั้นที่จะสามารถทำได้ แต่การกดดันและการขับไล่สามารถสร้าง data ที่อาจจะส่งผลเกิดการลาออกได้เช่นกัน)

5.8.5 Meta Class

Meta Class เป็นคลาสต้นแบบของคลาสทุกๆคลาสในภาษา CL ซึ่งทุกๆคลาสนั้นจะได้รับการถ่ายถอดจาก Meta Class โดยหากคลาสใดไม่ได้กำหนดว่าถ่ายทอดจากคลาสใด CL interpreter จะบังคับให้คลาสนั้นถูกถ่ายทอดมากจาก Meta Class ทั้งสิ้น

ภายใน Meta Class นั้นจะประกอบไปด้วยฟังก์ชันที่เป็นฟังก์ชันพิเศษของคลาส หรือฟังก์ชั่น Builtinsนั่นเอง นอกจากนั้นยังเก็บตัวแปร __name__ เป็นชื่อของคลาส __type__ เป็นชนิดของ namespace และ __inheritance__ ที่เก็บ Reference ของคลาสแม่อีกด้วยซึ่ง Meta Class จะเก็บเป็น None ไว้

```
dir(a.__inheritance__.__metaclass__.__builtins__)

{
'__lshift__': <function shifleftmethod at 0x02225F30>,
'__getslice__': <function getslicemethod at 0x02233170>,
'__str__': <function stringmethod at 0x022259F0>,
'__and__': <function andmethod at 0x02225FB0>,
'__abs__': <function absolutemethod at 0x02225EB0>,
'__inv__': <function invemethod at 0x02225EF0>,
'__rshift__': <function shifrightmethod at 0x02225F70>,
'__setattr__': <function setattributemethod at 0x02225AB0>,
'__delslice__': <function delslicemethod at 0x022331F0>,
'__getattr__': <function getattributemethod at 0x02225A70>,
'__cmp__': <function comparemethod at 0x02225B70>,
'__pos__': <function positivemethod at 0x02225E70>,
'__init__': <function constructor at 0x022259B0>,
'__call__': <function callmethod at 0x02225C30>,
'__len__': <function lenght at 0x02225CB0>,
'__repr__': <function representationmethod at 0x02225A30>,
'__getitem__': <function getitemmethod at 0x02225C70>,
'__del__': <function deletemethod at 0x02225B30>,
'__setitem__': <function setitemmethod at 0x022330F0>,
'__or__': <function ormethod at 0x02233090>,
'__add__': <function addmethod at 0x02225CF0>,
'__delitem__': <function delitemmethod at 0x02233130>,
'__nonzero__': <function nonzeromethod at 0x02225BF0>,
'__mod__': <function modmethod at 0x02225DF0>,
'__setslice__': <function setslicemethod at 0x022331B0>,
'__xor__': <function xormethod at 0x02233070>,
'__delattr__': <function deleteattributemethod at 0x02225AF0>,
'__div__': <function divmethod at 0x02225DB0>,
'__mul__': <function mulmethod at 0x02225D70>,
'__neg__': <function negativemethod at 0x02225E30>,
'__hash__': <function hashindexmethod at 0x02225BB0>,
'__sub__': <function submethod at 0x02225D30>,
'__not__': <function notmethod at 0x022330B0>
}
```

รูป 5.21 ฟังก์ชั่นพิเศษภายใน Meta Class

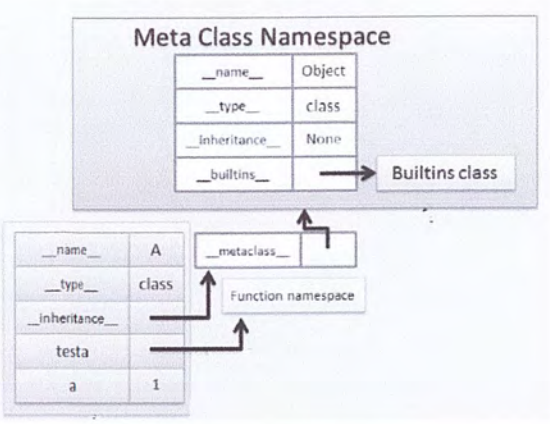
5.8.6 กระบวนการสร้าง Object

กระบวนการสร้าง Object นั้นเป็นกระบวนการที่สร้าง Namespace ของ Object ในแต่ละ Class ดังนั้นเราจะต้องมีความเข้าใจในรูปแบบการจัดเก็บและการอ้างอิง (Reference) ของ Namespace ในคลาส

```
Class A:
    a = 1
    def testa(self):
        print self.a
```

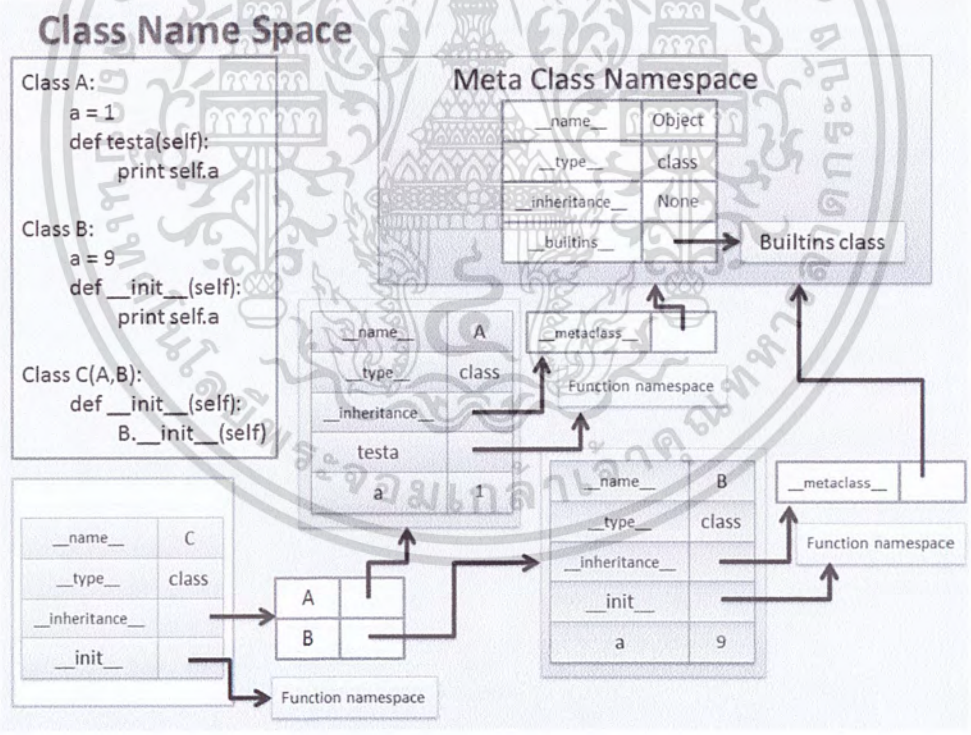
รูป 5.22 คำสั่งการนิยามคลาส

หากเราสร้างคลาสตามคำสั่งในรูปข้างต้นนี้ CL interpreter ซึ่งเป็นคลาสที่ไม่ได้กำหนดไว้ว่าถ่ายทอดจากคลาสใดเพราะฉะนั้นคลาส A นั้นจึงถ่ายทอดจาก Meta Class โดยตรง CL interpreter จึงสร้าง Reference ของ __inheritance__ ไปยัง ที่อยู่ของMeta Classดังรูปด้านล่างนี้



รูป 5.23 Namespace ของ Class

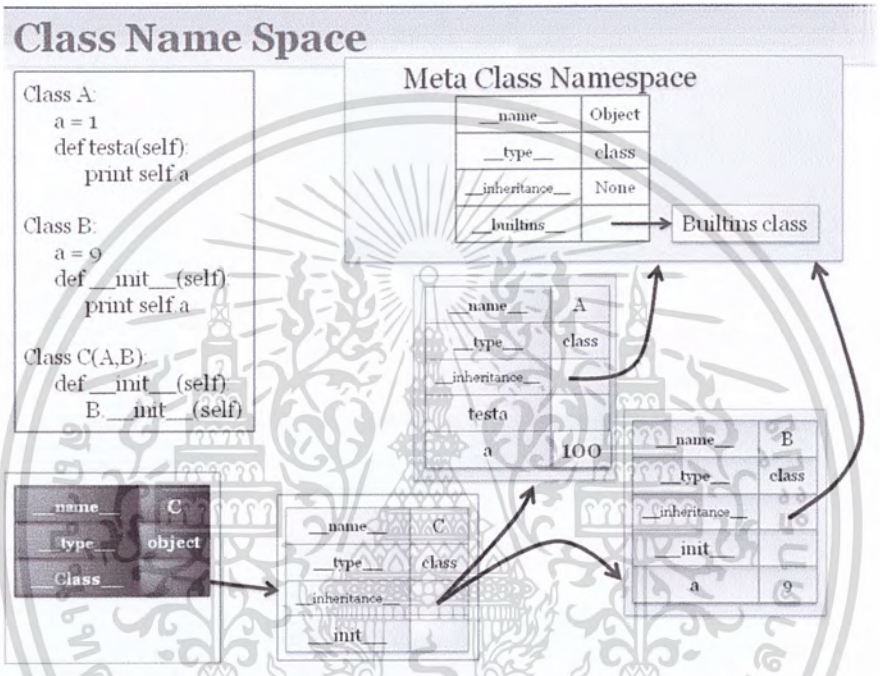
สมมติว่าเราได้นิยามคลาสดังรูป 5.24 นี้ CL interpreter จะจัดการระบบ Namespace ของ Class ต่าง เป็นดังรูป 5.24



รูป 5.24 Namespace ของ Class

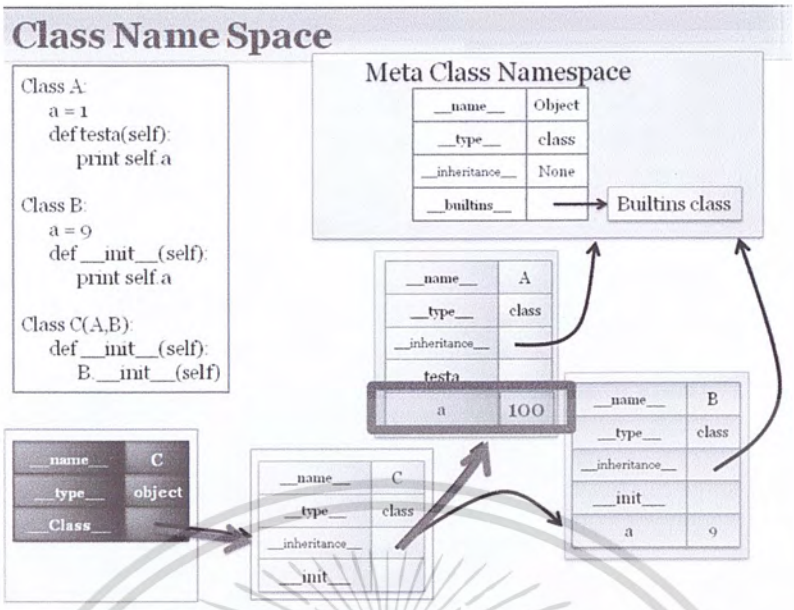
และเมื่อพบ Statement ที่สั่งให้ทำการสร้าง Object ขึ้นมา เช่น c = C( ) คือ การสร้าง Object ของคลาส C ซึ่งมีการถ่ายทอดเป็นแบบ Multiple inheritance โดยลำดับแรกนั้น CL

interpreter จะทำการสร้าง Namespace ของ Object ของคลาส C โดยมี Reference อ้างอิงไปยัง Namespace ของคลาส C แล้วทำการเรียก Constructor ของคลาส C ขึ้นมาหากไม่พบฟังก์ชัน Constructor CL interpreter จะไปเรียกของแม่ต่อไปเรื่อยๆ โดยปรกติแล้ว Constructor นั้นจะเป็นฟังก์ชั่นที่กำหนดค่าต่างๆให้กับ Object ซึ่งการกำหนดค่านั้นจะถูกบันทึกลงไปใน Namespace ของ Object เท่านั้นจะไม่มีการเปลี่ยนแปลงข้อมูลของคลาสแม่แต่อย่างใด



รูป 5.25 การสร้าง Object จาก Namespace ของคลาส

Object และ Class จะทำงานสัมพันธ์ซึ่งกันและกัน โดยหากเราต้องการเรียกใช้งานหรือเข้าถึงสมาชิกต่างๆภายใน Object เช่น คลาส C นั้นถ่ายทอดมาจากคลาส A ซึ่งมีสมาชิก a แล้วต้องการใช้งานตัวแปร a จาก Object c นั้น CL interpreter นั้นจะเข้าไปหา a ภายใน Namespace ของ Object C ก่อนหากไม่พบแล้วจะเข้าไปหา Namespace ของคลาส และ คลาสแม่ต่อไปเรื่อยๆ แต่หากมีการกำหนดค่า หรือ ฟังก์ชั่นให้กับ Object ใดๆ CL interpreter จะไปกำหนดใน Namespace ของ Object นั้นๆโดยจะไม่ส่งผลกระทบต่อข้อมูลใน Namespace ของคลาส และคลาสแม่



รูป 5.26 กระบวนการเข้าถึงข้อมูลของ Object และ Class

รูป 5.27 Object Namespace ของ Class C

5.8.7 Method Handling

Method หรือ Function ในคลาสนั้นจะถูกเก็บอยู่ใน Function namespace เช่นเดียวกันกับฟังก์ชัน ของภาษา CL ธรรมดา ใน Class namespace และ Object namespace นั้นจะเก็บ Reference ของ Function namespace ที่เป็นสมาชิกของตัวเอง ซึ่งการเรียกใช้นั้นจะถูกเรียกใช้งานผ่าน Object Namespace เท่านั้น เนื่องจากว่า ฟังก์ชันที่เป็นสมาชิกของ Class นั้น Argument ตัวแรก CL interpreter จะบังคับให้ Pass Reference ของ Object นั้นลงเข้าไปด้วย เพราะฉะนั้นการเรียกใช้ฟังก์ชันใน Class นั้น จะต้องทำการสร้าง Object ของ Class นั้นเสียก่อนจึงจะสามารถใช้งานฟังก์ชั่นนั้นได้

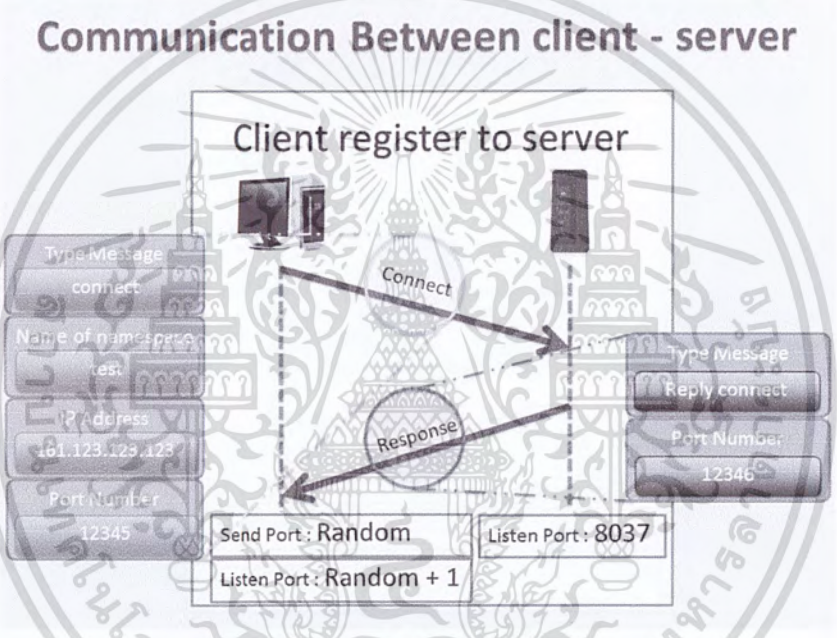
5.9 การติดต่อสื่อสารระหว่าง Client กับ Server

CL interpreter ได้ออกแบบไว้รองรับการทำงานเป็นในสถานะ Server และมี Module ที่ใช้ในการติดต่อกับฝั่ง Client ได้ซึ่งจะเป็นติดต่อสื่อสารโดยใช้การจัดรูปแบบ String เป็น Format ของ

XML เพื่อง่ายต่อการตีความหมายของข้อมูล รูปแบบการติดต่อสื่อสารระหว่างฝั่ง Server กับฝั่ง Client นั้นมีอยู่ 3 รูปแบบดังนี้

5.9.1 ฝั่ง Client ทำการ Connect ไปยังฝั่ง Server

การติดต่อสื่อสารในลักษณะนี้จะเกิดขึ้นเมื่อโปรแกรมทางฝั่ง Client เพิ่งเริ่มเปิดขึ้นมา หรือ เพิ่งต่อเข้ากับ Internet ได้ เพื่อเป็นการแจ้งสถานะของฝั่ง Client ว่า Client นี้มี IP Address อะไร Port ที่ฟังอยู่หมายเลขใดและใช้ Namespace อะไรในการประมวลผลคำสั่งภาษา CL



รูป 5.28 ฝั่ง Client ติดต่อไปยังฝั่ง Server

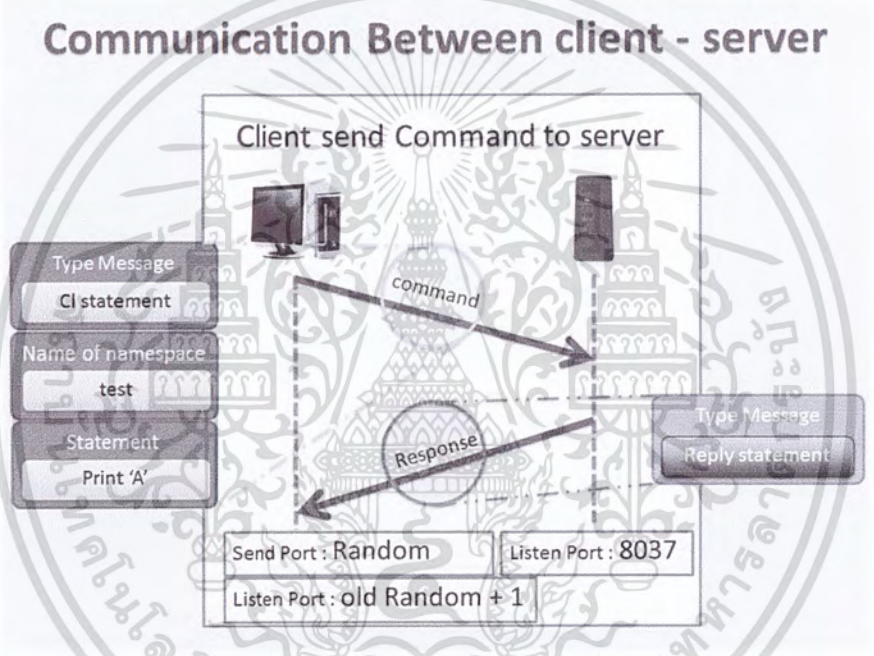
ขั้นตอนแรกเมื่อโปรแกรมในฝั่ง Client ได้ทำการเชื่อมต่อ Internet ได้แล้วนั้น โปรแกรมจะสร้าง Message ที่มีชื่อว่า Connect ขึ้นเพื่อส่งข้อมูลไปยังฝั่ง Server (ใช้ Port 8037 ในการฟังข้อมูล)ประกอบไปด้วย Type ของ message, Name of namespace ชื่อของ namespace ที่ใช้ปริการ, หมายเลข IP Address ของเครื่อง Client และ หมายเลข Port Number ที่ทำการ Random ขึ้นมาเพื่อใช้ในการส่ง Message นี้มา

เมื่อ CL interpreter ได้ Message นี้จะ Register ว่าได้มีเครื่อง Client หมายเข IP Address นี้ได้มีข้อมูลต่างๆที่ส่งมา หากพบว่า Namespace ที่ต้องการเรียกใช้นั้นไม่มีอยู่ใน Namespace ของ CL interpreter นั้น โปรแกรมทางฝั่ง Server จะทำการสร้าง Execute namespace ขึ้นมาใหม่เพื่อน

รองรับการทำงานจากฝั่ง Client เมื่อโปรแกรมทางฝั่ง Server ทำหน้าที่เสร็จแล้วCL interpreter จะตอบกลับมาโดยส่งหมายเลข Portnumber ที่จะต้องให้โปรแกรมฝั่ง Client เปิด Port ดังกล่าวเพื่อรอรับข้อมูลจาก CL interpreter โดยหมายเลข Port นี้จะเป็นหมายเลขที่เครื่อง Client Randomมาแล้วบวกอีกหนึ่งจึงเป็นหมายเลข Port ที่โปรแกรมฝั่ง Client ใช้กัน

5.9.2 ฝั่ง Client ทำการส่งคำสั่งไปยังฝั่ง Server

การติดต่อสื่อสารในลักษณะนี้จะเกิดขึ้นเมื่อโปรแกรมทางฝั่ง Client ได้ทำการส่งคำสั่งภาษา CL ไปยัง CL Interpreter ที่ฝั่ง server เพื่อประมวลผลคำสั่ง



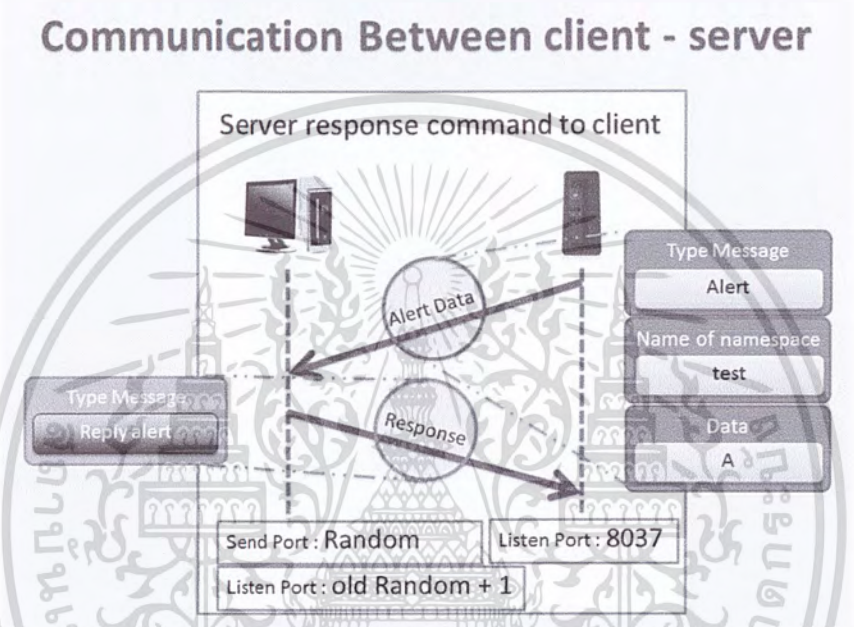
รูป 5.29 ฝั่ง Client ทำการส่งคำสั่งไปยังฝั่ง Server

การส่งคำสั่งจากฝั่ง Client นั้นจะเกิดหลังจากโปรแกรมในฝั่ง Client ได้ทำการ Register ไปยัง CL interpreter ในฝั่ง Server แล้ว Message ที่ Client ได้ส่งไปนั้นประกอบไปด้วย Type Message, ชื่อของ Namespace ที่ Client เรียกใช้งาน และ Statement ที่โปรแกรมฝั่ง Client ต้องการที่จะประมวลผล โดยจะทำการ Random Port ที่ใช้ในการส่งไปยัง Port 8037 ในฝั่ง Server ที่เปิดรอรับอยู่เมื่อได้รับคำสั่งเรียบร้อยแล้ว Cl interpreter ได้สร้างเธรดขึ้นมาเพื่อมารับผิดชอบในการประมวลคำสั่งนี้โดยคำสั่งจะถูกส่งไปยังฟังก์ชัน cl_statement_inputพร้อมกับ Execute namespace ของคำสั่งที่ได้เรียกใช้งาน เพื่อนำคำสั่งไปประมวลเป็น AST และ Execute ต่อไป เมื่อได้ส่งคำสั่งไป

ยังฟังก์ชัน cl_statemant_inputเรียบร้อยแล้ว CL interpreter จะตอบ Message กลับไปยัง Client นั้นตาม Port และ IP Address ที่ได้ทำการ Register ไว้ก่อนหน้านี้แล้ว

5.9.3 ฝั่ง Serverทำการตอบสนองคำสั่งและแสดงผลไปยังฝั่ง Client

การติดต่อสื่อสารในลักษณะนี้จะเกิดขึ้นเมื่อ CL interpreter ได้ทำการประมวลผลคำสั่งภาษา CL แล้วเกิดผลลัพธ์ที่ต้องแสดงในโปรแกรมทางฝั่ง Client



รูป 5.30 ฝั่ง Server ทำการตอบสนองคำสั่งและแสดงผลไปยังฝั่ง Client

ในระหว่างที่CL interpreter ประมวลคำสั่งอยู่นั้นได้เกิดผลลัพธ์ขึ้นและพบว่า Namespace ที่ใช้อยู่นั้นเป็น Namespace ที่โปรแกรมในฝั่ง Client ได้ทำการ Register เข้ามา CL interpreter จะทำการส่งผลลัพธ์ที่ได้นั้นโดยผ่านฟังก์ชั่น alert_runningเพื่อสร้าง Message ที่ประกอบไปด้วย Type Message, ชื่อของ Namespace ที่ใช้งาน และ ผลลัพธ์ที่ได้ แล้วผ่าน IP Address และหมายเลข Port ที่ Register ไว้เมื่อโปรแกรมทางฝั่ง Client ได้รับ Message แล้วจะตอบ Message กลับไปยัง CL interpreter ผ่าน Port 8037

5.10 สรุป

อินเตอร์พรีตเตอร์CL มีกระบวนการทำงานหลักๆอยู่2ขั้นตอน ได้แก่กระบวนการตีความหมาย และ กระบวนการประมวลผลคำสั่งโดยกระบวนการทั้งสองจะทำงานเป็นเธรดแบบ

อิสระต่อกันและเริ่มทำงานพร้อมกันตั้งแต่ตอนเริ่มระบบอินเตอร์พรีตเตอร์ และทำงานเรื่อยไป จนกระทั่งระบบอินเตอร์พรีตเตอร์เลิกทำงาน ในระหว่างการทำโปรแกรม CL อินเตอร์พรีตเตอร์จะมีการสร้างเธรด Code Interpreter เพื่อประมวลผล โปรแกรม จำนวนเธรดนี้จะขึ้นอยู่กับ จำนวนประโยคที่ต้องการให้ทำงานพร้อมๆ กัน

บทที่ 6
ผลการทดลองการใช้งานภาษา CL

จากที่กล่าวมาทั้งหมดจะเห็นว่าภาษา CL มีโครงสร้างภาษาที่สนับสนุนการทำงานคำสั่งได้เช่นเดียวกับภาษาคอมพิวเตอร์ทั่วไป อีกทั้งยังสามารถประมวลผลและจัดการการทำงานที่เกี่ยวข้องกับเวลาได้ นอกจากนั้นยังได้นำเสนอโครงสร้างข้อมูลสำหรับเก็บลำดับการทำงานของโปรแกรม CL รวมทั้งขั้นตอนการอินเตอร์พรีตโปรแกรม CL ดังนั้นในบทนี้เราจะทำการทดลองแนวคิดที่ได้เสนอมาแล้ว โดยทดลองอินเตอร์พรีต CL โปรแกรมแบบต่าง เพื่อดูโครงสร้างข้อมูลการจัดเก็บลำดับการทำงาน โดยได้แบ่งการทดลองออกเป็น 3 ส่วนคือ การทดลองเพื่อทดสอบ Syntax ,การทดลองเพื่อทดสอบเกี่ยวกับเวลา และการทดลองเกี่ยวกับ Graphic User Interface

6.1 การใช้งานอินเตอร์พรีตเตอร์ภาษา CL

อินเตอร์พรีตเตอร์ภาษา CL ที่พัฒนาขึ้นมีทำงานเป็นแบบ Interactive Interpreter ซึ่งคล้ายกับการสนทนาระหว่างผู้เขียนโปรแกรมกับอินเตอร์พรีตเตอร์ โดยในระหว่างการอินเตอร์พรีต เราสามารถติดต่อกับอินเตอร์พรีตเตอร์ได้ โดยเริ่มจากการรันไฟล์ cl_main.py

```
CL_interpreter Start !!!
>>>>>#Create parser INODE
>>>>>#Create timetable
Start Server side (Y/N) #Y
Client Handler Start !!!!
Start Comand line (Y/N) #Y
>>>>>#Shell Cmd Start >>>>>#Time Responder Start
>>>

CL >
CL >
CL >
```

รูป 6.1 เริ่มต้น CL interpreter

เมื่อเริ่มต้นอินเตอร์พรีตเตอร์ภาษา CL นั้นจะมีคำถามอยู่ 2 คำถามคือ 1. ต้องการเปิดบริการเป็น Server หรือ ไม่ 2. ต้องการให้มีระบบ Command line หรือไม่ ซึ่งคำถามที่ 2 ให้ตอบ Y เพื่อทำการทดลองในหน้า Interactive Interpreter ส่วนคำถามแรกให้ตอบ Y ในตอนที่ทำการทดลองกับ Graphic User Interface ซึ่งเป็นโปรแกรมฝั่ง Client ที่ติดต่อกับ CL interpreter ซึ่งอยู่ในฐานะ Server นอกจากนี้ยังมีคำสั่งต่างๆที่ที่ช่วยทำการทดลองดังนี้

ตาราง 6.1 คำสั่งพื้นฐานในหน้า Console ของ CL interpreter

คำสั่งของ Console	การทำงาน
show namespace	สั่งให้อินเตอร์พรีตเตอร์ CL แสดงตัวแปรGlobal ใน Namespaceที่มีทั้งหมด
show time	สั่งให้อินเตอร์พรีตเตอร์ CL แสดงเวลาปัจจุบัน
print ast	สั่งให้อินเตอร์พรีตเตอร์ CL แสดงAbstract Syntax Tree (AST : ซึ่งเก็บโครงสร้างของภาษา CL ที่ผ่านกระบวนการ Lexical Analysis)
print time	สั่งให้อินเตอร์พรีตเตอร์ CL แสดง Tree ที่เก็บเวลาที่ผู้ใช้ได้กำหนดเอาไว้แสดงผลออกมาทางหน้าจอ
run file	สั่งให้อินเตอร์พรีตเตอร์ CL อินเตอร์พรีตภาษา CL จากไฟล์ต่างๆ
dirr(namespace)	แสดงสมาชิกต่างๆภายใน Namespace นั้นๆ
list namespace	แสดง namespace ทั้งหมดที่มีอยู่ใน CL interpreter
change namespace	เปลี่ยน namespace ที่ให้ command line ใช้อยู่

```
CL > show namespace
['__builtins__', '__name__', '__connecttable__', '__type__', '__import__']
CL >
```

รูป 6.2 คำสั่ง show namespace

```
CL > list namespace
['Cmd', 'testCL/test1.cl']
CL >
```

รูป 6.3 คำสั่ง list namespace

```
CL > run file
Path file CL > testCL/test1.cl
#import test2 as testtest
#import test2 as fol
#import test2,test3
#import imt.test2

class A:
        a = 1
        def __init__(self,me):
                print me
                self.me  = me

class B:
        a = 2

class C(A,B):
        c = 3
        def __init__(self):
                A.__init__(5)

        def test(self,r):
                self.r = r
                return self.a

sa='ha'
#cc = C()
```

รูป 6.4 คำสั่ง show namespace

6.2 การทดลองเพื่อทดสอบ Syntax

6.2.1 การทดลองการใช้งาน Data type

1) List เปรียบเสมือน Array ในภาษาอื่นๆ

```
CL >
CL > testlist = [1,2,3,5,6,7]
CL > testlist
[1, 2, 3, 5, 6, 7]
CL > testlist[2]
3
CL >
```

รูป 6.5 ทดลองการใช้งาน list

2) Dictionary เป็น Data type ที่มีลักษณะคล้าย Array แต่แตกต่างตรงที่ใช้ Key ในการระบุในการเข้าถึงสมาชิก

```
CL > testdic = {}
CL > testdic
{}
CL > testdic['a'] = 9
CL > testdic['b'] = {'aa' : 4, 'bb' : 5}
CL > testdic
{'a': 9, 'b': {'aa': 4, 'bb': 5}}
CL >
```

รูป 6.6 ทดลองการใช้งาน dictionary

3) Timepoint เป็น Data type ที่ใช้ในการระบุจุดเวลาต่างๆ โดยพื้นฐานจะใช้ Data type ชนิด list เป็นพื้นฐานโดยการระบุเวลาใน timepoint นั้นจะต้องระบุตามลำดับคือ สมาชิกตัวแรกหมายถึง ปี ถัดมาคือ เดือน วัน ชั่วโมง นาที วินาที ตามลำดับ

```
CL > timepoint = [2011,2,14,20,0,30]
CL > nextime = [0,400,0,0,0,0]
CL > futuretime = timepoint addtime nextime
CL > futuretime
[2044, 6, 14, 20, 0, 30]
CL >
```

รูป 6.7 ทดลองการใช้งาน Timepoint

จากรูปเป็นตัวอย่างในการใช้ตัวแปร Timepoint โดยมีการสาธิตการบวกวันของตัวแปร Timepoint

6.2.2 การทดลองการกระทำทางคณิตศาสตร์

ในตัวอย่างนี้เป็นการแสดงการบวก-ลบเลขจำนวนเต็มกับทศนิยม โดยมีการ Assign ค่าตัวแปร มีการ Define function และสั่งให้มีการแสดง Abstract Syntax Tree เพื่อดูค่าที่เก็บ

```
CL > a=10
CL > b=5
CL > c=1.5
CL > d=2.5
CL > def add(a,b):
...      return a+b
...
CL > def minus(a,b):
...      return a-b
...
CL > printast
[cl-tree]
    [statement]
        [define-funct]
            minus
             [elementlist]
                 [id-exp]
                     a
                [id-exp]
                     b
             [stmt-list]
                [return-stmt]
                    [-]
                        [id-exp]
                            a
                        [id-exp]
                            b
```

รูป 6.8 การ Assign ค่าตัวแปร การDefine function และการแสดง Abstract Syntax Tree

จากนั้นทดลอง run เพื่อดูผลลัพธ์จะได้ผลดังรูป

```
CL > add(c,d)
4.0
```

ก)

```
CL > add(a,d)
12.5
```

ข)

```
CL > add(a,b)
15
```

ค)

รูป 6.9 ผลลัพธ์ของการบวก

ก) float กับ float

ข) int กับ float

ค) int กับ int

6.2.3 การทดลองการทำงานของฟังก์ชัน

ในตัวอย่างนี้จะแสดงฟังก์ชัน Factorial ซึ่งมีการเรียกใช้ Recursive

```
CL > def fac(n):
...     if n == 1:
...             return n
...     else:
...             return n*fac(n-1)
...
CL > fac(5)
120
CL > dirr(fac)
['__defaultarg__', '__type__', 'n', '__arg__', '
__return__', '__name__', '__code__', '__parent__
']
CL >
```

รูป 6.10 การ Define function factorial การเรียกใช้งาน fac(5) และ แสดง namespace ของ function

6.2.4 การทดลองการ Import ไฟล์โปรแกรมภาษา CL

การทดลองนี้จะทำการสร้างไฟล์สองไฟล์คือ test1.cl และ test2.cl โดยมี Code ต่างๆ ภายในไฟล์ดังนี้

```
1 import test2 as testtest
2
3 sa='ha'
4
5 testtest.test2_funciton(1,2)
6
7 def test1_function(a,b):
8     return a + b
9
```

รูป 6.11 ไฟล์ test1.cl

```
1  import test1
2
3  def test2_funciton(x,y):
4      return x+y
5
6  def taro1(x,y):
7      x
8      return x+y
9
10 i=1
11 y=2
12 a={'max':{'taro':2}}
13
14 if i==2:
15     i
16 else:
17     y
18
19 def fac(x):
20     if x==1:
21         return x
22     else:
23         return x*fac(x-1)
```

รูป 6.12 ไฟล์ test2.cl

```
CL > import test1
CL > show namespace
['test1', '__builtins__', '__import__', '__type__', '__name_
_', '__connecttable__']
CL > dirr(test1)
['A', 'C', 'B', 'testtest', '__type__', 'cc', 'test1_functio
n', '__name__', 'sa']
CL > dirr(test1.testtest)
['test1', 'a', 'i', '__type__', 'test2_funciton', 'taro1', '
fac', 'y', '__name__']
CL >
```

รูป 6.13 ผลลัพธ์การ import ไฟล์ test1

จากรูป 6.11 และ 6.12 ข้างต้นนี้เป็นการ import ไฟล์ test1.cl ซึ่งในไฟล์ test1.cl นั้น มีการ import ไฟล์ test2.cl แล้วเก็บไว้ใน key ที่มีชื่อว่า testtest และในไฟล์ test2.cl นั้นเองก็มีการ import ไฟล์ test1.cl กลับไปด้วยเช่นกัน โดยในรูป 6.13 นั้นเป็นการแสดง namespace ต่างๆตั้งแต่การแสดง Execute namespace ของ command line จะพบว่ามี test1 เป็น key ที่ใช้ในการอ้างถึง namespace ของไฟล์ test1.cl เมื่อเราพิมพ์ dirr(test1) เป็นการดู namespace ของ test1 ซึ่งเมื่อเข้าไปแล้วจะพบ namespace ของไฟล์ test1.cl จริงแสดงว่า key ใน namespace ของ command line เป็น key ที่เข้าถึง namespace ของไฟล์ test1.cl และใน namespace ของ test1 นั้นจะพบ key ที่มีชื่อว่า testtest นั้นซึ่งเป็น key ที่ในไฟล์ test1.cl นั้นกำหนดไว้อ้างถึง namespace ของไฟล์ test2.cl หากเราต้องการดู namespace ของไฟล์ test2.cl นั้นจะต้องมีการอ้างอิงผ่าน namespace ของไฟล์ test1.cl ก่อนจึงจะเข้าไปดูได้ โดยการพิมพ์คำสั่ง dirr(test1.testtest) ซึ่งจะพบ namespace ของไฟล์ test2.cl จริง และใน namespace ของไฟล์ test2.cl นั้นจะพบ key mมีชื่อว่า test1 ซึ่งเป็น keyที่ใช้อ้างอิงกลับไปยัง namespace ของไฟล์ test1.cl

```
CL > test1.sa
'ha'
CL > test1.test1_function(23,34)
57
CL > test1.sa
'ha'
CL > test1.testtest.test2_funciton(test1.sa,'nd')
'hand'
CL > test1.testtest.a
{'max': {'taro': 2}}
CL > test1.testtest.test1
< testCL/test1.cl is a import >
CL > test1.testtest.test1.sa
'ha'
CL >
```

รูป 6.14 ผลลัพธ์การใช้งาน Module ที่ Import เข้ามา

6.2.5การทดลองการทำงานแบบ Object Oriented Programming

6.2.5.1 ทดลองสร้าง Class และ Object แล้วทดลองใช้งาน

```
CL > class A:
...     aa = 1
...     def test_a(self):
...             self.bb = 8
...             print self
...     def __init__(self,b):
...             self.bb = b
...             print 'hello'
...
CL > dirr(A)
['aa', 'test_a', '__inheritance__', '__type__',
'__name__', '__init__']
CL > A.aa
1
CL > A.bb
NameError: name 'A' is not defined
CL > A.test_a()
test_a() takes exactly 1 arguments (0 given)
CL > a = A(3)
'hello'
CL > A.bb
NameError: name 'A' is not defined
CL > a.bb
3
CL > a.test_a()
< A is a object >
CL > a.bb
8
CL >
```

รูป 6.15 ผลลัพธ์การสร้าง Class และ Object และการใช้งาน

จากรูป 6.15 เป็นการสร้าง Class A โดยมีสมาชิก คือ aa มี test_a เป็นฟังก์ชันที่ print ตัวแปร self เมื่อสร้าง Class เสร็จแล้วจึงทำการทดลองดังนี้

1) ใช้คำสั่ง dirr(A) เพื่อดู namespace ของ Class A
2) พิมพ์ A.aa เป็นการเรียกตัวแปร aa ใน Class ซึ่งสามารถเรียกมาใช้งานได้เพราะเป็นสมาชิกของ Class ตั้งแต่ตอนสร้าง
3) พิมพ์ A.bb CL Interpreter ไม่สามารถเรียกใช้งาน bb ได้เพราะไม่สามารถหาตัวแปร bb ใน namespace ของ Class A จะใช้ได้ก็ต่อเมื่อทำการเรียกใช้ constructor ของ Class ก่อนจึงจะใช้งานได้
4) พิมพ์ A.test_a() ซึ่งเป็นการเรียกใช้ฟังก์ใน Class A ซึ่งไม่สามารถใช้งานได้ จะต้องใส่ argument อย่างนี้ 1 ตัวจึงจะใช้งานได้ แต่หากการเรียกใช้ฟังก์ชั่นใน Class นั้น CL interpreter จะบังคับใส่ argument 1 ตัวคือ Object ของตัวมันเอง
5) สร้าง Object ของ A โดยคำสั่ง a = A(3) ซึ่งจะเป็นการเรียกใช้งานฟังก์ชั่น __init__ หรือ constructor นั่นเอง

6) พิมพ์ A.bb ทดลองใช้งานซึ่งได้

7) เรียกใช้ฟังก์ชัน test_a() ในคลาสโดยผ่าน Object a ผลลัพธ์คือการแสดงผลลัพธ์ว่า argument ตัวแรกนั้นเป็น Object a จริง

6.2.5.2 ทดลองคุณสมบัติ Inheritance และ Polymorphism

```
CL > class A:
...     aa = 1
...     def test(self):
...             print 'test at A'
...     def __init__(self):
...             print 'init at A'
...
CL > class B(A):
...     bb = 2
...     def test(self):
...             print 'test at B'
...
CL > class C(A,B):
...     cc = 3
...     def __init__(self):
...             print 'init at C'
...
CL > A.aa
1
CL > B.aa
1
CL > B.bb
2
CL > C.aa
1
CL > C.bb
2
CL > C.cc
3
CL > a = A()
'init at A'
CL > b = B()
'init at A'
CL > c = C()
'init at C'
CL > a.test()
'test at A'
CL > b.test()
'test at B'
CL > c.test()
'test at A'
CL >
```

รูป 6.16 ผลลัพธ์การสร้าง Class และ Object และการใช้งาน

จากรูป 6.16 เป็นการสร้าง Class A , B , C โดยที่ Class B ถ่ายทอดมาจาก Class A และ Class C ถ่ายทอดมาจาก Class A และ B เมื่อสร้าง Class เสร็จแล้วจึงทำการทดลองดังนี้

1) ทำการใช้งานตัวแปร aa , bb , cc โดยการอ้างอิงผ่าน Class ต่างๆซึ่งพบว่าใช้งานได้เนื่องจาก หาก CL interpreter ไม่พบตัวแปรใน Class นั้นจะทำการค้นหาตัวไปที่คลาสแม่ไปเรื่อยๆ จนถึง Meta class หาไม่พบจะฟ้อง Error ออกมา

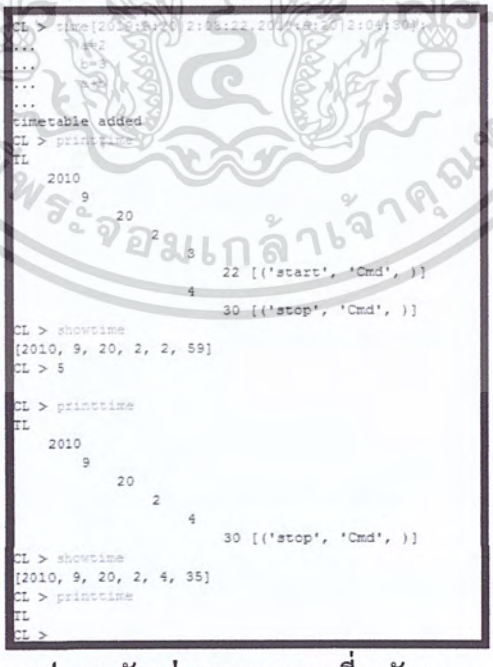
2) สร้าง Object ของ Class ต่างๆ ซึ่งเมื่อสร้าง Object จาก Class นั้นจะทำการเรียก constructor ของคลาสนั้นๆ หากไม่พบจะทำการเรียกใช้ในคลาสแม่

3) เรียกฟังก์ชั่น test ซึ่งมีอยู่ใน Class A และ B เมื่อเรียกฟังก์ชั่นผ่าน object c นั้น ปรากฏว่า ฟังก์ชั่น test ใน Class A นั้นถูกเรียกใช้งาน ซึ่งเป็นเพราะว่า ตอน Inheritance นั้นจะให้ลำดับความสำคัญของ Class A มาก่อน เนื่องจากพิมพ์ Class A ในส่วนของ Inheritance ขึ้นก่อน Class B

6.3 การทดลองเพื่อทดสอบการทำงานเกี่ยวกับเวลา

6.3.1 การทดลองกับเวลาที่กำหนดไว้แน่นอน

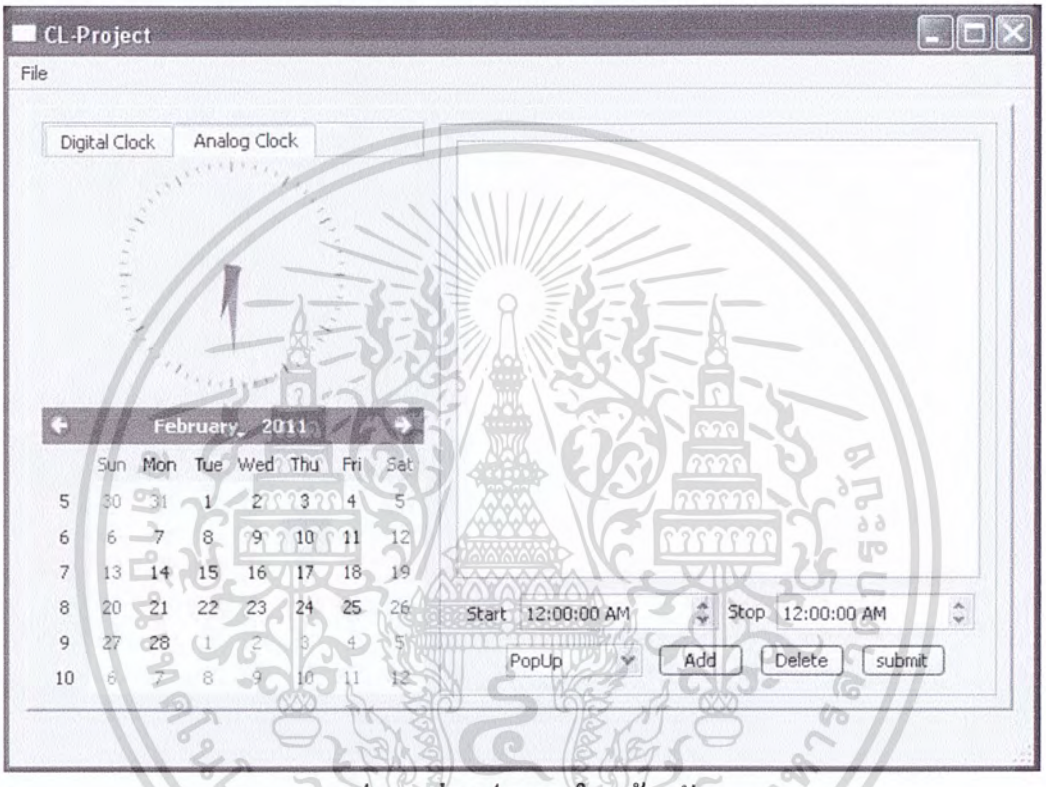
ในตัวอย่างนี้เป็นการแสดงการกระทำคำสั่งตามเวลาที่กำหนด โดยมีการกำหนดเวลาเริ่มต้น และเวลาสิ้นสุด จากนั้นเมื่อถึงเวลาที่กำหนดก็จะมีการกระทำตามคำสั่งนั้นๆ เช่น ในตัวอย่างสั่งให้ a+b ระหว่างเวลา 2 นาฬิกา 3 นาที 22 วินาที ของวันที่ 20 เดือน 9 ปี 2010 จนถึงเวลา 2 นาฬิกา 4 นาที 30 วินาที ของวันที่ 20 เดือน 9 ปี 2010 และเมื่อถึงเวลาที่กำหนดก็จะแสดงผลลัพธ์ a+b ออกมาคือค่า 5 จากนั้นเมื่อเลยเวลาเริ่มต้นไปแล้วจะทำการลบเวลาดังกล่าวออกไป ดังนั้นเมื่อเราสั่งคำสั่ง printtime ก็จะแสดงค่าเวลาสิ้นสุดออกมาเพียงอย่างเดียว ในทำนองเดียวกันเมื่อเลยเวลาสิ้นสุดไปแล้วจะทำการลบเวลาดังกล่าวออกไป ดังนั้นเมื่อเราสั่งคำสั่ง printtime ก็จะแสดงค่า root timeline หรือ TLออกมา



รูป 6.17 ตัวอย่างการทดสอบเกี่ยวกับเวลา

6.4 การทดลองเกี่ยวกับ Graphic User Interface

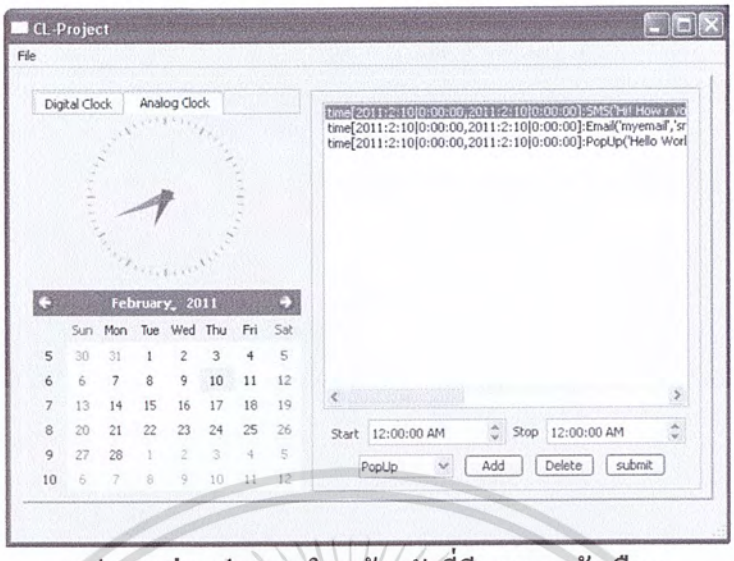
ในตัวอย่างนี้เป็นการแสดงส่วนติดต่อกับผู้ใช้ซึ่งเป็นโปรแกรมการจัดการส่วนบุคคล Personal organizer อย่างง่ายเพื่อทำการแจ้งเตือนตามกำหนดเวลา โดยกำหนดให้มีการแจ้งเตือนแบบ Pop-up , SMS และ Email โดยผู้ใช้สามารถใช้งานตามส่วนประกอบในหน้าหลักดังรูป ซึ่งมีรายละเอียดดังนี้



รูป 6.18 ส่วนประกอบในหน้าหลัก

6.4.1 รายละเอียดการใช้งาน

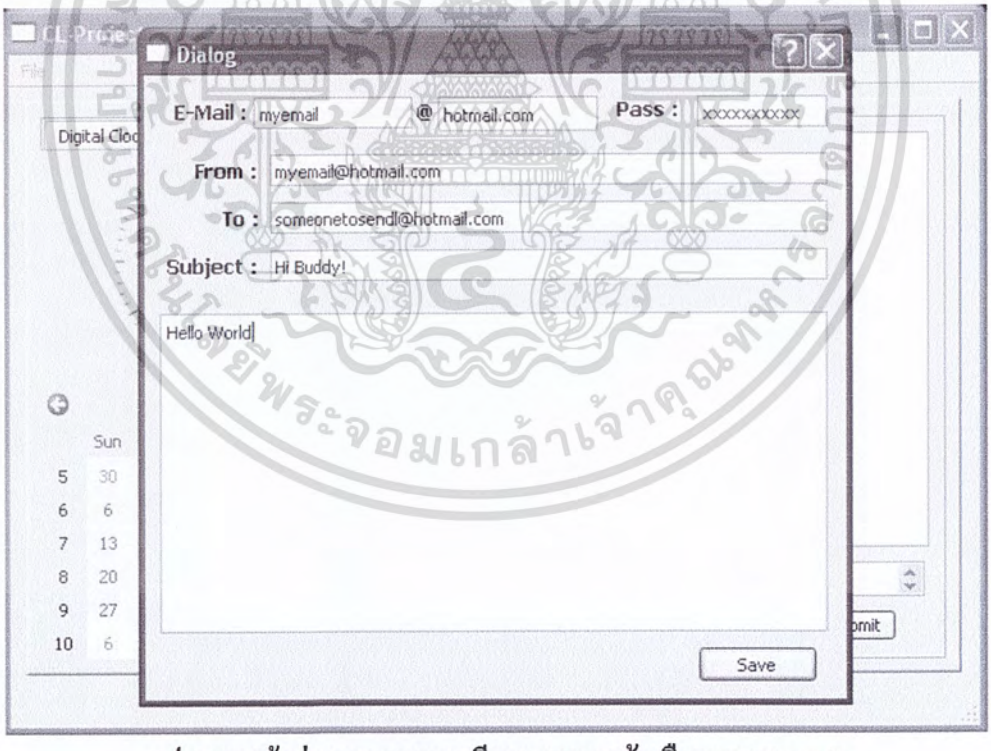
ในหน้าหลักจะพบว่าทางด้านซ้ายมีนาฬิกาที่แสดงเวลาปัจจุบันและปฏิทิน ทางด้านขวาจะเป็นส่วนที่โชว์การทำงานซึ่งตอนนี้ยังเป็นที่ว่าง ส่วนด้านล่างจะเป็น Time Edit ซึ่งเอาไว้กำหนดเวลาให้แจ้งเตือนโดยเลือกเวลาเริ่มต้นที่ Start และเลือกเวลาสิ้นสุดที่ Stop จากนั้นสามารถเลือกลักษณะการแจ้งเตือนได้ 3 ลักษณะคือ Pop-up , SMS และ Email เมื่อเลือกลักษณะการแจ้งเตือนได้แล้วกดปุ่ม Add เพื่อบันทึกการแจ้งเตือน ปุ่ม Delete มีไว้สำหรับลบรายการที่ไม่ใช้ และ ปุ่ม Submit เป็นปุ่มยืนยังการทำงาน เมื่อกดปุ่มนี้แล้วโปรแกรมนั้นจะทำการสร้าง Code ภาษา CL ขึ้น แล้วทำการส่งไปยัง CL interpreter ซึ่งเป็นฝั่ง Server เพื่อประมวลผมคำสั่งนั้นต่อไป เมื่อถึงกำหนดเวลาแล้ว CL interpreter ที่ทำงานในฝั่ง Server นั้นจะมี Message ส่งกลับมายัง โปรแกรมเพื่อทำการแจ้งเตือนต่อไป



รูป 6.19 ส่วนประกอบในหน้าหลักที่มีรายการแจ้งเตือน

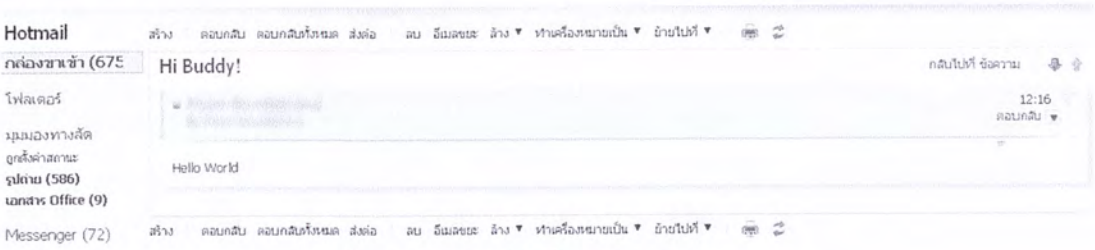
6.4.2 ทดลองใช้การแจ้งเตือนผ่าน E-mail

เมื่อเราเลือกการแจ้งเตือนแบบ E-mail โปรแกรมจะแสดงหน้าต่างดังรูปด้านล่าง



รูป 6.20 หน้าต่างกรอกรายละเอียดของการแจ้งเตือนทาง E-mail

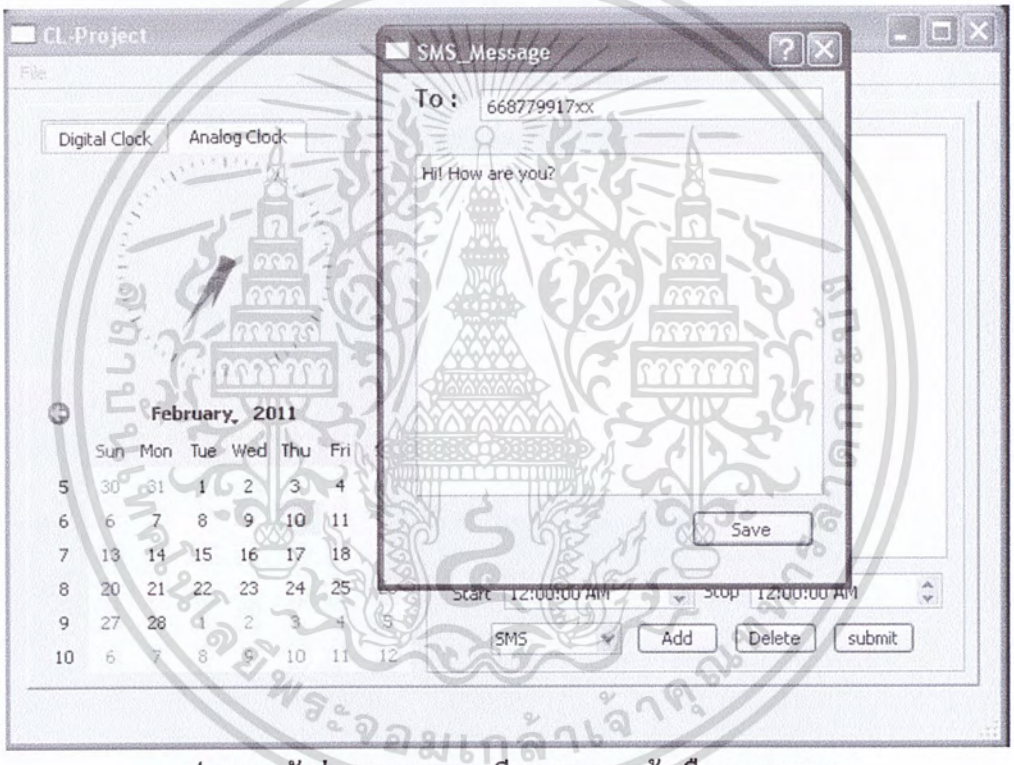
หลังจากกรอกรายละเอียดครบถ้วนแล้วให้กดปุ่ม save เมื่อถึงเวลาที่กำหนดไว้ CL interpreter ทางฝั่ง Server จะทำการส่ง E-mail ไปหาผู้รับดังรูป



รูป 6.21 ผลการแจ้งเตือนทาง E-mail

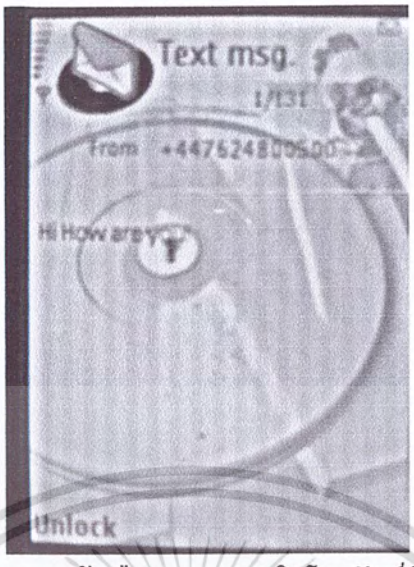
6.4.3 ทดลองใช้การแจ้งเตือนผ่าน SMS

เมื่อเราเลือกการแจ้งเตือนแบบ SMS โปรแกรมจะแสดงหน้าต่างดังรูปด้านล่าง



รูป 6.22 หน้าต่างกรอกรายละเอียดของการแจ้งเตือนทาง SMS

หลังจากกรอกรายละเอียดครบถ้วนแล้วให้กดปุ่ม save เมื่อถึงเวลาที่กำหนดไว้ CL interpreter ทางฝั่ง Server จะทำการส่ง SMS ไปตามเบอร์โทรศัพท์ได้ดังรูป

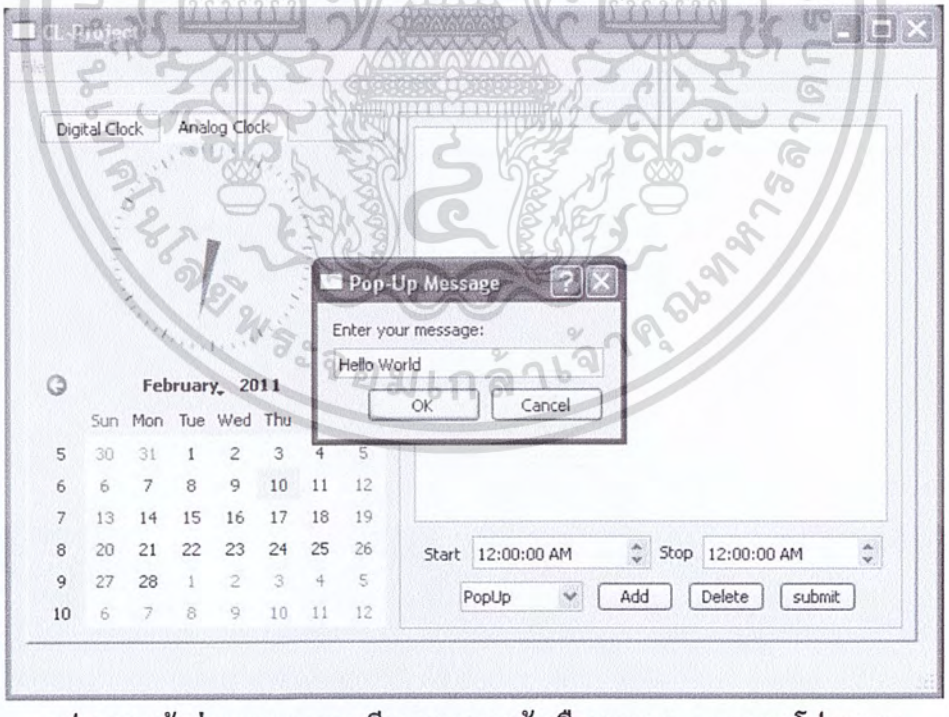


รูป 6.23 ผลการแจ้งเตือนทาง SMS ในโทรศัพท์มือถือผู้รับ

ซึ่งบริการทาง SMS นี้เป็นบริการของ textmagic

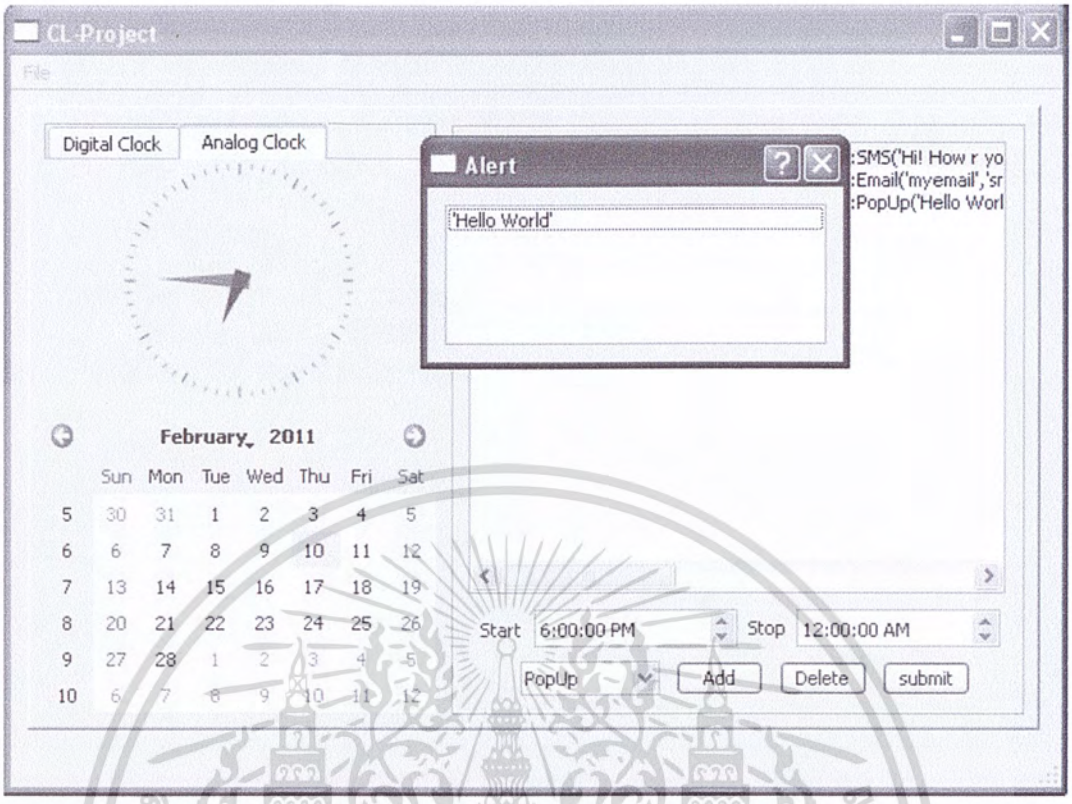
6.4.4 ทดลองใช้การแจ้งเตือนผ่าน Pop-up หน้า โปรแกรม

เมื่อเราเลือกการแจ้งเตือนแบบ Pop-up โปรแกรมจะแสดงหน้าต่างดังรูปด้านล่าง



รูป 6.24 หน้าต่างกรอกรายละเอียดของการแจ้งเตือนทาง Pop-up ของโปรแกรม

หลังจากกรอกรายละเอียดครบถ้วนแล้วให้กดปุ่ม Save เมื่อถึงเวลาที่กำหนดไว้ CL interpreter ทางฝั่ง Server จะทำการส่ง Message ไปยังโปรแกรมเพื่อทำการ Pop-up ดังรูป



รูป 6.25 หน้าต่างการแจ้งเตือนแบบ Pop-up ของโปรแกรม

บทที่ 7
สรุปผลการทำงาน และ ข้อเสนอแนะ

ในปริญญานิพนธ์นี้ได้นำเสนอการออกแบบ และพัฒนาอินเตอร์พรีตเตอร์ภาษา CL ให้มีรูปแบบของภาษาที่ง่ายต่อการเข้าใจและการใช้งาน คลอบคลุม Statement ทุกรูปแบบ พร้อมทั้งออกแบบส่วนประกอบที่เรียกว่า Namespace ที่ทำหน้าที่จัดเก็บค่าของตัวแปร นิยามของฟังก์ชัน และ นิยามของคลาส เพื่อรองรับการพัฒนาอินเตอร์พรีตเตอร์ ให้สามารถทำงานตามนิยามฟังก์ชันและคลาสได้ จนใกล้เคียงกับภาษาสากลในปัจจุบันนี้ โดยพัฒนาอินเตอร์พรีตเตอร์ภาษา CL ขึ้นจากภาษา Python ซึ่งการโปรแกรมภาษา CL จะกระทำบนภาษา Python อีกที เนื่องจากภาษา Python เป็นภาษาที่มีการทำงานแบบอินเตอร์พรีตเตอร์ (Interpreter) เหมาะสำหรับแนวคิดที่ต้องมีการตอบสนองต่อเหตุการณ์ภายนอกของภาษา CL อีกทั้งภาษา Python ยังมีไวยากรณ์ที่เข้าใจได้ง่าย ทำให้การพัฒนาภาษา CL เป็นไปได้สะดวกยิ่งขึ้น

โครงสร้างอินเตอร์พรีตเตอร์ CL มีการทำงานหลักๆ 2 ส่วน ส่วนของการตีความหมาย และส่วนของการประมวลผลคำสั่ง โดยขั้นตอนทั้งสองจะทำงานกันแบบอิสระต่อกันและเริ่มทำงานพร้อมกันตั้งแต่ตอนเริ่มระบบอินเตอร์พรีตเตอร์ และทำงานเรื่อยไปจนกระทั่งระบบอินเตอร์พรีตเตอร์เลิกทำงาน

7.1 ข้อจำกัดของอินเตอร์พรีตเตอร์ภาษา CL

ในการออกแบบและพัฒนาอินเตอร์พรีตเตอร์ภาษา CL นั้นยังขาดความสมบูรณ์ในเรื่องของ Builtins ฟังก์ชั่นซึ่งส่วนใหญ่สามารถทำงานได้แต่ Builtins ฟังก์ชั่นที่มีความหมายเฉพาะ เช่น isinsthance( ) เป็นต้น ซึ่งอินเตอร์พรีตเตอร์ยังไม่รองรับการทำงาน , อินเตอร์พรีตเตอร์ยังไม่สามารถทำงาน ในส่วนของฟังก์ชั่นพิเศษของคลาสได้อย่างครบถ้วน การทำงานตามเหตุการณ์ และการจัดการรูปแบบการประมวลผลคำสั่งตามเวลาทำได้เพียงประมวลผลเป็น Sequential แต่ยังขาดการทำงานแบบ Unordered, Parallel และ Alternative

7.2 ข้อเสนอแนะ

ในการพัฒนาอินเตอร์พรีตเตอร์ภาษา CL ให้ทำงานได้อย่างสมบูรณ์นั้นควรจะออกแบบอินเตอร์พรีตเตอร์ให้จัดการชนิดข้อมูลต่างๆ โดยจัดเก็บในระบบของคลาสและอ็อบเจ็คทั้งหมด ซึ่งจะออกแบบคลาสแต่ละคลาสในภาษา CL ให้แทนชนิดข้อมูลต่างๆของภาษา CL เมื่อมีการประกาศ

ตัวแปรขึ้น อินเตอร์พรีตเตอร์จะทำการสร้างอ็อปเจ็คขึ้นมารองข้อมูลของตัวแปรนั้นๆ เพื่อง่ายต่อการพัฒนารูปแบบของภาษา CL ดังนั้นควรพัฒนาอินเตอร์พรีตเตอร์ให้สามารถทำงานรองรับเรื่องฟังก์ชั่นพิเศษของคลาส และคุณสมบัติได้อย่างสมบูรณ์

7.3 แนวทางที่จะพัฒนาต่อไป

1) พัฒนาโปรแกรมจากภาษา CL ขึ้นมาใช้งานจริง
2) พัฒนาภาษา CL โดยเพิ่มแนวคิดของ Logical Statement ที่วิเคราะห์ความเป็นจริง เป็นเท็จเข้าไป
3) พัฒนาภาษา CL ที่ใช้เอเจนต์ในการติดต่อสื่อสาร Agent Communication Language.
4) ออกแบบอินเตอร์พรีตเตอร์ให้จัดการชนิดข้อมูลต่างๆ โดยจัดเก็บในระบบของคลาส และอ็อบเจ็คทั้งหมด

บรรณานุกรม

วชิระ ศิริพจนาวรรณ และ สุพัณณดา โชติพันธ์ . **"บราวเซอร์ที่โปรแกรมได้ "**. ปริญญานิพนธ์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยี เจ้าคุณทหารลาดกระบัง ปีการศึกษา 2543.

ธนวัฒน์ แก้วค้า และ บุญทวี สันติศรีวราภรณ์ . **"บราวเซอร์ที่โปรแกรมได้โดยใช้ภาษาไจธอน "**. ปริญญานิพนธ์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีเจ้าคุณทหารลาดกระบัง ปีการศึกษา 2544.

วิศิษฏ์ หิรัญกิตติ และคณะ . **"บราวเซอร์ที่สามารถโปรแกรมได้ "**. การประชุมวิทยาการและวิศวกรรมคอมพิวเตอร์แห่งชาติ ครั้งที่ 6 NCSEC2002. 2545.

ชาคริต เฮงสิริกุล และ เชิดเกียรติ แซ่แต้ . **"บราวเซอร์ที่โปรแกรมได้บนเครื่อง Pocket PC"**. ปริญญานิพนธ์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยี **เจ้าคุณทหารลาดกระบัง ปีการศึกษา 2546.**

วิศิษฏ์ หิรัญกิตติ และคณะ, **"บราวเซอร์ที่สามารถโปรแกรมได้ "**, การประชุมวิทยาการและวิศวกรรมคอมพิวเตอร์แห่งชาติ ครั้งที่ 6, 2545

วิศิษฏ์ หิรัญกิตติ และสุพัณณดา โชติพันธ์ **"CL: ภาษาสำหรับการสั่งงานคอมพิวเตอร์ด้วยเวลา และเหตุการณ์"**, การประชุมวิทยาการและวิศวกรรมคอมพิวเตอร์แห่งชาติ ครั้งที่ 7 , 2546.

สุพัณณดา โชติพันธ์ **"CL: ภาษาสำหรับการสั่งงานคอมพิวเตอร์ด้วยเวลาและเหตุการณ์"**, วิทยานิพนธ์ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง , 2547. (4.1)

Allen J., "Maintaining Knowledge about Temporal Intervals."**, Communications of ACM** 26(11), pp. 832-843, 1983 (2.1)

Allen J., **"Time and Time Again : The Many Ways to Represent Time"**, International Journal of Intelligent Systems, 6(4), pp. 341-355, July 1991. (2.2)

Allen J., **"Towards a General Theory of Action and Time"**, Artificial Intelligence, 23, pp. 123-154. 1984 (2.3)

A. Galton., **"A Critical Examination of Allen's Theory of Action and Time",** Artificial - Intelligence, 42, pp. 159-188, 1990. (2.4)

A. Galton, **"Towards a Qualitative Theory of Movement"**, [online] Available: http://citeseer.ist.psu.edu/372.html

Wikipedia, **"Real-Time Computing"** [online] Available: http://en.wikipedia.org/wiki/Real-time

K. Prouskas and J. Pitt., **"A Real-Time Architecture for Time-Aware Agents"** IEEE transactions on systems, man, and cybernetics , IEEE Explorer, pp. 1553-1568, 2004.

George Varghese and Anthony Lauck, **"Hashed and Hierarchical Timing Wheels: Efficient Data Structures for Implementing a Timer Facility"**, IEEE/ACM Transaction on networking, vol. 5, no. 6, pp 824-834, December 1997

Beazley D.M. **Python Essential Reference**. : New Riders. 1999.

Chun W. J. **Core Python Programming**. : Prentice Hall. Upper Saddle River. 2001.

Daniel P. Bovet & Marco Cesati , **Understanding the Linu Kernel:** O'reilly Sebstopol, CA USA January 2001