PPL
Final

Edited 1
12/05/2017

By AeAe & ดีอาร์อีเอเอ็ม

Contents

Data type ชนิดของข้อมูล มี 3 ชนิด

Primitive type ข้อมูลพื้นฐาน มี 3 ชนิด

Boolean ข้อมูลค่าความจริง (true / false)

operator : and, or, not

Char/String ข้อมูลตัวอักษร

สามารถใช้ เปรียบเทียบ, เชื่อม, substring ได้

Numeric ข้อมูลตัวเลข มี 3 ชนิด

Integer ข้อมูลจำนวนเต็ม

ค่าสูงสุดขึ้นอยู่กับ ภาษา / เครื่อง

ถ้า + - * เกินกว่า range ที่เก็บได้ => execute error

Fixed-Point ข้อมูลจำนวนจริงแบบระบุจำนวนทศนิยม

ประกอบด้วย **scale** จำนวนหลักทศนิยม

precision จำนวนเลขทั้งหมด

ถ้าคำนวณแล้ว scale มากกว่าที่จะเก็บได้ จะปัดทิ้ง(หรือตัดทิ้ง)

ถ้าคำนวณแล้วหน้า . มากกว่าที่จะเก็บได้ จะ error

Floating-Point ข้อมูลจำนวนจริงแบบไม่ระบุจำนวนทศนิยม

ex. 3.1452E2

ประกอบด้วย **faction** ex. 3.1452, 1.5502

base ex. 10, e, E

exponent ex. 2, -3

Array type ชุดข้อมูลแบบ array

ข้อมูลที่เก็บต้องมี type เดียวกัน

มีหลายมิติได้

เข้าถึงข้อมูลแต่ล่ะตัวโดยใช้ index / subscript

การ assign array (ex. A = B) array ทั้งสองตัวต้องมีขนาดเท่ากัน

Record type ชุดข้อมูลแบบ record

ข้อมูลที่เก็บ type ต่างกันได้

ex. struct ใน C

Variant Record

- recordเดียวกัน แต่มีโครงสร้างต่างกันได้ ขึ้นกับtag

ex. union ใน C

* การที่ operator ตัวนึง ทำงานต่างกันกับ data type ที่ต่างกัน เรียกว่า **Overloading of operator**

ex. operator + เมื่อทำงานกับ numeric จะเป็นการบวก แต่ถ้าทำงานกับ string จะเป็นการเชื่อม

Type checking การตรวจสอบชนิดของตัวแปร

Static type checking รู้ type ของตัวแปรตอน compile

Dynamic type checking รู้ type ของตัวแปรตอน execute

Control Structure

Flow graph ประกอบด้วย

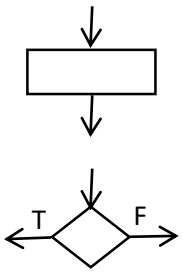
node มี 2 แบบ

basic action ใช้กับคำสั่งทั่วๆไป

เข้า 1 ออก 1

condition ใช้กับคำสั่งเงื่อนไข

เข้า 1 ออก T กับ F



connection เส้นเชื่อม

D-structure

basic action มี statement เดียว

sequence มีหลาย statement

condition มีเงื่อนไข

iterative loop *ต้องให้ T อยู่ซ้ายเสมอ

* D-structure เป็น one-entry one-exit

Proper Program คือโปรแกรมที่มีคุณสมบัติ

1. one entry และ one exit
2. ทุก node ต้องมาจากจุดเริ่มต้นได้

   ทุก node ต้องไปยังจุดจบได้

   (ทุกnodeต้องมีจุดเริ่มต้นและจุดสิ้นสุด)

Equivalent Program คือโปรแกรมที่ input เหมือนกันจะได้ output เหมือนกัน แต่ flowgraph อาจไม่เหมือนกัน

Higher Level Control Structure

goto goto i ไปบรรทัดที่ i

exit exit(i) ออกจาก loop จำนวน i loop

cycle cycle(i) กลับไปจุดเริ่มต้น loop จำนวน i loop

Subprogram มี 2 แบบ

Procedure ไม่มีการ return ค่ากลับ

Function มีการ return ค่ากลับ

Parameter Mode

Pass by value

parameter type : input

argument : เป็นได้ทั้ง expression, variable หรืออะไรก็ได้ที่มีค่า

ส่งค่าไปอย่างเดียว

Pass by result

parameter type : output

argument : เป็นได้แค่ variable

ส่งที่อยู่ไปให้อย่างเดียว เมื่อทำงานเสร็จจะ copy ค่ากลับมาที่ตำแหน่งเดิม

* parameter จะไม่ได้ค่าจาก argument เพราะงั้นถ้าใช้โดยไม่ได้ initialize จะ error

Pass by value-result

parameter type : update

argument : เป็นได้แค่ variable

ส่งที่อยู่และค่าของตัวแปรไปให้ เมื่อทำงานเสร็จจะ copy ค่ากลับมาที่ตำแหน่งเดิม

Pass by reference / location

parameter type : update

argument : เป็นได้แค่ variable

ส่งที่อยู่ไปให้ เวลาทำงานจะใช้ memory(ที่อยู่) เดียวกันเลย ดังนั้นจะไม่มีการ copy ค่ากลับ

* ผลการทำงานจะเหมือน pass by value-result แต่จะเร็วกว่าเพราะไม่ต้อง copy

* **aliases** คือ parameter 2 ตัว อ้างถึงaddressเดียวกัน

Pass by name

parameter type : update

argument : เป็นได้แค่ variable

copy code จาก procedure มาไว้ใน caller

Pass by constant

parameter type : input

argument : เป็นอะไรก็ได้ที่มีค่า

ส่งค่าไป แต่ห้ามเปลี่ยนแปลงตัวแปรที่รับค่าไป

* เหมือน pass by value แต่จะ memory เดียวกัน ทำให้ไม่เสียเวลาตอน copy ค่าไป

Evaluate argument location

แบบ evaluate ก่อนเรียก procedure

แบบ evaluate หลังเรียก procedure

dummy variable

ชื่อ parameter ที่ต่างกัน แต่การทำงานในprocedureเหมือนกัน

Function

PL/I ใช้ RETURN expression;

FORTRAN ใช้ FUN = expression;

RETURN

Pascal ใช้ FUN = expression;

#FUN คือชื่อของFUNCTION

Side effect ผลข้างเคียงของการใช้ function

1. การเปลี่ยนค่าของ argument

แก้โดยให้ parameter เป็นแบบ read-only

2. การเปลี่ยนค่าของ local static variable ใน function ส่งผลต่อการทำงานในอนาคต

แก้โดยให้ internal static variable ใน function เป็นแบบ read-only หรือ constant

3. การเปลี่ยนค่าของ global variable จาก function

แก้โดยให้ global variable เป็นแบบ read-only เมื่อใช้ในฟังก์ชั่น

4. การทำ input/output ใน function

ไม่ทำ input/output ใน function

บางครั้งก็อาจจงใจทำให้เกิด side effect ได้ เช่น

1. การพิมพ์ error message

2. random number generator ทำการแก้ไข static variable

3. function สำหรับทำ input/output

การทำงานของ subprogram

1. แบบ subprocedure จะมี main procedure เรียก subprocedure และจะทำงาน sub ให้จบค่อยกลับมาทำที่ main ต่อ

2. แบบ coroutine จะมี procedure 2 ตัวผลัดกันทำงานไปพร้อมๆกัน

Scope

ส่วนของโปรแกรมที่สามารถอ้างถึงตัวแปรใน declaration นั้นได้

Internal & External identifier

1. identifier ที่ declare ใน block นั้น เรียกว่า **Internal identifier** หรือ **Local identifier**

2. identifier ที่ declare ใน block ที่สูงกว่า และไม่ถูก redeclare จะเรียกว่า **External identifier** หรือ **Global**

Procedure Local Variable

variable ที่สร้างใน procedure จะยังไม่ถูกสร้างจนกว่า procedure นั้นจะถูกเรียก และจะใช้ได้จน procedure นั้นทำงานเสร็จ

Scope declaration structure

การกำหนด scope ของ declaration (ขอบเขตของตัวแปรที่ประกาศไว้) มี 2 แบบ คือ

Static structure

- ตัวแปรสามารถใช้ได้ภายใน block ที่ประกาศ
- ตัวแปรสามารใช้ได้ใน block ที่อยู่ต่ำกว่า ถ้าไม่ถูก redeclaration

* การหาตัวแปรจะหาจาก block ที่ใกล้ที่สุดไล่ขึ้นไปเรื่อยๆ
* structure นี้ช่วยป้องกันการใช้ variable และ procedure จากนอก block ได้
* scope สามารถรู้ได้ตอน compile

Dynamic structure

- ตัวแปรสามารถใช้ได้ภายใน block ที่ประกาศ
- ตัวแปรสามารถใช้ใน procedure ที่อยู่ใน stack ที่สูงกว่าได้

* การหาตัวแปรจะหาจาก caller ไล่กลับไปเรื่อยๆ
* structure แบบนี้ไม่สามารถป้องกันการใช้ variable และ procedure จากนอก block ได้
* scope สามารถรู้ได้ตอน execute

- ตัวแปรจะไม่เก็บค่าระหว่าง procedure call ทำให้ใช้ memory ได้อย่างมีประสิทธิภาพ

Activation record

- การเก็บข้อมูลของ procedure ที่ทำงานไว้เรียกว่า **activation record** ซึ่งจะเก็บไว้ใน stack
  - ตัวแปรที่ declare ใน block นี้
  - ข้อมูลเกี่ยวกับ parameter
  - temporary storage สำหรับไว้ใช้คำนวณ expression
  - return address
  - addressing information สำหรับตัวแปรที่ไม่ได้ประกาศใน block นี้
  - pointer ชี้ไปที่ activation record ของ caller

Storage Management

สามารถallocate memoryได้3แบบ

1. เวลา load program จะเก็บ program ไว้ใน storage
2. เวลาเข้าblock เรียก dynamic storage allocation
3. ระบุโดยใช้คำสั่ง/function

Lifetime ของตัวแปร

ระยะเวลาที่ตัวแปรจะอยู่ใน memory มีอยู่ด้วยกัน 3 แบบ

1. ให้ memory แก่ตัวแปรตลอดโปรแกรม

    เกิด : เริ่มโปรแกรม ตาย : จบโปรแกรม

2. dynamic storage allocation

    เกิด : เข้า block ตาย : จบ block

3. dynamic under program control ใช้การเขียนโปรแกรม(คำสั่ง)ควบคุม lifetime

    เกิด : จอง memory ตาย : คืน memory

Mini-Language Apply

- ตัวอย่างสำหรับ Functional Language

Highlight

<program> ::= **program**
<expression>
[<function-definition>]...
end

- รูปแบบของภาษานี้คือ 1 โปรแกรมจะมีหลายๆ function แต่จะมีเพียง 1 expression
- expression จะเรียกใช้งาน function ต่างๆที่กำหนดไว้ จากนั้นจะ output คำตอบออกมา

* output ให้เองไม่ต้องมีคำสั่ง output

<function-definition> ::= **where**<identifier>**(**<identifier>[**,**<identifier>]...**) is**
<expression>

- การสร้าง function จะประกาศชื่อแล้วตามด้วย parameter ในวงเล็บ และจะ return ค่าหลัง is

<selector-expression> ::= **select**
<condition> **=>** <expression>
[<condition> **=>** <expression>]...
[else **=>** <expression>]
end select

- คล้ายๆ if-else ดูตัวอย่างเอา

Built in function

- ฟังก์ชั่นที่มีให้ในระบบ

first(list)	return element ตัวแรกของ list
last(list)	return element ตัวสุดท้ายของ list
tail(list)	return list ที่ลบ element ตัวแรกออก
stem(list)	return list ที่ลบ element ตัวสุดท้ายออก
append(list, integer)	return list ที่เพิ่ม integer เข้าไปที่ตัวสุดท้าย
length(list)	return จำนวน element ของ list

*** Functional Language** ไม่มีคำสั่งloop, มีแต่recursion

Domain & Range

y = f(x)

- domain ชนิดของargument(x)
- range ชนิดของค่าที่ได้จากfunction