

Names and Bindings

Name is Symbolic identifier used to refer to variable, constant, operation, type etc., instead of referring to low-level concepts like address or fragment of code

ສັບຕິດສຳ

Binding is An association between a name and the thing it names

Binding time is Time at which an association is created (or time at which any implementation decision is made) *ການສ້າງອີງຕະຫຼາດເປັນແບບ* *C, C++, JAVA*

- **Static binding** = Things are bound before run time (early binding) → *ຈະໃນ flexible ເພິ່ມຂຶ້ນ
data type*
- Associated with **greater efficiency**, e.g. compiler decides on layout of variables in memory and generates efficient code to access them.
- *Compiled languages tend to have early binding times.* *ດີກວ່າ Run time* *ການຈະສ້າງອີງຕະຫຼາດ*
- **Dynamic binding** = Things are bound at run time (late binding)
ເຊື້ອ! ແລກປໍ່ໄດ້ໂປຣແກຣມ data type
- Associated with **greater flexibility**, e.g. decision on which data value of which type is bound to a variable name may be made at run time
- *Interpreted languages tend to have later binding times.* → *ສິ!*
ເກມກົດໃໝ່ Compiler

We will talk about binding of the identifiers to the variables they name.

2110216 DDOS 1

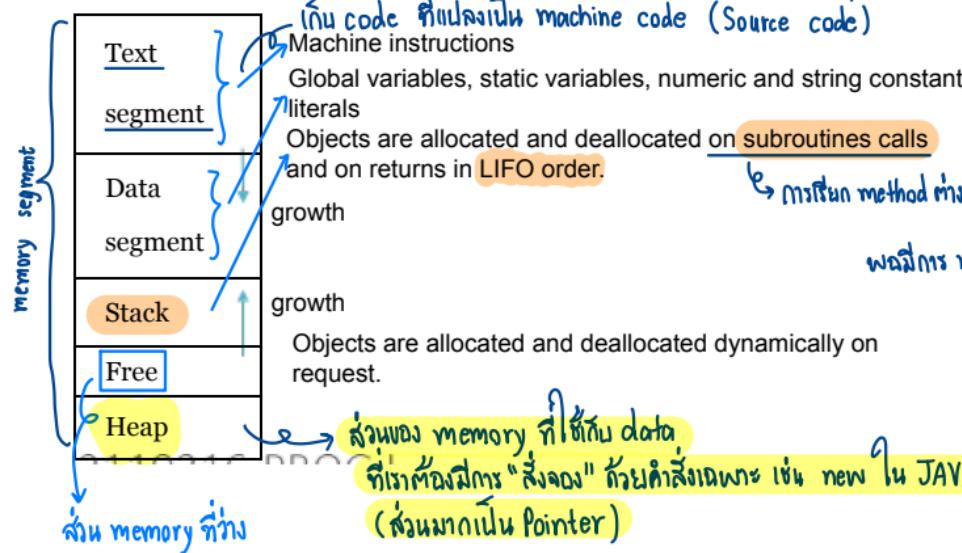
Object Lifetime and Storage Management

Lifetime is time between creation and destruction.

An object to which a name is bound has its object lifetime.

Object lifetime corresponds to its storage allocation

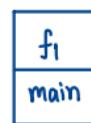
พื้นที่ที่ต้องการจะเป็น ชื่อที่ เครื่องพิมพ์



Fixed size. Static objects are given an absolute address and retained throughout program execution.

การเรียก method ต่างๆ

การเรียก method ที่ f1



ส่วนของ memory ที่ใช้กัน data
ที่เราต้องมาการ "ส่งจด" หัวไปดำเนินการพิมพ์ เช่น new ใน JAVA
(ส่วนของ memory ที่ใช้กัน Pointer)

Register ຕຳຫັນ Cache

ເງື່ອງໃຈ
ຫຼຸດລາຍລະອຽດ

Stack-Based Allocation

* ຂູ້ນໍາລົມບອນໄດ້ກຳປັກຮົມຂອງທີ່ push ອີ່ stack

Save state ຈະເຮັດກ່າວ Frame



Each instance of a called subroutine at run time has its own **frame** (or activation record) on stack which contains

- Arguments (if any)
- Return address
- Bookkeeping** : Saved values of registers, reference to other frame (see later) etc.
- Local variables (if any) → ສັກແປກໃຫ້ກຳຕົວໃຫ້ method ອີ່ນີ້
- Temporaries : Intermediate values produced in complex calculations

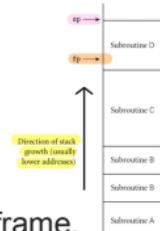
Stack pointer (sp) is a register for the address of the top of stack.

register ຕີ່ເປັນ frame ປຶ້ມບັນທຶກ frame

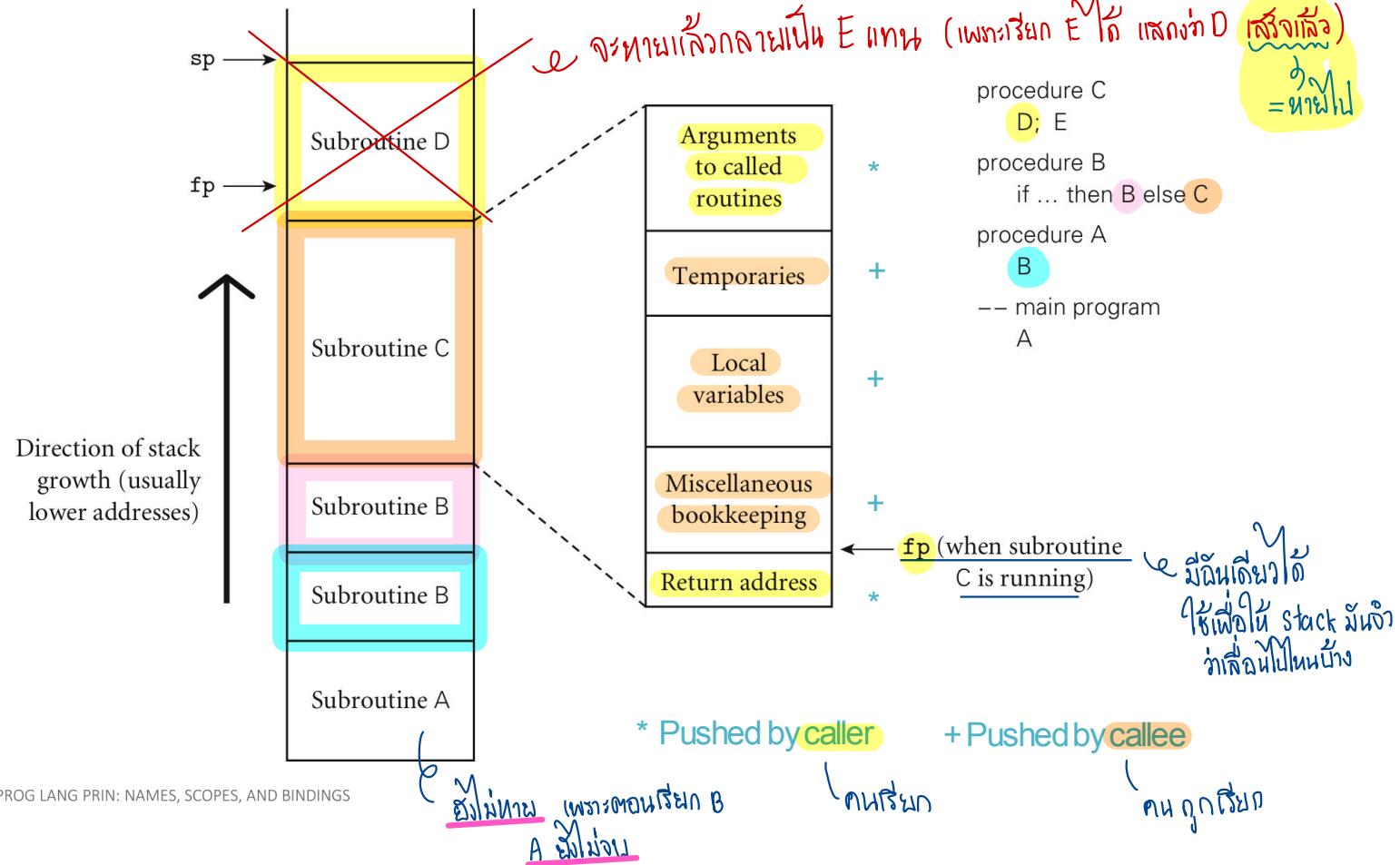
Compiler cannot tell the location of a frame but the offsets of objects in a frame.

- Frame pointer (fp)** is a register that points to a known (reference) location within a frame of the current subroutine. → ດີ້ທີ່ກົດຕົວໃຫ້ frame ວິຊົວ
- Code to access data within a frame adds a predetermined negative or positive offset to the value in fp.

2110216 DDOS 1



Calling Chain and Stack Frames



Maintenance of Stack

Compiler generates the following:

| Caller คaller | |
|--------------------------|---|
| ... jump ไปที่ method | ↑ หมายเหตุที่บันทึกไว้ใน stack |
| Pre-call | Push <u>arguments</u> onto the <u>stack</u> . ↑ หมายเหตุที่บันทึกไว้ใน stack |
| jsr callee | Call subroutine. This also <u>pushes the return address onto the stack</u> . |
| <u>Post-call</u> | <u>Deallocate the stack space</u> it allocated in the <u>pre-call</u> (adding positive offset to sp), and <u>continue at the address of the instruction immediately after the jsr</u> . |
| ... | |

| Callee callee | |
|---------------|--|
| Prologue | Push <u>old fp value</u> and update it to a <u>new frame</u> , push other <u>register values</u> that should not be changed, and push local variables (if any) onto the stack. ↑ หมายเหตุที่บันทึกไว้ใน stack |
| Main Body | Execute and store return value in a register (or in a location in the stack, if reserved by caller). |
| Epilogue | Restore the saved register values, and deallocate the stack space it allocated in the <u>prologue</u> (adding positive offset to sp). ↑ หมายเหตุที่บันทึกไว้ใน stack |
| rts | Return to caller, jumping back to the return address. |

N Naphat to Everyone
old fp กับ return address ตำแหน่งนี้มีครับ

Vishnu Kotajras to Everyone
return address มันใช้เก็บ เบร์ก point ที่ต้องกลับไป execute นะ คือสิ่ง ส่วนที่ใน source code
แล้ว old fp ใช้เก็บ marker ของตำแหน่ง

Vishnu Kotajras to Everyone
เป็นการเก็บค่าและอย่างไร สำหรับสิ่งที่จะ execute มันคือ สายที่เก็บไว้ใน main text ที่พูดไปในต้นข้อความ

N Naphat to Everyone
คือทำหน้าที่คล้ายกันแต่ทำงานคนละส่วนกันผมเข้าใจ ถูกมั้ยครับ

Vishnu Kotajras to Everyone
ใช่ เอาไว้สำหรับกลับไปดูที่ไหน แต่ที่กลับไปดู มันอยู่ค่าและส่วน

Exercise: Write sequence of stack allocation for calling add3()

```

int add2 (int a, int b) {
    return a + b;
}

int add3 (int a, int b, int c) {
    int res;
    res = add2(a, b);
    return res + c;
}

```

```

global variable ← int sum = 0;
int main() {
    sum += add3 (1, 2, 3);
    return 0;
}

```

| Caller | |
|------------|---|
| ... | |
| Pre-call | Push arguments onto the stack. |
| jsr callee | Call subroutine. This also pushes the return address onto the stack. |
| Post-call | Deallocate the stack space it allocated in the pre-call (adding positive offset to sp), and continue at the address of the instruction immediately after the jsr. |
| ... | |

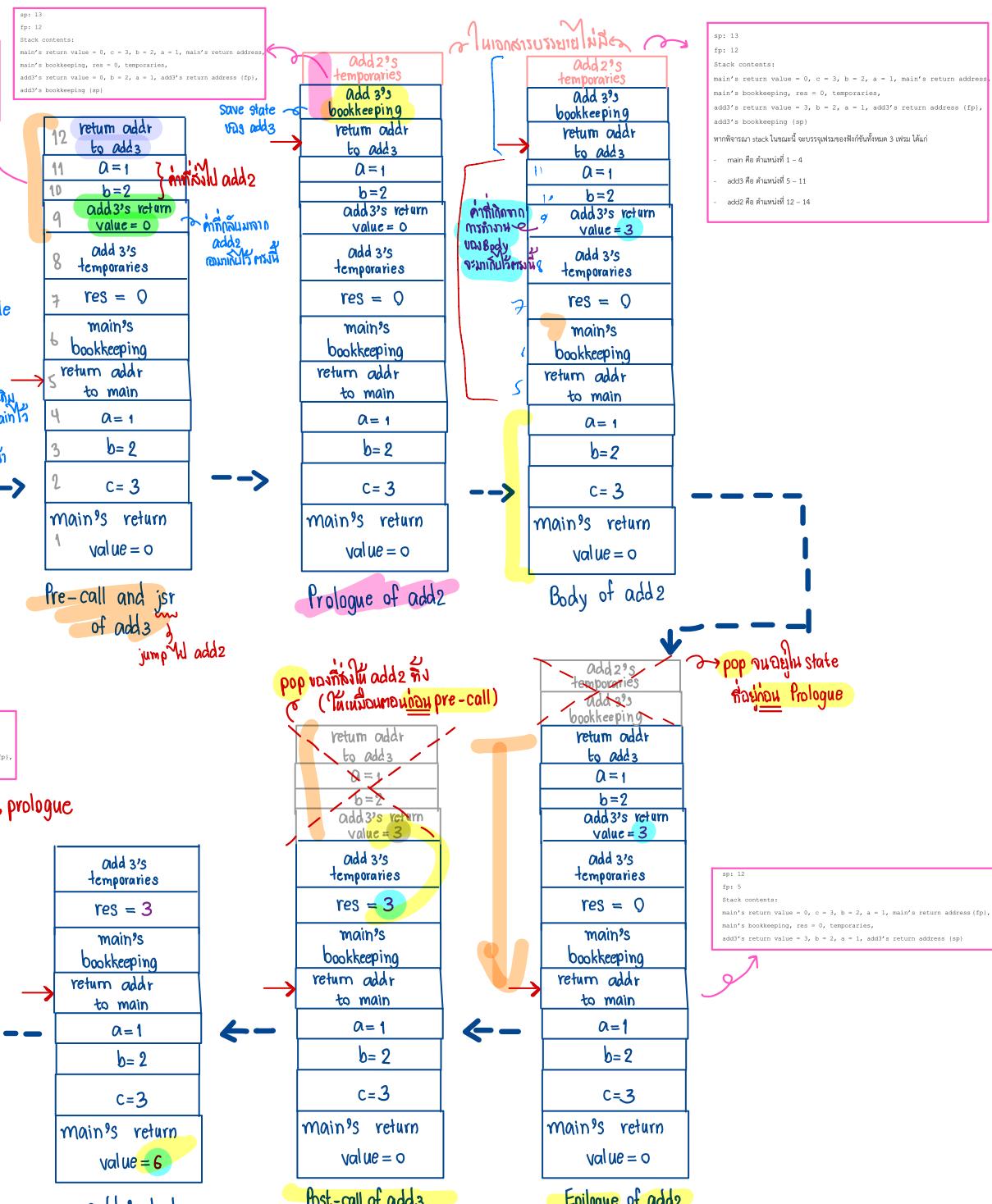
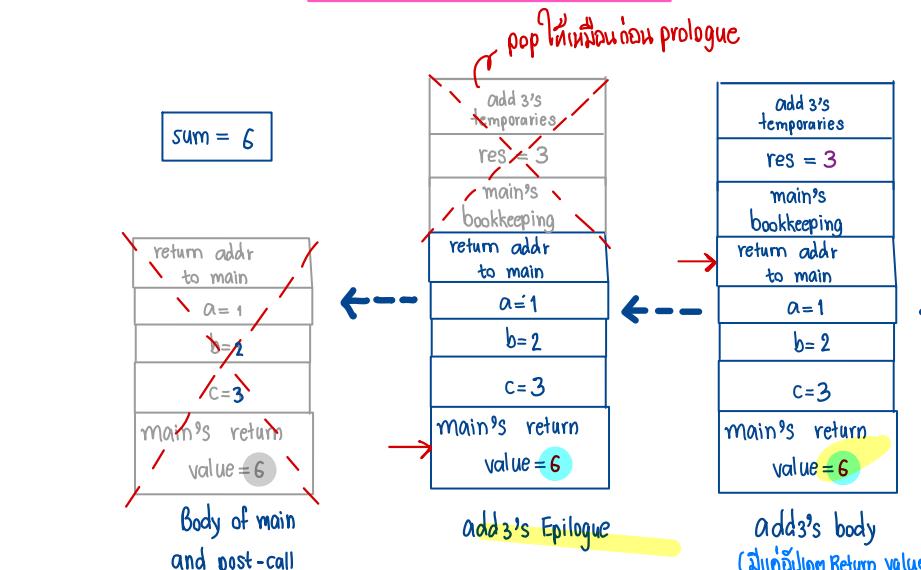
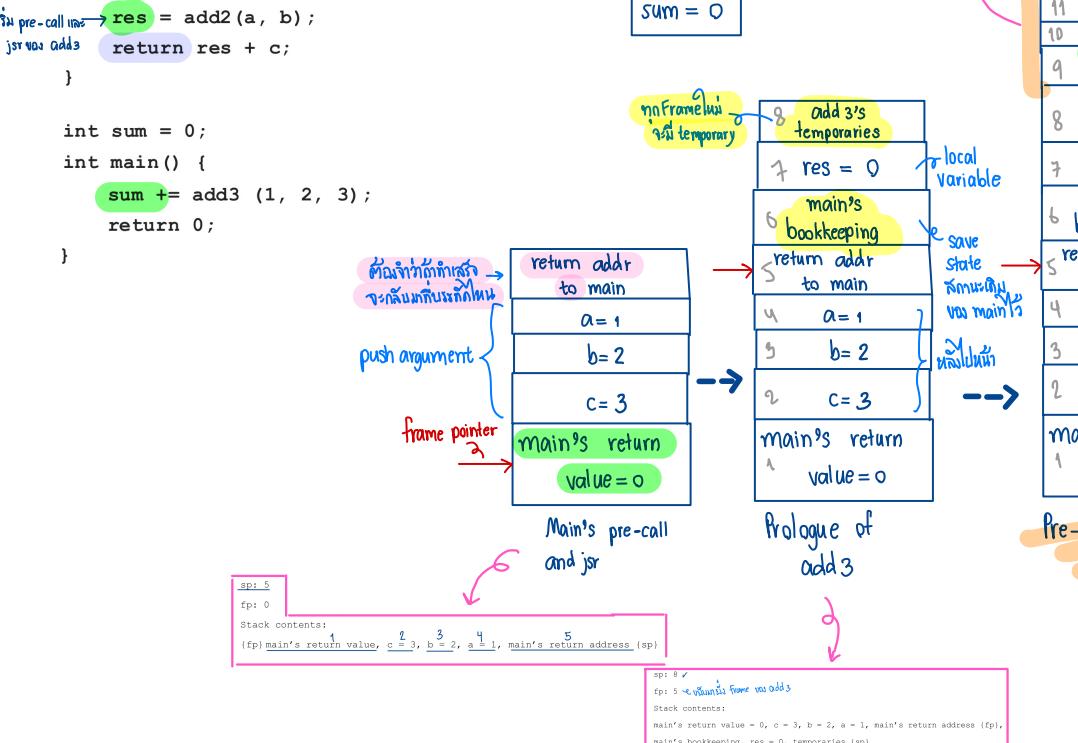
| Callee | |
|-----------|--|
| Prologue | Push old fp value and update it to a new frame, push other register values that should not be changed, and push local variables (if any) onto the stack. |
| Main Body | Execute and store return value in a register (or in a location in the stack, if reserved by caller). |
| Epilogue | Restore the saved register values, and deallocate the stack space it allocated in the prologue (adding positive offset to sp). |
| rts | Return to caller, jumping back to the return address. |

↑ ចូលលក់!
 * fp នឹងត្រួតពិនិត្យថា បានអ្វីណាដែលត្រូវរាយការណ៍នៅលើ method
 (មិនកំពង់ implement)

```
int add2 (int a, int b) {
    return a + b;
}
```

```
int add3 (int a, int b, int c) {
    int res;
    res = add2(a, b);
    return res + c;
}
```

```
int sum = 0;
int main() {
    sum += add3 (1, 2, 3);
    return 0;
}
```



Post call = pop จนหลัง pre call

Heap-Based Allocation

Stack → ក្នុងតាមពេលវេលាដែនឡាតាំង
Heap → បន្ទាន់ពេលវេលាដែន
(ដែល ទៅការីឡើលិន pointer point មកវា)

Heap is **storage** in which **subblocks** can be allocated and deallocated at arbitrary times.

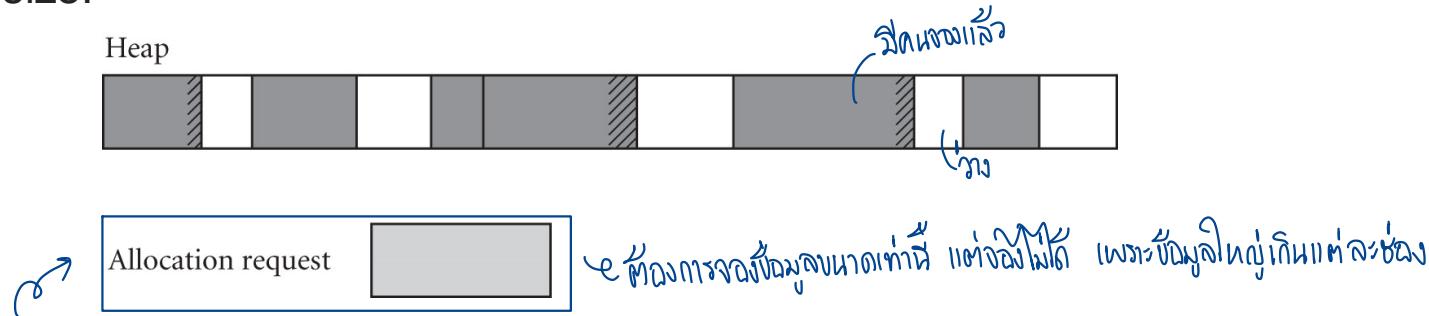
Heaps are required for dynamically allocated pieces of linked data structures or objects whose size may change (e.g. string, list, set).

Allocation is by some operation in a program, e.g. malloc() in C, new in Java, C++.

Deallocation is by garbage collection or some operation in a program, e.g. free() in C, delete in C++

Heap Storage Management

Storage management algorithms allocate blocks of the required size.



- **External fragmentation** happens when free spaces are scattered and not large enough to satisfy requests. → แก้ : Compact , จัดเรียงกลับ▪ *Compaction is needed to move already allocated blocks.*
- **Internal fragmentation** happens when unneeded space is left in the allocated block as it is smaller than some minimum threshold. ← เป็นต้องการขนาด 7 kB (เพื่อเข้ามาอยู่ใน Heap ที่ 8 kB)
จึงเป็น 8 kB (1 kB ถูกสูญเสียไปแล้ว) (เพื่อเข้ามาอยู่ใน Heap ที่ 1 kB จึงต้องใช้ก้อน)

Some storage management algorithms maintain pools of different standard block sizes.

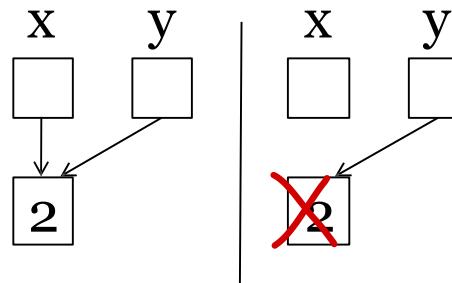
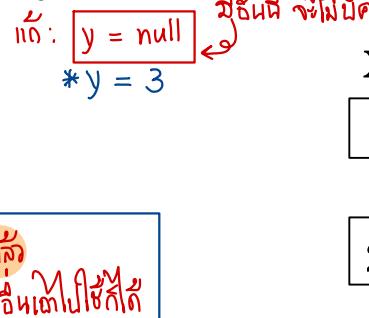
- Each request is rounded up to the next standard size, at the cost of internal fragmentation.

Problems with Manual Deallocation

รากำลังเสื่อม

Dangling reference is when the program accesses memory of the already released object (which may now used by another object).

```
int *x = new int;  
int *y;  
*x = 2;  
y = x;  
delete x;
```



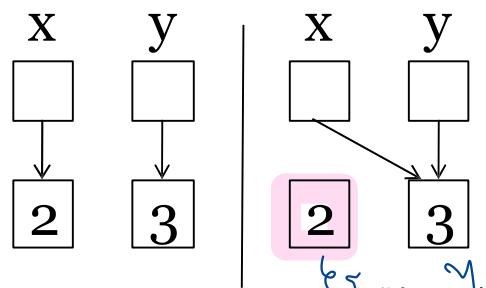
ทำงาน
ผ่านตัวแปรไปซึ่งแล้ว คาดว่า
ค่าเป็น null หมายความ

segmentation fault

Memory leak is when an object is not deallocated at the end of lifetime and heap space may run out.

```
int *x = new int;  
int *y = new int;  
*x = 2;  
*y = 3;  
x = y;
```

รากำลัง (ຈອງທກດົດໄຟ)



Exercise: Heap-Based Objects and Binding

Since **lifetime** means the **time between creation and destruction**,

Binding lifetime is **time between creation and destruction of name-to-object binding**.

Object lifetime is **time between creation and destruction of object**.

If **object lifetime** is **longer than binding lifetime**, we have **memory leak**...

If binding lifetime is longer than object lifetime, we have **dangling reference**

อธิบายเพิ่มเติม **object** คือ ส่วนความจำส่วน **Heap** ที่ถูกจดจำแล้ว **binding** คือ ส่วนของ **variable** ใน **Stack** หรือ **data segment**.

Garbage Collection

in background until memory leak

Garbage occurs when objects are no longer reachable from any program variables. Instead of explicit deallocation of heap-based objects, some languages provide run time library with garbage collection mechanism to implicitly identify and reclaim unreachable objects.

- More recent imperative languages, e.g. Java, C#
- Most functional and scripting languages

Benefit

- Reducing programming errors

Drawback

- Execution speed and complexity in language implementation

With language implementation becoming more complex, marginal complexity of garbage collection is reduced.

It is now an essential language feature with improved algorithms.

Scopes

Scope of binding is a textual region of the program in which a name-to-object binding is active.

ตัวแปรตัวประกอบที่มีผลเท่านั้น

Scope is a program region of maximal size in which no bindings change, e.g. the body of a class, subroutine, structured control-flow statement, or a block delimited with { }.

Referencing environment is a set of active bindings at any given point in a program's execution.

└ ลักษณะ method

- Referencing environment

เป็นเซตของ binding ที่สามารถเข้าถึงและใช้งานได้ ณ ตำแหน่งใดตำแหน่งหนึ่งของโปรแกรม

Exercise: Scope and Referencing Environment

```
Argument  
float op1(int x, float y) { int z; ... }  
// C  
float op2(int z) { ... }
```

op1 is considered one scope and op2 another scope.

Scope of x and y is op1

Scope of z is op1 / op2 (fl.2 binding)

Referencing environment of op1 consists of op1, x, y, z, op2

Referencing environment of op2 consists of op2, z, op1

Static Scoping (or Lexical Scoping)

ເກີນ ຂອງ ສຳເນົາ ຖໍ່ມີການ ອຸປະກອດ

Bindings between names and objects can be determined at compile time by examining the text of the program, without consideration of the flow of control at run time.

In most modern languages, scope of a binding is determined statically at compile time.

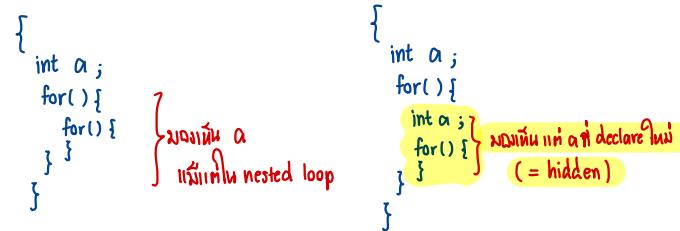
We can look at a C program, for example, and know which names refer to which objects at which points in the program based on purely textual rules. Then C is statically scoped.

Generally, scope of a local variable is limited to the subroutine in which it appears.

Classic Example: Nested Scope in Nested Subroutines

A feature in many languages (e.g. Pascal, Ada, Python) is nested subroutines. In nested subroutines, bindings of names to objects are resolved by closest nested scope rule.

- A name is known (or visible) in the scope in which it is declared, and in each internally nested scope.
- But a name can be hidden by another declaration of the same name in one or more nested scopes.
- To find the object that is bound to a given use of name
 - We look for its declaration in the current innermost scope.
 - If there is one, it defines the active binding for the name.
 - Otherwise we continue outward, examining successively surrounding scopes.



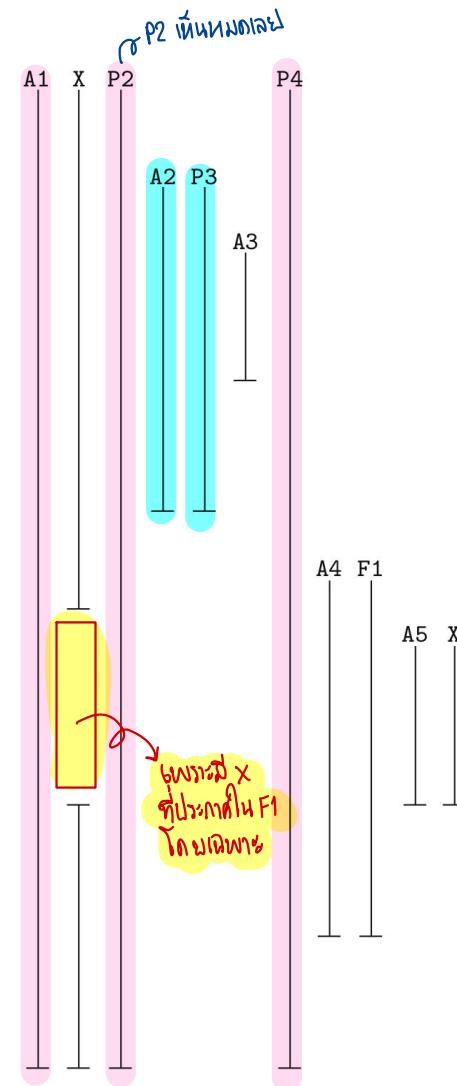
Nested Subroutines

Pascal

```

procedure P1(A1 : T1);
var X : real;
...
procedure P2(A2 : T2);
...
procedure P3(A3 : T3);
...
begin
  X=A3; (* body of P3 *)
end;
...
begin
  P3(3);      (* body of P2 *)
end;
...
procedure P4(A4 : T4);
...
function F1(A5 : T5) : T6;
var X : integer;
...
begin
  ...      (* body of F1 *)
end;
...
begin
  P2(2);      (* body of P4 *)
end;
...
begin
  P4(4);      (* body of P1 *)
end

```



Access to Non-local Objects in Nested Subroutines

To find objects in lexically surrounding scopes, **static link** is maintained in each frame when the frame is active at run time.

Static link points to parent frame, i.e. the frame of the most recent invocation of the lexically surrounding subroutines.

- Static link is the value of the fp of the parent frame, computed and passed in a register by caller, and stored as part of bookkeeping information.

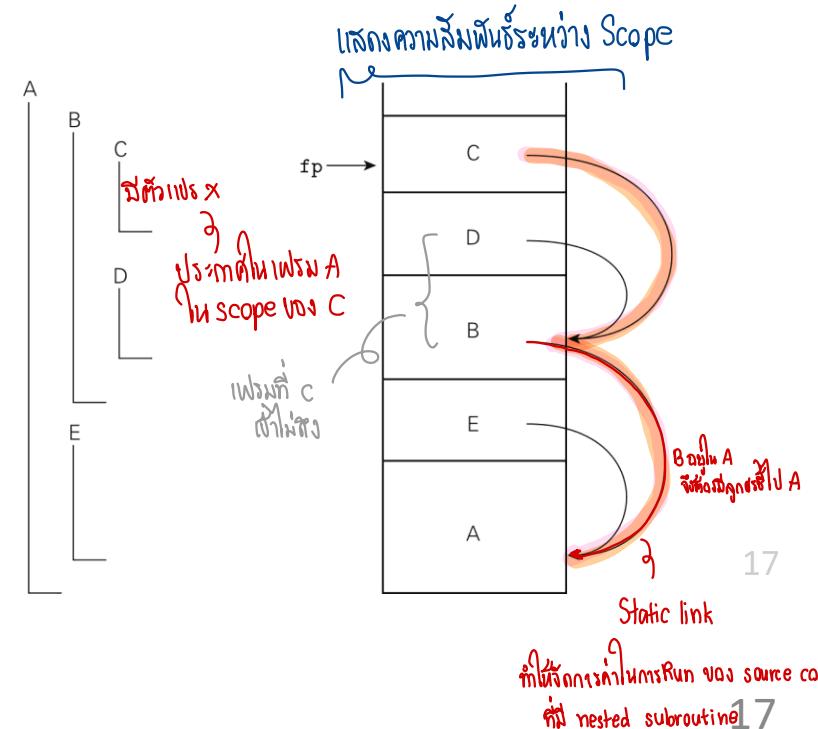
Sequence of nested calls at run time is A, E, B, D, C

C can find local objects in surrounding scope B by dereferencing **static chain** once and adding offset.

C can find local objects in B's surrounding scope, A, by dereferencing **static chain** twice and adding offset.

จากข้อมูลใน Stack เฟรมจะสามารถถดถอยความสัมพันธ์ระหว่าง Scope เอ้าได้ และเมื่อต้องค้นหา binding การท่องตาม scope จะสามารถทำได้ถูกต้องผ่าน static link เช่น เมื่อต้องการหาตัวแปรชื่อ X ที่ ประกาศไว้อยู่ในเฟรม A จาก scope ของ C

การค้นหาก็จะเริ่มจากเฟรม C เมื่อไม่พบก็จะย้ายตาม Static link นั้นคือ ย้ายไปยังเฟรม B จนกระทั่งไปพบในเฟรม A ซึ่งจะข้ามเฟรม D และ E ไปถึงเป็น scope ที่เฟรม C ไม่สามารถจะเข้าถึงได้ โดยจะเป็นไปตามโครงสร้างที่เขียนไว้ในโปรแกรม



Exercise: Static Link in Nested Subroutines

Exercise: Static Link in Nested Subroutines

bookkeeping = នូវរាយនៃក្នុង Frame ណាមួយ

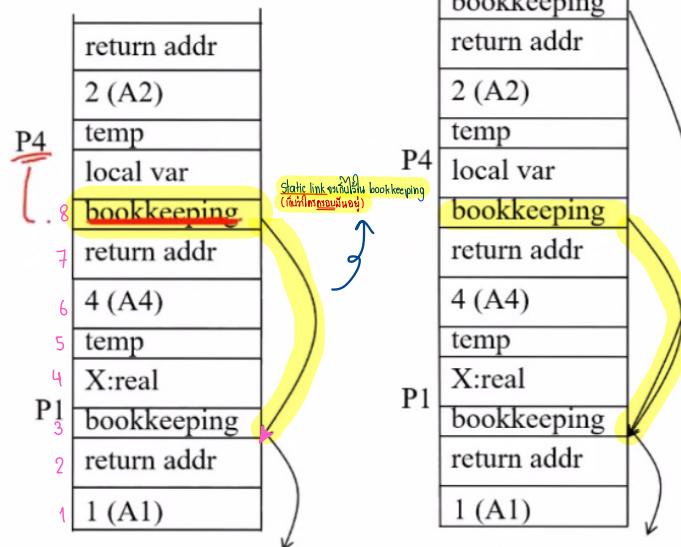
លើកដំឡើន P1(1)

```

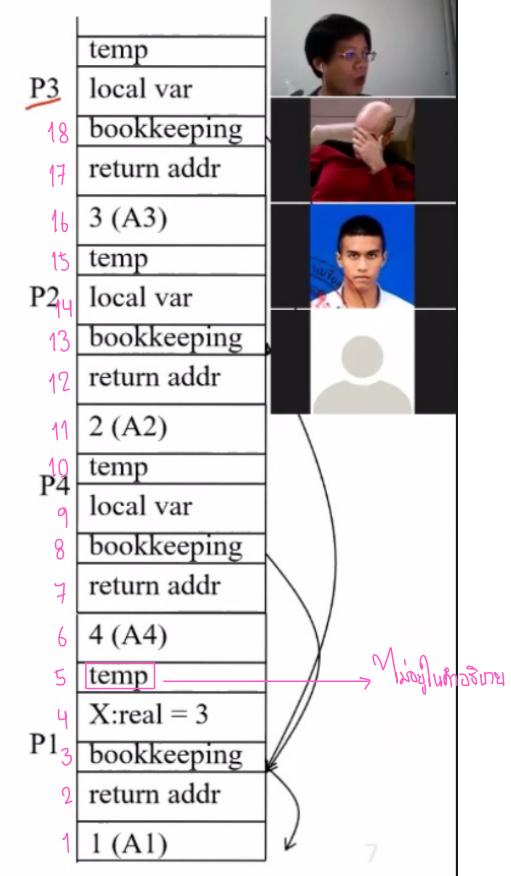
procedure P1(A1 : T1);
var X : real;
procedure P2(A2 : T2);
begin
    X := A3;
end;
begin
    P3(3);
end;
procedure P4(A4 : T4);
begin
    P2(2);
end;
begin
    P4(4);
end;

```

A1 = 1, Return Address, Bookkeeping {parent's fp, caller's fp}, X = 0
A4 = 4, P1's Return Address, Bookkeeping {parent's fp = 2, caller's fp = 2}



Parent frame = ពីរដែលមែនក្រោមគ្នា
 Caller frame = រាយការណ៍នៃខ្លួន



1 1 (A1), Return Address, Bookkeeping {parent's fp, caller's fp}, X = 0,
2 2 (A2), Return Address, Bookkeeping {parent's fp = 2, caller's fp = 2},
3 3 (A3), Return Address, Bookkeeping {parent's fp = 3, caller's fp = 3},
4 4 (A4), Return Address, Bookkeeping {parent's fp = 4, caller's fp = 4},
5 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 409, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 443, 444, 445, 446, 447, 448, 449, 450, 451, 452, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536, 537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 5610, 5611, 5612, 5613, 5614, 5615, 5616, 5617, 5618, 5619, 5620, 5621, 5622, 5623, 5624, 5625, 5626, 5627, 5628, 5629, 5630, 5631, 5632, 5633, 5634, 5635, 5636, 5637, 5638, 5639, 56310, 56311, 56312, 56313, 56314, 56315, 56316, 56317, 56318, 56319, 56320, 56321, 56322, 56323, 56324, 56325, 56326, 56327, 56328, 56329, 56330, 56331, 56332, 56333, 56334, 56335, 56336, 56337, 56338, 56339, 563310, 563311, 563312, 563313, 563314, 563315, 563316, 563317, 563318, 563319, 563320, 563321, 563322, 563323, 563324, 563325, 563326, 563327, 563328, 563329, 563330, 563331, 563332, 563333, 563334, 563335, 563336, 563337, 563338, 563339, 5633310, 5633311, 5633312, 5633313, 5633314, 5633315, 5633316, 5633317, 5633318, 5633319, 5633320, 5633321, 5633322, 5633323, 5633324, 5633325, 5633326, 5633327, 5633328, 5633329, 5633330, 5633331, 5633332, 5633333, 5633334, 5633335, 5633336, 5633337, 5633338, 5633339, 56333310, 56333311, 56333312, 56333313, 56333314, 56333315, 56333316, 56333317, 56333318, 56333319, 56333320, 56333321, 56333322, 56333323, 56333324, 56333325, 56333326, 56333327, 56333328, 56333329, 56333330, 56333331, 56333332, 56333333, 56333334, 56333335, 56333336, 56333337, 56333338, 56333339, 563333310, 563333311, 563333312, 563333313, 563333314, 563333315, 563333316, 563333317, 563333318, 563333319, 563333320, 563333321, 563333322, 563333323, 563333324, 563333325, 563333326, 563333327, 563333328, 563333329, 563333330, 563333331, 563333332, 563333333, 563333334, 563333335, 563333336, 563333337, 563333338, 563333339, 5633333310, 5633333311, 5633333312, 5633333313, 5633333314, 5633333315, 5633333316, 5633333317, 5633333318, 5633333319, 5633333320, 5633333321, 5633333322, 5633333323, 5633333324, 5633333325, 5633333326, 5633333327, 5633333328, 5633333329, 5633333330, 5633333331, 5633333332, 5633333333, 5633333334, 5633333335, 5633333336, 5633333337, 5633333338, 5633333339, 56333333310, 56333333311, 56333333312, 56333333313, 56333333314, 56333333315, 56333333316, 56333333317, 56333333318, 56333333319, 56333333320, 56333333321, 56333333322, 56333333323, 56333333324, 56333333325, 56333333326, 56333333327, 56333333328, 56333333329, 56333333330, 56333333331, 56333333332, 56333333333, 56333333334, 56333333335, 56333333336, 56333333337, 56333333338, 56333333339, 563333333310, 563333333311, 563333333312, 563333333313, 563333333314, 563333333315, 563333333316, 563333333317, 563333333318, 563333333319, 563333333320, 563333333321, 563333333322, 563333333323, 563333333324, 563333333325, 563333333326, 563333333327, 563333333328, 563333333329, 563333333330, 563333333331, 563333333332, 563333333333, 563333333334, 563333333335, 563333333336, 563333333337, 563333333338, 563333333339, 5633333333310, 5633333333311, 5633333333312, 5633333333313, 5633333333314, 5633333333315, 5633333333316, 5633333333317, 5633333333318, 5633333333319, 5633333333320, 5633333333321, 5633333333322, 5633333333323, 5633333333324, 5633333333325, 5633333333326, 5633333333327, 5633333333328, 5633333333329, 5633333333330, 5633333333331, 5633333333332, 5633333333333, 5633333333334, 5633333333335, 5633333333336, 5633333333337, 5633333333338, 5633333333339, 56333333333310, 56333333333311, 56333333333312, 56333333333313, 56333333333314, 56333333333315, 56333333333316, 56333333333317, 56333333333318, 56333333333319, 56333333333320, 56333333333321, 56333333333322, 56333333333323, 56333333333324, 56333333333325, 56333333333326, 56333333333327, 56333333333328, 56333333333329, 56333333333330, 56333333333331, 56333333333332, 56333333333333, 56333333333334, 56333333333335, 56333333333336, 56333333333337, 56333333333338, 56333333333339, 563333333333310, 563333333333311, 563333333333312, 563333333333313, 563333333333314, 563333333333315, 563333333333316, 563333333333317, 563333333333318, 563333333333319, 563333333333320, 563333333333321, 563333333333322, 563333333333323, 563333333333324, 563333333333325, 563333333333326, 563333333333327, 563333333333328, 563333333333329, 563333333333330, 563333333333331, 563333333333332, 563333333333333, 563333333333334, 563333333333335, 563333333333336, 563333333333337, 563333333333338, 563333333333339, 5633333333333310, 5633333333333311, 5633333333333312, 5633333333333313, 5633333333333314, 5633333333333315, 5633333333333316, 5633333333333317, 5633333333333318, 5633333333333319, 5633333333333320, 5633333333333321, 5633333333333322, 5633333333333323, 5633333333333324, 5633333333333325, 5633333333333326, 5633333333333327, 5633333333333328, 5633333333333329, 5633333333333330, 5633333333333331, 5633333333333332, 5633333333333333, 5633333333333334, 5633333333333335, 5633333333333336, 5633333333333337, 5633333333333338, 5633333333333339, 56333333333333310, 56333333333333311, 56333333333333312, 56333333333333313, 56333333333333314, 56333333333333315, 56333333333333316, 56333333333333317, 56333333333333318, 56333333333333319, 56333333333333320, 56333333333333321, 56333333333333322, 56333333333333323, 56333333333333324, 56333333333333325, 56333333333333326, 56333333333333327, 56333333333333328, 56333333333333329, 56333333333333330, 56333333333333331, 56333333333333332, 56333333333333333, 56333333333333334, 56333333333333335, 56333333333333336, 56333333333333337, 56333333333333338, 56333333333333339, 563333333333333310, 563333333333333311, 563333333333333312, 563333333333333313, 563333333333333314, 563333333333333315, 563333333333333316, 563333333333333317, 563333333333333318, 563333333333333319, 563333333333333320, 563333333333333321, 563333333333333322, 563333333333333323, 563333333333333324, 563333333333333325, 563333333333333326, 563333333333333327, 563333333333333328, 563333333333333329, 563333333333333330, 563333333333333331, 563333333333333332, 563333333333333333, 563333333333333334, 563333333333333335, 563333333333333336, 563333333333333337, 563333333333333338, 563333333333333339, 5633333333333333310, 5633333333333333311, 5633333333333333312, 5633333333333333313, 5633333333333333314, 5633333333333333315, 5633333333333333316, 5633333333333333317, 5633333333333333318, 5633333333333333319, 5633333333333333320, 5633333333333333321, 5633333333333333322, 5633333333333333323, 5633333333333333324, 5633333333333333325, 5633333333333333326, 5633333333333333327, 5633333333333333328, 5633333333333333329, 5633333333333333330, 5633333333333333331, 5633333333333333332, 5633333333333333333, 5633333333333333334, 5633333333333333335, 5633333333333333336, 5633333333333333337, 5633333333333333338, 5633333333333333339, 56333333333333333310, 56333333333333333311, 56333333333333333312, 56333333333333333313, 56333333333333333314, 56333333333333333315, 56333333333333333316, 56333333333333333317, 56333333333333333318, 56333333333333333319, 56333333333333333320, 56333333333333333321, 56333333333333333322, 56333333333333333323, 56333333333333333324, 56333333333333333325, 56333333333333333326, 56333333333333333327, 56333333333333333328, 56333333333333333329, 56333333333333333330, 56333333333333333331, 56333333333333333332, 56333333333333333333, 56333333333333333334, 56333333333333333335, 56333333333333333336, 56333333333333333337, 56333333333333333338, 56333333333333333339, 563333333333333333310, 563333333333333333311, 563333333333333333312, 563333333333333333313, 563333333333333333314, 563333333333333333315, 563333333333333333316, 563333333333333333317, 563333333333333333318, 563333333333333333319, 563333333333333333320, 563333333333333333321, 563333333333333333322, 563333333333333333323, 563333333333333333324, 563333333333333333325, 563333333333333333326, 563333333333333333327, 563333333333333333328, 563333333333333333329, 563333333333333333330, 563333333333333333331, 563333333333333333332, 563333333333333333333, 563333333333333333334, 563333333333333333335, 563333333333333333336, 563333333333333333337, 563333333333333333338, 563333333333333333339, 5633333333333333333310, 5633333333333333333311, 5633333333333333333312, 5633333333333333333313, 5633333333333333333314, 5633333333333333333315, 5633333333333333333316, 5633333333333333333317, 5633333333333

nested subroutine = និង method នៃ គ្រប់គ្រាន់ method នេះ

nested block = block គ្រប់គ្រាន់ នៃ for loop គ្រប់គ្រាន់ , និងក្នុង method
(អាមេរិកសាស្ត្រការណើង)

Nested Blocks

In some languages (e.g. C, C++, Java), declarations in nested {...} blocks hide outer declarations with the same name.

The space is allocated with local variables in subroutine prologue and deallocated in the epilogue.

```
// Java
class a {
    void b(...){
        int c; ...
        while(...){
            int d; ...
            {
                int e;
                c = d + e; ✓
            }
        }
    }
}
```

```
// Java
class a {
    int c;
    void b(...){
        int c; // តិច c ពេល
        c = 1; // តិច c ពេល
        this.c = 2;
        ...
    }
}
```

ក្នុង instance variable

Dynamic Scoping

Scope តិចនីងមាន run
→ ក្រឡេ source code នៅឡើ គណនោ scope ដូច

Bindings between names and objects depend on the flow of control, and on the order in which subroutines are called.

The current binding for a given name is the one encountered most recently during execution, and not yet destroyed by returning from its scope.

Type checking in expressions and argument checking in subroutine calls must be deferred until runtime.

Languages with dynamic scoping tend to be interpreted, rather than compiled, e.g. Lisp, Perl.

Bindings in Dynamic Scoping

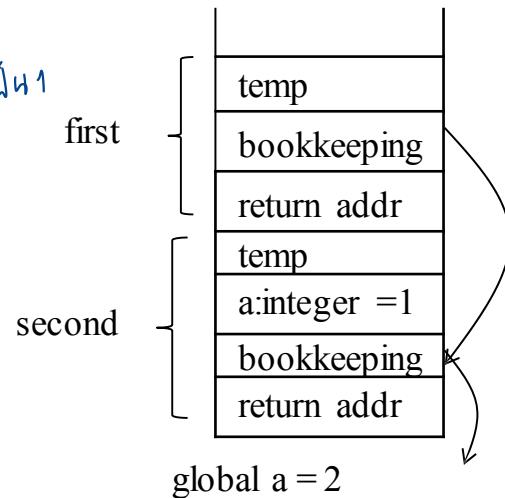
With static scope rule, global a is reassigned to 1.

With dynamic scope rule, and if first is entered from second, local a is assigned to 1 .
The write refers to global a = 2.

```
1: a : integer      -- global declaration
2: procedure first
3:   a := 1
4: procedure second
5:   a : integer      -- local declaration
6:   first ()
7:   a := 2
8:   if read_integer () > 0
9:     second ()
10: else
11:   first ()
12: write_integer (a)
```

Annotations:

- Line 1: "a is global" (yellow oval)
- Line 3: "a := 1" (yellow oval)
- Line 5: "local declaration" (pink oval)
- Line 7: "a := 2" (blue oval)
- Line 12: "global a = 2" (pink oval)
- Line 4: "In Static Scope ดีกว่าใน global variable จะ H 1" (blue text)
- Line 5: "ไม่ใช่ local" (pink text)
- Line 5: "(จะ H 1)" (red text)



Bindings in Dynamic Scoping

With static scope rule, global a is reassigned to 1.

With dynamic scope rule, and if first is entered from second, local a is assigned to 1 . The write refers to global a = 2.

```

1: a : integer      -- global declaration

2: procedure first
3:   a := 1          ก่อน declare (nm nms Run)
4:   procedure second
5:     a : integer    ใน method second() ก่อน run method เช่น
6:     first ()       ใน method main (โปรแกรมที่ 1) ที่ a เป็น a global
7:   a := 2           second

8: if read_integer () > 0
9:   second ()
10: else
11:   first ()
12: write_integer (a)

```

ก่อน declare (nm nms Run)

ก่อน run method เช่น second()

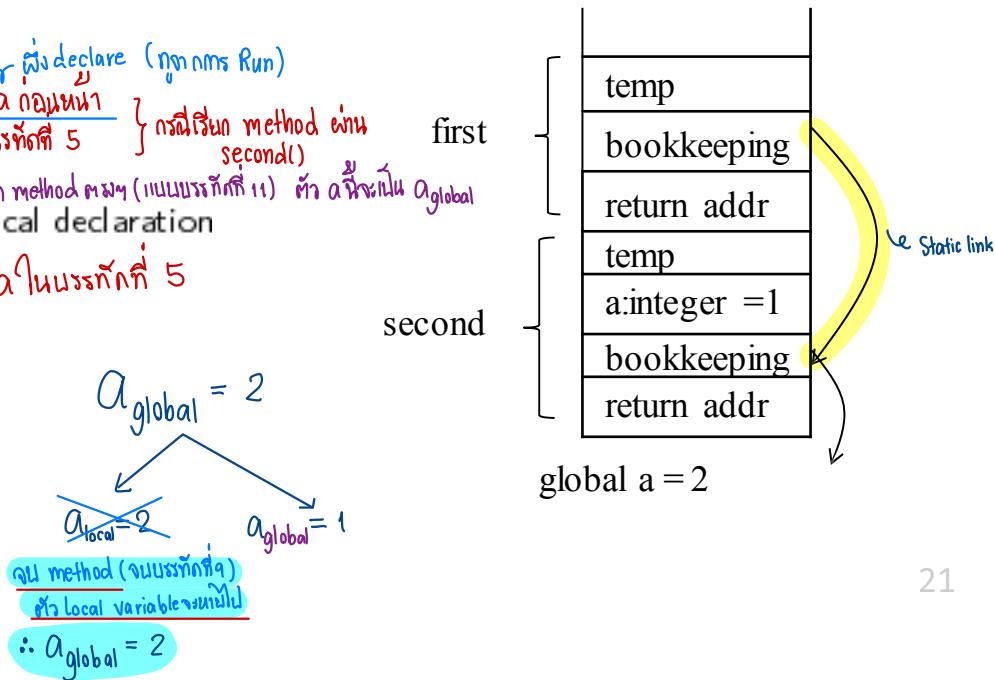
ใน method second() ก่อน run method เช่น main (โปรแกรมที่ 1) ที่ a เป็น a global

second

$a_{\text{global}} = 2$

$a_{\text{local}} = 2$

$a_{\text{global}} = 1$

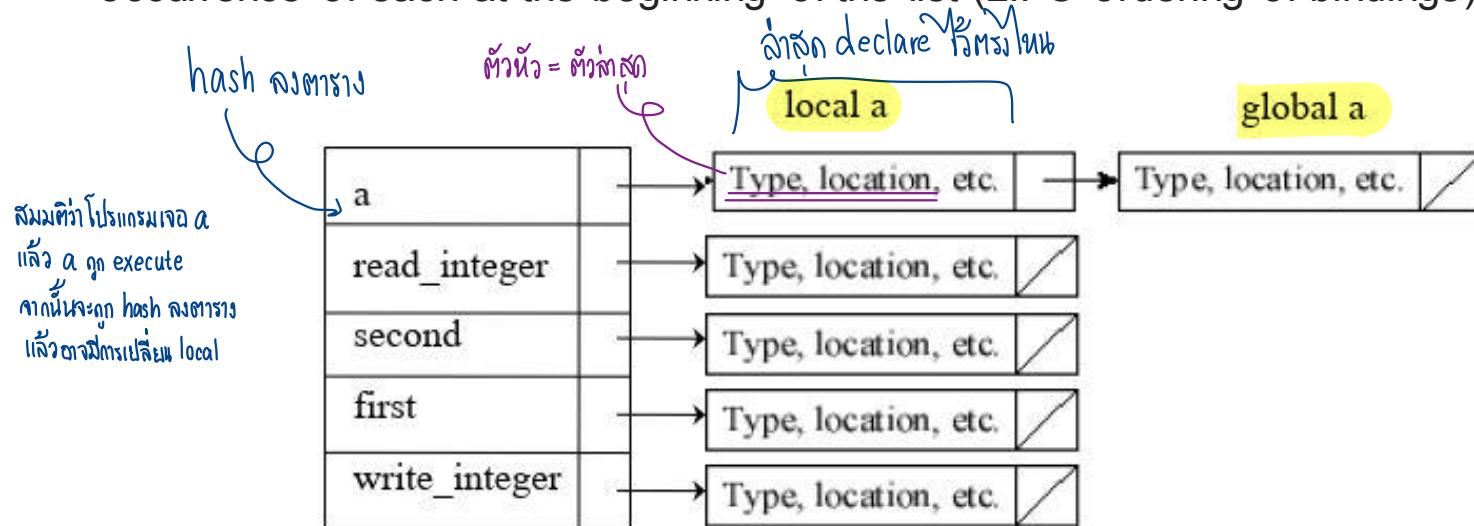


(Hash table)

Looking up for Bindings in Dynamic Scoping

Instead of traversing run time stack, a central reference table can be additionally implemented.

It contains a list of entries for each distinct name in the program, with the most recent occurrence of each at the beginning of the list (LIFO ordering of bindings).



Exercise: Bindings in Dynamic Scoping

With dynamic scope rule, if first is entered from main

- What does write_integer refer to, global a or local a? global
- What does write_integer write? 1

```
1:  a : integer      -- global declaration
2:  procedure first
3:    a := 1
4:  procedure second
5:    a : integer      -- local declaration
6:    first ()
7:    a := 2
8:    if read_integer () > 0
9:      second ()
10:   else
11:     first () → အေဒီဂါန် dynamic
12:     write_integer (a) အေဒီဂါန် global အေး
```

