

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.* *ດີນອານຸ Run ໃນ
ການຈະສ້າງອີງຕະຫຼາດ*
- **Dynamic binding** = Things are bound at run time (late binding)
ເຊື້ອ! ເວລາປໍ່ໄດ້ໂປຣແກຣມ 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!*
ເກມກົດໃນໜີ Compiler

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

2110216 DDOS 1

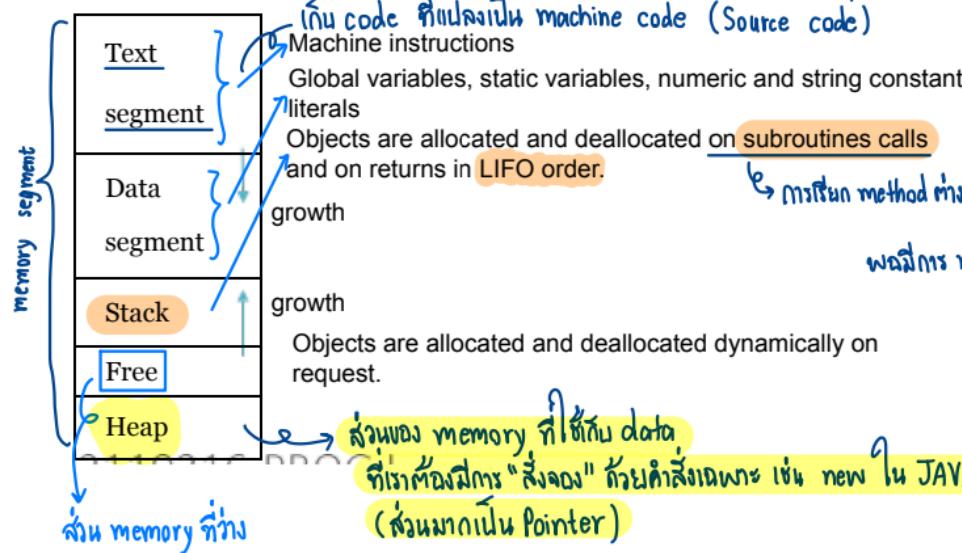
Object Lifetime and Storage Management

Lifetime is time between creation and destruction.

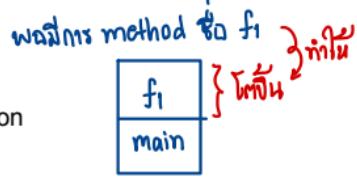
An object to which a name is bound has its object lifetime.

Object lifetime corresponds to its storage allocation

พื้นที่ที่ต้องการจะเป็น ชื่อที่ เครื่องพิมพ์



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



Register ต้องบันทึกค่า

ก่อนหน้า CPU จึงจะสามารถเรียกใช้ได้

Stack-Based Allocation

* ชุดข้อมูลของแต่ละโปรแกรมนั้นๆ push ลงใน stack

Save state ของแต่ละ Frame



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

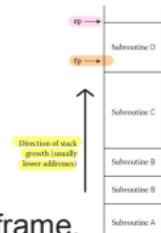
- Arguments (if any)
- Return address
- Bookkeeping : Saved values of registers, reference to other frame (see later) etc.
- Local variables (if any) → ต้องมีการกำหนดให้ตัวแปรภายใน method นั้นๆ
- Temporaries : Intermediate values produced in complex calculations

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

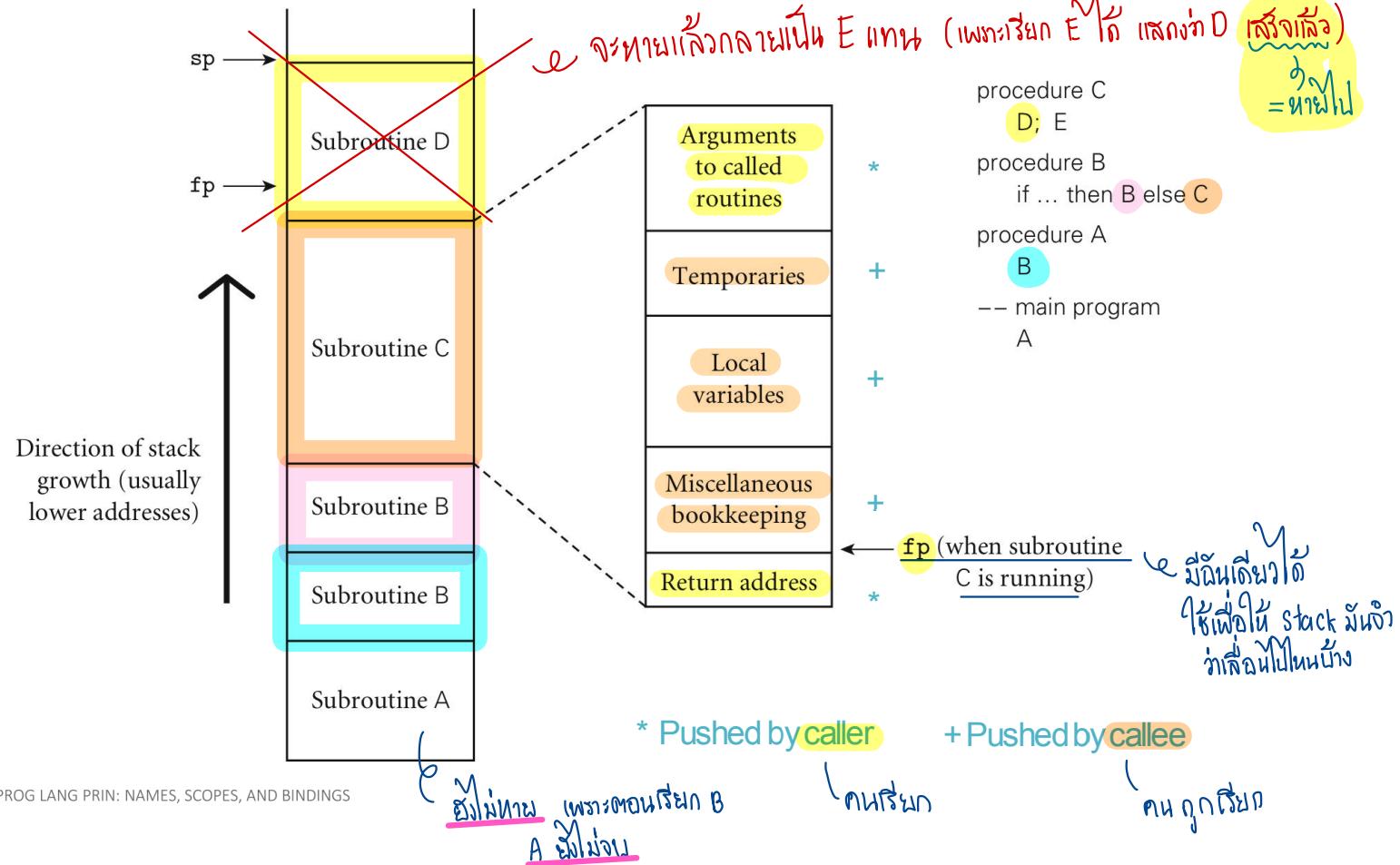
register ที่เก็บ frame ปัจจุบันนี้เป็น frame

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.
- Code to access data within a frame adds a predetermined negative or positive offset to the value in fp.



Calling Chain and Stack Frames

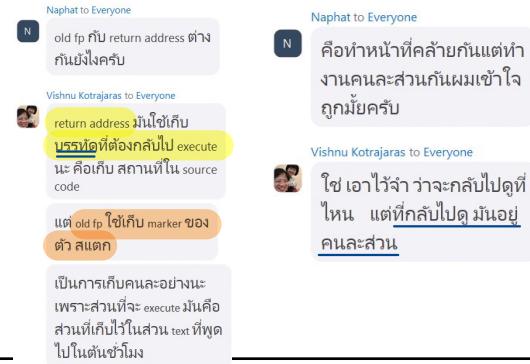


Maintenance of Stack

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



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.
...	

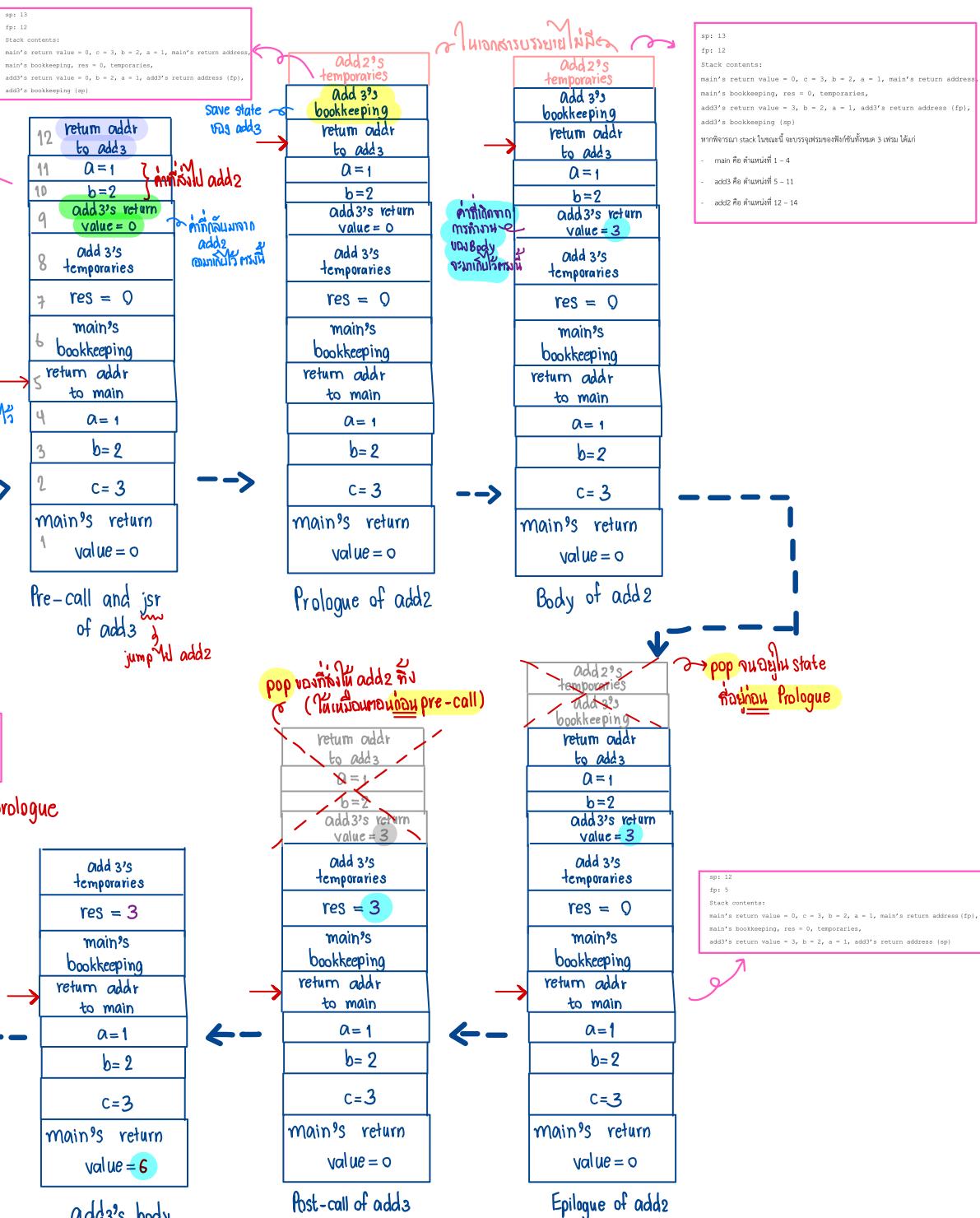
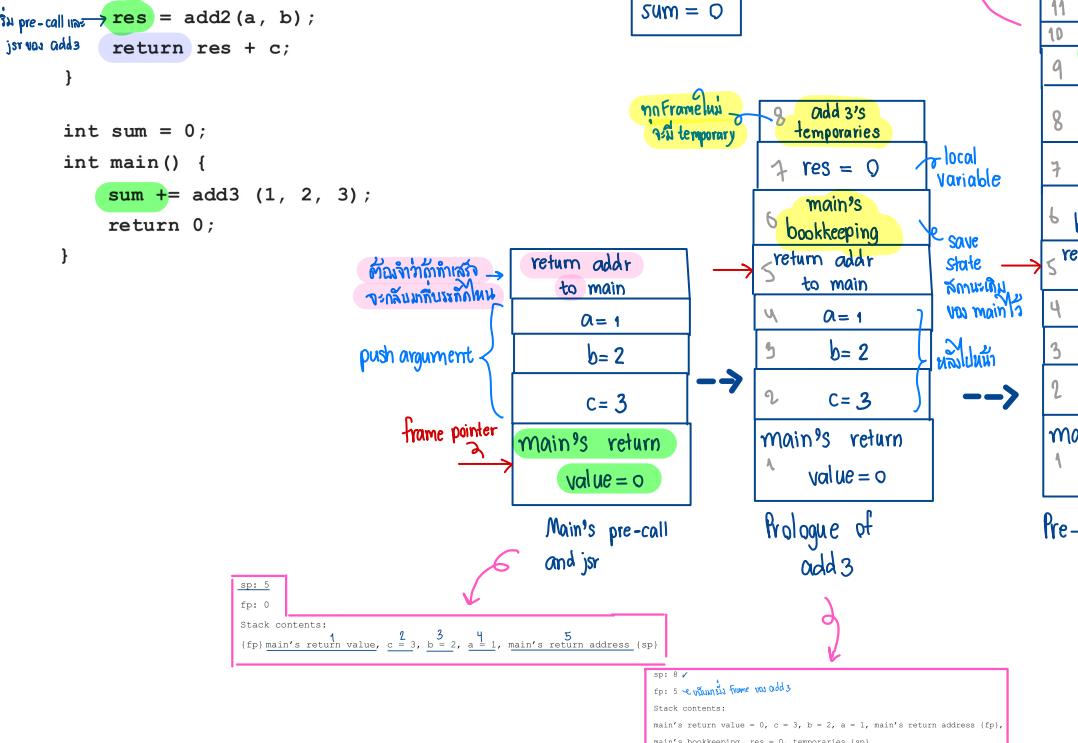
Callee	
Prologue	Push old fp value and update it to a new frame, push other register values that should not be changed, and push local variables (if any) onto the stack.
Main Body	Execute and store return value in a register (or in a location in the stack, if reserved by caller).
Epilogue	Restore the 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 → ក្នុងតាមពេលវេលាដែនឡើង
Heap → បន្ទាន់ពេលវេលាដែន
(ដែល ទៅការថាអ្នកត្រូវផ្តល់ pointer ចូលរួម)

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

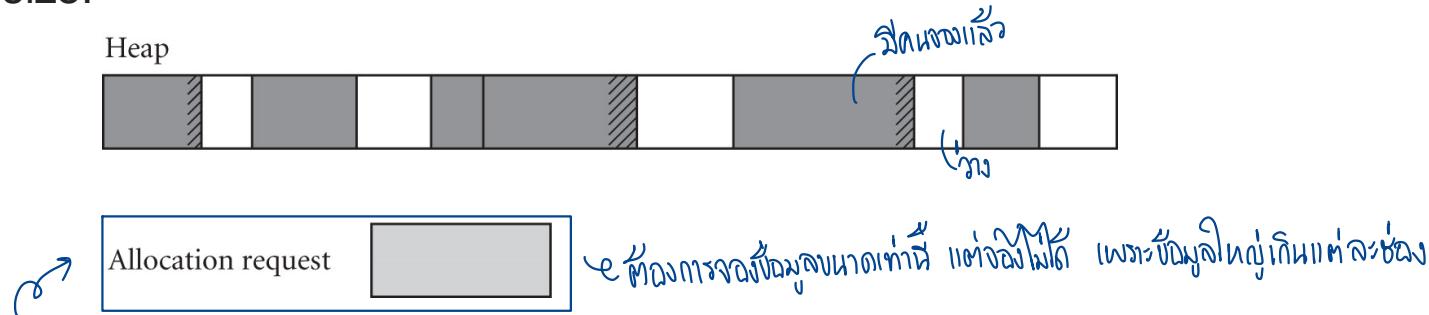
Heaps are required for dynamically allocated pieces of linked data structures or objects whose size may change (e.g. string, list, set).

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.
 - ← เป็นต้องการจะบันทึก 7 kB (เพื่อเข้ามาคีย์ไปใน Heap ที่มีขนาด 8 kB (1 kB ดูแล้วไม่เป็นปัญหา) เพราะต้องหักไปแล้ว)

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

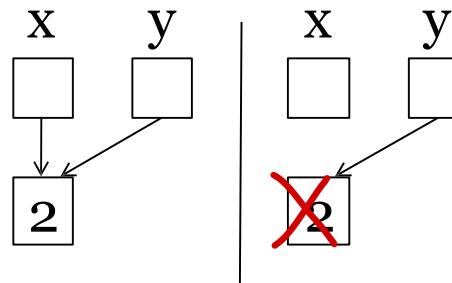
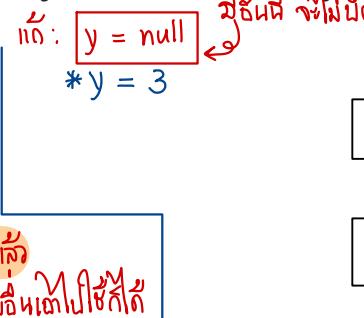
- Each request is rounded up to the next standard size, at the cost of internal fragmentation.

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;
```



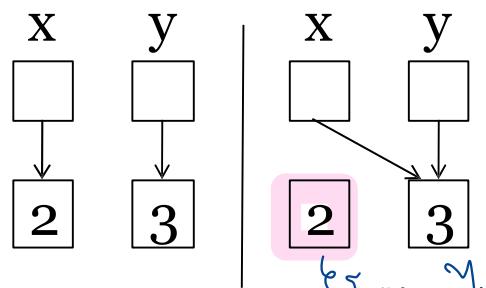
กาวแก้
หันตัวไปใช้ชื่อเลื่อน คาดว่า
ก็เป็น null มาก่อน

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

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.

ตัวแปรตัวแปรงที่มีผลเท่านั้น

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

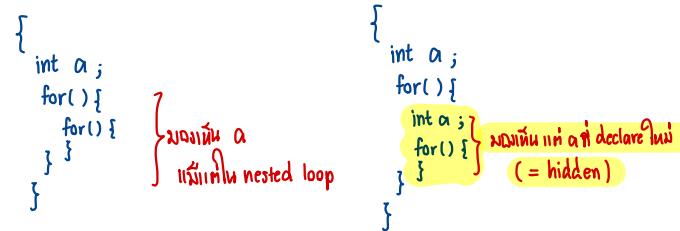
We can look at a C program, for example, and know which names refer to which objects at which points in the program based on purely textual rules. Then C is statically scoped.

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.



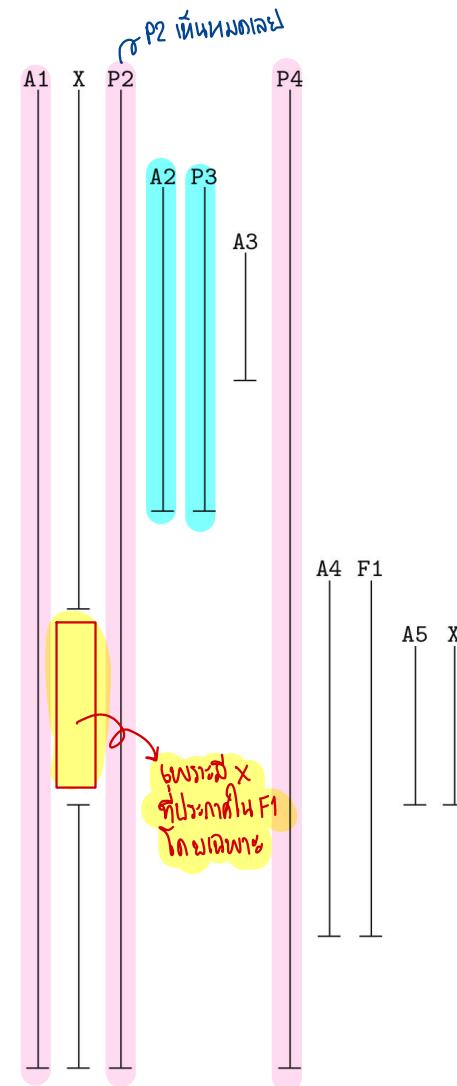
Nested Subroutines

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.

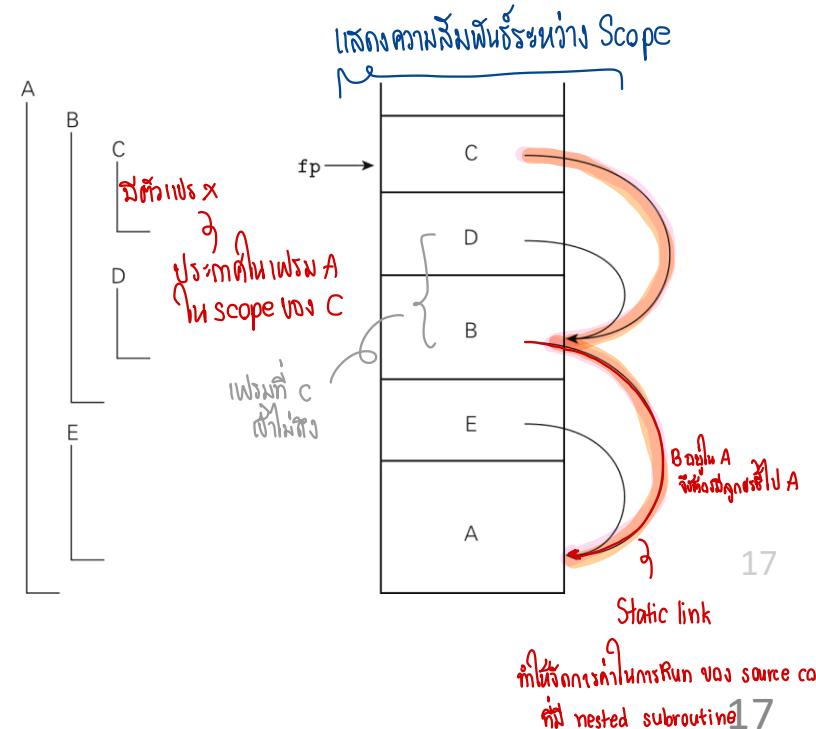
Sequence of nested calls at run time is A, E, B, D, C

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 = ទីតាំងនៃអាជីវកម្ម Frame នៃខ្លួន

លើកដំឡើន P1(1)

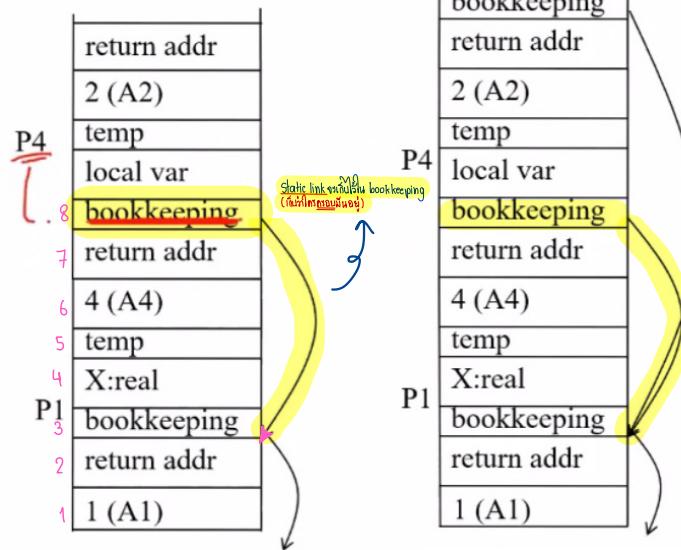
```

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;

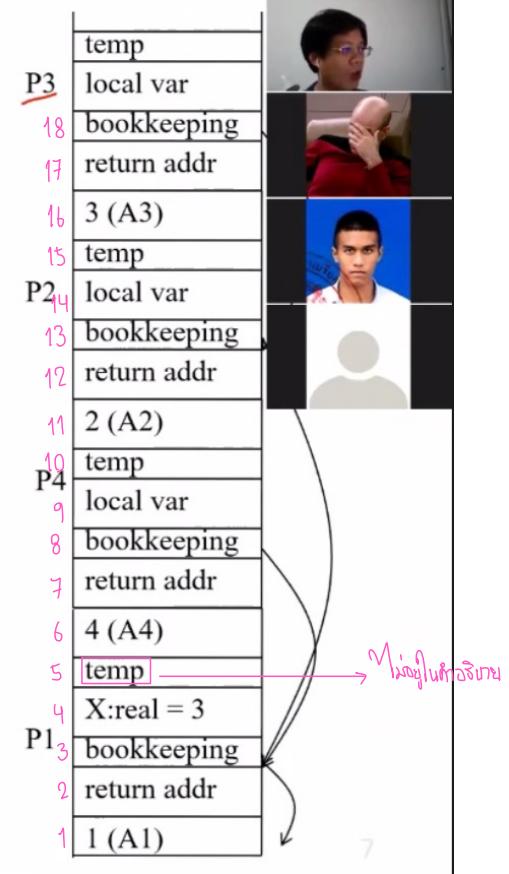
```

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 = ផ្ទាល់ (រៀបចំការងារសង្គម)
Caller frame = រៀបចំ Frame នៃខ្លួន



18

18

nested subroutine = นิยาม method ที่อยู่ใน function method นั้น

nested block = block ซึ่งมีลักษณะ เช่น for loop หรือ , รีบก็คือ method
(ไม่ใช่การเรียก)

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; // ตัวนี้
        c = 1; // ตัวนี้
        this.c = 2;
        ...
    }
}
```

↑ กล่าวถึง instance variable

Dynamic Scoping

Scope តិចនីងមាន run
→ ក្រឡុយ 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

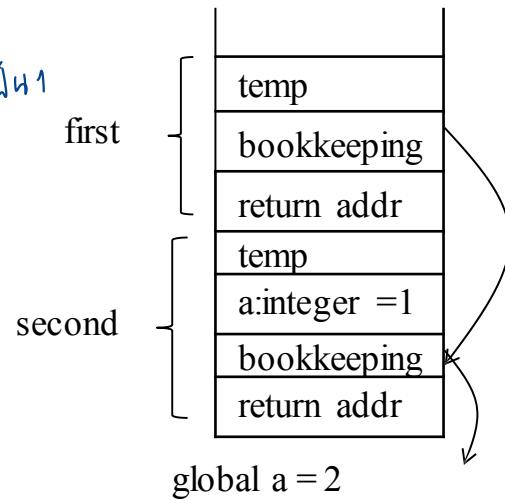
With static scope rule, global a is reassigned to 1.

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 circle)
- Line 3: "a := 1" (yellow oval)
- Line 5: "local declaration" (pink oval)
- Line 7: "a := 2" (blue oval)
- Line 8: "if read_integer () > 0" (pink oval)
- Line 12: "write_integer (a)" (blue oval)
- Handwritten note: "In Static Scope នៅលើង global variable នេះ 1" (in Static Scope, this global variable is 1)
- Handwritten note: "រាយការណ៍ (មិនអាច)" (not possible)



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          จัด declare (ก่อน Run)
4:   procedure second
5:     a : integer    ใน class บรรทัดที่ 5 } ก็จะ return method คือ second()
6:     first ()       และ return method ของ (แบบบรรทัดที่ 1) ที่ a คือเป็น a global
7:   a := 2
8:   if read_integer () > 0
9:     second ()      จัด return a ที่บรรทัดที่ 5
10:  else
11:    first ()
12:  write_integer (a)

```

จัด declare (ก่อน Run)

ใน class บรรทัดที่ 5 } ก็จะ return method คือ second()

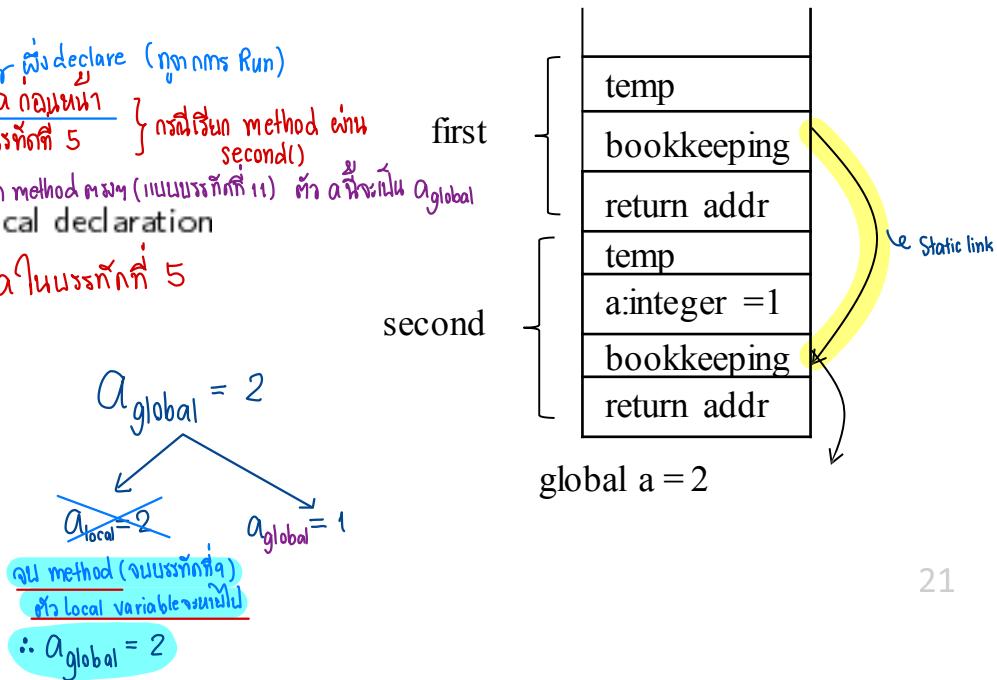
และ return method ของ (แบบบรรทัดที่ 1) ที่ a คือเป็น a global

จัด return a ที่บรรทัดที่ 5

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

$\begin{cases} a_{\text{local}} = 2 \\ a_{\text{global}} = 1 \end{cases}$

จัด return a ที่บรรทัดที่ 5

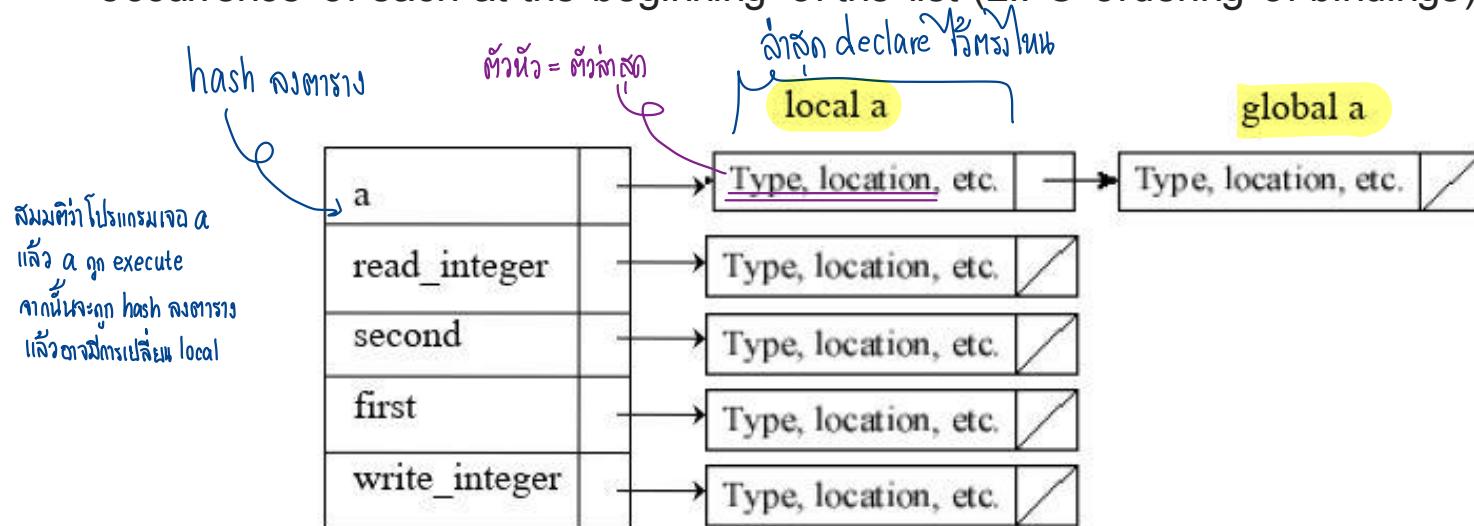


(Hash table)

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
```

