

Scope, Functions and Storage Assignment Project Exam Help Management

<https://powcoder.com>

Add WeChat powcoder
Mitchell Chapter 7

Implementing Programming Languages

- In this chapter, storage management for block-structured languages is described by the run-time data structures that are used in a simple *reference implementation*.
- Association between names in programs and memory locations is a key aspect.
Assignment Project Exam Help
 - Scope: allows two syntactically identical names to refer to different locations, or two different names to refer to the same location
 - Function calls: require a new memory area to store function parameters and local variables

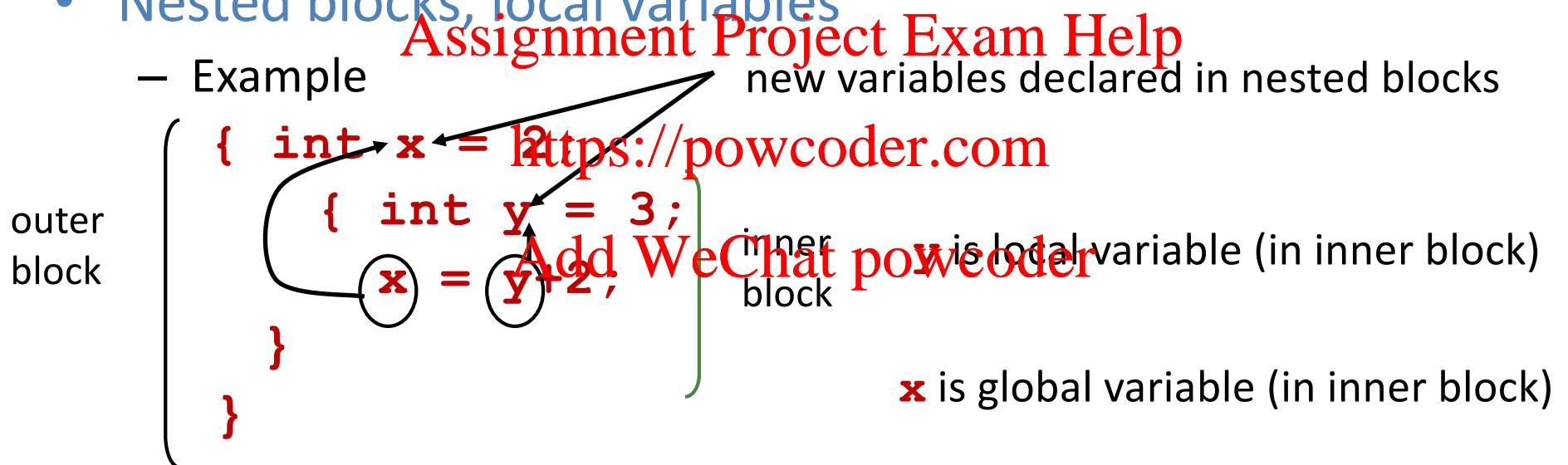
Reference Implementation

- An implementation of a language
- Designed to define the behavior of the language
- Need not be efficient
- Goal is to understand programming languages (not build a compiler) **Assignment Project Exam Help**
 - How blocks are implemented in most languages
<https://powcoder.com>
 - When storage needs to be allocated
 - What kinds of data are stored on the run-time stack
 - How executing programs access the data and location they need
 - Simple and direct because goal is understanding programming language
 - Compiler books discuss more efficient methods

Block-Structured Languages

- Blocks
 - A **block** is a region of program text, identified by begin and end markers, that may contain declarations local to this region.

- Nested blocks, local variables



- Storage management

- Enter block: allocate space for variables
- Exit block: some or all space may be deallocated

Block-Structured Languages

The diagram illustrates nested code blocks using curly braces as delimiters. The outermost brace is labeled "outer block" and encloses the entire code. The inner brace is labeled "inner block" and encloses the block starting with '{ int y = 3;'. The code itself is written in red:

```
{ int x = 2;  
    { int y = 3;  
        x = y+2;  
    }  
}
```

Assignment Project Exam Help

- Summary
 - The above is C code. This chapter also uses Algol-like pseudo-code and ML. (The ML is translated to OCaml on the slides.)
 - A local variable is a variable declared in a block
 - A global variable is a variable declared in an enclosing block.
 - In this example, in the inner block, **y** is local and **x** is global.
 - In the outer block, **x** is local and **y** is not visible.

Examples

- Blocks in common languages

- C { ... }
- Algol begin ... end
- ML (OCaml) let ... in ...

Assignment Project Exam Help

- Two forms of blocks

<https://powcoder.com>

- In-line blocks
- Blocks associated with functions or procedures

Add WeChat [powcoder](https://powcoder.com)

- Current Topic

- block-based memory management, access to local variables, parameters, global variables

Kinds of Blocks (OCaml Examples)

- Simple in-line block

```
let x=3 in  
(x,x)
```

- Nested block

```
let x=3 in      Assignment Project Exam Help  
    let y = 4 in      https://powcoder.com  
        (x,y)
```

- Functions and procedures

```
Add WeChat powcoder  
let f x = x+3 in  
f 2
```

Kinds of Function Blocks (in OCaml)

- Functions with static scope
(first-order functions)

```
let x=3 in  
  let f z = x+z in  
    f 5
```

- Functions as arguments and results (higher-order programming)

Assignment Project Exam Help
<https://powcoder.com>

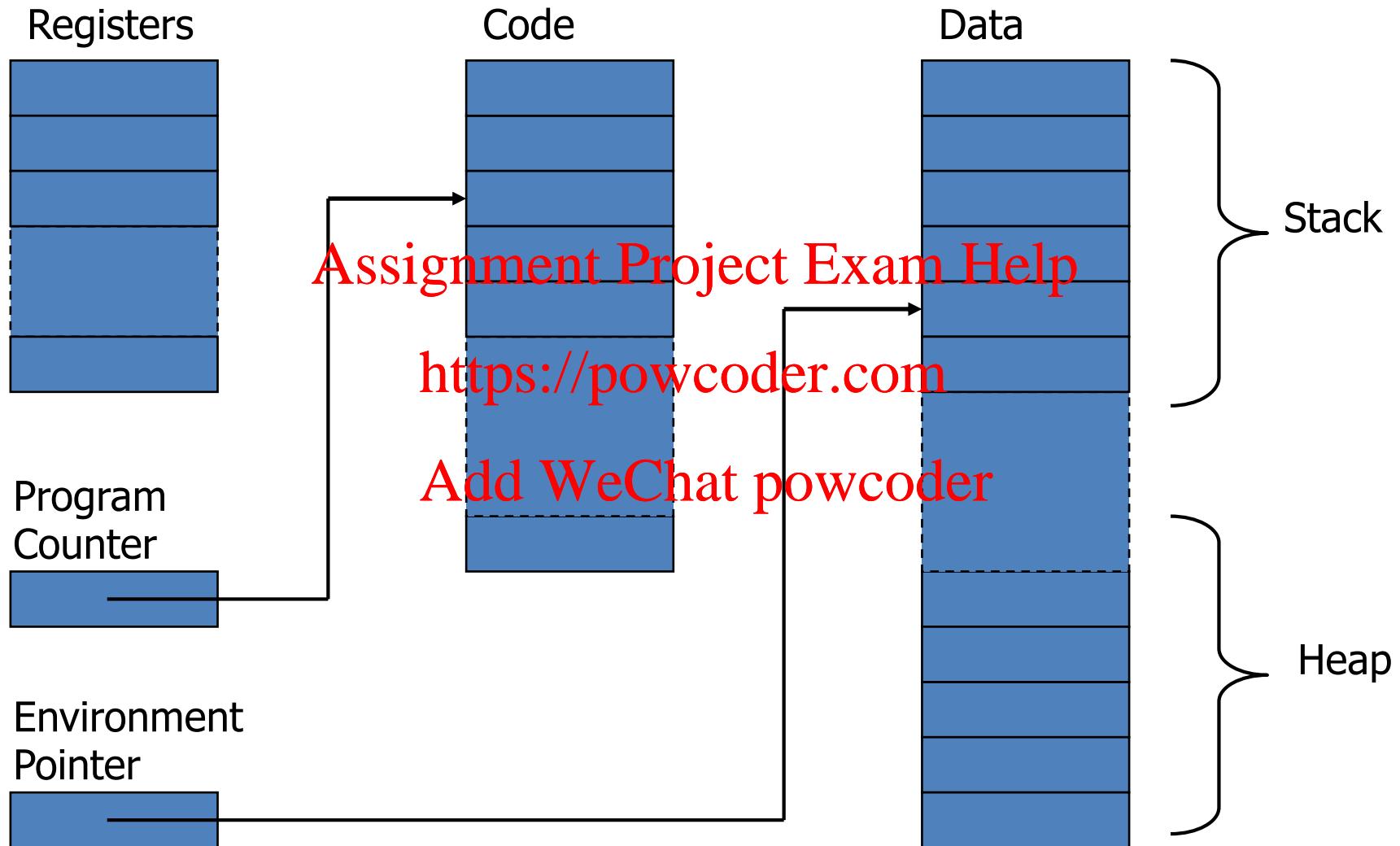
```
let compose f g =  
  (fun x -> g (f x)) in
```

- Functions returned from nested scope (with pointers/references)

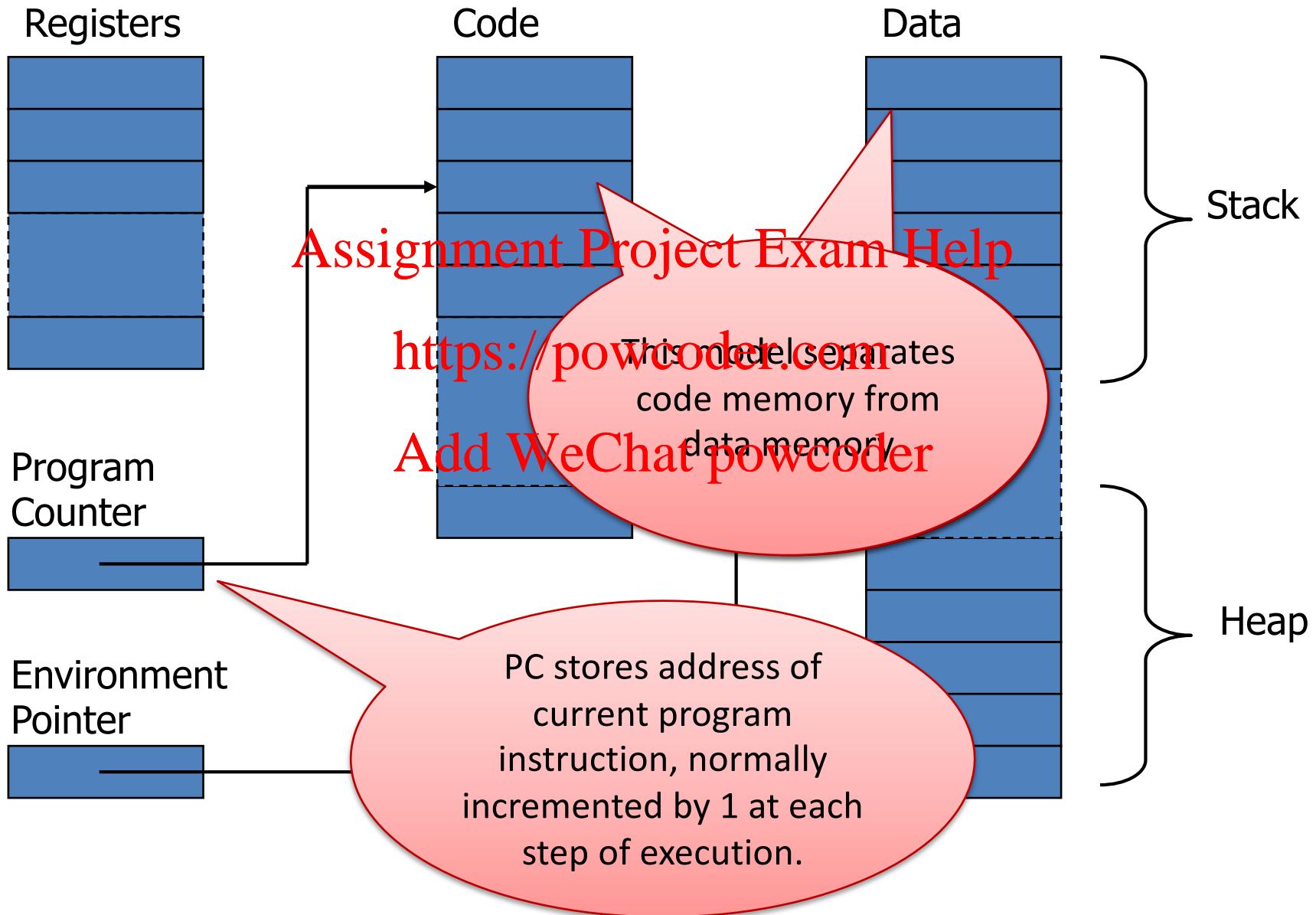
Add WeChat powcoder

```
let counter () =  
  let c = ref 0 in  
    fun () ->  
      let v = !c in  
        (c := v+1 ; v)
```

Simplified Machine Model



Simplified Machine Model



Interested in Memory Management Only

- Registers, Code segment, Program counter
 - Ignore registers
 - Details of instruction set will not matter
- Data Segment
 - Stack contains data related to block entry/exit
 - Heap contains data of varying lifetime (e.g., the cells pointed to by references) <https://powcoder.com>
 - Environment pointer points to current stack position
 - On block entry: add *new activation* record to stack
 - On block exit: remove most recent activation record
 - Activation record sometimes called a *stack frame*.

In-line Blocks

- Activation record
 - Data structure stored on run-time stack
 - Contains space for local variables
- Example

Assignment Project Exam Help

```
{ int x=0;  
    int y=x+1;  
    {  int z=(x+y)  
    };  
};
```

<https://powcoder.com>

Push record with space for **x, y**

Set values of **x, y**

Push record for inner block

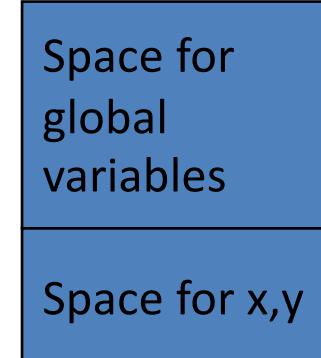
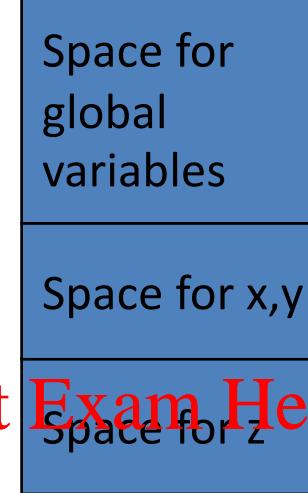
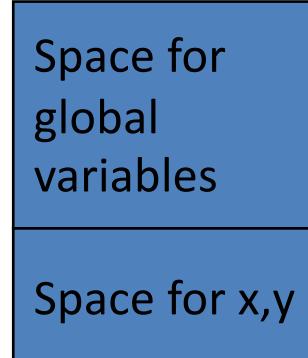
Set value of **z**

Pop record for inner block

Pop record for outer block

May need space for variables and intermediate results like **(x+y), (x-y)**

Activation Stack



Assignment Project Exam Help

Space for z

```
{ int x=0;  
    int y=x+1;  
    { int z=(x+y)AddWeChat powcoder  
    };  
};
```

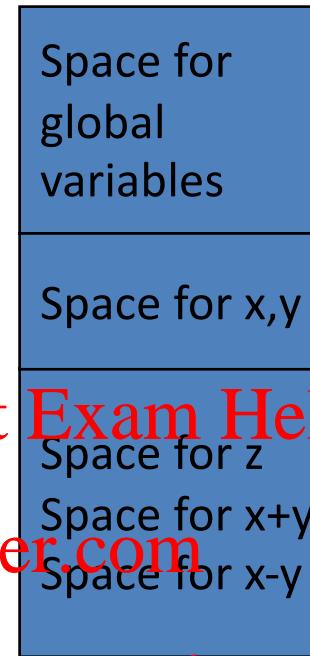
<https://powcoder.com>

Note: stacks are drawn “upside down” (they grow downward).

Activation Stack

```
{ int x=0;  
    int y=x+1;  
    {  int z=(x+y)AddWeChat powcoder  
    };  
};
```

Assignment Project Exam Help
<https://powcoder.com>
Add WeChat powcoder



Stack with space for intermediate results in the inner block.

Scope and Lifetime

- Scope
 - Region of program text where declaration is visible
- Lifetime
 - Period of time when location is allocated to program

Assignment Project Exam Help

```
{ int x = ... ;           - Inner declaration of x hides outer one.  
| { int y = ... ;         - https://powcoder.com  
| | { int x = ... ;       - Called “hole in scope”  
| | | ....  
| | | };  
| | };  
| };  
};
```

Add WeChat powcoder

- Lifetime of outer x includes time when inner block is executed
- Lifetime \neq scope
- Lines indicate “contour model” of scope.

Blocks in ML: Standard ML vs. OCaml

- let ... in ... end in Standard ML (SML) forms one block.
- equivalent to {...;...} in C

SML:

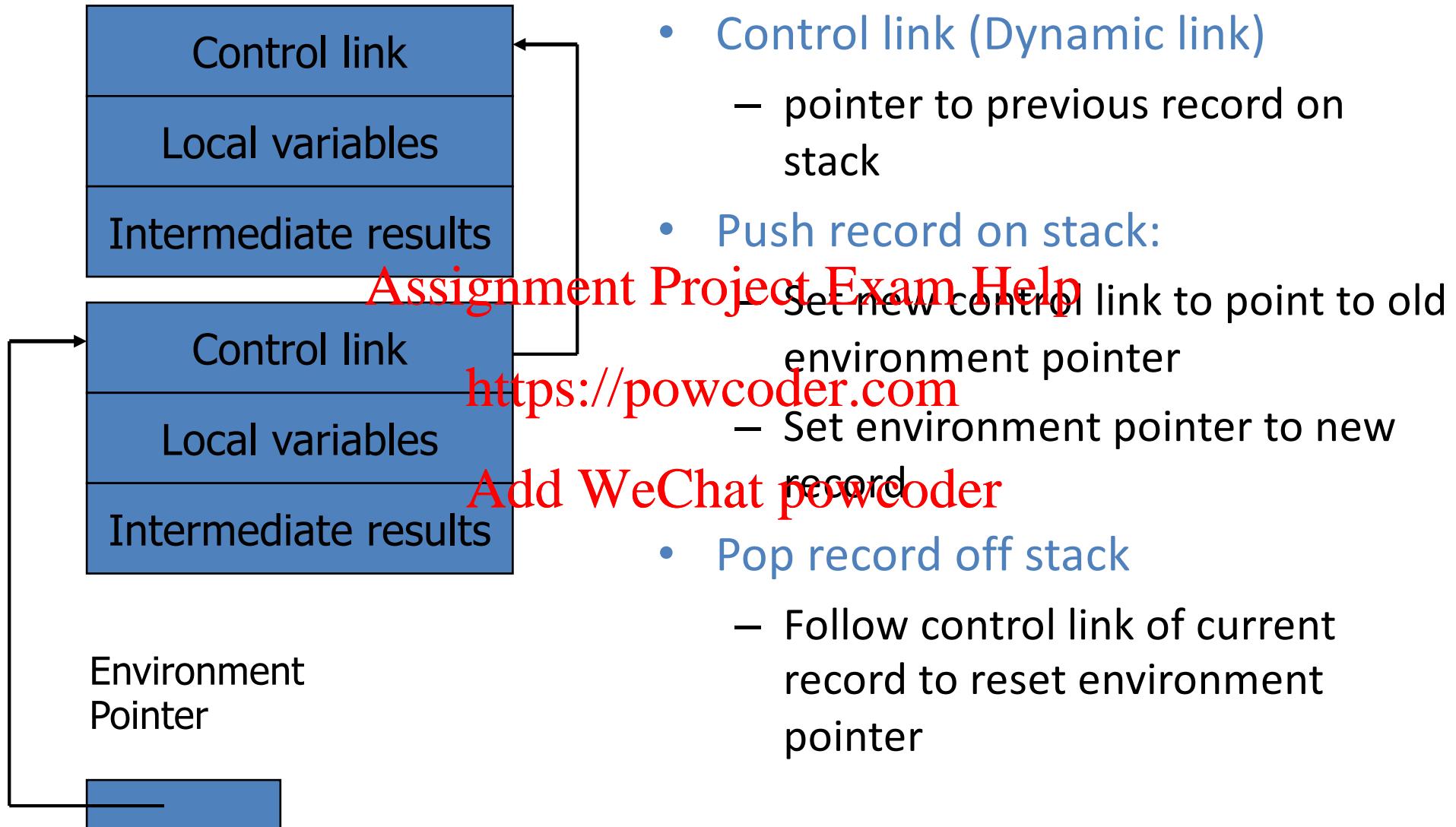
```
let fun g(y) = y+3  
    fun h(z) = g(g(z))  
in  
    h(3)  
end
```

OCaml:

```
let g y = y+3 in  
  let h z = g(g(z)) in  
    h(3)
```

- In SML, g and h will be given values in the same activation record.
- In OCaml, each let ... in ... is a separate block, but compilers usually can optimize to combine blocks.
- In SML and OCaml, ...in... separates declarations from expressions. In C, declarations and expressions can be mixed.

Activation Record for In-line Block



Control links can be optimized away, but assume not for purpose of discussion.

Example

```
{ int x=0;  
    int y=x+1;  
    { int z=(x+y)*(x-y);  
    };  
};
```

Assignment Project Exam Help

<https://powcoder.com>

Push record with space for x, y

Set values of x, y

Add WeChat powcoder

Push record for inner block

Set value of z

Pop record for inner block

Pop record for outer block

Control link	
x	0
y	1

Control link	
z	-1
$x+y$	1
$x-y$	-1

Environment
Pointer

Possible Optimizations

- Compiler can generate code for block entry that is tailored to the size of the activation record (which is known at compile time)
- Compiler can compute number of blocks between current block and location of a global variable and generate lookup code that avoids following pointer chains.

Add WeChat powcoder

Scoping Rules

- Global and local variables

- x, y are local to outer block
- z is local to inner block
- x, y are global to inner block
- z is not visible in outer block

```
{ int x=0;  
    int y=x+1;  
    { int z=(x+y)*(x-y);  
    };
```

<https://powcoder.com>

- Static scope

Add WeChat powcoder

- global refers to declaration in closest enclosing block

- Dynamic scope

- global refers to most recent activation record

Dynamic and static are the same until we consider function calls.

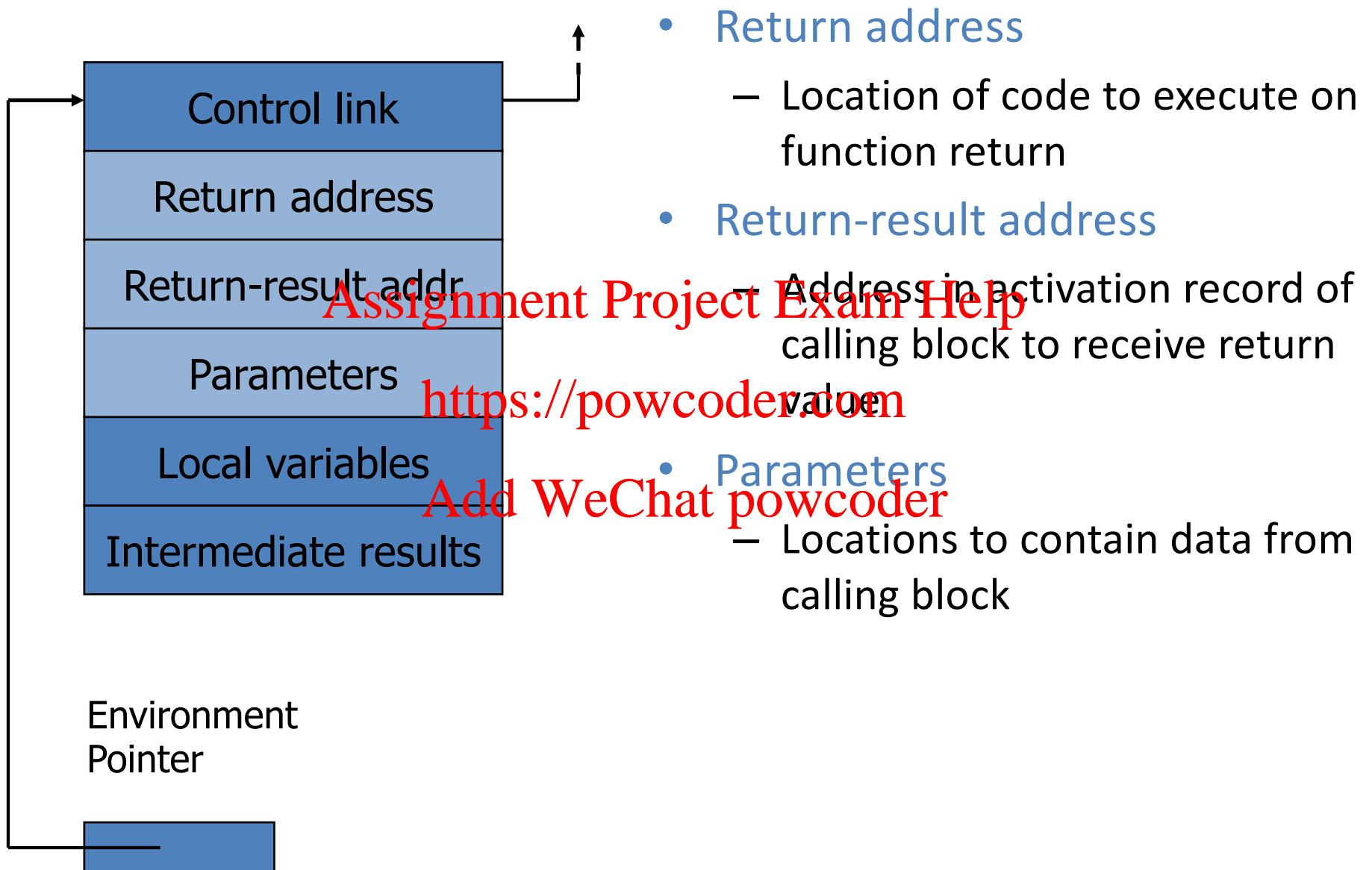
Functions and Procedures

- Syntax of procedures (Algol) and functions (C)

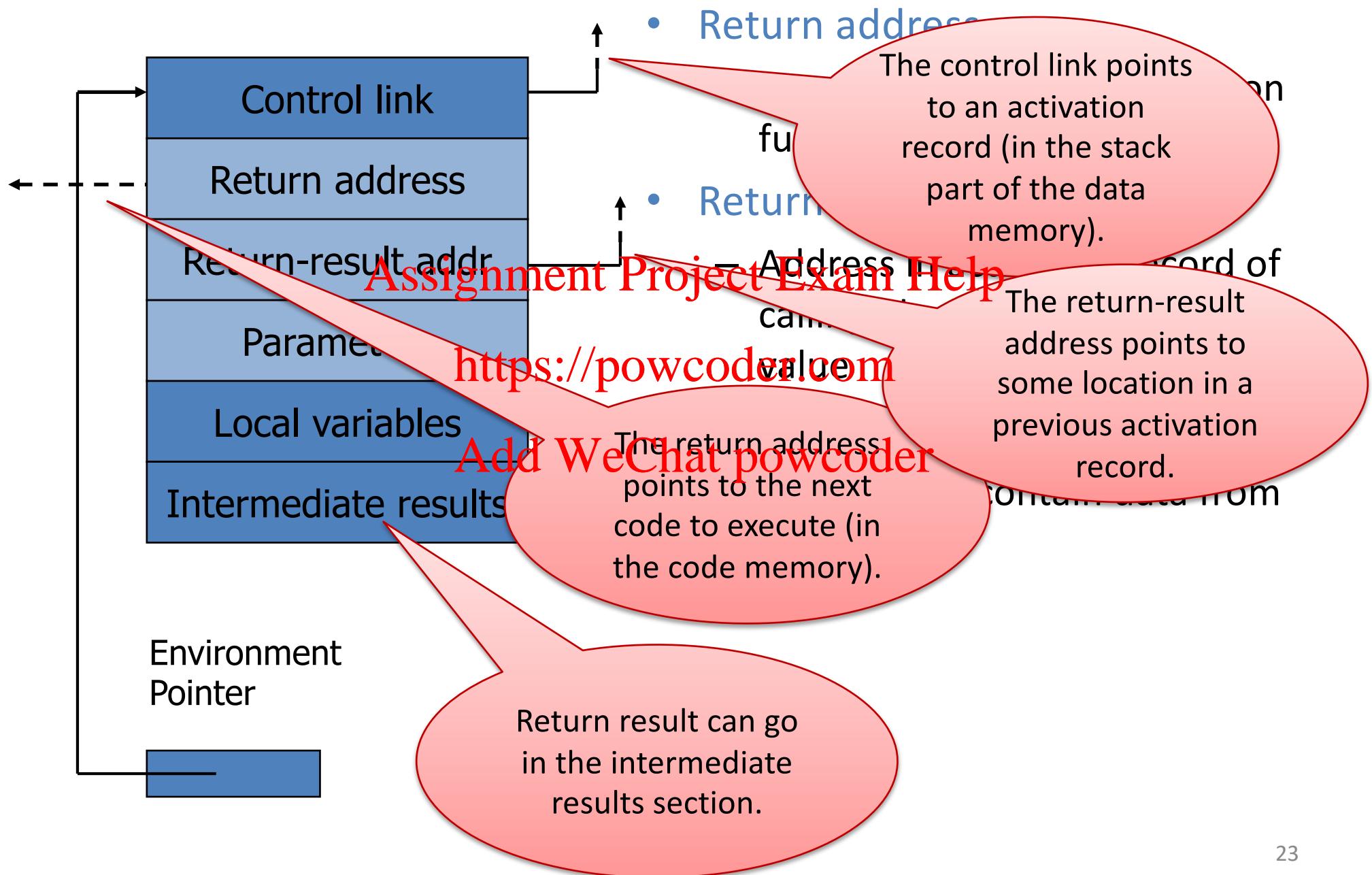
```
procedure P (<parameters>)      <type> function f(<parameters>)
begin                                {
  <local variables>           <local variables>
  <proc body>                  <function body>
end;                                     https://powcoder.com
                                         };
```

- Activation record
 - parameters
 - return address
 - return value (an intermediate result)
 - location to put return value on function exit

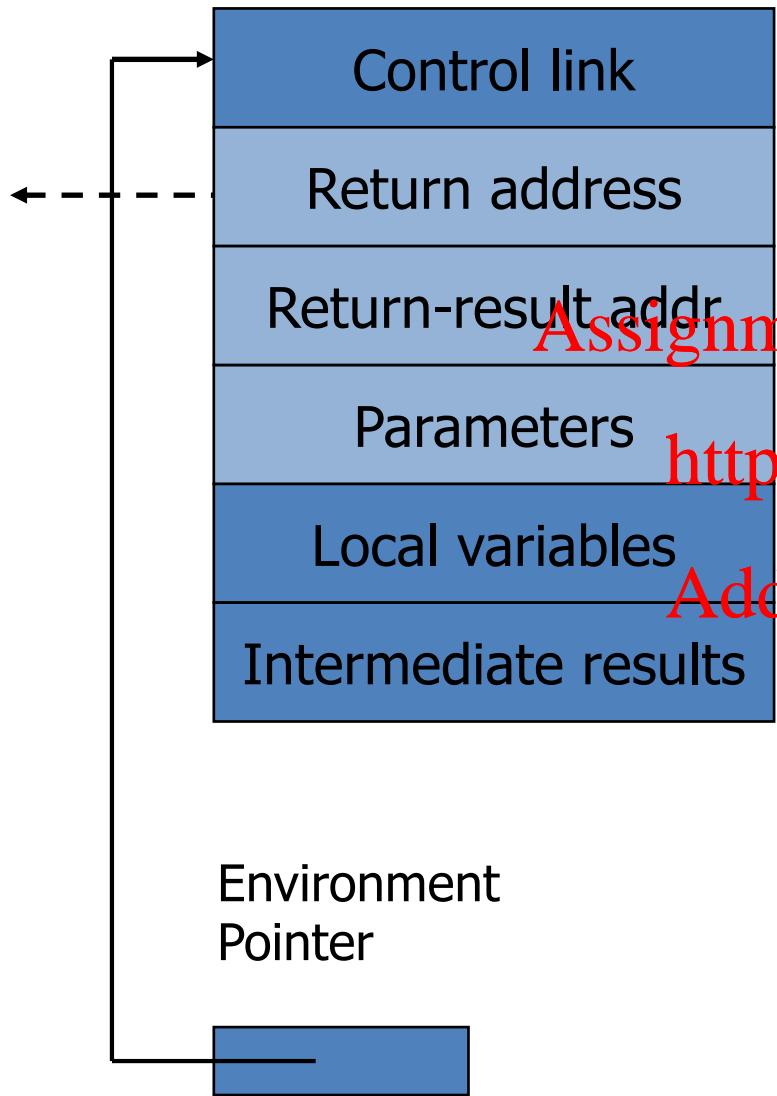
Activation Record for Function



Activation Record for Function



Example



- Function

$\text{fact}(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ n * \text{fact}(n-1) & \text{else} \end{cases}$

- Return-result address

— location to put the result of the function evaluation $\text{fact}(n)$

- Parameter

— set to value of n when function is called

- Local variables

— none

- Intermediate result

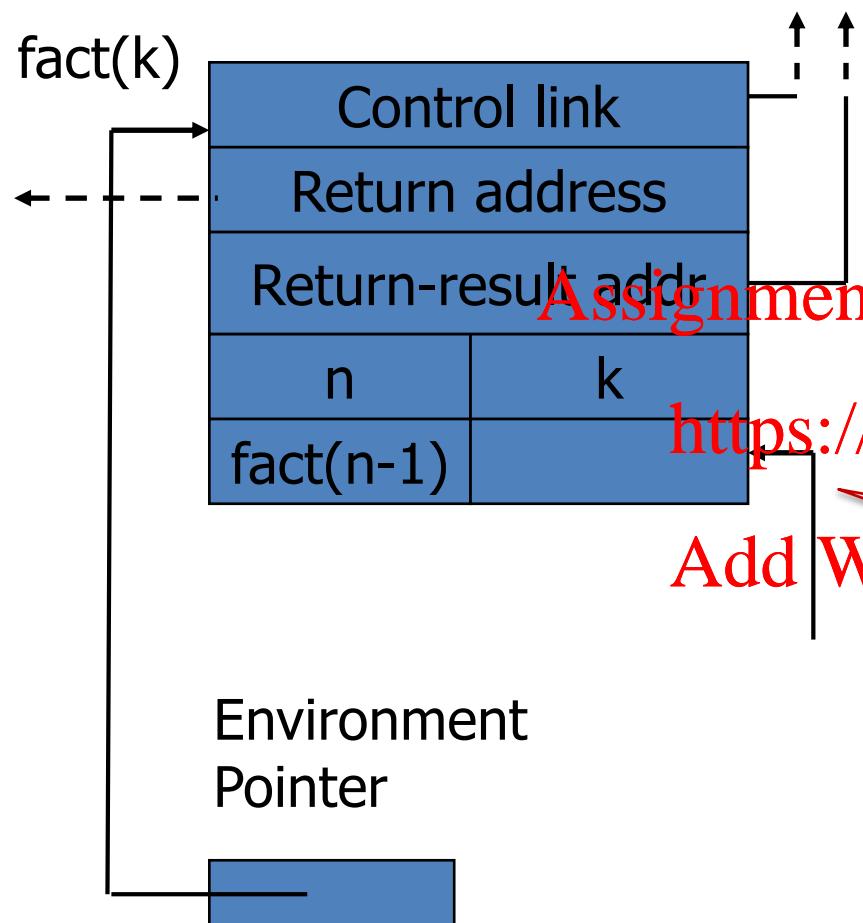
— value of $\text{fact}(n-1)$

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Function Call



$\text{fact}(n) = \begin{cases} 1 & \text{if } n \leq 1 \\ n * \text{fact}(n-1) & \text{else} \end{cases}$

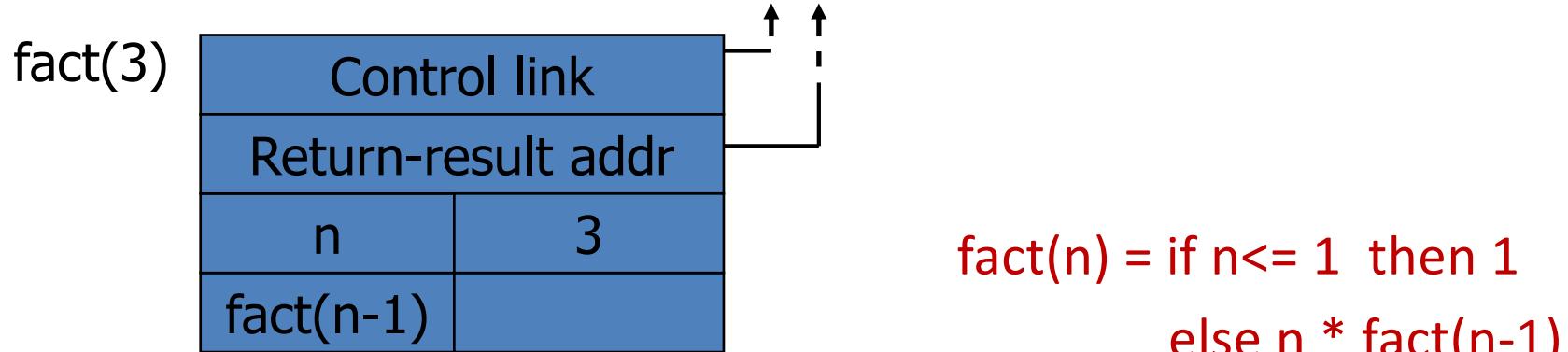
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

the return-result
address of recursive
call

Function Call



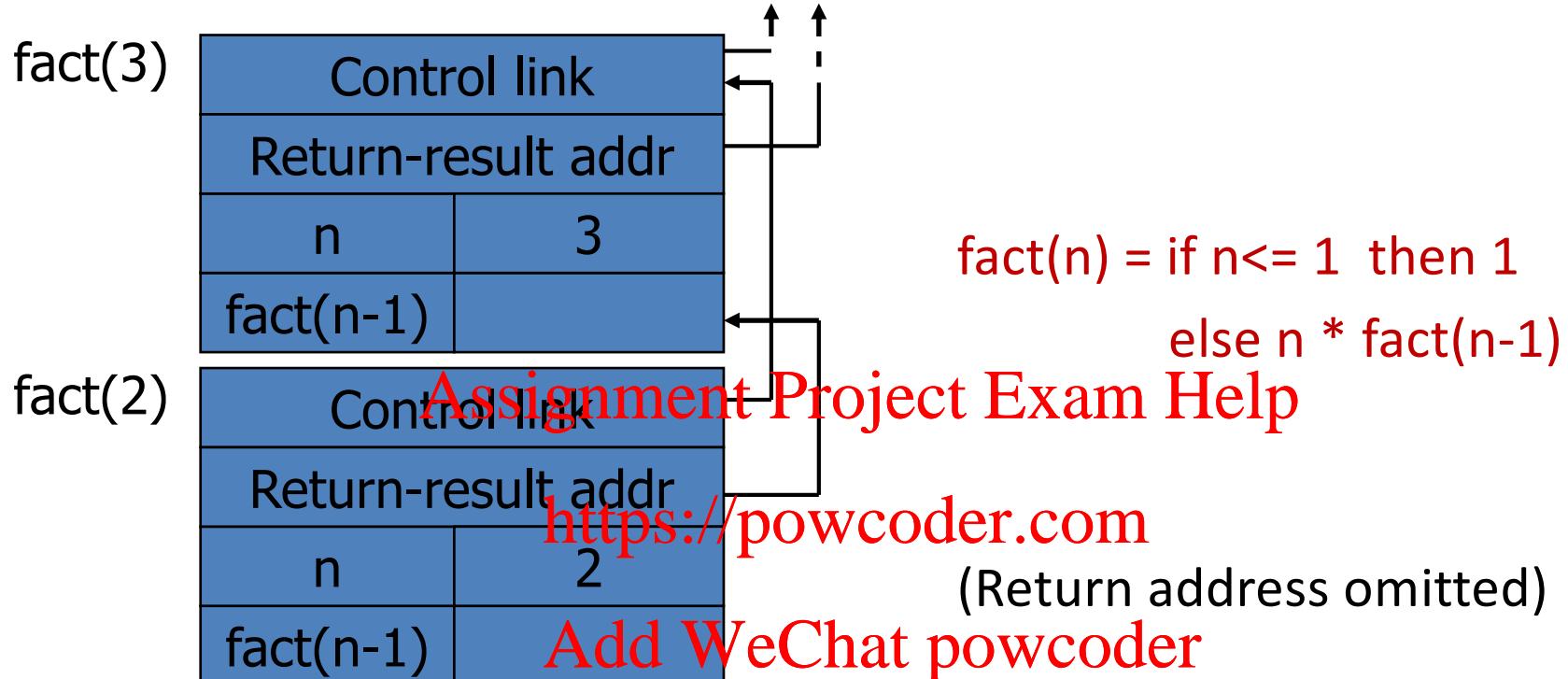
Assignment Project Exam Help

<https://powcoder.com>

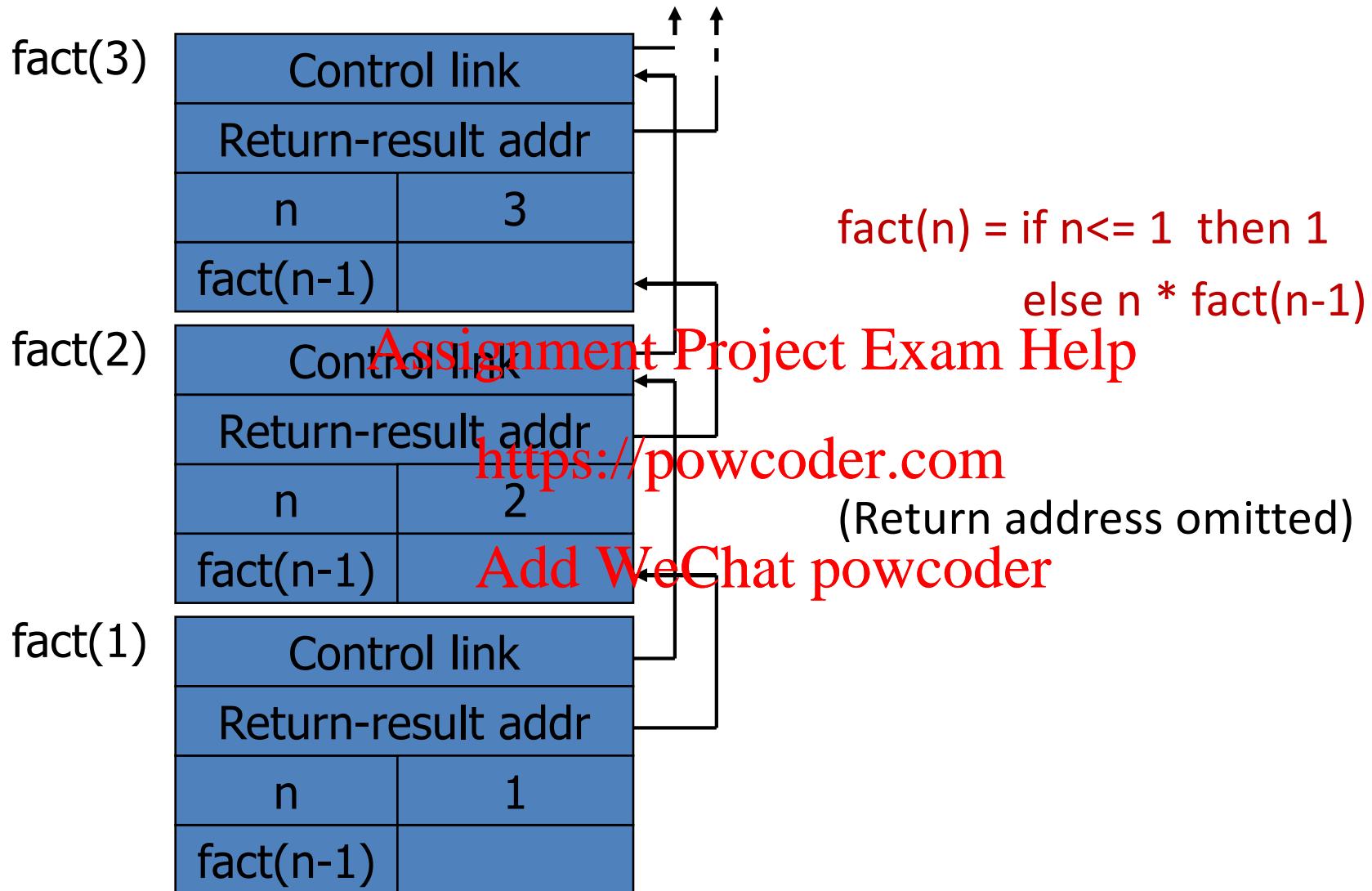
(Return address omitted)

Add WeChat powcoder

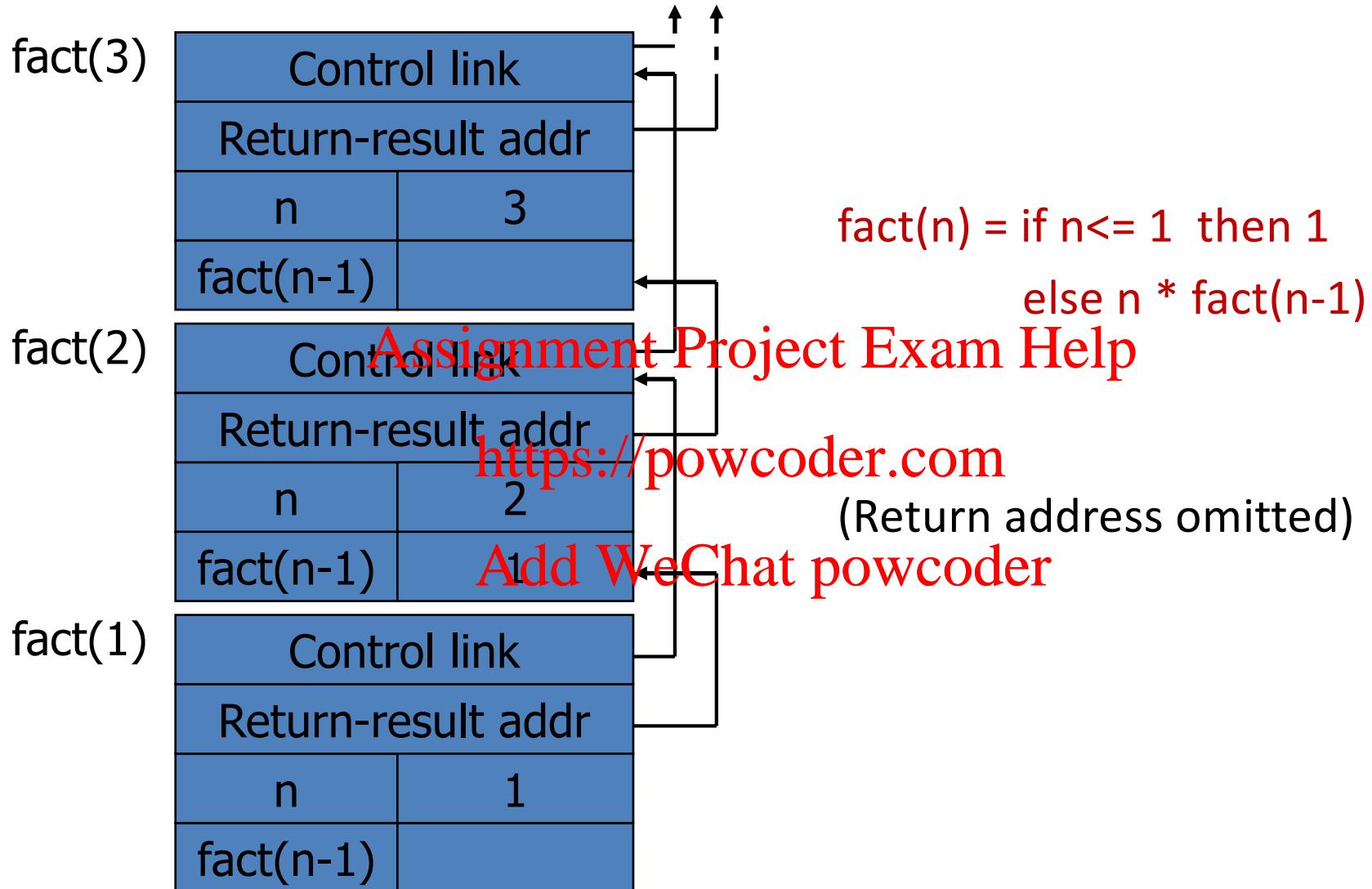
Function Call



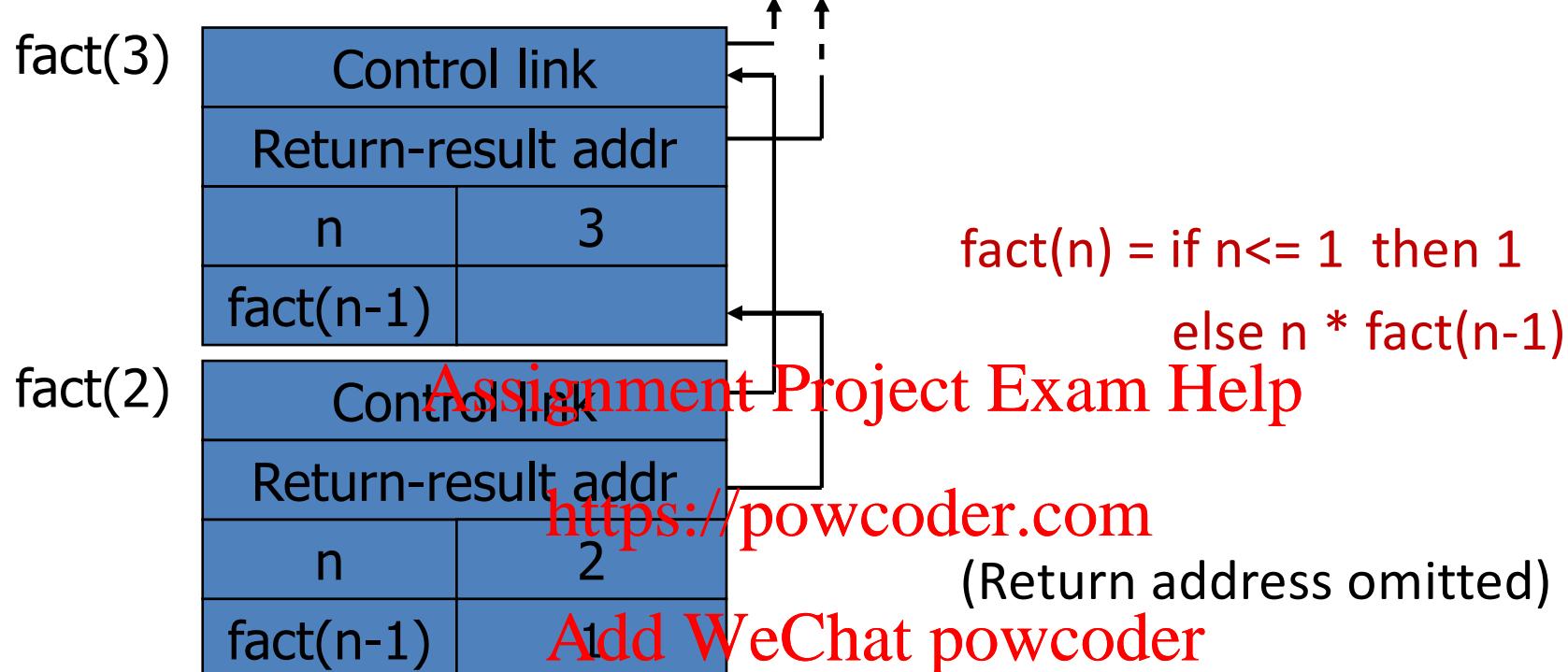
Function Call



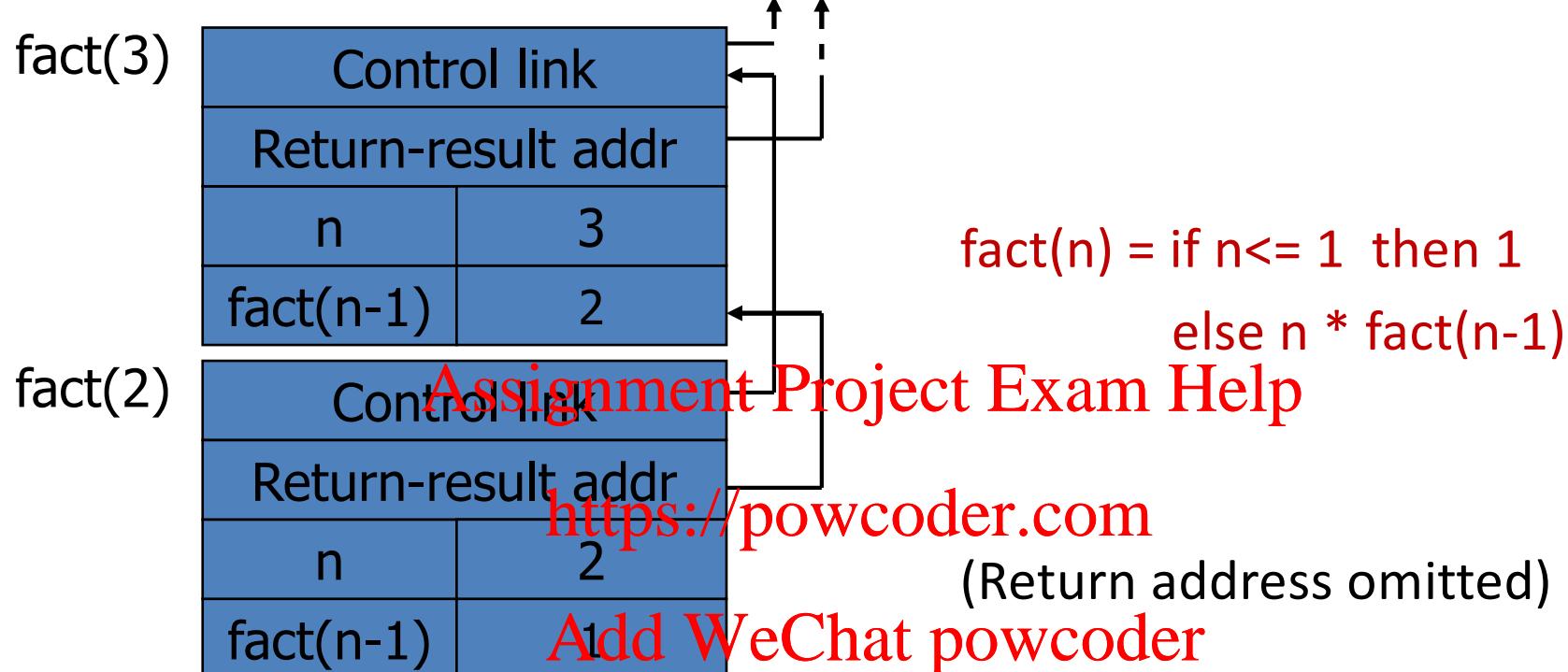
Function Return



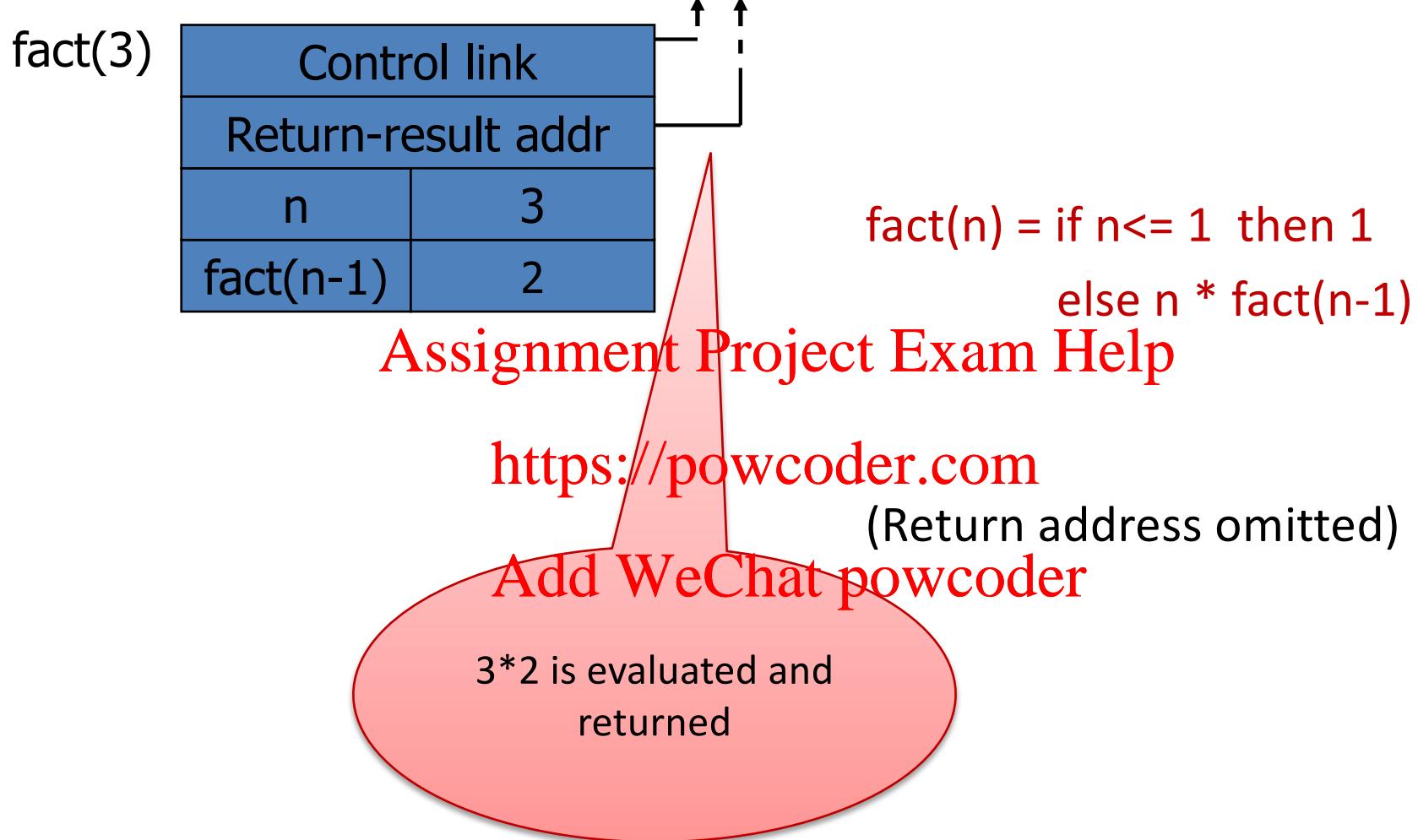
Function Return



Function Return



Function Return



Topics for First-Order Functions

- Parameter passing
 - use ML reference cells to describe pass-by-value, pass-by-reference
- Access to global variables
 - global variables are contained in an activation record higher “up” the stack
- Tail recursion
 - an optimization for certain recursive functions

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Parameter Passing

```
proc p (int x, int y) {  
    if (x > y) then ... else ...;  
    ...  
    x := y*2+3;  
    ...  
}  
p (z, 4*z+1);
```

- The identifiers **x** and **y** are *formal parameters* of the procedure **p**.
- The *actual parameters* in the call to **p** are **z** and **4*z+1**.
 - The way that actual parameters are evaluated and passed to the function depends on the kind of parameter-passing mechanism that is used (different for different languages).

Assignment Project Exam Help

<https://powcoder.com>
Add WeChat **powcoder**

ML Imperative Features

- General terminology: L-values and R-values
 - Assignment $y := x+3$
 - Identifier on left refers to location, called its L-value
 - Identifier on right refers to contents, called R-value
- Recall ML (OCaml) reference cells and assignment
 - Different types for location and contents
 - $x : \text{int}$ non-assignable integer value
 - $y : \text{int ref}$ location whose contents must be integer
 - $!y$ the contents of cell y
 - $\text{ref } x$ expression creating new cell initialized to x
 - ML form of assignment
 - $y := x+3$ place value of $x+3$ in location (cell) y
 - $y := !y + 3$ add 3 to contents of y and store in location y

Parameter Passing

- Pass-by-value
 - Caller places R-value (**contents**) of actual parameter in activation record
 - Function cannot change value of caller's variable
 - Reduces aliasing (alias: two names refer to same location)
- Pass-by-reference
 - Caller places L-value (**address**) of actual parameter in activation record
 - Function can assign to variable that is passed

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Example

Pseudo-code:

```
function f (x) =  
    { x := x+1; return x }  
  
var y : int = 0;  
f(y)+y;
```

pass-by-reference

pass-by-value

OCaml:

```
let f (x : int ref) =  
    ( x := !x+1; !x ) in  
let y = ref 0 in  
let z = f(y) in  
z + !y
```

```
Add WeChat powcoder  
let f (w : int) =  
    (let x = ref w in  
     x := !x+1; !x) in  
let y = ref 0 in  
let z = f(!y) in  
z + !y
```

Assignment Project Exam Help

Access to Global Variables

- Two possible scoping conventions, recall:
 - Static scope: global refers to declaration in closest enclosing block
 - Dynamic scope: global refers to most recent activation record
 - As before, each let ... in ... is a separate block, but only at the top level. Inside function definitions a series of let ... in ... declarations in parentheses will be considered local variables of the function. <https://powcoder.com>
- Example

```
let x = 1 in  
let g z = x + z in  
let f y =  
  (let x = y + 1 in  
    g (y*x)) in  
f 3
```

Skeleton of Activation Stack
Add WeChat powcoder

outer block	x	1
f(3)	y	3
	x	4
g(12)	z	12

Which x is used for expression x+z?

Access to Global Variables

Which **x** is used for expression **x+z**?

- **Dynamic Scope**

Under dynamic scope, the identifier **x** in **x+z** will be interpreted as the one from the most recently created activation record, namely **x=4**.

Assignment Project Exam Help

How do we implement access to global variables?

<https://powcoder.com>

Answer: follow the control link (which is also called a *dynamic link*).

Add WeChat powcoder

```
let x = 1 in
let g z = x + z in
let f y =
  (let x = y + 1 in
    g (y*x)) in
f 3
```

outer block	x	1
f(3)	y	3
	x	4
g(12)	z	12

Access to Global Variables

Which **x** is used for expression **x+z**?

- **Static Scope**

Under static scope, the identifier **x** in **x+z** refers to the declaration of **x** from the closest enclosing program block, looking upward from the place that **x+z** appears in the program. Thus, **x=1**.

OCaml is static.

<https://powcoder.com>

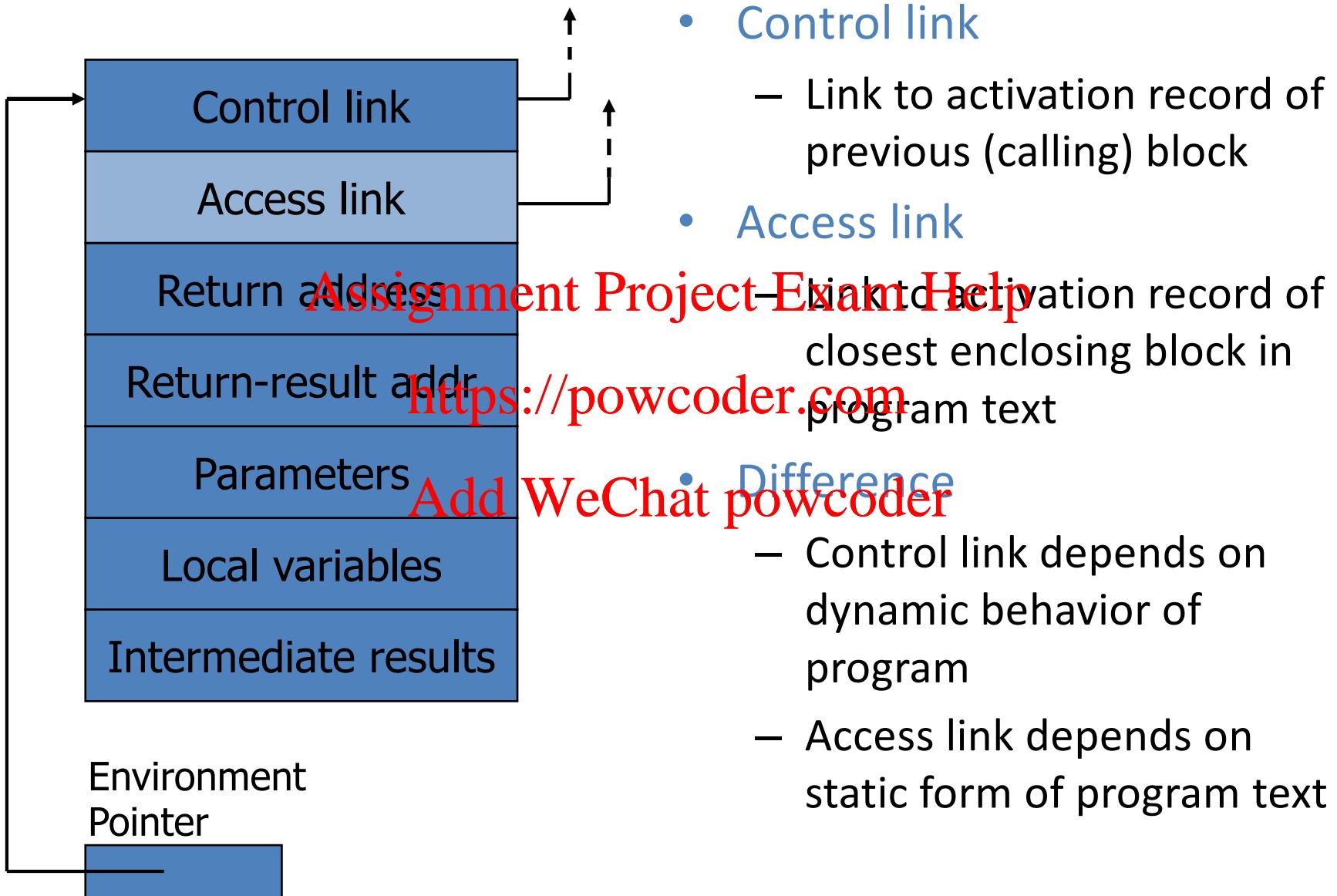
How do we implement access to global variables in this case?

Add WeChat powcoder

```
let x = 1 in  
let g z = x + z in  
let f y =  
  (let x = y + 1 in  
   g (y*x)) in  
f 3
```

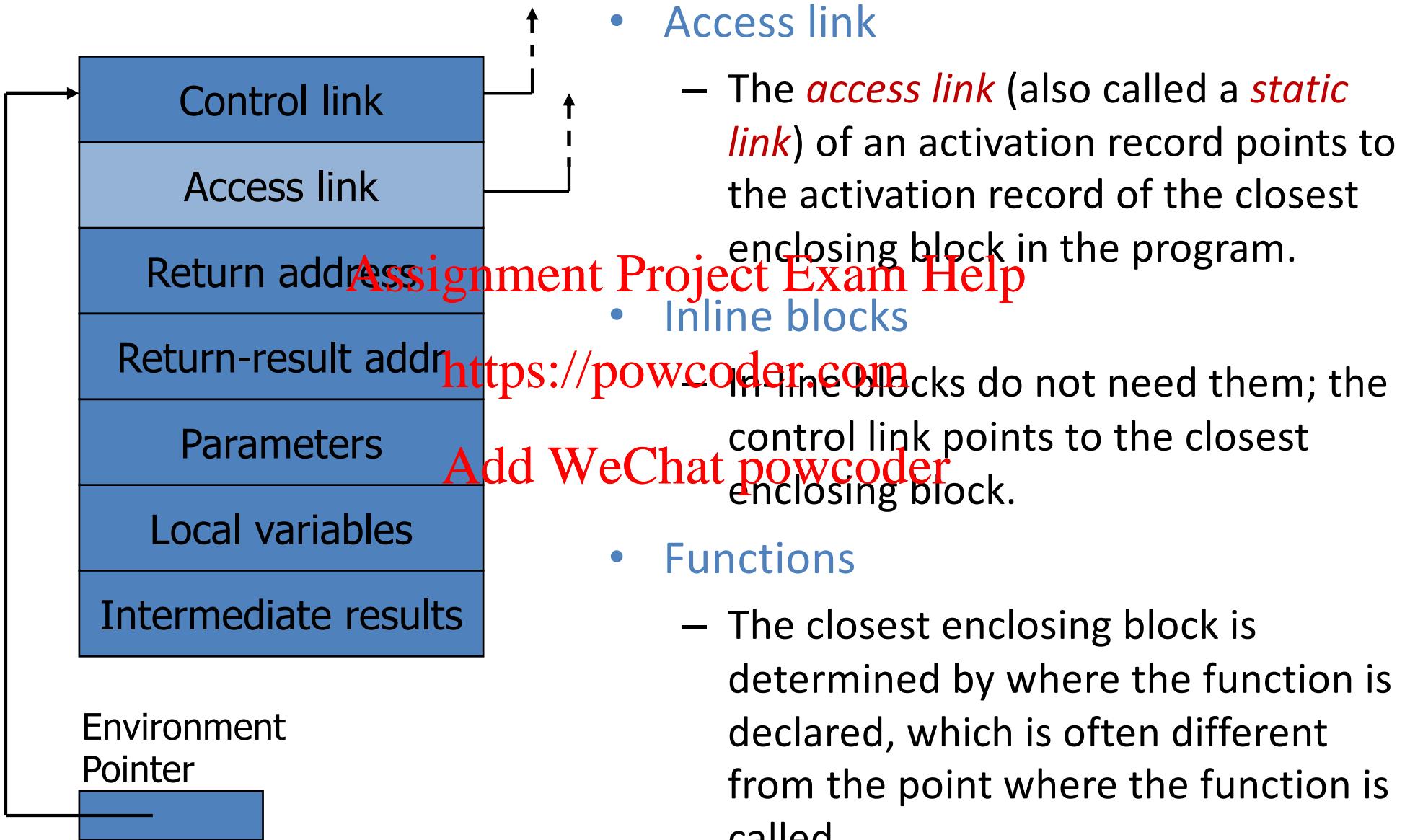
outer block	x	1
f(3)	y	3
	x	4
g(12)	z	12

Activation Record for Static Scope



- **Control link**
 - Link to activation record of previous (calling) block
- **Access link**
 - Link to activation record of closest enclosing block in program text
- Control link depends on dynamic behavior of program
- Access link depends on static form of program text

Access Links for Static Scope



Static Scope with Access Links

```
let x = 1 in
```

```
  let g z = x + z in
```

```
    let f y =
```

```
      (let x = y + 1 in
```

```
        g (y*x))
```

```
f 3
```

outer block

control link	
access link	
x	1

Assignment Project Exam Help

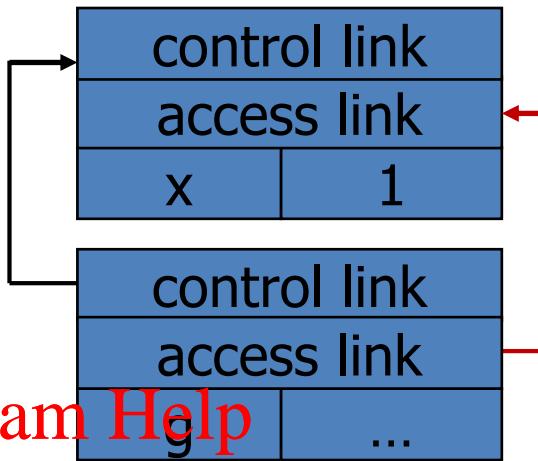
<https://powcoder.com>

Add WeChat powcoder

Static Scope with Access Links

```
let x = 1 in  
  let g z = x + z in  
    let f y =  
      (let x = y + 1 in  
        g (y*x))  
    f 3
```

outer block



Use access link to find ~~global~~ WeChat powcoder variable:

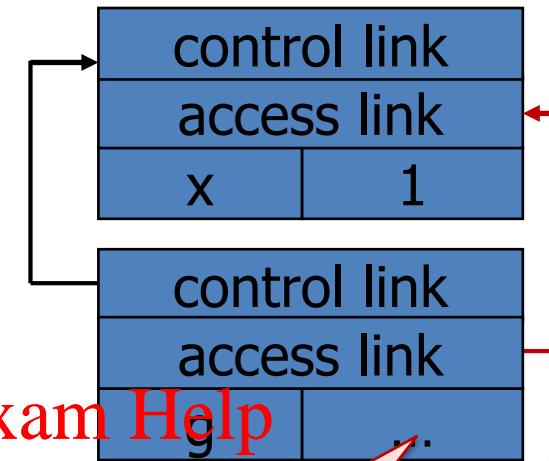
- Access link is always set to frame of closest enclosing lexical block

<https://powcoder.com>

Static Scope with Access Links

```
let x = 1 in  
  let g z = x + z in  
    let f y =  
      (let x = y + 1 in  
        g (y*x))  
    f 3
```

outer block



Use access link to find global variable:

Assignment Project Exam Help
<https://powcoder.com>

Add WeChat powcoder includes pointer to code

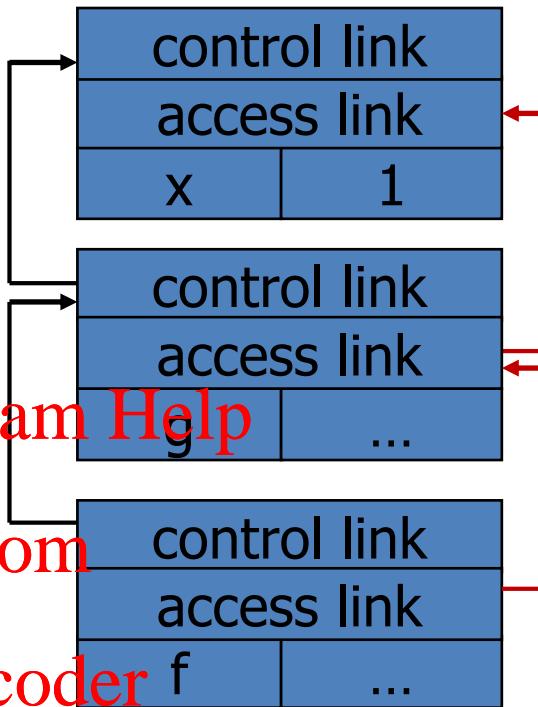
- Access link is always set to frame of closest enclosing lexical block

For declarations, access link and control link are the same.

Static Scope with Access Links

```
let x = 1 in  
  let g z = x + z in  
    let f y =  
      (let x = y + 1 in  
        g (y*x))  
    f 3
```

outer block



Use access link to find global variable:
Add WeChat powcoder

- Access link is always set to frame of closest enclosing lexical block

Static Scope with Access Links

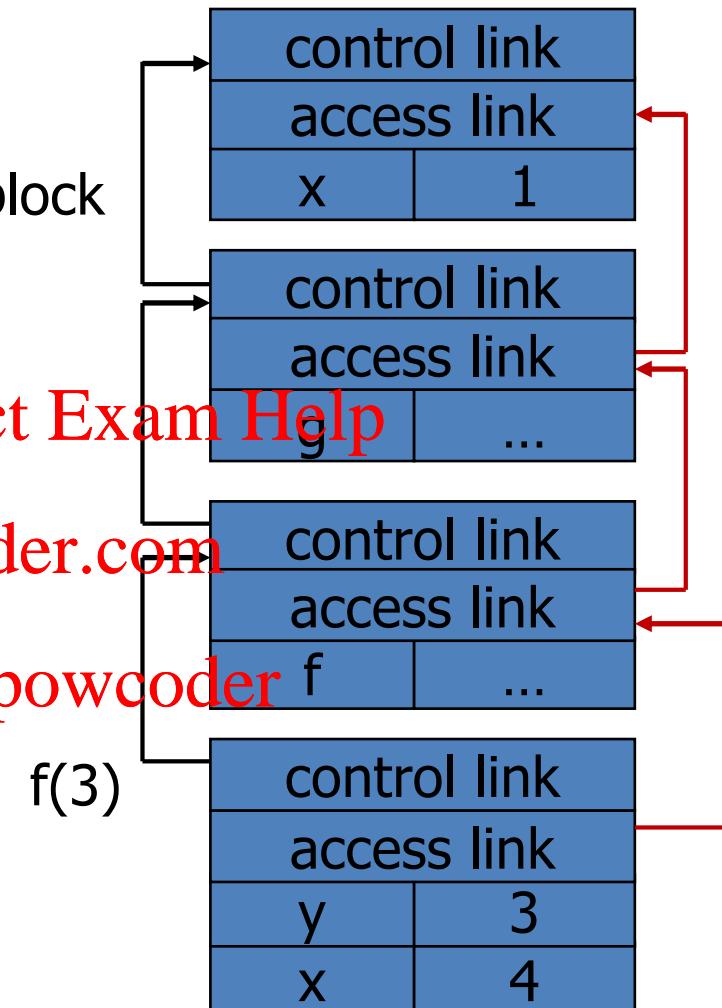
```
let x = 1 in  
  let g z = x + z in  
    let f y =  
      (let x = y + 1 in  
        g (y*x))  
    f 3
```

outer block

Assignment Project Exam Help
<https://powcoder.com>

Use access link to find global variable:

- Access link is always set to frame of closest enclosing lexical block
- For function body, this is block that contains function declaration



Static Scope with Access Links

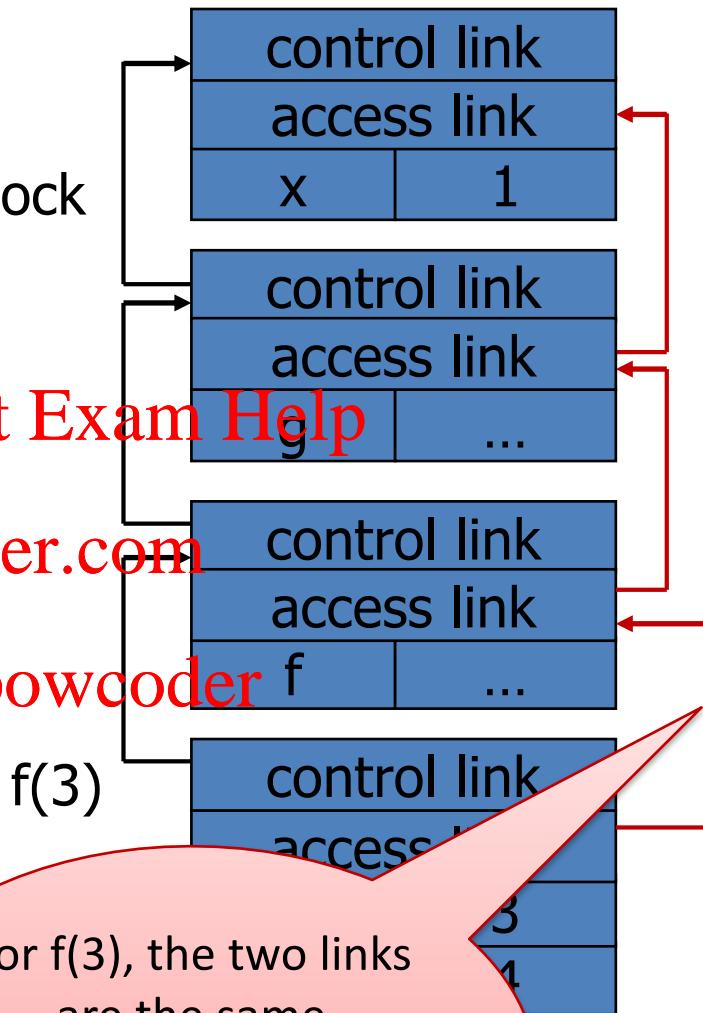
```
let x = 1 in  
  let g z = x + z in  
    let f y =  
      (let x = y + 1 in  
        g (y*x))  
    f 3
```

outer block

Assignment Project Exam Help
<https://powcoder.com>

Use access link to find global variable:

- Access link is always set to frame of closest enclosing lexical block
- For function body, this is block that contains function declaration



For `f(3)`, the two links are the same because `f` is called right after it is defined

Static Scope with Access Links

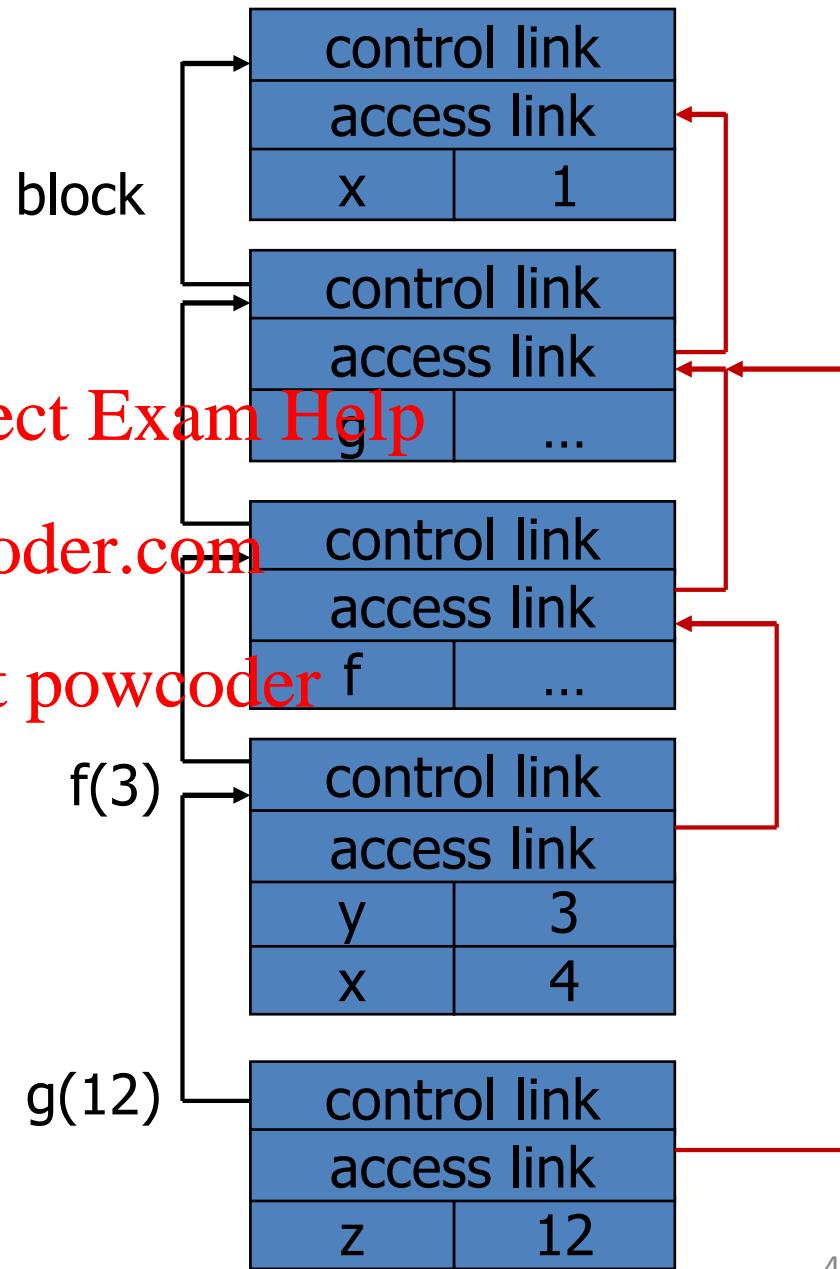
```
let x = 1 in  
  let g z = x + z in  
    let f y =  
      (let x = y + 1 in  
        g (y*x))  
      f 3
```

outer block

Assignment Project Exam Help
<https://powcoder.com>

Use access link to find global variable:

- Access link is always set to frame of closest enclosing lexical block
- For function body, this is block that contains function declaration



Static Scope with Access Links

```
let x = 1 in  
  let g z = x + z in  
    let f y =  
      (let x = y + 1 in  
        g (y * x))  
    f 3
```

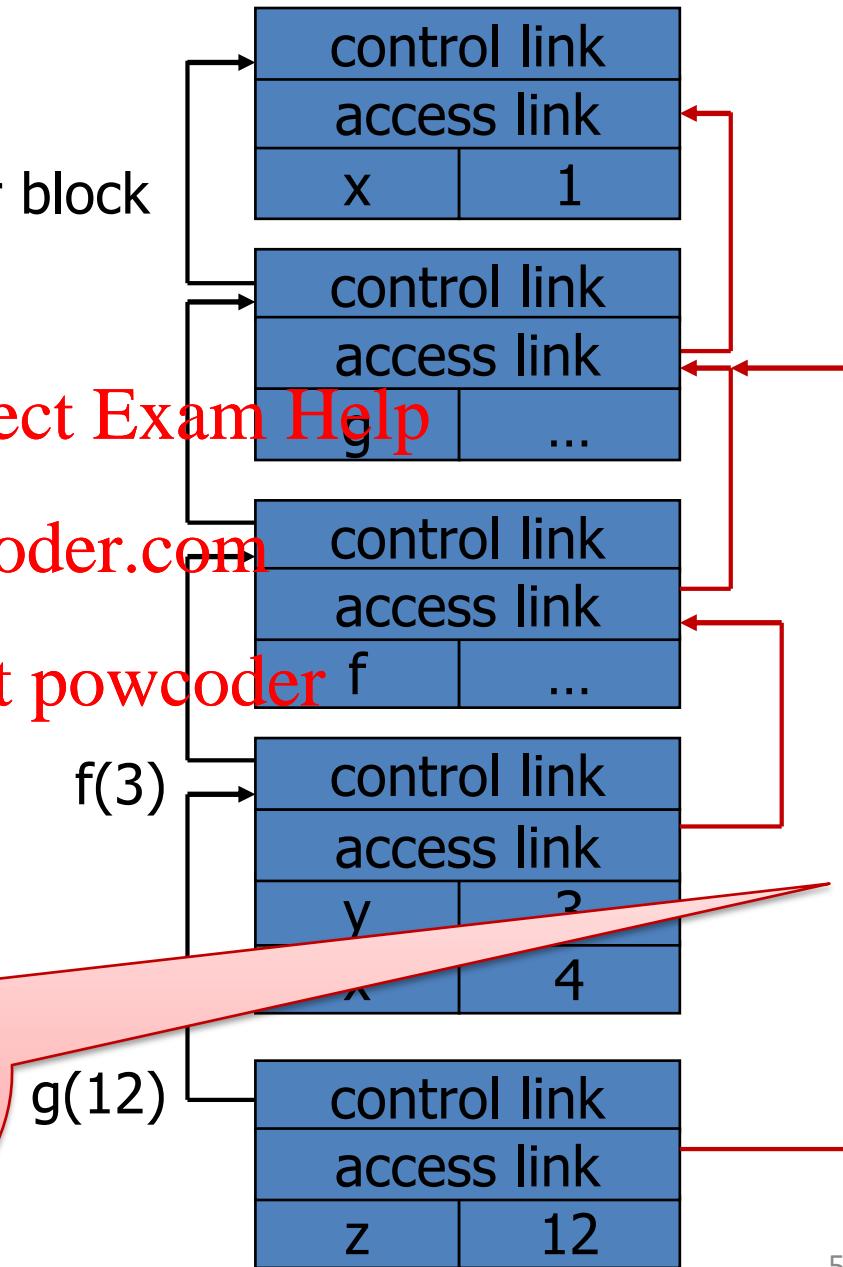
outer block

Assignment Project Exam Help
<https://powcoder.com>

Use access link to find global variable:

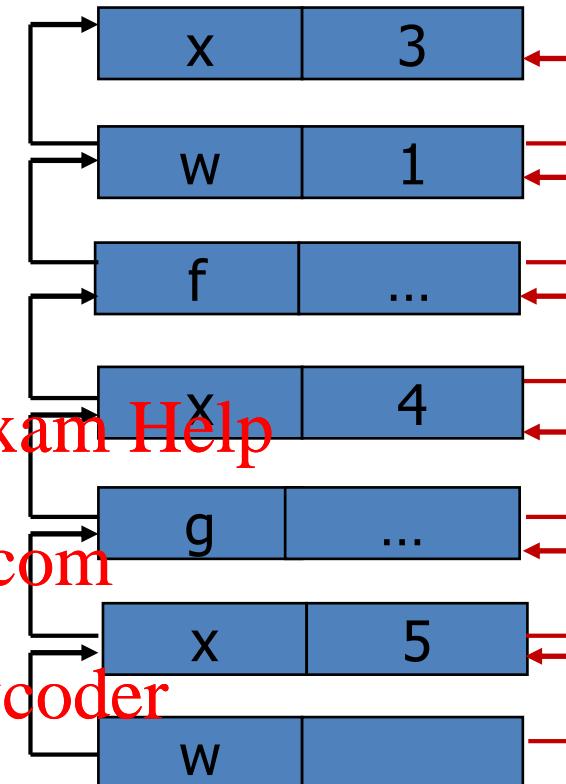
- Access link is always + to frame of closest enclosing block
- For functions that contain declarations

The compiler can figure this out from program text (g called from inside f)



Another Example

```
let x = 3 in  
let w = 1 in  
let f y = w+x+y in  
let x = 4 in  
let g z = f(x*z) in  
let x = 5 in  
let w = https://powcoder.com  
w  
Access links are red. Add WeChat powcoder  
Control links are black.
```

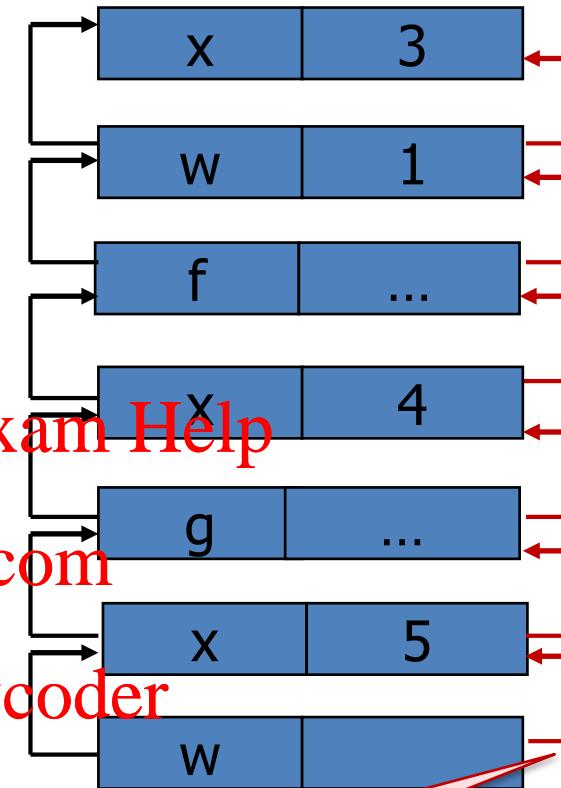


Another Example

```
let x = 3 in  
let w = 1 in  
let f y = w+x+y in  
let x = 4 in  
let g z = f(x*z) in  
let x = 5 in  
let w = https://powcoder.com in  
w
```

Access links are red. Add WeChat powcoder

Control links are black.



Follow access link
to find the value of
x for function call
(g x). Value is 5.

Another Example

```

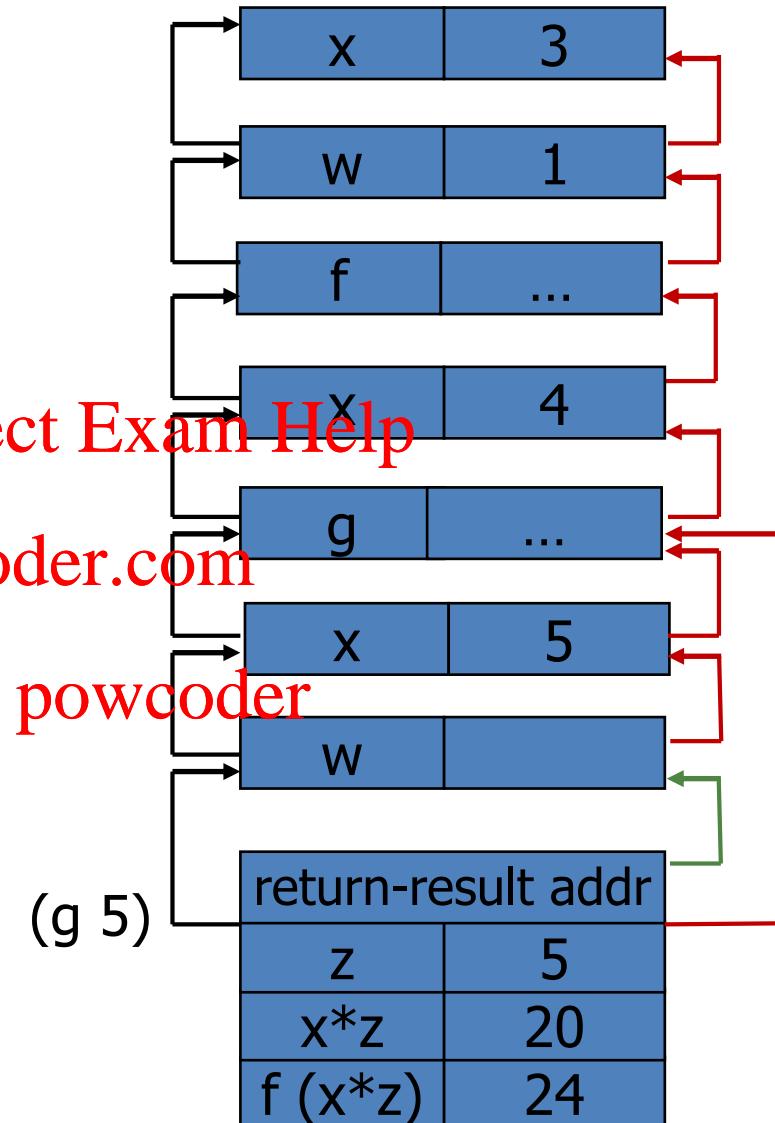
let x = 3 in
let w = 1 in
let f y = w+x+y in
let x = 4 in
let g z = f (x*z) in
let x = 5 in
let w = https://powcoder.com
w
  
```

Access links are red. Add WeChat powcoder

Control links are black.

Return-result addr links are green.

Note that access and control links point to a block and return-result links point to a single location in a block.



Another Example

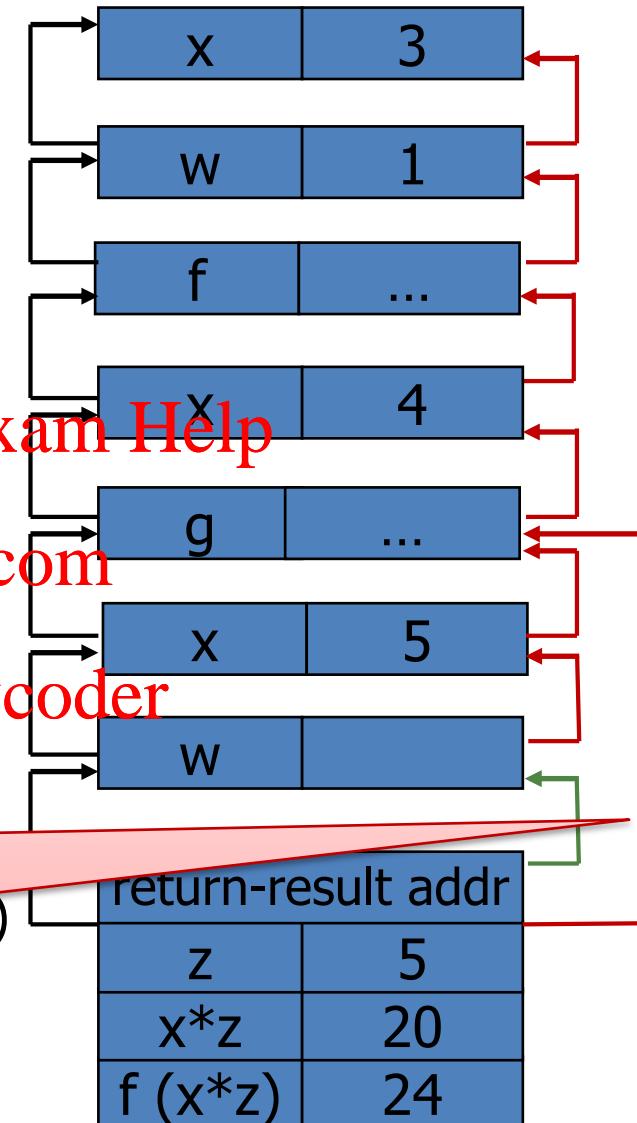
```

let x = 3 in
let w = 1 in
let f y = w+x+y in
let x = 4 in
let g z = f (x*z) in
let x = 5 in
let w = https://powcoder.com

```

Access link
Control
Return
green
Note that
links point
and
return-result links point to a
single location in a block.

Add WeChat powcoder
Follow access link
to find the value of
x used in
expression $x*z$.
Value is 4, and
thus $x*z$ is 20.



Another Example

```

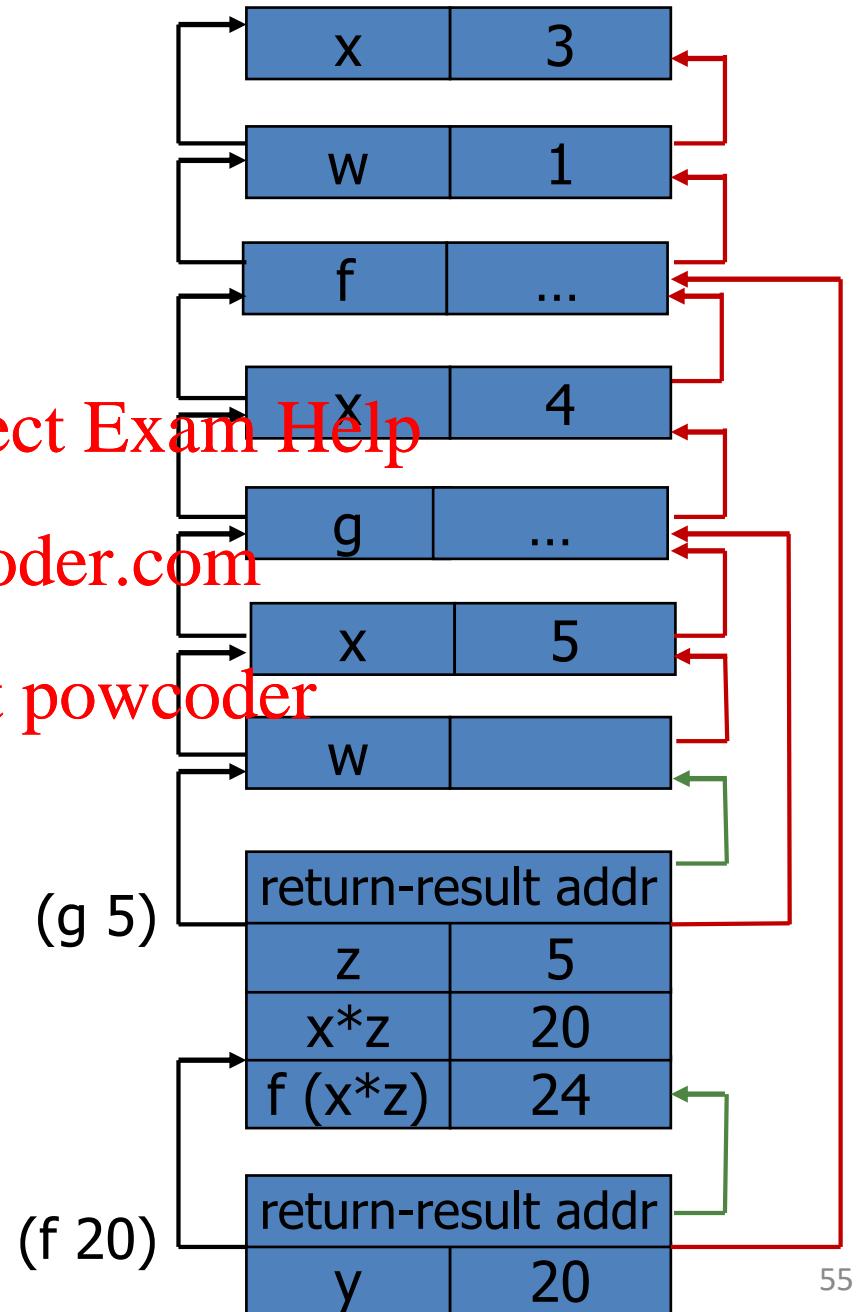
let x = 3 in
let w = 1 in
let f y = w+x+y in
let x = 4 in
let g z = f (x*z) in
let x = 5 in
let w = https://powcoder.com
w
  
```

Access links are red. Add WeChat powcoder

Control links are black.

Return-result addr links are green.

Note that access and control links point to a block and return-result links point to a single location in a block.



Another Example

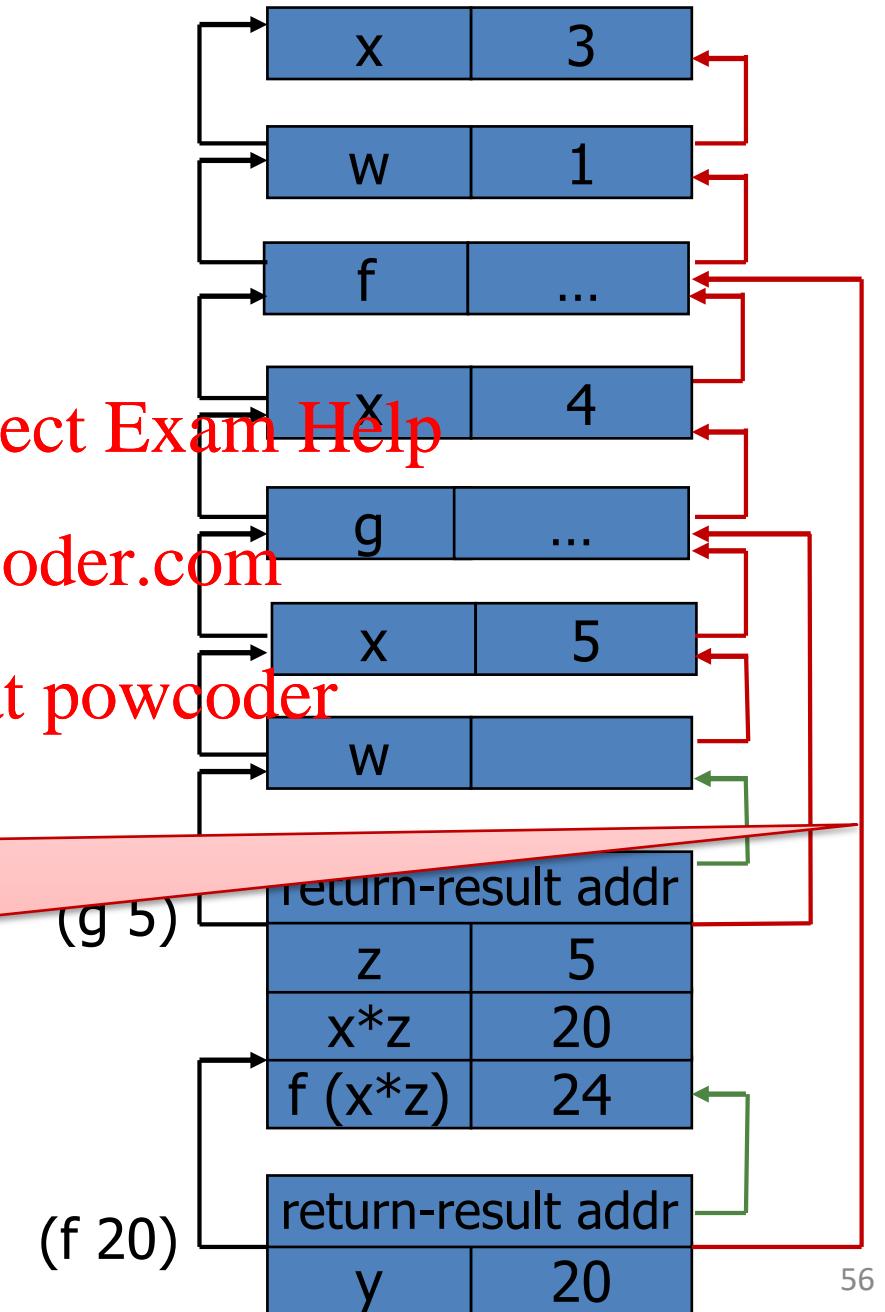
```

let x = 3 in
let w = 1 in
let f y = w+x+y in
let x = 4 in
let g z = f (x*z) in
let x = 5 in
let w = https://powcoder.com

```

Access link
Control
Return
green
Note that
links point
and
return-result links point to a
single location in a block.

Add WeChat powcoder
Follow access links
to find the values
of w and x used in
expression $w+x+y$.
Must follow 3 links
to get to x.



Tail Recursion (First-Order Case)

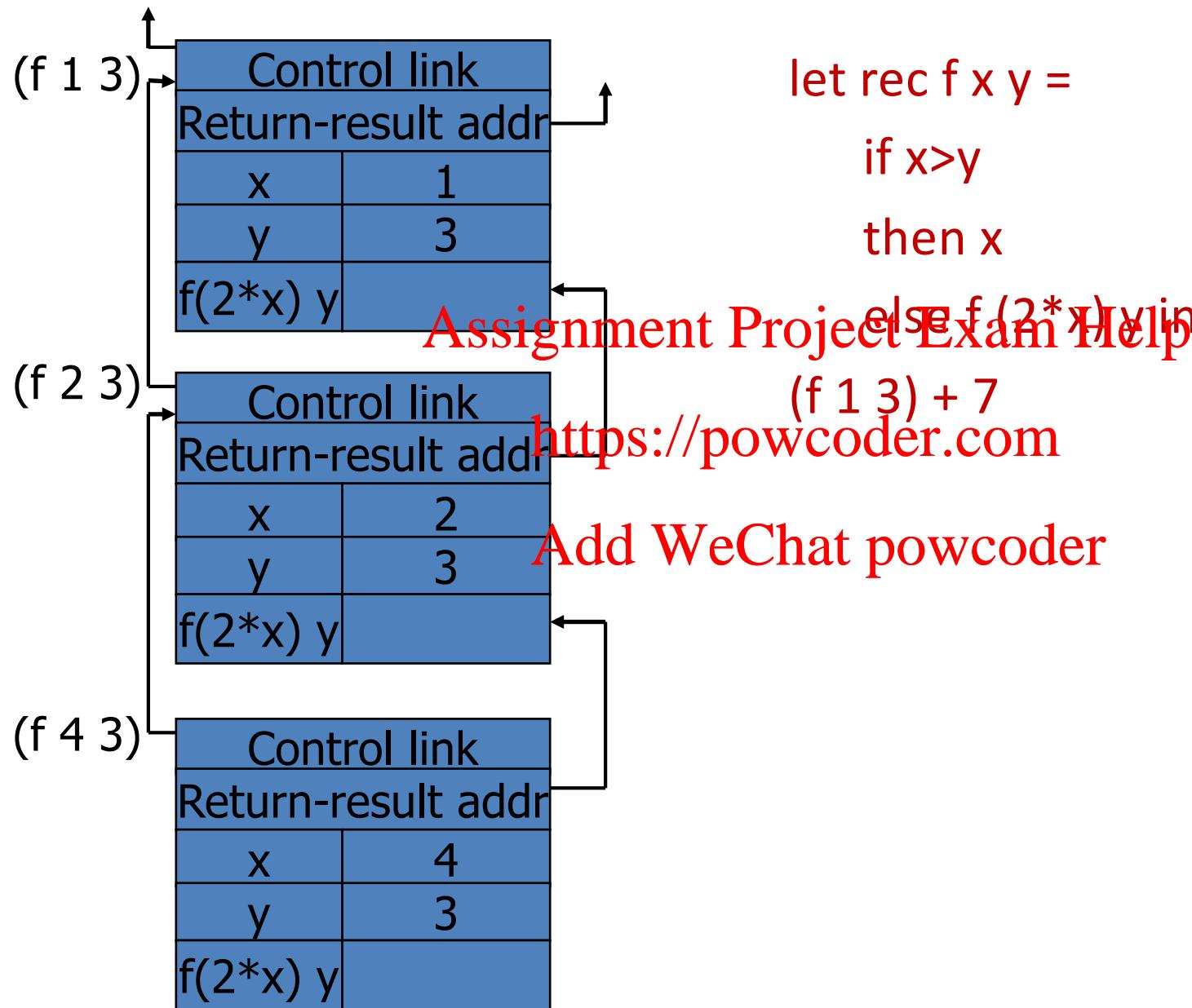
- Function g makes a *tail call* to function f if
 - Return value of function f is return value of g
- Example

tail call not a tail call
~~Assignment Project Exam Help~~
let g x = if x>0 then f x else (f x)*2

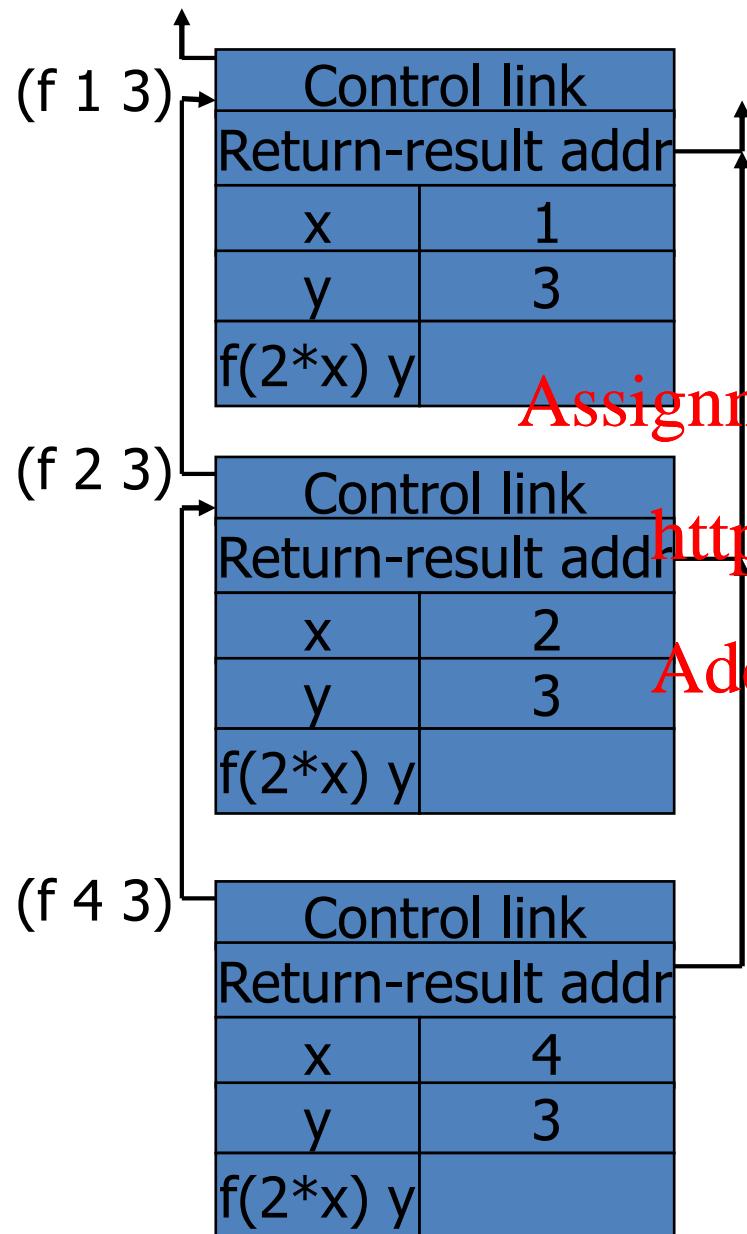
<https://powcoder.com>

- Optimization
 - Can pop activation record on a tail call
 - Especially useful for recursive tail call
 - next activation record has exactly same form
- A function f is *tail recursive* if all recursive calls in the body are tail calls to f.

Example: Calculate least power of 2 greater than y



Example: Calculate least power of 2 greater than y



let rec f x y =

if x>y

then x

else $f(2^*x) y$

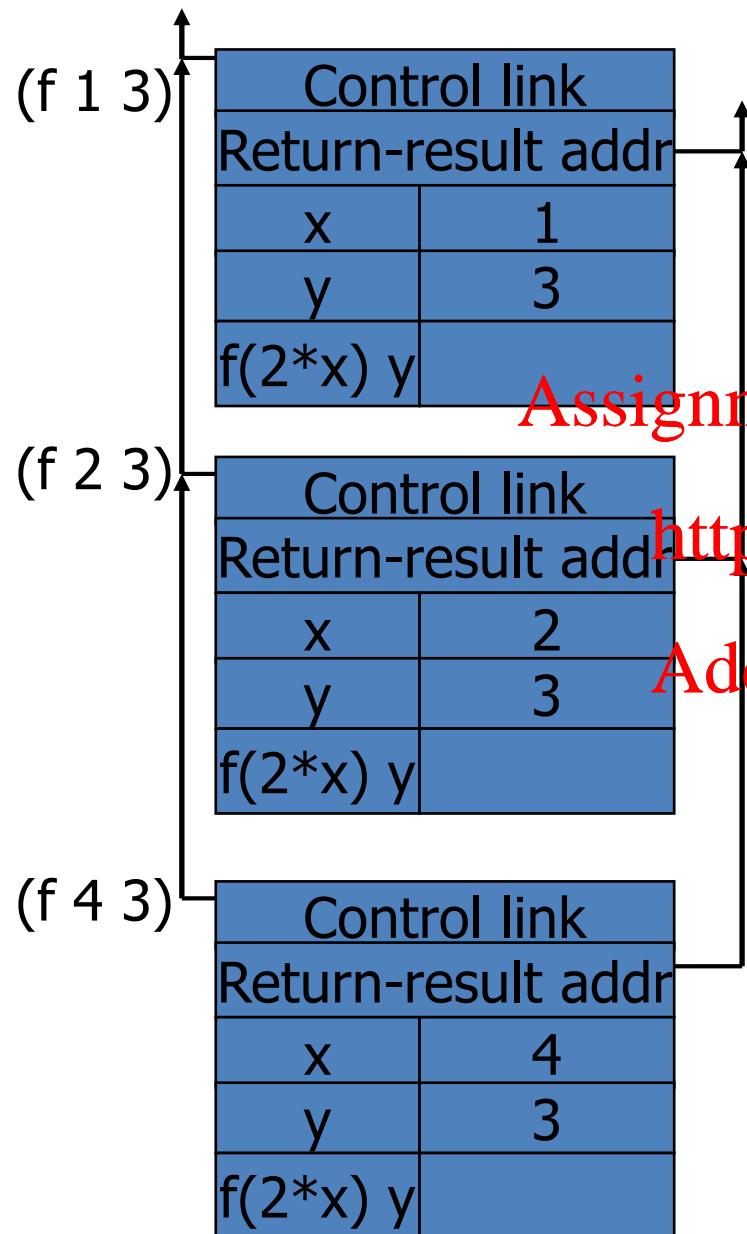
(f 1 3) + 7
<https://powcoder.com>

Add WeChat Optimization 1

- Set return-result address to the return-result address of caller.
- Can we do the same with the control link?

Yes, since the activation records are not needed after final result is determined and passed all the way up.

Example: Calculate least power of 2 greater than y



- **Optimization 2**

- When 3rd call finishes, we can pop 3rd and 2nd activation records together.

– In fact, we can pop all three.

- **Optimization 3**

- The activation record for the first call is no longer needed when the second call begins; pop the first before allocating the second.

- **Optimization 4**

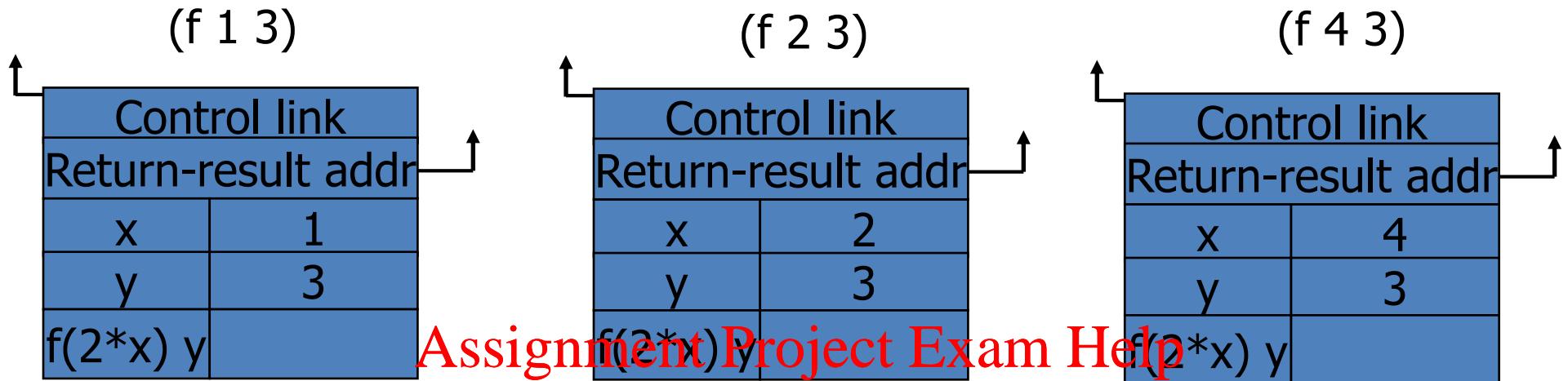
- Even better, don't deallocate and reallocate, just reuse.

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

Tail Recursion Elimination



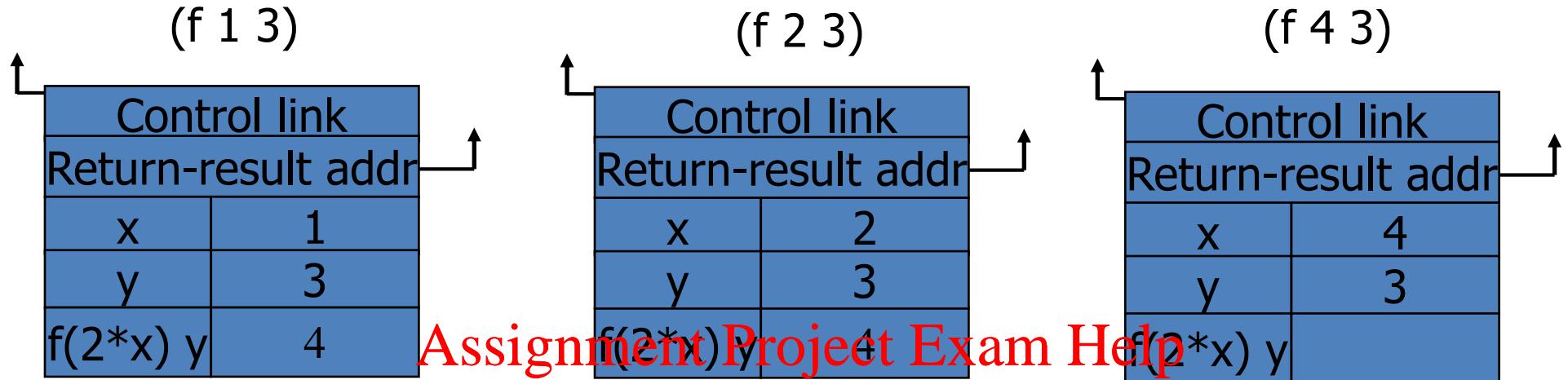
<https://powcoder.com>

let rec f x y =
 if x>y
 then x
 else f (2*x) y in
(f 1 3) + 7

Add WeChat [powcoder](#)

- Tail recursive function equivalent to iterative loop

Tail Recursion and Iteration



<https://powcoder.com>

```
let rec f x y =
  if x > y
  then x
  else f (2*x) y in
  (f 1 3) + 7
```

test
loop body
initial value

Add WeChat powcoder

```
fun g(y) = {
  x := 1;
  while not(x > y) do
    x := 2*x;
  return x;
};
```

Higher-Order Functions

- A language has first-class functions if functions can be:
 - Declared within any scope
 - Passed as arguments to other functions
 - Returned as results of functions
- Need to maintain environment of function
- Simpler case
 - Function passed as argument
 - Need pointer to activation record “higher up” in stack
- More complicated second case
 - Function returned as result of function call
 - Need to keep activation record of returning function

Pass Function as Argument

```
let x = 4 in { int x = 4;
let f(y) = x*y in { int f(int y) {return x*y;};
let g(h) = { int g(int→int h) {
(let x = 7 int x=7;
in      Assignment Project Exam Helpn h(3) + x;
h(3) + x) in https://powcoder.com } g(f);
g(f)
```

Add WeChat powcoder

There are two declarations of **x**.

Which one is used for each occurrence of **x**?

Pass Function as Argument

```
let x = 4 in
  let f(y) = x*y in
    let g(h) =
      (let x = 7
       in h(3) + x) in
        g(f)
```

{ int x = 4;
{ int f(int y) {return x*y;} }
int h) {
+ x;

The body of f contains
the global variable x.
This is the environment
of f.

Assignment Project Exam Help
<https://powcoder.com>
Add WeChat powcoder

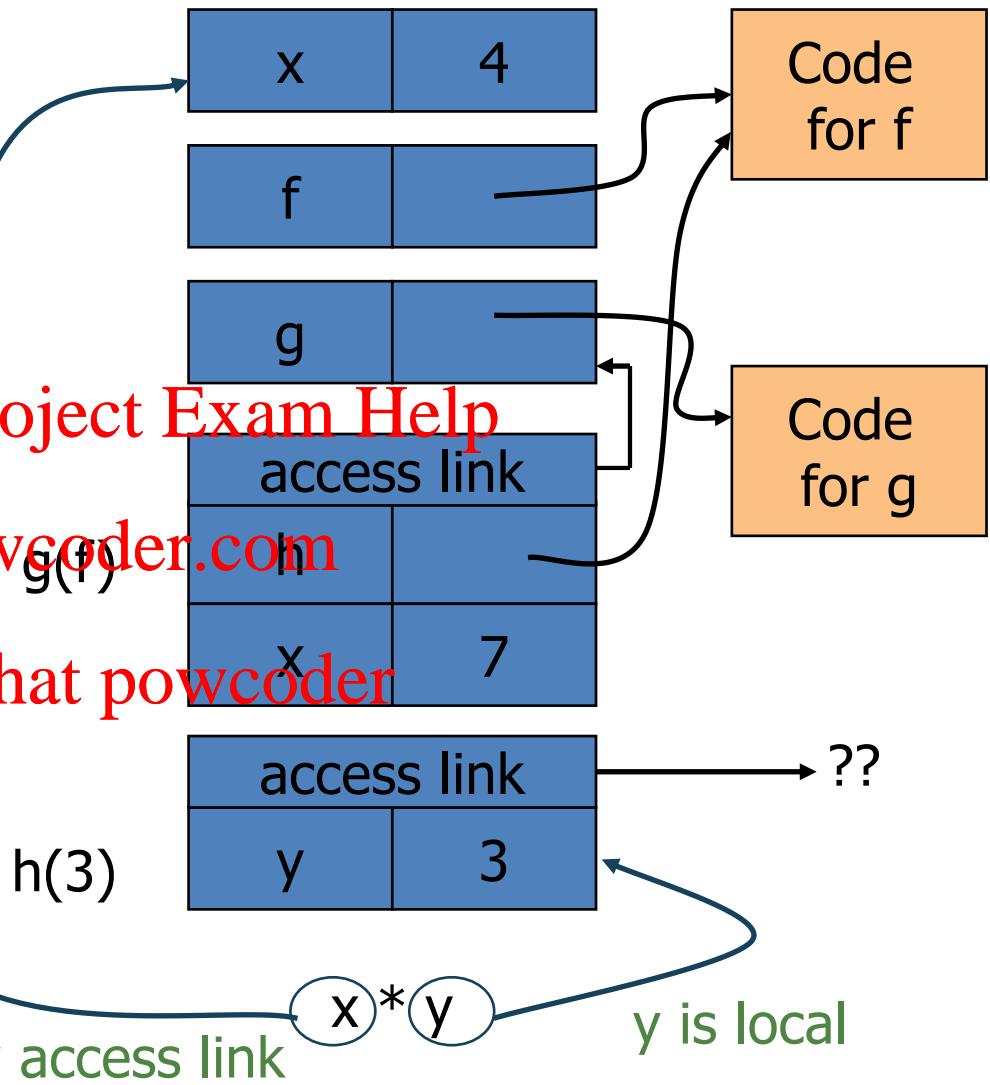
When f is called inside
g (because f is the
value of parameter h),
the value of x must be
retrieved from the
activation record of the
outer block (static
scope).

There are two dec...
ns of
Which one is used for

Static Scope for Function Argument

```
let x = 4 in  
let f(y) = x*y in  
let g(h) =  
(let x = 7  
in  
h(3) + x) in  
g(f)
```

x is global; follow access link

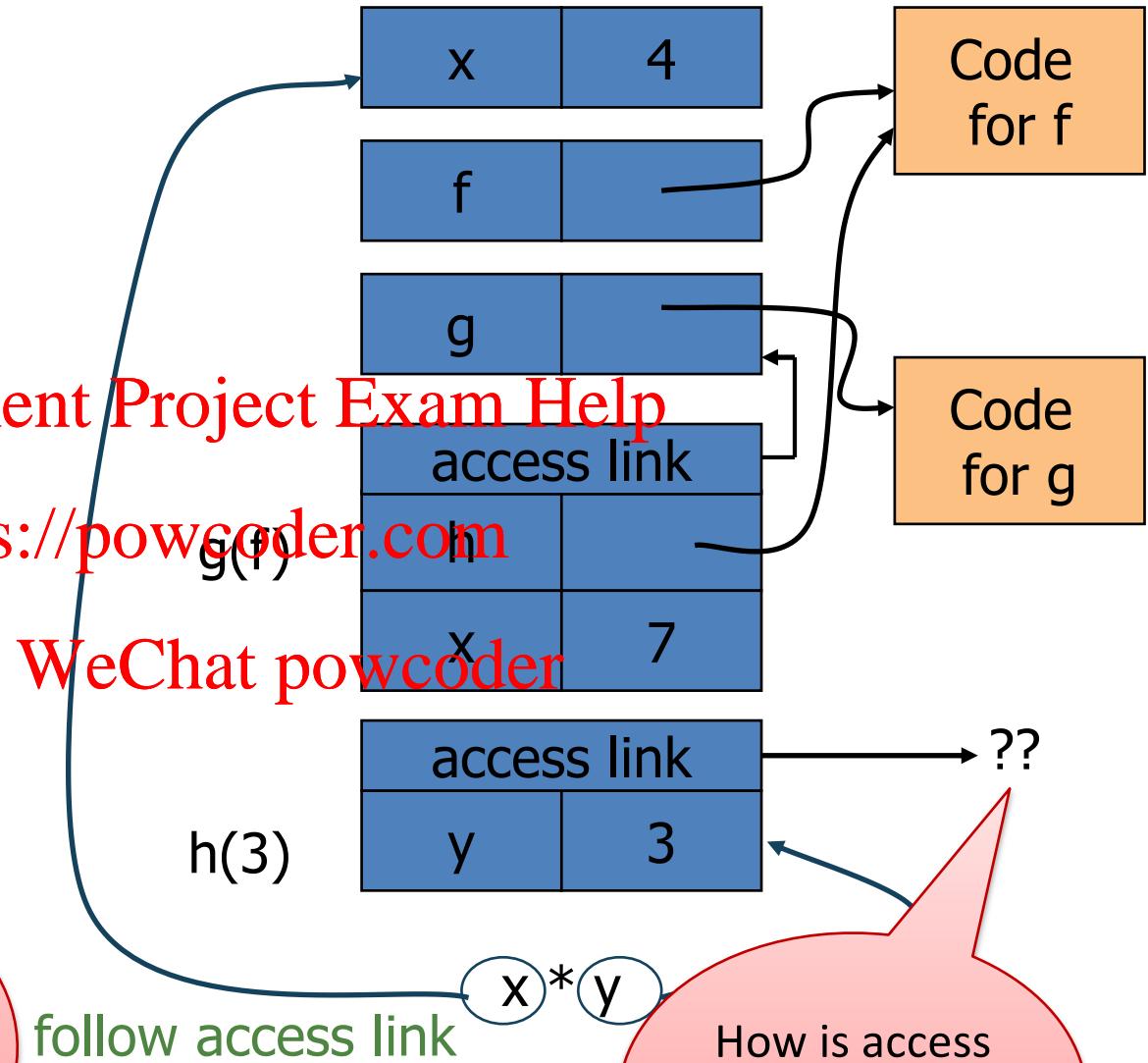


Static Scope for Function Argument

```
let x = 4 in  
let f(y) = x*y in  
let g(h) =  
(let x = 7  
in  
h(3) + x) in
```

g()

g can call different functions depending on value of h; can't tell from program text what values h will get



How is access link for h(3) set?

Static Scope for Function Argument

```
{ int x = 4;  
{ int f(int y) {return  
    x*y;}  
{ int g(int→int h) {  
    int x=7;  
    return h(3) + x;  
}  
g(f);  
} } }
```

Assignment Project Exam Help

<https://powcoder.com>

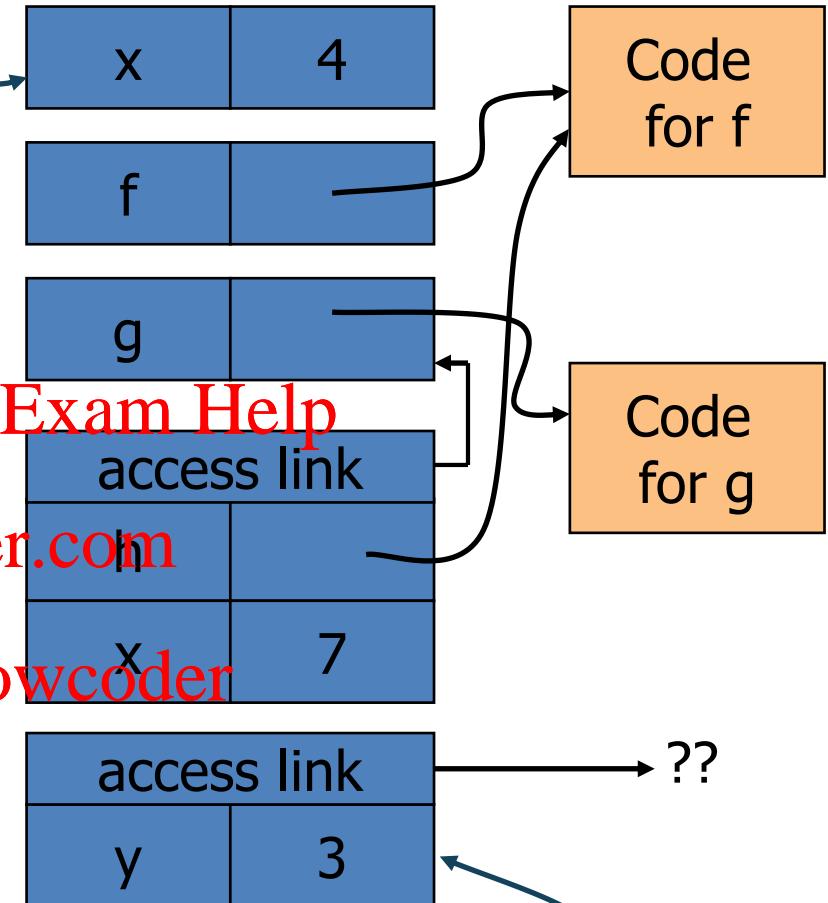
Add WeChat powcoder

x is global; follow access link

x

y

y is local



Closures

- With first-class functions and static scope, need a *closure* which is a pair:
 - Pointer to function code
 - Pointer to activation record (the environment of the function)
- Function value (when passed as a parameter) is a closure
- When a function represented by a closure is called,
 - Allocate activation record for call (as always)
 - Set the access link in the activation record using the environment pointer from the closure
- C and C++ do not support closures (because of implementation costs, though objects are like closures because they combine data with code for functions).

Function Argument and Closures

Run-time stack with access links

→ let x = 4 in
let f(y) = x*y in
let g(h) =
(let x = 7
in
h(3) + x) in
g(f)



Code
for f

Assignment Project Exam Help

<https://powcoder.com>

Code
for g

Add WeChat powcoder

Function Argument and Closures

Run-time stack with access links

```
let x = 4 in  
→ let f(y) = x*y in  
let g(h) =  
(let x = 7  
in  
h(3) + x) in  
g(f)
```

Assignment Project Exam Help



<https://powcoder.com>

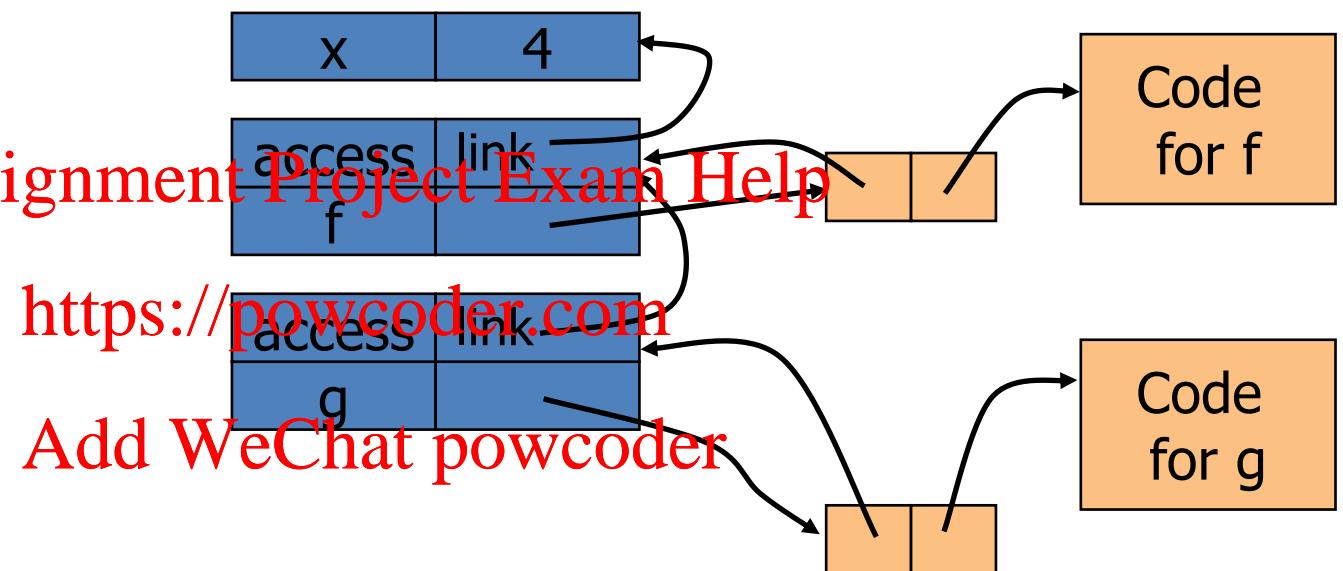
Add WeChat powcoder

Code
for g

Function Argument and Closures

```
let x = 4 in  
  let f(y) = x*y in  
    → let g(h) =  
      (let x = 7  
       in  
        h(3) + x) in  
      g(f)
```

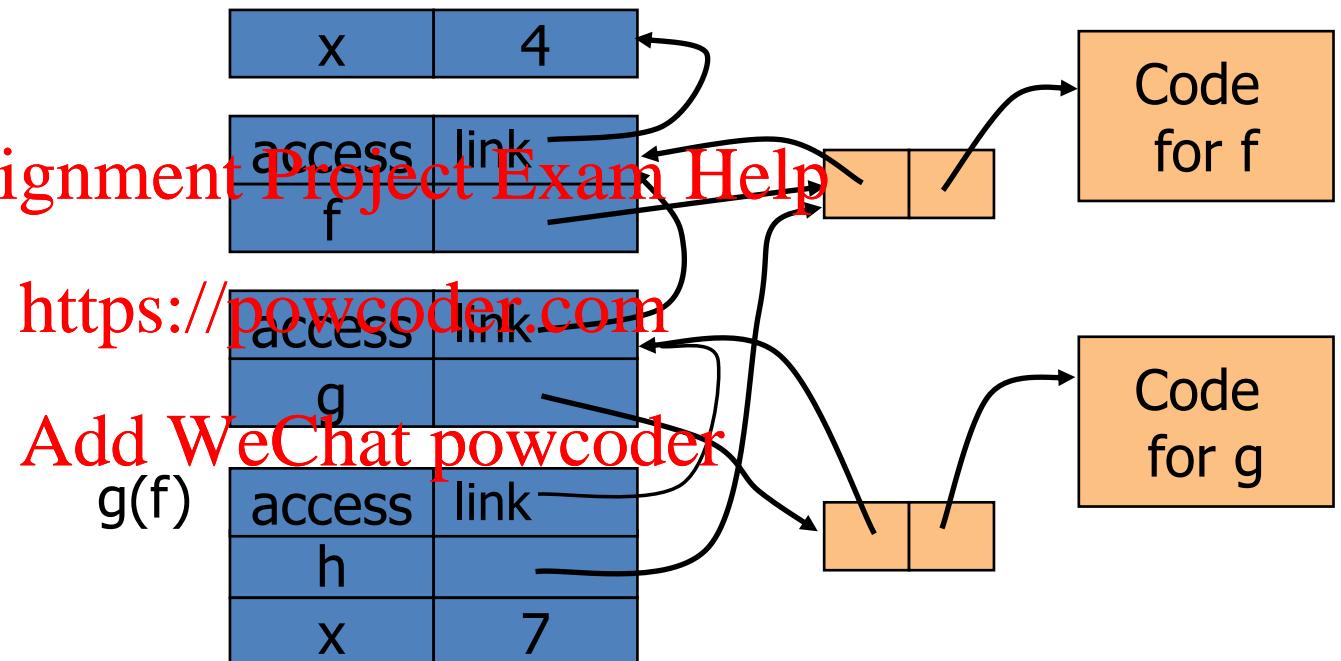
Run-time stack with access links



Function Argument and Closures

```
let x = 4 in  
  let f(y) = x*y in  
    let g(h) =  
      (let x = 7  
       in  
        h(3) + x) in  
      g(f)
```

Run-time stack with access links

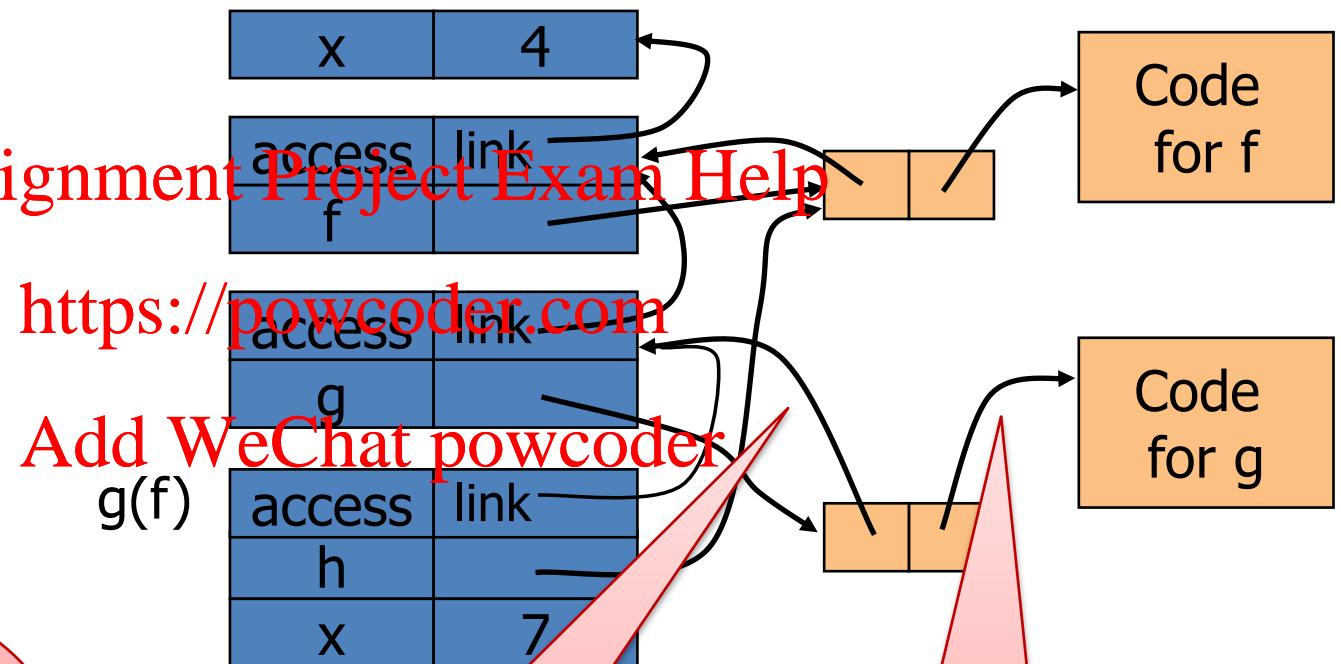


Function Argument and Closures

```
let x = 4 in  
  let f(y) = x*y in  
    let g(h) =  
      (let x = 7  
       in  
        h(3+x) in  
      g(f)
```

g(f) called from block where g is local.

Run-time stack with access links

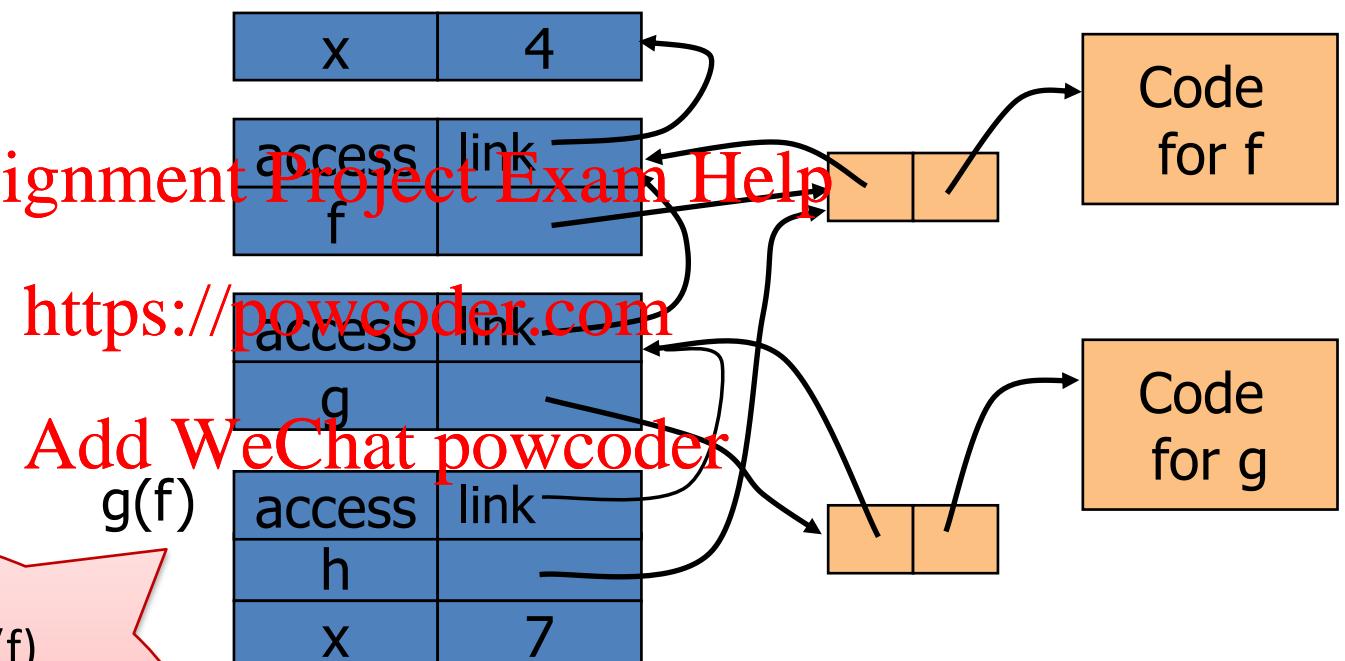


Function Argument and Closures

```
let x = 4 in  
  let f(y) = x*y in  
    let g(h) =  
      (let x = 7  
       in  
        h(3) + x) in  
      g(f)
```

f is global in the call g(f)
so is accessed via an
access link; f is passed as
actual parameter for
formal parameter h

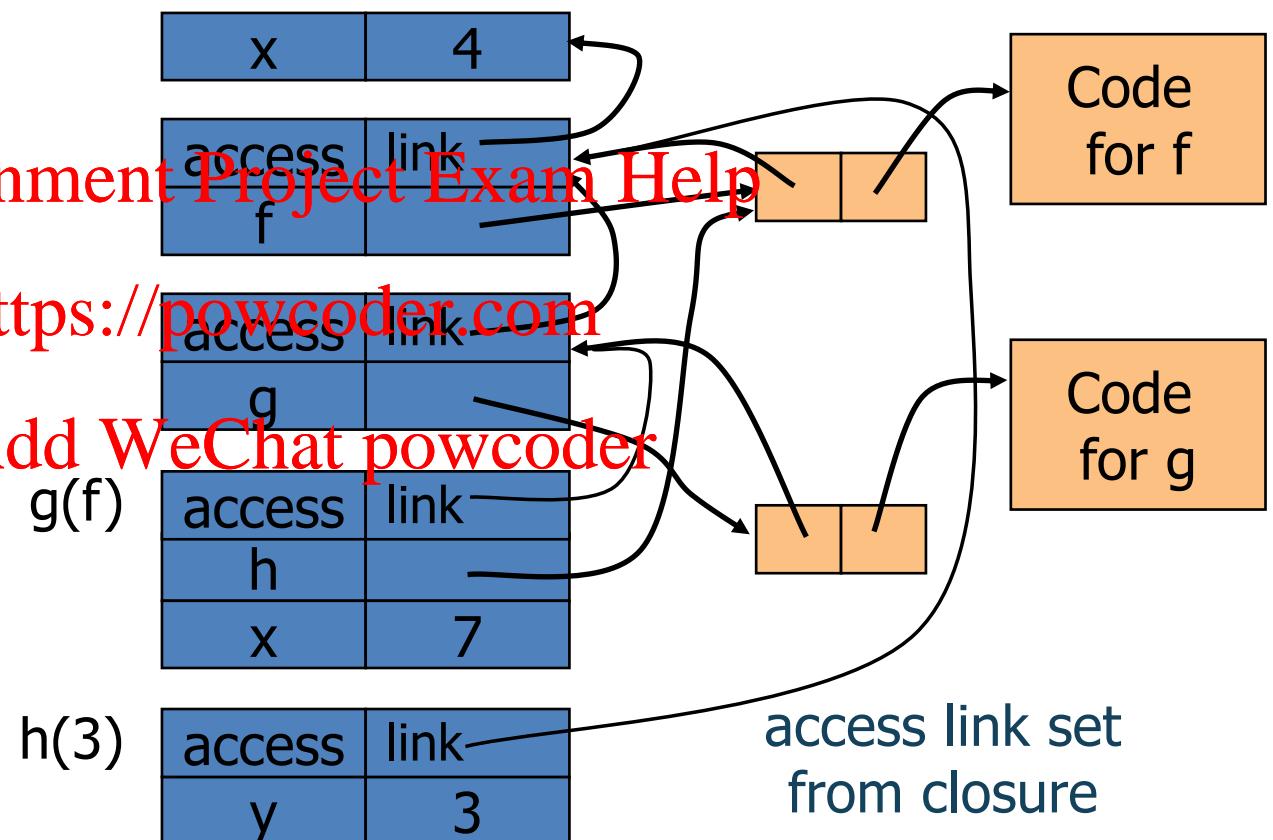
Run-time stack with access links



Function Argument and Closures

```
let x = 4 in  
  let f(y) = x*y in  
    let g(h) =  
      (let x = 7  
       in  
        h(3) + x) in  
      g(f)
```

Run-time stack with access links



Function Argument and Closures

Run-time stack with access links

To execute $h(3)$, follow pointer to closure for h (which provides code and access link from declaration of f .)

h
→ $h(3) + x$ in
 $g(f)$

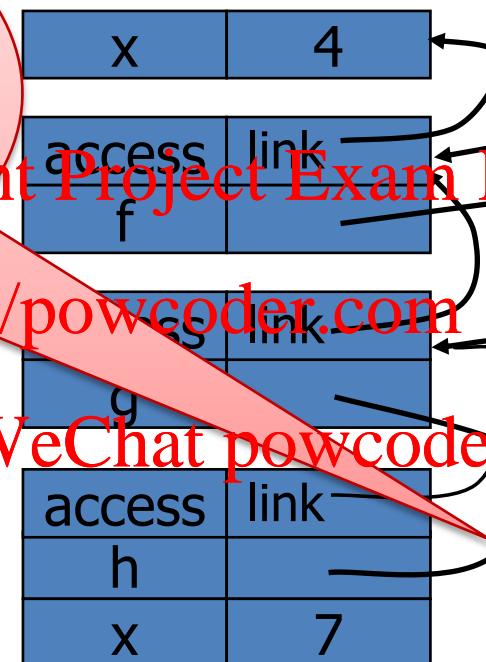
Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder

$h(3)$

access	link
y	3

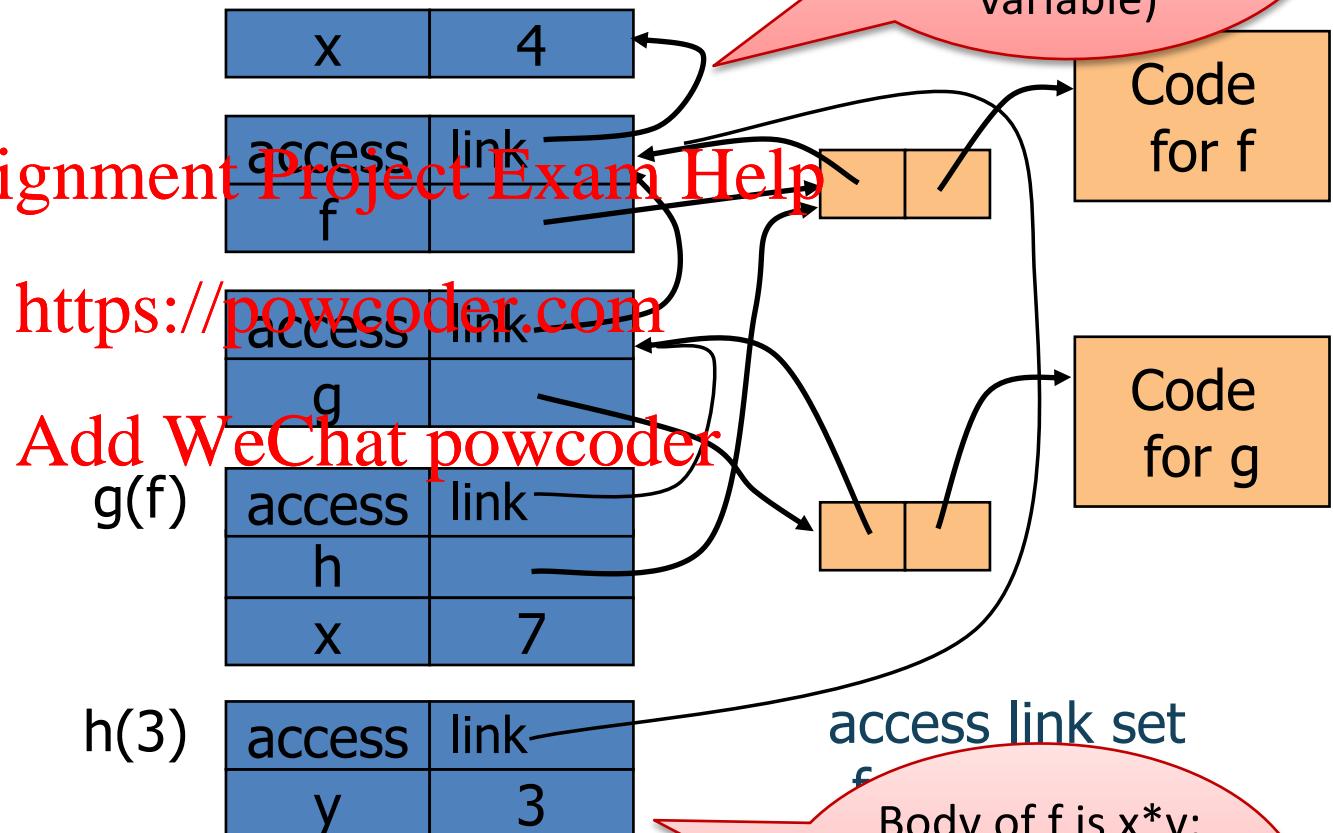


access link set
from closure

Function Argument and Closures

```
let x = 4 in  
  let f(y) = x*y in  
    let g(h) =  
      (let x = 7  
       in  
        h(3) + x) in  
      g(f)
```

Run-time stack with



Summary: Function Arguments

- Use closure to maintain a pointer to the static environment of a function body
- When called, set access link from closure
- All access links point “up” in stack
 - May jump ~~Assignment Project Exam Help~~ activation records to find global variables
 - Still deallocate activation records using stack order
<https://powcoder.com>

Add WeChat powcoder

Return Function as Result

- Language Feature
 - Functions that return “new” functions
 - Need to maintain environment of function
- Example

let compose f g = (fun x -> g(f x))

Assignment Project Exam Help

- Function “Created” Dynamically
 - expression with free variables
values are determined at run time
 - function value is closure = ⟨env, code⟩
 - The code pointer of this closure points to compiled code for “get function parameter x and then compute g(f(x))”.

Example: Return a Function with Private State

```
let mk_counter (init : int) =  
  (let count = ref init in
```

```
    let counter (inc : int) =  
      (count := !count + inc; !count)
```

```
    in  
      counter)
```

```
in
```

```
let c = mk_counter(1) in  
c(2) + c(2)
```

- The function `counter` is returned; the code for `mk_counter` builds it.
- Note that there is a parameter `inc` and a global variable `count` in the returned function.

Example: Return a Function with Private State

```
let mk_counter (init : int) =  
  (let count = ref init in
```

```
    let counter (inc : int) =  
      (count := !count + inc; !count)
```

```
    in  
      counter)           https://powcoder.com
```

```
in  
let c = mk_counter(1) in  
c(2) + c(2)
```

Add WeChat powcoder
Function to “make counter”
returns a closure

- How is correct value of **count** determined in **c(2)** ?

Function Results and Closures

```
let mk_counter (init : int) =  
  (let count = ref init in  
    let counter (inc : int) = (count := !count + inc; !count)  
    in counter)  
  in  
let c = mk_counter(1) in  
  c(2) + c(2)
```

Code for
mk_counter

Assignment Project Exam Help
<https://powcoder.com>

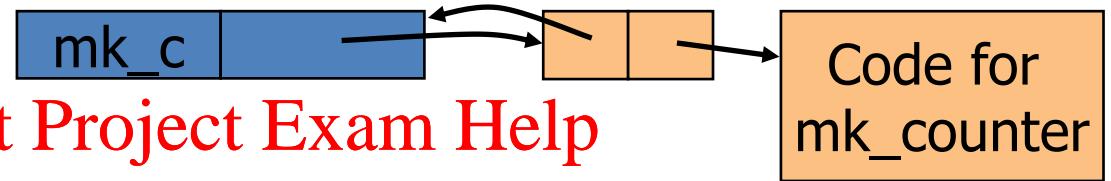
Add WeChat powcoder

Code for
counter

Function Results and Closures

```
let mk_counter (init : int) =  
  (let count = ref init in  
    let counter (inc : int) = (count := !count + inc; !count)  
    in counter)  
  in
```

let c = mk_counter(1)
c(2) + c(2)



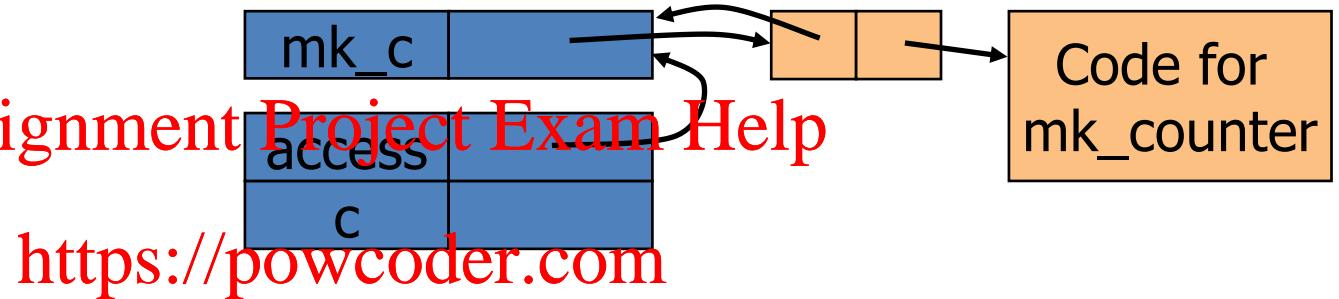
[Assignment Project Exam Help](https://powcoder.com)
<https://powcoder.com>

Add WeChat powcoder

Code for
counter

Function Results and Closures

```
let mk_counter (init : int) =  
  (let count = ref init in  
    let counter (inc : int) = (count := !count + inc; !count)  
    in counter)  
  in  
let c = mk_counter(1) in  
c(2) + c(2)
```

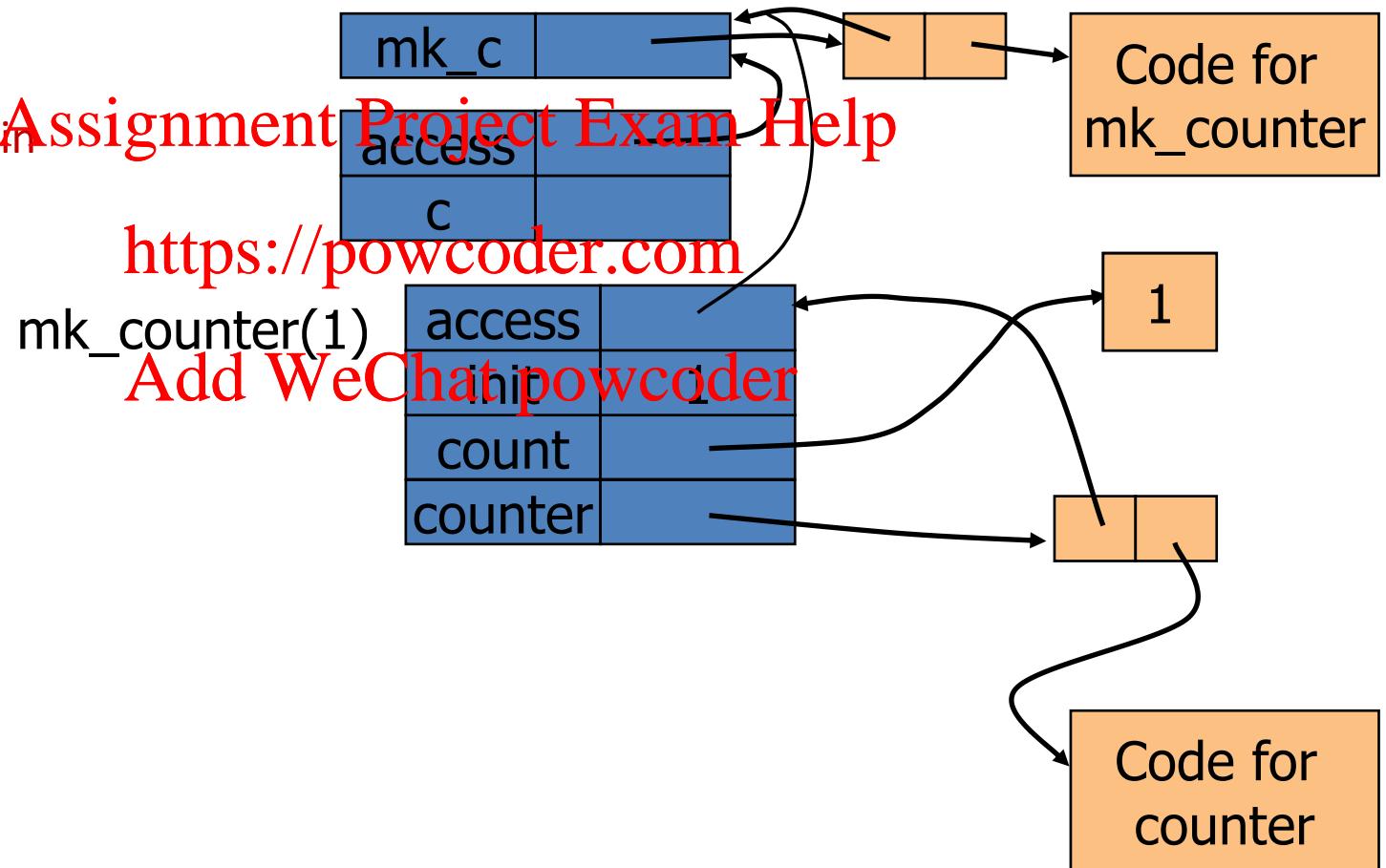


Add WeChat powcoder

Code for
counter

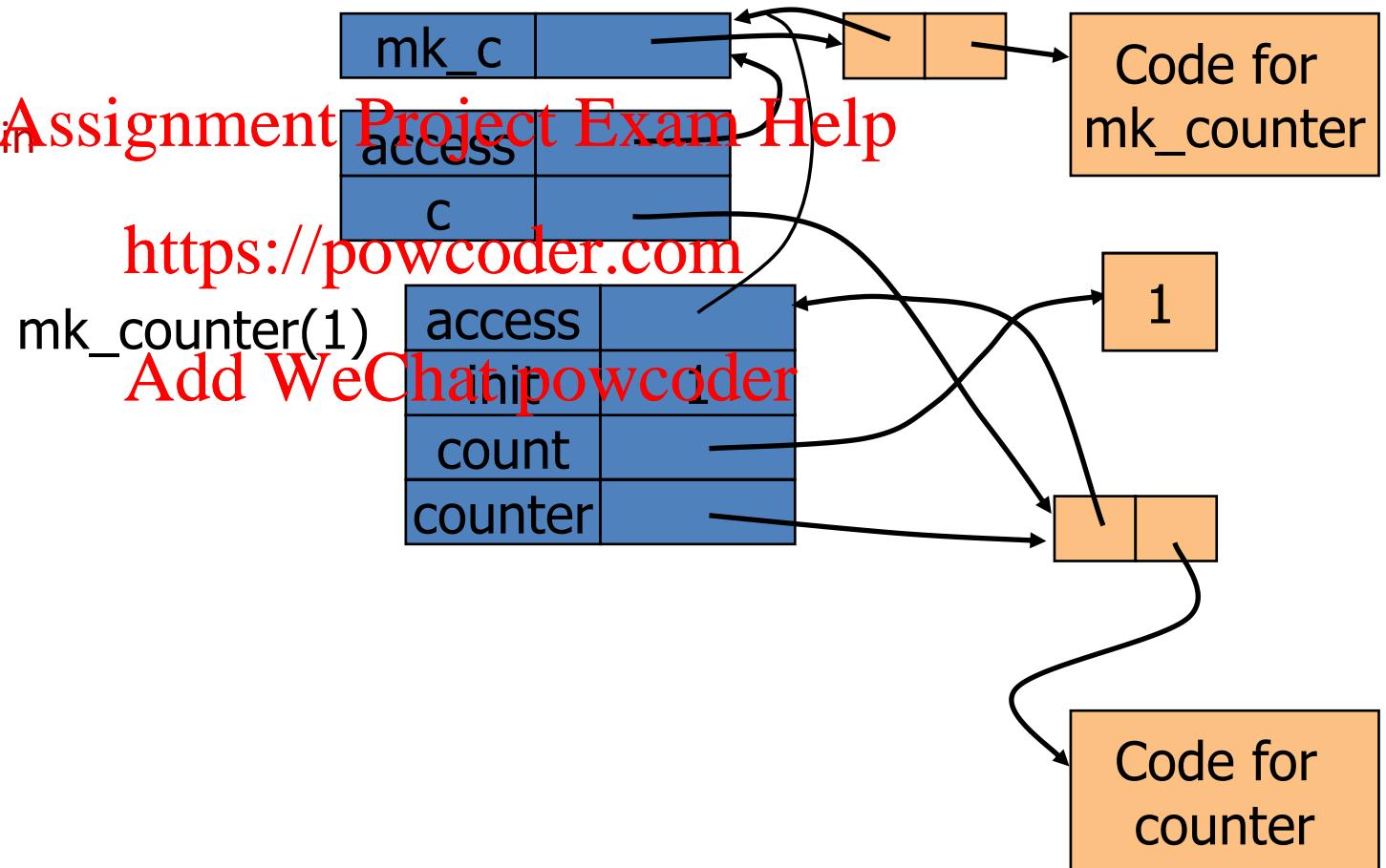
Function Results and Closures

```
let mk_counter (init : int) =  
  (let count = ref init in  
    let counter (inc : int) = (count := !count + inc; !count)  
    in counter)  
  in  
let c = mk_counter(1) in  
c(2) + c(2)
```



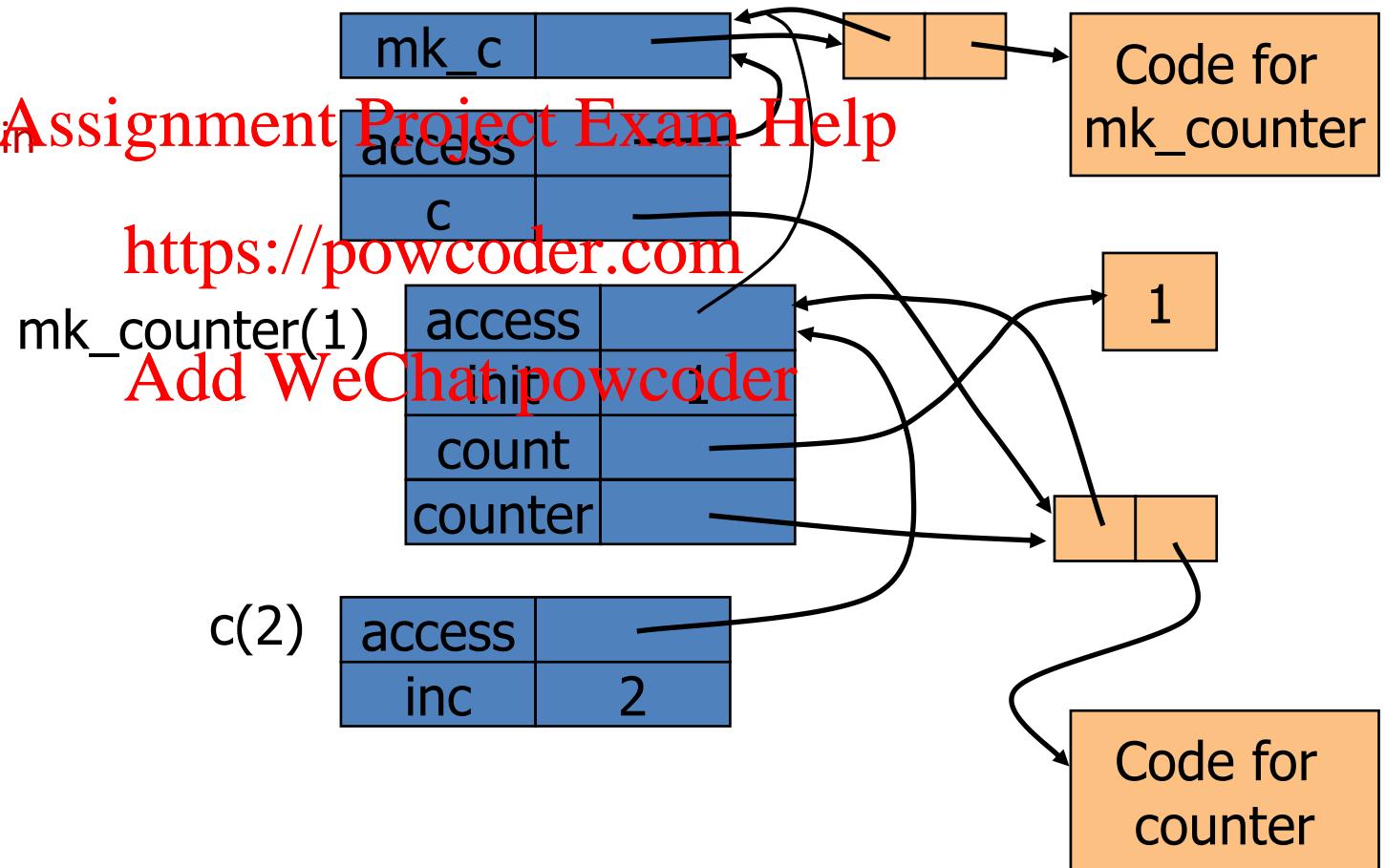
Function Results and Closures

```
let mk_counter (init : int) =  
  (let count = ref init in  
    let counter (inc : int) = (count := !count + inc; !count)  
    in counter)  
  in  
let c = mk_counter(1) in  
c(2) + c(2)
```



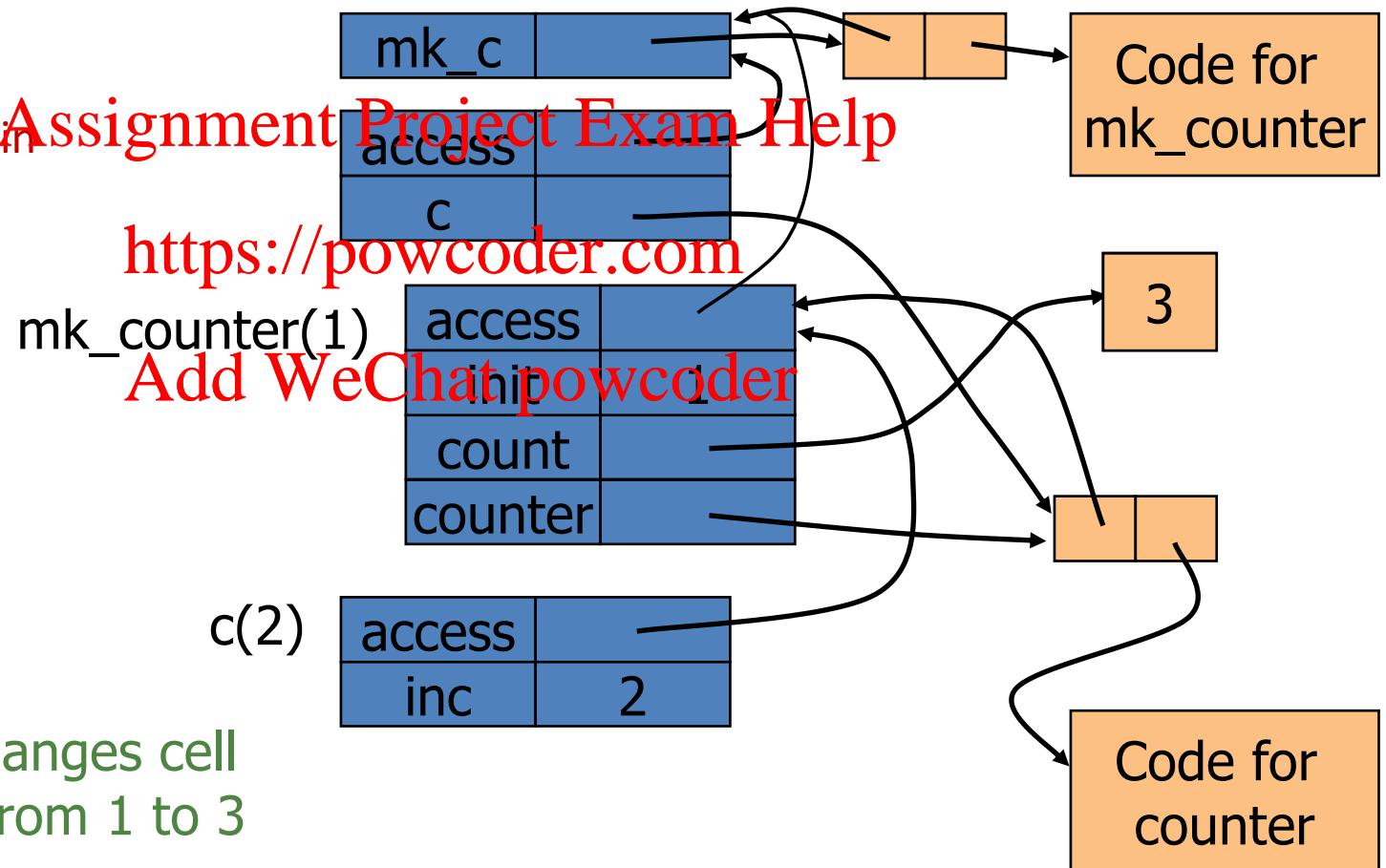
Function Results and Closures

```
let mk_counter (init : int) =  
  (let count = ref init in  
    let counter (inc : int) = (count := !count + inc; !count)  
    in counter)  
  in  
let c = mk_counter(1) in  
c(2) + c(2)
```



Function Results and Closures

```
let mk_counter (init : int) =  
  (let count = ref init in  
    let counter (inc : int) = (count := !count + inc; !count)  
    in counter)  
  in  
let c = mk_counter(1) in  
c(2) + c(2)
```



Summary of Example

- Closures are used to set the access pointer when a function returned from a nested scope is called.
- When functions are returned from nested scopes, activation record cannot be popped off the stack.
 - Stack (last-in-first-out) order fails!
- Possible alternative “stack” implementation
 - Forget about explicit deallocation
 - Put activation records on heap
 - Invoke garbage collector as needed
 - May seem inefficient but not necessarily
 - May only need to search reachable data

Summary of Scope Issues

- Block-structured languages use stack of activation records
 - Activation records contain parameters, local variables, ...
 - Also pointers to enclosing scope
- Several different parameter passing mechanisms
- Tail calls may be optimized
- Function parameters/results require closures
 - Closure environment pointer used on function call
 - Stack deallocation may fail if function returned from call
 - Closures *not* needed if functions not in nested blocks