Concepts of Programming Languages

Lecture 14 - Functions

Spring 2014

Patrick Donnelly

Montana State University

MONTANA STATE UNIVERSITY

Numerical Intelligent Systems Laboratory

It is better to have 100 functions operate on one data structure than 10 functions on 10 data structures.

Alan Perlis
Epigrams on Programming, 1982

Lecture Objectives

- Review subroutine terminology
- Understand the 5 Passing mechanisms:
  - `pass by value`
  - `pass by result`
  - `pass by value-result`
  - `pass by reference`
  - `pass by name`
- Practice passing with an example walkthrough
- Activation records
- Recursive functions

Procedures

Definition

Procedures are collection of statements that define parameterized computations.

Procedures are non-value-returning functions:

- "procedures" in Ada,
- "subroutines" in Fortran,
- "void functions/methods" in C/C++/Java

Procedures are often called from a separate statement:

```
strcpy(s1, s2);
```

Definition

Functions structurally resemble procedures but are semantically modeled on mathematical functions.

Value-returning functions:

- known as "non-void functions/methods' 'in C/C++/Java.
- called from within an expression.
- they are expected to produce no side effects, but in practice, program functions have side effects.

```
x = (b * b - sqrt(4 * a * c)) / 2 * a
```

Terminology Review

```
1  int x := 0;
2
3  void foo(int y){
4      x := x + 1;
5      y := y + 4;
6      print(x);
7  }
8
9  foo(x);
10 print(x);
```

Terminology Review

```
1  int x := 0;
2
3  void foo(int y){
4      x := x + 1;
5      y := y + 4;
6      print(x);
7  }
8
9  foo(x);
10 print(x);
```

What is line 3?

Terminology Review

```
1  int x := 0;
2
3  void foo(int y){
4      x := x + 1;
5      y := y + 4;
6      print(x);
7  }
8
9  foo(x);
10 print(x);
```

What is line 3?

function declaration

Terminology Review

```
1  int x := 0;
2
3  void foo(int y){
4      x := x + 1;
5      y := y + 4;
6      print(x);
7  }
8
9  foo(x);
10 print(x);
```

What are lines 4-7?

Terminology Review

```
1  int x := 0;
2
3  void foo(int y){
4      x := x + 1;
5      y := y + 4;
6      print(x);
7  }
8
9  foo(x);
10 print(x);
```

What are lines 4-7?

function body

Terminology Review

```
1  int x := 0;
2
3  void foo(int y){
4      x := x + 1;
5      y := y + 4;
6      print(x);
7  }
8
9  foo(x);
10 print(x);
```

What is line 9?

Terminology Review

```
1  int x := 0;
2
3  void foo(int y){
4      x := x + 1;
5      y := y + 4;
6      print(x);
7  }
8
9  foo(x);
10 print(x);
```

What is line 9?

function call

Arguments and Parameters

Definition

An ***argument*** is an expression that appears in a function call.

Definition

A ***parameter*** is an identifier that appears in a function declaration.

Arguments and Parameters

```
1  int x := 0;
2
3  void foo(int y){
4      x := x + 1;
5      y := y + 4;
6      print(x);
7  }
8
9  foo(x);
10 print(x);
```

The function declaration `foo` has parameter `y`.

The function calls `foo` and `print` have argument `x`.

Parameter-Argument Matching

Usually by number and by position.

i.e., any call to `A` must have two arguments, and they must match the corresponding parameters' types.

Parameter-Argument Matching

Usually by number and by position.

i.e., any call to `A` must have two arguments, and they must match the corresponding parameters' types.

Exception – Perl

Parameters are not declared in a function header. Instead, parameters are available in an array `@_`, and are accessed using a subscript on this array.

Exception – Ada

Arguments and parameters can be linked by name.

- e.g., the call `A(y=>b, x=>a)` is the same as `A(a, b)`

Motivating Example

```c
#include <stdio.h>

void swap(int i, int j) {
  int t = i;
  i = j;
  j = t;
}

void main() {
  int a = 1, b = 2;
  printf("a: %d, b: %d\n",
         a, b);
  swap(a, b);
  printf("a: %d, b: %d\n",
         a, b);
}
```

Which output should this code snippet print?

Motivating Example

```c
#include <stdio.h>

void swap(int i, int j) {
  int t = i;
  i = j;
  j = t;
}

void main() {
  int a = 1, b = 2;
  printf("a: %d, b: %d\n",
         a, b);
  swap(a, b);
  printf("a: %d, b: %d\n",
         a, b);
}
```

Which output should this code snippet print?

```
a:  1, b:  2
a:  1, b:  2
```

Motivating Example

```c
#include <stdio.h>

void swap(int i, int j) {
  int t = i;
  i = j;
  j = t;
}

void main() {
  int a = 1, b = 2;
  printf("a: %d, b: %d\n",
         a,b);
  swap(a,b);
  printf("a: %d, b: %d\n",
         a,b);
}
```

Which output should this code snippet print?

```
a:  1, b:  2
a:  1, b:  2
```

or

```
a:  1, b:  2
a:  2, b:  1
```

Models of Parameter Passing

call by value: copy going into the procedure

Models of Parameter Passing

call by value: copy going into the procedure

call by result: copy going out of the procedure

Models of Parameter Passing

call by value: copy going into the procedure

call by result: copy going out of the procedure

call by value result: copy going in, and again going out

Models of Parameter Passing

call by value: copy going into the procedure

call by result: copy going out of the procedure

call by value result: copy going in, and again going out

call by reference: pass a pointer to the actual parameter, and indirect through the pointer

Models of Parameter Passing

call by value: copy going into the procedure

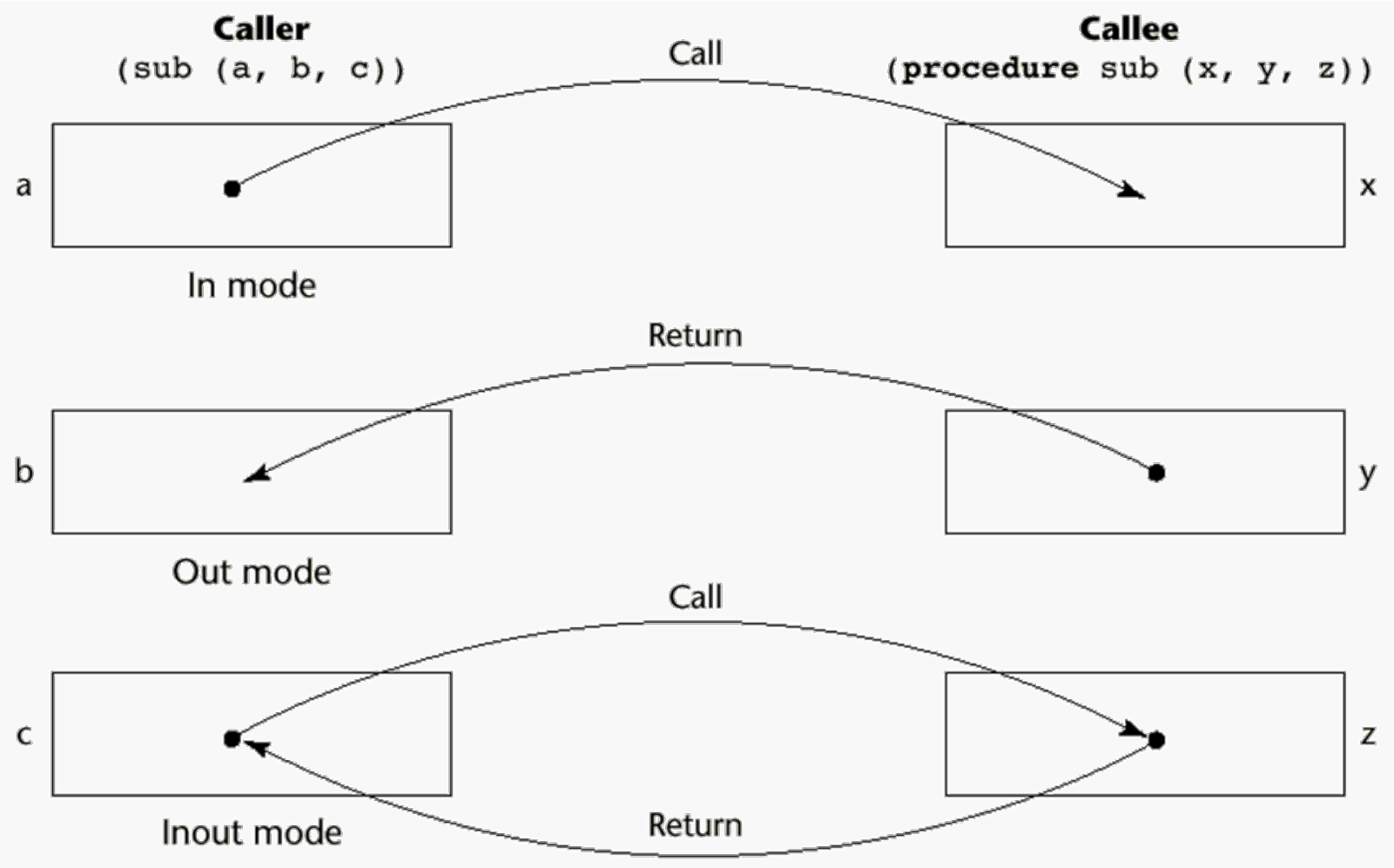
call by result: copy going out of the procedure

call by value result: copy going in, and again going out

call by reference: pass a pointer to the actual parameter, and indirect through the pointer

call by name: re-evaluate the actual parameter on every use

Models of Parameter Passing



Pass by Value

Definition

Passing an argument ***by value*** means that the value of the argument is computed at the time of the call and copied to the corresponding parameter.

Normally implemented by copying going into the procedure.

Mode: In Mode

Pass by Value Example

```
1  int x := 0;
2
3  void foo(int y){
4       x := x + 1;
5       y := y + 4;
6       print(x);
7  }
8
9  foo(x);
10 print(x);
```

»

Pass by Value Example

```
1  int x := 0;
2
3  void foo(int y){
4      x := x + 1;
5      y := y + 4;
6      print(x);
7  }
8
9  foo(x);
10 print(x);
```

y = 0

x = 1

y = 4

»

Pass by Value Example

```
1  int x := 0;
2
3  void foo(int y){
4      x := x + 1;
5      y := y + 4;
6      print(x);
7  }
8
9  foo(x);
10 print(x);
```

y = 0
x = 1
y = 4

» 1 1

In Java, all primitive parameters are passed are passed by value.

```
public void swap(int x, int y){
    int t = x;
    x = y;
    y = t;
}

public static void main(String [] args){
  int a = 10;
  int b = 20;
  System.out.println("a: " + a + ",  b: " + b);
  swap(a,b);
  System.out.println("a: " + a + ",  b: " + b);
}
```

After executing `main()`:

» `a:  10, b:  20`

» `a:  10, b:  20`

Pass by Value

Languages: C, Pascal, Ada, Scheme, Algol68, Java

Advantage: original values are not changed.

Disadvantages: additional storage and computation are required.

Pass by Result

Definition

An argument passed ***by result*** is implemented by copying the final value computed for the parameters out to the argument at the end of the life of the call.

Mode: Out Mode

Languages: Ada

Disadvantages: Require extra storage location and copy operation

Pass by Result

```
int x := 0;

void foo(int y){
     x := x + 1;
     y := y + 4;
     print(x);
}

foo(x);
print(x);
```

»

Pass by Result

```
int x := 0;

void foo(int y){
     x := x + 1;
     y := y + 4;
     print(x);
}

foo(x);
print(x);
```

y = ??

x = 1

y = 4

»

Pass by Result

```
int x := 0;

void foo(int y){
     x := x + 1;
     y := y + 4;
     print(x);
}

foo(x);
print(x);
```

y = ??

x = 1

y = 4

» 1 4

x changed after calling foo

Potential problems:

```
sub(p1, p1);
```

whichever formal parameter is copied back
will represent the current value of `p1`

```
sub(list[sub], sub);
```

Compute address of `list[sub]` at the beginning of the subprogram or end?

Pass by Value-Result

Definition

An argument passed ***by value-result*** is implemented by copying the argument's value into the parameter at the beginning of the call and then copying the computer result back to the corresponding argument at the end of the call.

Mode: In-out Mode

Value-result is often called copy-in-copy-out, a combination of pass by value and pass by result.

Pass by Value-Result

Languages: Fortran, sometimes Ada

Advantages and **Disadvantages**:

- All those of pass by result
- and all those of pass by value

Pass by Value-Result

```
int x := 0;

void foo(int y){
     x := x + 1;
     y := y + 4;
     print(x);
}

foo(x);
print(x);
```

»

Pass by Value-Result

```
int x := 0;

void foo(int y){
     x := x + 1;
     y := y + 4;
     print(x);
}

foo(x);
print(x);
```

y = 0
x = 1
y = 4

»

Pass by Value-Result

```
int x := 0;

void foo(int y){
     x := x + 1;
     y := y + 4;
     print(x);
}

foo(x);
print(x);
```

y = 0
x = 1
y = 4

» 1 4

x changed after calling foo

Pass by Reference

Definition

Passing an argument ***by reference*** or ***by address*** means that the memory address of the argument is copied to the corresponding parameter so the parameter becomes an indirect reference (pointer) to the actual argument.

Mode: In-out Mode

Also called pass-by-sharing

Pass via an access path: Instead of physically move a value, we move an access path to a value

Aliasing

Definition

Aliasing occurs when, within a function or procedure, the same memory location can be accessed using different names.

Aliasing

Definition

Aliasing occurs when, within a function or procedure, the same memory location can be accessed using different names.

Example

Examples of Aliasing:

- the same variable is both passed and globally referenced from the called function
- the same variable is passed for two different parameters.
- two or more references (i.e., pointers) to the same location

Pass by Reference

Languages: Fortran, Perl, Pascal var params, Java objects, permitted in C++, C#, Cobol

Advantage: Passing process is efficient

Disadvantages:

- Slower accesses to formal parameters
- Potentials for unwanted side effects (collisions)
- Unwanted aliases

Pass by Reference

```
int x := 0;

void foo(int y){
     x := x + 1;
     y := y + 4;
     print(x);
}

foo(x);
print(x);
```

»

Pass by Reference

```
int x := 0;

void foo(int y){
     x := x + 1;
     y := y + 4;
     print(x);
}

foo(x);
print(x);
```

x = y = 0

x = y = 1

x = y = 5

»

Pass by Reference

```
int x := 0;

void foo(int y){
    x := x + 1;
    y := y + 4;
    print(x);
}

foo(x);
print(x);
```

x = y = 0
x = y = 1
x = y = 5

» 5 5

Calling `foo` changed `x`

Passing in C

Swap Fails

```
#include <stdio.h>

void swap(int i, int j) {
    int t = i;
    i = j;
    j = t;
}

void main() {
    int a = 1, b = 2;
    printf("a: %d, b: %d\n",a,b);
    swap(a,b);
    printf("a: %d, b: %d\n",a,b);
}
```

```
a:  1, b:  2
a:  1, b:  2
```

Swap with Pointers

```
#include <stdio.h>

void swap(int *i, int *j) {
    int t = *i;
    *i = *j;
    *j = t;
}

void main() {
    int a = 1, b = 2;
    printf("a: %d b: %d\n",a,b);
    swap(&a, &b);
    printf("a: %d, b: %d\n",a,b);
}
```

```
a:  1, b:  2
a:  2, b:  1
```

Pass by Name

Definition

An argument passed ***by name*** behaves as though it is textually substituted for each occurrence of the parameter.

Formals are bound to an access method at the time of the call, but actual binding to a value or address takes place at the time of a reference or assignment. Expression is re-evaluated on each access.

Originated with Algol 60, but was dropped by Algol's successors – Pascal, Ada, Modula.

Pass by Name

Definition

Pass by name is an example of ***late binding***, since evaluation of the argument is delayed until its occurrence in the function body is actually executed.

Definition

Late binding is associated with ***lazy evaluation*** in functional languages

Pass by Value versus Pass by Name

```
int x := 0;

void bar(int y){
     print(y);
     x := x + 1;
     print(y);
}

bar(x+10);
```

»

»

Pass by Value versus Pass by Name

```
int x := 0;

void bar(int y){
     print(y);
     x := x + 1;
     print(y);
}

bar(x+10);
```

» Pass by Value

»

Pass by Value versus Pass by Name

```
int x := 0;

void bar(int y){
     print(y);
     x := x + 1;
     print(y);
}

bar(x+10);
```

» 10 10 Pass by Value

» Pass by Name

Pass by Value versus Pass by Name

```
int x := 0;

void bar(int y){
     print(y);
     x := x + 1;
     print(y);
}

bar(x+10);
```

» 10 10 — Pass by Value

» 10 11 — Pass by Name

Parameter Passing Mode Example

Consider this simple code snippet in Ada.

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Parameter Passing Mode Example

Consider this simple code snippet in Ada.

Pass by ...

- value
- result
- value-result
- reference

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Parameter Passing Mode Example: Value

Pass by Value

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Parameter Passing Mode Example: Value

Pass by Value

Which values are written?

Line 4:

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value:

- In mode
- from caller to callee

Parameter Passing Mode Example: Value

Pass by Value

Which values are written?

Line 4: 5
Line 6:

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value:

- In mode
- from caller to callee

Parameter Passing Mode Example: Value

Pass by Value

Which values are written?

Line 4: 5
Line 6: 6
Line 11:

Review

Pass by Value:

- In mode
- from caller to callee

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Parameter Passing Mode Example: Value

Pass by Value

Which values are written?

Line 4: 5
Line 6: 6
Line 11: 5

Review

Pass by Value:

- In mode
- from caller to callee

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Parameter Passing Mode Example: Result

Pass by Result

Which values are written?

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value/Result:

- Out Mode
- from callee to caller

Parameter Passing Mode Example: Result

Pass by Result

Which values are written?

Line 4:

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value/Result:

- Out Mode
- from callee to caller

Parameter Passing Mode Example: Result

Pass by Result

Which values are written?

Line 4: ??
Line 6:

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value/Result:

- Out Mode
- from callee to caller

Parameter Passing Mode Example: Result

Pass by Result

Which values are written?

Line 4: ??
Line 6: 6
Line 11:

Review

Pass by Value/Result:

- Out Mode
- from callee to caller

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Parameter Passing Mode Example: Result

Pass by Result

Which values are written?

Line 4: ??
Line 6: 6
Line 11: 1

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value/Result:

- Out Mode
- from callee to caller

Parameter Passing Mode Example: Value/Result

Pass by Value/Result

Which values are written?

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value/Result:

- In-out Mode
- from caller to callee
- from callee to caller

Parameter Passing Mode Example: Value/Result

Pass by Value/Result

Which values are written?

Line 4:

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value/Result:

- In-out Mode
- from caller to callee
- from callee to caller

Parameter Passing Mode Example: Value/Result

Pass by Value/Result

Which values are written?

Line 4: 5

Line 6:

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value/Result:

- In-out Mode
- from caller to callee
- from callee to caller

Parameter Passing Mode Example: Value/Result

Pass by Value/Result

Which values are written?

Line 4: 5
Line 6: 6
Line 11:

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value/Result:

- In-out Mode
- from caller to callee
- from callee to caller

Parameter Passing Mode Example: Value/Result

Pass by Value/Result

Which values are written?

Line 4: 5
Line 6: 6
Line 11: 1

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Value/Result:

- In-out Mode
- from caller to callee
- from callee to caller

Parameter Passing Mode Example: Reference

Pass by Reference

Which values are written?

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Reference:

- In-out Mode
- pass-by-sharing

Parameter Passing Mode Example: Reference

Pass by Reference

Which values are written?

Line 4:

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Reference:

- In-out Mode
- pass-by-sharing

Parameter Passing Mode Example: Reference

Pass by Reference

Which values are written?

Line 4: 5

Line 6:

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Reference:

- In-out Mode
- pass-by-sharing

Parameter Passing Mode Example: Reference

Pass by Reference

Which values are written?

Line 4: 5
Line 6: 2
Line 11:

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Reference:

- In-out Mode
- pass-by-sharing

Parameter Passing Mode Example: Reference

Pass by Reference

Which values are written?

Line 4: 5
Line 6: 2
Line 11: 1

```
1  var y:integer;
2  procedure A(x:integer);
3      begin
4          write(x);
5          x := 1;
6          write(y+x);
7      end;
8  begin
9      y := 5;
10     A(y);
11     write(y);
12 end;
```

Review

Pass by Reference:

- In-out Mode
- pass-by-sharing

Review

call by value: copy going into the procedure

call by result: copy going out of the procedure

call by value result: copy going in, and again going out

call by reference: pass a pointer to the actual parameter, and indirect through the pointer

call by name: re-evaluate the actual parameter on every use

Activation Records

Definition

Activation records is a block of information associated with each function activation, including the function's parameters and local variables.

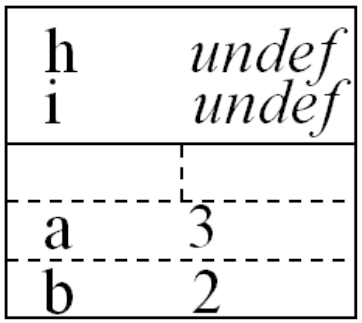
An individual activation record has space for:

- Parameters and local variables
- Return address
- Saved registers
- Temporary variables
- Return value
- Static link - to the function's static parent
- Dynamic link - to the activation record of the caller

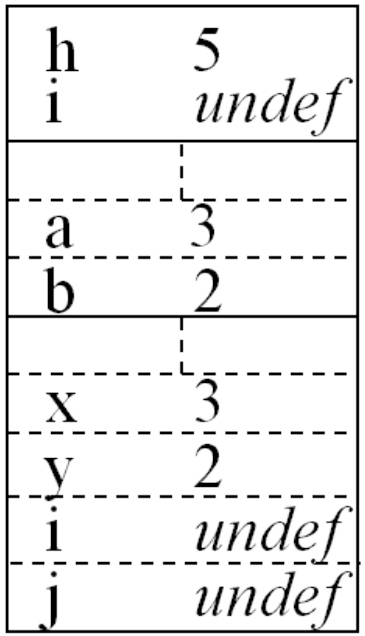
Example C/C++ Program

```
int h, i;
void B(int w) {
        int j, k;
        i = 2*w;
        w = w+1;
}

void A(int x, int y) {
        bool i, j;
        B(h);
}

int main() {
        int a, b;
        h = 5; a = 3; b = 2;
        A(a, b);
}
```

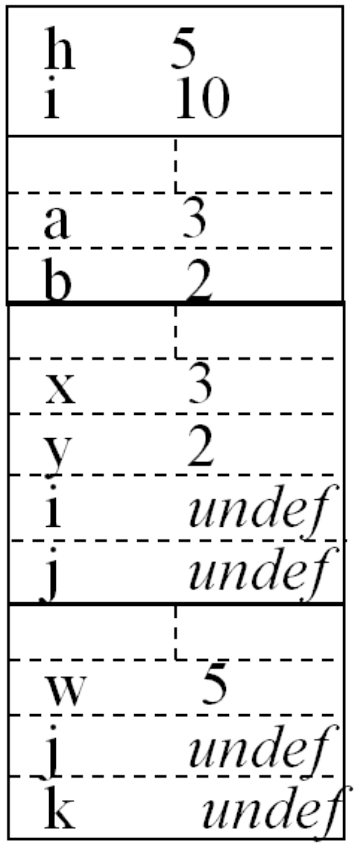
Run-Time Stack of Activation Records



Activation of main



main calls A



A calls B

Recursive Functions

Definition

A ***recursive function*** is a function that can call itself, either directly or indirectly.

Example

```
int factorial (int n) {
      if (n < 2)
            return 1;
      else
            return n * factorial(n-1);
}
```

Run Time Stack

Definition

A ***run-time stack*** is a stack of activation records used to model the semantics of function call and return.

A stack of activation records:

- Each new call pushes an activation record, and each completing call pops the topmost one.
- So, the topmost record is the most recent call, and the stack has all active calls at any run-time moment.

Run Time Stack

```
int factorial (int n) {
      if (n < 2)
            return 1;
      else
            return n * factorial(n-1);
}
```

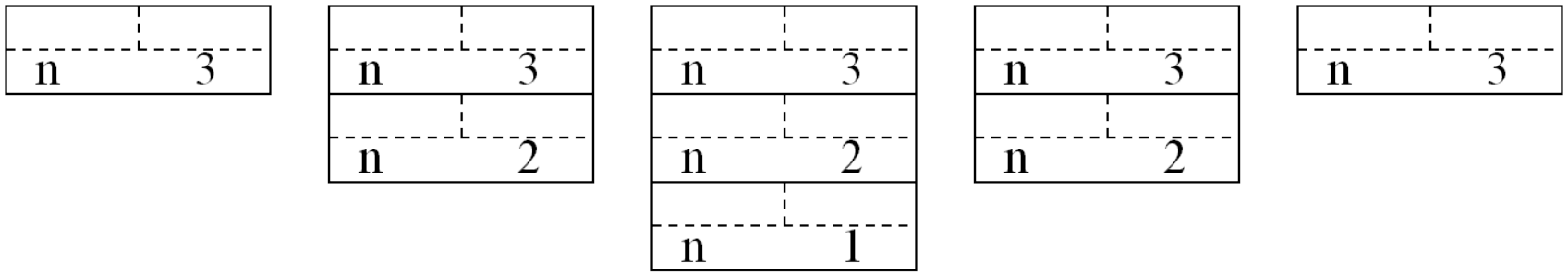
Example

Consider the call `factorial(3)`.

This places one activation record onto the stack and generates a second call `factorial(2)`.

This call generates the call `factorial(1)`, so that the stack gains three activation records.

Stack Activity for the Call `factorial(3)`



(a) First Call

(b) Second Call

(c) Third call returns 1

(d) Second call returns 2*1=2

(e) First call returns 3*2=6