16 Subroutines - Design Landscape

Monday, November 16, 2020 9:06 AM

Vocabulary and Design Considerations

- **Caller, function call, invocation**

```
double a = area(r);
```

- **Signature**

```
double area (double);
```

- **Arguments/actual parameters, formal parameters**
- **Return**

```
return <expression>
```

  - primitive type only?
  - Can be composite type?
  - Can be a function?
  - Can return 0 or 1 value?
  - Can return multiple values?

- **Parameter Association**
  - **Positional**

```
drawCircle(x1, y1, r1);
```

  - **Named/Keyword Parameters**

```
drawCircle(x=x1, y=y1, r=r1)
drawCircle(r=r1, y=y1, x=x1)
```

  - **Mixed Association**

```
drawCircle(x1, y1, r=r1)
drawCircle(x1, y=y1, r)
```

  - **Default Parameters**

```
def convert(n, base=2):
    . . .

binary = convert(12)
octal = convert(12, 8)
hexadecimal = convert(12, base=16)
```

  - **Variable number of arguments**

printf(*string*, *v1*, *v2*, *v3*, …);

16 Subroutines - Under the hood

Thursday, November 12, 2020 9:34 AM

```
void f (int[] x, int[] y, int[] z) {
   . . .
} // f()

main() {
   . . .
   f(a, b, c);
  …
}
```

On Invocation:

- Save CPU state
- Set up referencing environment and stack frame
- Transfer parameters
- Execute function code

On Return:

- Transfer parameters (if needed) and return value(s)
- Restore CPU state
- Continue caller execution

y = f(x) — in

swap(a, b); — in/out

max(a, b, c) — a, b: in; c: out

Role of parameters:

- Receive some data as input to subroutine
- Return some results in parameters
- Both

e.g. in | in out | out

```
void f (int[] x, int[] y, int[] z) {
   int[] m = new int[x.length];
   for (int i=0; i < x.length; i++) {
      y[i] = x[i] + y[i];
      z[i] = y[i];
   }
} // f()
```

Common Parameter Passing Schemes:

1. Pass by value — in
2. Pass by result — out
3. Pass by value-result — in out
4. Pass by reference — in out
5. Pass by name

Big Question

Which parameter passing scheme is used by the PLs I know?

16 Subroutines - Parameter Passing Schemes

Monday, November 16, 2020 8:49 AM

Pass by value in

On invocation:
- Allocate space for formal parameters on stack frame
- Evaluate actual parameters
- Copy value of actual parameters in to formal parameters
- Execute function

On Return
- Transfer return value (if any)
- Pop stack frame

Pass by result out

On invocation:
- Allocate space for formal parameters on stack frame
- Execute function

On Return
- Copy values of formal parameters into actual parameters
- Transfer return value (if any)
- Pop stack frame

Pass by value-result in out

On invocation:
- Allocate space for formal parameters on stack frame
- Evaluate actual parameters
- Copy value of actual parameters in to formal parameters
- Execute function

On Return:
- Copy values of formal parameters into actual parameters
- Transfer return value (if any)
- Pop stack frame

pass by copy

Pass by reference in out

On Invocation:
- Copy the reference of actual parameters into formal parameters in stack frame

On Return:
- Pop the stack frame

Pass by name

On Invocation
- Substitute textually the names (expressions) of actual parameters into text of function
- Execute the function

On Return
- Pop the stack frame

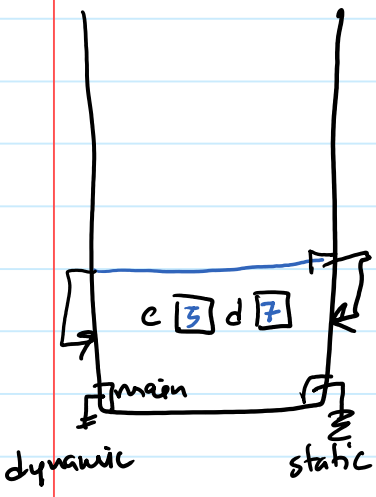
f(c,d)

f(a,b)
f(a,5)
f(a, b+c)
f(a, g(b,c))

16 Subroutines - Parameter Passing Examples

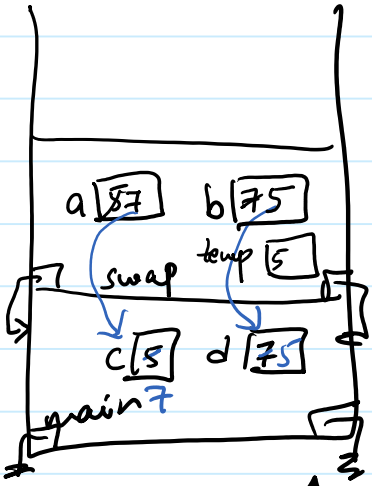
Monday, November 16, 2020 8:49 AM

pass-by-value



pass-by-result X

pass-by-value-result



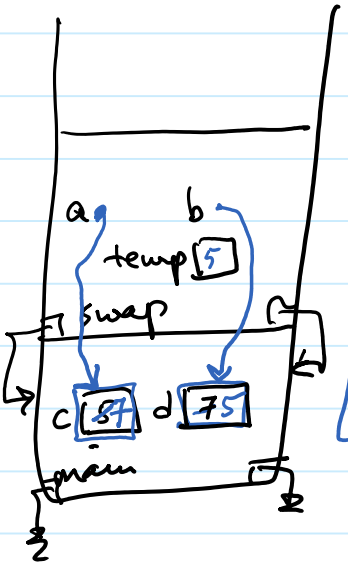

```
void swap (int a, int b)
{
    int temp = a;
    a = b;
    b = temp;
} // swap()
```

```
int c = 5;
int d = 7;
swap(c,d);
```

5 7

c = 7
d = 5

pass-by-reference



c = 7
d = 5

same

C - passes all parameters by value.

But simulate pass-by-reference because C has pointers + pointer operators.

```
void swap (int *a, int *b){
    int temp = *a;
    *a = *b;
    *b = temp;
} //swap()
swap(&c, &d);
```

C++ has reference parameters

```
void swap (int &a, int &b){
    int temp = a;
    a = b;
    b = temp;
} //swap()
```

swap(c,d);

procedure swap(c, d : integer) — value

procedure swap (var c, d : —) — ref

16 Parameter Passing - Examples

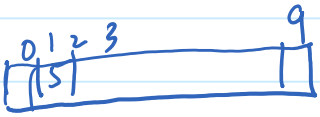
Monday, November 16, 2020 8:49 AM

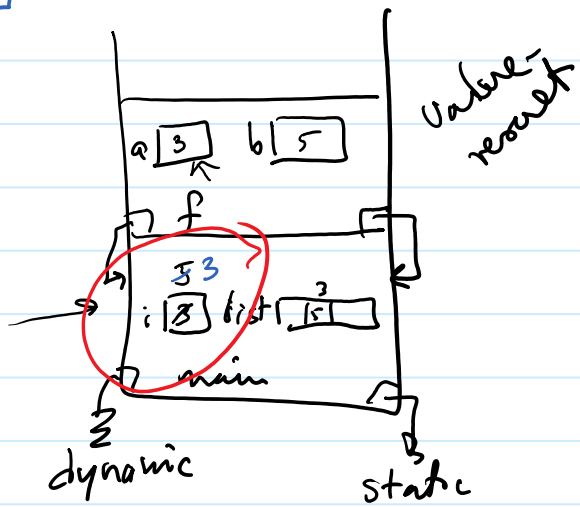
Pass by value-result versus pass by reference when aliasing is involved

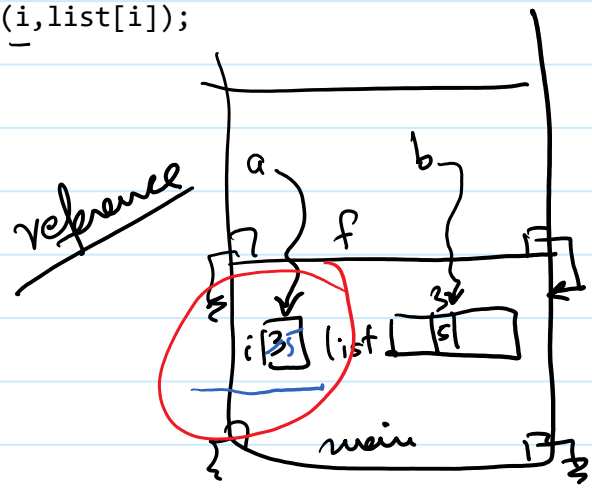
```
int i = 3;

void f(int a, int b) {
    i = b;
} // f()

int list[10];
list[i] = 5;
f(i,list[i]);
```



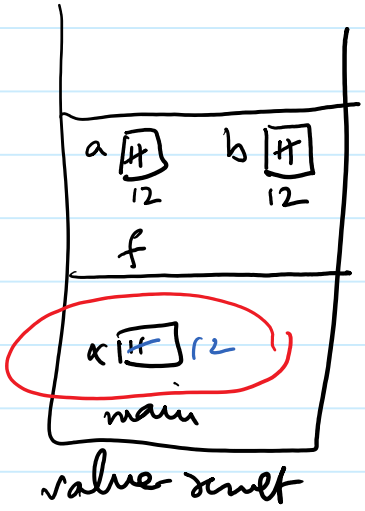


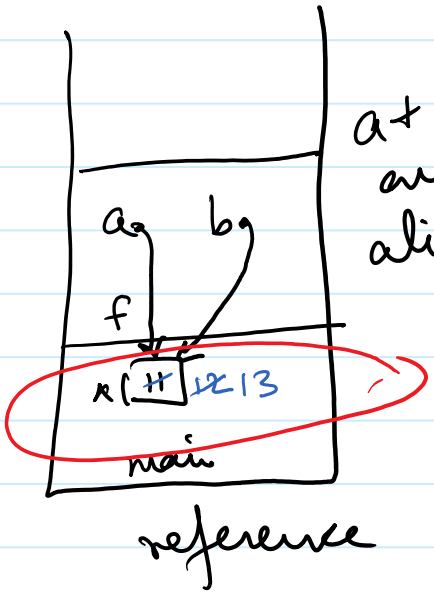



```
void f(int a, int b) {
    a++;
    b++;
} // f()

int x = 11;
f(x, x);
```

x = 12





16 Subroutines - Pass by Name

Monday, November 16, 2020 8:49 AM

```
void swap(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
} // swap()

int c = 5;
int d = 7;

swap(c, d);
```

int temp = c;
c = d;
d = temp

c [5|7] d [7|5]

temp [5]

```
swap(i, a[i])
     3
```

int temp = i;
i = a[i];
a[i] = temp

i = [3|3] 0 0 3

a [0|1|2|3]

temp [3]

```
int i;
int a[10];

void f(int x) {
    i++;
    x++;
}

    i = 1;
    a[i] = 1;
    a[2] = 2;
    f(a[i]);
```

Try with reference + value-result

Call-by-Name

- macros

Functional PLs use pass-by-Name → closure

#define N 5

#define max(a,b) a>b?a:b

x = max(c,d)

16 Subroutines - Parameter Schemes in Programming Languages

Monday, November 16, 2020 9:06 AM

C : pass-by-value, can do pass-by reference using pointers
Arrays are by reference

C++ : pass by value, but also has reference types

C# : value, has pass-by reference

```
void swap(ref int a, ref int b){
|
}                          swap(ref c, ref d);
```

Python + Ruby : pass-by value ??
- reference model for variables
- some types are immutable → value

fast.com