* Subroutines
* Parameters Passing

* Lab 7+8 posted
* Assignment 6 posted.

11/21

Subroutines

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

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

On invocation

- save the 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 ← pop stack frame
- Continue caller execution

Role of Parameters

- recieve some data as input to subroutine
- return some results in parameters
- both

e.g. `y = f(x);` — return value; x: in

`swap(a, b);` — a, b: in/out

Common Parameter Passing Schemes

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

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

1. Pass-by-value [in]

Invocation
- allocate space for formal parameters on stack frame
- Evaluate actual parameters
- Copy value of actual parameters into formal parameta
- Execute function

Return
- Transfer return value (if any)
- pop stack frame.

2. Pass-by-result [out]

Invocation
- Allocate space for formal parameters on stack frame
- Execute function

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

3. Pass-by-value-result [in-out]

Invocation
- Same as pass by value

Return
- same as pass by result.

4. Pass by reference [in-out]

Invocation
- Copy the reference of actual parameters into formal parameters
- Execute function

Return
- Pop stack frame.

5. Pass by Name

Invocation
- textually substitute the names (or expressions) formal parameters
- Execute the function

Return
- Pop the stack frame.

Examples →

C/Java

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

main()
    int x = 5, y = 7;
    swap(x, y);
```

1. pass-by-value
2. pass by result
3. pass by value-result
4. pass by reference
5. pass by name.

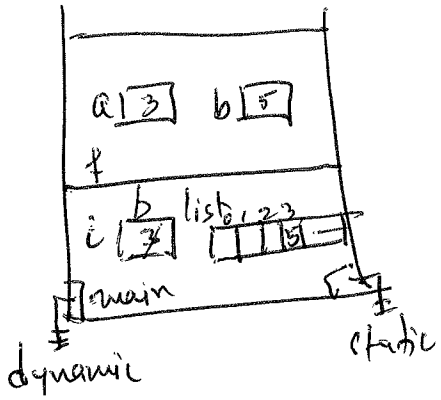
Issues

① Aliasing can lead to issues.

e.g.

```
int i=3;

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

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

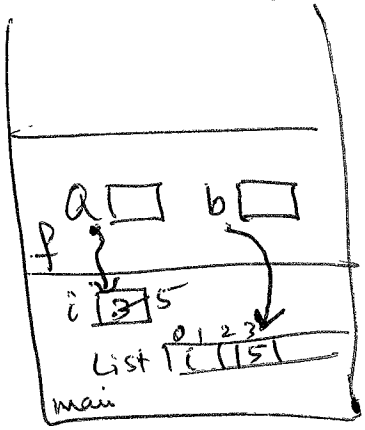
Value-Result



On return a=3 is copied into i, so change in i=5 is gone.

Reference

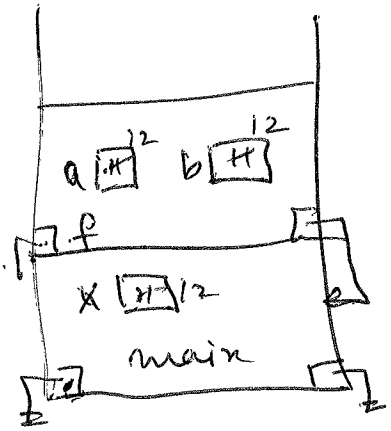
a+b are aliases!



e.g.

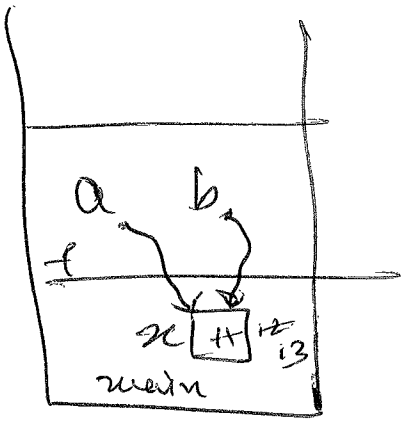
```
void f (int a, int b) {
    a++;
    b++;
}

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

Value-Result



$x = 12$

Reference



$x = 13$

In Swap

by swap(i, a[i])

all by name.

Parameter Passing Schemes in PLs

C: pass-by-value,
can do pass-by-reference using pointers.
Arrays are "by reference" since a[0] ≡ @a[]

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

C#: pass by value, also has pass-by reference

```
void swap (ref int a, ref int b){
   |
   3
```

call: swap (ref x, ref y);

Python: · pass-by-~~value~~ assignment
· a combination of pass-by-value
+ pass by reference

- immutable objects (numbers, strings, tuples)
use pass by value
- mutable objects (lists, dictionaries)
ref to object is assigned to formal parameter
i.e. passed by reference