Procedures and Functions

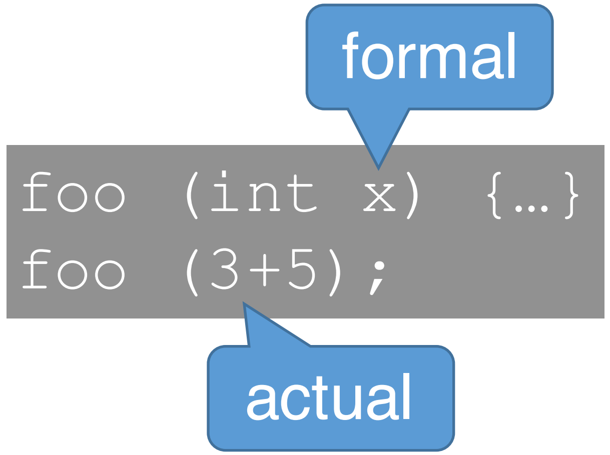
CMPSC 461
Programming Language Concepts
Penn State University
Fall 2016

Parameter Passing

How caller communicates with callee

Formal parameters: names in the declaration of a function

Actual parameters: variables/expressions passed to a function



```
foo (int x) {…}
foo (3+5);
```

Parameter Modes

Call-By-Value (CBV)

Call-By-Reference (CBR)

Call-By-Value-Return (CBVR)

Call-By-Name (CBN)

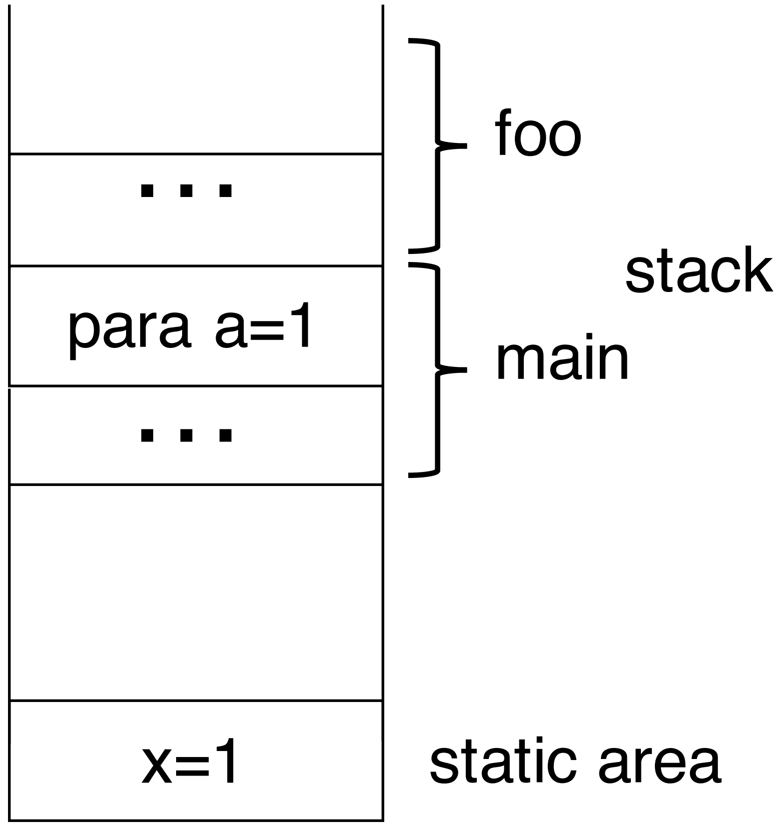
Call-By-Value

Calling mechanism

- Arguments are evaluated to their values
- Memory or registers allocated for arguments on AR
- Argument values copied to AR
- AR destroyed when callee returns

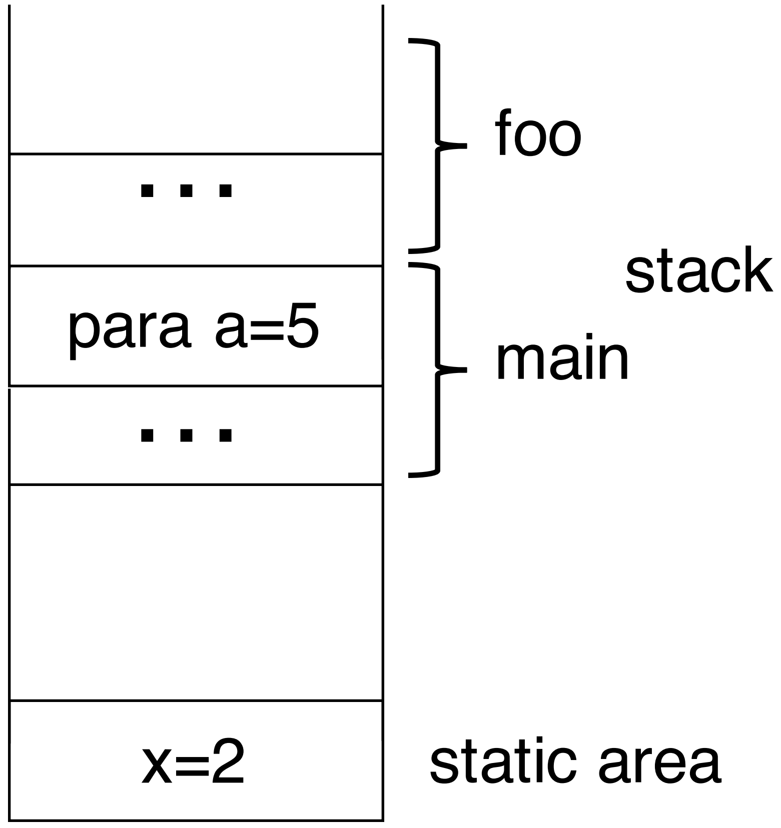
Call-By-Value

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    foo(x); //result?
    print(x); //result?
}
```



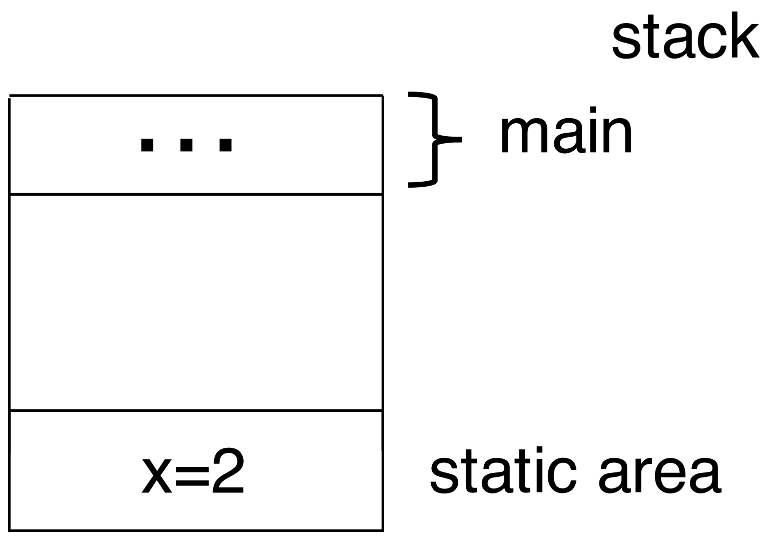
Call-By-Value

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    foo(x); //result?
    print(x); //result?
}
```



Call-By-Value

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    foo(x); //result?
    print(x); //result?
}
```



Call-By-Value

Formal & Actual parameters have separate memory: their values may diverge

Characteristics

- Actual parameters may not directly be changed in callee (unless pass in pointers)
- Arguments can be complex expressions
- Simple and intuitive (less error-prone)

Call-By-Value: Performance

Primitive types: cost per parameter is small

Arrays, records, structures: copying value could be slow

C: programmer has to pass pointers/references to avoid copying cost

Java: only primitive types call-by-value

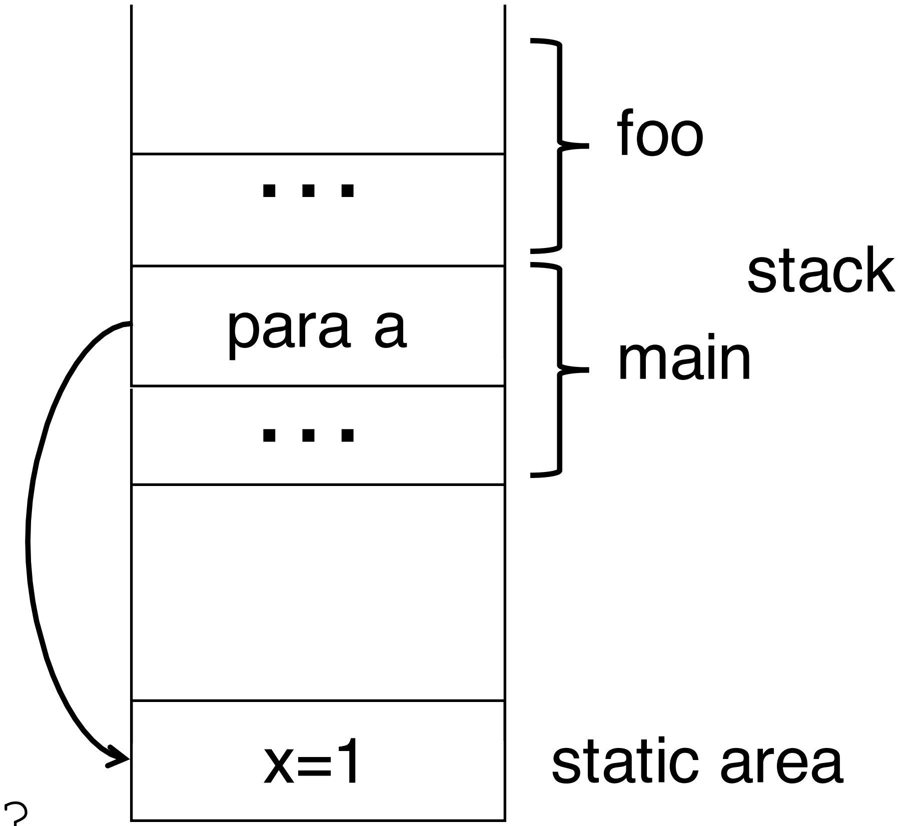
Call-By-Reference

Calling mechanism

- Arguments are evaluated to their values
- Memory or registers allocated for arguments on AR
- Argument ***address*** stored in AR
- AR destroyed when callee returns

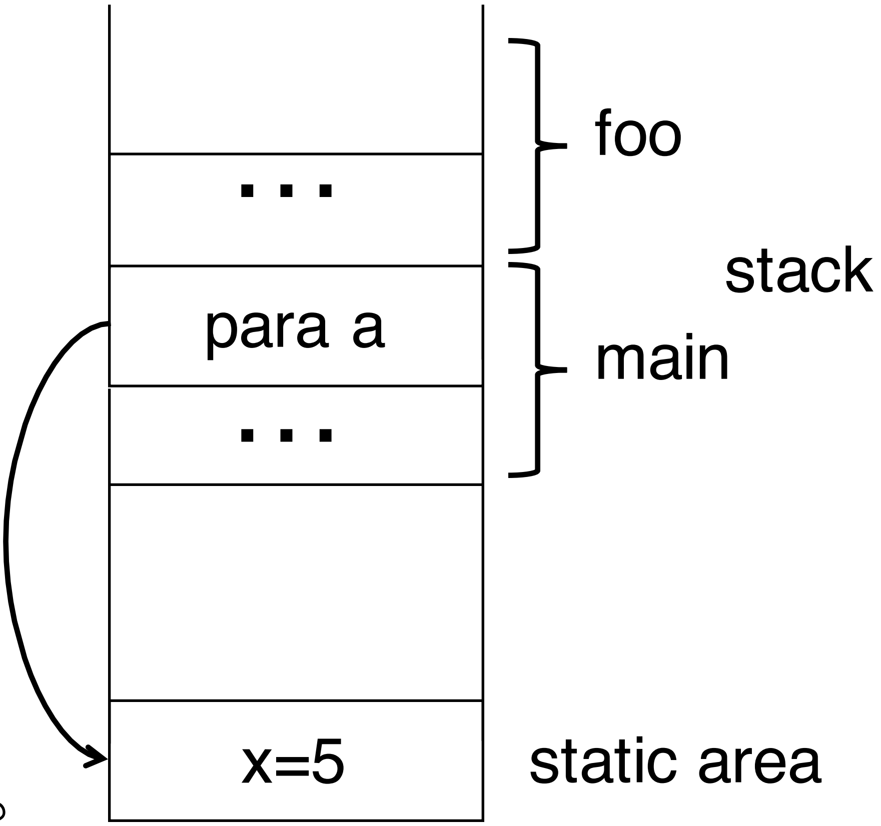
Call-By-Reference

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    foo(x); //result?
    print(x); //result?
}
```



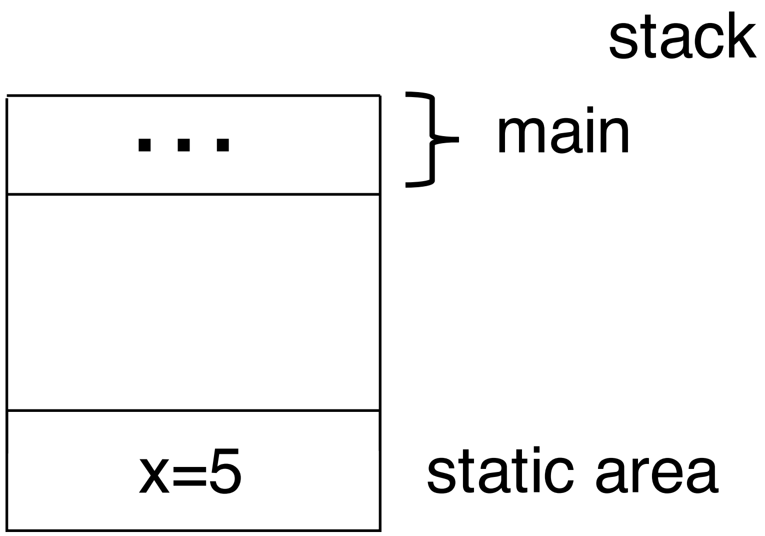
Call-By-Reference

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    foo(x); //result?
    print(x); //result?
}
```



Call-By-Reference

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    foo(x); //result?
    print(x); //result?
}
```



Call-By-Reference

Formal parameter is an alias of actual parameter: their values are the same

Characteristics

- Actual parameters may directly be changed in callee
- Some language disallows complex expressions as arguments
- Programs are harder to understand (more error-prone)

Call-By-Reference: Performance

Avoids the cost of memory copy

Indirect memory access: address → value

C: call-by-value is the default mode

Java: constructed types call-by-reference

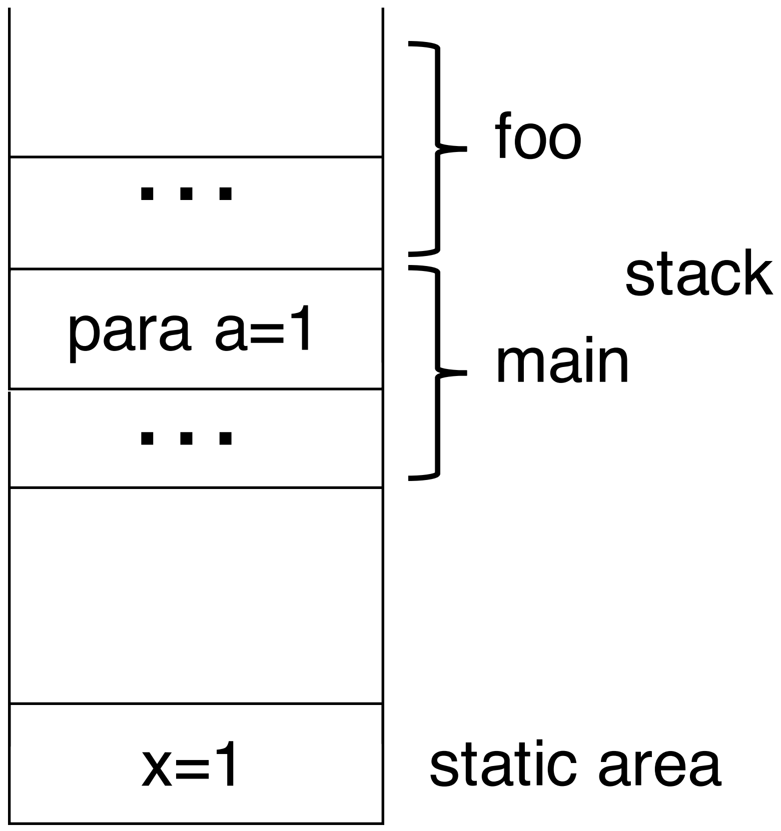
Call-By-Value-Result (In-Out)

Calling mechanism

- Arguments are evaluated to their values
- Memory or registers allocated for arguments on AR
- Argument values stored in AR
- Before callee returns, ***AR values copied back to actual arguments***
- AR destroyed when callee returns

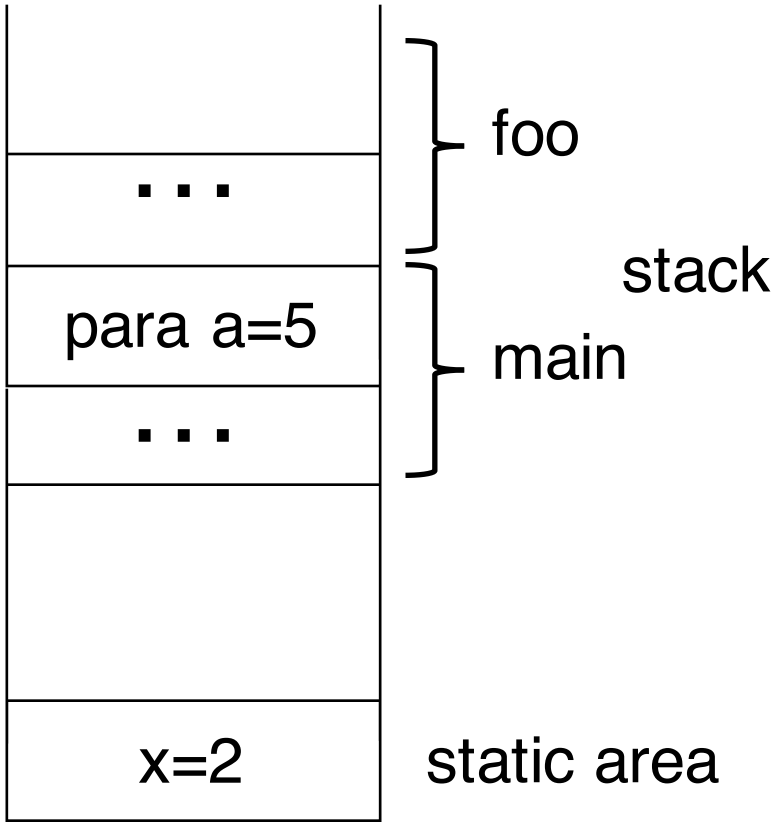
Call-By-Value-Result

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    foo(x); //result?
    print(x); //result?
}
```



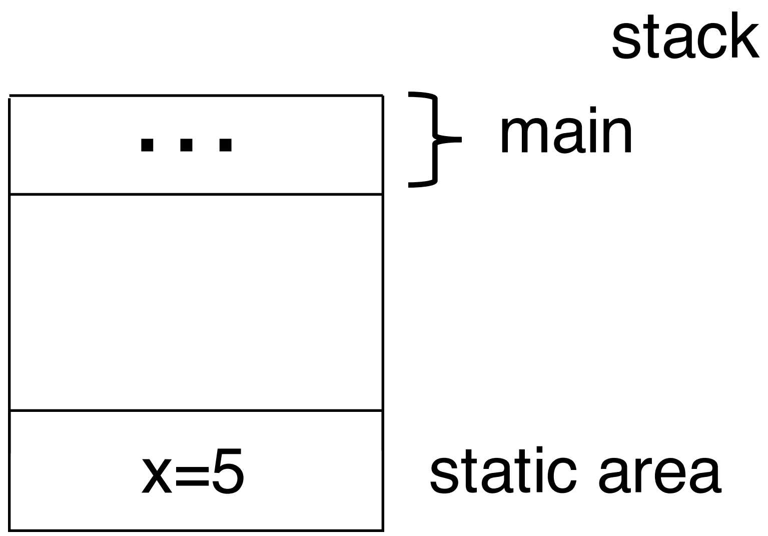
Call-By-Value-Result

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    foo(x); //result?
    print(x); //result?
}
```



Call-By-Value-Result

```
int x=1;
int foo (int a) {
    x = 2;
    a = 5;
    return x+a;
}
void main() {
    foo(x); //result?
    print(x); //result?
}
```



Call-By-Value-Return

Characteristics

- Mostly identical to call-by-value, except an extra step of copying values back to actual parameter

Call-By-Name (Lazy Evaluation)

Calling mechanism

- Arguments are ***not*** evaluated to their values (like macros)
- Actual parameters replace all formal parameters in body

```
void swap (int a, int b) {
    int t = a;
    a = b;
    b = t;
}
swap(i, A[i]) // value swapped?
```

Still, a useful mode in functional programming, e.g., Haskell

Return Values

Fixed size: callee stores values in caller’s AR

Size determined at run time: callee stores results in heap, and store their addresses in caller’s AR