Parameter Passing

COMP360

“Part of what made the Macintosh great was that the people working on it were musicians, poets, and artists, and zoologists, and historians. They also happened to be the best computer scientists in the world. But if it hadn't been computer science, these people would have been doing amazing things in other fields.”

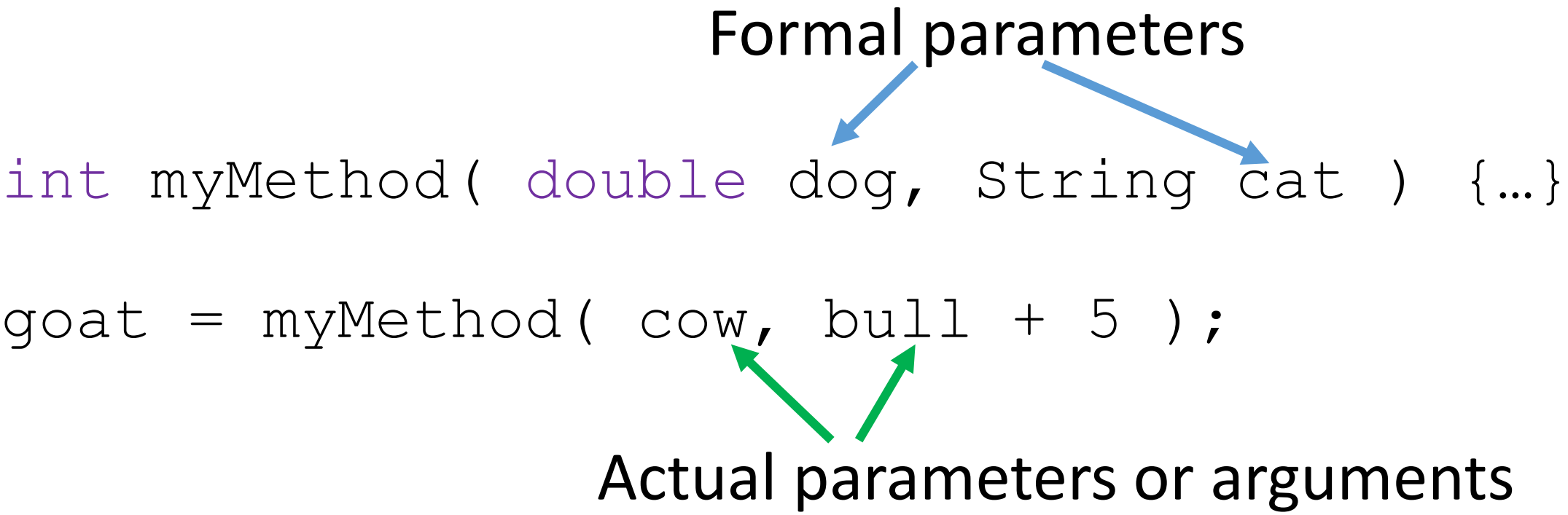
Steve Jobs

Goals

- Understand the different ways parameters can be passed to a method
- Know how parameter passing and method calls are implemented at the hardware level

Formal and Actual

- A formal parameter is defined in a method header
- An actual parameter is the value passed to a method



Parameter Passing Paradigms

Call by

- reference
- value (in)
- value result (in out)
- result (out)
- constant value
- name

Call by Value

- The value of the actual parameter is copied to the formal parameter
- Changing the formal parameter in the method has no impact on the actual parameter in the calling program
- Used for simple variables in Java and C++
- Sometimes referred to as an **IN** parameter

Call by Reference

- Changing the formal parameter in the method changes the actual parameter in the calling program
- The actual parameter must be a variable and not an expression
- Used for object parameters in Java

Value and Reference Examples

```
void bothWays( int cat, Widget dog ) {
    cat = 47;
    dog.value = 13;
}

frog = 5;
toad.value = 2;
bothWays( frog, toad );
```

- Frog is still 5 but toad.value is 13

Call by Result

- Used to return a value from the method to the calling program
- The initial value of the actual parameter is never used
- Sometimes referred to as an **OUT** parameter
- Identical to pass by reference, but ignores initial value

Call by Value Result

- The value of the actual parameter is copied to the local formal parameter at the beginning of the method
- At the end of the method the value is copied back to the calling program
- Similar to call by reference
  - can be implemented differently
  - during method execution the actual parameter is not changed

Call by Value Result Example

```
int cow = 5;                    // Class instance variable
// in main
methodVR( cow );
print cow;                      // displays 7

void methodVR( int bull ) {
        bull = 7
        print cow, bull;        // displays 5 7
}
```

Call by Constant Value

- Similar to call by value, but the parameter may not be changed
- The compiler does not allow such a parameter to appear on the left of an equals sign

Which paradigm allows you to change the value of a parameter?

A. Pass by value
B. Pass by reference
C. Pass by constant value
D. All of the above
E None of the above

Call by Name

- Works as if the actual parameter is copied to the formal parameter every place it is used in the method
- Used in most macro expansions, such as #defines in C++
- Can be difficult to implement
- Used by Algol and Snobol

C Macros

```
#define  goat   cat + dog

cow = goat;              // cow = cat + dog;

cow = 2 * goat;          // cow = 2 * cat + dog;
```

Call by Name Example

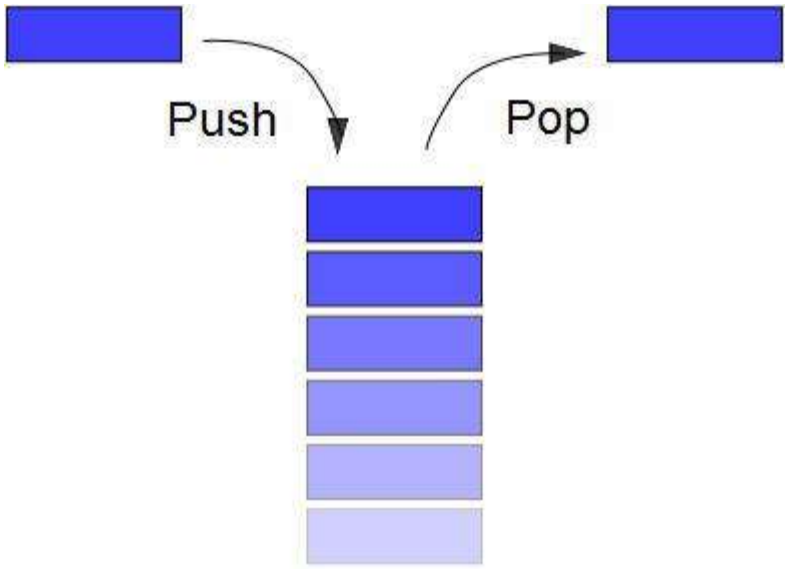
```
int  cat = 1, dog;
dog = afunc( cat + 5 );
        . . .
int afunc( int pig ) {
        int horse = pig;        // horse = 6
        cat = 3;
        int cow = pig;          // cow = 8
}
```

Thunk

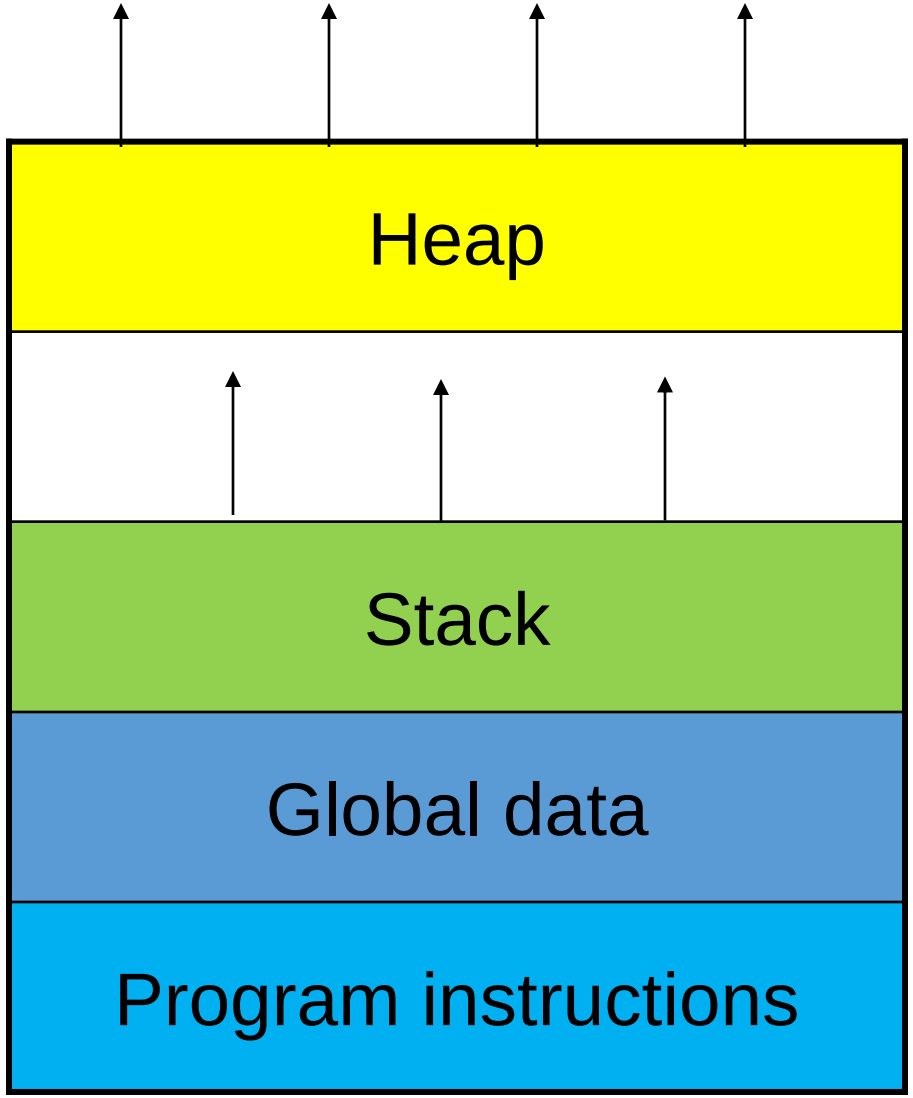
- In most cases, when a language supports call by name, the complier automatically generates a function to calculate the parameter
- This function is called a thunk
- The thunk is called in a method whenever it needs the value of the parameter.

Stacks

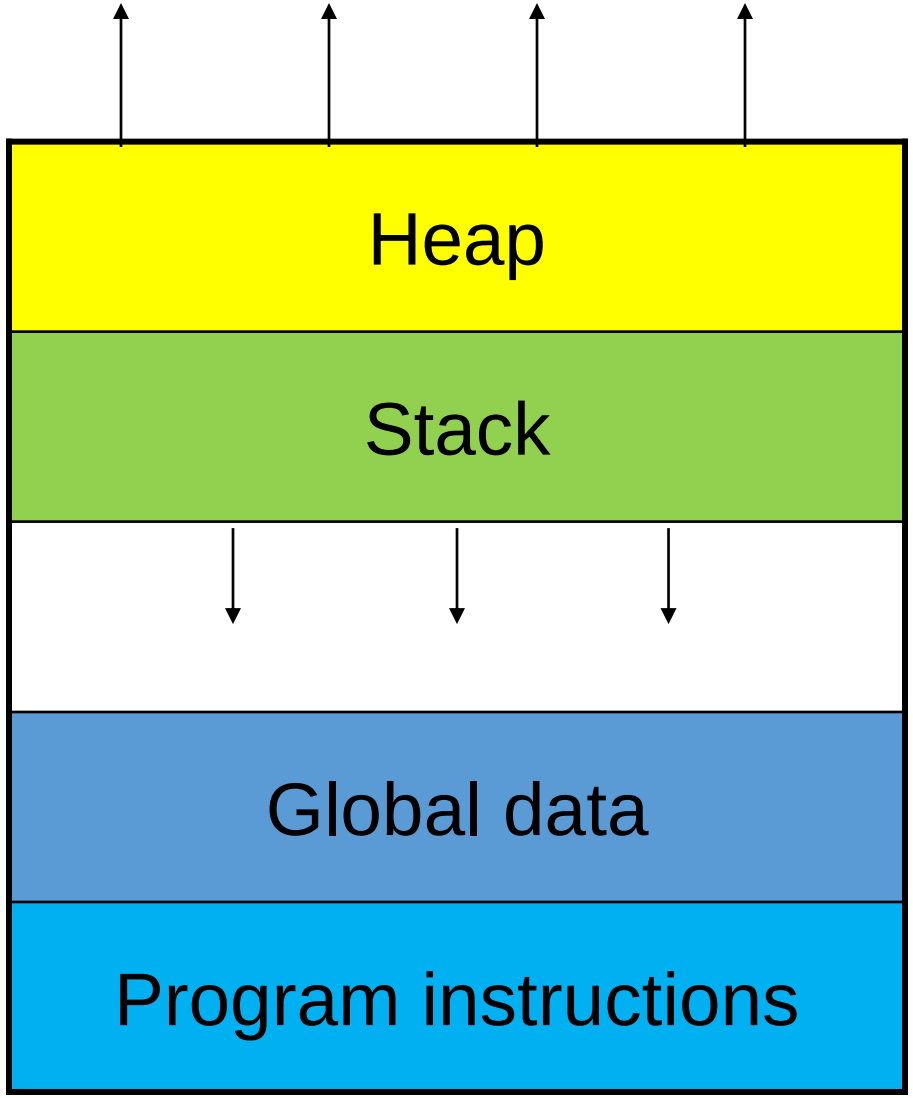
- Many programming languages use stacks to pass parameters
- Many computer architectures have stack instructions to help implement these programming languages
- Most architectures have stack pointer register. The stack pointer always points to the top item on the stack.



Program Memory Organization



Program Memory Organization



Intel method

Function Call Hardware

- All computers have machine language instructions to support function calls
- The level of hardware support varies with modern computers providing more support

Intel Call instruction

- The **`CALL`** instruction basically pushes the program counter on the stack and branches to a new location
- There are many versions of the Intel **`CALL`** instruction to support different addressing modes and changes in privileges

Intel RET instruction

- The **`RET`** or return instruction pops a value from the stack and places it in the program counter register
- Since the program counter contains the address of the next instruction to execute, this has the effect of branching back to the calling program

The return address pushed on the stack points to an address in

A. program instructions
B. global data
C. stack
D. heap
E none of the above

Basic Steps to Call a Method

- Compute any equations used in the parameters, such as `x=func(a + b);`
- Push the parameter values on the stack
- Execute a call instruction to push the return address on the stack and start execution at the first address of the function

Upon function entry

- Save the contents of the registers
  - Many systems have the convention that a method should return with the registers just the way they were when called
- Link the activation records
- Increase the stack pointer to reserve memory for the local variable
- Start executing the function code

Upon function exit

- Reduce the stack by the size of the local variable
- Pop the register values
- Execute the return instruction to pop the address from the stack into the program counter

When a method is called many times in a program, how does it know where to return?

A. Call address in the machine language
B. Return address on the stack
C. Returns to earliest call in the source code
D. Depends on count in Program Counter

Activation Records

- An activation record or frame contains the stack information for a method call
- The activation records are linked together

Stack Activation Frame Format

locals	The local variables of the method. This can vary in size.
Frame pointer	The address of the previous activation frame.
Return address	The address of the instruction after the method call in the calling program.
parameter 2	The second parameter to the method
parameter 1	The first parameter to the method

Example Function Call

- Consider the function

```
void thefunc(Widget b, int a ){
    int r = a;
}
```

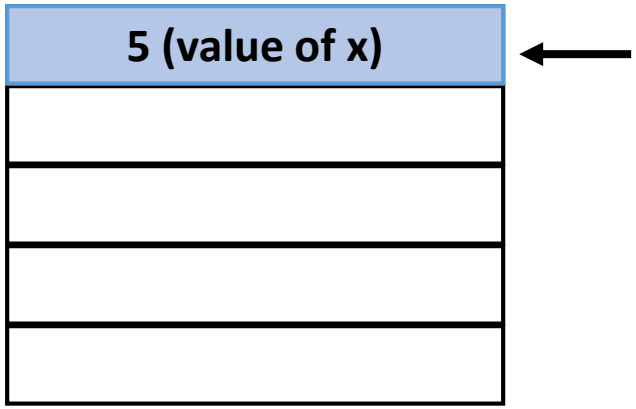
- that is called by the main program

```
int x = 5;
Widget y = new Widget();
thefunc( y, x );
```

- The Widget y is passed by reference. The int x is passed by value.

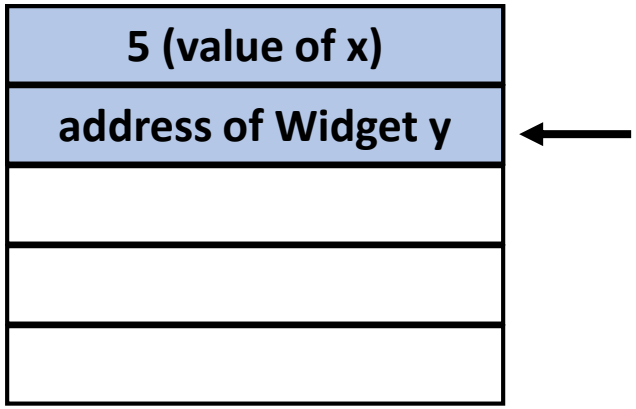
Stack for Call Parameters

- push x



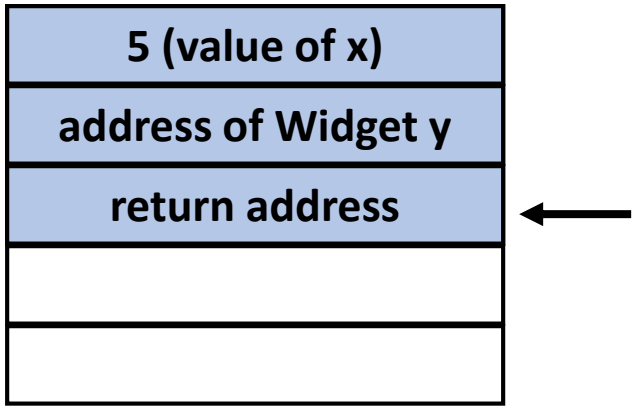
Stack for Call Parameters

- push x
- push address of y



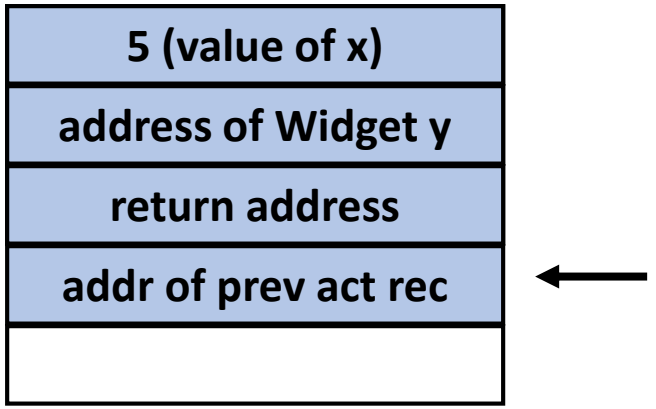
Stack for Call

- push x
- push address of y
- call thefunc



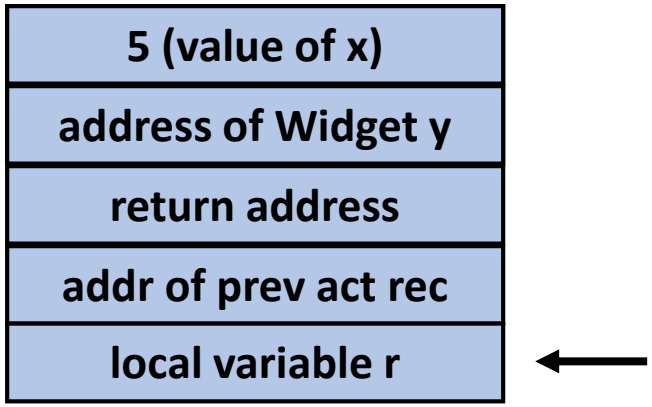
Stack with Activation Records

- push x
- push address of y
- call thefunc
- Link to previous activation record



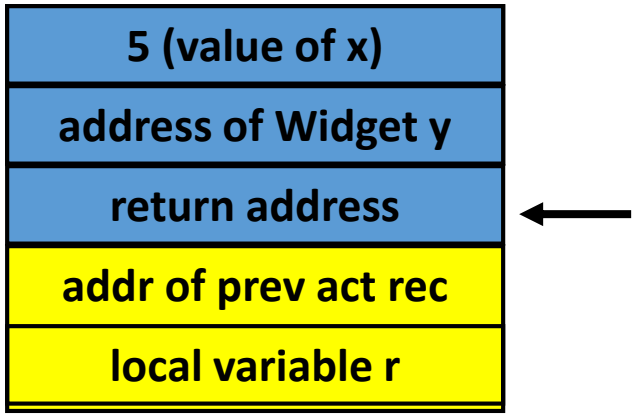
Stack Use by Function

- push x
- push address of y
- call **`thefunc`**
- Link to previous activation record
- increment stack



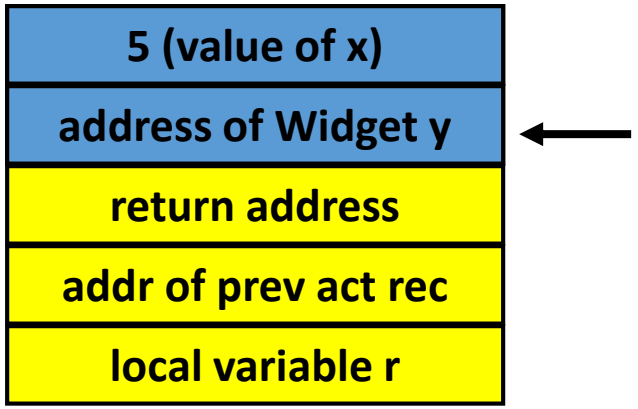
Stack for Return

- push x
- push address of y
- call **`thefunc`**
- Link to previous activation record
- increment stack
- decrement stack



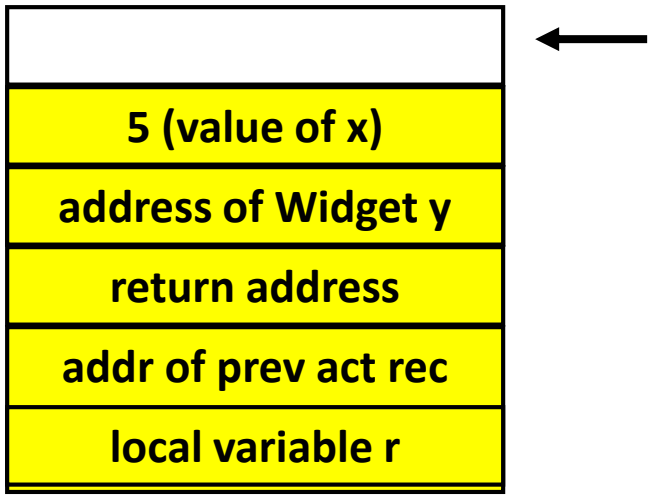
Stack for Return

- push x
- push address of y
- call **`thefunc`**
- Link to previous activation record
- increment stack
- decrement stack
- return



Cleanup Stack

- push x
- push address of y
- call **`thefunc`**
- increment stack
- decrement stack
- return
- decrement stack by 2



Explain the Implementation

Working in teams of students, explain how the following parameter passing paradigms can be implemented

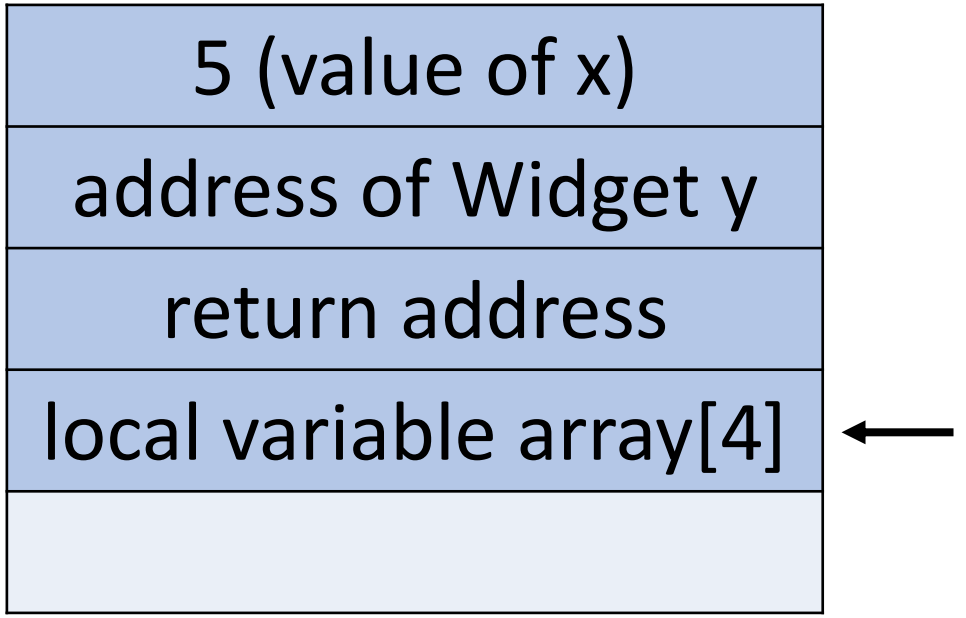
- reference
- value (in)
- value result (in out)
- result (out)

Exceeded Array Bounds

- In many languages, including C and C++, the system will not detect that a method has indexed beyond the end of an array
- If a program stores data in an array using an index bigger than the size of the array, the data will be stored in whatever memory follows the array

Stack Overflow Attack

- A common security attack is to cause a program to overflow the stack
- If the program stores a value into array[4], it will right in the data past array, the return address
- Instructions might be loaded in the rest of the stack



Stack Protection

- Good programs should check all indexes to ensure they are within range
- Avoid functions that do not check limits, such as **`cin`**
- Some processors prohibit instructions from being fetched from the stack
- Java always checks array indexes

Haskell

- Review Functional programming in Wikipedia
- Download and install Haskell from www.Haskell.org
- Start learning to program in Haskell
- A nice tutorial is available at http://learnyouahaskell.com/introduction