
NJIT
THE EDGE IN KNOWLEDGE

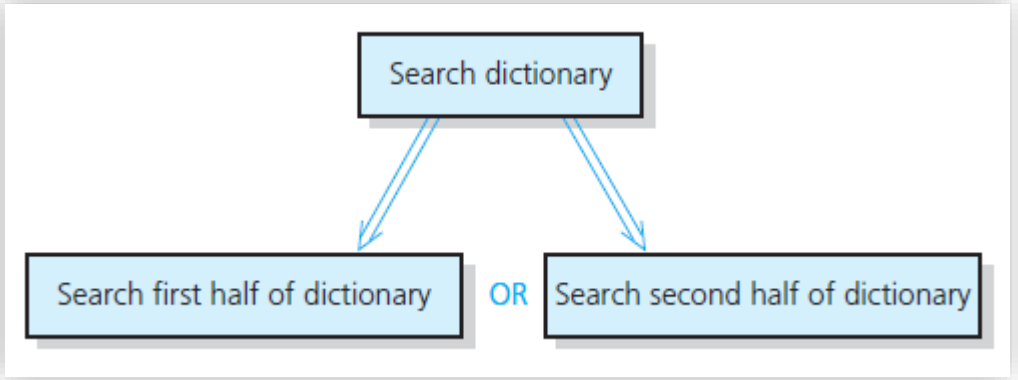
CS 280
Programming Language Concepts

Spring 2023

Recursion

Recursion

- Recursion breaks problem into smaller identical problems
  - An alternative to iteration



Recursive Solutions

- A recursive function calls itself
- Each recursive call solves an identical, but smaller, problem
- Test for base case enables recursive calls to stop
- Eventually, one of smaller problems must be the base case

Questions for constructing recursive solutions

1. How to define the problem in terms of a smaller problems of same type?
2. How does each recursive call diminish the size of the problem?
3. What instance of problem can serve as base case?
4. As problem size diminishes, will you reach base case?

Finding a Recursive Solutions

- A recursive solution to a problem must be written carefully
- The idea is for each successive recursive call to bring you one step closer to a situation in which the problem can easily be solved
- This easily solved situation is called the base case
- Each recursive algorithm must have at least one base case, as well as a general (recursive) case

General format for Many Recursive Functions

```
if   (some easily-solved condition)     // Base case

     solution statement

else                                    // General case

     recursive function call
```

Some examples . . .

Writing a Recursive Function to Find the Sum of the Numbers from 1 to n

DISCUSSION

The function call Summation(4) should have value 10, because that is 1 + 2 + 3 + 4

For an easily-solved situation, the sum of the numbers from 1 to 1 is certainly just 1

So our base case could be along the lines of

```
if  (n == 1)
        return 1;
```

Writing a Recursive Function to Find the Sum of the Numbers from 1 to n

Now for the general case. . .

The sum of the numbers from 1 to n, that is,
1 + 2 + . . . + n can be written as

n + the sum of the numbers from 1 to (n - 1),
that is, n + 1 + 2 + . . . + (n - 1)

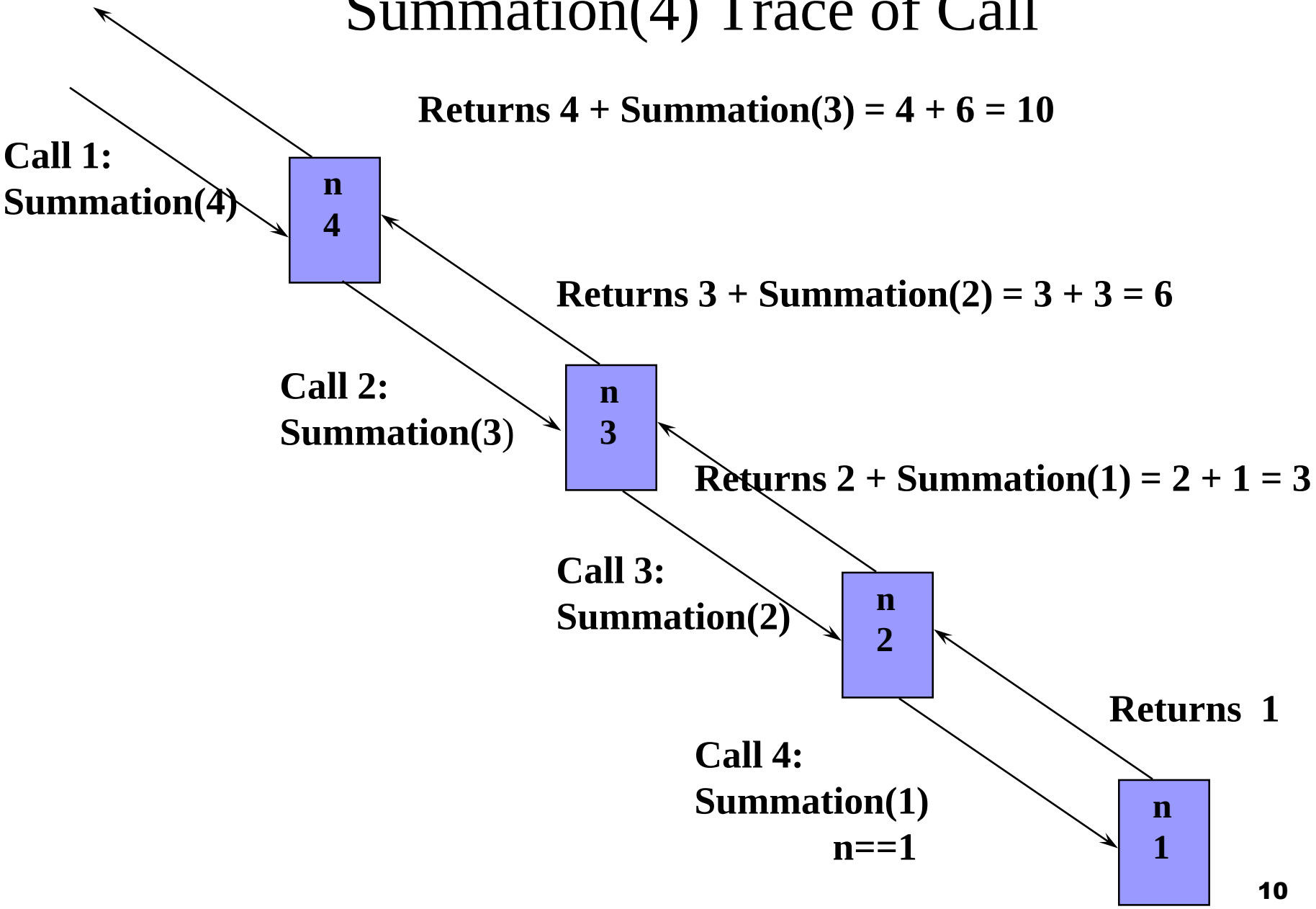
or, n + Summation(n - 1)

And notice that the recursive call Summation(n - 1) gets us "closer" to the base case of Summation(1)

Finding the Sum of the Numbers from 1 to n

```
int   Summation (/* in */  int   n)
{
    if  (n == 1)      // Base case
        return  1;
    else              // General case
        return (n + Summation (n - 1));
}
```

Summation(4) Trace of Call



Writing a Recursive Function to Find n Factorial

DISCUSSION

The function call Factorial(4) should have value 24, because that is 4 * 3 * 2 * 1

For a situation in which the answer is known, the value of 0! is 1

So our base case could be along the lines of

```
if  (number == 0)
        return 1;
```

Writing a Recursive Function to Find Factorial(n)

Now for the general case . . .

The value of Factorial(n) can be written as
n * the product of the numbers from (n - 1) to 1,
that is,

$$n * \underbrace{(n - 1) * \ldots * 1}$$

or, n * Factorial(n - 1)

And notice that the recursive call Factorial(n - 1) gets us "closer" to the base case of Factorial(0)

Recursive Solution

```
int   Factorial ( int   number)
// Pre: number is assigned and number >= 0
{
    if  (number == 0)     // Base case
        return  1;
    else                  // General case
        return
          number * Factorial (number - 1);
}
```

The Base Case Can Be "Do Nothing"

```
void   PrintStars (/* in */  int   n)
// Prints n asterisks, one to a line
// Precondition:   n is assigned
// Postcondition:
//    IF n <= 0, n stars have been written
//    ELSE call PrintStarg
{
    if  (n <= 0)  // Base case: do nothing
    else
    {
        cout   <<  '*'  <<  endl;
        PrintStars (n - 1);
    }
}
```

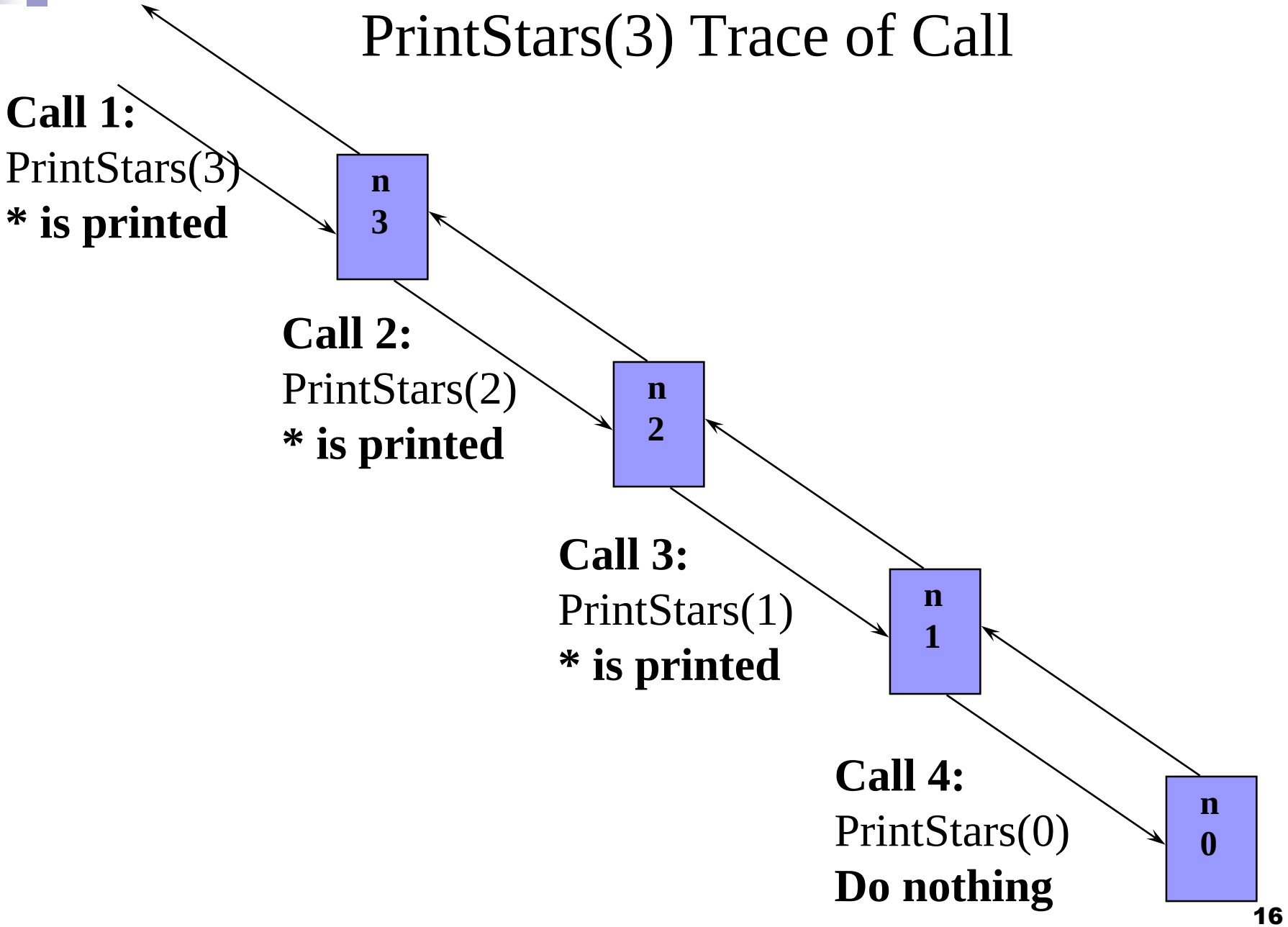
// Can rewrite as . . .

Recursive Void Function

```
void  PrintStars (/* in */  int   n)
//    Prints n asterisks, one to a line
//    Precondition:  n is assigned
//    Postcondition:
//         IF n > 0, call PrintSars
//         ELSE n stars have been written
{
    if  (n > 0)  // General case
    {
        cout  <<  '*'  <<  endl;
        PrintStars (n - 1);
    }
    // Base case is empty else-clause
}
```


15

PrintStars(3) Trace of Call

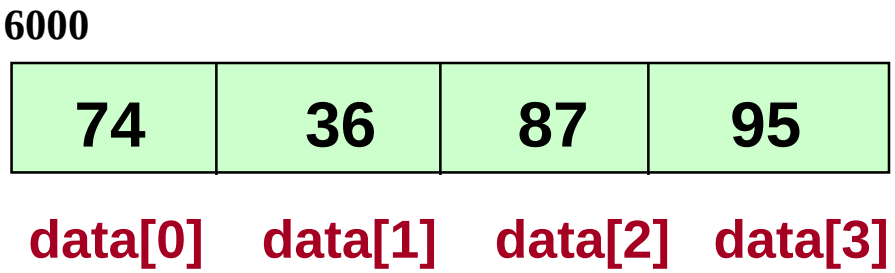


Writing a Recursive Function to Print Array Elements in Reverse Order

DISCUSSION

For this task, we will use the prototype:

void PrintRev(const int data[ ], int first, int last);



The call

PrintRev (data, 0, 3);

should produce this output: 95 87 36 74

Base Case and General Case

A base case may be a solution in terms of a "smaller" array

Certainly for an array with 0 elements, there is no more processing to do.

The general case needs to bring us closer to the base case situation

if the length of the array to be processed decreases by 1 with each recursive call, we eventually reach the situation where 0 array elements are left to be processed.

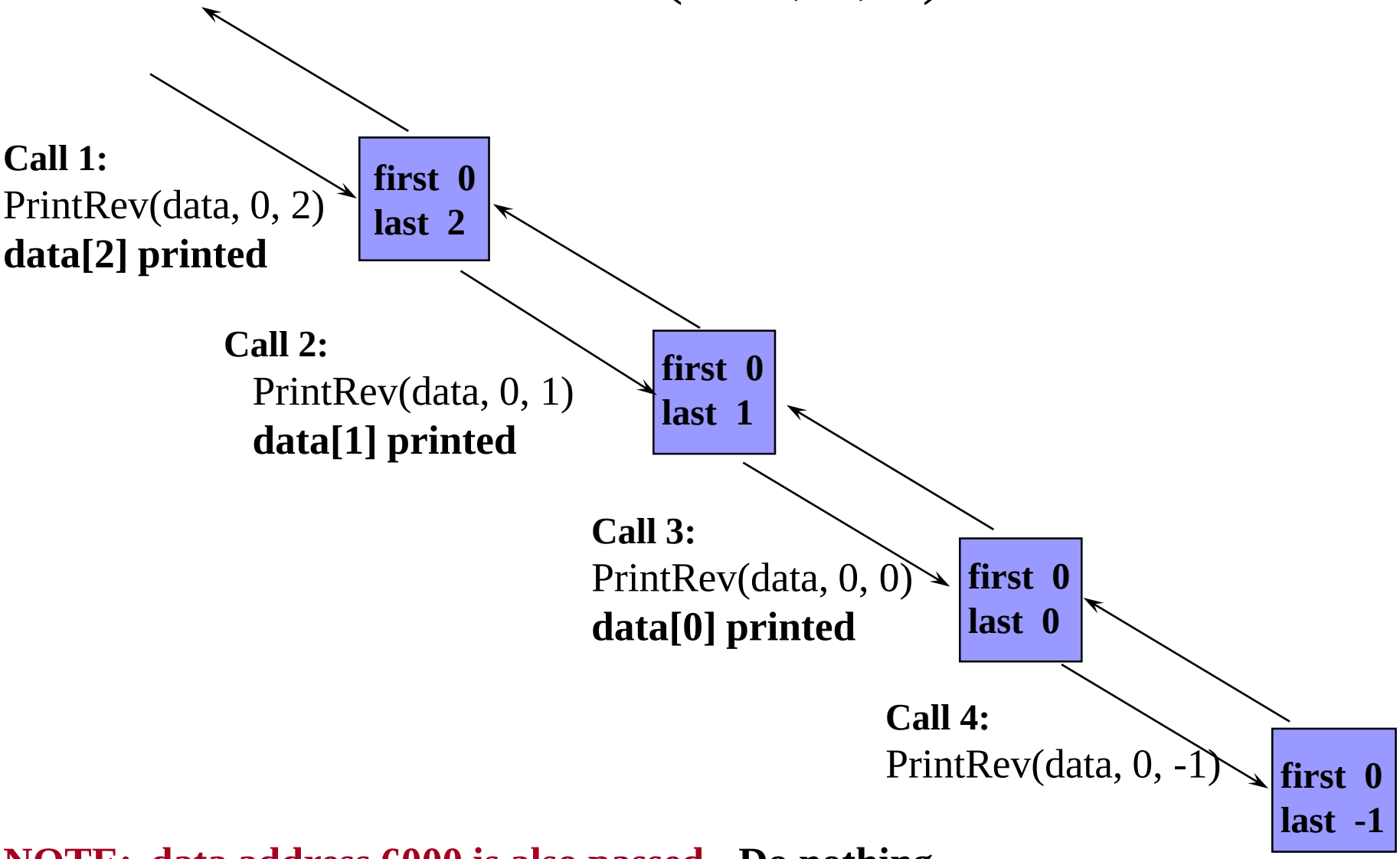
In the general case, we could print either the first element, that is, data[first] or we could print the last element, that is, data[last]

Let's print data[last]: After we print data[last], we still need to print the remaining elements in reverse order.

Using Recursion with Arrays

```
int PrintRev (
const int data [ ], int  first,  int  last )
// Prints items in data [first..last] in reverse order
{
    if  (first  <=  last)      // General case
    {
        cout << data[last]  << "    "; // Print last
        PrintRev(data, first, last - 1); //Print rest
    }
    // Base case is empty else-clause
}
```

PrintRev(data, 0, 2) Trace



NOTE: data address 6000 is also passed **Do nothing**

Why use recursion?

- **These examples could all have been written more easily using iteration.**
- **However, for certain problems the recursive solution is the most natural solution.**
- **This often occurs when structured variables are used.**
- **Remember The iterative solution uses a loop, and the recursive solution uses a selection statement.**

Writing Recursive Functions

- **There must be at least one base case and at least one general (recursive) case--the general case should bring you "closer" to the base case**

- **The arguments(s) in the recursive call cannot all be the same as the formal parameters in the heading, otherwise, infinite recursion would occur**

- **In function `PrintRev()`, the base case occurred when `(first > last)` was true--the general case brought us a step closer to the base case, because in the general case the call was to `PrintRev(data, first, last - 1)`, and the argument (`last - 1`) was closer to first (than `last` was).**

When a function is called...

- A **transfer of control** occurs from the calling block to the code of the function
- It is necessary that there be a return to the correct place in the calling block after the function code is executed
- This correct place is called the **return address**
- When any function is called, the **run-time stack** is used--**activation record** for the function call is placed on the stack

Stack Activation Record

- The activation record (stack frame) contains the return address for this function call, the parameters, local variables, and space for the function's return value (if non-void)
- The activation record for a particular function call is popped off the run-time stack when the final closing brace in the function code is reached, or when a return statement is reached in the function code
- At this time the function's return value, if non-void, is brought back to the calling block return address for use there

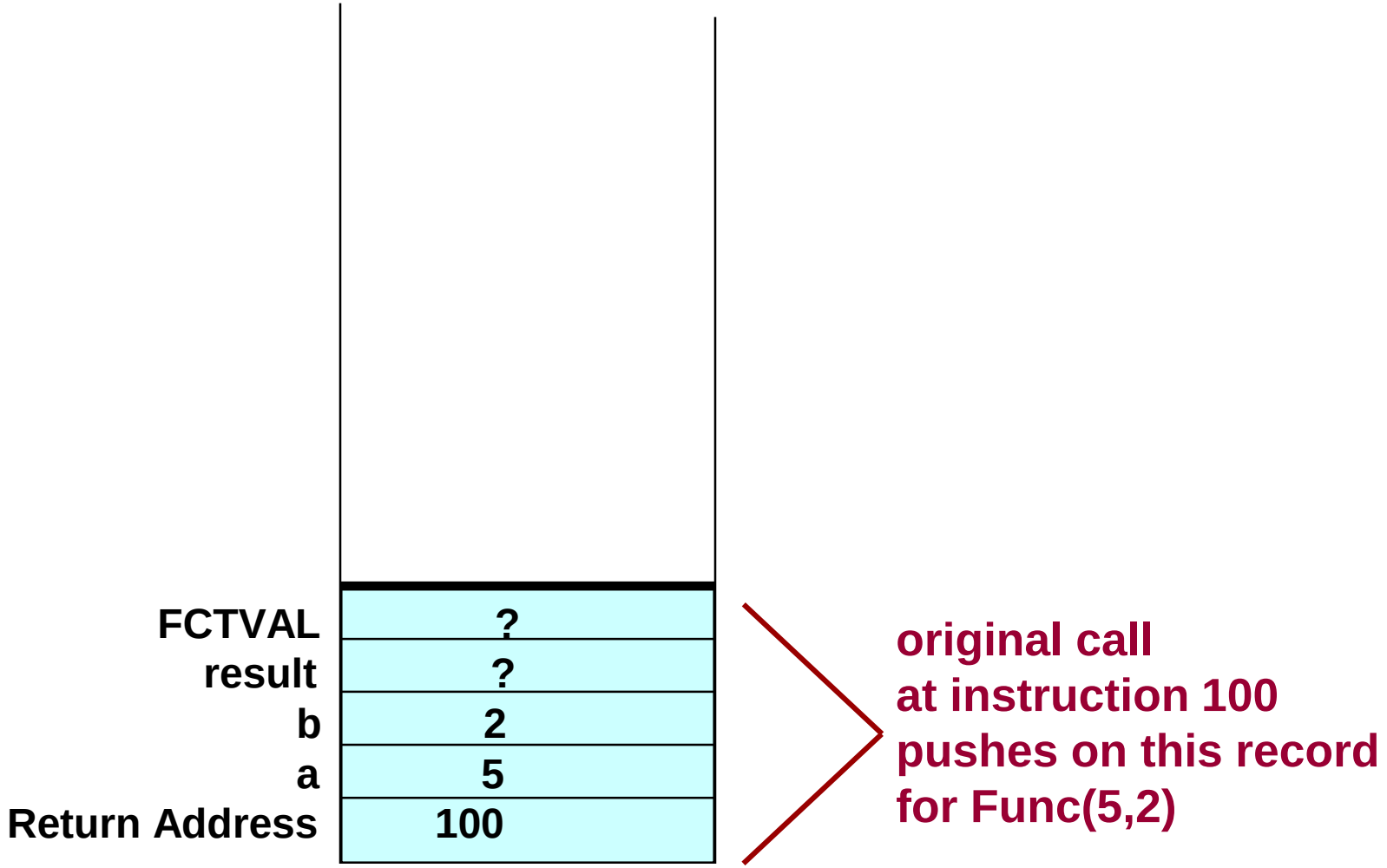
```
// Another recursive function

int Func (int  a, int  b)
{
    int  result;
    if  (b == 0)        // Base case
        result = 0;
    else if  (b > 0) // First general case
        result = a + Func (a, b - 1));
          // Say address location 50
    else                // Second general case
        result = Func (- a, - b);
          // Say address location 70
    return  result;
}
```


25

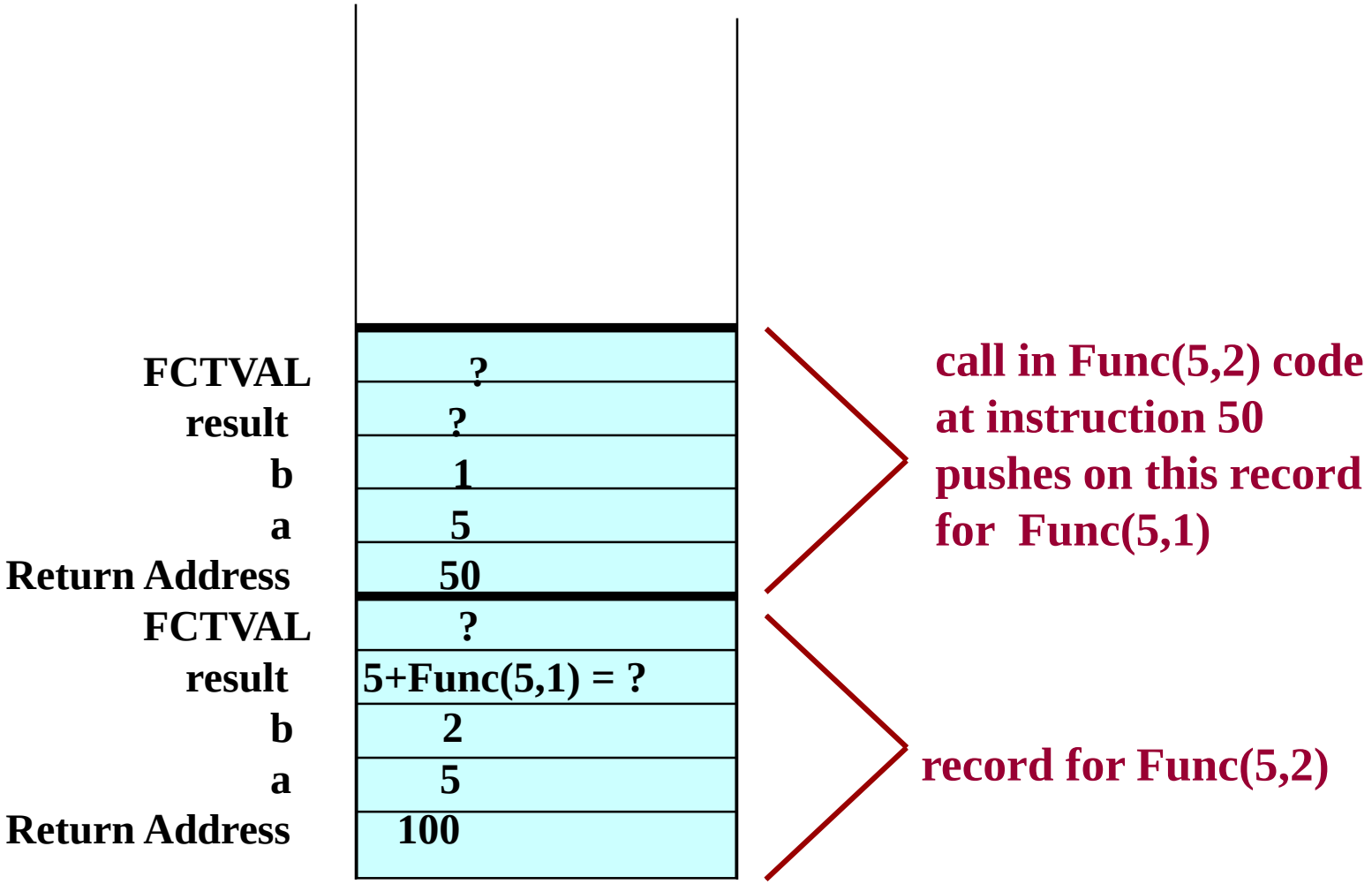
Run-Time Stack Activation Records

```
x = Func(5, 2);
```
// original call at instruction 100



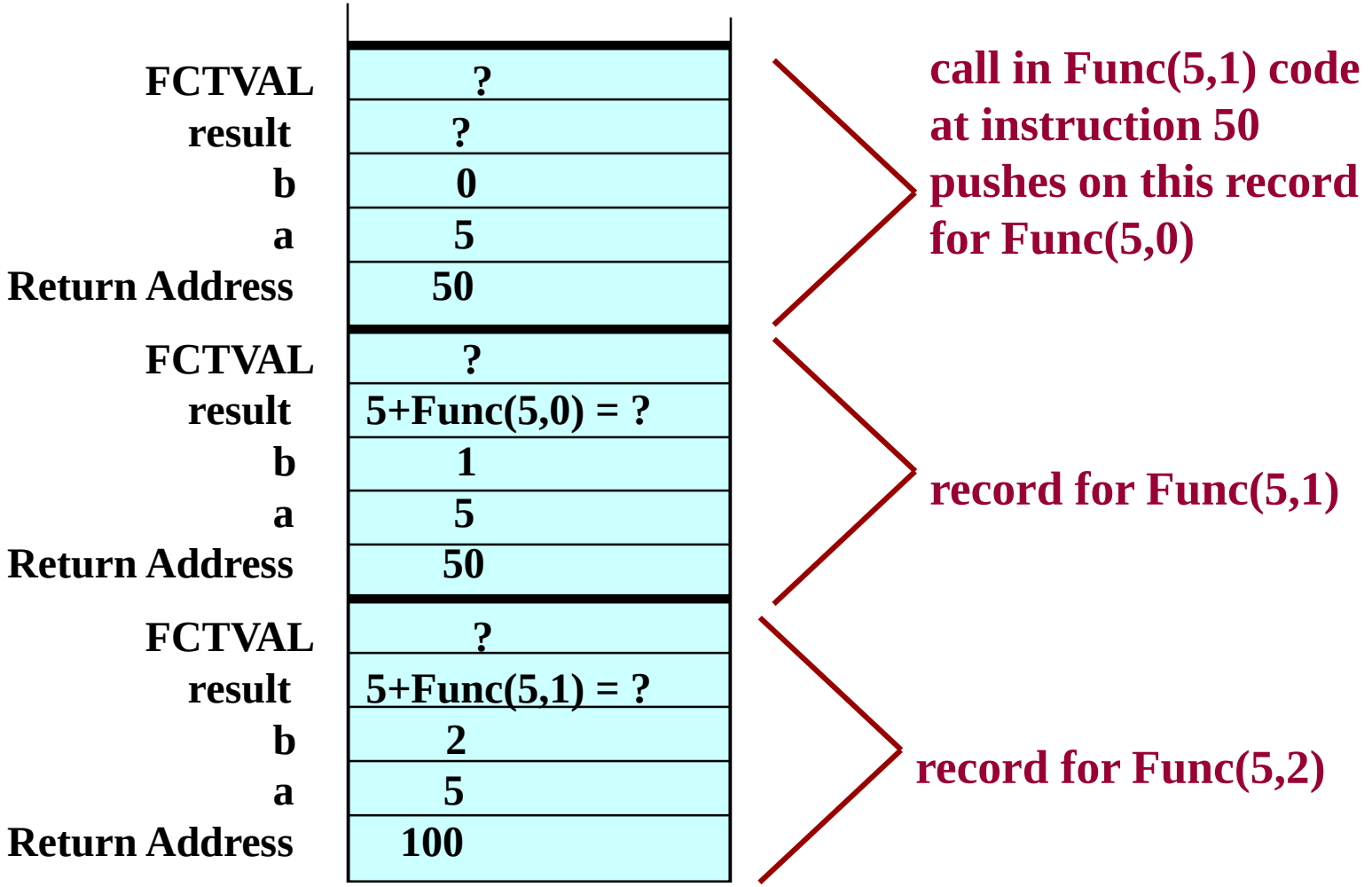
Run-Time Stack Activation Records

```
x = Func(5, 2);
```
// original call at instruction 100



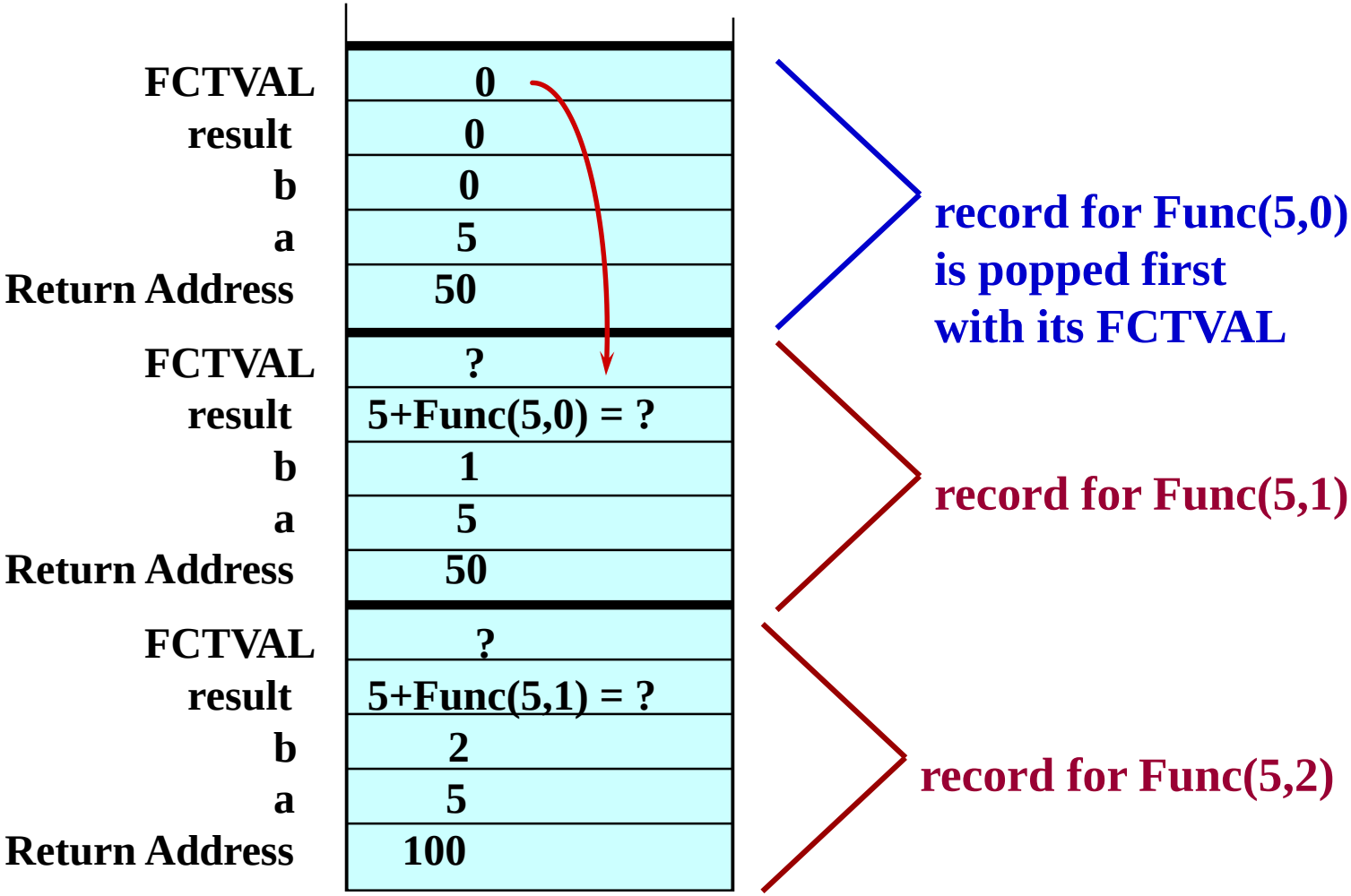
Run-Time Stack Activation Records

```
x = Func(5, 2);
```
// original call at instruction 100



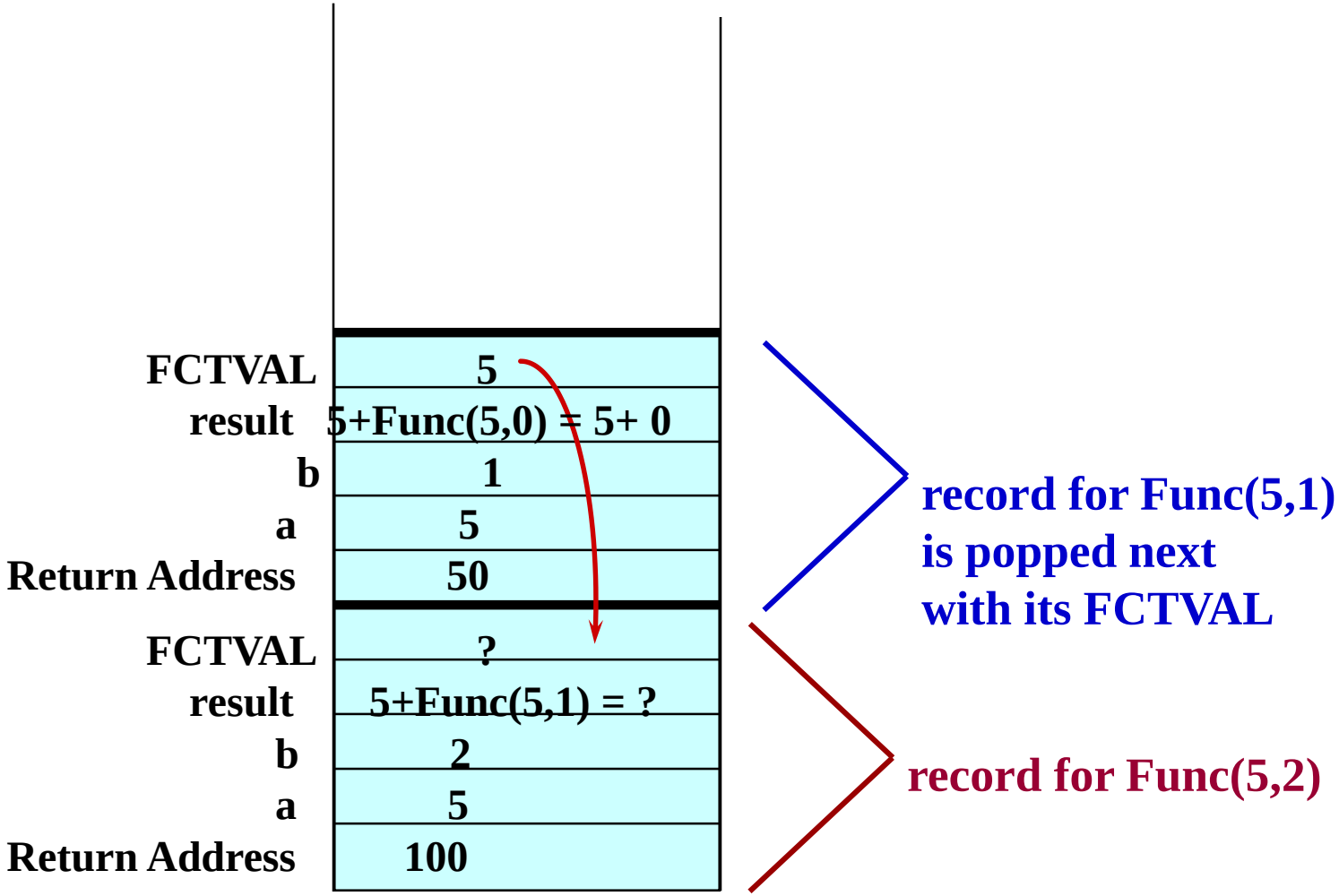
Run-Time Stack Activation Records

```
x = Func(5, 2);
```
// original call at instruction 100



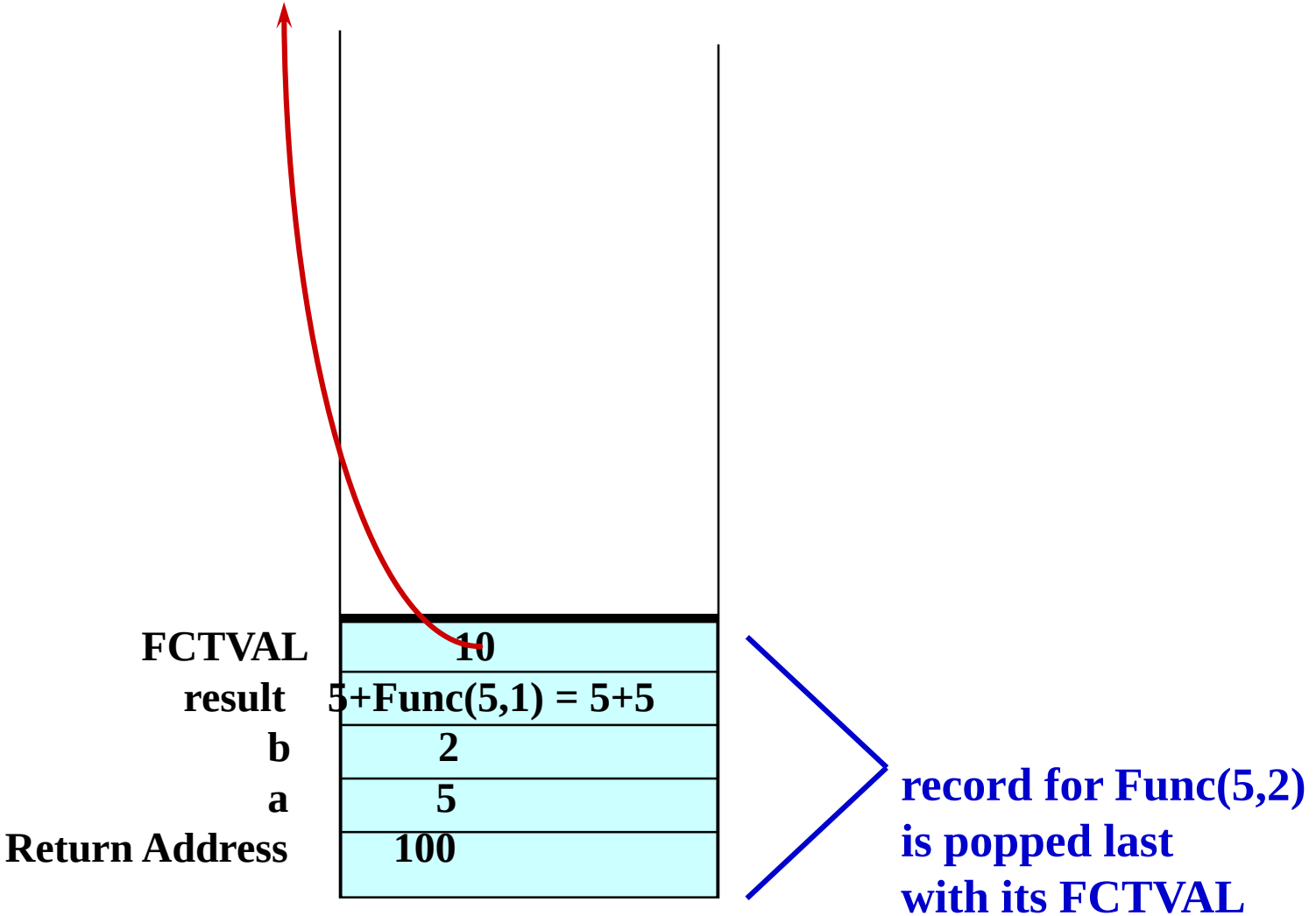
Run-Time Stack Activation Records

```
x = Func(5, 2);
```
// original call at instruction 100



Run-Time Stack Activation Records

```
x = Func(5, 2);
```
// original call at instruction 100



Recitation Assignment 6
Decimal to Base N Conversion

- Write a C++ recursive function that performs conversion from decimal to any number of base *n*, where *n* is in the range 2-10 or *n* = 16 (hexadecimal). The algorithm for this conversion is to repeatedly divide the decimal number by *n*, until it is 0. Each division produces a remainder, which becomes a digit in the converted number. For example, if we want convert the decimal 14 number to a number of base 3, we would apply the following steps:

14/3 = 4	remainder 2
4/3 = 1	remainder 1
1/3 = 0	remainder 1

Recitation Assignment 6
Decimal to Base N Conversion

- Thus, the result is $(112)_3$. The only problem with this algorithm is that the first division generates the low-order digit, the next division generates the second-order digit, and so on, until the last division produces the high order digit. Thus, if we output the digits as they are generated, they will be in reverse order. Thus, you should use recursion to reverse the order of output.
  - Implement a recursive C++ function that performs the conversion from decimal to any number of base *n* that has the following header definition:

    ```
    string DecToBaseN(int num, int base);
    ```

  - Where, *num* is the passed decimal number and *base* is the base of the number to be converted to in the range 2-10 or 16. The function returns the converted number as a string of digits. In the case of base 16, the function returns the converted number as a string of digits and letters (A-F which represent the values 10-15, respectively). For any other illegal base *n*, the function displays the message "Invalid Base: *n*" and returns a null string.

Recitation Assignment 6
Decimal to Base N Conversion

- Example 1:

```
Enter a decimal number:
198
Enter the base of conversion:
16
result of converting the decimal number 198 to base 16: "C6"
```

- Example 2:

```
Enter a decimal number:
235
Enter the base of conversion:
12
Invalid Base: 12
result of converting the decimal number 235 to base 12: ""
```


NJIT
THE EDGE IN KNOWLEDGE