APS105
Winter 2012

Jonathan Deber
jdeber -at- cs -dot- toronto -dot- edu

Lecture 26
March 21, 2012

Today

- Debugging
- Recursion

Debugging

All Software Has Bugs

- Unavoidable
- Techniques to help avoid them, and to help find them

Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it?

Brian Kernighan

Photo # NH 96566-KN (Color) First Computer "Bug", 1947

Grace Hopper's Lab Notebook

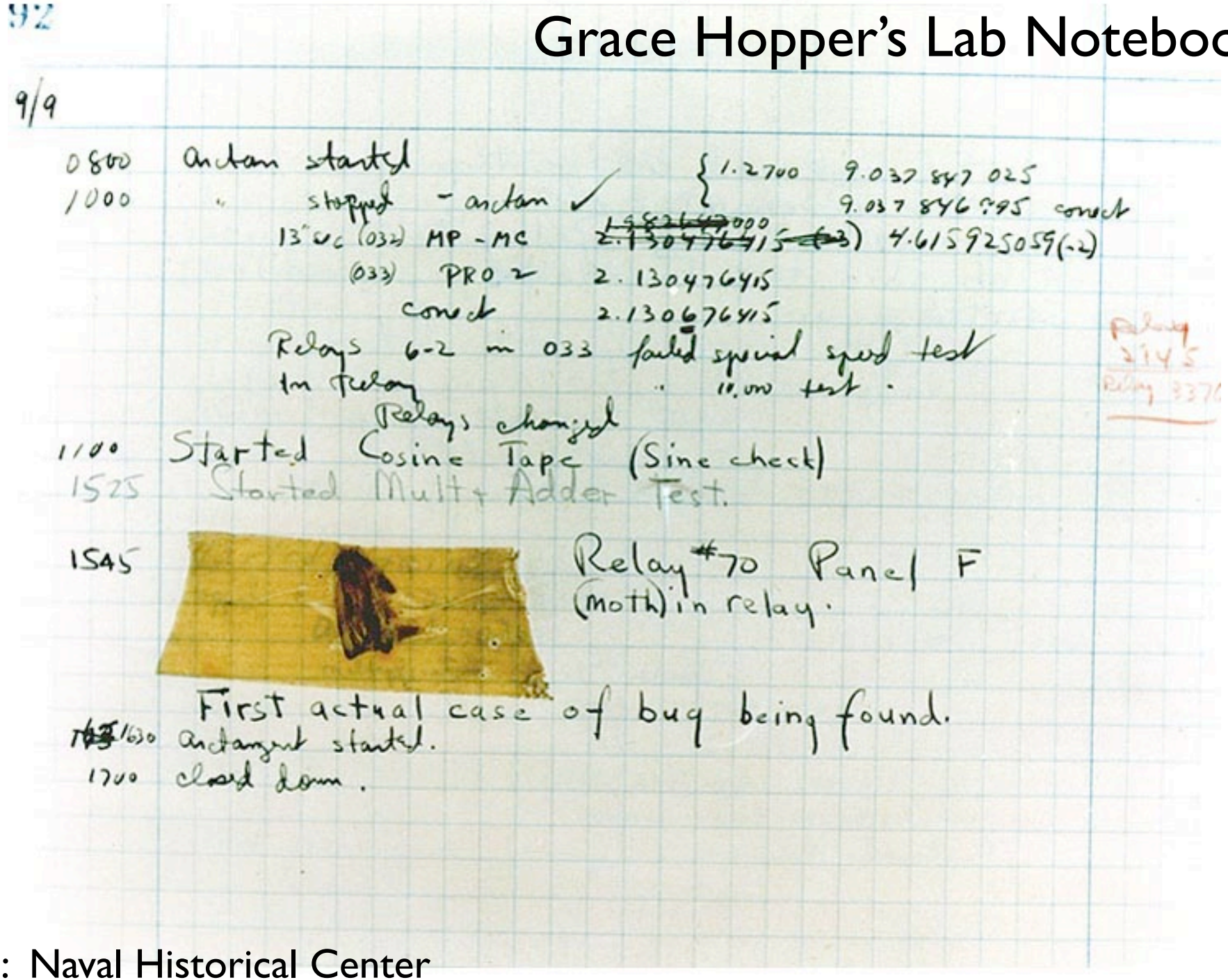


Image: Naval Historical Center

Types of Bugs

- Code that does not do what it's supposed to
- Code that does what it's supposed to, but that isn't the right thing to do
  - May be a bug in the design

Diagnosing Bugs

- Need to be precise

My program doesn't work.

My program crashes.

My program crashes with a Segmentation Fault.

My program crashes with a Segmentation Fault
when given the input 1.

My program crashes with a Segmentation Fault
on line 128 when given the input 1.

My program crashes with a Segmentation Fault on line 128 when given the input 1 because I'm never storing a pointer in `first`.

Fixing Them

- In order to fix a bug, you have to find it
- Finding it is usually the hard part

Diagnosing Bugs

- Manual `printf()` statements
- Debugger

Knowing Where You Crashed

- Often that's enough
- A debugger will tell you which line
- Need to compile with `-g` flag (`gcc -Wall -g ...`)

```
$ gdb square
(gdb) run
```

```
Reason: KERN_INVALID_ADDRESS at address: 0x0000000000000000
0x0000000100000eab in square (n=38745) at square.c:9
9               *p = n;
```

Crash is on this line

printf()

- Add in printf() statements to tell you what's going on

```
int createHistogramRow(char row[], char symbol,
                int value, int maxValue, int maxRowLength)
{
  double percentage = (double)value / maxValue;

  int numSymbols = rint(percentage * maxRowLength);

  printf("Drawing %d out of %d as %d symbols. \n",
            value, maxValue, numSymbols);
  ...
```

```
#define DEBUGGING false
```

```
if (DEBUGGING)
{
  printf("Printing row %d. \n", i);
}
```

Debugger

- "Breakpoints" let you stop your program at a specific line
- Allows you to step through your code
  - Step Over (go to the next line, skipping function calls)
  - Step Into (go to the next line, entering function calls)
  - Step Out (go back to the caller)
- Displays values of variables

- gdb is gcc's debugger, they're also built-in to IDEs

The first 90% of the code accounts for the first 90% of the development time.

The remaining 10% of the code accounts for the other 90% of the development time.

Tom Cargill

Recursion

In order to understand recursion,
you must first understand recursion.

Anonymous

Recursion

- A problem defined in terms of itself

If I need to put away a pile of books:

Put away the first book

Put away the remaining pile

If I need to climb a flight of stairs:

Climb one stair

Climb the remaining flight of stairs

If I need to calculate n!

Multiply `n * (n - 1)!`

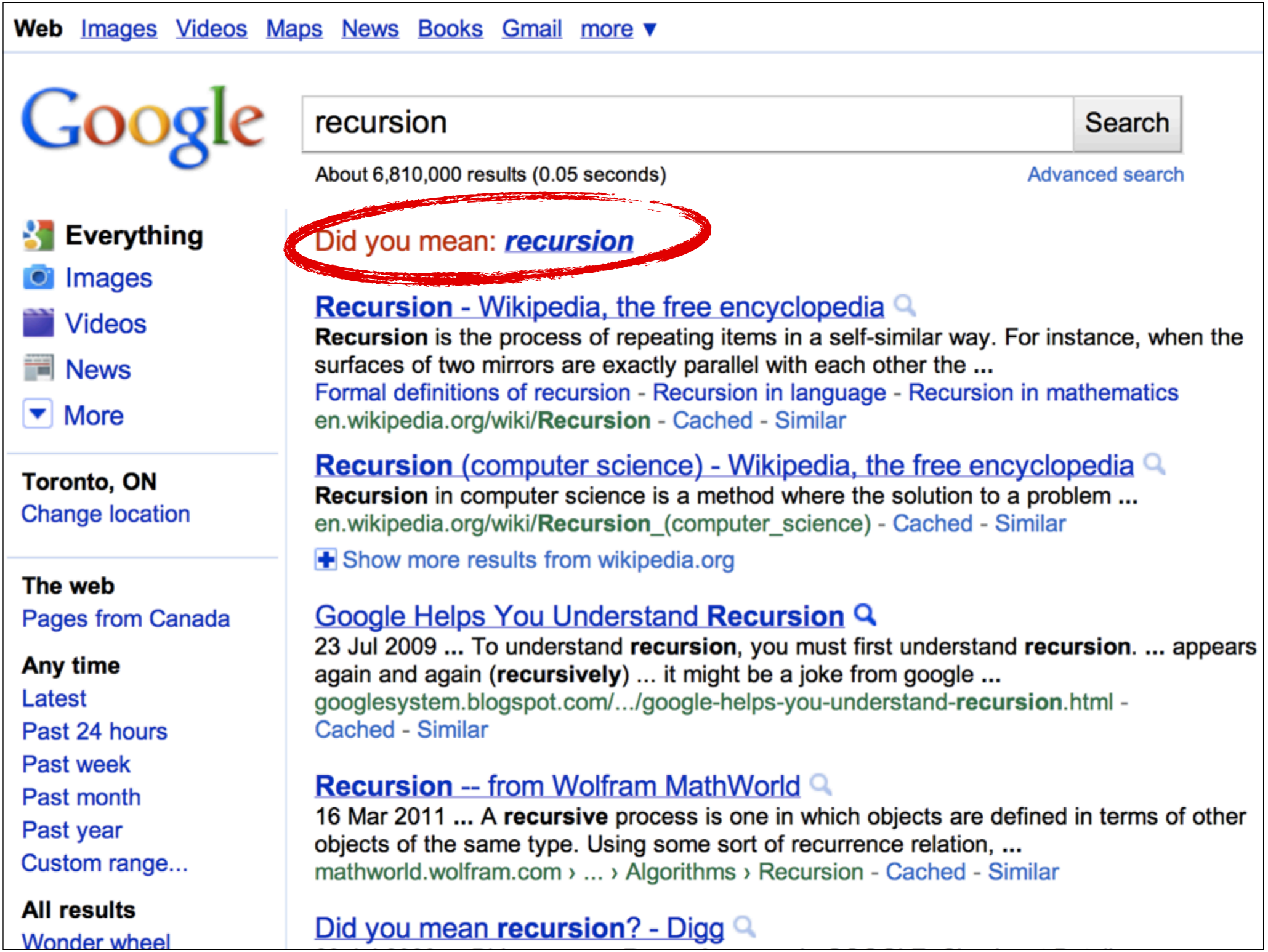
recursion

Advanced searc
Language tools

Google Search | I'm Feeling Lucky

Google.ca offered in: français

Advertising Programs | Business Solutions | About Google | Go to Google.com


Web Images Videos Maps News Books Gmail more ▼
Google
recursion
Search
About 6,810,000 results (0.05 seconds)
Advanced search
Everything
Images
Videos
News
More
Toronto, ON
Change location
The web
Pages from Canada
Any time
Latest
Past 24 hours
Past week
Past month
Past year
Custom range...
All results
Wonder wheel
Did you mean: recursion
Recursion - Wikipedia, the free encyclopedia
Recursion is the process of repeating items in a self-similar way. For instance, when the surfaces of two mirrors are exactly parallel with each other the ...
Formal definitions of recursion - Recursion in language - Recursion in mathematics
en.wikipedia.org/wiki/Recursion - Cached - Similar
Recursion (computer science) - Wikipedia, the free encyclopedia
Recursion in computer science is a method where the solution to a problem ...
en.wikipedia.org/wiki/Recursion_(computer_science) - Cached - Similar
Show more results from wikipedia.org
Google Helps You Understand Recursion
23 Jul 2009 ... To understand recursion, you must first understand recursion. ... appears again and again (recursively) ... it might be a joke from google ...
googlesystem.blogspot.com/.../google-helps-you-understand-recursion.html - Cached - Similar
Recursion -- from Wolfram MathWorld
16 Mar 2011 ... A recursive process is one in which objects are defined in terms of other objects of the same type. Using some sort of recurrence relation, ...
mathworld.wolfram.com › ... › Algorithms › Recursion - Cached - Similar
Did you mean recursion? - Digg

Recursion

- A problem defined in terms of itself
- No more (or less) powerful than loops (iteration)

If I need to put away a pile of books:

Put away the first book

Put away the remaining pile

Put away the books one at a time

If I need to climb a flight of stairs:

Climb one stair

Climb the remaining flight of stairs

Climb the stairs one at a time

If I need to calculate n!

Multiply `n * (n - 1)!`

Start at 1 and multiply it by each number up to n

If I need to put away a pile of books:

Put away the first book

Put away the remaining pile

If I need to climb a flight of stairs:

Climb one stair

Climb the remaining flight of stairs

If I need to calculate n!

Multiply `n * (n - 1)!`

If I need to put away a pile of books:

Put away the first book

Put away the remaining pile

What happens when I run out of books?

If I need to climb a flight of stairs:

Climb one stair

Climb the remaining flight of stairs

What happens when I get to the top?

If I need to calculate n!

Multiply `n * (n - 1)!`

When do I stop?

Recursion

- A problem defined in terms of itself
- No more (or less) powerful than loops (iteration)
- All recursive problems must have a "base case", which is a trivial version of the problem

If I need to put away a pile of books:

If there are no more books, we're done

Otherwise

Put away the first book

Put away the remaining pile

If I need to climb a flight of stairs:

If we're at the top, we're done

Otherwise

Climb one stair

Climb the remaining flight of stairs

If I need to calculate n!

If n is 1,

n! is 1

Otherwise

Multiply `n * (n - 1)!`

Recursion and Programming

- A recursive function is a function that calls itself
- That's it!
- Provides another way to iterate
- No more (or less) powerful than loops
  - Similar to `for` loops vs. `while` loops
- Why bother? Makes some algorithms simpler

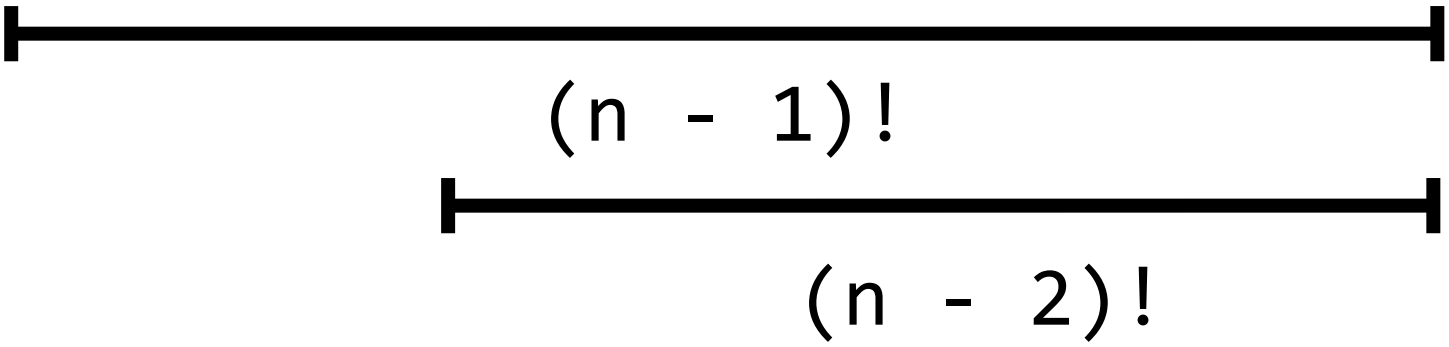
Factorial

n! is n * (n - 1) * (n - 2) * ... * 2 * 1

```
int factorialIterative(int n)
{
    int result = 1;

    for (int i = 1; i <= n; i++)
    {
        result *= i;
    }

    return result;
}
```

Factorial

```
n! is n * (n - 1) * (n - 2) * ... * 2 * 1
```




```
n! is n * (n - 1)!
```

Factorial

n! is n * (n - 1)!

```
int factorialRecursive(int n)
{
  int result;

  result =  n * factorialRecursive(n - 1);

  return result;
}
```

Factorial

```
n! is n * (n - 1)!    if n > 1

n! is 1               if n == 1
```

```
int factorialRecursive(int n)
{
  int result;

  result =  n * factorialRecursive(n - 1);

  return result;
}
```

```
int factorialRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }

  result =  n * factorialRecursive(n - 1);

  return result;
}
```

```
int factorialRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result =  n * factorialRecursive(n - 1);
  }
  return result;
}
```

Recursive Structure

- Steps for writing a recursive algorithm:
  - Figure out the recursive structure of the problem
    - Must be able to break the problem into two parts
    - Each must be either trivial or a smaller version of the same problem
  - Figure out the trivial base case(s)
  - Convert pseudocode to C

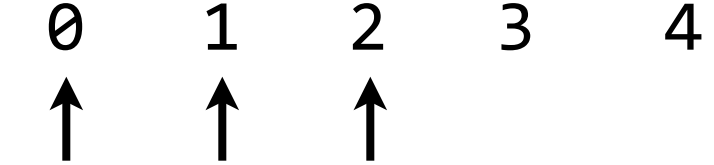
Fibonacci

$$F(n) = F(n-1) + F(n-2)$$

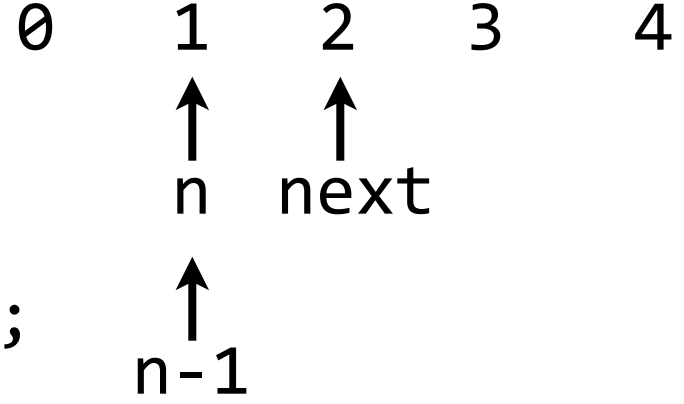
$$F(0) = 0,\ F(1) = 1$$

0, 1, 1, 2, 3, 5, 8, 13, 21,
34, 55, 89, 144, 233, 377, 610,
987, 1597, 2584, 4181, ...

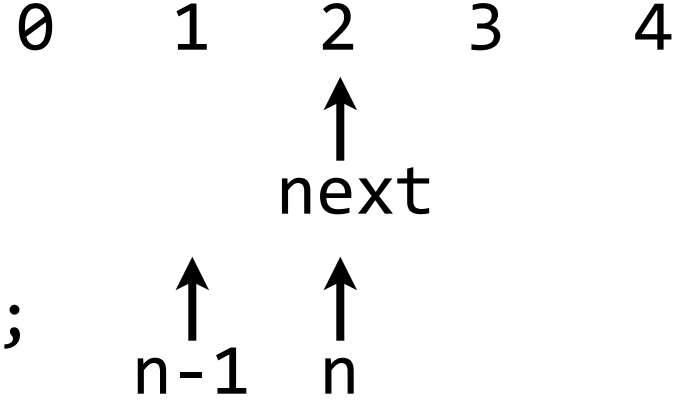
```
int fibonacciIterative(int n)
{
    int fib_n = 1;         // fib(1)
    int fib_nMinus1 = 0; // fib(0)

    for (int i = 2; i <= n; i++)
    {
        int nextTerm = fib_n + fib_nMinus1;

        fib_nMinus1 = fib_n;
        fib_n = nextTerm;
    }

    int result;

    if (n == 0)
    {
        result = fib_nMinus1;
    }
    else
    {
        result = fib_n;
    }

    return result;
}
```



```
int fibonacciIterative(int n)
{
    int fib_n = 1;         // fib(1)
    int fib_nMinus1 = 0; // fib(0)

    for (int i = 2; i <= n; i++)
    {
        int nextTerm = fib_n + fib_nMinus1;

        fib_nMinus1 = fib_n;
        fib_n = nextTerm;
    }

    int result;

    if (n == 0)
    {
        result = fib_nMinus1;
    }
    else
    {
        result = fib_n;
    }

    return result;
}
```



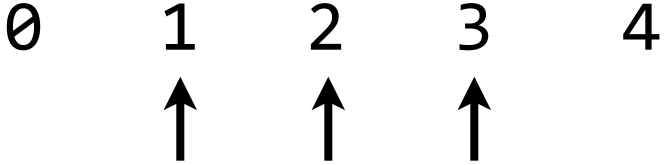
```
int fibonacciIterative(int n)
{
    int fib_n = 1;         // fib(1)
    int fib_nMinus1 = 0; // fib(0)

    for (int i = 2; i <= n; i++)
    {
        int nextTerm = fib_n + fib_nMinus1;

        fib_nMinus1 = fib_n;
        fib_n = nextTerm;
    }

    int result;

    if (n == 0)
    {
        result = fib_nMinus1;
    }
    else
    {
        result = fib_n;
    }

    return result;
}
```



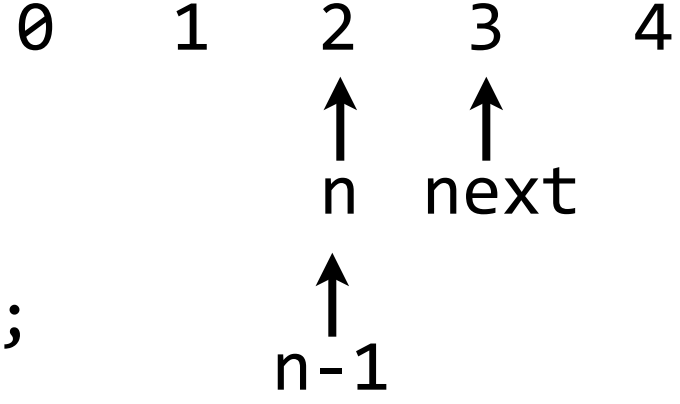
```
int fibonacciIterative(int n)
{
    int fib_n = 1;         // fib(1)
    int fib_nMinus1 = 0; // fib(0)

    for (int i = 2; i <= n; i++)
    {
        int nextTerm = fib_n + fib_nMinus1;

        fib_nMinus1 = fib_n;
        fib_n = nextTerm;
    }

    int result;

    if (n == 0)
    {
        result = fib_nMinus1;
    }
    else
    {
        result = fib_n;
    }

    return result;
}
```

```
                0   1   2   3   4
                    ↑   ↑
                  n-1   n
```

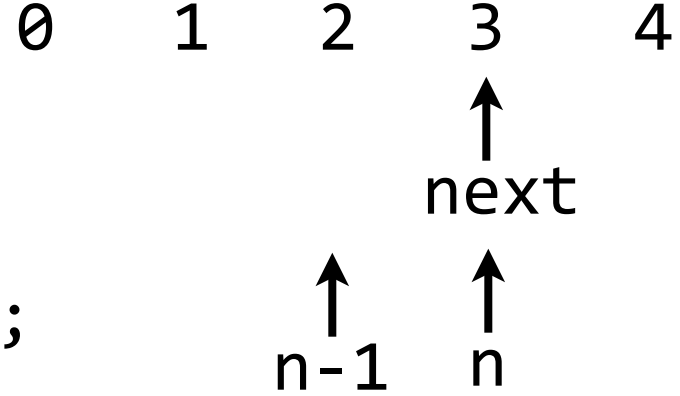
```
int fibonacciIterative(int n)
{
    int fib_n = 1;         // fib(1)
    int fib_nMinus1 = 0; // fib(0)

    for (int i = 2; i <= n; i++)
    {
        int nextTerm = fib_n + fib_nMinus1;

        fib_nMinus1 = fib_n;
        fib_n = nextTerm;
    }

    int result;

    if (n == 0)
    {
        result = fib_nMinus1;
    }
    else
    {
        result = fib_n;
    }

    return result;
}
```



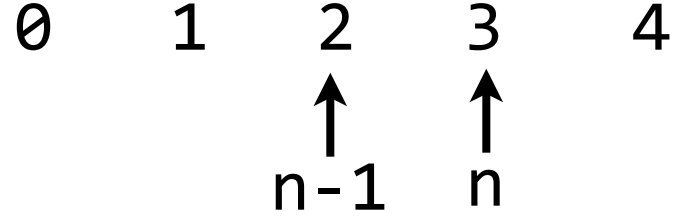
```
int fibonacciIterative(int n)
{
    int fib_n = 1;         // fib(1)
    int fib_nMinus1 = 0; // fib(0)

    for (int i = 2; i <= n; i++)
    {
        int nextTerm = fib_n + fib_nMinus1;

        fib_nMinus1 = fib_n;
        fib_n = nextTerm;
    }

    int result;

    if (n == 0)
    {
        result = fib_nMinus1;
    }
    else
    {
        result = fib_n;
    }

    return result;
}
```



```
int fibonacciIterative(int n)
{
    int fib_n = 1;         // fib(1)
    int fib_nMinus1 = 0; // fib(0)

    for (int i = 2; i <= n; i++)
    {
        int nextTerm = fib_n + fib_nMinus1;

        fib_nMinus1 = fib_n;
        fib_n = nextTerm;
    }

    int result;

    if (n == 0)
    {
        result = fib_nMinus1;
    }
    else
    {
        result = fib_n;
    }

    return result;
}
```



```
int fibonacciIterative(int n)
{
    int fib_n = 1;         // fib(1)
    int fib_nMinus1 = 0; // fib(0)

    for (int i = 2; i <= n; i++)
    {
        int nextTerm = fib_n + fib_nMinus1;

        fib_nMinus1 = fib_n;
        fib_n = nextTerm;
    }

    int result;

    if (n == 0)
    {
        result = fib_nMinus1;
    }
    else
    {
        result = fib_n;
    }

    return result;
}
```



```
int fibonacciRecursive(int n)
{
    int result;

    if (n == 0)
    {
        result = 0;
    }
    else if (n == 1)
    {
        result = 1;
    }
    else
    {
        result = fibonacciRecursive(n - 1)
                        + fibonacciRecursive(n - 2);
    }

    return result;
}
```

```
int fibonacciIterative(int n)
{
  int fib_n = 1;        // fib(1)
  int fib_nMinus1 = 0; // fib(0)

  for (int i = 2; i <= n; i++)
  {
    int nextTerm = fib_n + fib_nMinus1;

    fib_nMinus1 = fib_n;
    fib_n = nextTerm;
  }

  int result;

  if (n == 0)
  {
    result = fib_nMinus1;
  }
  else
  {
    result = fib_n;
  }

  return result;
}
```

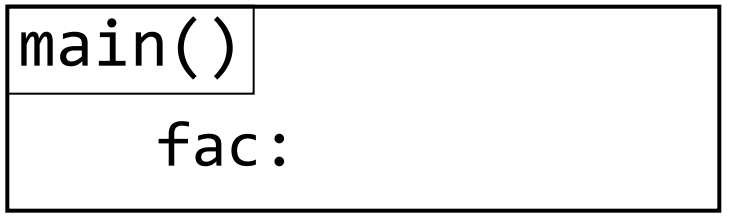
```
int fibRecursive(int n)
{
  int result;

  if (n == 0)
  {
    result = 0;
  }
  else if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = fibRecursive(n - 1)
             + fibRecursive(n - 2);
  }

  return result;
}
```

Tracing Functions

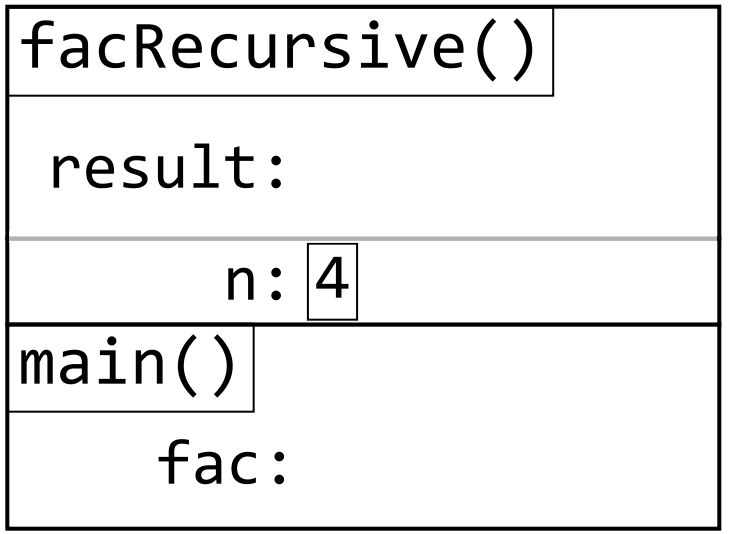
- A key to understanding recursion is tracing the function calls

```
int factorialRecursive(int n)
{
   int result;

   if (n == 1)
   {
      result = 1;
   }
   else
   {
      result = n * factorialRecursive(n - 1);
   }

   return result;
}
```

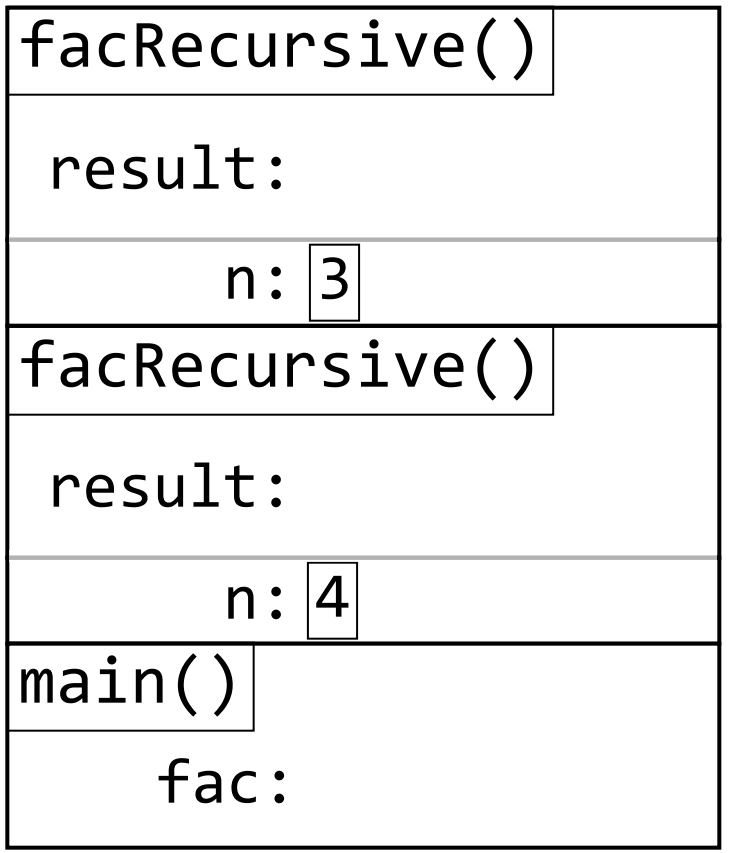
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



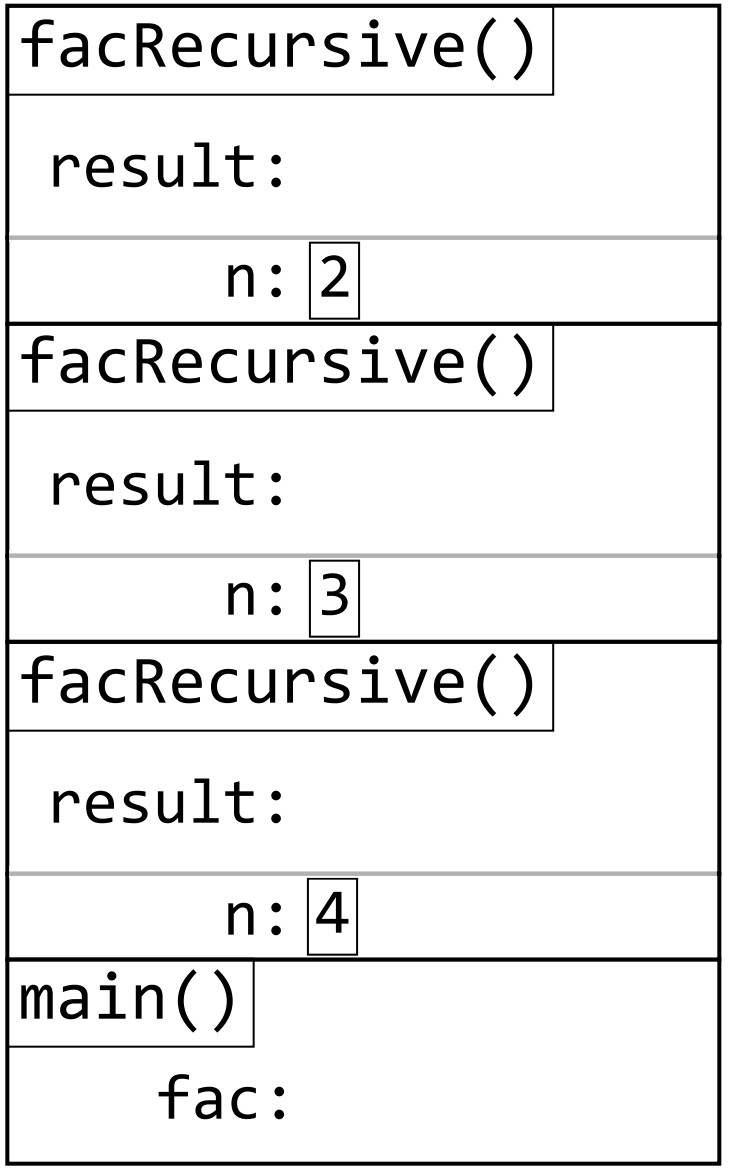
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



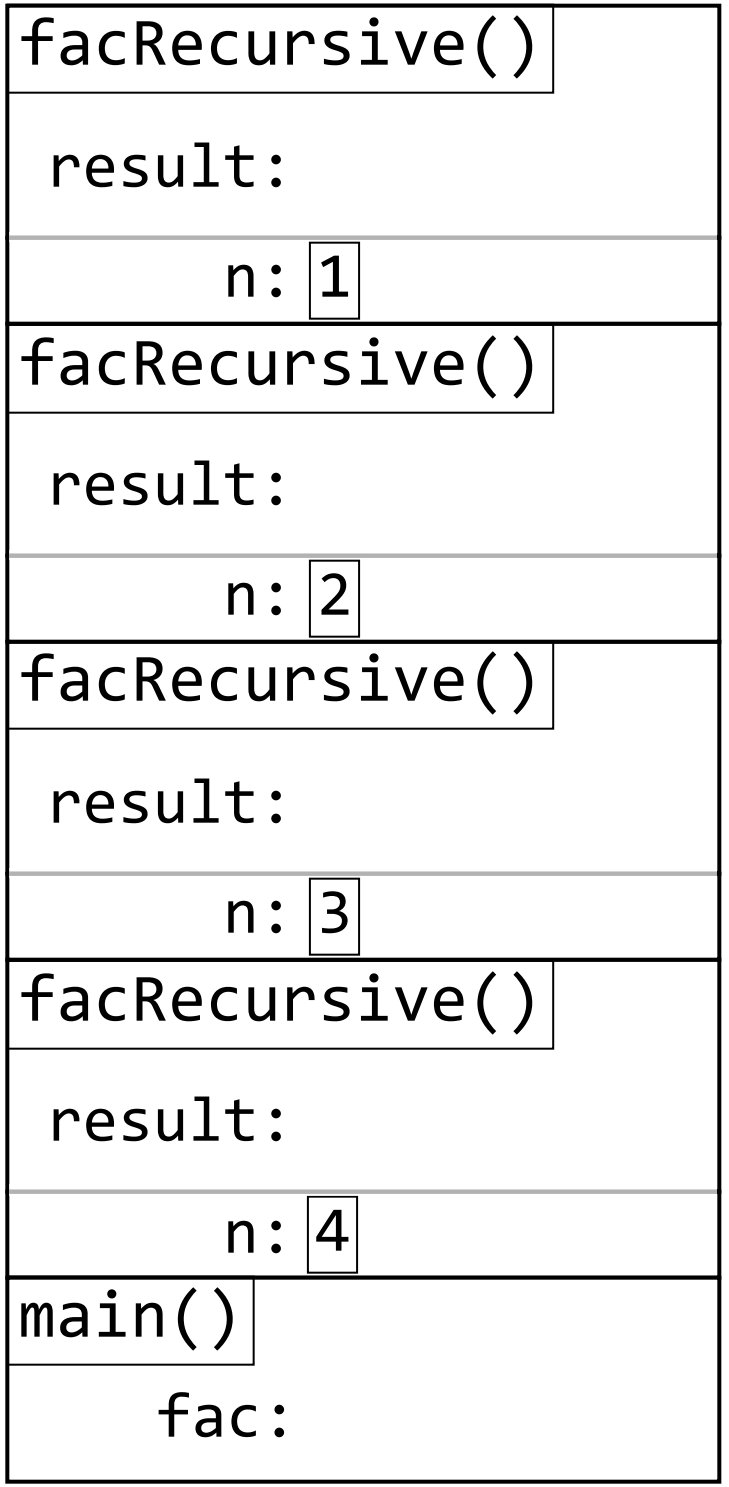
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



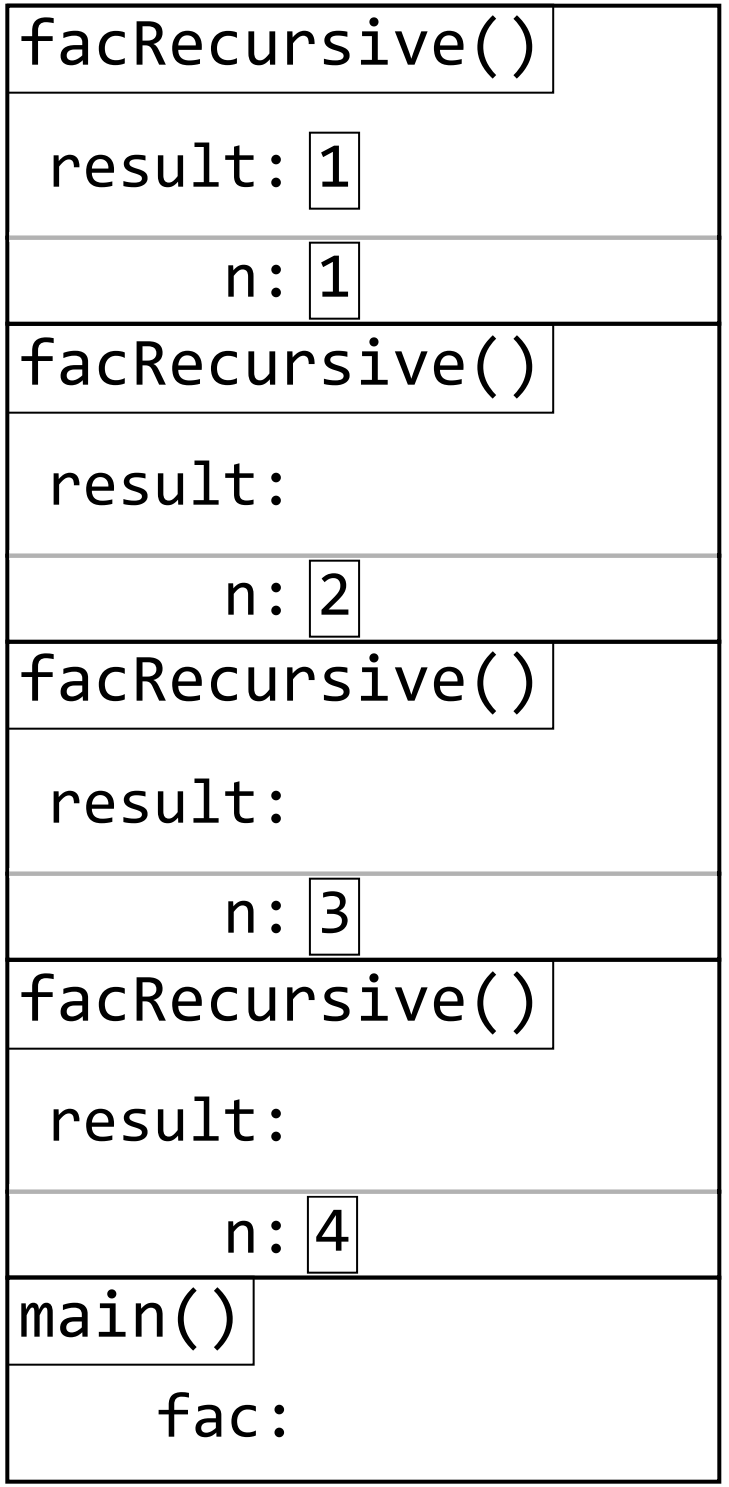
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



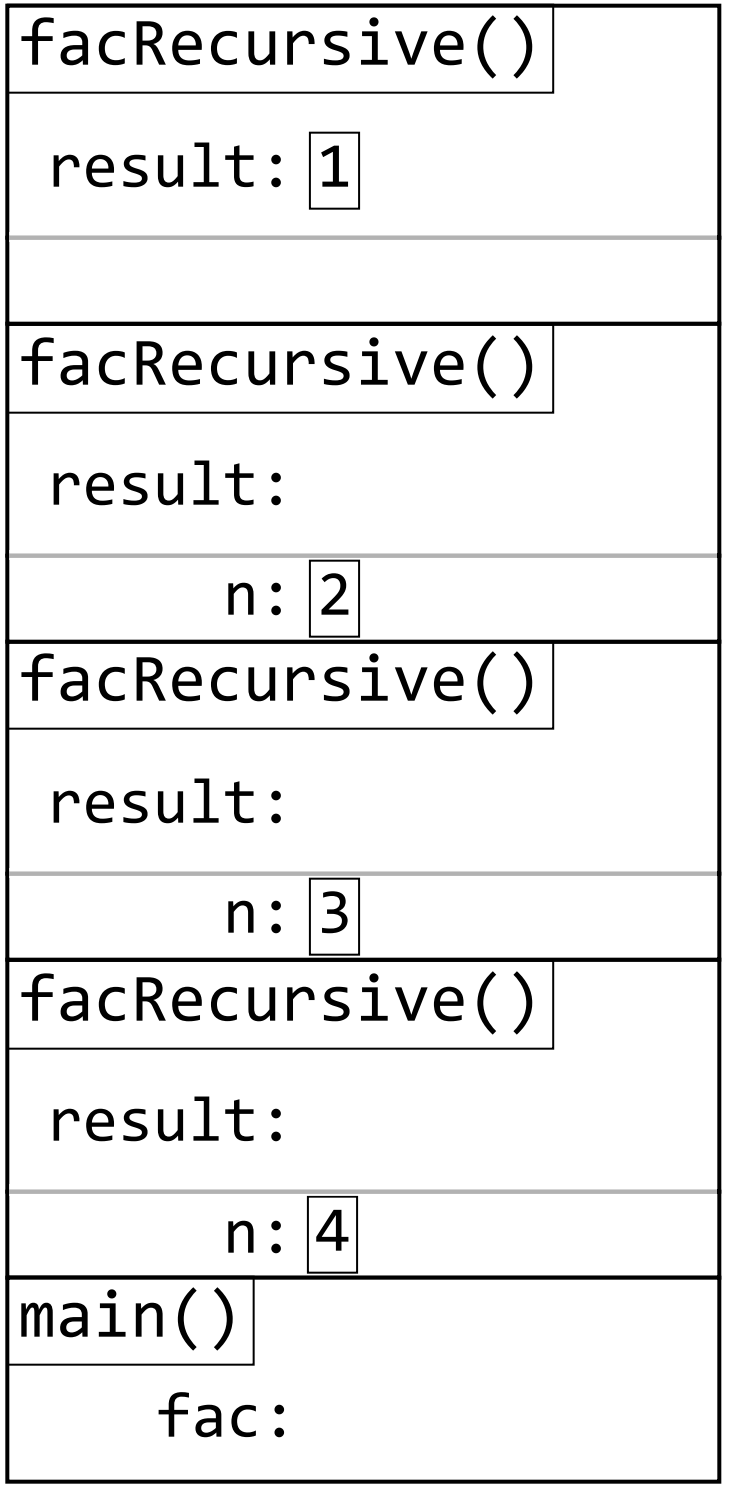
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



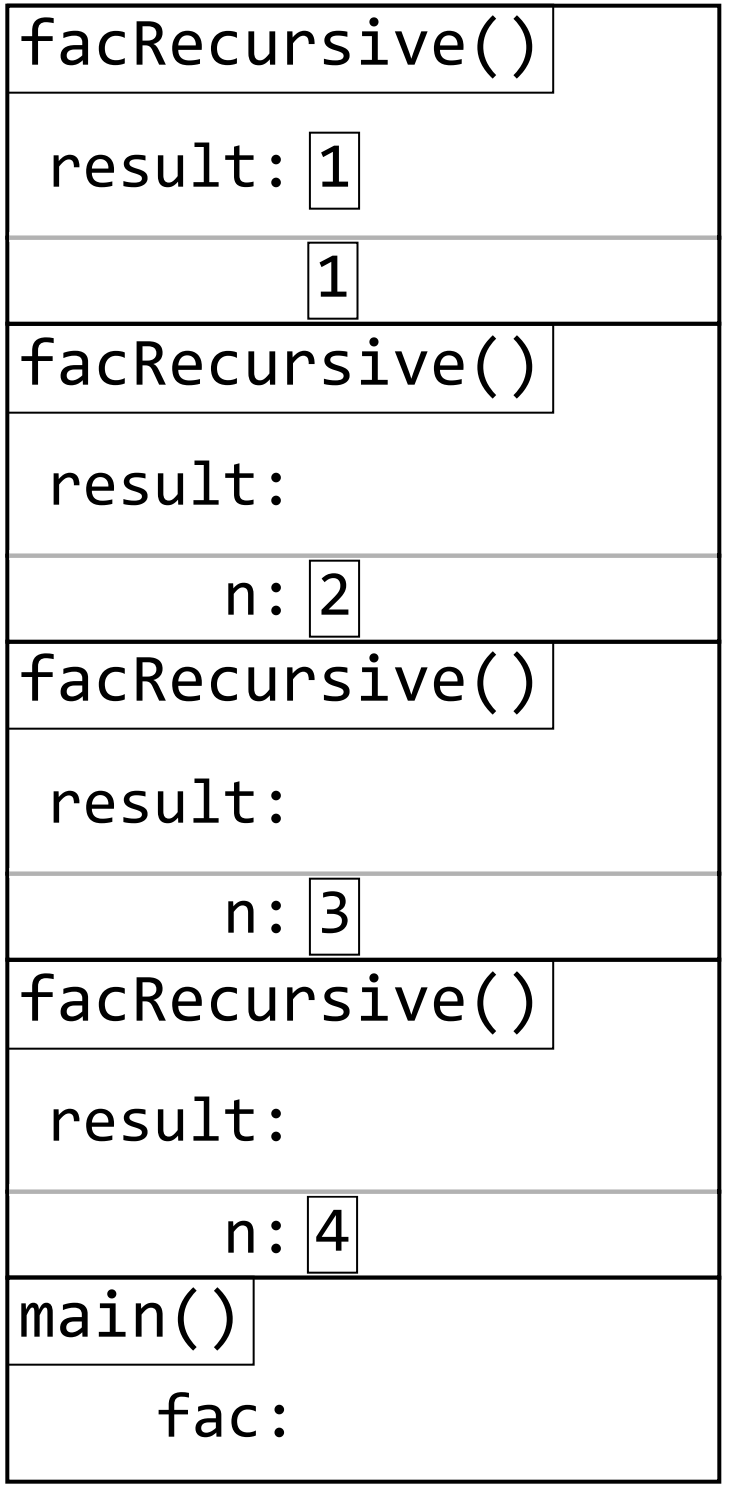
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



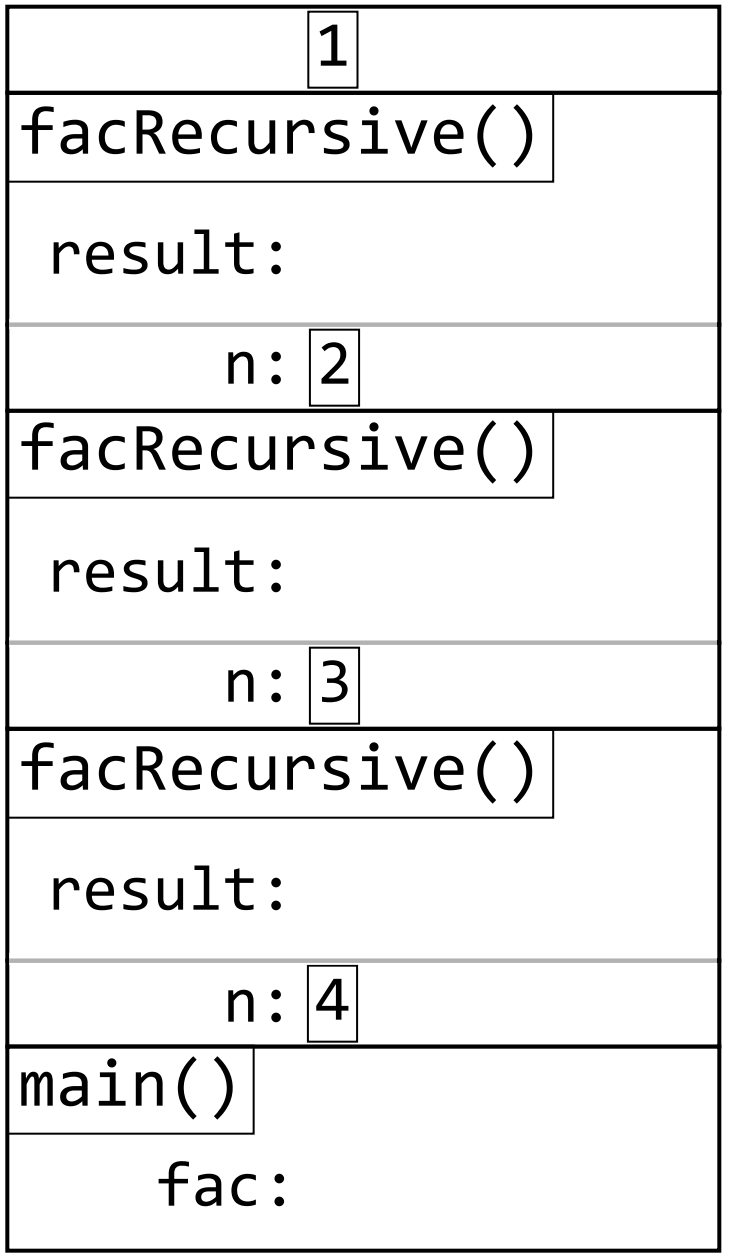
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



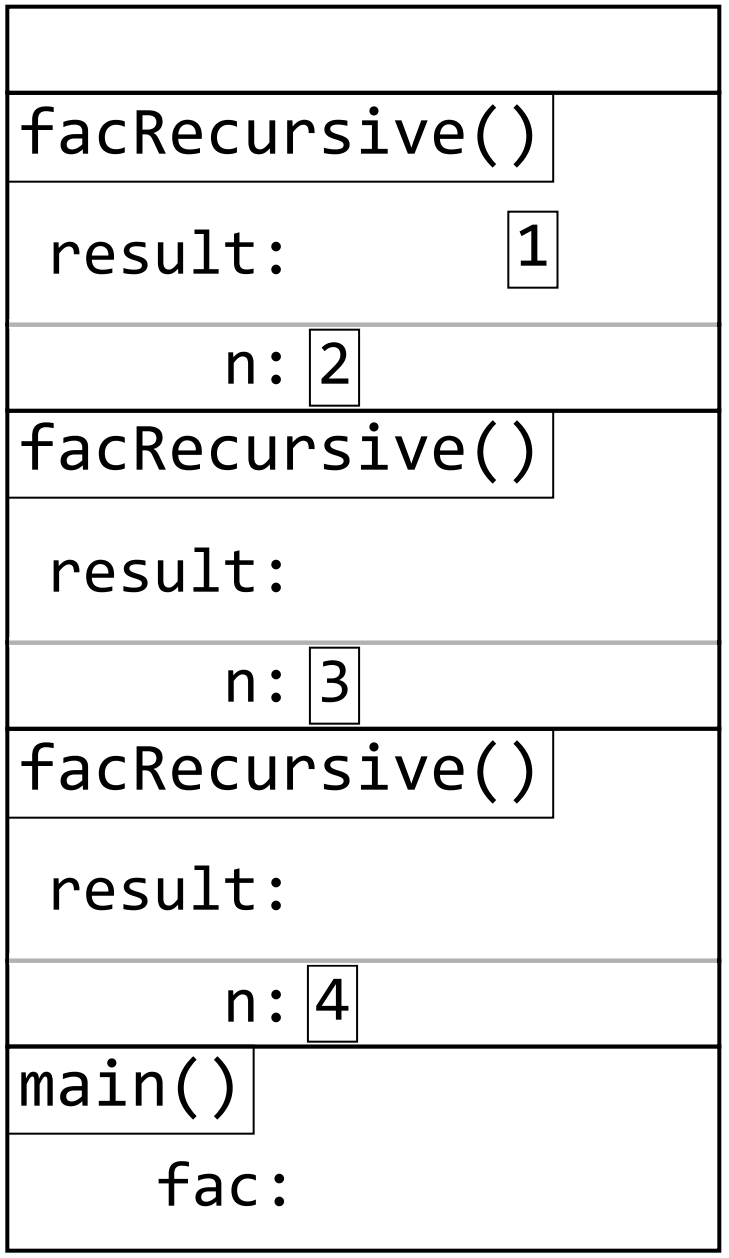
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



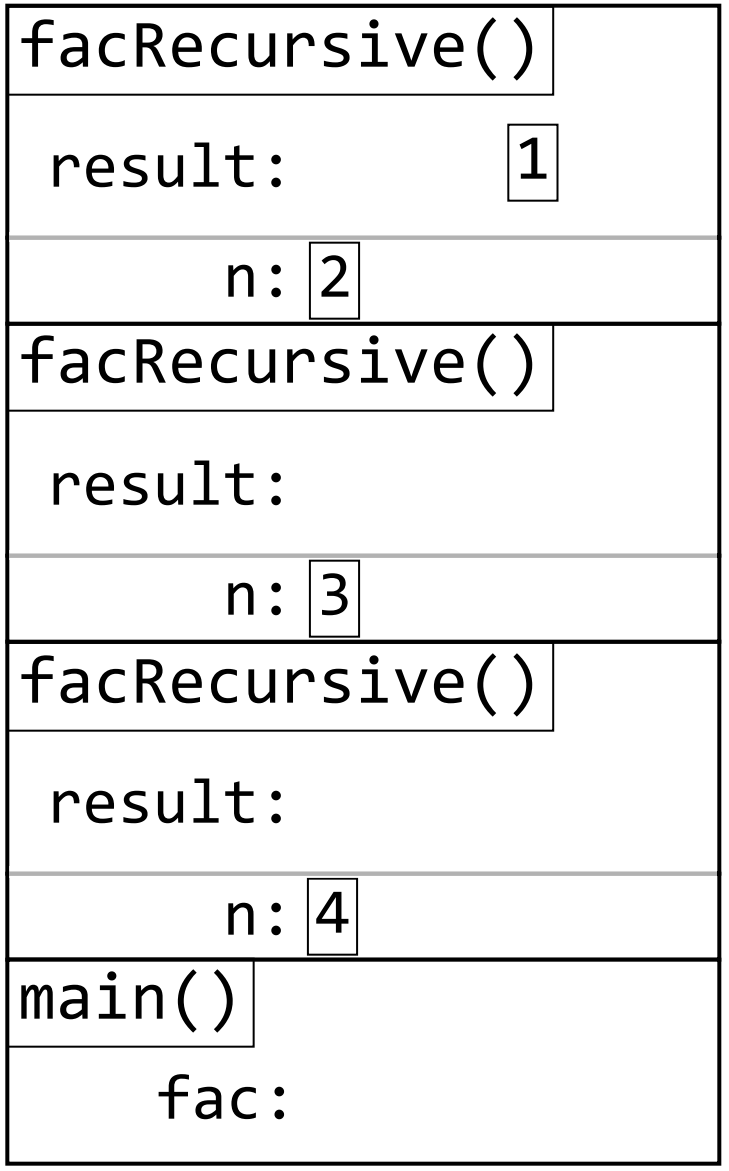
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



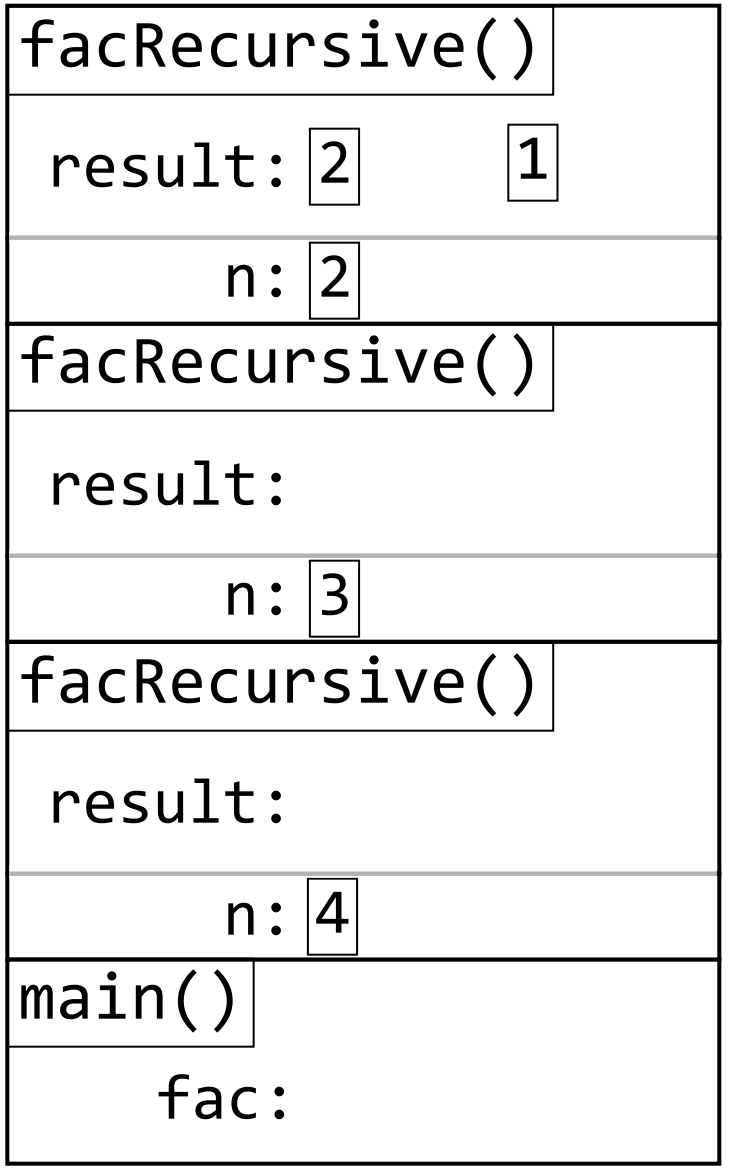
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



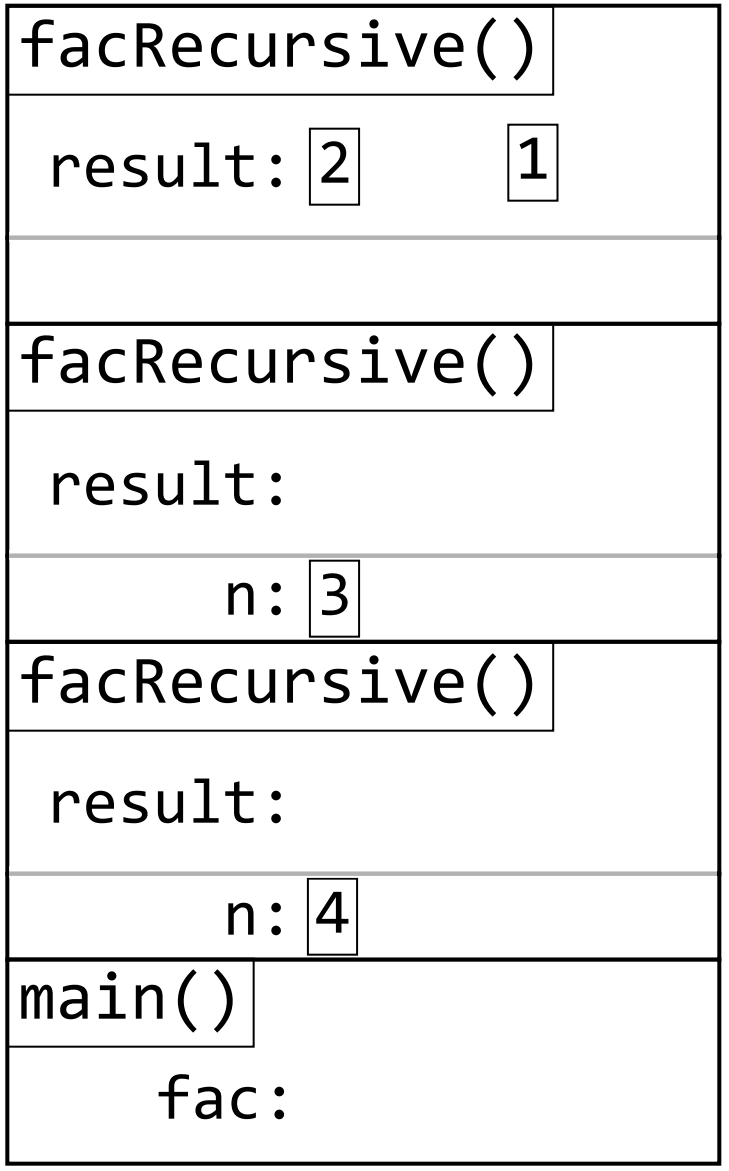
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



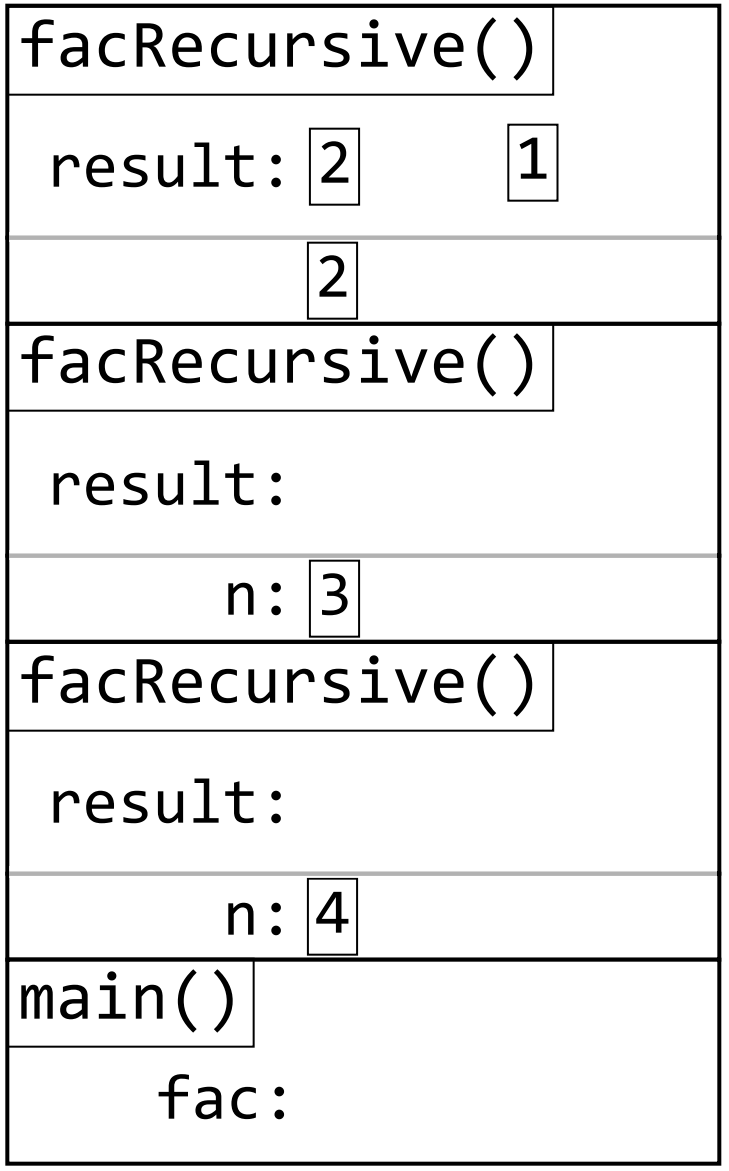
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



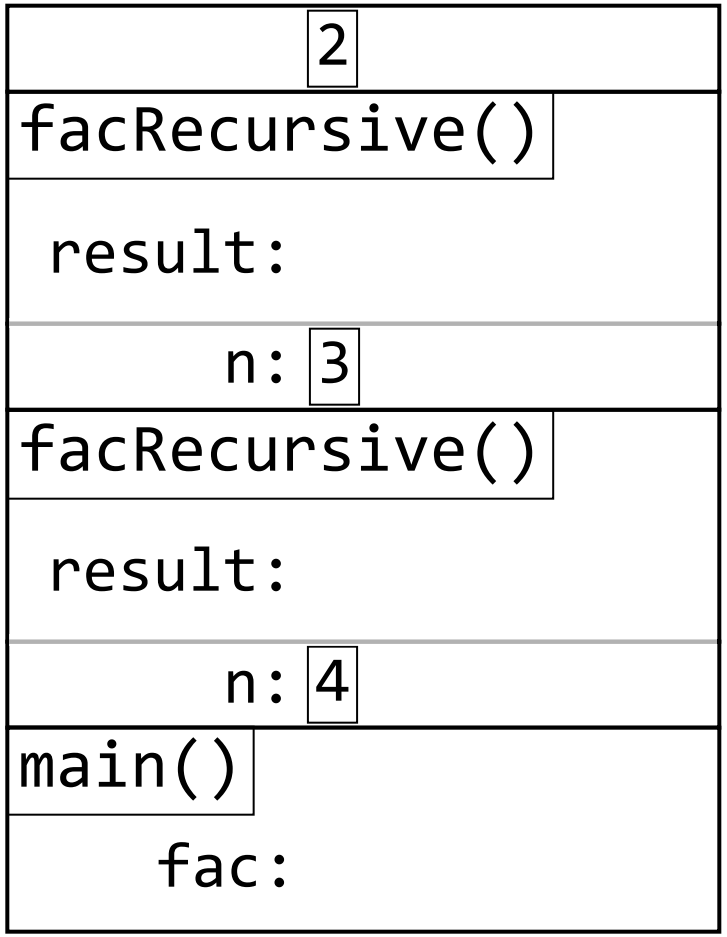
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



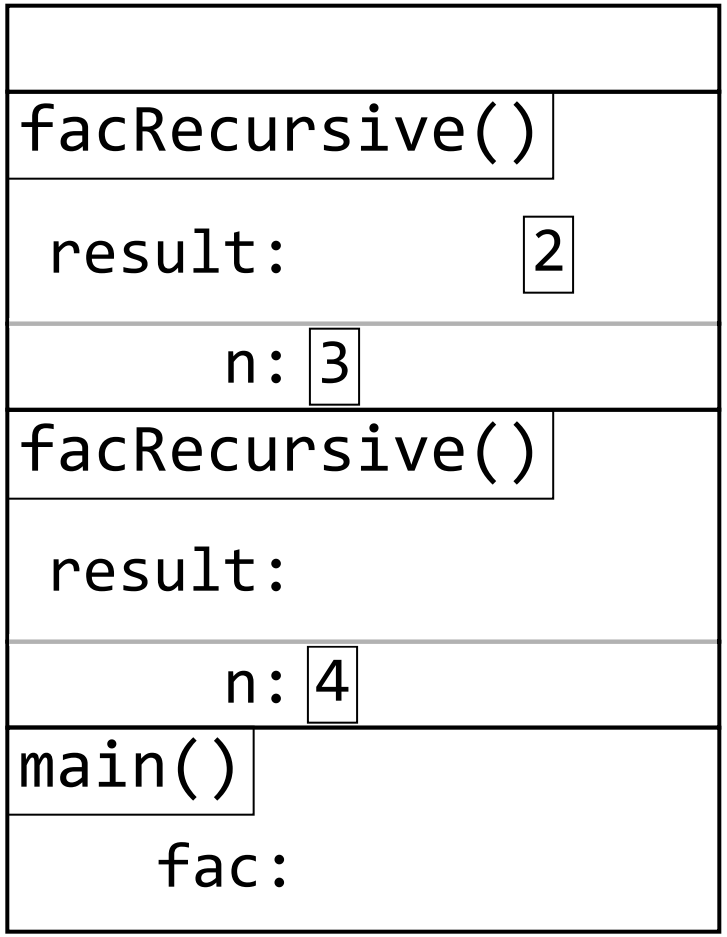
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



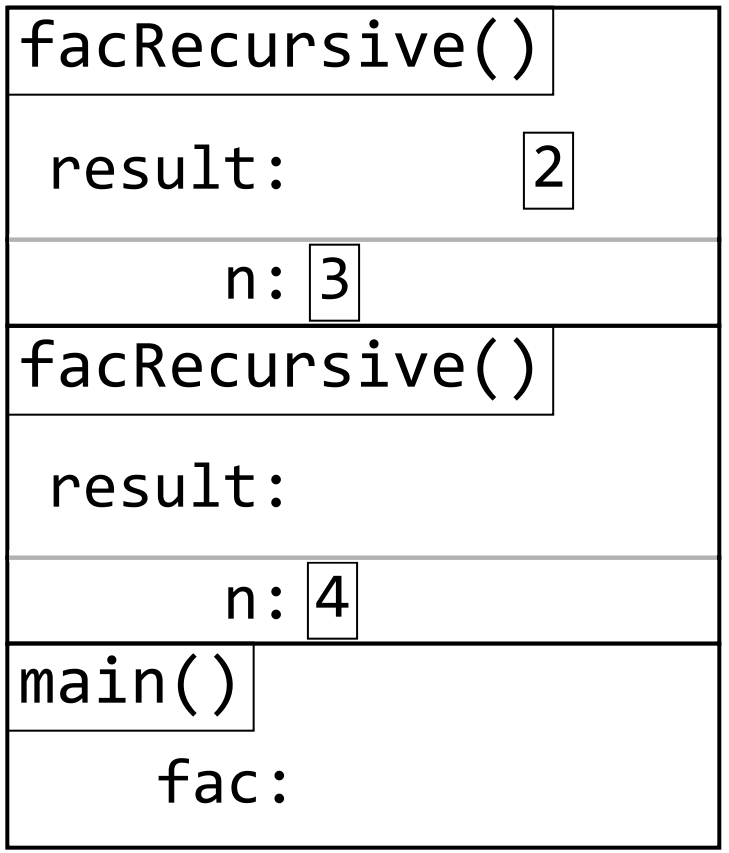
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



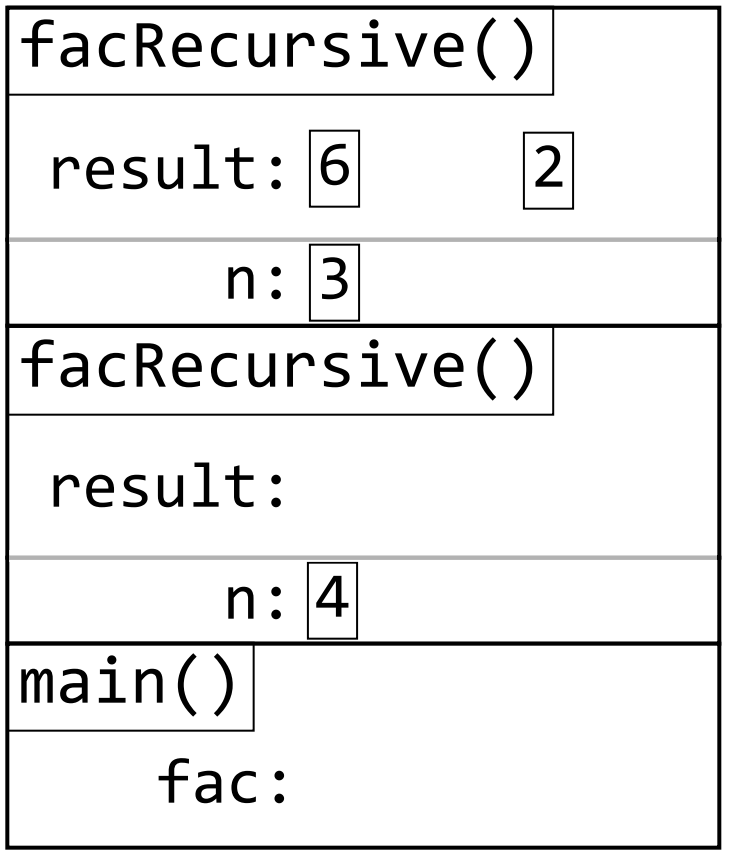
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



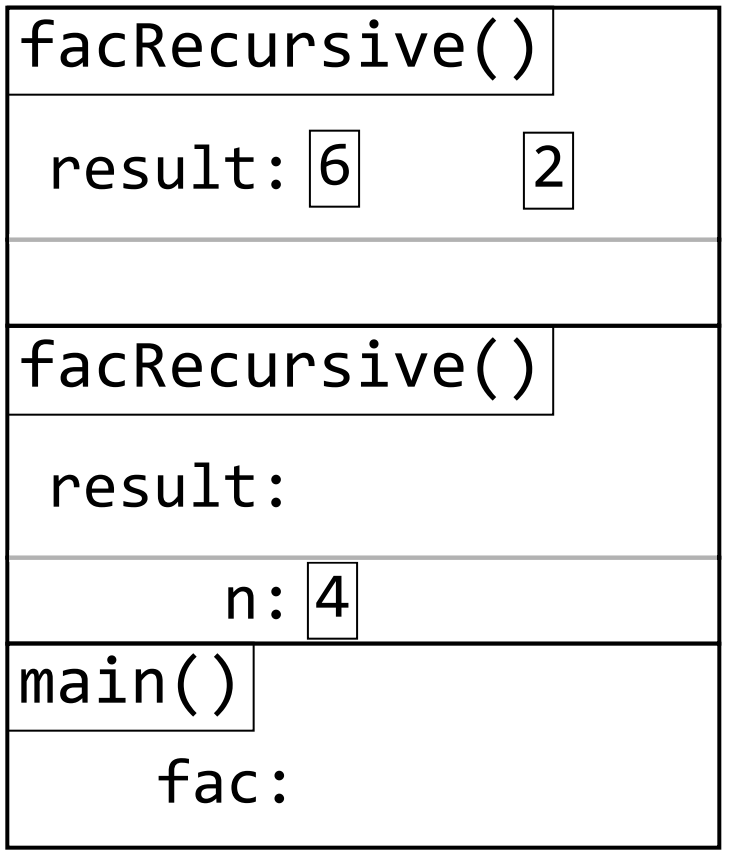
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



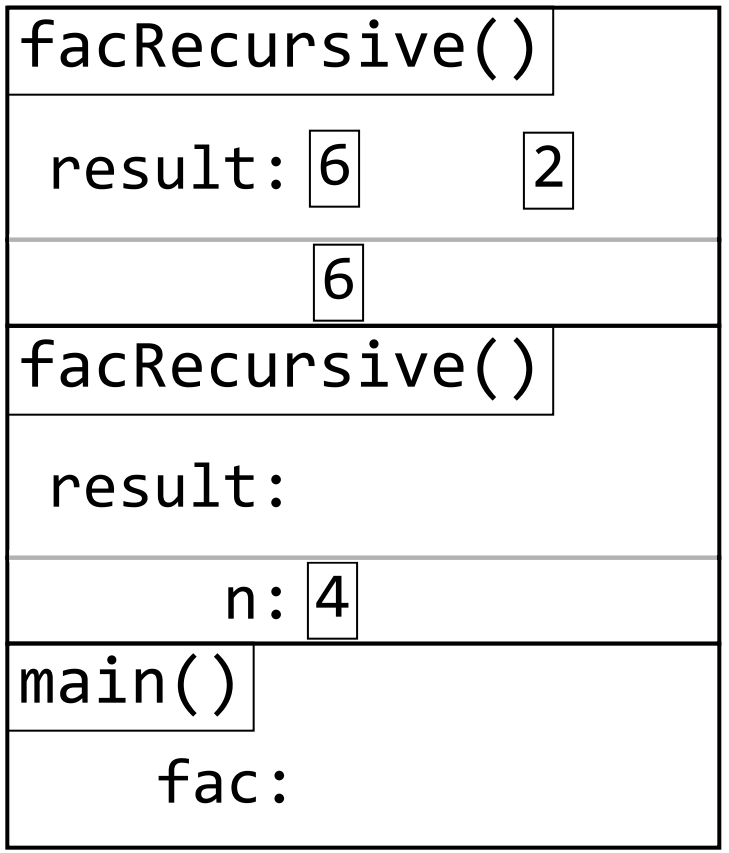
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



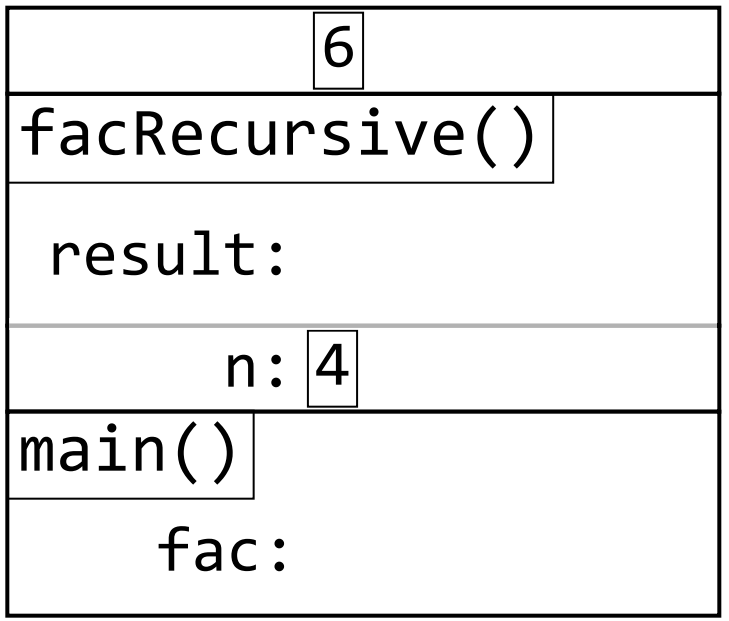
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



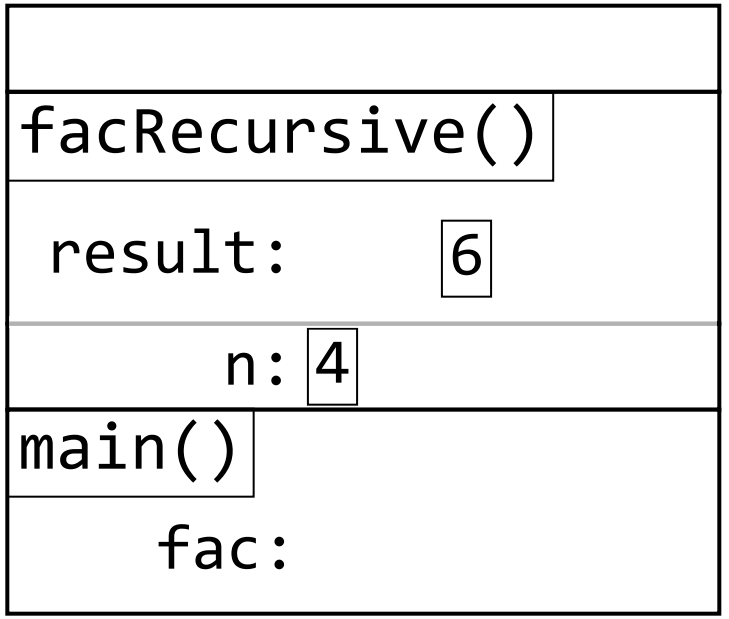
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



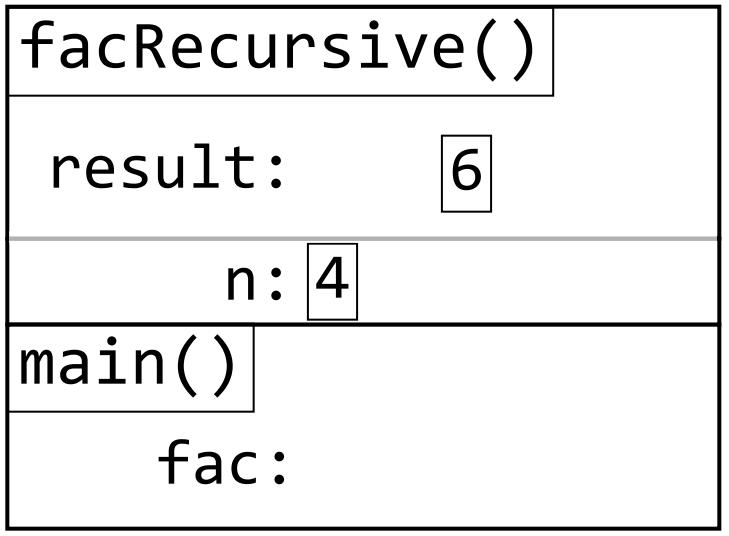
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



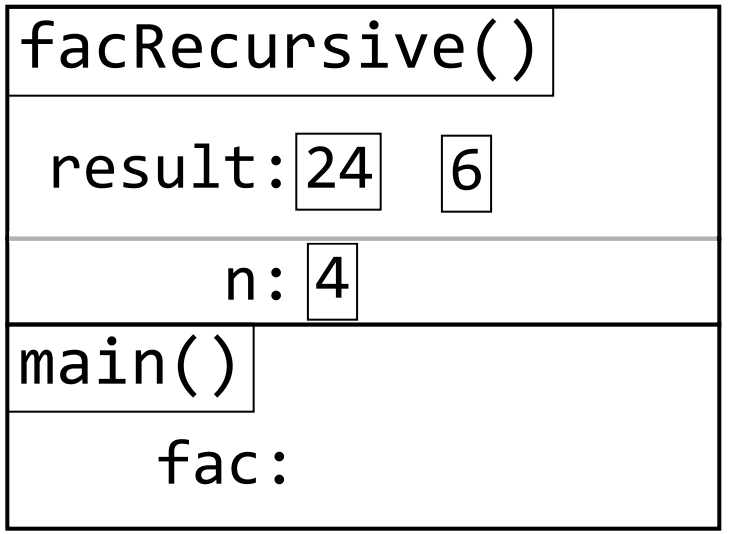
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



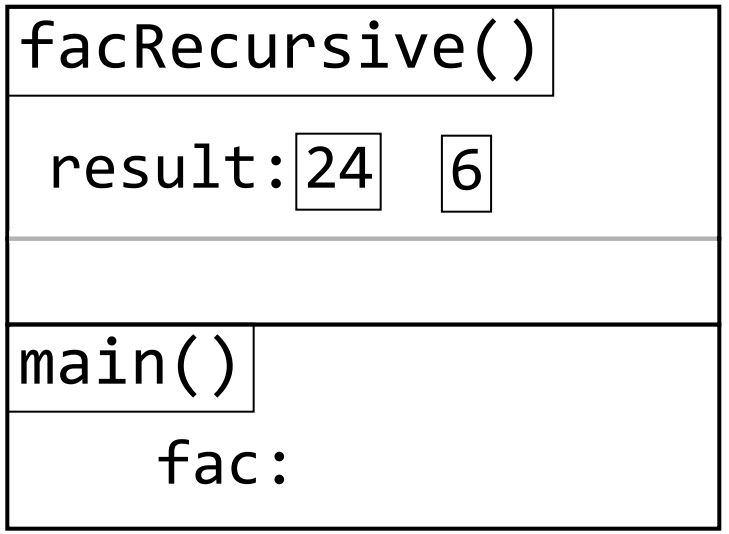
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



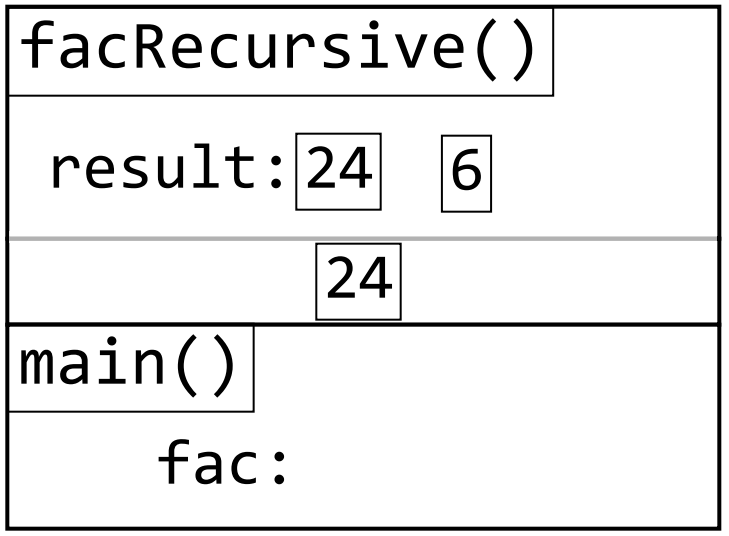
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



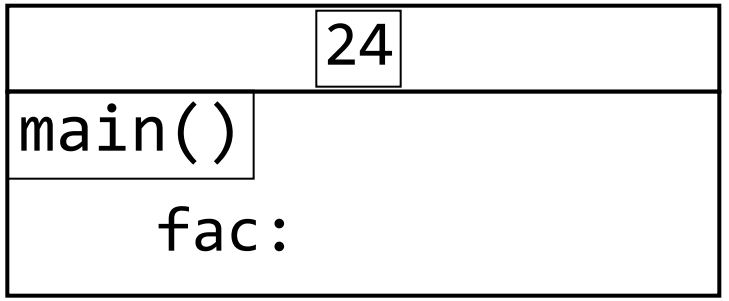
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



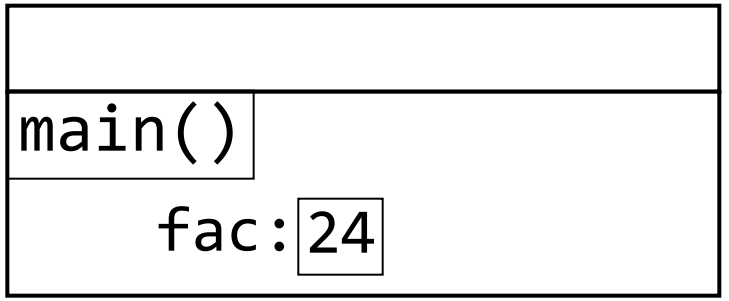
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



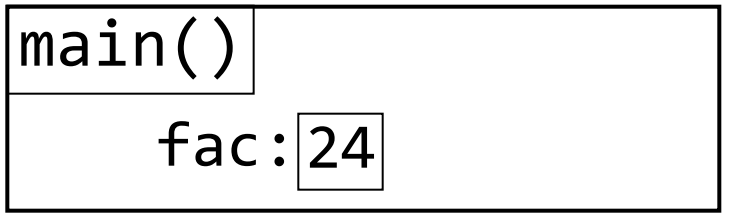
```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



```
int facRecursive(int n)
{
  int result;

  if (n == 1)
  {
    result = 1;
  }
  else
  {
    result = n * facRecursive(n-1);
  }

  return result;
}

int main(void)
{
  int fac = facRecursive(4);

  return 0;
}
```



Print out a line of n stars

```
n = 1      *
n = 5      *****
```

```
void printRow(int n)
{
    for (int i = 0; i < n; i++)
    {
        printf("*");
    }
}
```

Print out a line of n stars

n = 1 *

n = 5 *****