Recursion

Recursion

- *Recursion is a technique that leads to elegant solutions to problems that are difficult to program using simple loops.*

- To use recursion is to program using *recursive functions*—functions that invoke themselves.

- Recursion is a useful programming technique. In some cases, it enables you to develop a natural, straightforward, simple solution to an otherwise difficult problem.

Example: Factorials

- *A recursive function is one that invokes itself.*
- Base Case
- The recursive algorithm for computing factorial(n) can be simply described as follows:

```
if (n == 0)
  return 1;
else
  return n * factorial(n - 1);
```

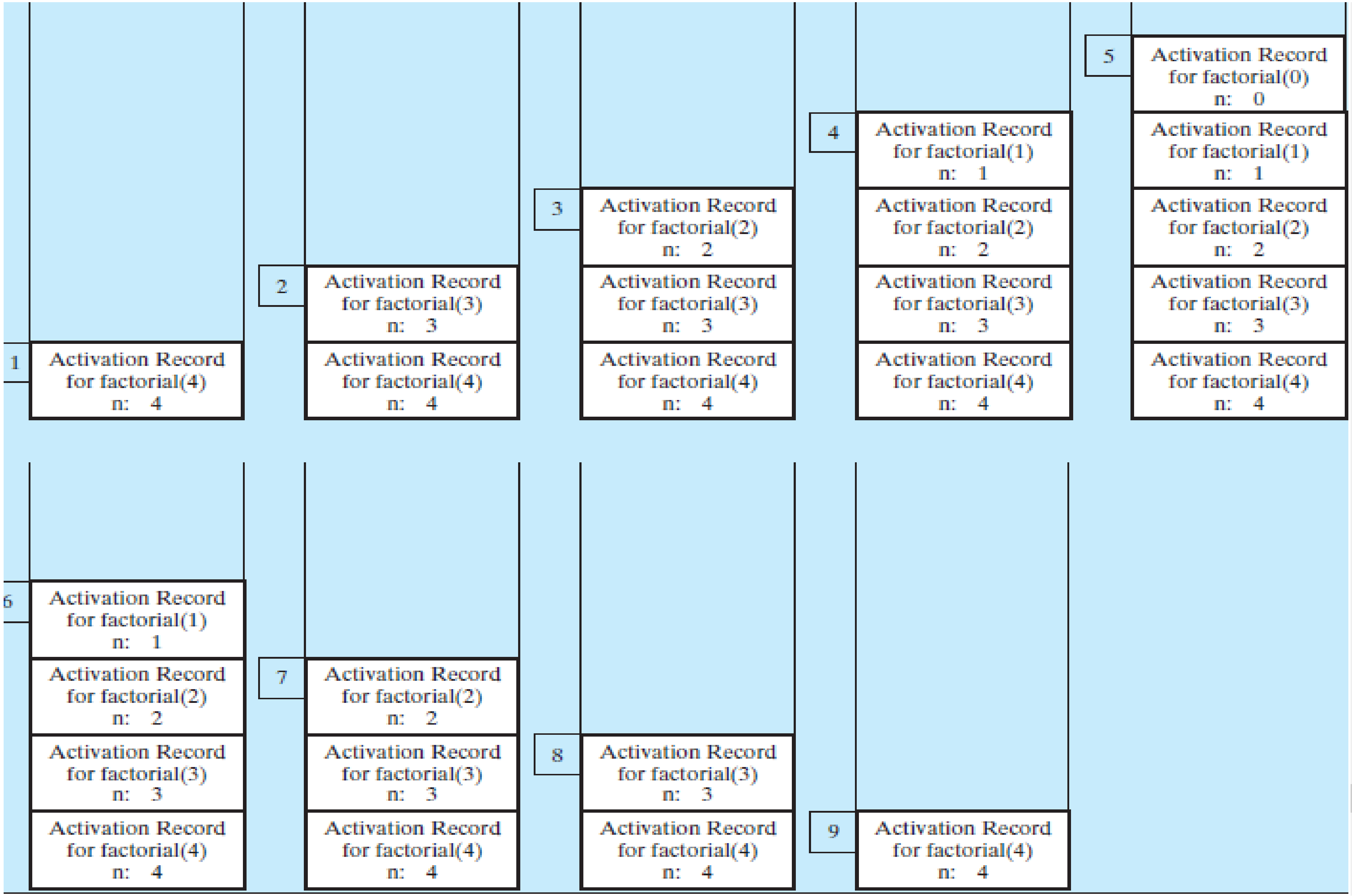
Recursive Call

- A recursive call can result in many more recursive calls, because the function is dividing a subproblem into new subproblems.
- For a recursive function to terminate, the problem must eventually be reduced to a stopping case.
- At this point the function returns a result to its caller.

```
#include <iostream>
using namespace std;

// Return the factorial for a specified index
int factorial(int);

int main()
{
  // Prompt the user to enter an integer
  cout << "Please enter a non-negative integer: ";
  int n;
  cin >> n;

  // Display factorial
  cout << "Factorial of " << n << " is " << factorial(n);

  return 0;
}

// Return the factorial for a specified index
int factorial(int n)
{
  if (n == 0) // Base case
    return 1;
  else
    return n * factorial(n - 1); // Recursive call
}
```

base case

recursion



1
Activation Record for factorial(4) n: 4
2
Activation Record for factorial(3) n: 3
Activation Record for factorial(4) n: 4
3
Activation Record for factorial(2) n: 2
Activation Record for factorial(3) n: 3
Activation Record for factorial(4) n: 4
4
Activation Record for factorial(1) n: 1
Activation Record for factorial(2) n: 2
Activation Record for factorial(3) n: 3
Activation Record for factorial(4) n: 4
5
Activation Record for factorial(0) n: 0
Activation Record for factorial(1) n: 1
Activation Record for factorial(2) n: 2
Activation Record for factorial(3) n: 3
Activation Record for factorial(4) n: 4
6
Activation Record for factorial(1) n: 1
Activation Record for factorial(2) n: 2
Activation Record for factorial(3) n: 3
Activation Record for factorial(4) n: 4
7
Activation Record for factorial(2) n: 2
Activation Record for factorial(3) n: 3
Activation Record for factorial(4) n: 4
8
Activation Record for factorial(3) n: 3
Activation Record for factorial(4) n: 4
9
Activation Record for factorial(4) n: 4

Infinite Recursion

- *Infinite recursion* can occur if recursion does not reduce the problem in a manner that allows it to eventually converge into the base case or a base case is not specified.
- For example, suppose you mistakenly write the factorial function as follows:

```
int factorial(int n)
{
  return n * factorial(n - 1);
}
```

- The function runs infinitely and causes the stack overflow.

Show the output of the following programs and identify base cases and recursive calls.

```
#include <iostream>
using namespace std;

int f(int n)
{
  if (n == 1)
    return 1;
  else
    return n + f(n - 1);
}

int main()
{
  cout << "Sum is " << f(5) << endl;

  return 0;
}
```

```
#include <iostream>
using namespace std;

void f(int n)
{
  if (n > 0)
  {
    cout << n % 10;
    f(n / 10);
  }
}

int main()
{
  f(1234567);

  return 0;
}
```

Case Study: Fibonacci Numbers

```
The series: 0  1  1  2  3  5  8  13  21  34  55  89 . . .
   Indices: 0  1  2  3  4  5  6  7   8   9   10  11
```

The Fibonacci series begins with 0 and 1, and each subsequent number is the sum of the preceding two numbers in the series. The series can be defined recursively as follows:

```
fib(0) = 0;
fib(1) = 1;
fib(index) = fib(index - 2) + fib(index - 1); index >= 2
```

Recursive Algorithm for Fibonacci Series

- The recursive algorithm for computing fib(index) can be simply described as follows:

```
if (index == 0)
  return 0;
else if (index == 1)
  return 1;
else
  return fib(index - 1) + fib(index - 2);
```

Show the output of the following two programs:

```
#include <iostream>
using namespace std;

void f(int n)
{
  if (n > 0)
  {
    cout << n << " ";
    f(n - 1);
  }
}

int main()
{
  f(5);

  return 0;
}
```

```
#include <iostream>
using namespace std;

void f(int n)
{
  if (n > 0)
  {
    f(n - 1);
    cout << n << " ";
  }
}

int main()
{
  f(5);

  return 0;
}
```

What is wrong in the following function?

```
#include <iostream>
using namespace std;

void f(double n)
{
  if (n != 0)
  {
    cout << n;
    f(n / 10);
  }
}

int main()
{
  f(1234567);

  return 0;
}
```