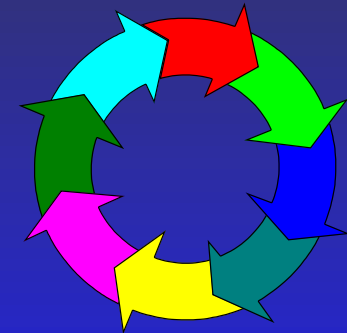


Chapter 12: Recursion



Mr. Dave Clausen
La Cañada High School

Recursion: Definitions

← Recursion

← The process of a subprogram (function) calling itself.

← A clearly defined *stopping state* must exist.

← The well defined termination of the recursive process.

← Any recursive subprogram could be rewritten using iteration (for, while, do...while)

← Recursive step

← A step in the recursive process that solves a similar problem of a smaller size and will eventually lead to a termination of the process.

Recursive Problems in Mathematics

← Factorials

← The definition of factorials is as follows:

← $0! = 1; 1! = 1; \text{ for } n > 1, n! = n * (n-1) * (n-2) * \dots * 2 * 1$

← To find $n!$ you can calculate $n * (n-1)!$

← Each step solves a similar problem of smaller size.

← Fibonacci sequence

← The first term is 1, the second term is also 1, and every successive term is the sum of the previous two terms.

← An Arithmetic Series using sigma notation $\sum_{x=1}^n x$

Recursive Example 1

← First recursion example

← (do not compile or run, there is no stopping state)

[recursion1.cpp](#)

```
void Example (int value)
{
    cout<<"Hello There! "<<value<<endl;
    Example (value + 1);
}
```

Recursive Example 2

← Second Example

[recursion2.cpp](#)

```
void Example2 (int value)
{
    cout<<"Hello, value= "<<value<<endl;
    if (value < 5)
        Example2 (value + 1);
}
```

Example 2 Animation: 1

Value

1

Monitor

Hello, value = 1

Value < 5

TRUE

```
cout<<"Hello, value= "<<value<<endl;  
if (value < 5)  
    Example2 (value + 1);
```

Example 2 Animation: 2

Value

2

Monitor

Hello, value = 1

Hello, value = 2

Value < 5

TRUE

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

Example 2 Animation: 3

Value

3

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Value < 5

TRUE

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```


Example 2 Animation: 4

Value

4

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

Value < 5

TRUE

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

Example 2 Animation: 5

Value

5

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

Hello, value = 5

Value < 5

FALSE

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
```

Example 2 Animation: 6

Value

5

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

Hello, value = 5

Value < 5

FALSE



Tail-recursive Algorithms

← Tail-recursive algorithms perform no work after the recursive call.

← Examples 1 and 2 are tail recursive as well as the following example:

```
int tailRecursiveFactorial (int n, int result){  
    if (n == 1)  
        return result;  
    else  
        return tailRecursiveFactorial (n - 1, n * result);  
}
```

Recursive Example 3

← Third Example (not tail recursive)

[recursion3.cpp](#)

```
void Example3 (int value)
{
    cout<<"Hello, value= "<<value<<endl;
    if (value < 5)
        Example3 (value + 1);
    cout<<"Goodbye, value= "<<value<<endl;
}
```

Example 3

Value

1

Monitor

Hello, value = 1

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

Example 3

Value

2

Monitor

Hello, value = 1

Hello, value = 2

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

Example 3

Value

3

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```


Example 3

Value

4

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

Example 3

Value

5

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

Hello, value = 5

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

Example 3

Value

5

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

Hello, value = 5

Goodbye, value = 5

```
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;  
if (value < 5)  
    Example2 (value + 1);  
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;  
if (value < 5)  
    Example2 (value + 1);  
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;  
if (value < 5)  
    Example2 (value + 1);  
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;  
if (value < 5)  
    Example2 (value + 1);  
cout<<"Goodbye, value= "<<value<<endl;
```

Example 3

Value

4

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

Hello, value = 5

Goodbye, value = 5

Goodbye, value = 4

```
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

Example 3

Value

3

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

Hello, value = 5

Goodbye, value = 5

Goodbye, value = 4

Goodbye, value = 3

```
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;
if (value < 5)
    Example2 (value + 1);
cout<<"Goodbye, value= "<<value<<endl;
```

Example 3

Value

2

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

Hello, value = 5

Goodbye, value = 5

Goodbye, value = 4

Goodbye, value = 3

Goodbye, value = 2

```
cout<<"Goodbye, value= "<<value<<endl;
```

```
cout<<"Hello, value= "<<value<<endl;  
if (value < 5)  
    Example2 (value + 1);  
cout<<"Goodbye, value= "<<value<<endl;
```

Example 3

Value

1

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

Hello, value = 5

Goodbye, value = 5

Goodbye, value = 4

Goodbye, value = 3

Goodbye, value = 2

Goodbye, value = 1

```
cout<<"Goodbye, value= "<<value<<endl;
```

Example 3

Value

1

Monitor

Hello, value = 1

Hello, value = 2

Hello, value = 3

Hello, value = 4

Hello, value = 5

Goodbye, value = 5

Goodbye, value = 4

Goodbye, value = 3

Goodbye, value = 2

Goodbye, value = 1

Recursive Example 4

← Arithmetic Series (sigma) Example

P668ex1.cpp

$$\sum_{x=1}^n x$$

```
int sigma(int n)
{
    if(n <= 1)
        return n;
    else
        return n + sigma(n-1);
}
```

Sigma Example Function Calls

- ← If we call the sigma function with the statement, $\text{sum}=\text{sigma}(5)$
- ← the else portion of the first function calls the function again, $\text{return } 5 + \text{sigma}(4);$
 - ← at this point the value of $\text{sigma}(4)$ must be computed
 - ← this calls $\text{return } 4 + \text{sigma}(3)$
 - ← which calls $\text{return } 3 + \text{sigma}(2)$
 - ← then $\text{return } 2 + \text{sigma}(1)$
 - ← finally this returns the value of 1.

Visualizing the sigma Function Calls



←return 5 + sigma(4)

←return 4 + sigma(3)

←return 3 + sigma(2)

←return 2 + sigma(1)

←return 1

←at the end of the recursive calls the steps are reversed for assigning the values.

←return 1

←return(2+1) (=3)

←return(3+3) (=6)

←return(4+6) (=10)

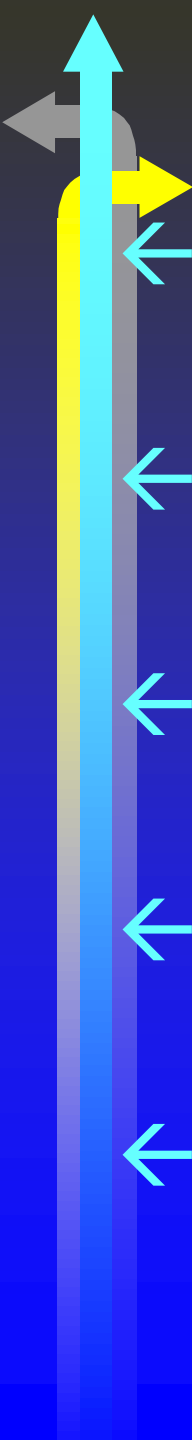
←return(5+10) (=15)

Stacks

- ← Imagine a stack as a pile of trays in a cafeteria. The last tray put on the stack is the first one taken off of the stack.
- ← This is what occurs in memory when a subprogram calls itself.
- ← A stack is a dynamic data structure where access can be made only from one end. This is called a Last In First Out (LIFO) structure.



Stack Animation



← Level 5	n: 1	Sigma: 1
← Level 4:	n: 2	Sigma: 3
← Level 3:	n: 3	Sigma: 6
← Level 2:	n: 4	Sigma: 10
← Level 1:	n: 5	Sigma: 15



Reverse Print Recursion Example

← This example will read a character of text from a file until a period is read. It then uses recursive techniques to print the text in reverse order.

[Revprint.cpp](#)

- ◆ This is a short sentence.
- ◆ .ecnetnes trohs a si sihT

The Costs and Benefits of Recursion

A decorative vertical bar on the left side of the slide, featuring a blue-to-yellow gradient and several arrows pointing in various directions (up, down, left, right).

← Costs

- ← A recursive process that requires N recursive calls uses $N+1$ units of stack memory and processor time to manage the process.
- ← An equivalent iterative process always uses one stack of memory and processor time to manage the function call, therefore, recursion requires more memory and processor power than the non-recursive process.

← Benefits

- ← Short, simple & elegant solutions using divide & conquer algorithms (solve the problem by repeatedly dividing it into simpler problems.)



Some Recursive Algorithms

← The Binary Search

← The Quick Sort

← The Merge Sort

M. C. Escher & Recursion

- ← M.C. Escher “Drawing Hands” lithograph
- ← Illustrates the concept of Recursion

