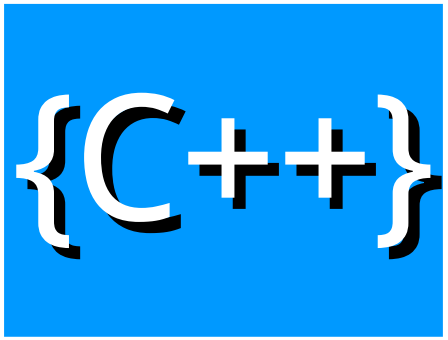
Functions

Week 6



Yang-Cheng Chang
Yuan-Ze University
yczhang@saturn.yzu.edu.tw

The example of a recursive function

- Write a recursive function to print a triangle

```
*
**
***
****
*****
```

The example of a for-loop function

```
// use for-loop to draw a triangle
#include <iostream>
using namespace std;

int main(int argc, const char *argv[])
{
    for(int level = 1; level <= 5; level++){
        for(int i = 0; i < level; i++){
            cout << "*";
        }
        cout << endl;
    }
    return 0;
}
```

The example of a recursive function

```
// use a recursive function to draw a triangle

#include <iostream>

using namespace std;

void drawTriangle(int level);

int main(int argc, const char *argv[])
{
    drawTriangle(5);
    return 0;
}
```

① drawTriangle(5)

```
void drawTriangle(int level)
{
    if( level > 1 ) {
        drawTriangle(level-1);
        for (int i = 0; i < level; i++) {
            cout << "*";
        }
        cout << endl;
    }
    else{
        cout << "*" << endl;
    }
}
```

② drawTriangle(4)
③ drawTriangle(3)
④ drawTriangle(2)
⑤ drawTriangle(1)

The output sequence of a recursive function

The execute sequence

① drawTriangle(5)

② drawTriangle(4)

③ drawTriangle(3)

④ drawTriangle(2)

⑤ drawTriangle(1)

*****

****

***

**

*

The output sequence

```
*
**
***
****
*****
```

The same code

```
// use for-loop to draw a triangle
#include <iostream>
using namespace std;

int main(int argc, const char *argv[])
{
    for(int level = 1; level <= 5; level++){
        for(int i = 0; i < level; i++){
            cout << "*";
        }
        cout << endl;
    }
    return 0;
}
```

```
// use a recursive function to draw a triangle

#include <iostream>

using namespace std;

void drawTriangle(int level);

int main(int argc, const char *argv[])
{
    drawTriangle(5);
    return 0;
}
void drawTriangle(int level)
{
    if( level > 1 ) {
        drawTriangle(level-1);
        for (int i = 0; i < level; i++) {
            cout << "*";
        }
        cout << endl;
    }
    else{
        cout << "*" << endl;
    }
}
```

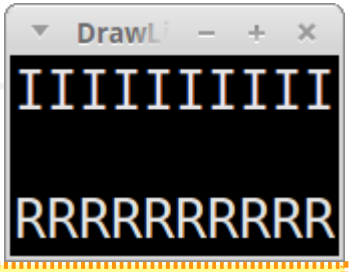
The different code

```
// use for-loop to draw a triangle
#include <iostream>
using namespace std;

int main(int argc, const char *argv[])
{
    for(int level = 1; level <= 5; level++){
        for(int i = 0; i < level; i++){
            cout << "*";
        }
        cout << endl;
    }
    return 0;
}
```

1. initialization
2. condition
3. iteration expression

```
// use a recursive function to draw a triangle

#include <iostream>

using namespace std;

void drawTriangle(int level);

int main(int argc, const char *argv[])
{
    drawTriangle(5);
    return 0;
}
void drawTriangle(int level)
{
    if( level > 1 ) {
        drawTriangle(level-1);
        for (int i = 0; i < level; i++) {
            cout << "*";
        }
        cout << endl;
    }
    else{
        cout << "*" << endl;
    }
}
```

Recursion example: draw line



Iterative

```
void DrawLine_Iterative( int amount, char symbol )
{
    for ( int i = 0; i < amount; i++ )
    {
        cout << symbol;
    }
}
```

Sometimes, we can implement something as a simple loop...

Recursive

```
void DrawLine_Recursive( int amount, char symbol )
{
    if ( amount == 0 )  return;

    cout << symbol;
    DrawLine_Recursive( amount-1, symbol );
}
```

...or as a recursive function, with relative ease.

However, sometimes a recursive function makes for the simplest implementation

Recursion example: draw line

Recursive

```
void DrawLine_Recursive( int amount, char symbol )
{
    if ( amount == 0 )  return;

    cout << symbol;
    DrawLine_Recursive( amount-1, symbol );
}
```

When creating a function that will call itself, we need to make sure to include a **Base case (aka stopping case)** so the function does not loop forever.

But if that base case is not hit, we will do the process over again, and call the function with new parameters.

Recursion example: Factorial

Factorial

- The factorial of a nonnegative integer n, written n! (and pronounced “n factorial”), is the product

  $n! = n * (n - 1) * (n - 2) * \ldots * 2 * 1$

- with 1! equal to 1, and 0! defined to be 1.

Recursion example: Factorial

Iterative Factorial

- The factorial of an integer, number, greater than or equal to 0, can be calculated iteratively (nonrecursively) by using a loop.

Recursive Factorial

- A recursive definition of the factorial function is arrived at by observing the following algebraic relationship:
  - $n! = n * (n - 1)!$

Recursion example: Factorial

Iterative

```
int Factorial_Iterative( int number )
{
    int value = number;
    for ( int i = number-1; i > 0; i-- )
    {
        value *= i;
    }
    return value;
}
```

Recursive

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

A recursive function can be void, or it can return a value.

It can take some work to think in terms of "recursion" instead of "iteration".

Build your function a step at a time, and test after each addition.

Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`

Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of `Factorial_Recursive( 3 )`

Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call:
`Factorial_Recursive( 5 );`

Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`

Call #2:
`Factorial_Recursive( 4 );`

Number isn't 0, so multiply 4 by the result of `Factorial_Recursive( 3 )`

Call #3:
`Factorial_Recursive( 3 );`

Number isn't 0, so multiply 3 by the result of `Factorial_Recursive( 2 )`

Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call: `Factorial_Recursive( 5 );`	Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`
Call #2: `Factorial_Recursive( 4 );`	Number isn't 0, so multiply 4 by the result of `Factorial_Recursive( 3 )`
Call #3: `Factorial_Recursive( 3 );`	Number isn't 0, so multiply 3 by the result of `Factorial_Recursive( 2 )`
Call #4: `Factorial_Recursive( 2 );`	Number isn't 0, so multiply 2 by the result of `Factorial_Recursive( 1 )`

Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call: `Factorial_Recursive( 5 );`	Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`
Call #2: `Factorial_Recursive( 4 );`	Number isn't 0, so multiply 4 by the result of `Factorial_Recursive( 3 )`
Call #3: `Factorial_Recursive( 3 );`	Number isn't 0, so multiply 3 by the result of `Factorial_Recursive( 2 )`
Call #4: `Factorial_Recursive( 2 );`	Number isn't 0, so multiply 2 by the result of `Factorial_Recursive( 1 )`
Call #5: `Factorial_Recursive( 1 );`	Number isn't 0, so multiply 1 by the result of `Factorial_Recursive( 0 )`

Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call: `Factorial_Recursive( 5 );`	Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`
Call #2: `Factorial_Recursive( 4 );`	Number isn't 0, so multiply 4 by the result of `Factorial_Recursive( 3 )`
Call #3: `Factorial_Recursive( 3 );`	Number isn't 0, so multiply 3 by the result of `Factorial_Recursive( 2 )`
Call #4: `Factorial_Recursive( 2 );`	Number isn't 0, so multiply 2 by the result of `Factorial_Recursive( 1 )`
Call #5: `Factorial_Recursive( 1 );`	Number isn't 0, so multiply 1 by the result of `Factorial_Recursive( 0 )`
Call #6: `Factorial_Recursive( 0 );`	Number is 0, so return 1

Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call: Factorial_Recursive(5);	Number isn't 0, so multiply 5 by the result of Factorial_Recursive(4)
Call #2: Factorial_Recursive(4);	Number isn't 0, so multiply 4 by the result of Factorial_Recursive(3)
Call #3: Factorial_Recursive(3);	Number isn't 0, so multiply 3 by the result of Factorial_Recursive(2)
Call #4: Factorial_Recursive(2);	Number isn't 0, so multiply 2 by the result of Factorial_Recursive(1)
Call #5: Factorial_Recursive(1);	Number isn't 0, so multiply 1 by the result of Factorial_Recursive(0)
Call #6: Factorial_Recursive(0);	Number is 0, so return 1

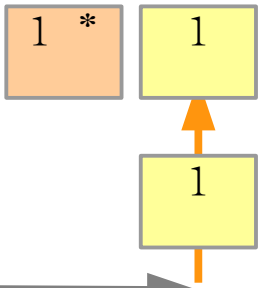
1

Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call: Factorial_Recursive(5);	Number isn't 0, so multiply 5 by the result of Factorial_Recursive(4)
Call #2: Factorial_Recursive(4);	Number isn't 0, so multiply 4 by the result of Factorial_Recursive(3)
Call #3: Factorial_Recursive(3);	Number isn't 0, so multiply 3 by the result of Factorial_Recursive(2)
Call #4: Factorial_Recursive(2);	Number isn't 0, so multiply 2 by the result of Factorial_Recursive(1)
Call #5: Factorial_Recursive(1);	Number isn't 0, so multiply 1 by the result of Factorial_Recursive(0)
Call #6: Factorial_Recursive(0);	Number is 0, so return 1

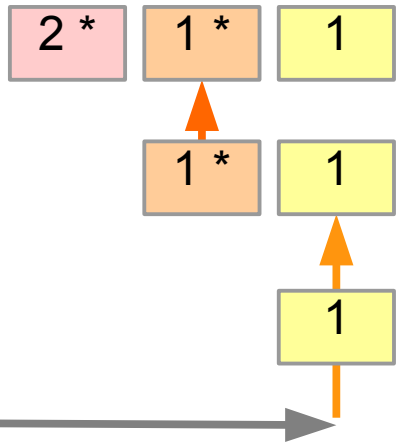


Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call: `Factorial_Recursive( 5 );`	Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`
Call #2: `Factorial_Recursive( 4 );`	Number isn't 0, so multiply 4 by the result of `Factorial_Recursive( 3 )`
Call #3: `Factorial_Recursive( 3 );`	Number isn't 0, so multiply 3 by the result of `Factorial_Recursive( 2 )`
Call #4: `Factorial_Recursive( 2 );`	Number isn't 0, so multiply 2 by the result of `Factorial_Recursive( 1 )`
Call #5: `Factorial_Recursive( 1 );`	Number isn't 0, so multiply 1 by the result of `Factorial_Recursive( 0 )`
Call #6: `Factorial_Recursive( 0 );`	Number is 0, so return 1

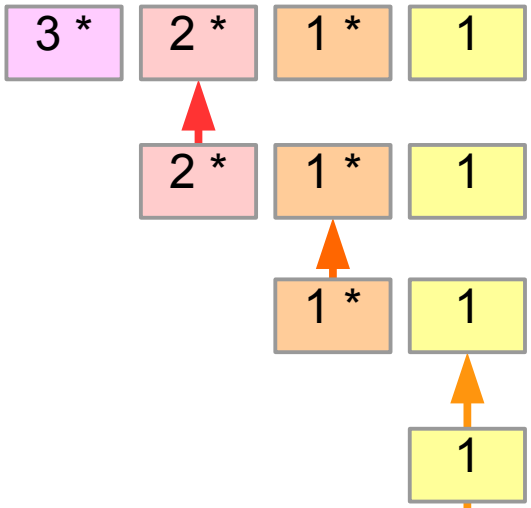


Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call: Factorial_Recursive(5);	Number isn't 0, so multiply 5 by the result of Factorial_Recursive(4)
Call #2: Factorial_Recursive(4);	Number isn't 0, so multiply 4 by the result of Factorial_Recursive(3)
Call #3: Factorial_Recursive(3);	Number isn't 0, so multiply 3 by the result of Factorial_Recursive(2)
Call #4: Factorial_Recursive(2);	Number isn't 0, so multiply 2 by the result of Factorial_Recursive(1)
Call #5: Factorial_Recursive(1);	Number isn't 0, so multiply 1 by the result of Factorial_Recursive(0)
Call #6: Factorial_Recursive(0);	Number is 0, so return 1

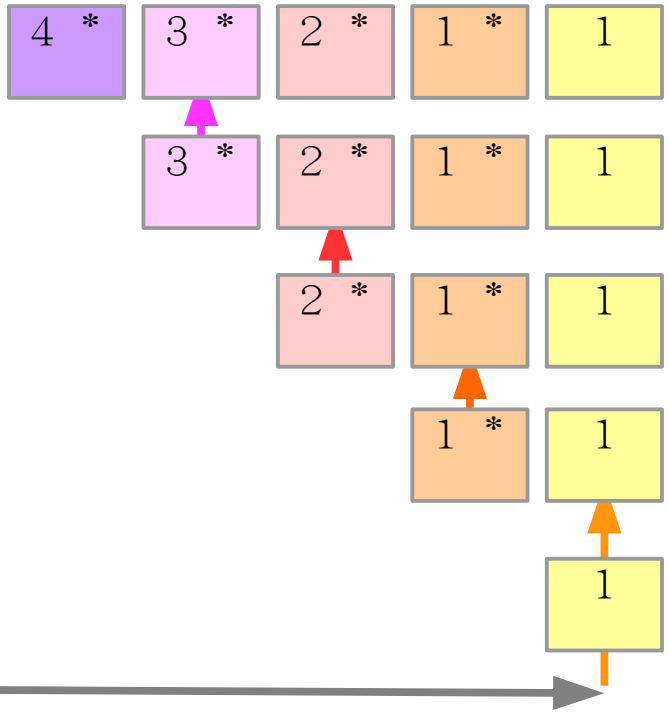


Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call: Factorial_Recursive(5);	Number isn't 0, so multiply 5 by the result of Factorial_Recursive(4)
Call #2: Factorial_Recursive(4);	Number isn't 0, so multiply 4 by the result of Factorial_Recursive(3)
Call #3: Factorial_Recursive(3);	Number isn't 0, so multiply 3 by the result of Factorial_Recursive(2)
Call #4: Factorial_Recursive(2);	Number isn't 0, so multiply 2 by the result of Factorial_Recursive(1)
Call #5: Factorial_Recursive(1);	Number isn't 0, so multiply 1 by the result of Factorial_Recursive(0)
Call #6: Factorial_Recursive(0);	Number is 0, so return 1

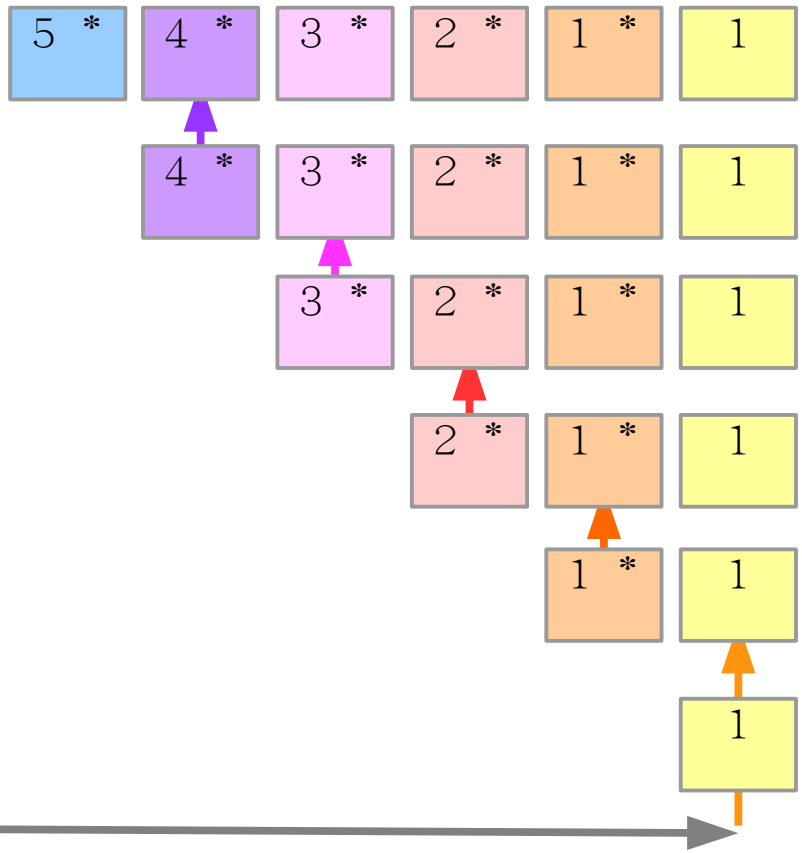


Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

Initial call: `Factorial_Recursive( 5 );`	Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`
Call #2: `Factorial_Recursive( 4 );`	Number isn't 0, so multiply 4 by the result of `Factorial_Recursive( 3 )`
Call #3: `Factorial_Recursive( 3 );`	Number isn't 0, so multiply 3 by the result of `Factorial_Recursive( 2 )`
Call #4: `Factorial_Recursive( 2 );`	Number isn't 0, so multiply 2 by the result of `Factorial_Recursive( 1 )`
Call #5: `Factorial_Recursive( 1 );`	Number isn't 0, so multiply 1 by the result of `Factorial_Recursive( 0 )`
Call #6: `Factorial_Recursive( 0 );`	Number is 0, so return 1



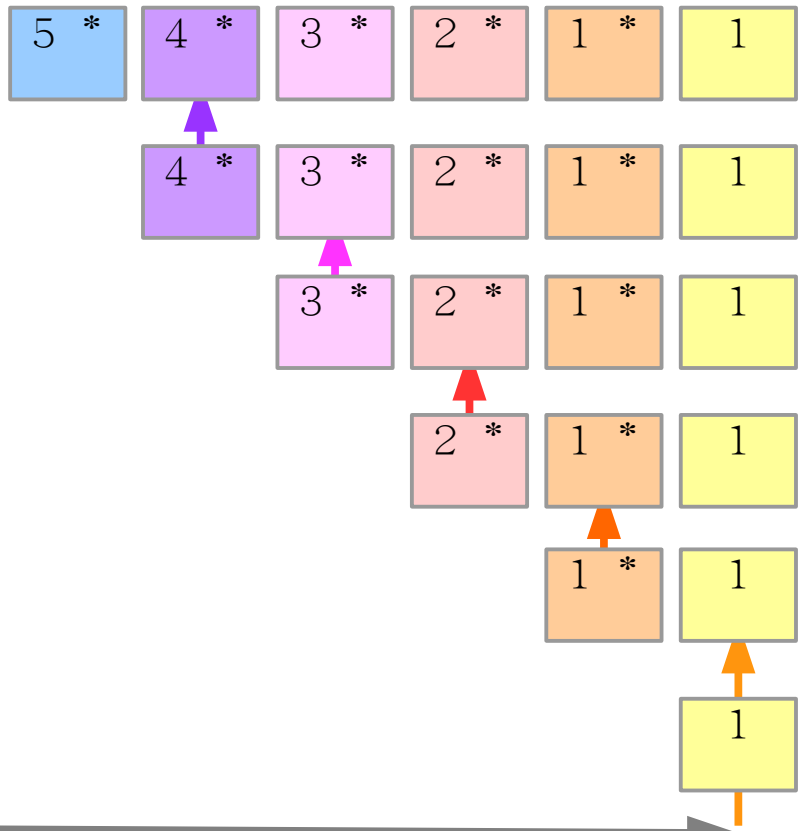
Recursive evaluation of 5!

Let's step through the execution of this function...

```
int Factorial_Recursive( int number )
{
    if ( number == 0 )
        return 1;

    return ( number * Factorial_Recursive( number - 1 ) );
}
```

120

Initial call: `Factorial_Recursive( 5 );`	Number isn't 0, so multiply 5 by the result of `Factorial_Recursive( 4 )`
Call #2: `Factorial_Recursive( 4 );`	Number isn't 0, so multiply 4 by the result of `Factorial_Recursive( 3 )`
Call #3: `Factorial_Recursive( 3 );`	Number isn't 0, so multiply 3 by the result of `Factorial_Recursive( 2 )`
Call #4: `Factorial_Recursive( 2 );`	Number isn't 0, so multiply 2 by the result of `Factorial_Recursive( 1 )`
Call #5: `Factorial_Recursive( 1 );`	Number isn't 0, so multiply 1 by the result of `Factorial_Recursive( 0 )`
Call #6: `Factorial_Recursive( 0 );`	Number is 0, so return 1



Week 6 Assignment

- Input an integer number(0~32767), write a program to convert it from decimal numbers to hexadecimal numbers.
  $2001 = 7D1_{16}$
- Write two version of the program
  - Iterative(for-loop or while)
  - recursive(function)

Week 6 Assignment

- Tips
  - 2001 / 16 = 125 2001 % 16 = 1
    125 / 16 = 7 125 % 16 = 13
    7 / 16 = 0 7 % 16 = 7
  - 10=A_{16}, 11=B_{16}, 12=C_{16}, 13=D_{16}, 14=E_{16}, 15=F_{16}