

CS 2133

Recursion

Recursion

- In some problems, it may be natural to **define the problem in terms of the problem itself**.
(A *recursive function* is a function that *calls itself*)
- Recursion is useful for problems that can be represented by a **simpler version** of the same problem.
- Example: the factorial function

$$6! = 6 * 5 * 4 * 3 * 2 * 1$$

We could write:

$$6! = 6 * 5!$$

Example : factorial function

In general, we can express the factorial function as follows:

$$n! = n * (n-1)!$$

Is this correct? Well... almost.

The factorial function is only defined for *positive* integers. So we should be a bit more precise:

$$\begin{array}{ll} n! = 1 & \text{(if } n \text{ is equal to } 1) \\ n! = n * (n-1)! & \text{(if } n \text{ is larger than } 1) \end{array}$$

factorial function

The C++ equivalent of this definition:

```
int fac(int numb){  
    if(numb<=1)  
        return 1;  
    else  
        return numb * fac(numb-1);  
}
```

recursion means that a function calls itself

factorial function

- Assume the number typed is 3, that is, numb=3.

fac(3) :

3 <= 1 ?

No.

fac(3) = 3 * fac(2)

fac(2) :

2 <= 1 ?

No.

fac(2) = 2 * fac(1)

fac(1) :

1 <= 1 ?

Yes.

return 1

fac(2) = 2 * 1 = 2

return fac(2)

fac(3) = 3 * 2 = 6

return fac(3)

fac(3) has the value 6

```
int fac(int numb) {  
    if(numb<=1)  
        return 1;  
    else  
        return numb * fac(numb-1);  
}
```

factorial function

For certain problems (such as the factorial function), a recursive solution often leads to short and elegant code. Compare the recursive solution with the iterative solution:

Recursive solution

```
int fac(int numb) {  
    if (numb <= 1)  
        return 1;  
    else  
        return numb * fac (numb-1) ;  
}
```

Iterative solution

```
int fac(int numb) {  
    int product=1;  
    while (numb > 1) {  
        product *= numb;  
        numb--;  
    }  
    return product;  
}
```

Recursion

We have to pay a price for recursion:

- calling a function **consumes more time and memory** than adjusting a loop counter.
- high performance applications (graphic action games, simulations of nuclear explosions) hardly ever use recursion.

In less demanding applications recursion is an attractive alternative for iteration (for the right problems!)

Recursion

If we use iteration, we must be careful not to create an infinite loop by accident:

```
for(int incr=1; incr!=10;incr+=2)  
    ...
```



Oops!

```
int result = 1;  
while(result >0) {  
    ...  
    result++;  
}
```



Oops!


Recursion

Similarly, if we use recursion we must be careful not to create an infinite chain of function calls:

```
int fac(int numb) {  
    return numb * fac(numb-1);  
}
```

Oops!
**No termination
condition**

Or:

```
int fac(int numb) {  
    if (numb <= 1)   
        return 1;  
    else  
        return numb * fac(numb+1);  
}
```

Oops!

Recursion

We must always make sure that the recursion *bottoms out*:

- A recursive function must contain **at least one non-recursive branch**.
- The recursive calls must eventually lead to a non-recursive branch.

Recursion

- Recursion is one way to decompose a task into smaller subtasks. At least one of the subtasks is a smaller example of the same task.
- The smallest example of the same task has a non-recursive solution.

Example: The factorial function

$$n! = n * (n-1)! \text{ and } 1! = 1$$

Example: Fibonacci numbers

- Fibonacci numbers:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

where each number is the sum of the preceding two.

- Recursive definition:

- $F(0) = 0;$

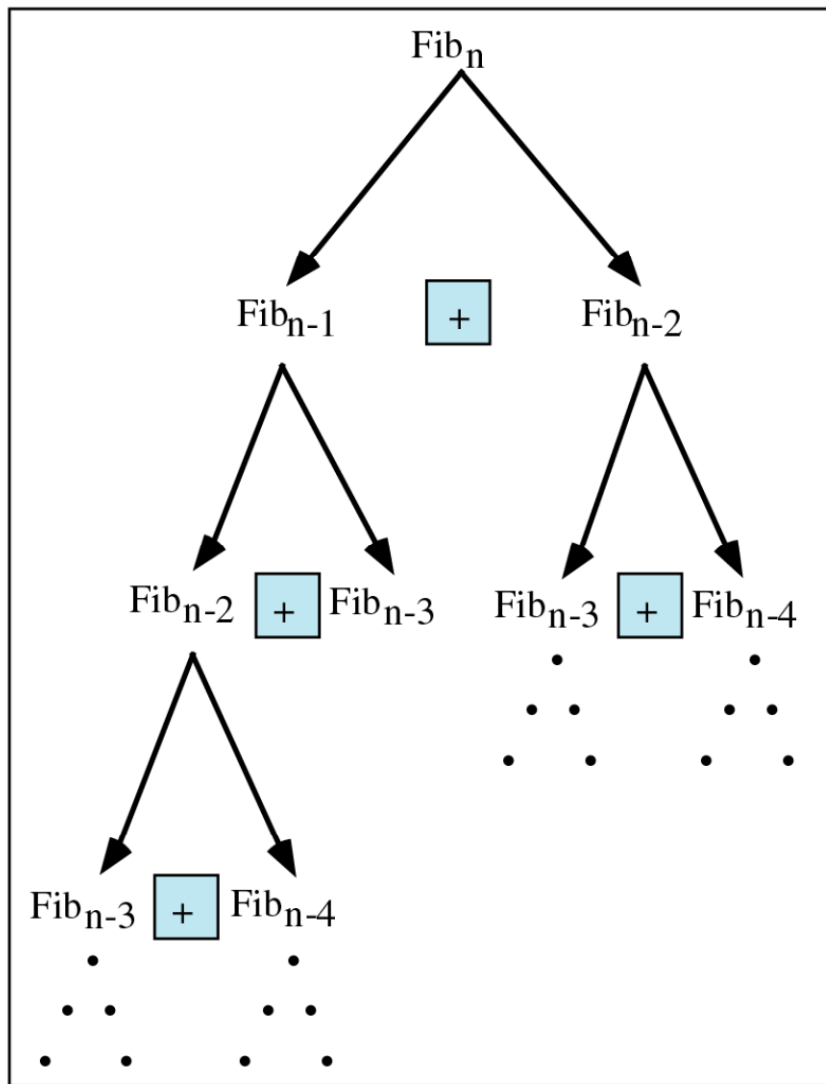
- $F(1) = 1;$

- $F(\text{number}) = F(\text{number}-1) + F(\text{number}-2);$

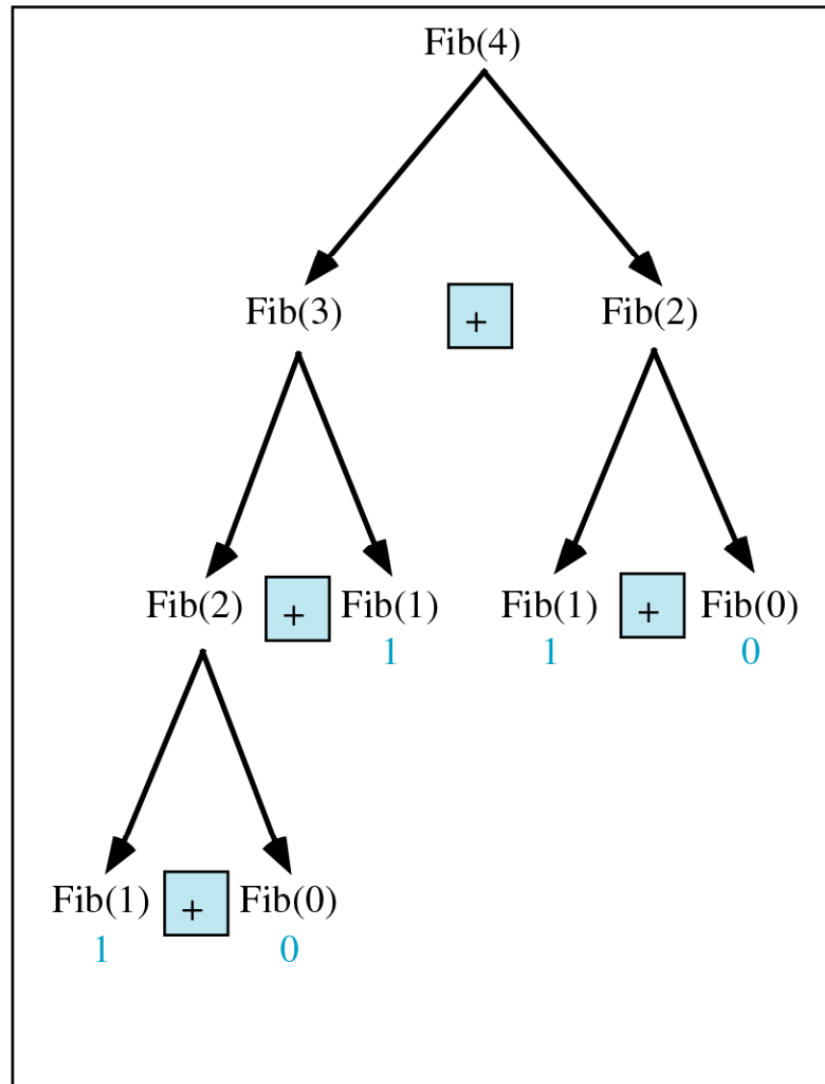
Example: Fibonacci numbers

```
//Calculate Fibonacci numbers using recursive function.
//A very inefficient way, but illustrates recursion well
int fib(int number)
{
    if (number == 0) return 0;
    if (number == 1) return 1;
    return (fib(number-1) + fib(number-2));
}

int main() {
    int inp_number;
    cout << "Please enter an integer: ";
    cin >> inp_number;
    cout << "The Fibonacci number for "<< inp_number
        << " is "<< fib(inp_number)<<endl;
    return 0;
}
```



(a) $\text{Fib}(n)$



(b) $\text{Fib}(4)$

Trace a Fibonacci Number

- Assume the input number is 4, that is, num=4:

fib(4) :

4 == 0 ? No; 4 == 1? No.

fib(4) = fib(3) + fib(2)

fib(3) :

3 == 0 ? No; 3 == 1? No.

fib(3) = fib(2) + fib(1)

fib(2) :

2 == 0? No; 2==1? No.

fib(2) = fib(1)+fib(0)

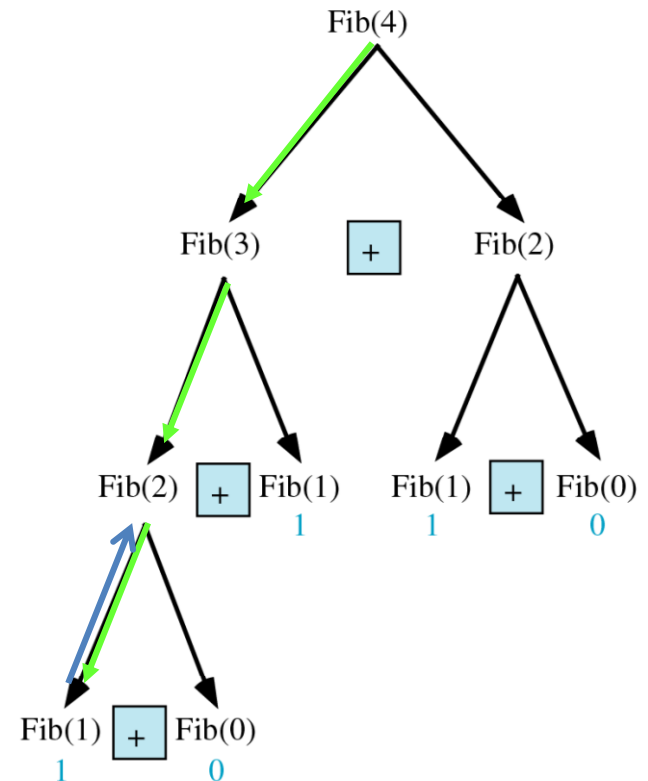
fib(1) :

1== 0 ? No; 1 == 1? Yes.

fib(1) = 1;

return fib(1);

```
int fib(int num)
{
    if (num == 0) return 0;
    if (num == 1) return 1;
    return
        (fib(num-1)+fib(num-2));
}
```



Trace a Fibonacci Number

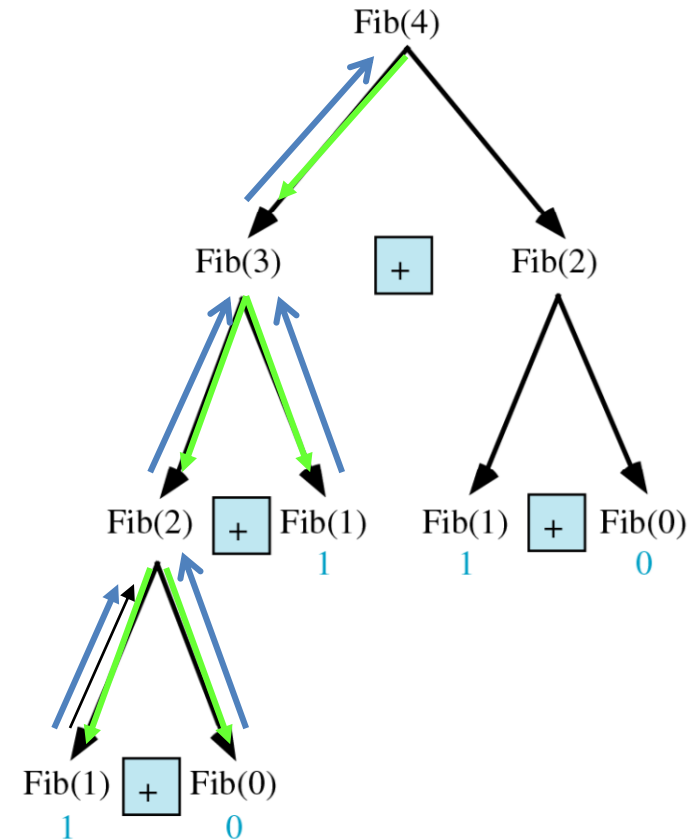
```
fib(0):  
    0 == 0 ? Yes.  
    fib(0) = 0;  
    return fib(0);
```

```
fib(2) = 1 + 0 = 1;  
return fib(2);
```

```
fib(3) = 1 + fib(1)
```

```
fib(1):  
    1 == 0 ? No; 1 == 1? Yes  
    fib(1) = 1;  
    return fib(1);
```

```
fib(3) = 1 + 1 = 2;  
return fib(3)
```



Trace a Fibonacci Number

```
fib(2):
```

```
2 == 0 ? No; 2 == 1? No.
```

```
fib(2) = fib(1) + fib(0)
```

```
fib(1):
```

```
1 == 0 ? No; 1 == 1? Yes.
```

```
fib(1) = 1;
```

```
return fib(1);
```

```
fib(0):
```

```
0 == 0 ? Yes.
```

```
fib(0) = 0;
```

```
return fib(0);
```

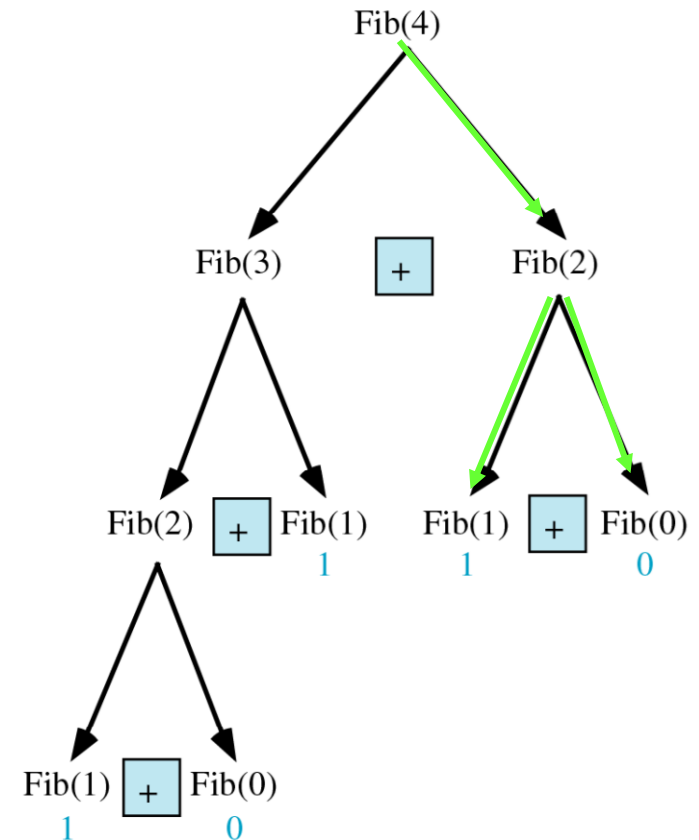
```
fib(2) = 1 + 0 = 1;
```

```
return fib(2);
```

```
fib(4) = fib(3) + fib(2)
```

```
      = 2 + 1 = 3;
```

```
return fib(4);
```



Example : Fibonacci number w/o recursion

```
//Calculate Fibonacci numbers iteratively  
//much more efficient than recursive solution
```

```
int fib(int n)  
{  
    int f[100];  
    f[0] = 0; f[1] = 1;  
    for (int i=2; i<= n; i++)  
        f[i] = f[i-1] + f[i-2];  
    return f[n];  
}
```

Fibonacci Numbers

- Fibonacci numbers can also be represented by the following formula.

$$F_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1 + \sqrt{5}}{2} \right)^n - \left(\frac{1 - \sqrt{5}}{2} \right)^n \right)$$

Recursion General Form

- How to write recursively?

```
int recur_fn(parameters) {  
    if(stopping condition)  
        return stopping value;  
    // other stopping conditions if needed  
    return function of recur_fn(revised parameters)  
}
```

Example : exponential func

- How to write `exp(int numb, int power)` recursively?

```
int exp(int numb, int power) {  
    if(power ==0)  
        return 1;  
    return numb * exp(numb, power -1) ;  
}
```

Example : number of zero

- Write a recursive function that counts the number of zero digits in an integer
- **zeros(10200)** returns 3.

```
int zeros(int numb){  
    if(numb==0)                // 1 digit (zero/non-zero) :  
        return 1;             // bottom out.  
    else if(numb < 10 && numb > -10)  
        return 0;  
    else                        // > 1 digits: recursion  
        return zeros(numb/10) + zeros(numb%10);  
}
```

```
zeros(10200)  
zeros(1020)          + zeros(0)  
zeros(102)           + zeros(0) + zeros(0)  
zeros(10)             + zeros(2) + zeros(0) + zeros(0)  
zeros(1) + zeros(0) + zeros(2) + zeros(0) + zeros(0)
```

Problem Solving Using Recursion

Let us consider a simple problem of printing a message for n times. You can **break the problem into two subproblems**: one is to print the message one time and the other is to print the message for $n-1$ times. **The second problem is the same as the original problem with a smaller size**. The base case for the problem is $n==0$. You can solve this problem using recursion as follows:

```
void nPrintln(char * message, int times)
{
    if (times >= 1) {
        cout << message << endl;
        nPrintln(message, times - 1);
    } // The base case is n == 0
}
```

Think Recursively

So, many of the problems can be solved easily using recursion if you *think recursively*.

Remember this:

```
int recur_fn(parameters) {  
    if(stopping condition)  
        return stopping value;  
    // other stopping conditions if needed  
    return function of recur_fn(revised parameters)  
}
```


Example : Towers of Hanoi



- Only one disc could be moved at a time
- A larger disc must never be stacked above a smaller one
- One and only one extra needle could be used for intermediate storage of discs

Towers of Hanoi

```
void hanoi(int from, int to, int num)
{
    int temp = 6 - from - to; //find the temporary
                               //storage column

    if (num == 1){
        cout << "move disc 1 from " << from
              << " to " << to << endl;
    }
    else {
        hanoi(from, temp, num - 1);
        cout << "move disc " << num << " from " << from
              << " to " << to << endl;
        hanoi(temp, to, num - 1);
    }
}
```

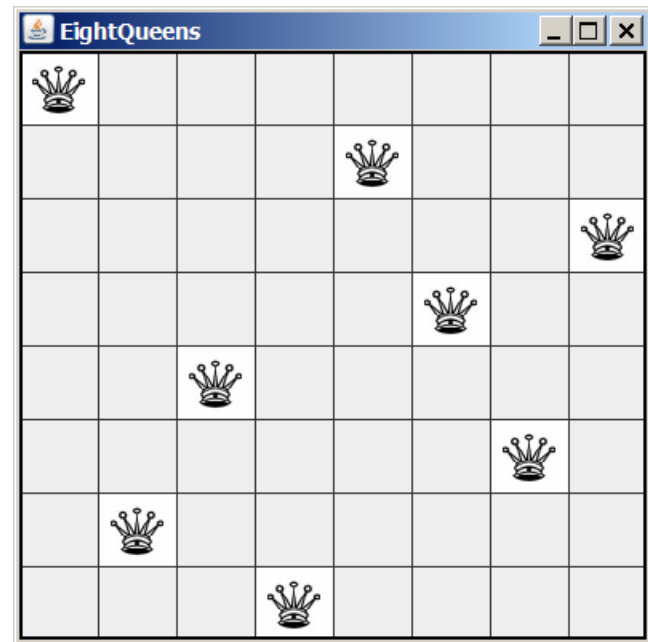
Towers of Hanoi

```
int main() {  
    int num_disc;    //number of discs  
  
    cout << "Please enter a positive number (0 to quit)";  
    cin >> num_disc;  
  
    while (num_disc > 0) {  
        hanoi(1, 3, num_disc);  
        cout << "Please enter a positive number ";  
        cin >> num_disc;  
    }  
    return 0;  
}
```

Eight Queens

Place eight queens on the chessboard such that no queen attacks any other one.

queens[0]	0
queens[1]	4
queens[2]	7
queens[3]	5
queens[4]	2
queens[5]	6
queens[6]	1
queens[7]	3



```

bool empty(int t[], int row, int col) {
    for( int j = 0; j < row; j++) {
        if (t[j] == col)           //same column
            return false;
        if (abs(t[j] - col) == (row - j)) //on cross
line        return false;
    }
    return true;
}

bool queens(int t[], int row, int col) {
    if (row == SIZE) // found one answer
        return true;
    for (col = 0; col < SIZE; col++)
    {
        t[row] = col;
        if (empty(t, row, col) && queens(t, row +1, 0))
            return true;
    }
    return false;
}

```

```

void print(int t[]){
    // print solution
    for(int i = 0; i < SIZE; i++) {
        for(int j = 0; j < SIZE; j++) {
            if (j == t[i])
                cout << " Q ";
            else
                cout << " _ ";
        }
        cout << endl;
    }
}

int main() {
    int t[SIZE]={0};
    for (int i= 0; i <SIZE; i++){
        t[0] = i;  //on first row, Queen on different
column
        cout << endl << endl <<"i is: " << i << endl;
        if (queens(t, 1,0))
            print(t);
    }
}

```