Lecture 8
RECURSION

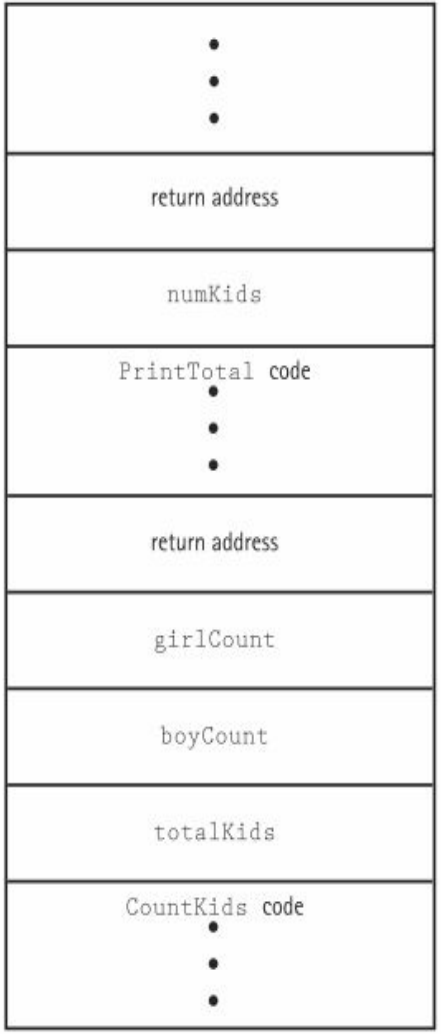
September 23, 2021
Thursday

FUNCTION CALLS

- If the function has formal parameters, they have to be initialized to the values passed as actual parameters.
- The system has to know where to resume the execution after the function is completed.
- Any other function can call this function, call from **main ( )** ?
- Hence, a return address must be stored in main memory.
- But do we only require return address..?

STATIC ALLOCATION

- The function formal parameters and local variables are bound to actual addresses in memory at compile time.
- Think of having a 60 seats and sending invites to only 60 people. You can't entertain the 61st person.



DYNAMIC ALLOCATION

- A runtime stack is a much better solution to preserve the required information about a function.
- The distinction among local variables of different functions.
- What about the distinction between local variables of the same function which has been called more than once.

ACTIVATION RECORD

- The system characterizes each function
  - By the contents of all local variables
  - By the value of function's parameters.
  - By return address indicating

The data area containing all this information is called **Activation Stack** Or **Stack Frame**

ACTIVATION RECORD

- Activation record is allocated on run-time stack.
- Exists as long as function owning it is executing.
- Consider it as the private repository which provides all the necessary information for proper execution and where to return after this function completes execution.
- Activation records usually have short lifespan because they are allocated on function entry and deallocated upon function exit.

Which Activation Record lives the most?

ACTIVATION RECORD | COMPONENTS

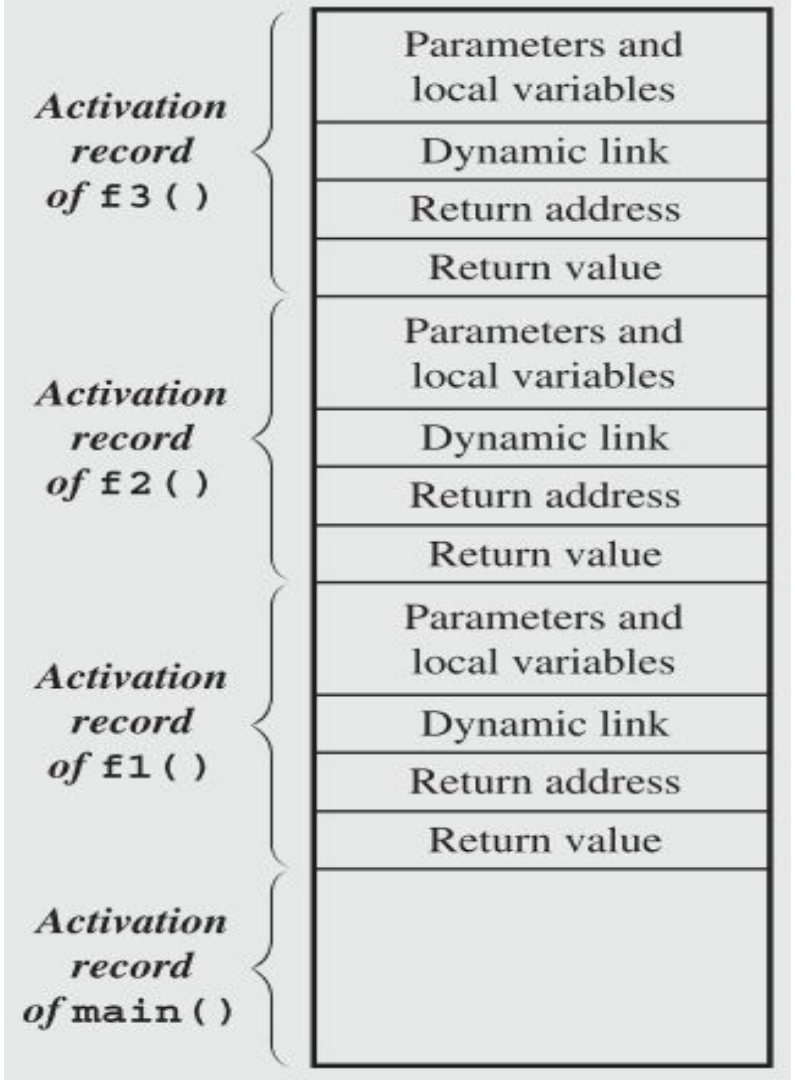
- Values for all parameters to the function.
  - Base address in case of array.
  - Address of variables passed by reference,
  - copies of all data items in case of pass by value.
- Local Variables which can be stored elsewhere in memory.
  - Their pointers will be saved in this case.
- Return Address to resume the control by the caller.
  - The address of the caller’s instruction immediately following the call.

ACTIVATION RECORD | COMPONENTS

- Dynamic Link
  - A pointer to the caller’s activation record.

- Returned Value
  - If function is not declared as void.
  - The size of the activation record may vary from one call to the other.
  - The returned value is placed right above the activation record of the caller.

main ( ) calls f1 ( )
f1 ( ) calls f2 ( )
f2 calls f3 ( )

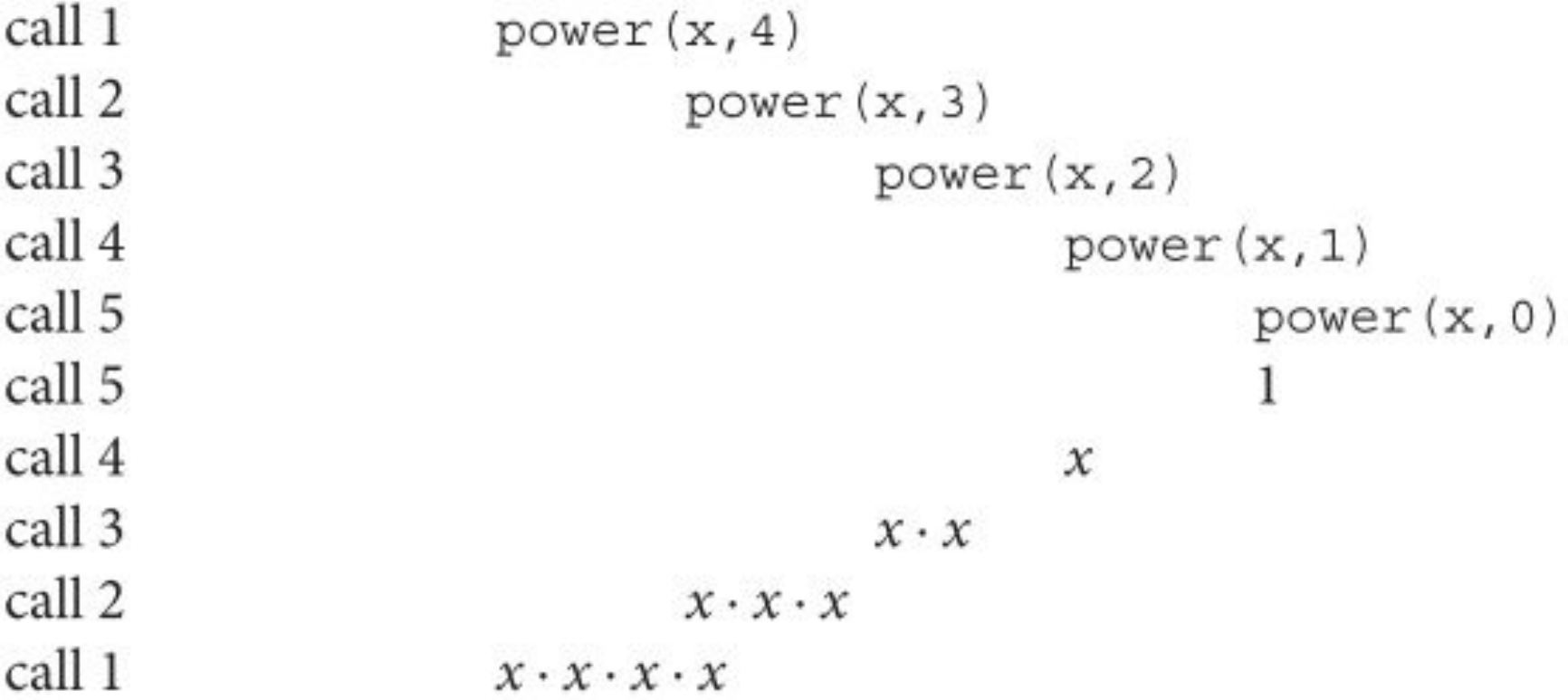


RECURSIVE CALL | AN ANATOMY

$$x^n = \begin{cases} 1 & \text{if } n = 0 \\ x \cdot x^{n-1} & \text{if } n > 0 \end{cases}$$

```
/* 102 */ double power (double x, unsigned int n) {
/* 103 */       if (n == 0)
/* 104 */             return 1.0;
            // else
/* 105 */           return x * power(x,n-1);
          }
```

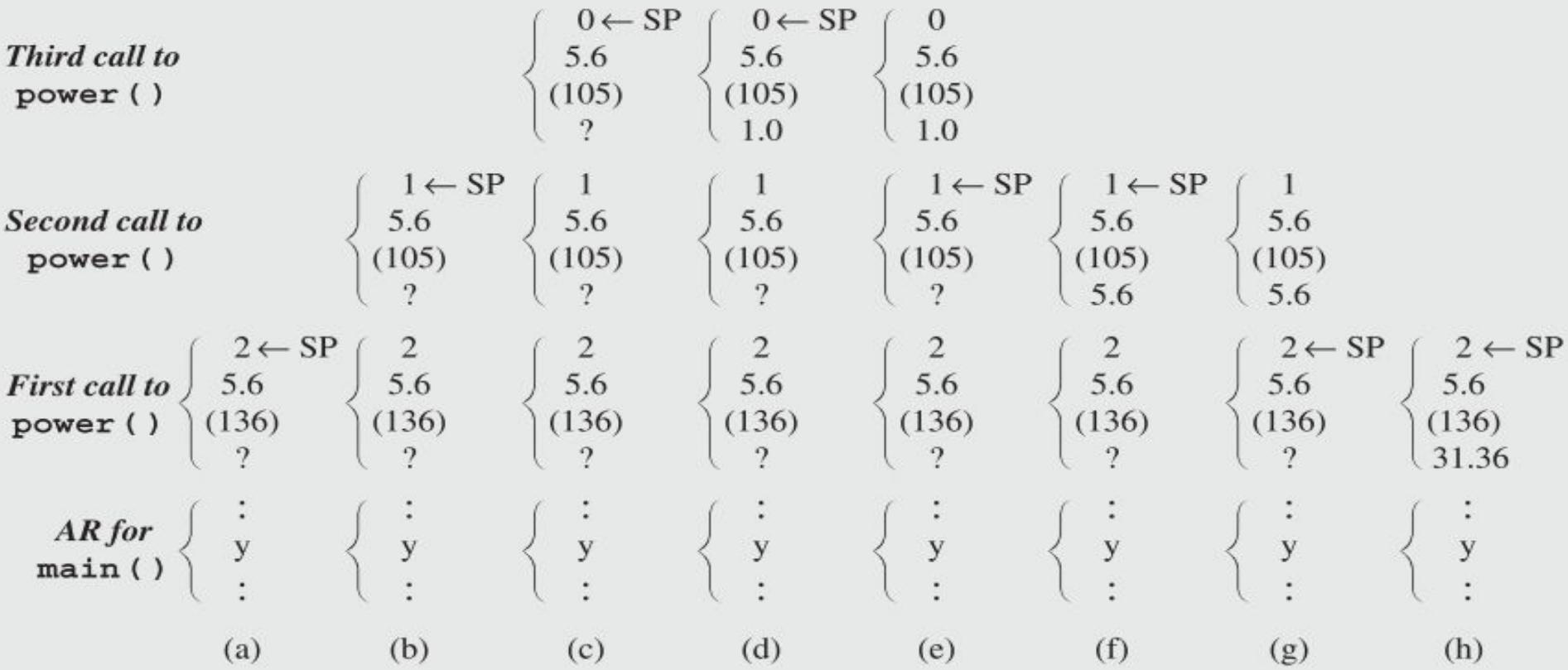
RECURSIVE CALL | AN ANATOMY



RECURSIVE CALL | AN ANATOMY

```
            int main ( ) {
                    …
/* 136 */           y = power(5.6, 2);
                    …
            }
```

```
call 1          power(5.6,2)
call 2                  power(5.6,1)
call 3                          power(5.6,0)
call 3                          1
call 2                  5.6
call 1          31.36
```


Third call to power ( )
Second call to power ( )
First call to power ( )
AR for main ( )
0 ← SP
5.6
(105)
?
1.0
1 ← SP
2 ← SP
(136)
31.36
y
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
Key:
SP Stack pointer
AR Activation record
? Location reserved for returned value

TAIL RECURSION

- Use of only one Recursive Call.
- Recursive call is the last statement of the function.
- There are no earlier recursive calls.
  - Neither Directo nor indirect

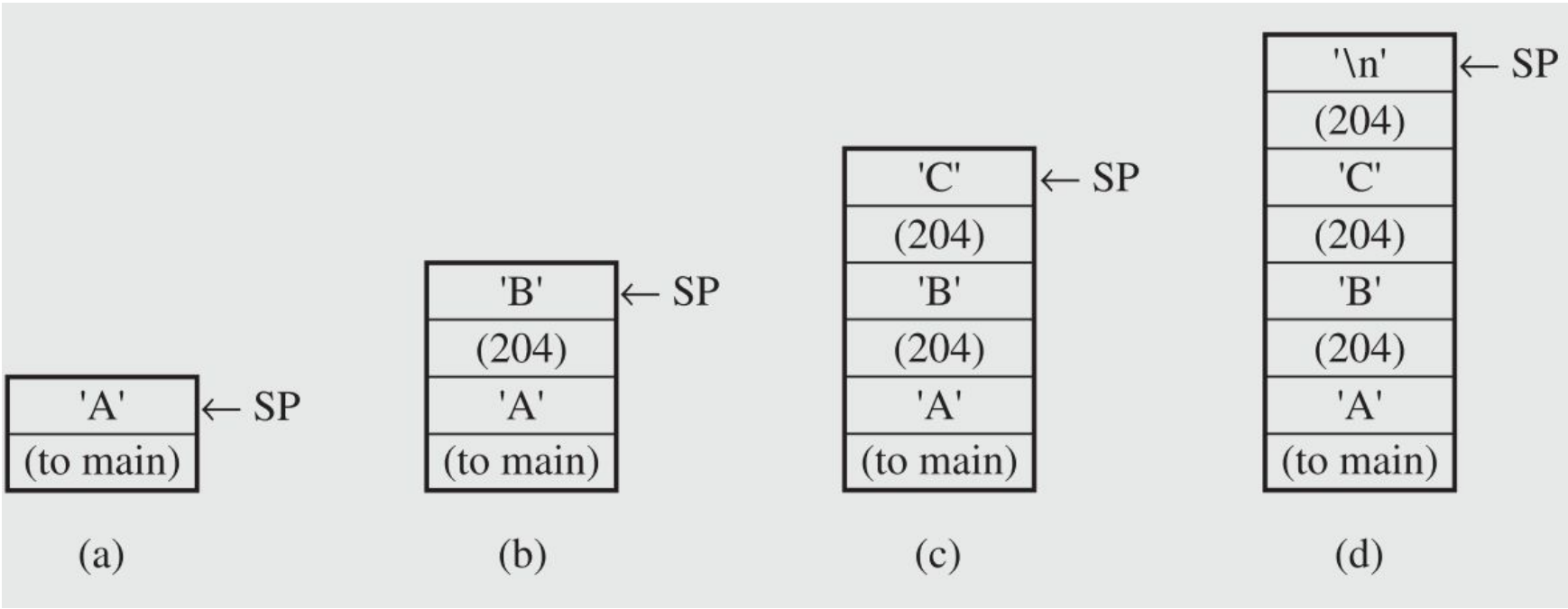
Tail recursion is used in languages like Proglo, which does not have a loop construct.

```
void tail ( int i ) {
        if ( i > 0){
                cout<<i<<' ';
                tail ( i - 1 );
        }
}
```

NON-TAIL RECURSION | INVERT A LINE

```
void reverse( ) {                    /*   200      */
        char ch;
        cin.getch(ch);               /*   201      */
        if ( ch != '\n'){            /*   202      */
                reverse ( ) ;        /*   203      */
                cout.put(ch);        /*   204      */
        }
}
```

NO-TAIL RECURSION



NESTED RECURSION

A function is not only defined in terms of itself, but also is used as one of the parameters.

$$h(n) = \begin{cases} 0 & \text{if } n = 0 \\ n & \text{if } n > 4 \\ h(2 + h(n)) & \text{if } n \leq 4 \end{cases}$$

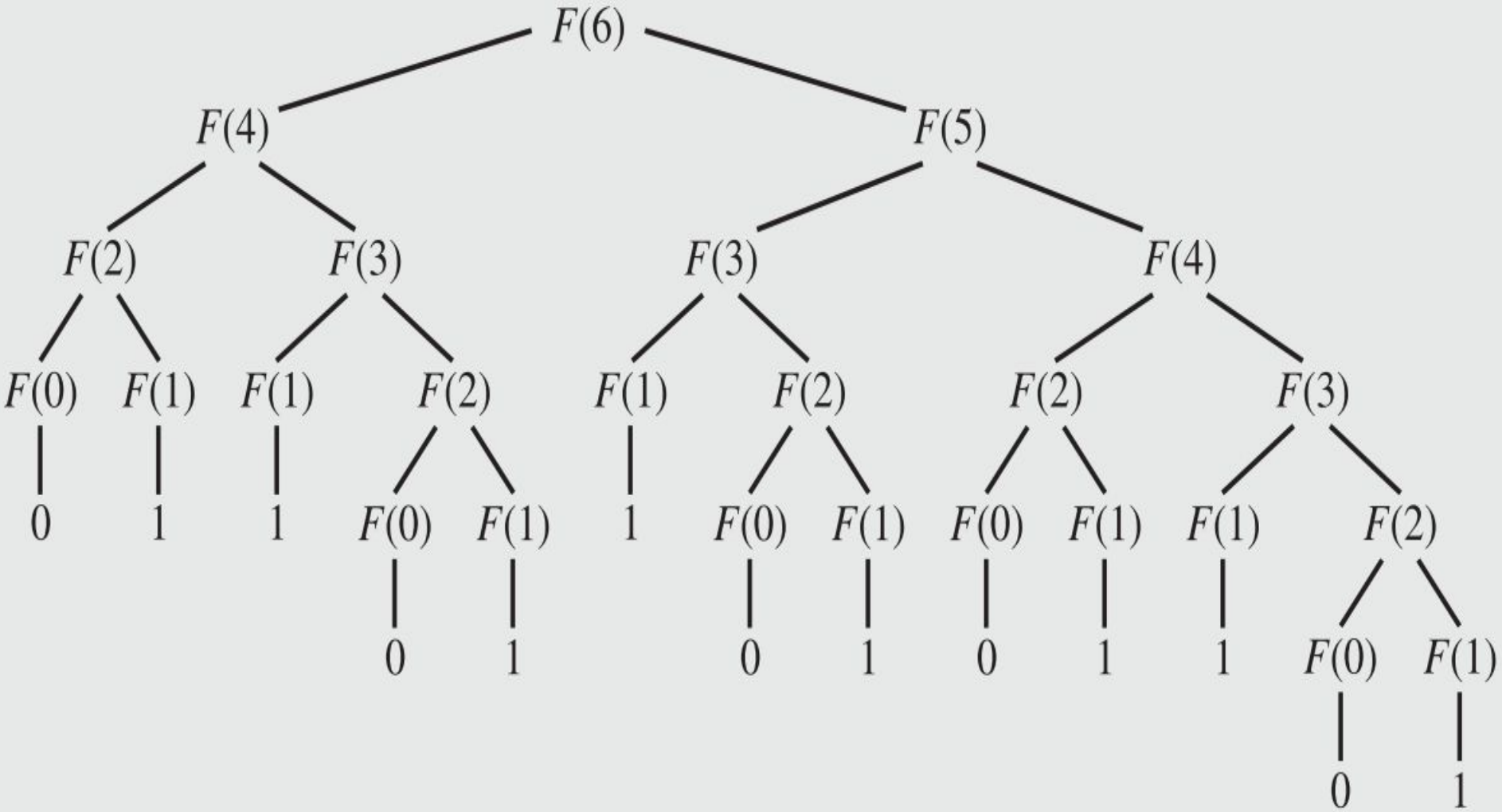
EXCESSIVE RECURSION

- If the recursion is too deep ($5.6^{100,000}$), we will run out of space on stack.
  - Program crashes.
  - In case of power range of variable might not allow such a big number as well.
- But what if some recursive functions repeat themselves for a very long time even for very simple cases.
  - Consider the case of Fibonacci numbers.

$$\text{Fib}(n) = \begin{cases} n & \text{if } n < 2 \\ \text{Fib}(n-2) + \text{Fib}(n-1) & \text{otherwise} \end{cases}$$

EXCESSIVE RECURSION

```
unsigned long Fib ( unsigned long n ) {
        if (n < 2)
                return n;
        else
                return Fib (n-2) + Fib (n-1);
}
```


F(6)
F(4)
F(5)
F(2)
F(3)
F(3)
F(4)
F(0)
F(1)
F(1)
F(2)
F(1)
F(2)
F(2)
F(3)
0
1
1
F(0)
F(1)
1
F(0)
F(1)
F(0)
F(1)
F(1)
F(2)
0
1
0
1
0
1
1
F(0)
F(1)
0
1

NUMBER OF ADDITIONS Vs NUMBER OF CALLS

n	Fib(n+1)	Number of Additions	Number of Calls
6	13	12	25
10	89	88	177
15	987	986	1,973
20	10,946	10,945	21,891
25	121,393	121,392	242,785
30	1,346,269	1,346,268	2,692,537

NUMBER OF ADDITIONS Vs NUMBER OF CALLS

```
unsigned long FibIterative (unsigned long n) {
        if (n < 2 )
                return n;
        else {
                long i = 2, temp, current = 1, last =0;
                for (; i <= n; ++){
                        temp = current;
                        current += last;
                        last = temp;
                }
        return current;
}
```

ACTIVITY

Check Recursively If given

Word is Palindrome?
Sentence is Palindrome ignoring the space?

Given a C++ Program check if all the opening brackets

({ [< have their corresponding closing >] })

Find an element in 2D DynamicArray