

Recursion

Review

- role of functions
- scoping rules
- storage classes
 - auto enforces the scoping rule

Recursion

- A function calls itself.
- A recursive function can be converted to an iterative one

```
int fact(int n)
/* recursive version */
{
    if (n <= 1)
        return 1;
    else
        return (n * fact(n-1));
}
```

```
int fact(int n)
/* iterative version */
{
    int product = 1;
    for ( ; n > 1; --n)
        product *=n;
    return product;
}
```

Recursion

Recursion is said to be more elegant and requires fewer variables, but function calls are costly in time and space.

Recursion

- Fibonacci numbers

$$- f_{i+1} = f_i + f_{i-1}$$

```
int fib(int n)
{
    if (n <= 1)
        return n;
    else
        return (fib(n-1) + fib(n-2));
}
```

- Exponential increase in function calls

n	fib(n)	number of recursive calls
0	0	1
1	1	1
2	1	3
3	2	5
4	3	9
...
23	28657	92735
24	43368	150049
...
42	267914296	866988873
43	433494437	1402817465
...

Recursion

- iterative version

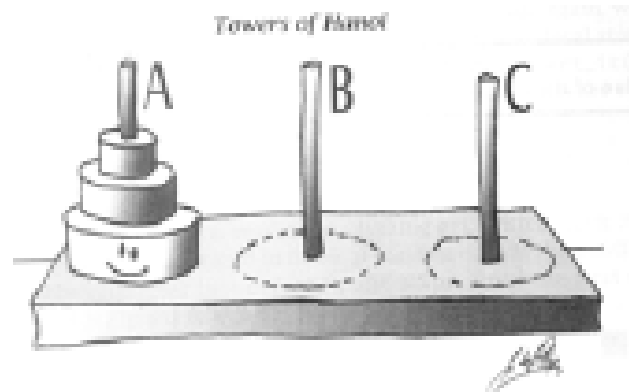
```
int f[n+1];

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

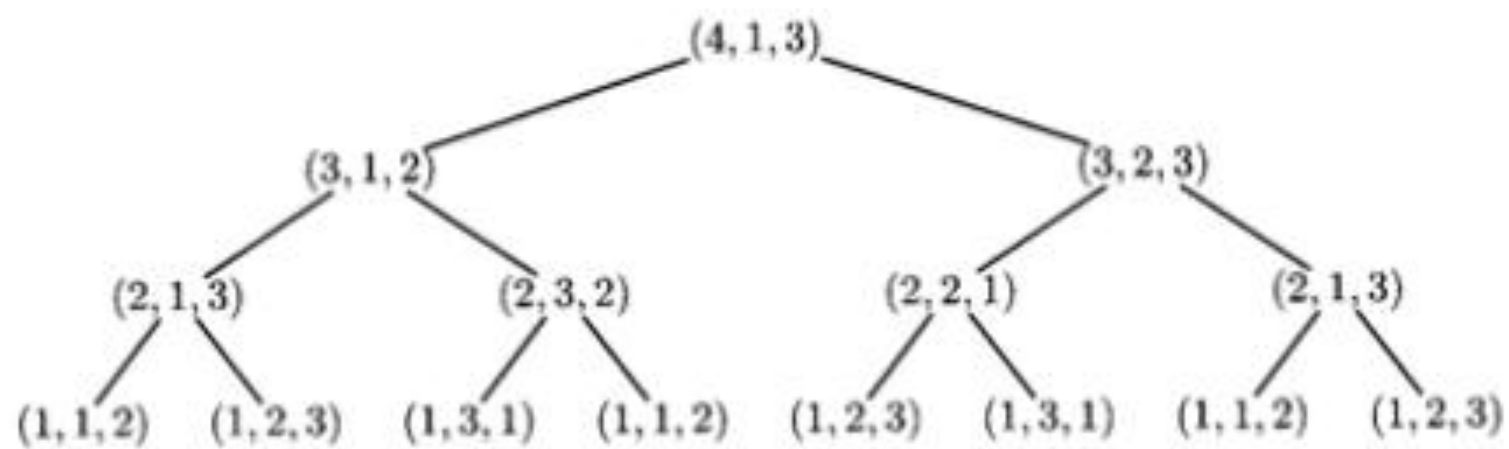
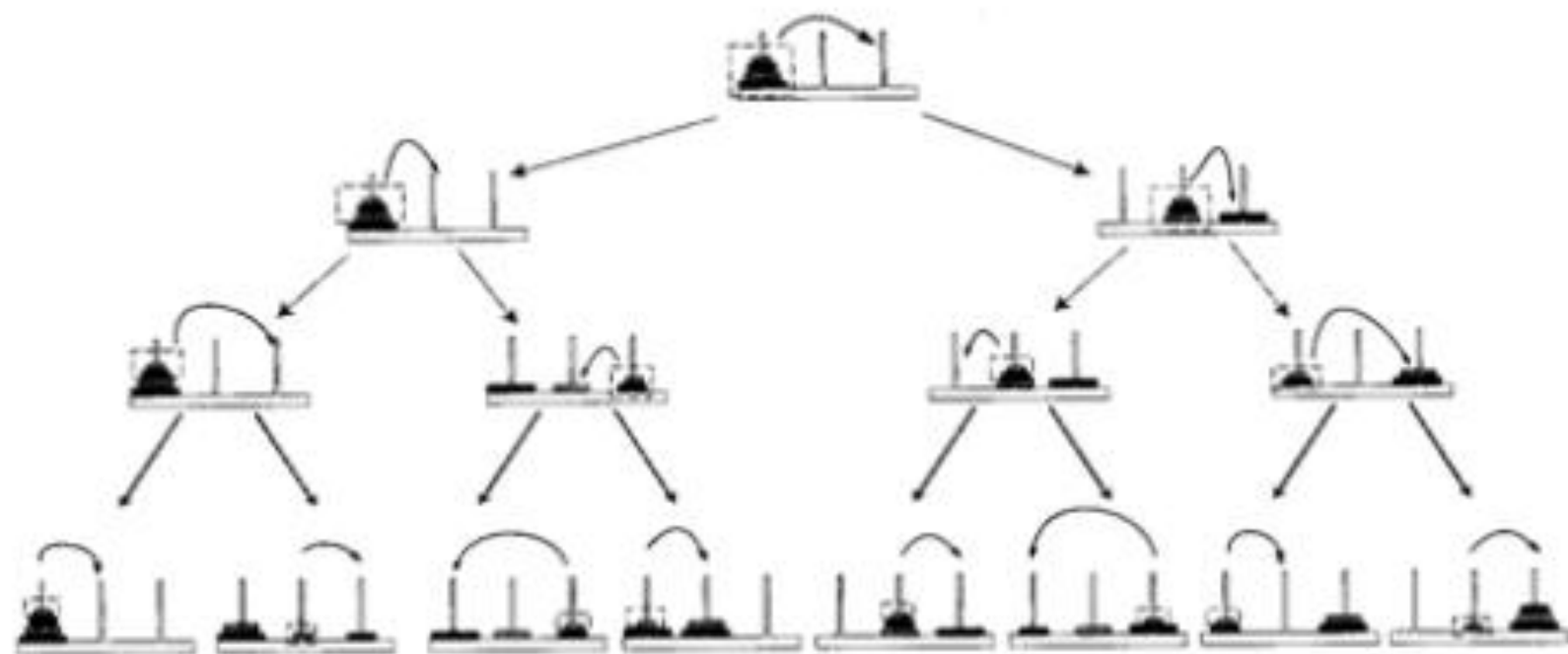
write backward

```
/* Write a line backwards. */  
  
#include <stdio.h>  
  
void wrt_it(void);  
  
int main(void)  
{  
  
    printf("Input a line: ");  
    wrt_it();  
    printf("\n\n");  
    return 0;  
}  
  
void wrt_it(void)  
{  
    int c;  
  
    if ((c = getchar()) != '\n')  
        wrt_it();  
    putchar(c);  
}
```

Towers of Hanoi



- only one disk can be moved at a time
- after each move, the disks on each tower must be in order
- move disks on A to C



Arrays, Pointers, and Strings

One Dimensional Array

- An array is a set of subscripted variables of the same type
- In C, arrays and pointers are interrelated
 - array name is a pointer
- pointer parameters can implement “call-by-reference”
- arrays can be initialized

```
int a[100] = {0};
```
- external or static arrays are initialized to zero by default
- array declaration without size

```
int a[] = {2, 3, 4, 7};  
char s[] = "abc";
```

Pointers

- address
- pointers take addresses as values
 - `NULL == 0 == FALSE`
- Usage
 - `p = & a;`
 - `b = *p;`
 - `v == *&v` (for any variable `v`)
 - `p == &*p` (if `p` is a pointer)

call-by-reference

```
#include <stdio.h>

void swap(int *, int *);

int main(void)
{
    int i = 3, j = 5;

    swap(&i, &j);
    printf("%d %d\n", i, j);
    /* 5 3 is printed */
    return 0;
}
```

```
void swap(int *p, int *q)
{
    int tmp;

    tmp = *p;
    *p = *q;
    *q = tmp;
}
```

swap

Example: swapping values of variables in a calling environment by using pointers called-by-value to achieve call-by-reference effect

Arrays and Pointers

- If the variable `p` is a pointer to a type(say, integer) then `p+1` yields the address for the next variable of that type

```
int i;          /* 4 bytes integer */  
int a[ ];      /* array or pointer */
```

- `a[i] == *(a+i)`
- `(a + i) == &a[i]`