```
* university of tehran
  school of ece
 * advanced programming
 * ramtin khosravi
#define LECTURES 4,5
int main()
   cout << "Functions, "</pre>
         << "Recursion\n";</pre>
```

Call-by-Value

```
1. void f(int x)
                                          main()
2. {
                                                   10
3. int i = 5;
4. x = i + 1;
5. }
6. int main()
7. {
      int a = 10;
     f(a);
     cout << a << "\n";
10.
11. }
```

Call-by-Value

```
1. void f(int x)
2. {
3. int i = 5;
4. x = i + 1;
5. }
6. int main()
7. {
8. int a = 10;
9. f(a);
10. cout << a << "\n";
11. }</pre>
```

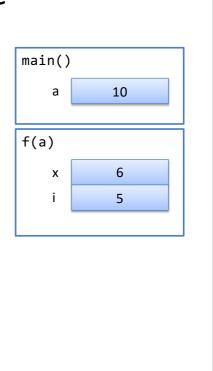
```
main()
a 10

f(a)
x 10
i ?
```

3

Call-by-Value

```
1. void f(int x)
2. {
3.    int i = 5;
4.    x = i + 1;
5. }
6. int main()
7. {
8.    int a = 10;
9.    f(a);
10.    cout << a << "\n";
11. }</pre>
```



Call-by-Reference

```
1. void f(int x, int& y)
2. {
3.
      int i = 5;
     x = i + 1;
5.
     y = 18;
6. }
7. int main()
8. {
9.
      int a = 10;
10.
      int b = 11;
11.7
      f(a, b);
      cout << a << b;
12.
13. }
```

```
main()

a 10
b 11 A<sub>1</sub>
```

5

Call-by-Reference

```
    void f(int x, int& y)

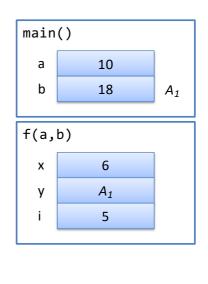
2. {
3.\Rightarrow int i = 5;
     x = i + 1;
5.
      y = 18;
6. }
7. int main()
8. {
9.
       int a = 10;
      int b = 11;
10.
11. f(a, b);
     cout << a << b;
12.
13. }
```

```
main()
a 10
b 11
A<sub>1</sub>

f(a,b)
x 10
y A<sub>1</sub>
i ?
```

Call-by-Reference

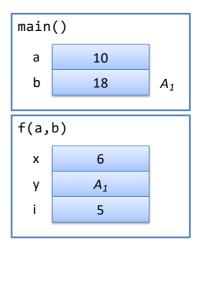
```
1. void f(int x, int& y)
2. {
3.
      int i = 5;
     x = i + 1;
      y = 18;
7. int main()
8. {
9.
      int a = 10;
10. int b = 11;
11.
     f(a, b);
12.
     cout << a << b;
13. }
```



7

Call-by-Reference

```
1. void f(int x, int& y)
2. {
3. int i = 5;
4.
      x = i + 1;
      y = 18;
7. int main()
8. {
9.
      int a = 10;
     int b = 11;
10.
11. f(a, b);
     cout << a << b;
12.
13. }
```



Is it possible to write f(a, a+1)?

Nested Calls

```
    void g(int& y) {
    int j = 2;
    y = j * 3;
    }
    void f(int x) {
    g(x);
    }
    int main() {
    int a = 10;
    f(a);
    cout << a << "\n";</li>
    }
```

```
main()
a 10

f(a)
x 10 A<sub>1</sub>
```

9

Nested Calls

```
1. void g(int& y) {
2.     int j = 2;
3.     y = j * 3;
4. }
5. void f(int x) {
6.     g(x);
7. }
8. int main() {
9.     int a = 10;
10.     f(a);
11.     cout << a << "\n";
12. }</pre>
```

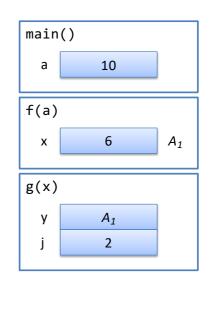
```
main()
a 10

f(a)
x 10 A<sub>1</sub>

g(x)
y A<sub>1</sub>
j ?
```

Nested Calls

```
1. void g(int& y) {
2. int j = 2;
     y = j * 3;
5. void f(int x) {
6. g(x);
7. }
8. int main() {
9. int a = 10;
10.
     f(a);
     cout << a << "\n";
11.
12. }
```



11

```
Nested Calls
1. void g(int& y) {
                                          main()
2.
    int j = 2;
                                                 10
3. y = j * 3;
4. }
                                          f(a)
                                               6
                                                        A_1
5. void f(int x) {
6. g(x);
                                          g(x)
7. }
                                                 A_1
8. int main() {
                                               2
      int a = 10;
9.
10. f(a);
11.
     cout << a << "\n";
12. }
                        What happens if we have void f(int& x)?
```

Accessing Variables

```
1. void f(int n) { n=7; }
                                       movl $0x7, 0x8(%ebp)
2. void f(int& n) { n=7; }
                                       mov 0x8(%ebp), %eax
                                       movl $0x7, (%eax)
3. int n;
   void f() { n=7; }
                                      movl $0x7,0x804a01c
```

13

Recursive Calls

```
1. void f(int n) {
                                         main()
      if (n == 1)
          return;
                                         f()
4. f(n - 1);
                                             3
5. }
6. int main() {
7. f(3);
8. }
```

Recursive Calls

```
1. void f(int n) {
2.    if (n == 1)
3.        return;
4.    f(n - 1);
5. }
6. int main() {
7.    f(3);
8. }
```

```
f()
n 3

f()
n 2
```

15

Recursive Calls

```
1. void f(int n) {
2.    if (n == 1)
3.        return;
4.    f(n - 1);
5. }
6. int main() {
7.    f(3);
8. }
```

```
f()
n 3

f()
n 2

f()
n 1
```

Recursive Calls

```
f()
n 3

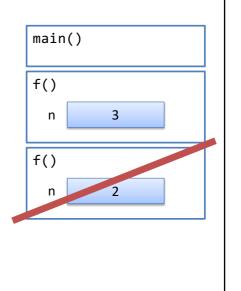
f()
n 2

f()
n 1
```

17

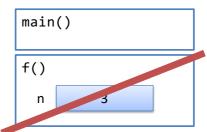
Recursive Calls

```
1. void f(int n) {
2.    if (n == 1)
3.        return;
4.    f(n - 1);
5  }
6. int main() {
7.    f(3);
8. }
```



Recursive Calls

```
1. void f(int n) {
2.    if (n == 1)
3.        return;
4.    f(n - 1);
5 → }
6. int main() {
7.    f(3);
8. }
```



19

```
1. int fact(int n) {
2.    if (n <= 1)
3.        return 1;
4.    int f = fact(n - 1);
5.    return f * n;
6. }

7. int main() {
8.    int a = 3;
9.    int b = fact(a);
10. }</pre>
```

```
main()

a 3
b ?
```

```
1. int fact(int n) {
2.    if (n <= 1)
3.        return 1;
4.    int f = fact(n - 1);
5.    return f * n;
6. }

7. int main() {
8.    int a = 3;
9.    int b = fact(a);
10. }</pre>
```

21

```
1. int fact(int n) {
2.     if (n <= 1)
3.         return 1;
4.     int f = fact(n - 1);
5.     return f * n;
6. }

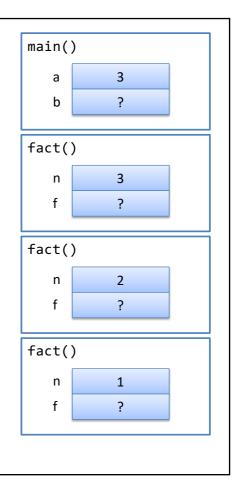
7. int main() {
8.     int a = 3;
9.     int b = fact(a);
10. }</pre>
```

```
1. int fact(int n) {
2.
      if (n <= 1)
            return 1;
3.
   int f = fact(n - 1);
4.
     return f * n;
5.
6. }
7. int main() {
8.
      int a = 3;
      int b = fact(a);
9.
10.}
```

23

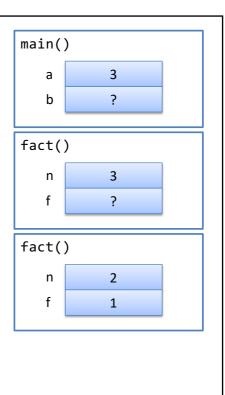
```
1. int fact(int n) {
2.     if (n <= 1)
3.         return 1;
4.     int f = fact(n - 1);
5.     return f * n;
6. }

7. int main() {
8.     int a = 3;
9.     int b = fact(a);
10. }</pre>
```



```
1. int fact(int n) {
2.    if (n <= 1)
3.        return 1;
4.    int f = fact(n - 1);
5.    return f * n;
6. }

7. int main() {
8.    int a = 3;
9.    int b = fact(a);
10. }</pre>
```



25

```
1. int fact(int n) {
2.    if (n <= 1)
3.        return 1;
4.    int f = fact(n - 1);
5.    return f * n;
6. }

7. int main() {
8.    int a = 3;
9.    int b = fact(a);
10. }</pre>
```

```
main()

a 3
b 6
```

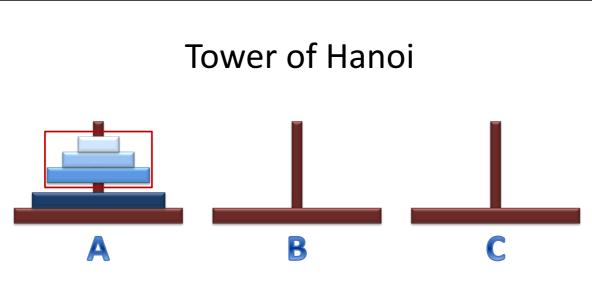
```
1. int fact(int n) {
2.    if (n <= 1)
3.        return 1;
4.    int f = fact(n - 1);
5.    return f * n;
6. }

7. int main() {
8.    int a = 3;
9.    int b = fact(a);
10. }</pre>
```

27

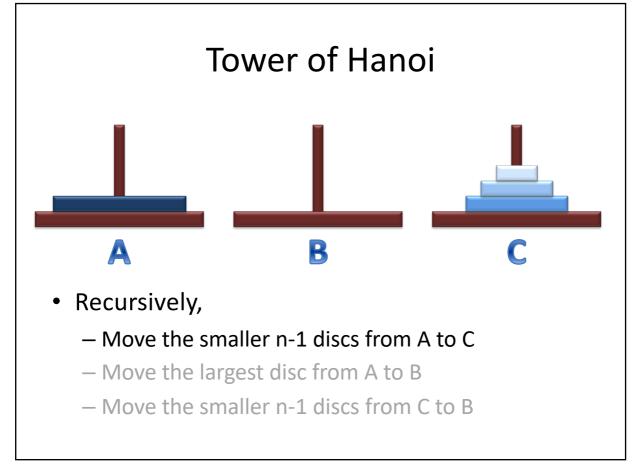
Tower of Hanoi A B C

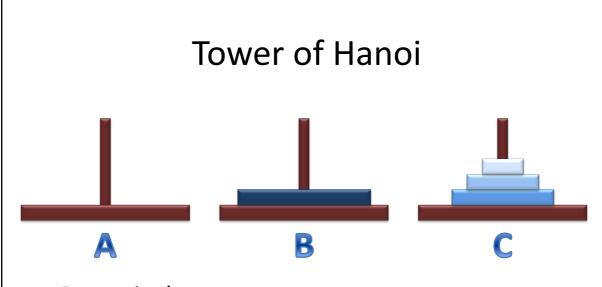
- Move the blue discs from A to B
 - Move one disc at a time
 - Never put a larger disc on a smaller one
- Originally designed by Eduard Lucas (1883) for 8 discs



- Recursively,
 - Move the smaller n-1 discs from A to C
 - Move the largest disc from A to B
 - Move the smaller n-1 discs from C to B

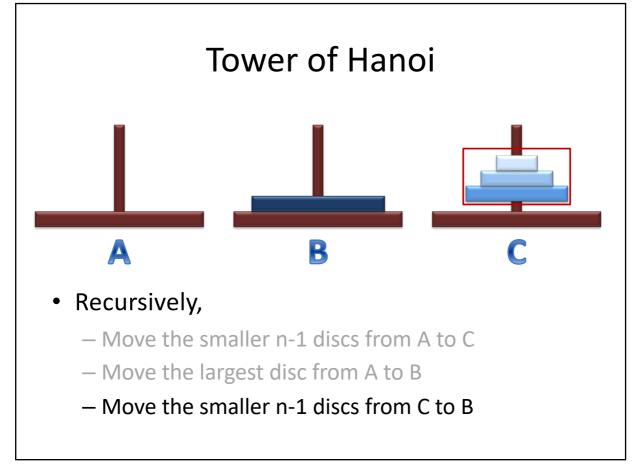
29

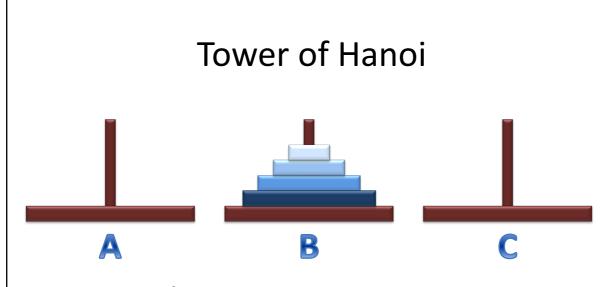




- Recursively,
 - Move the smaller n-1 discs from A to C
 - Move the largest disc from A to B
 - Move the smaller n-1 discs from C to B

31





- Recursively,
 - Move the smaller n-1 discs from A to C
 - Move the largest disc from A to B
 - Move the smaller n-1 discs from C to B

33

Tower of Brahma

- God initially placed 64 golden discs
- Ordered a group of priests to move them according to the rules described before
- When they finish, the tower will crumble and the world will end
- Q: When the world will end?

Calculating Number of Moves

• T(n) = number of moves required to move n discs

•
$$T(n) = 2 T(n-1) + 1$$

• $U(n) = T(n) + 1$
 $\Rightarrow U(n) = 2 T(n-1) + 2 = 2 U(n-1)$
 $\Rightarrow U(n) = 2^n$

 \Rightarrow T(n) = $2^n - 1$

35

The priests are still working!

- 2⁶⁴-1 is about 18 quintillions
- If they move a disc in one microsecond(!) Still 5000 centries to go!

Recursive Computation on Lists

- Finding the smallest number in a list
- min_list([a₁, a₂, ..., a_n]) =
 min(a₁, min_list([a₂, a₃, ..., a_n]))
- min_list([a₁]) = a₁

37

Recursive Computation on Lists

- Finding the sum of the numbers of a list
- sum_list([a₁, a₂, ..., a_n]) = a₁ + sum_list([a₂, a₃, ..., a_n])
- **sum_list**([]) = 0

Recursive Computation on Lists

- Sort a list of numbers in ascending order
- Find the smallest number in the list
- Swap the first element with the smallest
- Sort the rest of the array
- This method is called selection sort