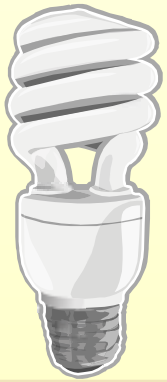
Recursion vs Iteration

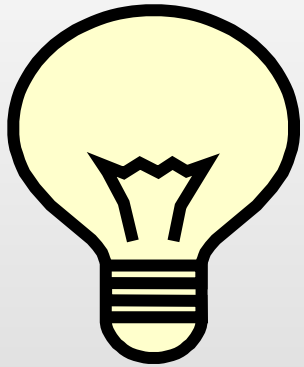
```
int fib(int n)
{
   int first = 0, second = 1;
   int next, c;
   if (n <= 1)
      return n;
   for ( c = 1; c < n ; c++ ) {
      next = first + second;
      first = second;
      second = next;
    }
    return next;
}
```



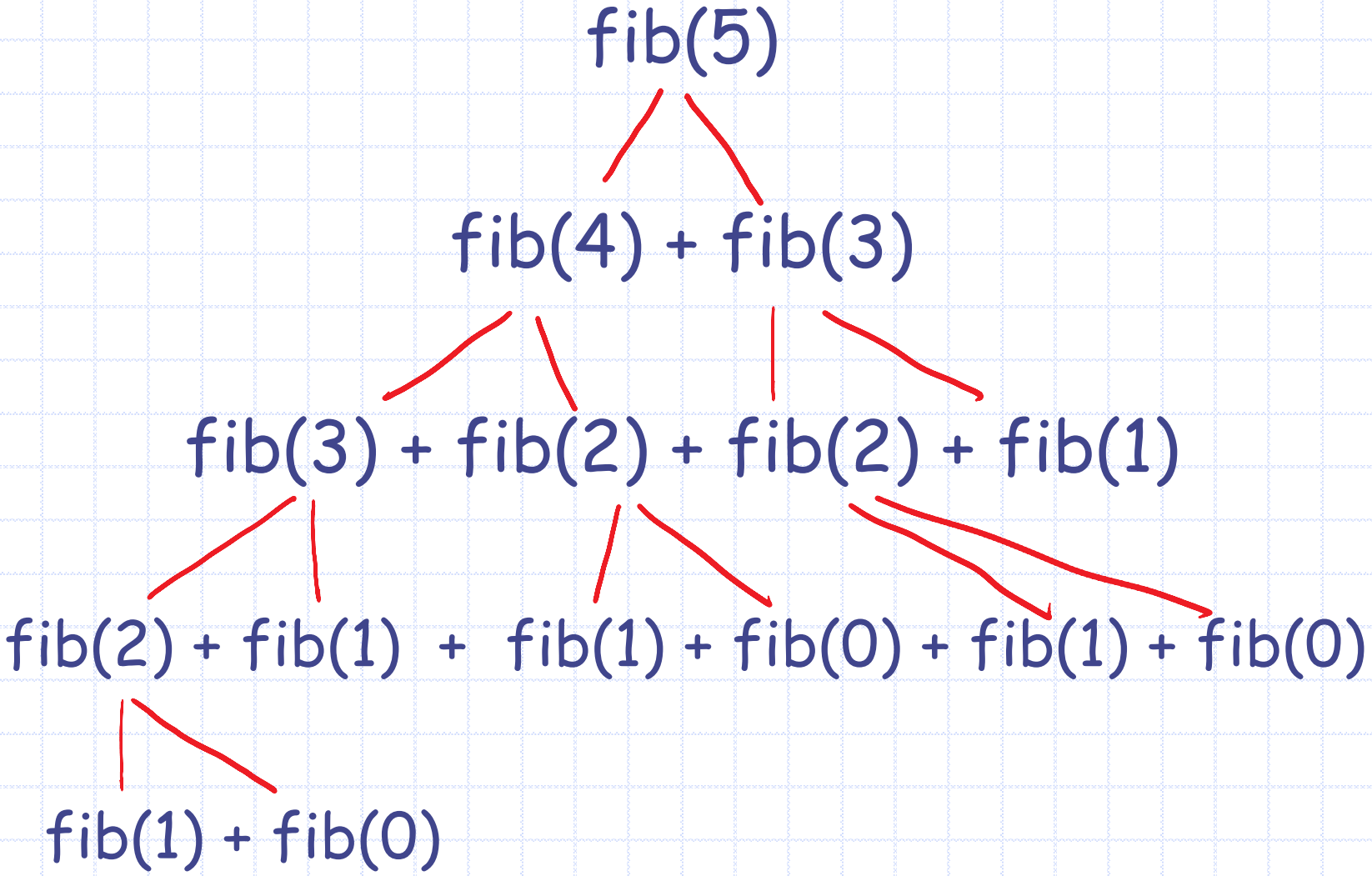
The recursive program is closer to the definition and easier to read.

But very very inefficient

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



Recursive fib



Rec. fib: How fast #calls grow

```
#include<stdio.h>
int count = 0; /*Global: #fib calls */

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

int main() {
  int num, res;
  for (num=5; num<=30; num=num+5) {
    count = 0; /* reset the count*/
    res = fib(num);
    printf("%d, %d\n", res, count);
  }
  return 0;
}
```

num	fib(num)	Count
5	5	15
10	55	177
15	610	1973
20	6765	21891
25	75025	242785
30	832040	2692537

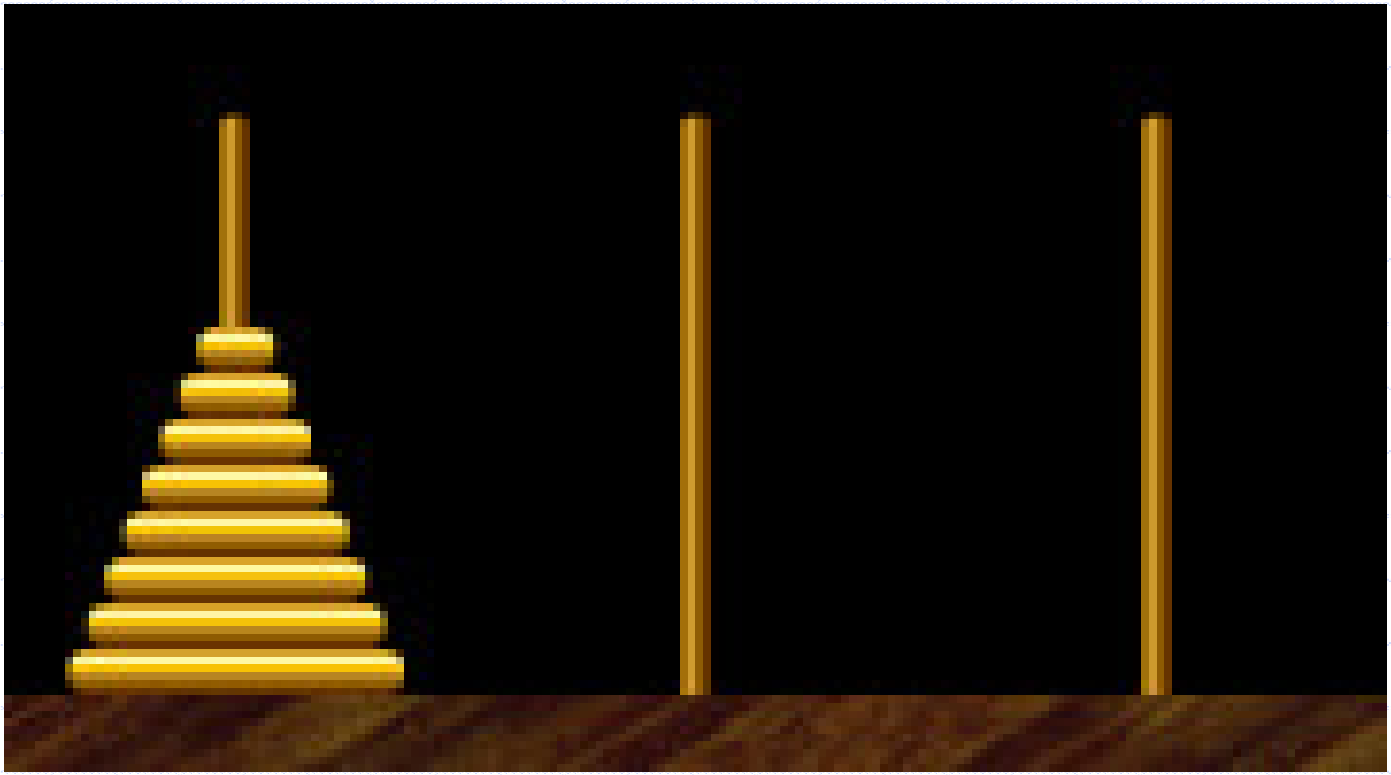
Recursion: Summary

- Advantages
  - Elegant. Solution is cleaner.
  - Fewer variables.
  - Once the recursive definition is figured out, program is easy to implement.
- Disadvantages
  - Debugging can be considerably more difficult.
  - Figuring out the logic of the recursive function is not easy sometimes.
  - Can be inefficient (requires more time and space), if not implemented carefully.

Around Easter 1961, a course on ALGOL 60 was offered … It was there that I first learned about recursive procedures and saw how to program the sorting method which I had earlier found such difficulty in explaining. It was there that I wrote the procedure, immodestly named QUICKSORT, on which my career as a computer scientist is founded. Due credit must be paid to the genius of the designers of ALGOL 60 who **included recursion in their language** and enabled me to describe my invention so elegantly to the world. I have regarded it as the highest goal of programming language design to enable good ideas to be elegantly expressed.

- **The Emperor's Old Clothes, C. A. R. Hoare, ACM Turing Award Lecture, 1980**

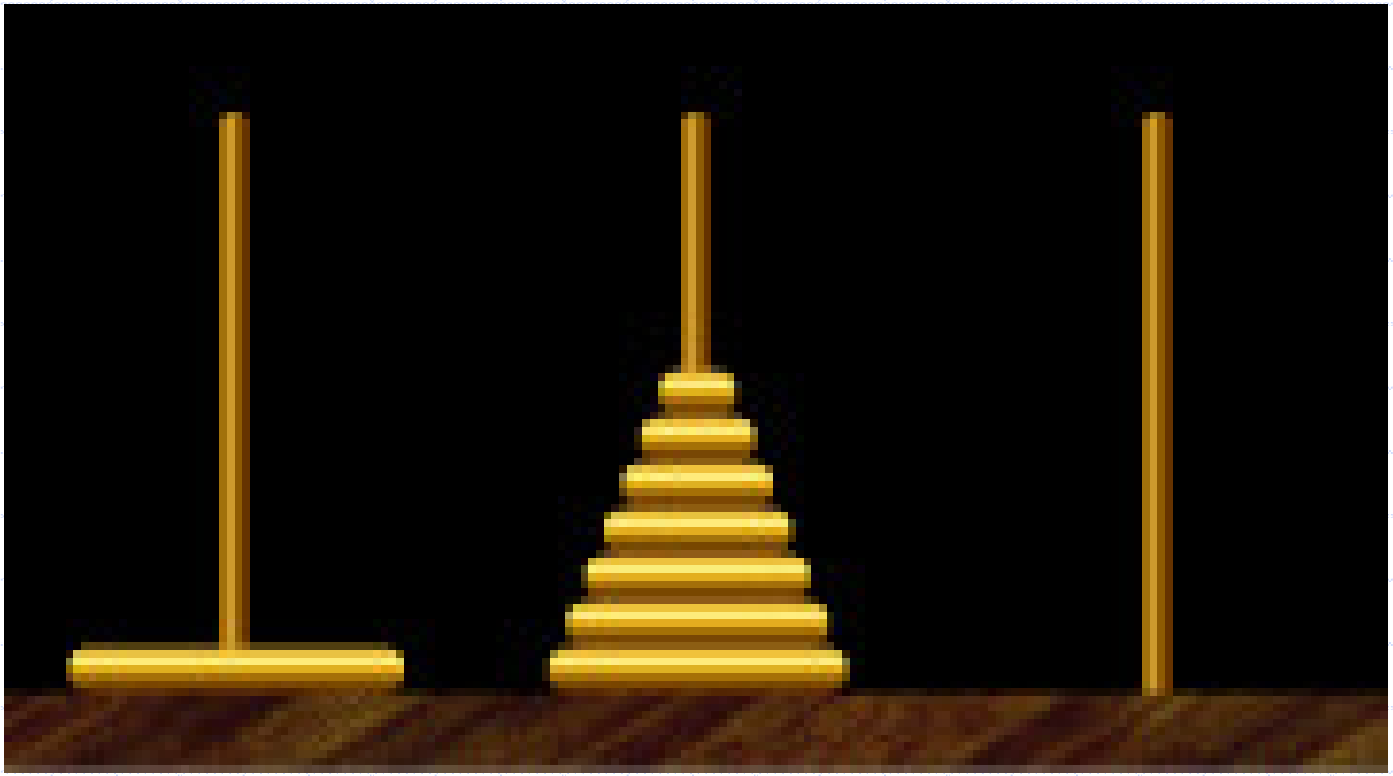
Recursion : Tower of Hanoi



No disk may be placed on top of a smaller disk.

Image Source:
http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

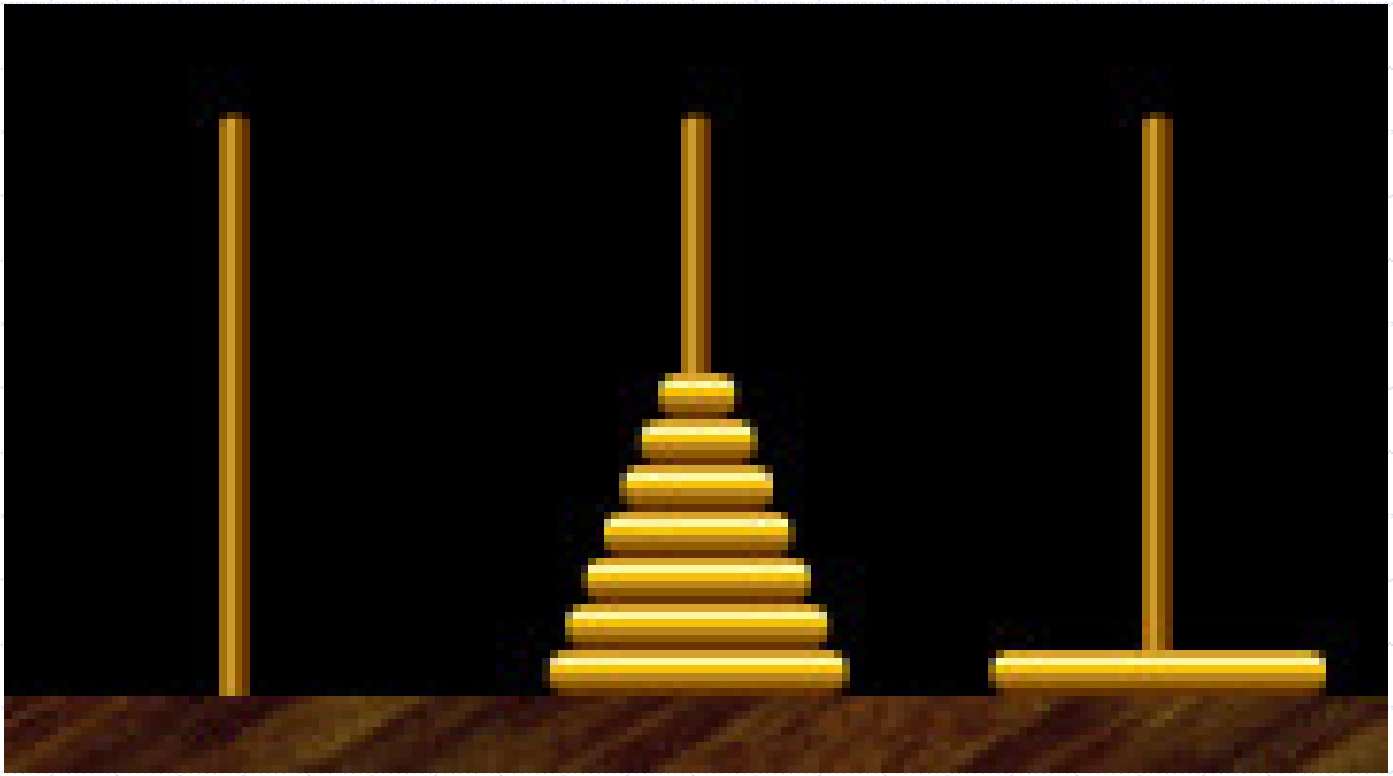
Recursion : Tower of Hanoi ..2



No disk may be placed on top of a smaller disk.

Image Source:
http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

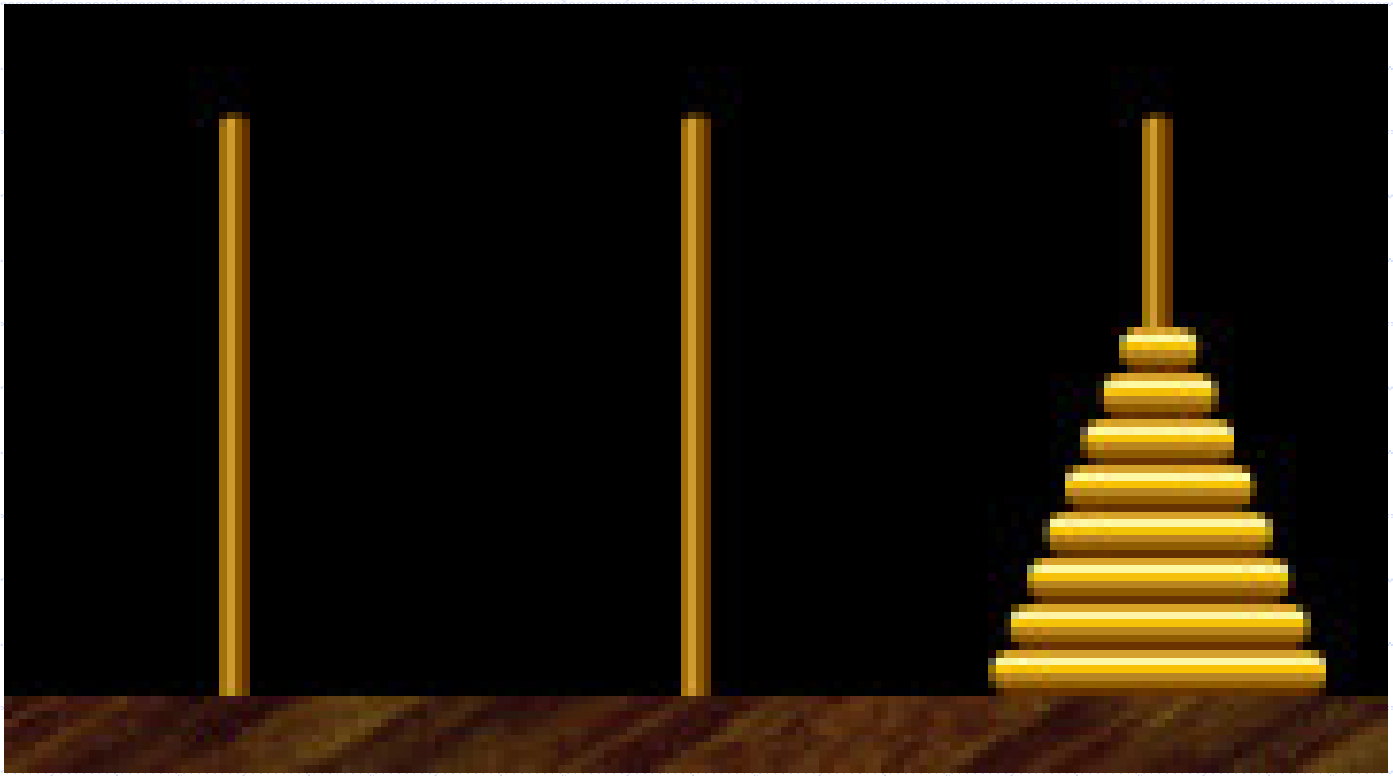
Recursion : Tower of Hanoi ..3



No disk may be placed on top of a smaller disk.

Image Source:
http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

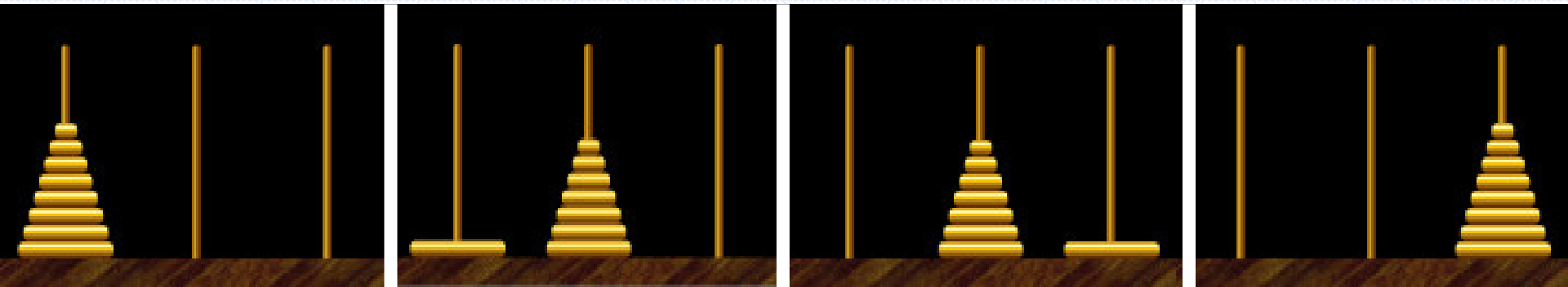
Recursion : Tower of Hanoi ..4



No disk may be placed on top of a smaller disk.

Image Source:
http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

```
// move n disks From A to C using B as auxx
void hanoi(int n, char A, char C, char B) {
  if (n==0) { return; } // nothing to move!!
   // recursively move n-1 disks
   // from A to B using C as auxx
   hanoi(n-1, A, B, C);
   // atomic move of a single disk
   printf("Move 1 disk from %c to %c\n", A, C);
   // recursively move n-1 disks
   // from B to C using A as auxx
   hanoi(n-1, B, C, A);
}
```



OUTPUT for hanoi(4, 'A', 'C', 'B')

Move 1 disk from A to B
Move 1 disk from A to C
Move 1 disk from B to C
Move 1 disk from A to B
Move 1 disk from C to A
Move 1 disk from C to B
Move 1 disk from A to B
Move 1 disk from A to C
Move 1 disk from B to C
Move 1 disk from B to A
Move 1 disk from C to A
Move 1 disk from B to C
Move 1 disk from A to B
Move 1 disk from A to C
Move 1 disk from B to C

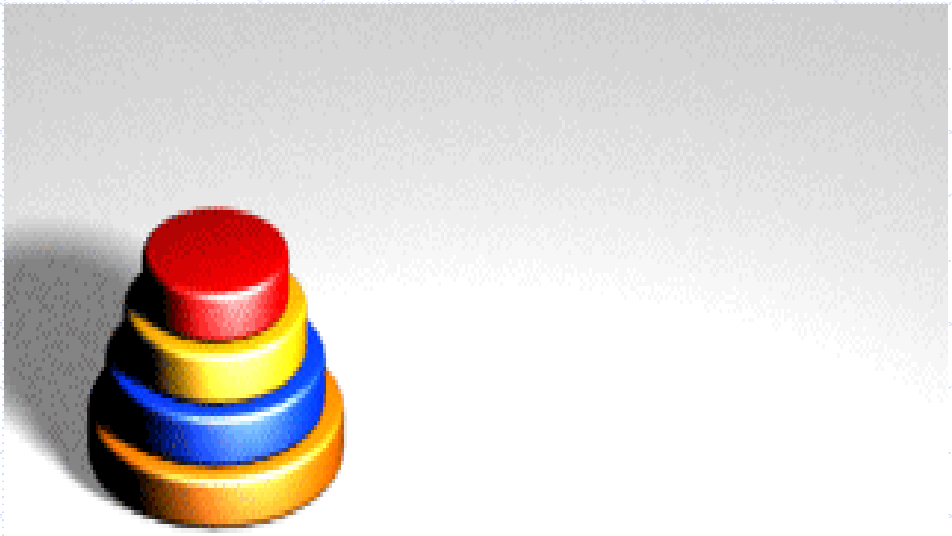


Image Source:
http://upload.wikimedia.org/wikipedia/commons/6/60/Tower_of_Hanoi_4.gif

The puzzle was invented by the French mathematician **Édouard Lucas** in 1883.

There is a story about a temple in Kashi Vishwanath, which contains a large room with three posts surrounded by 64 golden disks. Brahmin priests have been moving these disks, in accordance with the immutable rules of the Brahma. The puzzle is therefore also known as the **Tower of Brahma** puzzle.

According to the legend, when the last move of the puzzle will be completed, the world will end.

If the legend were true, and if the priests were able to move disks at a rate of one per second, using the smallest number of moves, it would take them $2^{64}-1$ seconds or roughly 585 billion years or about 127 times the current age of the sun.

Source:
https://en.wikipedia.org/wiki/Tower_of_Hanoi

ESC101: Introduction to Computing

Pointers

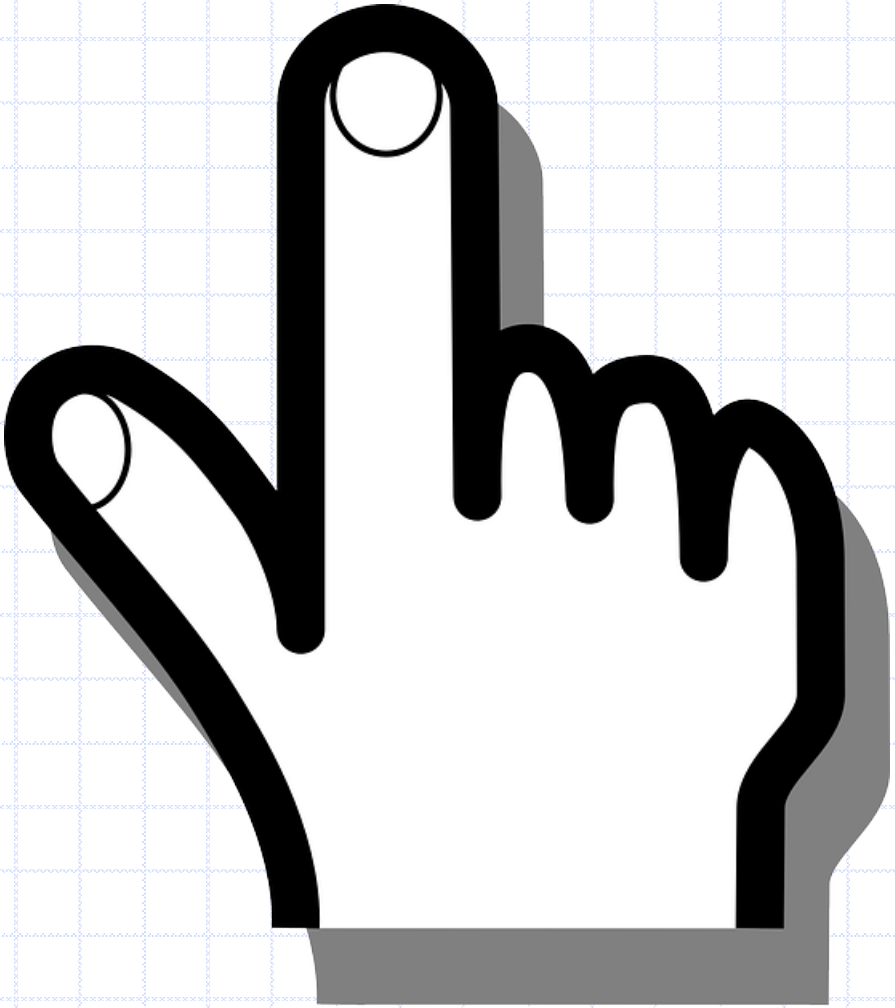
Pointer: Dictionary Definition

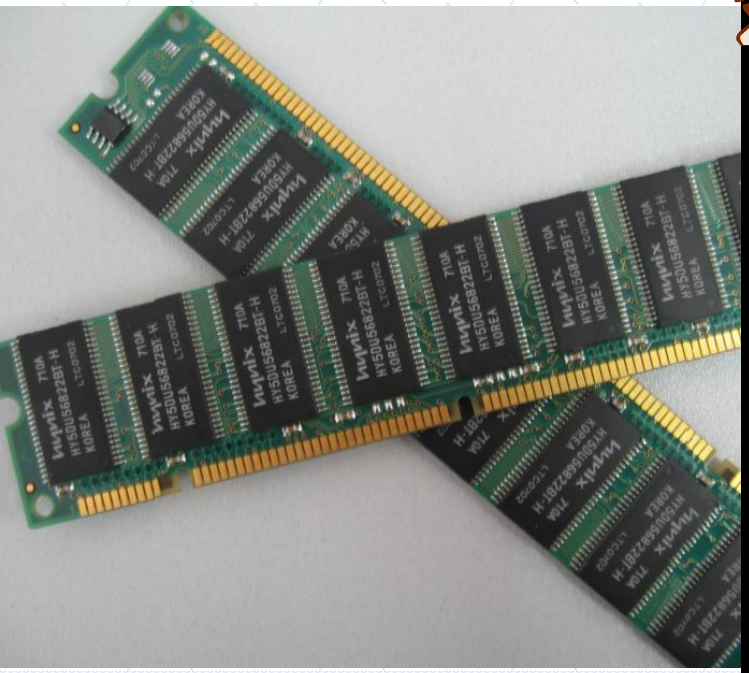
point·er (poin'tər)

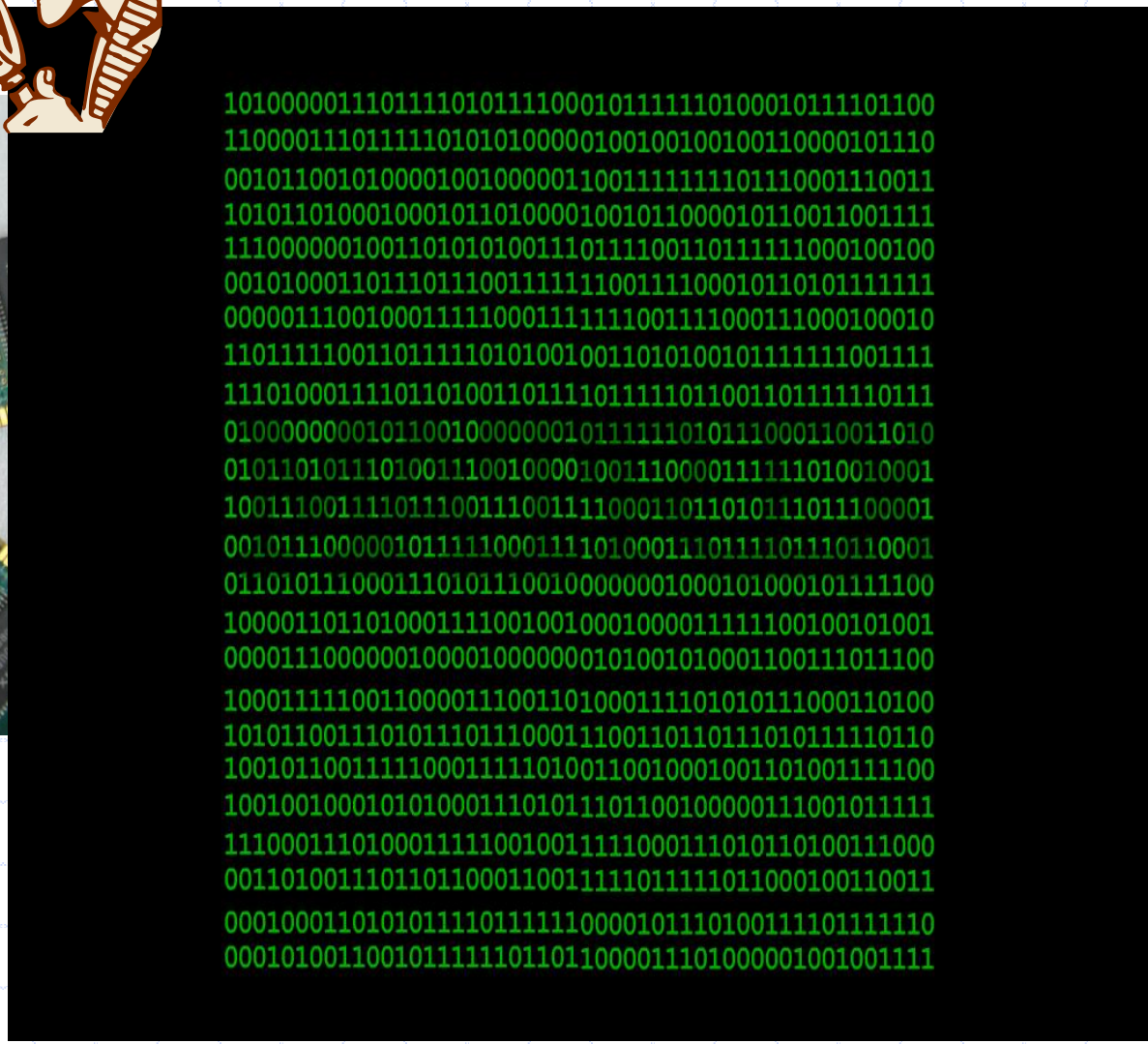
n.

1. One that directs, indicates, or points.
2. A scale indicator on a watch, balance, or other measuring instrument.
3. A long tapered stick for indicating objects, as on a chart or blackboard.
4. Any of a breed of hunting dogs that points game, typically having a smooth, short-haired coat that is usually white with black or brownish spots.
5.
  a. A piece of advice; a suggestion.
  b. A piece of indicative information: *interest rates and other pointers in the economic forecast.*
6. *Computer Science* A variable that holds the address of a core storage location.
7. *Computer Science* A symbol appearing on a display screen in a GUI that lets the user select a command by clicking with a pointing device or pressing the enter key when the pointer symbol is positioned on the appropriate button or icon.
8. Either of the two stars in the Big Dipper that are aligned so as to point to Polaris.

Pointer we are all born with

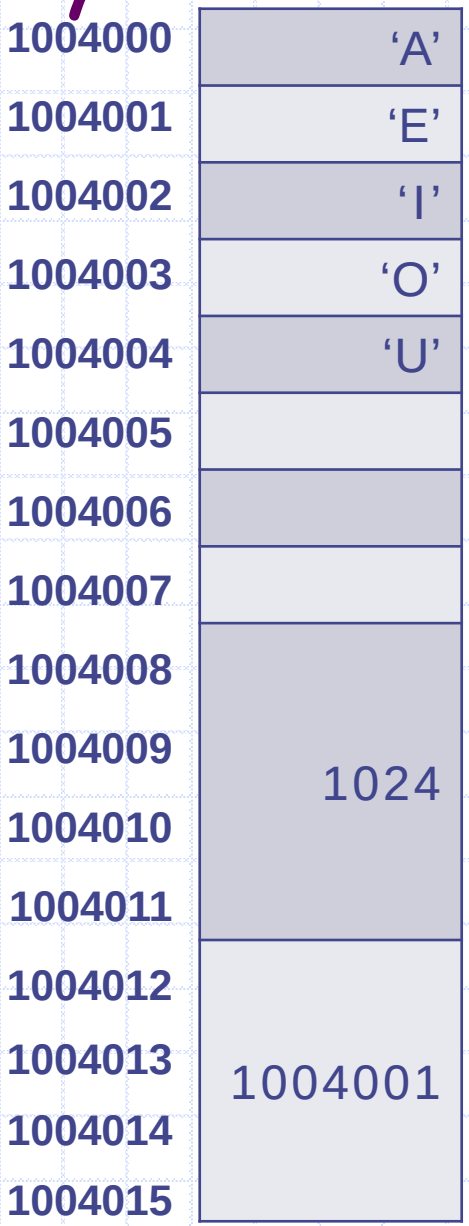






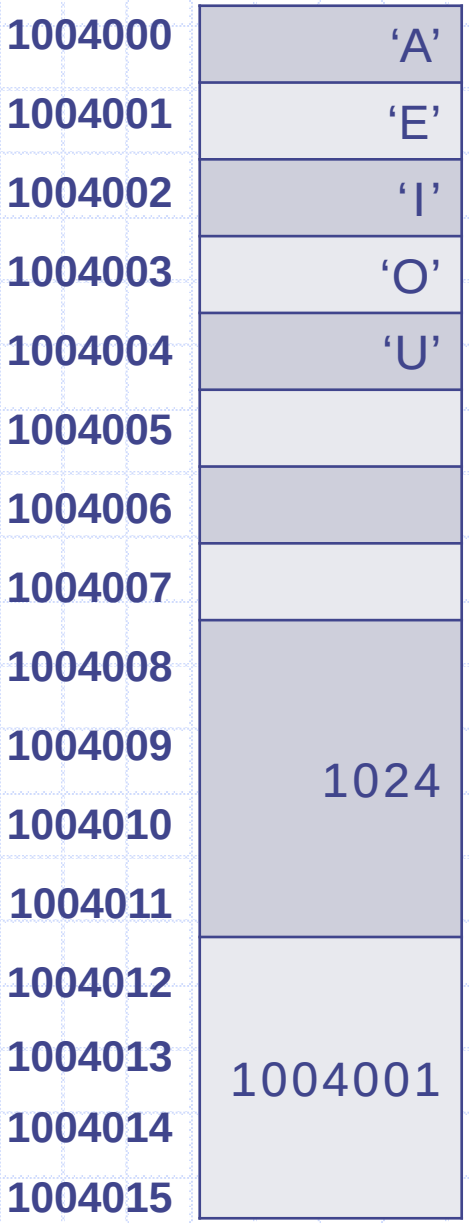
Simplified View of Memory

- "Array" of blocks
- Each block can hold a byte (8-bits)
- "char" stored in 1 block
- "int" (32-bit) stored in 4 consecutive blocks
- Finite number of blocks
  - Limited by the capacity of (Virtual) Memory
  - Blocks are addressable - $[0 \ldots 2^N-1]$



Simplified View of Memory

- Blocks are addressable.
- Address range: $[0 \ldots 2^N-1]$
- N is the number of bits in address (number of digits in binary world)
- Any integer in the above range
  - Can be used as an index in the MEMORY ARRAY
- Since memory array is unique, we can use this index alone
  - If context is clear



Simplified View of Memory

- Content of the 4-blocks starting at address 1004012
  - ✓1004001
- Without knowing the context it is not possible to deter… significance of numbe…
  - ✓It could be an inte… 1004001
  - ✓It could be the "location" of the block that stores 'E'

"Type" helps us disambiguate.

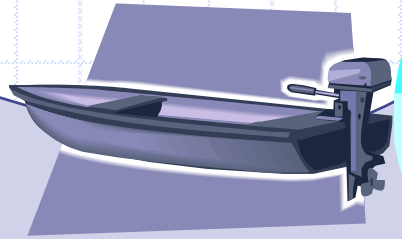
How do we decide what it is?

What is a Pointer

- **Pointer**: *A* **special type** of variable that contains an address of a memory location.
- Think of a pointer as a new data type (a new kind of box) that holds memory addresses.
- Pointers are almost always associated with the type of data that is contained in the memory location.
  - For example, an integer pointer is a memory location that contains an integer.
  - Character pointer, float pointer
  - Even pointer to pointer (more on this later …)

too much theory. Give examples please

Banti



OK.
Let me take you on a journey of Pointers in ~~SEA~~ C

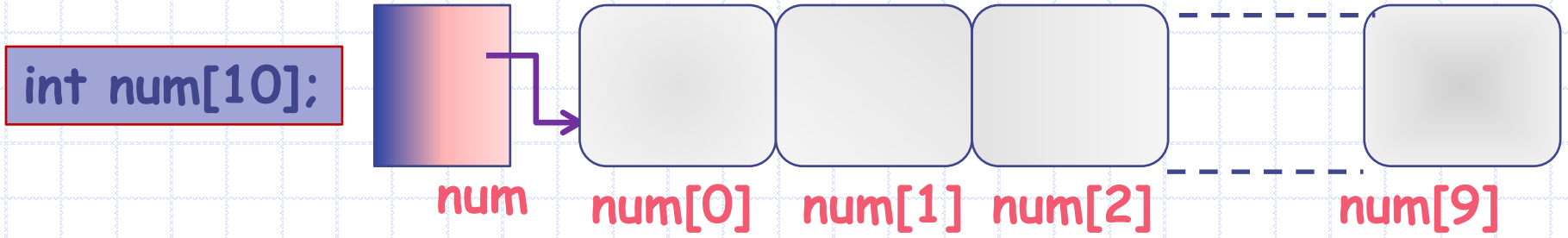
Owlie

Remember Arrays?

The memory allocated to array has two components:

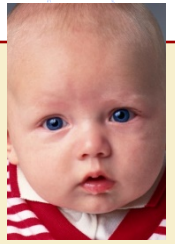
A consecutively allocated segment of memory boxes of the same type, and

A box with the same name as the array. This box holds the address of the base (i.e., first) element of the array.



This definition for num[10] gives **11 boxes**, 10 of type int, and 1 of type address of an int box.

Hmm...



1. We represent the **address of a box x** by an **arrow to the box x**. So **addresses** are referred to as **pointers**.
2. The contents of an address box is a pointer to the box whose address it contains. e.g., num points to num[0] above.

What can we do with a box? e.g., an integer box?

```
int num[10];
```

That's simple. We can do operations that are supported for the data type of the box.

True!. But we can also take the address of a box.We do this when we use scanf for reading using the & operator.

For integers, we can do + - * / % etc. for each of num[0] through num[9].

OK. Say i want to take the address of num[1] and store it in an address variable ptr.

```
ptr = &num[1];
```

ptr would be of type **address of int**. In C this type is **int ***.

But what is the type of **ptr**? And how do i define **ptr**?

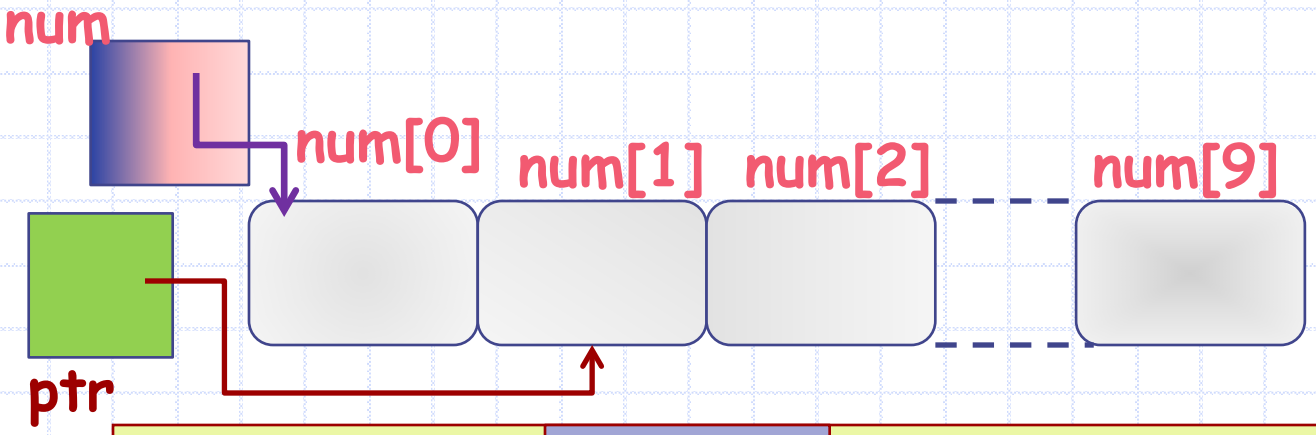
```
int * ptr;
ptr= &num[1];
```

To see the meaning of ptr =&num[1], let's look at the memory state.

```
int num[10];
int * ptr;
ptr = &num[1];
```

Here is the state after int num[10] gets defined.

OK, ptr is of type pointer to integer. But what does `ptr = &num[1];` mean?

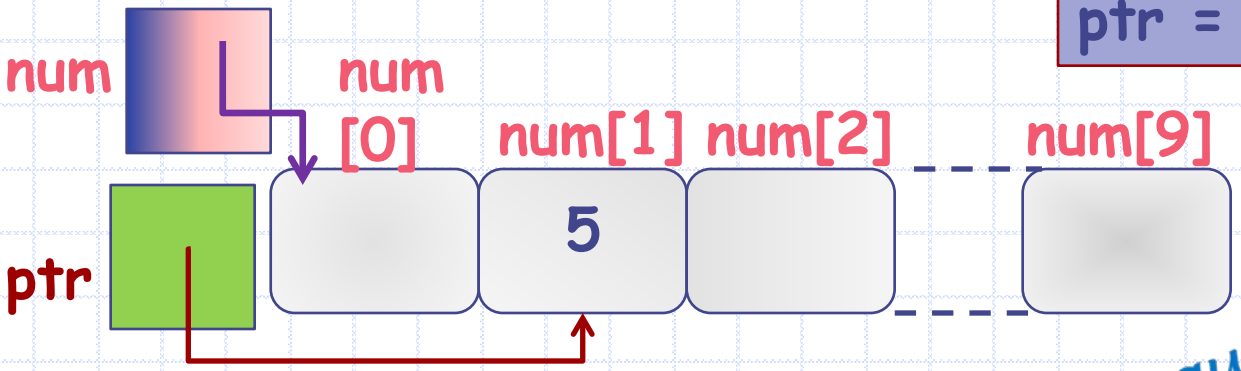


The statement `int *ptr;` creates a new box of type "**address of an int box**", more commonly referred to as, of type "**pointer to integer**".

The statement `ptr = &num[1];` assigns to ptr the address of the box num[1]. Commonly referred to as: **ptr now points to num[1].**

OK, i see. The program fragment below results in this memory state.

```
int num[10];
int * ptr;
ptr = &num[1];
```



question

1. Yes! scanf("%d",ptr) reads input integer into the box pointed to by the corresponding argument.
2. The box pointed to by ptr is num[1].
3. So num[1] becomes 5.

Suppose I now add the following statement after above fragment

```
scanf("%d",ptr);
```

Input

and input is :

5

Does num[1] become 5?