

"To Iterate is Human, to Recurse, Divine"

- James O. Coplien

# CSE102

## Computer Programming with C

2020-2021 Spring Semester

Recursion

© 2015-2021 Yakup Genç

These slides are largely adapted from J.R. Hanly, E.B. Koffman, F.E. Sevilgen, and others...

1

## Functions in C

```
#include <stdio.h>

int f2(int x) {
    return x*2;
}

int f3(int x) {
    return f2(x)*3;
}

int f4(int x) {
    return f3(x)*4;
}

int f5(int x) {
    return f4(x)*5;
}

int f6(int x) {
    return f5(x)*6;
}

void main() {
    int a = f6(10);
}
```

April 2021

CSE102 Computer Programming

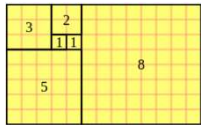
2

2

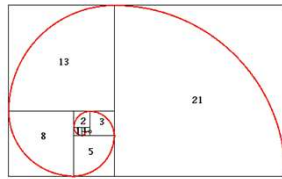
## Fibonacci Numbers

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

1+1=2  
1+2=3  
2+3=5  
3+5=8  
5+8=13  
8+13=21  
13+21=34  
21+34=55  
...



$$f(n) = \begin{cases} n=0 & 0 \\ n=1 & 1 \\ n>1 & f(n-1) + f(n-2) \end{cases}$$



April 2021

CSE102 Computer Programming

3

## Recursive Functions

$$\text{fact}(n) = \begin{cases} 1 & \text{if } n = 1 \\ n \cdot \text{fact}(n-1) & \text{if } n > 1 \end{cases}$$

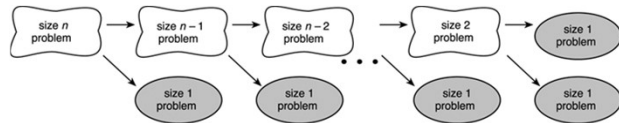
April 2021

CSE102 Computer Programming

4

4

## Splitting a Problem into Smaller Problems



April 2021

CSE102 Computer Programming

5

## Recursive Function multiply

```

1. /*
2.  * Performs integer multiplication using + operator.
3.  * Pre:  m and n are defined and n > 0
4.  * Post: returns m * n
5.  */
6. int
7. multiply(int m, int n)
8. {
9.     int ans;
10.
11.     if (n == 1)
12.         ans = m; /* simple case */
13.     else
14.         ans = m + multiply(m, n - 1); /* recursive step */
15.
16.     return (ans);
17. }

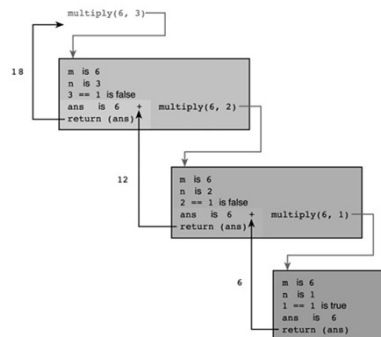
```

April 2021

CSE102 Computer Programming

6

## Trace of Function multiply



April 2021

CSE102 Computer Programming

7

## Output from multiply(8, 3)

```

7. int
8. multiply(int m, int n)
9. {
10.     int ans;
11.
12.     printf("Entering multiply with m = %d, n = %d\n", m, n);
13.
14.     if (n == 1)
15.         ans = m; /* simple case */
16.     else
17.         ans = m + multiply(m, n - 1); /* recursive step */
18.     printf("multiply(%d, %d) returning %d\n", m, n, ans);
19.
20.     return (ans);
21. }
22.
23. Entering multiply with m = 8, n = 3
24. Entering multiply with m = 8, n = 2
25. Entering multiply with m = 8, n = 1
26. multiply(8, 1) returning 8
27. multiply(8, 2) returning 16
28. multiply(8, 3) returning 24

```

April 2021

CSE102 Computer Programming

8

## Recursive Algorithm Development

Counting occurrences of 's' in

Mississippi sassafras

If I could just get someone to count the s's in this list

...then the number of s's is either that number or 1 more, depending on whether the first letter is an s.

April 2021

CSE102 Computer Programming

9

## Count a Character in a String

```

1. /*
2.  * Count the number of occurrences of character ch in string str
3.  */
4. int
5. count(char ch, const char *str)
6. {
7.
8.     int ans;
9.
10.    if (str[0] == '\0')                /* simple case */
11.        ans = 0;
12.    else                                /* redefine problem using recursion */
13.        if (ch == str[0]) /* first character must be counted */
14.            ans = 1 + count(ch, &str[1]);
15.        else /* first character is not counted */
16.            ans = count(ch, &str[1]);
17.
18.    return (ans);
19. }

```

April 2021

CSE102 Computer Programming

10

10

## Function reverse\_input\_words

```

1. /*
2.  * Take n words as input and print them in reverse order on separate lines.
3.  * Pre: n > 0
4.  */
5. void
6. reverse_input_words(int n)
7. {
8.     char word[WORDSIZ]; /* local variable for storing one word */
9.
10.    if (n <= 1) { /* simple case: just one word to get and print */
11.
12.        scanf("%s", word);
13.        printf("%s\n", word);
14.    } else { /* get this word; get and print the rest of the words in
15.              reverse order; then print this word */
16.
17.        scanf("%s", word);
18.        reverse_input_words(n - 1);
19.        printf("%s\n", word);
20.    }
21. }
22.

```

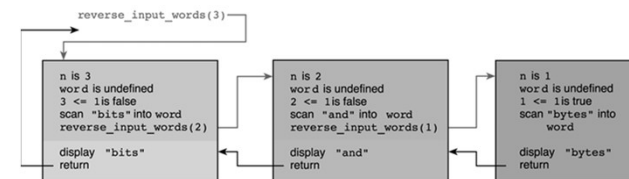
April 2021

CSE102 Computer Programming

11

11

## reverse\_input\_words(3): "bits" "and" "bytes"



April 2021

CSE102 Computer Programming

12

12

## Sequence of Events for Trace

Call `reverse_input_words` with `n` equal to 3.  
 Scan the first word ("bits") into word.  
 Call `reverse_input_words` with `n` equal to 2.  
   Scan the second word ("and") into word.  
   Call `reverse_input_words` with `n` equal to 1.  
     Scan the third word ("bytes") into word.  
     Display the third word ("bytes").  
     Return from third call.  
   Display the second word ("and").  
   Return from second call.  
 Display the first word ("bits").  
 Return from original call.

April 2021

CSE102 Computer Programming

13

## Recursive factorial Function

```

1. /*
2.  * Compute n! using a recursive definition
3.  * Pre: n >= 0
4.  */
5. int
6. factorial(int n)
7. {
8.     int ans;
9.
10.    if (n == 0)
11.        ans = 1;
12.    else
13.        ans = n * factorial(n - 1);
14.
15.    return (ans);
16. }
```

April 2021

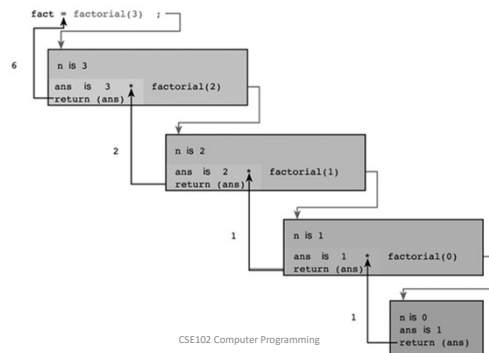
CSE102 Computer Programming

14

13

14

## Trace of `fact = factorial(3);`



April 2021

CSE102 Computer Programming

15

## Iterative Function factorial

```

1. /*
2.  * Computes n!
3.  * Pre: n is greater than or equal to zero
4.  */
5. int
6. factorial(int n)
7. {
8.     int i,          /* local variables */
9.     product = 1;
10.
11.    /* Compute the product n x (n-1) x (n-2) x ... x 2 x 1 */
12.    for (i = n; i > 1; --i) {
13.        product = product * i;
14.    }
15.
16.    /* Return function result */
17.    return (product);
18. }
```

April 2021

CSE102 Computer Programming

16

15

16

## Recursive Function fibonacci

```

1. /*
2.  * Computes the nth Fibonacci number
3.  * Pre: n > 0
4.  */
5. int
6. fibonacci(int n)
7. {
8.     int ans;
9.
10.    if (n == 1 || n == 2)
11.        ans = 1;
12.    else
13.        ans = fibonacci(n - 2) + fibonacci(n - 1);
14.
15.    return (ans);
16. }

```

April 2021

CSE102 Computer Programming

17

## Recursive Function gcd

```

7. /*
8.  * Finds the greatest common divisor of m and n
9.  * Pre: m and n are both > 0
10. */
11. int
12. gcd(int m, int n)
13. {
14.     int ans;
15.
16.     if (m % n == 0)
17.         ans = n;
18.     else
19.         ans = gcd(n, m % n);
20.
21.     return (ans);
22. }

```

April 2021

CSE102 Computer Programming

18

## Extract Capital Letters from a String

```

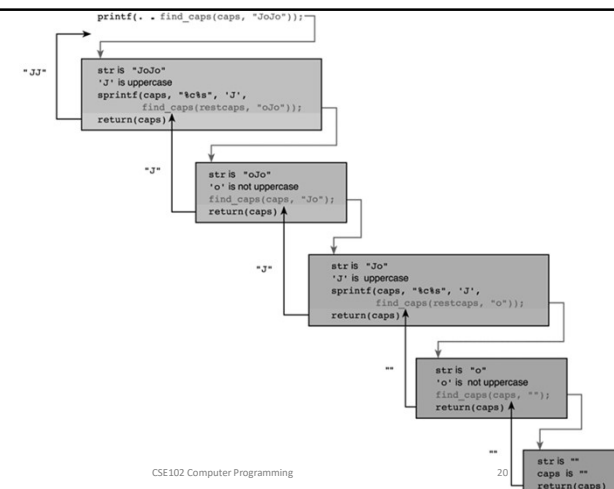
1. /*
2.  * Forms a string containing all the capital letters found in the input
3.  * parameter str.
4.  * Pre: caps has sufficient space to store all caps in str plus the null
5.  */
6. char *
7. find_caps(char *caps, /* output - string of all caps found in str */
8.            const char *str) /* input - string from which to extract caps */
9. {
10.     char restcaps[STRSI2]; /* caps from reststr */
11.
12.     if (str[0] == '\0')
13.         caps[0] = '\0'; /* no letters in str => no caps in str */
14.     else
15.         if (isupper(str[0]))
16.             sprintf(caps, "%c%s", str[0], find_caps(restcaps, &str[1]));
17.         else
18.             find_caps(caps, &str[1]);
19.
20.     return (caps);
21. }

```

April 2021

CSE102 Computer Programming

19



April 2021

CSE102 Computer Programming

20

20

## Sequence of Events

```

Call find_caps with input argument "JoJo" to determine value to print.
  Since 'J' is a capital letter,
  prepare to use sprintf to build a string with 'J'
  and the result of calling find_caps with input argument "oJo".
  Since 'o' is not a capital letter,
  call find_caps with input argument "Jo".
  Since 'J' is a capital letter,
  prepare to use sprintf to build a string with 'J'
  and the result of calling find_caps with input argument "o".
  Since 'o' is not a capital letter,
  call find_caps with input argument "".
  Return "" from fifth call.
  Return "" from fourth call.
  Complete execution of sprintf combining 'J' and "".
  Return "J" from third call.
  Complete execution of sprintf combining 'J' and "J".
  Return "JJ" from second call.
Complete call to printf to print Capital letters in JoJo are JJ.

```

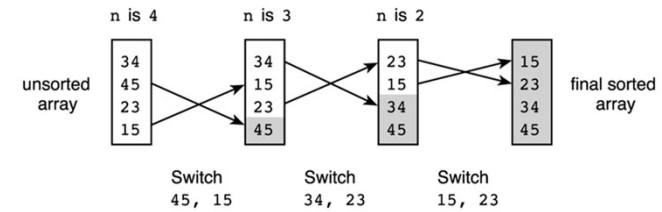
April 2021

CSE102 Computer Programming

21

## Trace of Selection Sort

n = size of unsorted subarray



April 2021

CSE102 Computer Programming

22

## Recursive Selection Sort

```

31. /*
32.  * Sorts n elements of an array of integers
33.  * Pre: n > 0 and first n elements of array are defined
34.  * Post: array elements are in ascending order
35.  */
36. void
37. select_sort(int array[], /* input/output - array to sort */
38.             int n)      /* input - number of array elements to sort */
39. {
40.
41.     if (n > 1) {
42.         place_largest(array, n);
43.         select_sort(array, n - 1);
44.     }
45. }

```

April 2021

CSE102 Computer Programming

23

```

1. /*
2.  * Finds the largest value in list array[0]..array[n-1] and exchanges it
3.  * with the value at array[n-1]
4.  * Pre: n > 0 and first n elements of array are defined
5.  * Post: array[n-1] contains largest value
6.  */
7. void
8. place_largest(int array[], /* input/output - array in which to place largest */
9.              int n)      /* input - number of array elements to
10.                          consider */
11. {
12.     int temp, /* temporary variable for exchange */
13.         j, /* array subscript and loop control */
14.         max_index; /* index of largest so far */
15.
16.     /* Save subscript of largest array value in max_index */
17.     max_index = n - 1; /* assume last value is largest */
18.     for (j = n - 2; j >= 0; --j)
19.         if (array[j] > array[max_index])
20.             max_index = j;
21.
22.     /* Unless largest value is already in last element, exchange
23.     largest and last elements */
24.     if (max_index != n - 1) {
25.         temp = array[n - 1];
26.         array[n - 1] = array[max_index];
27.         array[max_index] = temp;
28.     }
29. }
30.

```

April 2021

CSE102 Computer Programming

24

## Case Study: Recursive Set Operations

### Sets represented as character strings

```

15. #define SETSIZE 65 /* 52 uppercase and lowercase letters, 10 digits,
16.    {, }, and '\0' */
17. #define TRUE 1
18. #define FALSE 0
19.
20. int is_empty(const char *set);
21. int is_element(char ele, const char *set);
22. int is_set(const char *set);
23. int is_subset(const char *sub, const char *set);
24. char *set_union(char *result, const char *set1, const char *set2);
25. void print_with_commas(const char *str);
26. void print_set(const char *set);
27. char *get_set(char *set);

```

(continued)

April 2021

CSE102 Computer Programming

25

25

```

31. int
32. main(void)
33. {
34.     char ele, set_one[SETSIZE], set_two[SETSIZE], set_three[SETSIZE];
35.
36.     printf("A set is entered as a string of up to 52 letters\n",
37.           SETSIZE - 1);
38.     printf("and digits enclosed in {} ");
39.     printf("(no duplicate characters)\n");
40.     printf("For example, {a, b, c} is entered as {abc}\n");
41.
42.     printf("Enter a set to test validation function> ");
43.     get_set(set_one);
44.     putchar('\n');
45.     print_set(set_one);
46.     if (!is_set(set_one))
47.         printf(" is a valid set\n");
48.     else
49.         printf(" is invalid\n");
50.
51.     printf("Enter a single character, a space, and a set> ");
52.     while (isspace(ele = getchar())) /* gets first character after
53.                                     whitespace */
54.         continue;
55.     get_set(set_one);
56.     printf("\n{", ele);
57.     if (is_element(ele, set_one))
58.         printf("is an element of ");
59.     else
60.         printf("is not an element of ");
61.     print_set(set_one);
62.
63.     printf("\nEnter two sets to test set_union> ");
64.     get_set(set_one);
65.     get_set(set_two);
66.     printf("\nThe union of ");
67.     print_set(set_one);
68.     printf(" and ");
69.     print_set(set_two);
70.     printf(" is ");
71.     print_set(set_union(set_one, set_two));
72.     putchar('\n');
73.
74.     return (0);

```

April 2021

CSE102 Computer Programming

26

26

```

75. /*
76.  * Determines if set is empty. If so, returns 1; if not, returns 0.
77.  */
78. int
79. is_empty(const char *set)
80. {
81.     return (set[0] == '\0');
82. }
83.
84. /*
85.  * Determines if ele is an element of set.
86.  */
87. int
88. is_element(char ele, /* input - element to look for in set */
89.            const char *set) /* input - set in which to look for ele */
90. {
91.     int ans;
92.
93.     if (is_empty(set))
94.         ans = FALSE;
95.     else if (set[0] == ele)
96.         ans = TRUE;
97.     else
98.         ans = is_element(ele, &set[1]);
99.
100.    return (ans);
101. }
102.
103.

```

April 2021

CSE102 Computer Programming

27

27

```

104. /*
105.  * Determines if string value of set represents a valid set (no duplicate
106.  * elements)
107.  */
108. int
109. is_set(const char *set)
110. {
111.     int ans;
112.
113.     if (is_empty(set))
114.         ans = TRUE;
115.     else if (is_element(set[0], &set[1]))
116.         ans = FALSE;
117.     else
118.         ans = is_set(&set[1]);
119.     return (ans);
120. }
121.
122. /*
123.  * Determines if value of sub is a subset of value of set.
124.  */
125. int
126. is_subset(const char *sub, const char *set)
127. {
128.     int ans;
129.
130.     if (is_empty(sub))
131.         ans = TRUE;
132.     else if (!is_element(sub[0], set))
133.         ans = FALSE;
134.     else
135.         ans = is_subset(&sub[1], set);
136.
137.     return (ans);
138. }
139.

```

April 2021

CSE102 Computer Programming

28

28

```

140. /*
141.  * Finds the union of set1 and set2.
142.  * Pre: size of result array is at least SETSIZE;
143.  *      set1 and set2 are valid sets of characters and digits
144.  */
145. char *
146. set_union(char *result, /* output - space in which to store
147.
148.      const char *set1, /* input - sets whose
149.      const char *set2) /* union is being formed
150.
151. {
152.     char temp[SETSIZE]; /* local variable to hold result of call
153.                          to set_union embedded in sprintf call
154.
155.     if (is_empty(set1))
156.         strcpy(result, set2);
157.     else if (is_element(set1[0], set2))
158.         set_union(result, &set1[1], set2);
159.     else
160.         sprintf(result, "%c%c", set1[0],
161.                 set_union(temp, &set1[1], set2));
162.
163.     return (result);
164. }
165.
166. /*
167.  * Displays a string so that each pair of characters is separated by a
168.  * comma and a space.
169.  */
170. void
171. print_with_commas(const char *str)
172. {
173.     if (strlen(str) == 1) {
174.         putchar(str[0]);
175.     } else {
176.         printf("%c, ", str[0]);
177.         print_with_commas(&str[1]);
178.     }
179. }

```

April 2021

CSE102 Computer Programming

29

29

```

180. /*
181.  * Displays a string in standard set notation.
182.  * e.g. print_set("abc") outputs {a, b, c}
183.  */
184. void
185. print_set(const char *set)
186. {
187.     putchar('{');
188.     if (!is_empty(set))
189.         print_with_commas(set);
190.     putchar('}');
191. }
192.
193. /*
194.  * Gets a set input as a string with brackets (e.g., {abc})
195.  * and strips off the brackets.
196.  */
197. char *
198. get_set(char *set) /* output - set string without brackets {}
199.
200. {
201.     char inset[SETSIZE];
202.
203.     scanf("%s", inset);
204.     strncpy(set, &inset[1], strlen(inset) - 2);
205.     set[strlen(inset) - 2] = '\0';
206.     return (set);

```

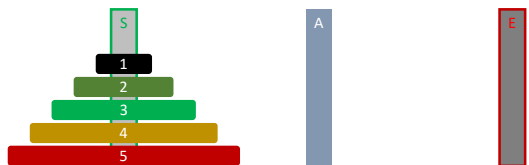
April 2021

CSE102 Computer Programming

30

30

## Towers of Hanoi



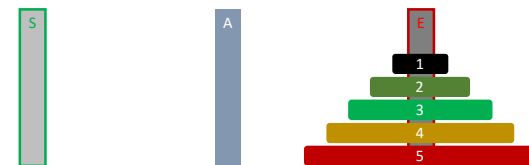
April 2021

CSE102 Computer Programming

31

31

## Towers of Hanoi



April 2021

CSE102 Computer Programming

32

32



```

1  /*
2  *   Displays instructions for moving n disks from from_peg to to_peg using
3  *   aux_peg as an auxiliary. Disks are numbered 1 to n (smallest to
4  *   largest). Instructions call for moving one disk at a time and never
5  *   require placing a larger disk on top of a smaller one.
6  */
7  void
8  tower(char from_peg,    /* input - characters naming          */
9        char to_peg,     /* the problem's          */
10        char aux_peg,    /* three pegs             */
11        int n)           /* input - number of disks to move */
12  {
13      if (n == 1) {
14          printf("Move disk 1 from peg %c to peg %c\n", from_peg, to_peg);
15      } else {
16          tower(from_peg, aux_peg, to_peg, n - 1);
17          printf("Move disk %d from peg %c to peg %c\n", n, from_peg, to_peg);
18          tower(aux_peg, to_peg, from_peg, n - 1);
19      }
20  }

```

April 2021

CSE102 Computer Programming

33

```

graph TD
    A["tower('A', 'C', 'B', 3)"] --> B["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', 2))"]
    B --> C["lower('A', 'A', 'C', 2);  
move 3 from A to C  
lower('A', 'A', 'C', 2);  
return"]
    C --> D["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', 1))"]
    D --> E["lower('B', 'A', 'C', 1);  
move 2 from A to B  
lower('B', 'A', 'C', 1);  
return"]
    E --> F["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', 1))"]
    F --> G["lower('A', 'A', 'C', 1);  
move 1 from A to C  
lower('A', 'A', 'C', 1);  
return"]
    G --> H["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', 0))"]
    H --> I["lower('B', 'A', 'C', 0);  
move 2 from A to B  
lower('B', 'A', 'C', 0);  
return"]
    I --> J["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', 0))"]
    J --> K["lower('A', 'A', 'C', 0);  
move 1 from A to C  
lower('A', 'A', 'C', 0);  
return"]
    K --> L["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -1))"]
    L --> M["lower('B', 'A', 'C', -1);  
move 2 from A to B  
lower('B', 'A', 'C', -1);  
return"]
    M --> N["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -1))"]
    N --> O["lower('A', 'A', 'C', -1);  
move 1 from A to C  
lower('A', 'A', 'C', -1);  
return"]
    O --> P["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -2))"]
    P --> Q["lower('B', 'A', 'C', -2);  
move 2 from A to B  
lower('B', 'A', 'C', -2);  
return"]
    Q --> R["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -2))"]
    R --> S["lower('A', 'A', 'C', -2);  
move 1 from A to C  
lower('A', 'A', 'C', -2);  
return"]
    S --> T["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -3))"]
    T --> U["lower('B', 'A', 'C', -3);  
move 2 from A to B  
lower('B', 'A', 'C', -3);  
return"]
    U --> V["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -3))"]
    V --> W["lower('A', 'A', 'C', -3);  
move 1 from A to C  
lower('A', 'A', 'C', -3);  
return"]
    W --> X["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -4))"]
    X --> Y["lower('B', 'A', 'C', -4);  
move 2 from A to B  
lower('B', 'A', 'C', -4);  
return"]
    Y --> Z["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -4))"]
    Z --> AA["lower('A', 'A', 'C', -4);  
move 1 from A to C  
lower('A', 'A', 'C', -4);  
return"]
    AA --> AB["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -5))"]
    AB --> AC["lower('B', 'A', 'C', -5);  
move 2 from A to B  
lower('B', 'A', 'C', -5);  
return"]
    AC --> AD["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -5))"]
    AD --> AE["lower('A', 'A', 'C', -5);  
move 1 from A to C  
lower('A', 'A', 'C', -5);  
return"]
    AE --> AF["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -6))"]
    AF --> AG["lower('B', 'A', 'C', -6);  
move 2 from A to B  
lower('B', 'A', 'C', -6);  
return"]
    AG --> AH["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -6))"]
    AH --> AI["lower('A', 'A', 'C', -6);  
move 1 from A to C  
lower('A', 'A', 'C', -6);  
return"]
    AI --> AJ["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -7))"]
    AJ --> AK["lower('B', 'A', 'C', -7);  
move 2 from A to B  
lower('B', 'A', 'C', -7);  
return"]
    AK --> AL["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -7))"]
    AL --> AM["lower('A', 'A', 'C', -7);  
move 1 from A to C  
lower('A', 'A', 'C', -7);  
return"]
    AM --> AN["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -8))"]
    AN --> AO["lower('B', 'A', 'C', -8);  
move 2 from A to B  
lower('B', 'A', 'C', -8);  
return"]
    AO --> AP["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -8))"]
    AP --> AQ["lower('A', 'A', 'C', -8);  
move 1 from A to C  
lower('A', 'A', 'C', -8);  
return"]
    AQ --> AR["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -9))"]
    AR --> AS["lower('B', 'A', 'C', -9);  
move 2 from A to B  
lower('B', 'A', 'C', -9);  
return"]
    AS --> AT["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -9))"]
    AT --> AU["lower('A', 'A', 'C', -9);  
move 1 from A to C  
lower('A', 'A', 'C', -9);  
return"]
    AU --> AV["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -10))"]
    AV --> AW["lower('B', 'A', 'C', -10);  
move 2 from A to B  
lower('B', 'A', 'C', -10);  
return"]
    AW --> AX["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -10))"]
    AX --> AY["lower('A', 'A', 'C', -10);  
move 1 from A to C  
lower('A', 'A', 'C', -10);  
return"]
    AY --> AZ["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -11))"]
    AZ --> BA["lower('B', 'A', 'C', -11);  
move 2 from A to B  
lower('B', 'A', 'C', -11);  
return"]
    BA --> BB["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -11))"]
    BB --> BC["lower('A', 'A', 'C', -11);  
move 1 from A to C  
lower('A', 'A', 'C', -11);  
return"]
    BC --> BD["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -12))"]
    BD --> BE["lower('B', 'A', 'C', -12);  
move 2 from A to B  
lower('B', 'A', 'C', -12);  
return"]
    BE --> BF["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -12))"]
    BF --> BG["lower('A', 'A', 'C', -12);  
move 1 from A to C  
lower('A', 'A', 'C', -12);  
return"]
    BG --> BH["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -13))"]
    BH --> BI["lower('B', 'A', 'C', -13);  
move 2 from A to B  
lower('B', 'A', 'C', -13);  
return"]
    BI --> BJ["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -13))"]
    BJ --> BK["lower('A', 'A', 'C', -13);  
move 1 from A to C  
lower('A', 'A', 'C', -13);  
return"]
    BK --> BL["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -14))"]
    BL --> BM["lower('B', 'A', 'C', -14);  
move 2 from A to B  
lower('B', 'A', 'C', -14);  
return"]
    BM --> BN["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -14))"]
    BN --> BO["lower('A', 'A', 'C', -14);  
move 1 from A to C  
lower('A', 'A', 'C', -14);  
return"]
    BO --> BP["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -15))"]
    BP --> BQ["lower('B', 'A', 'C', -15);  
move 2 from A to B  
lower('B', 'A', 'C', -15);  
return"]
    BQ --> BR["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -15))"]
    BR --> BS["lower('A', 'A', 'C', -15);  
move 1 from A to C  
lower('A', 'A', 'C', -15);  
return"]
    BS --> BT["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -16))"]
    BT --> BU["lower('B', 'A', 'C', -16);  
move 2 from A to B  
lower('B', 'A', 'C', -16);  
return"]
    BU --> BV["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -16))"]
    BV --> BW["lower('A', 'A', 'C', -16);  
move 1 from A to C  
lower('A', 'A', 'C', -16);  
return"]
    BW --> BX["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -17))"]
    BX --> BY["lower('B', 'A', 'C', -17);  
move 2 from A to B  
lower('B', 'A', 'C', -17);  
return"]
    BY --> BZ["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -17))"]
    BZ --> C1["lower('A', 'A', 'C', -17);  
move 1 from A to C  
lower('A', 'A', 'C', -17);  
return"]
    C1 --> C2["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -18))"]
    C2 --> C3["lower('B', 'A', 'C', -18);  
move 2 from A to B  
lower('B', 'A', 'C', -18);  
return"]
    C3 --> C4["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -18))"]
    C4 --> C5["lower('A', 'A', 'C', -18);  
move 1 from A to C  
lower('A', 'A', 'C', -18);  
return"]
    C5 --> C6["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -19))"]
    C6 --> C7["lower('B', 'A', 'C', -19);  
move 2 from A to B  
lower('B', 'A', 'C', -19);  
return"]
    C7 --> C8["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -19))"]
    C8 --> C9["lower('A', 'A', 'C', -19);  
move 1 from A to C  
lower('A', 'A', 'C', -19);  
return"]
    C9 --> C10["from_peg('B', 'C', 'aux_peg('A', 'A', 'C', -20))"]
    C10 --> C11["lower('B', 'A', 'C', -20);  
move 2 from A to B  
lower('B', 'A', 'C', -20);  
return"]
    C11 --> C12["from_peg('A', 'C', 'aux_peg('B', 'A', 'C', -20))"]
    C12 --> C13["lower
```

April 2021

CSE102 Computer Programming

34

Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C

April 2021

CSE102 Computer Programming

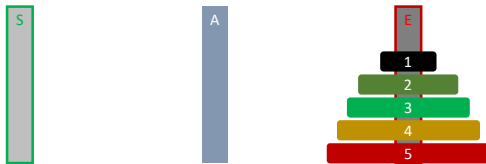
35

April 2021

CSE102 Computer Programming

36

## Towers of Hanoi

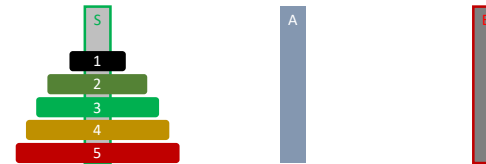


April 2021

CSE102 Computer Programming

37

## Towers of Hanoi



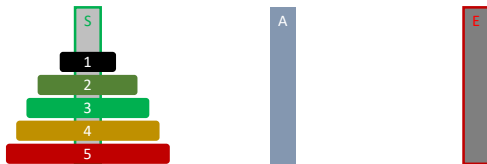
Tower(S, E, A, {1,2,3,4,5})

April 2021

CSE102 Computer Programming

38

## Towers of Hanoi



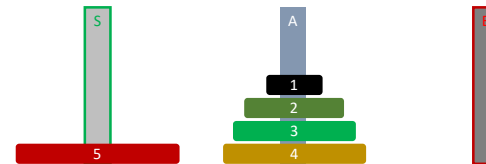
Tower(S,E,A,{1,2,3,4,5}) :  
Step1: Tower(S,A,E,{1,2,3,4}) , Step2: Move 5 from S to E, Step3: Tower(A,E,S,{1,2,3,4})

April 2021

CSE102 Computer Programming

39

## Towers of Hanoi



Tower(S,E,A,{1,2,3,4,5}) :  
Step1: Tower(S,A,E,{1,2,3,4}) , Step2: Move 5 from S to E, Step3: Tower(A,E,S,{1,2,3,4})

April 2021

CSE102 Computer Programming

40

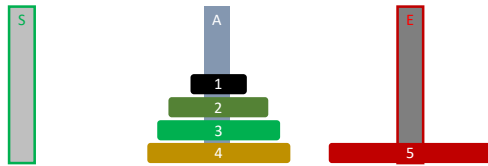
37

38

39

40

## Towers of Hanoi



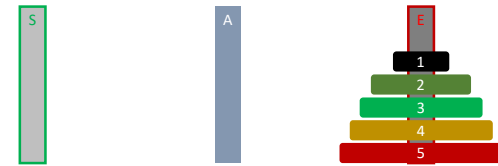
Tower(S,E,A,{1,2,3,4,5}) :  
 Step1: Tower(S,A,E,{1,2,3,4}) , Step2: Move 5 from S to E, Step3: Tower(A,E,S,{1,2,3,4})

April 2021

CSE102 Computer Programming

41

## Towers of Hanoi



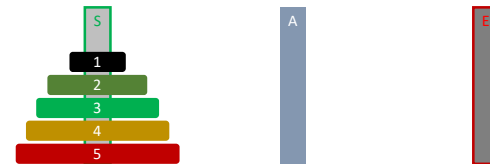
Tower(S,E,A,{1,2,3,4,5}) :  
 Step1: Tower(S,A,E,{1,2,3,4}) , Step2: Move 5 from S to E, Step3: Tower(A,E,S,{1,2,3,4})

April 2021

CSE102 Computer Programming

42

## Towers of Hanoi



Tower(S,A,E,{1,2,3,4}) :  
 Step1: Tower(S,E,A,{1,2,3}) , Step2: Move 4 from S to A, Step3: Tower(E,A,S,{1,2,3})

April 2021

CSE102 Computer Programming

44

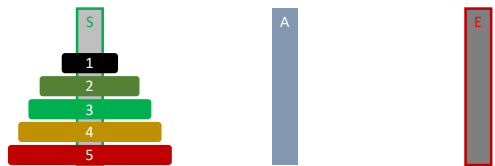
41

42

43

44

## Towers of Hanoi



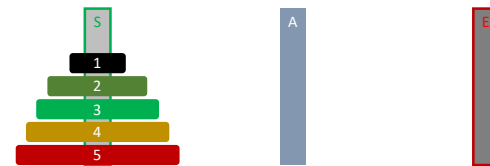
Tower(S,E,A,{1,2,3}) :  
 Step1: Tower(S,A,E,{1,2}) , Step2: Move 3 from S to E, Step3: Tower(E,A,S,{1,2})

April 2021

CSE102 Computer Programming

45

## Towers of Hanoi



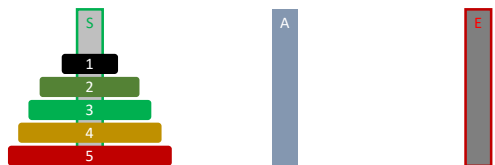
Tower(S,A,E,{1,2}) :  
 Step1: Tower(S,E,A,{1}) , Step2: Move 2 from S to A, Step3: Tower(E,A,S,{1})

April 2021

CSE102 Computer Programming

46

## Towers of Hanoi



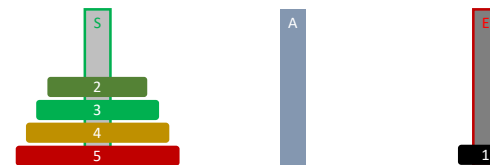
Tower(S,E,A,{1}) :  
 Move 1 from S to E

April 2021

CSE102 Computer Programming

47

## Towers of Hanoi



Tower(S,E,A,{1}) :  
 Move 1 from S to E

April 2021

CSE102 Computer Programming

48

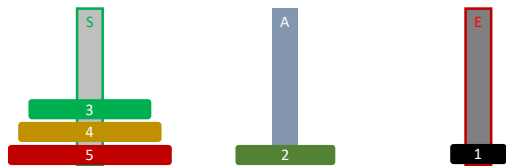
45

46

47

48

## Towers of Hanoi



Tower(S,A,E,{1,2}) :  
 Step1: Tower(S,E,A,{1}) , Step2: Move 2 from S to A, Step3: Tower(E,A,S,{1})

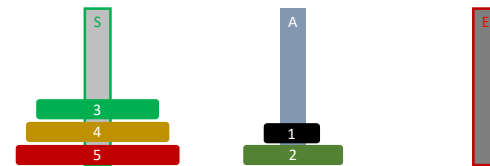
April 2021

CSE102 Computer Programming

49

49

## Towers of Hanoi



Tower(S,A,E,{1,2}) :  
 Step1: Tower(S,E,A,{1}) , Step2: Move 2 from S to A, Step3: Tower(E,A,S,{1})

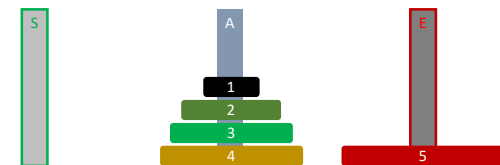
April 2021

CSE102 Computer Programming

50

50

## Towers of Hanoi



Tower(A,E,S,{1,2,3,4}) :  
 Step1: Tower(A,S,E,{1,2,3}) , Step2: Move 4 from A to E, Step3: Tower(S,E,A,{1,2,3})

April 2021

CSE102 Computer Programming

52

52

April 2021

CSE102 Computer Programming

51

51

Thanks for listening!