ECS 36A, April 30, 2024

Announcements

- Change in office hours today:

  1:00pm–1:30pm in 2203 Watershed Sciences

  2:30pm–3:00pm in 2209 Watershsed Sciences
- Midterm has been moved to Tuesday, May 7
  - It was scheduled for Thursday, May 2
- Midterm study guide, sample midterm are on Canvas
  - Sample midterm is shorter than the real one will be
  - Answers are now posted to Canvas
- Tutoring is available from the CS Tutoring Club
  - See the announcement on Canvas

Recursive String Length

- length of string with no characters = 0
- length of string otherwise = 1 + (length of string not including the first char)

```
int rstrlen(char *s)
{
    /* Base case: empty string */
    if (*s == '\0')    return(0);
    /* recursive case: 1 more than rest of string */
    return(1 + rstrlen(s+1));
}
```

Call rstrlen from main

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```

call rstrlen("hello") from line 16

Call rstrlen from Line 8

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```

call rstrlen("ello") from line 8
call rstrlen("hello") from line 16

Call rstrlen from Line 8

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```

call rstrlen("llo") from line 8
call rstrlen("ello") from line 8
call rstrlen("hello") from line 16

Call rstrlen from Line 8

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```

call rstrlen("lo") from line 8
call rstrlen("llo") from line 8
call rstrlen("ello") from line 8
call rstrlen("hello") from line 16

Call rstrlen from Line 8

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```

call rstrlen("o") from line 8
call rstrlen("lo") from line 8
call rstrlen("llo") from line 8
call rstrlen("ello") from line 8
call rstrlen("hello") from line 16

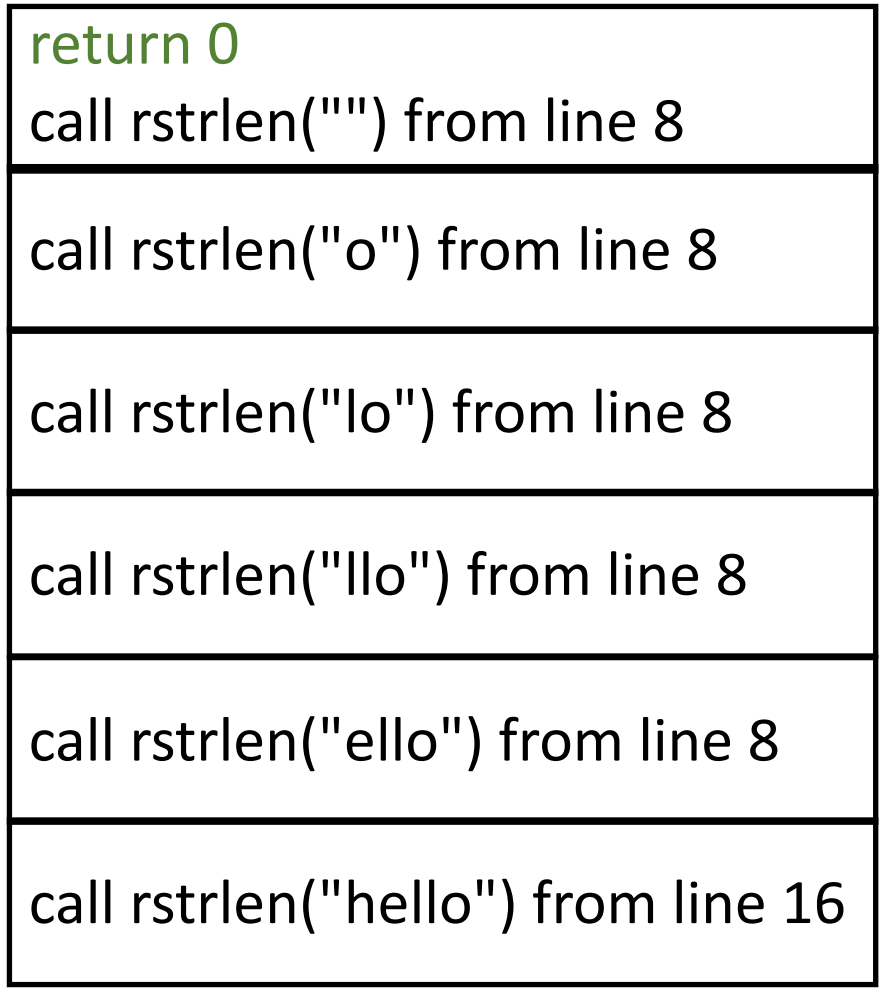
Call rstrlen from Line 8

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```

call rstrlen("") from line 8
call rstrlen("o") from line 8
call rstrlen("lo") from line 16
call rstrlen("llo") from line 8
call rstrlen("ello") from line 8
call rstrlen("hello") from line 16

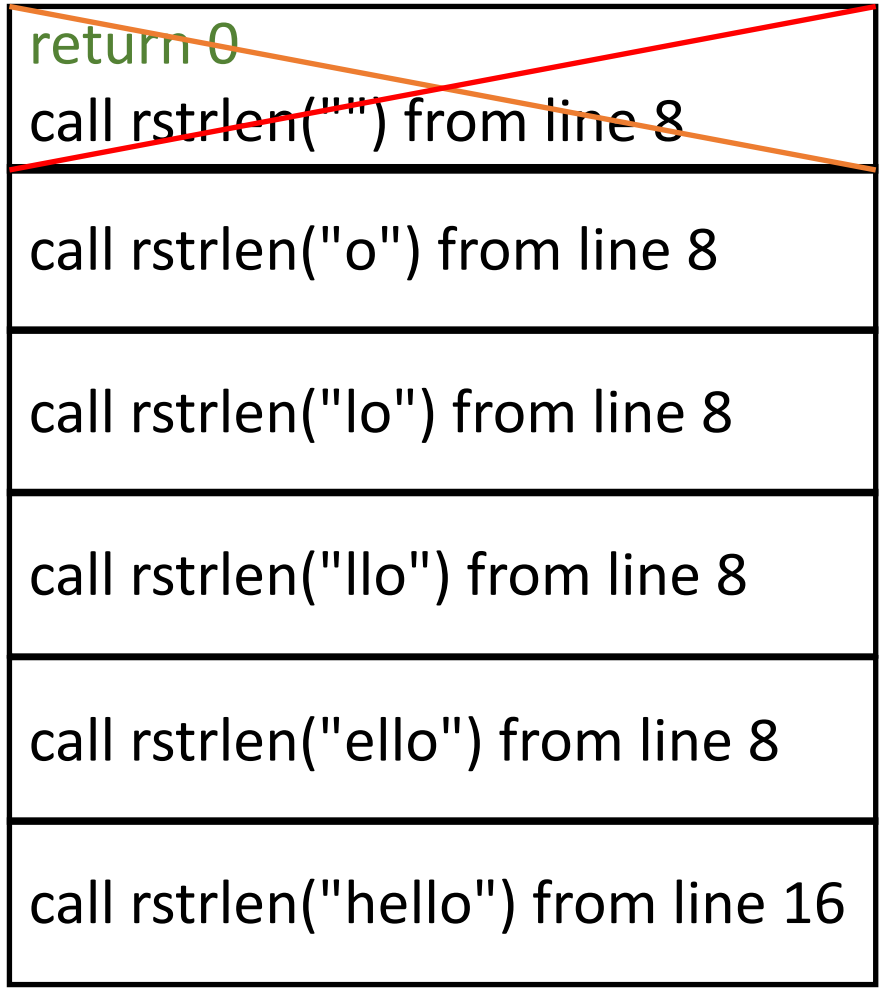
Return 0

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



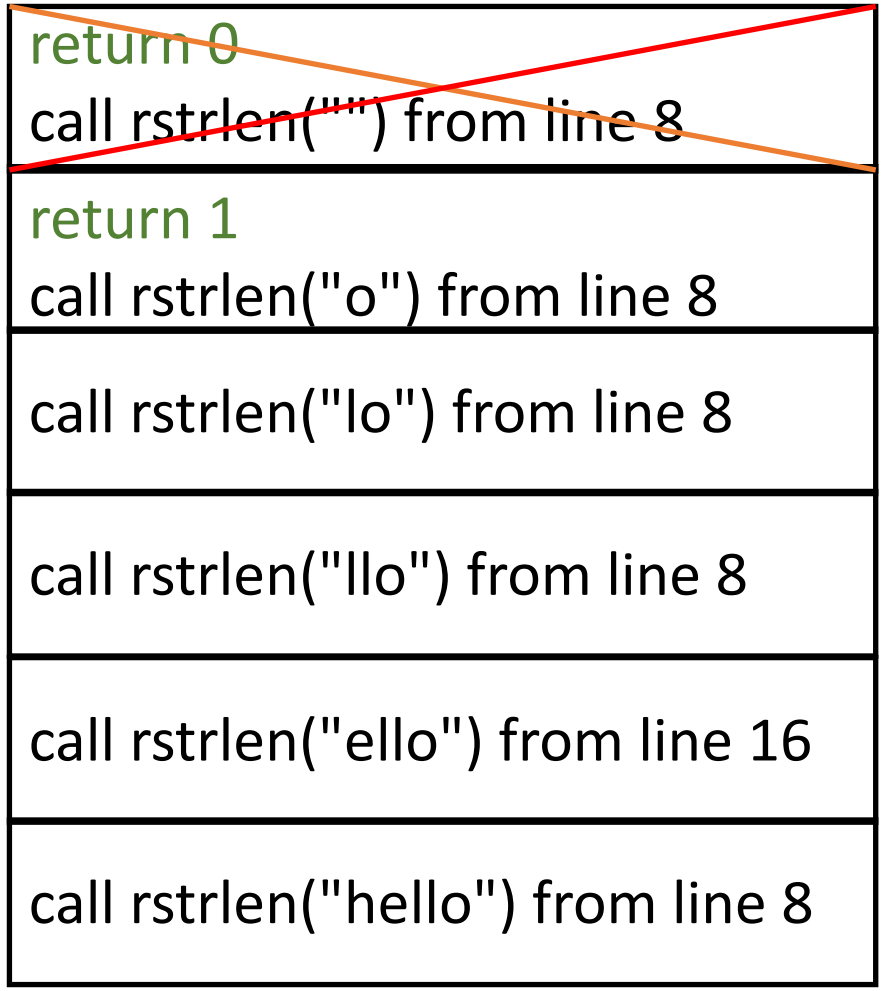
Return to rstrlen on Line 8

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



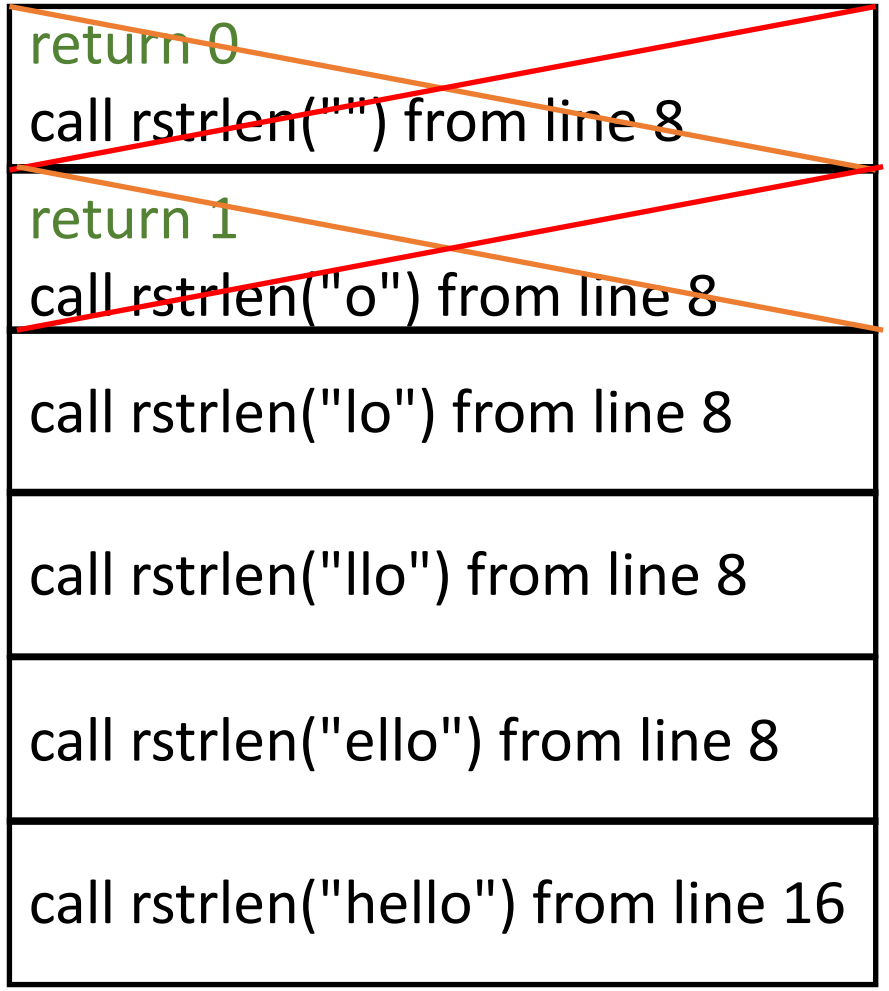
Return 1

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



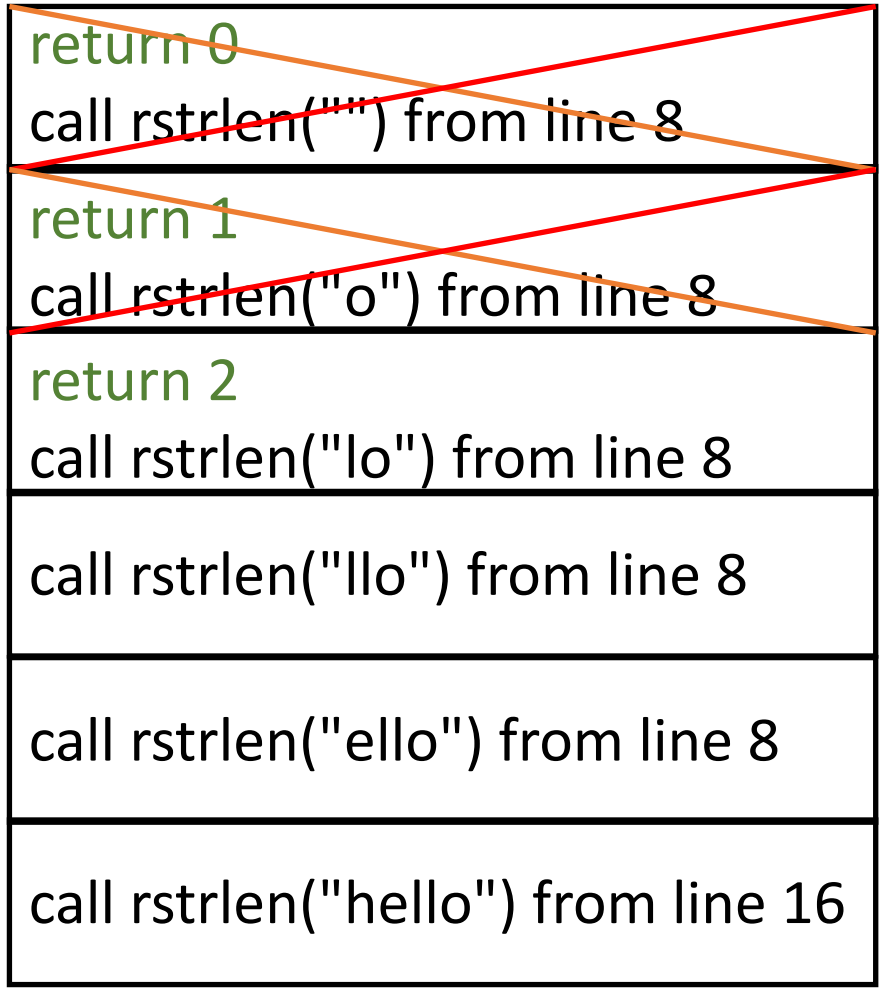
Return to rstrlen on Line 8

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



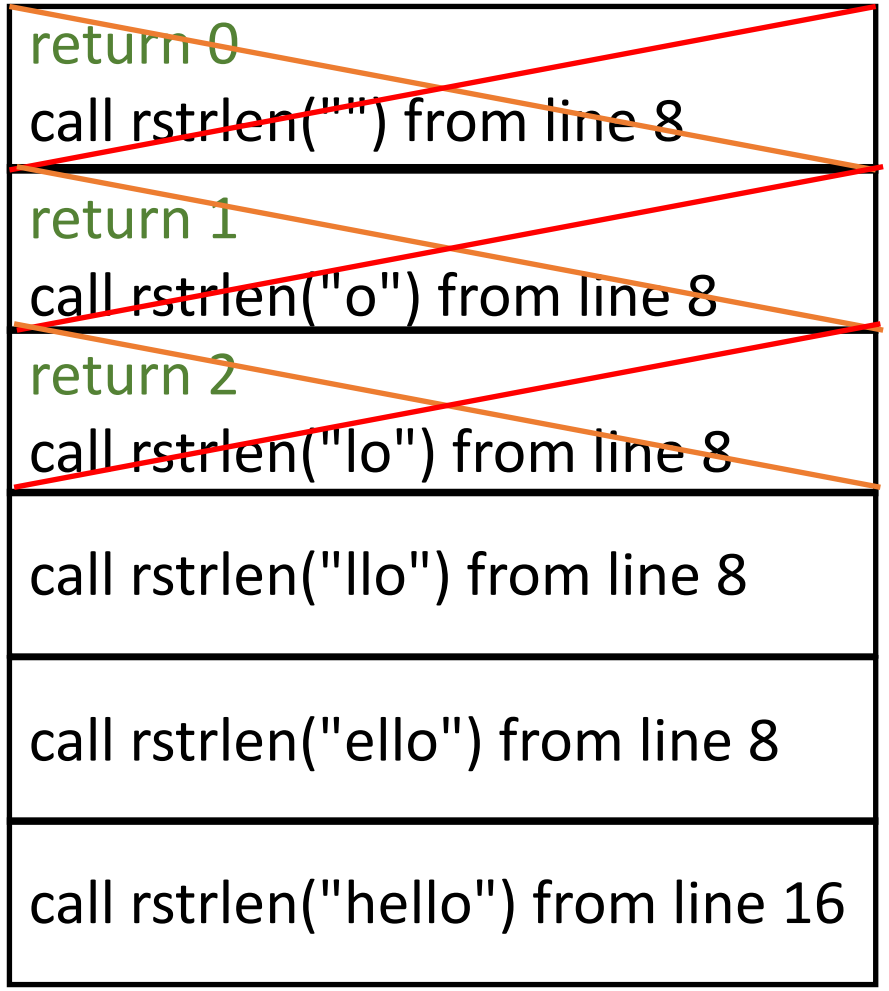
Return 2

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



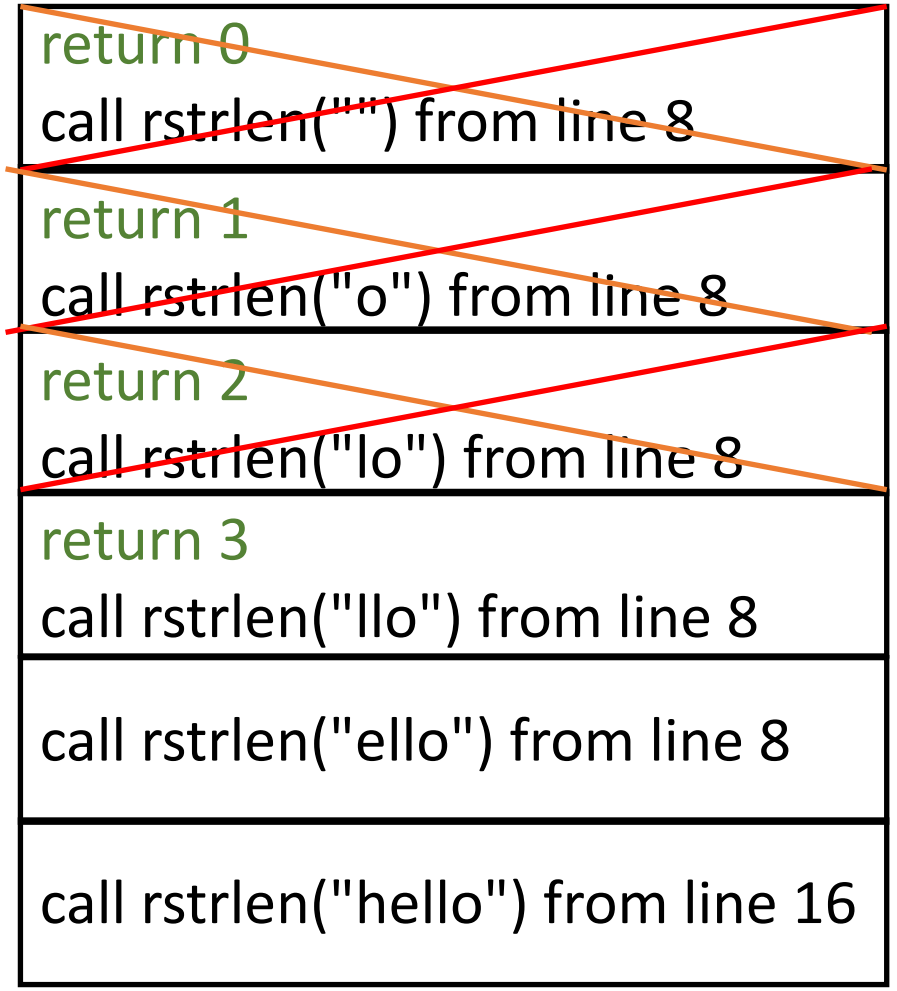
Return to rstrlen on Line 8

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



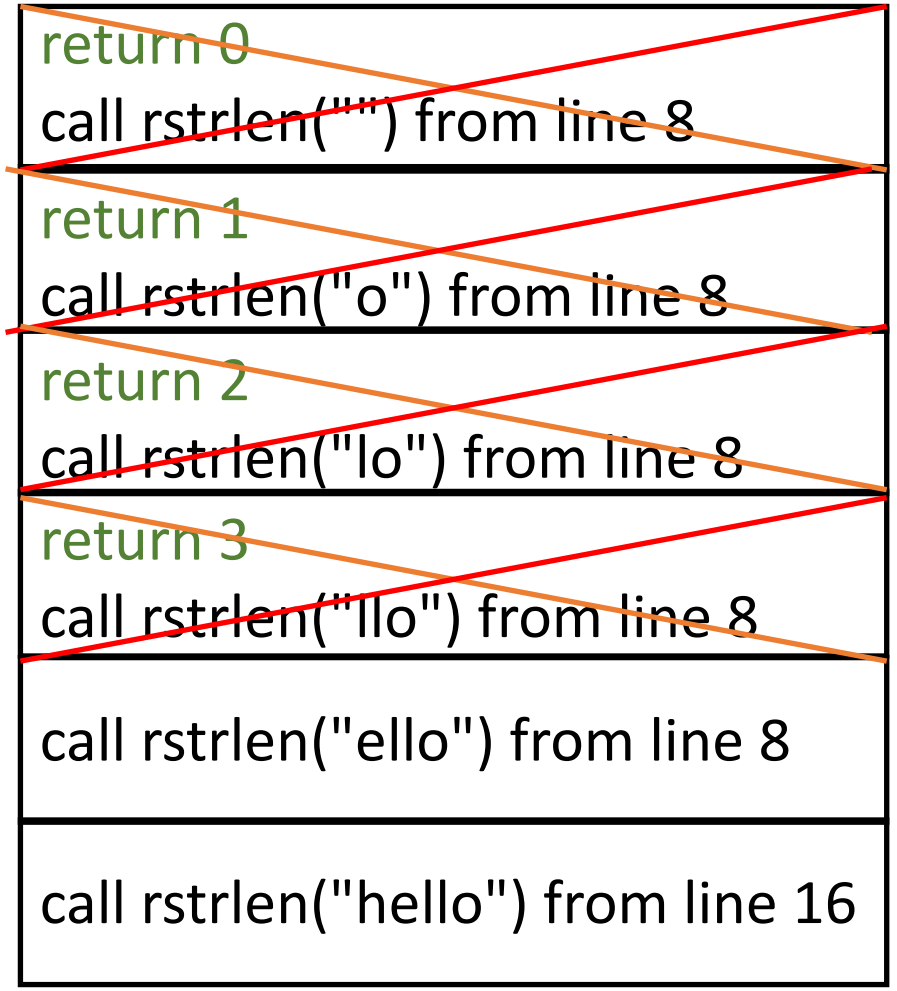
Return 3

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



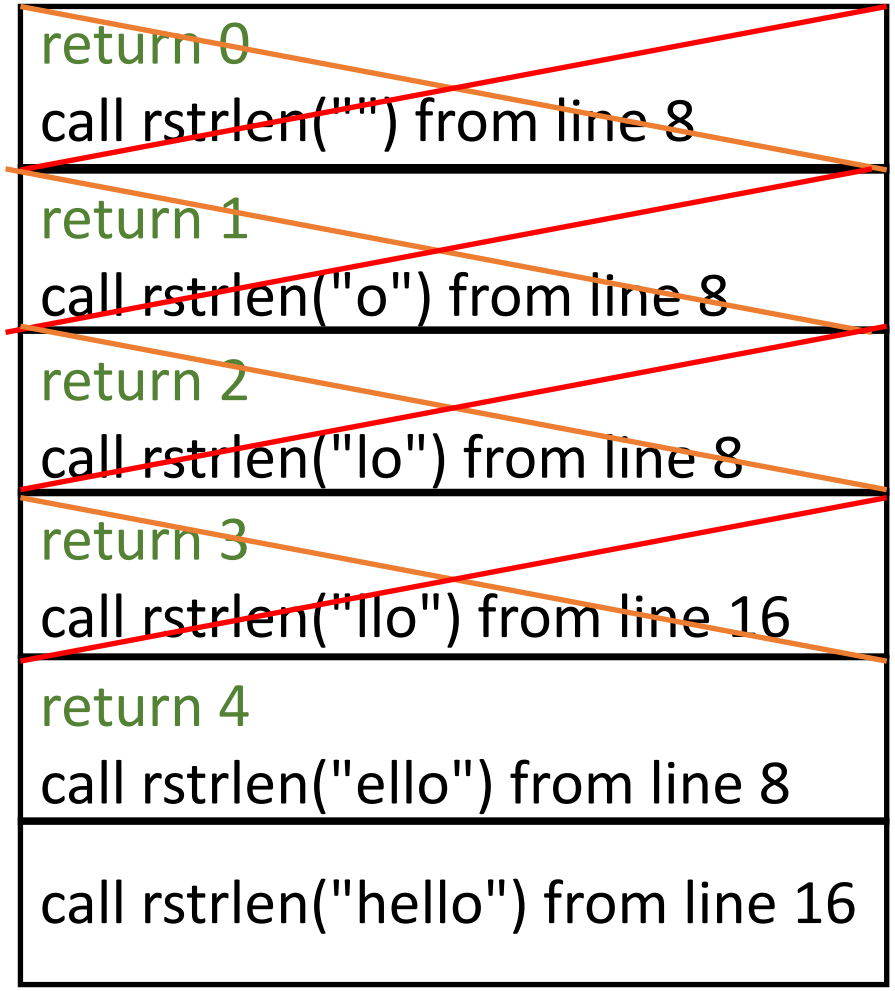
Return to rstrlen on Line 8

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



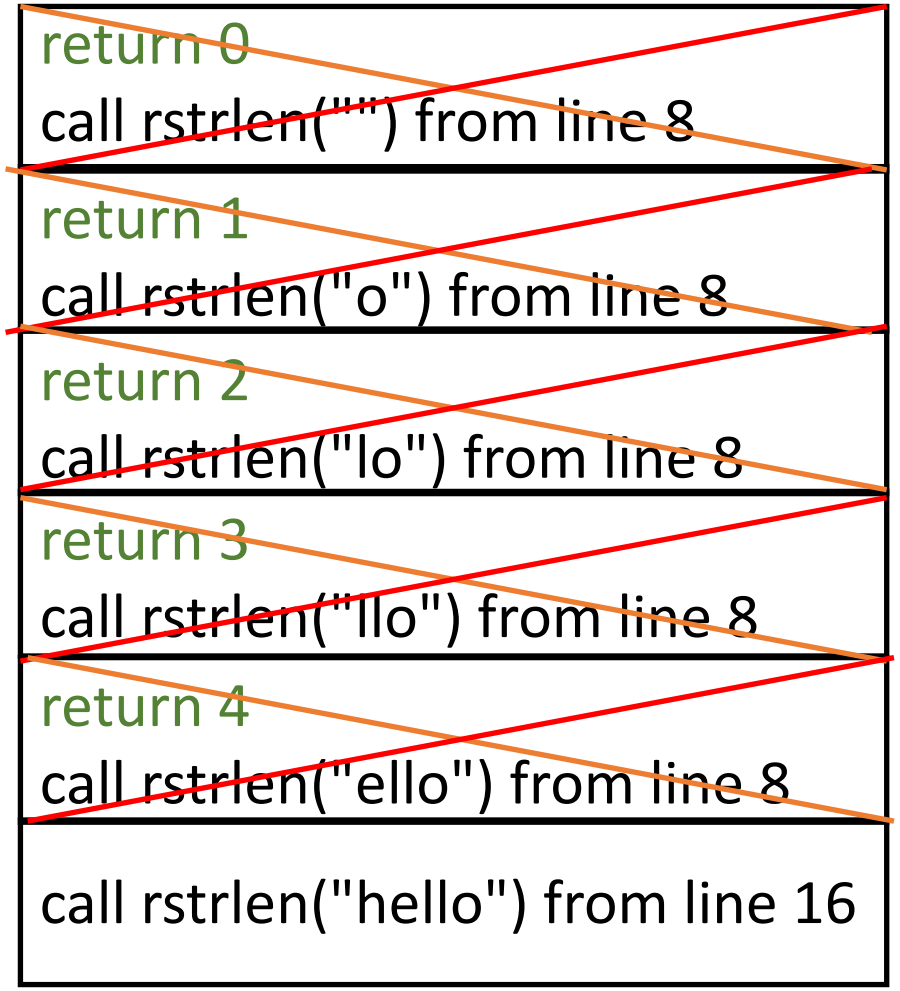
Return 4

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



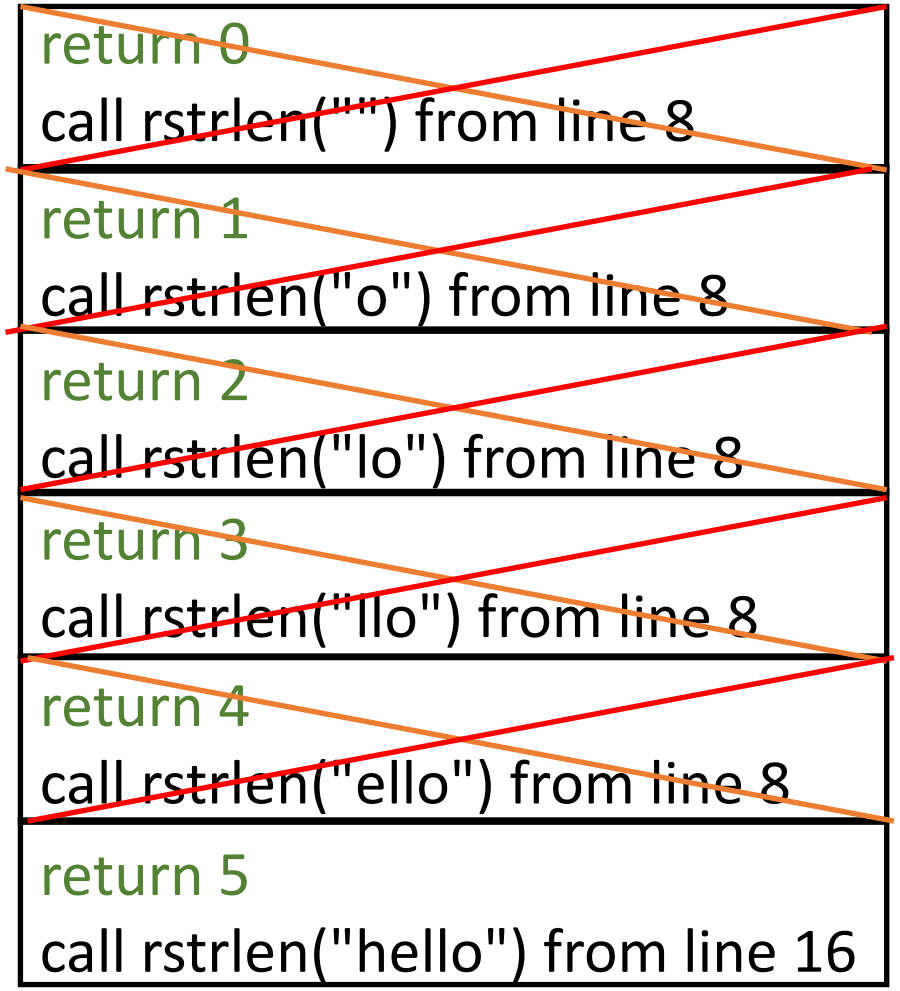
Return to rstrlen on Line 8

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



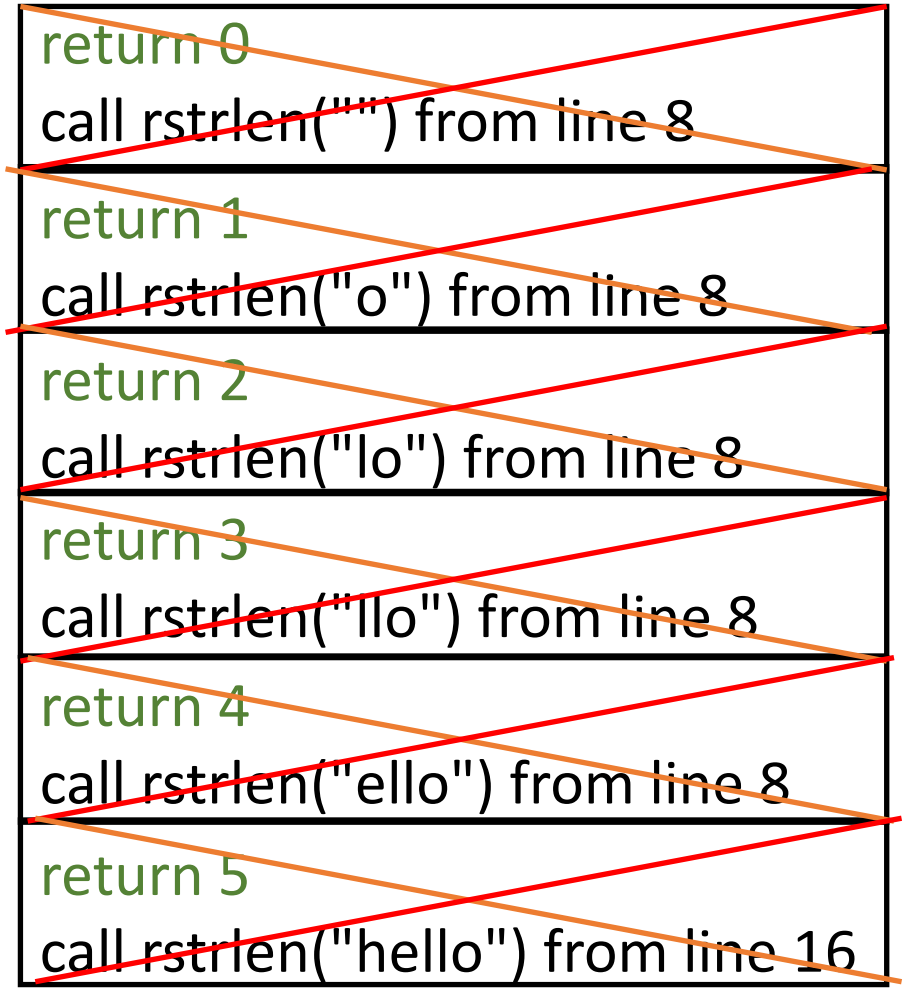
Return 5

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



Return to main on line 16

```
 1. int rstrlen(char *s)
 2. {
 3.    /* base case */
 4.    if (*s == '\0')
 5.           return(0);
 6.
 7.    /* recursive case */
 8.    return(1 + rstrlen(s+1));
 9. }
10. int main(void)
11. {
12.    char buf[1000];
13.
14.    strcpy(buf, "hello");
15.    printf("string '%s' length", buf);
16.    printf(" %d\n", rstrlen(buf));
17.    return(0);
18. }
```



Another Recursive Program: sort.c

- This sorts integers by finding the smallest number and putting it at the beginning
- Basic idea:

```
if number of elements in list is 1 or 0:
   list is sorted – just return
find the smallest number in the list
swap it and the first number
sort the rest of the list
```

sort.c Interface

```
void sort(int list[], int sizelist)
{
    /* base case */

    /* recursion */
}
```

list is array of integers to sort

sizelist is number of integers to sort

sort.c Base Case

- The list is sorted if there is either 0 elements or 1 element

```
/* base case */
if (sizelist <= 1)
    return;
```

- Return if so

sort.c Recursion, Part 1

```
/* find index of smallest number in array */
min = 0;
for(i = 1; i < sizelist; i++)
     if (list[i] < list[min])
          min = i;
```

- Do not include list[0] in the for loop
  - Assume initially it is the smallest element

sort.c Recursion, Part 2

```
    /* move smallest element to 0-th element */
    /* swapping it with whatever is there    */
    if (min != 0){
        tmp = list[0];
        list[0] = list[min];
        list[min] = tmp;
    }
    /* recurse */
    sort(&list[1], sizelist-1);
}
```

Ackermann's Function

- Important in theoretical computer science
- Formula:

  $A(0, n) = n + 1$

  $A(m + 1, 0) = A(m, 1)$

  $A(m + 1, n + 1) = A(m, A(m + 1, n))$
- For programming purposes:

  $A(0, n) = n + 1$

  $A(m, 0) = A(m - 1, 1)$

  $A(m, n) = A(m - 1, A(m, n - 1))$

Ackermann's Function, Part 1

```
int A(int m, int n)
{
    /* A(0, n) = n + 1 */
    if (m == 0)
        return(n + 1);
```

Ackermann's Function, Part 2

```
    /* A(m - 1, 0) = A(m - 1, 1) */
    if (n == 0)
        return(A(m - 1, 1));

    /*  A(m, n) = A(m - 1, A(m, n - 1)) */
    return(A(m - 1, A(m, n - 1)));
}
```

Print the Numbers from 0 to N Inclusive

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
18.
19. int main(void)
20. {
21.     fun1(0, 6);
22.     putchar('\n');
23.
24.     return(0);
25. }
```

Call fun1(0, 6) from Main

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

call fun1(n = 0, N = 6); return to line 21

Call fun2(1, 6) from fun1(0, 6)

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

0

call fun2(n = 1, N = 6); return to line 6
call fun1(n = 0, N = 6); return to line 21

Call fun1(2, 6) from fun2(1, 6)

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

```
0 1
```

fun1(n = 2, N = 6); return to line 15
fun2(n = 1, N = 6); return to line 6
fun1(n = 0, N = 6); return to line 21

Call fun2(3, 6) from fun1(2, 6)

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

```
0 1 2
```

fun2(n = 3, N = 6); return to line 6
fun1(n = 2, N = 6); return to line 15
fun2(n = 1, N = 6); return to line 6
fun1(n = 0, N = 6); return to line 21

Call fun1(4, 6) from fun2(3, 6)

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

```
0 1 2 3
```

fun1(n = 4, N = 6); return to line 15
fun2(n = 3, N = 6); return to line 6
fun1(n = 2, N = 6); return to line 15
fun2(n = 1, N = 6); return to line 6
fun1(n = 0, N = 6); return to line 21

Call fun2(5, 6) from fun1(4, 6)

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

```
0 1 2 3 4
```

fun2(n = 5, N = 6); return to line 6
fun1(n = 4, N = 6); return to line 15
fun2(n = 3, N = 6); return to line 6
fun1(n = 2, N = 6); return to line 15
fun2(n = 1, N = 6); return to line 6
fun1(n = 0, N = 6); return to line 21

Call fun1(6, 6) from fun2(5, 6)

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

```
0 1 2 3 4 5 6
```

fun1(n = 6, N = 6); return to line 15
fun2(n = 5, N = 6); return to line 6
fun1(n = 4, N = 6); return to line 15
fun2(n = 3, N = 6); return to line 6
fun1(n = 2, N = 6); return to line 15
fun2(n = 1, N = 6); return to line 6
fun1(n = 0, N = 6); return to line 21

Return from fun1(6, 6) to fun2(5, 6), line 15

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

```
0 1 2 3 4 5
```

~~fun1(n = 6, N = 6); return to line 15~~
fun2(n = 5, N = 6); return to line 6
fun1(n = 4, N = 6); return to line 15
fun2(n = 3, N = 6); return to line 6
fun1(n = 2, N = 6); return to line 15
fun2(n = 1, N = 6); return to line 6
fun1(n = 0, N = 6); return to line 21

Return from fun2(5, 6) to fun1(4, 6), line 6

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

```
0 1 2 3 4 5
```

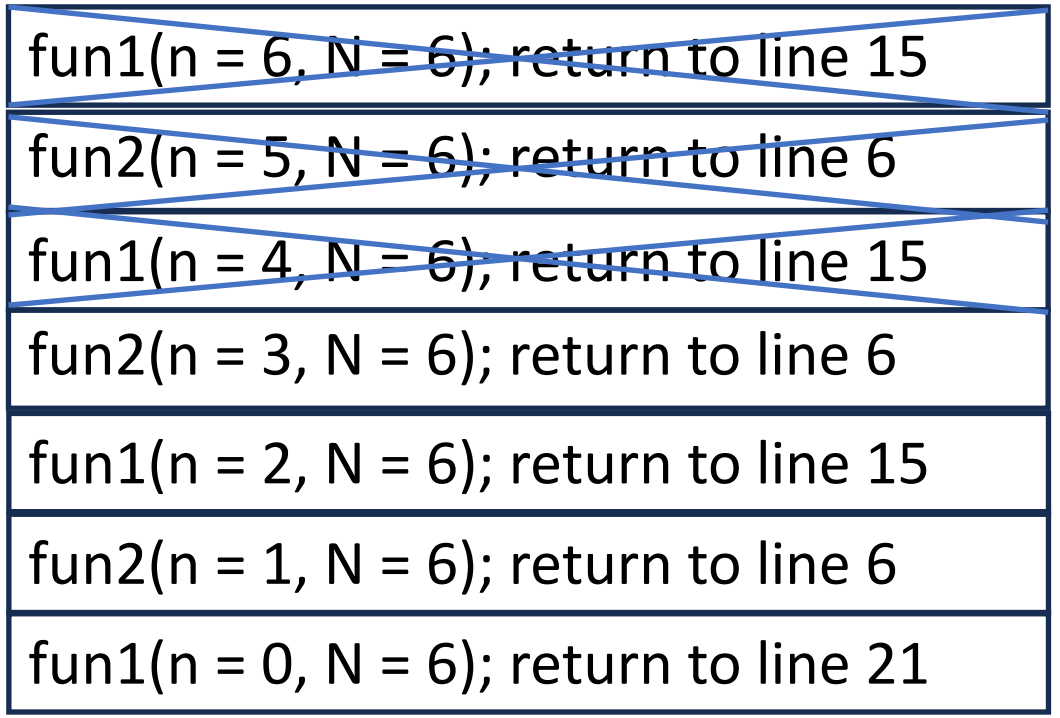
~~fun1(n = 6, N = 6); return to line 15~~
~~fun2(n = 5, N = 6); return to line 6~~
fun1(n = 4, N = 6); return to line 15
fun2(n = 3, N = 6); return to line 6
fun1(n = 2, N = 6); return to line 15
fun2(n = 1, N = 6); return to line 6
fun1(n = 0, N = 6); return to line 21

Return from fun1(4, 6) to fun2(3, 6), line 15

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

0 1 2 3 4 5

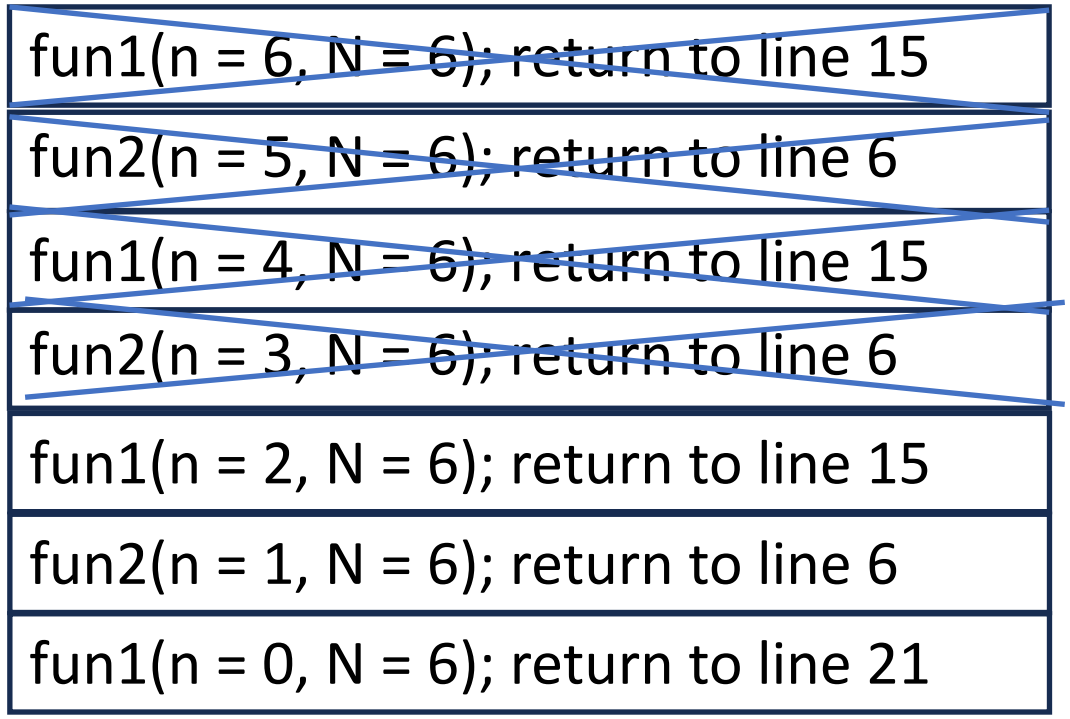


Return from fun2(3, 6) to fun1(2, 6), line 6

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

0 1 2 3 4 5

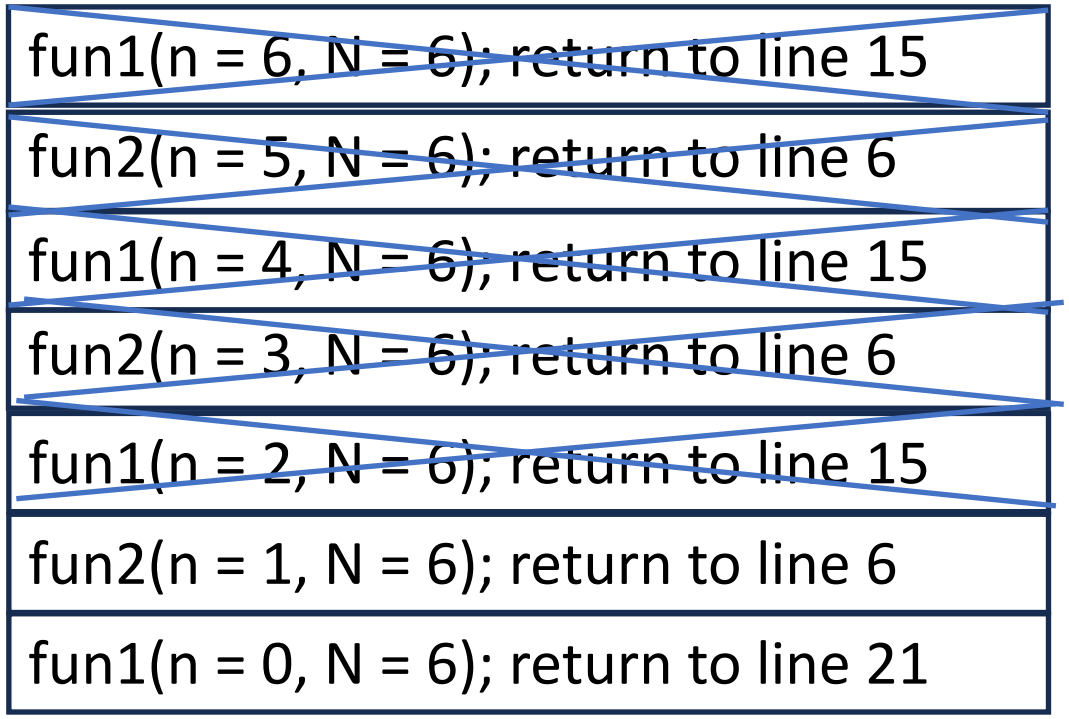


Return from fun1(2, 6) to fun2(1, 6), line 15

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

```
0 1 2 3 4 5
```

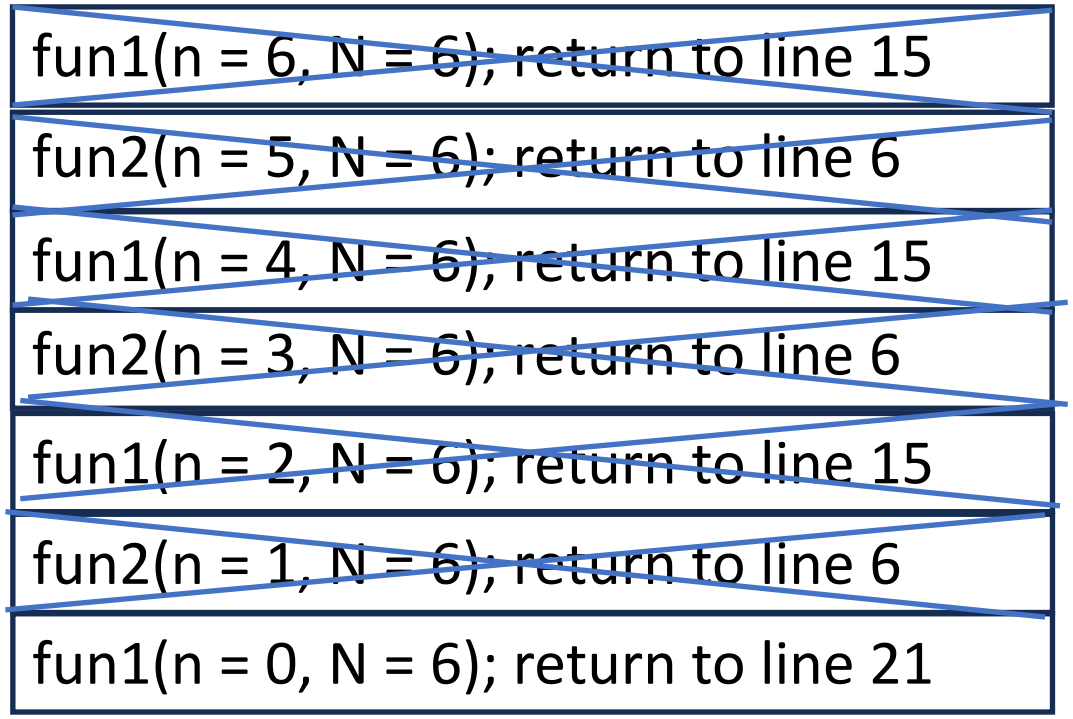


Return from fun2(2, 6) to fun1(0, 6), line 6

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

```
0 1 2 3 4 5
```

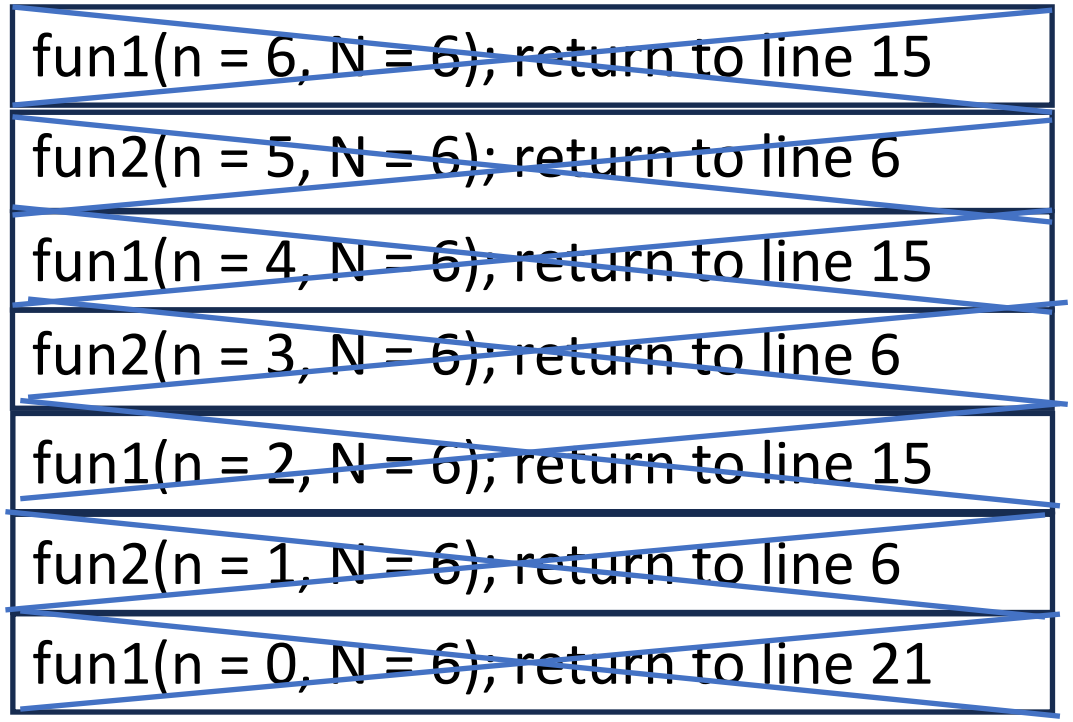


Return from fun1(0, 6) to main, line 19

```
 1. void fun1(int n, int N)
 2. {
 3.         if (n <= N){
 4.                 printf("%d ", n);
 5.                 n++;
 6.                 fun2(n, N);
 7.         }
 8. }
 9.
10. void fun2(int n, int N)
11. {
12.         if (n <= N){
13.                 printf("%d ", n);
14.                 n++;
15.                 fun1(n, N);
16.         }
17. }
```

output:

0 1 2 3 4 5



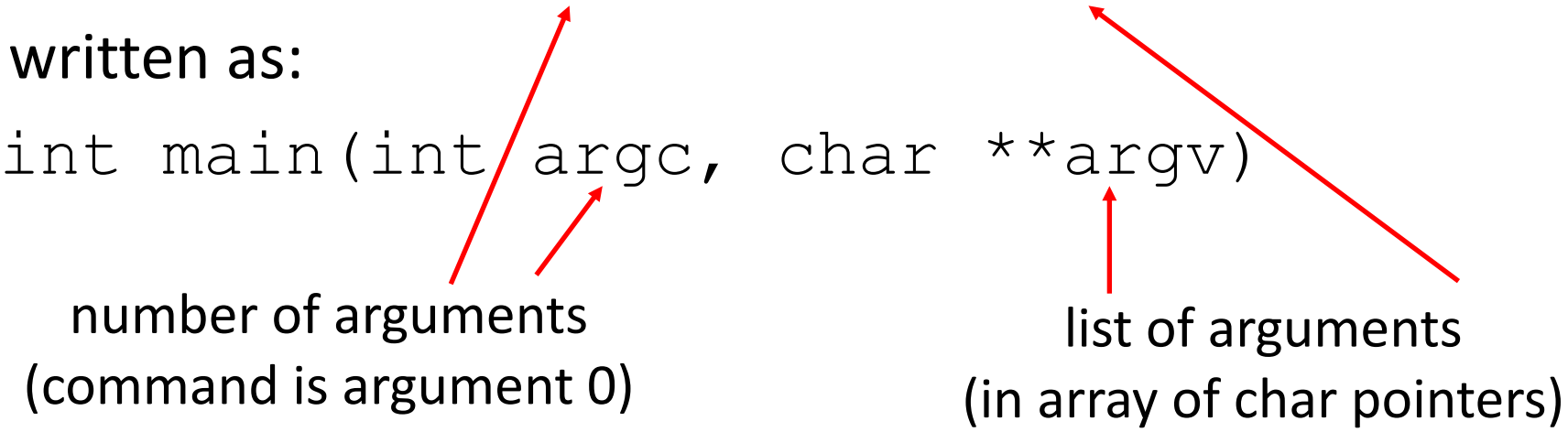
Command-Line Arguments

- Command is loopy 5 9
- Declaration of main function:

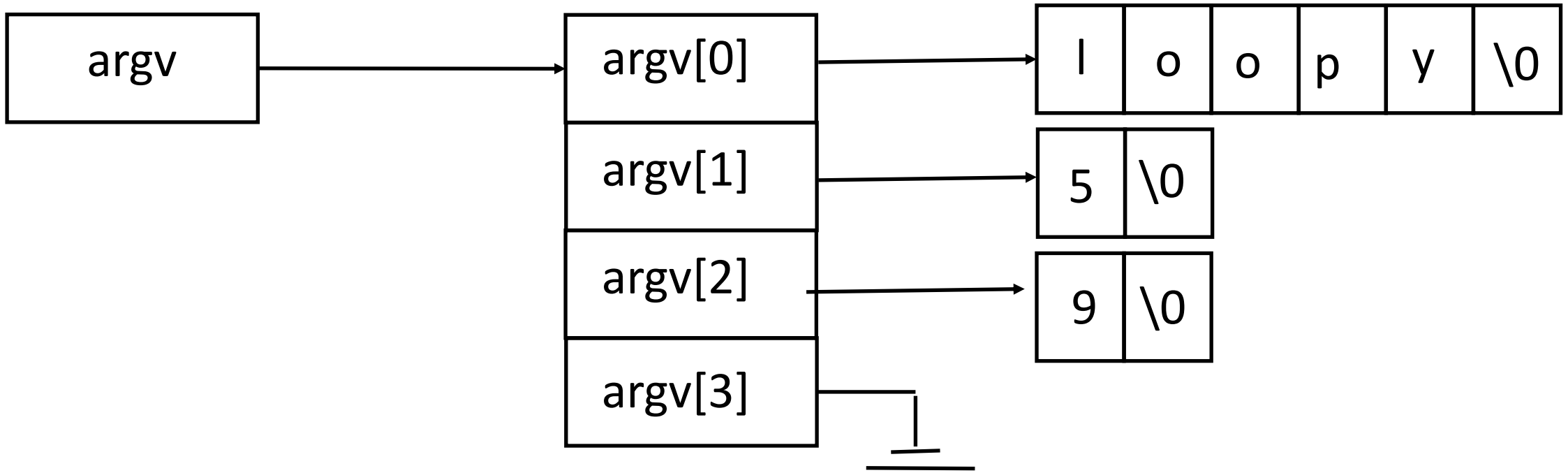
```
int main(int argc, char *argv[])
```

- Sometimes written as:

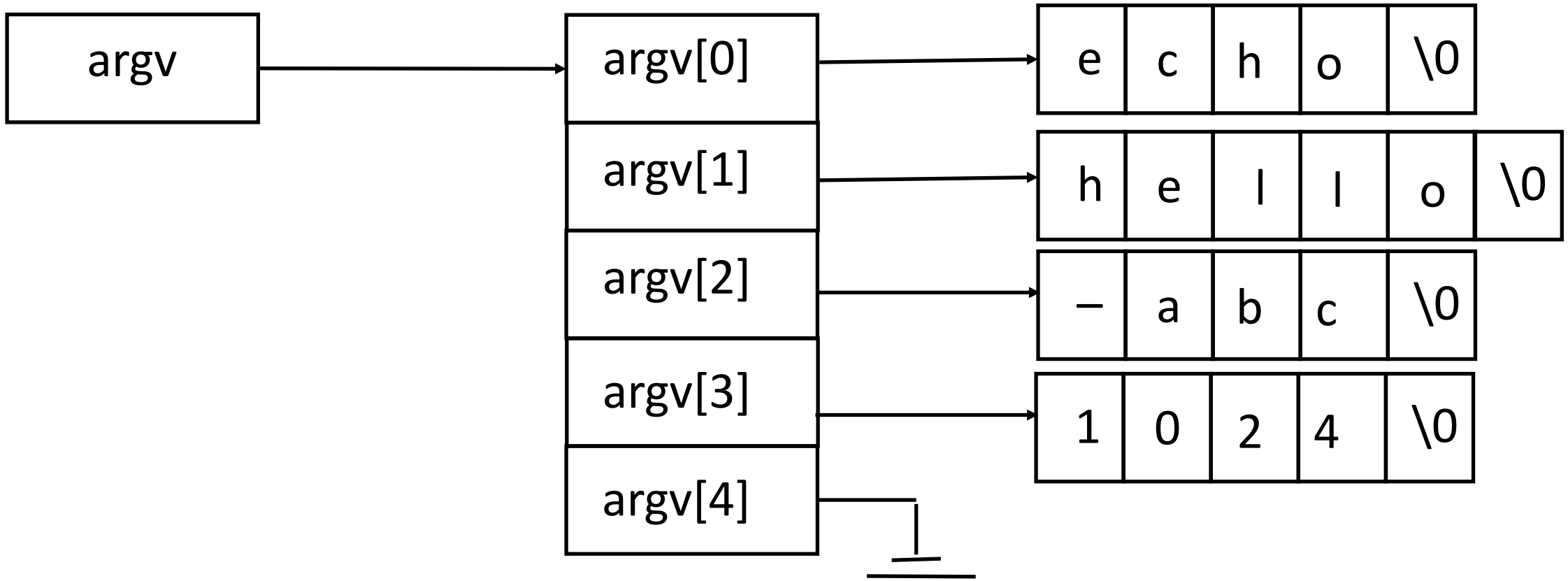
```
int main(int argc, char **argv)
```



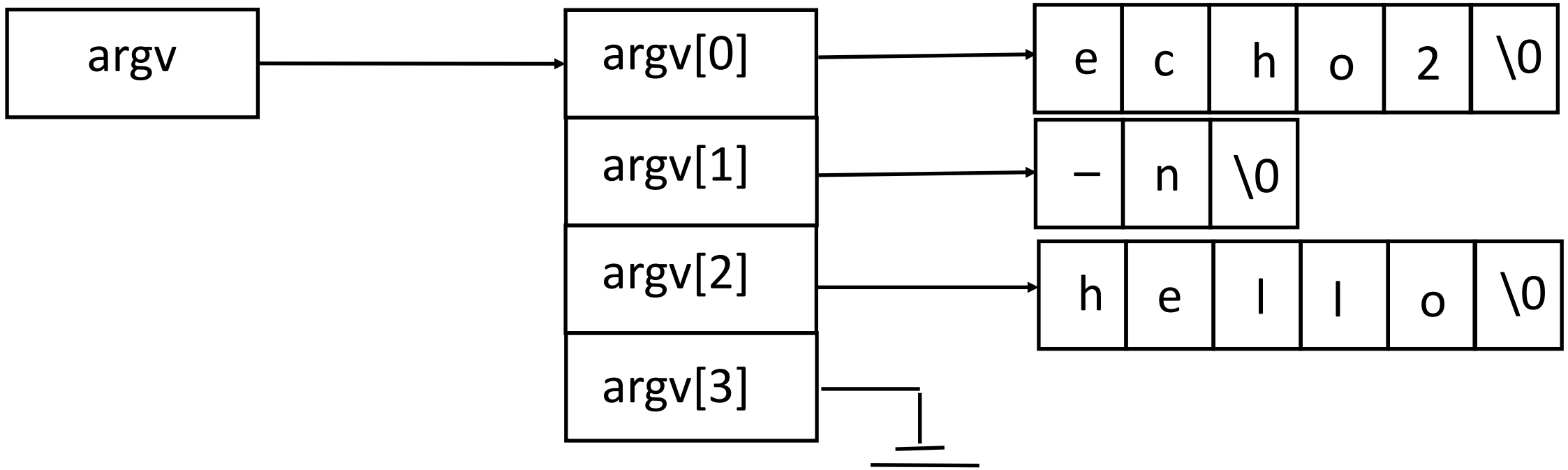
Visually:



Example: echo hello –abc 106



Option Example: echo2 –n hello



getopt

- Used for processing options to programs

```
int getopt(int argc, char *argv[], char *optstring)
```

- *argc*, *argv* are first two arguments to main()
- *optstring* is list of options
- Returns a single character:
  - It is an option given in *optstring*
  - '?' if an option given that is not in *optstring*

Options to getopt

- a single letter indicates a 1 character option
- a single letter followed by a colon '`:`' indicates a single character option that **must** be followed by an argument
  - `-e hello` or `-ehello` indicates the option 'e' with argument 'hello' (either form works)
- a single letter followed by 2 colons indicates a single character option that **may** be followed by an argument
  - `-xhello` indicates the option 'x' with (optional) argument 'hello'
  - `-x hello` indicates the option 'x' without an argument (note the separation between x and hello)

Other Variables

- `extern char *optarg` points to argument to option, if any
- `extern int optind` is index of next argument or option
- `extern int optopt` contains character that is an illegal option
  - On error, *getopt* returns '?'
  - It also prints an error message on stderr (but see opterr, below)
- `extern int opterr` says whether getopt should print error message on stderr when it encounters an illegal option
  - Set to 0 to suppress error message; set to non-zero to print it
  - Default is non-zero
  - On error, *getopt* returns '?'

Comma Operator

- `a = ( b , c )`: evaluate *b*, evaluate *c*, assign value of *c* to *a*
  - Parentheses needed as "," has lowest precedence
- Example: suppose a = 5, b = 2; then

  ```
  x = ( a = a+5 , b++ )
  ```

  sets a = 10, b = 3, and x = 2
  - Note it's "b++", so the value is used and then *b* is incremented
- Common use: prompting in a loop
- Example:

```
while(printf("> "), scanf("%d", &x) != EOF)
```

Useful String Functions

- Length: `strlen(char *str)` gives length of string `str`
- Copy: `strcpy(char *a, char *b)` copies contents of `b` into `a`
  - Better variant: `strncpy(char *a, char *b, unsigned int n)` copies first `n` characters of `b` into `a`, and if room adds a trailing '\0'
- Concatenation: `strcat(char *a, char *b)` appends contents of `b` to `a`
  - Better variant: `strncat(char *a, char *b, unsigned int n)` appends first `n` characters of `b` to `a`, and if room adds a trailing '\0'

Useful String Functions

- Comparison: `strcmp(char *a, char *b)` compares string `a` to string `b`, returns positive if `a` comes first, negative if `b` comes first, 0 if two are equal
  - `strncmp(char *a, char *b, unsigned int n)` does same but uses only first `n` characters of strings `a`, `b`
- First occurrence: `strchr(char *s, char c)`: return pointer to first occurrence of character `c` in `s`; if `c` is not in `s`, return **`NULL`**
- Last occurrence: `strrchr(char *s, char c)`: return pointer to last occurrence character `c` in `s`; if `c` is not in `s`, return **`NULL`**

Useful String Functions

- First occurrence of string: `strstr(char *s, char *p)`: return pointer to first occurrence of string `p` in `s`; if `p` is not in `s`, return **`NULL`**
- Break a string into parts: `strtok(char *s, char *d)`: return the next part of the string `s` (token) delimited by characters in `d`; if none, return **`NULL`**
  - If you want to do it repeatedly to the same string, set `s` to **`NULL`** on the second and subsequent calls
- Duplicate a string: `strdup(char *s)`: returns a pointer to a copy of string `s`