CMPE 252
C PROGRAMMING

SPRING 2021

WEEK 2-3

Common Errors

- if (0 <= x <= 4)
  - printf("Condition is true\n");

- For example, let's consider the case when `x` is `5` .
- The value of `0 <= 5` is `1` , and `1` is certainly less than or equal to `4`!
- In order to check if `x` is in the range `0` to `4` , you should use the condition
  - (0 <= x && x <= 4)

- if (x = 10)
  - printf("x is 10");

What about this one ?

```
if (x > 0)
        sum = sum + x;
        printf("Greater than zero\n");
else
        printf("Less than or equal to zero\n");
```

Null Statement

- The null statement is merely a semicolon alone.
- ;
- A null statement does not do anything. It does not store a value anywhere. It does not cause time to pass during the execution of your program.
- Most often, a null statement is used as the body of a loop statement, or as one or more of the expressions in a for statement. Here is an example of a for statement that uses the null statement as the body of the loop (and also calculates the integer square root of n, just for fun):
- for (i = 1; i*i < n; i++)
-   ;
- Here is another example that uses the null statement as the body of a for loop and also produces output:
- for (x = 1; x <= 5; printf ("x is now %d\n", x), x++)
-   ;
- A null statement is also sometimes used to follow a label that would otherwise be the last thing in a block.

«break» in loops

- You can use the **break** statement to terminate a *while*, *do*, *for*, statement. Here is an example:

```
int x;
for (x = 1; x <= 10; x++)
 {
   if (x == 8)
     break;
   else
     printf ("%d ", x);
 }
```

«continue» in loops

- You can use the **continue** statement in loops **to terminate an iteration of the loop and begin the next iteration**. Here is an example:
- for (x = 0; x < 100; x++)
- {
- if (x % 2 == 0)
- continue;
-
- sum_of_odd_numbers + = x;
- }
- If you put a continue statement inside a loop which itself is inside a loop, <u>then it affects only the innermost loop.</u>

RECURSION
CHAPTER 9

Problem Solving & Program Design in C

Eighth Edition
Global Edition

Jeri R. Hanly & Elliot B. Koffman

Recursion

- A recursive function is one that calls itself or that is part of a cycle in the sequence of function calls.

- The ability to invoke itself enables a recursive function to be repeated with different parameter values.

Recursion

- It can be used an an alternative to iteration - looping.
- Recursion is typically used to specify a natural, simple solution that would otherwise be very difficult to solve.

The Nature of Recursion

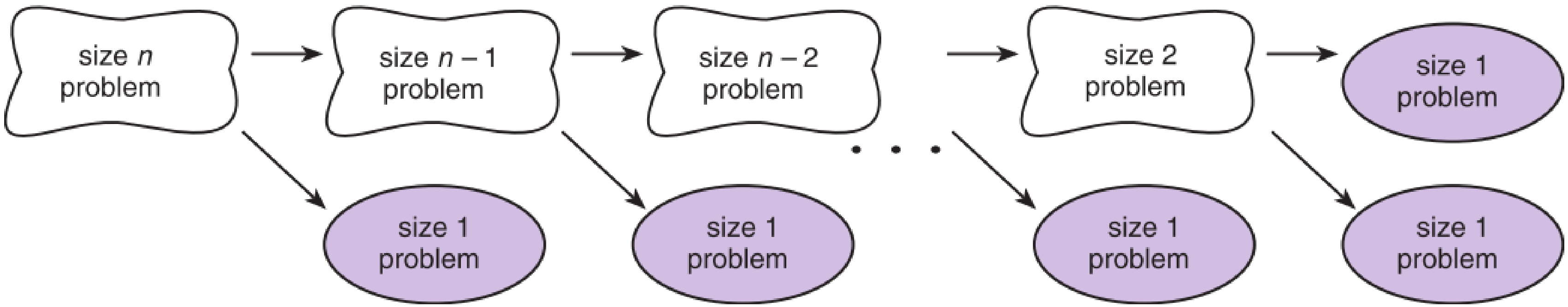
- One or more simple cases of the problem have a straightforward, nonrecursive solution.
- The other cases can be redefined in terms of problems that are closer to the simple cases.

The Nature of Recursion

- By applying this redefinition process every time the recursive function is called, eventually the problem is reduced entirely to simple cases, which are relatively easy to solve.

if this is a simple case
    solve it
else
    redefine the problem using recursion

FIGURE 9.1 Splitting a Problem into Smaller Problems



Quick Check – Multiply: x*y

```c
int multiply(int m, int n)
{
    int ans;

    if (n == 1)
          ans = m;     /* simple case */
    else
          ans = m + multiply(m, n - 1); /* recursive step */

    return (ans);
}

int main(void)
{
    printf("Result is: %d", multiply(6,3));
}
```

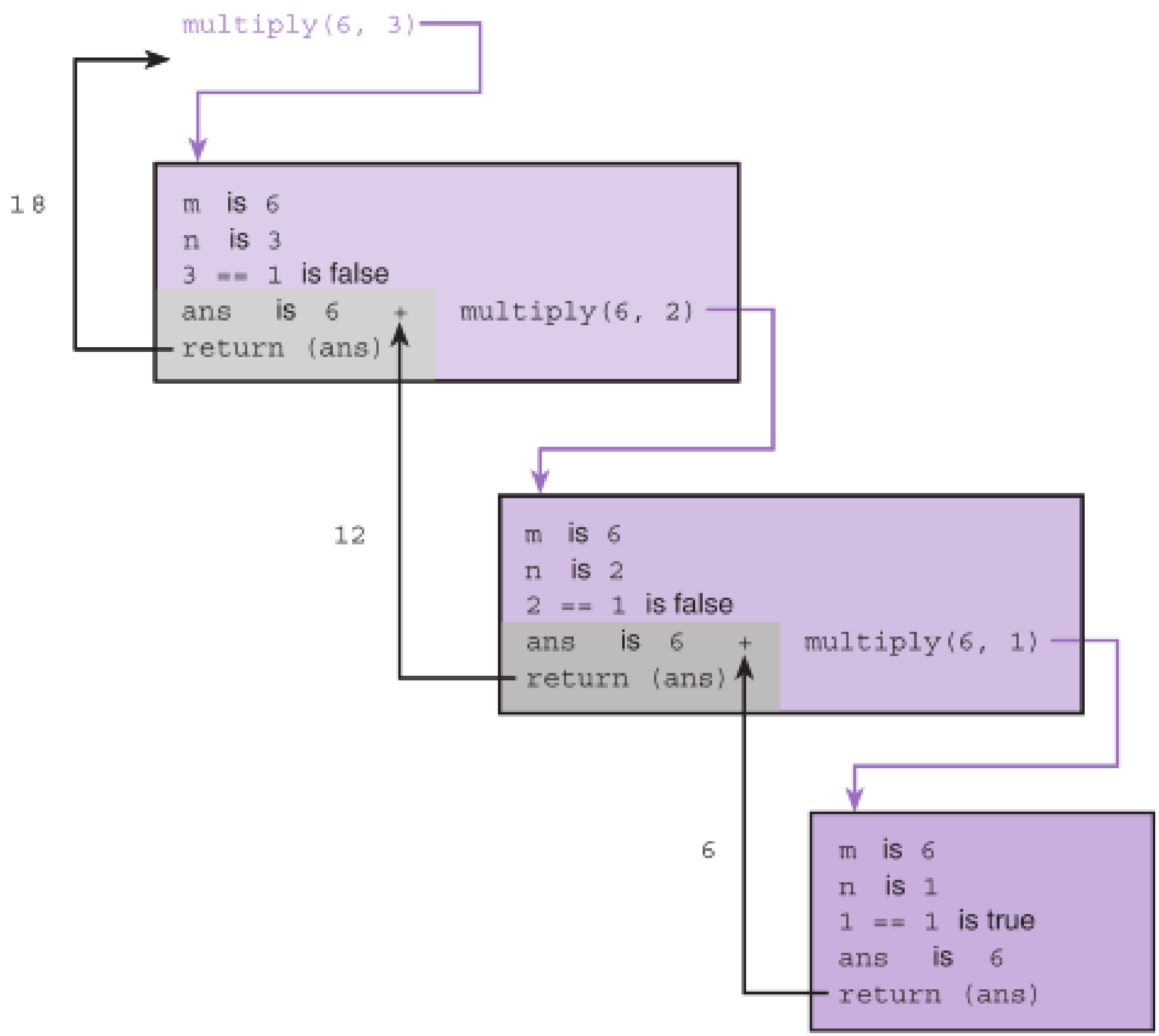
SPLIT:

- Multiply 6 by 2
- Add 6 to the result

- Multiply 6 by 1
- Add 6 to the result

Tracing Recursive Functions

- activation frame
  - representation of one call to a function
- terminating condition
  - a condition that is true when a recursive algorithm is processing a simple case
- system stack
  - area of memory where parameters and local variables are allocated when a function is called and deallocated when the function returns



multiply(6, 3)
18
m is 6
n is 3
3 == 1 is false
ans is 6 +
return (ans)
multiply(6, 2)
12
m is 6
n is 2
2 == 1 is false
ans is 6 +
return (ans)
multiply(6, 1)
6
m is 6
n is 1
1 == 1 is true
ans is 6
return (ans)

Quick Check

- Write a recursive function which computes the sum of its two integer parameters.

```
int add(int m, int n)
{
        int ans;
        if (n == 0)
                ans = m;
        else
                ans = 1 + add(m, n-1);

        return (ans);
}
```

Parameter and Local Variable Stacks

- stack
  - a data structure in which the last data item added is the first data item processed
- C keeps track of the values of variables from different recursive function calls by using a stack data structure.

Implementation of Parameter Stacks in C

- The compiler actually maintains a single system stack for the tasks.

- system stack
  - area or memory where parameters and local variables are allocated when a function is called and deallocated when the function returns

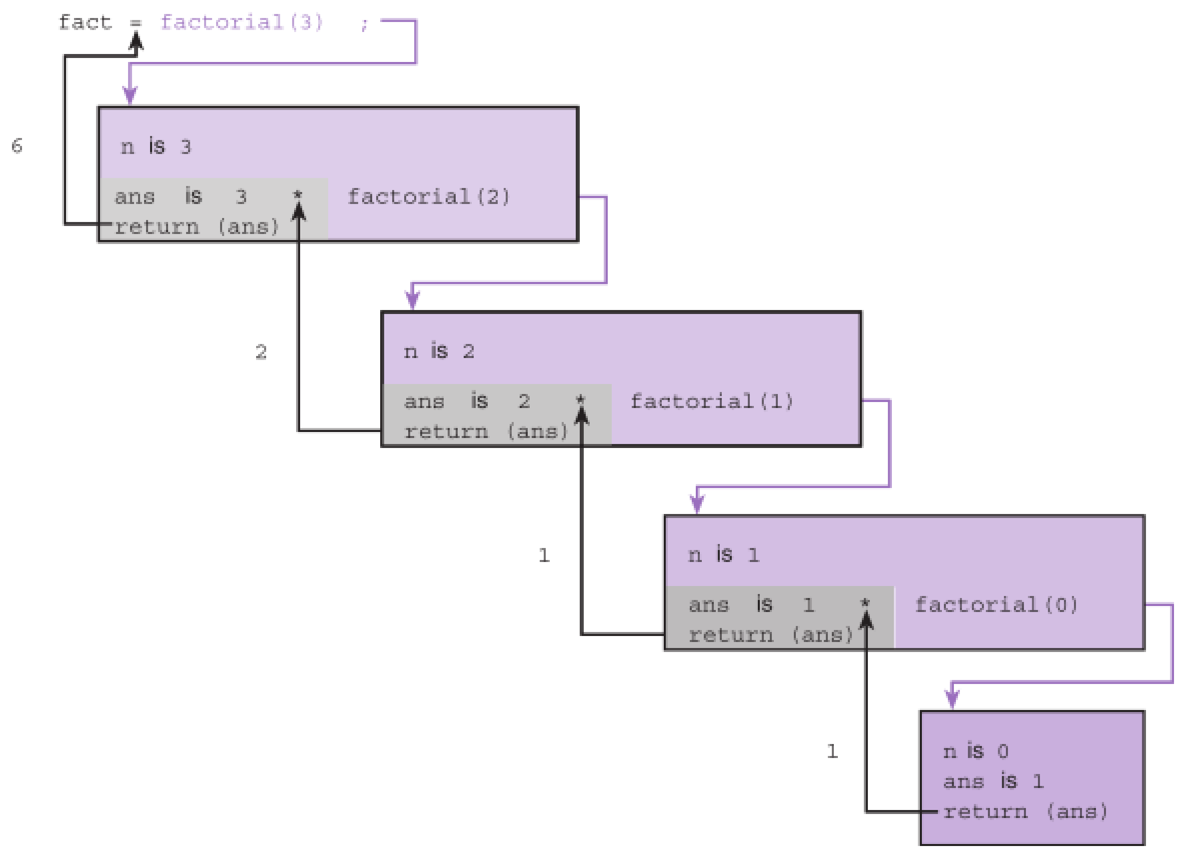
```
Stack trace of multiply(6, 3)   n   m    ans
                                3   6     ?

Recursive call multiply(6, 2)   n   m    ans
                                3   6     ?
                                2   6     ?

Recursive call multiply(6, 1)   n   m    ans
                                3   6     ?
                                2   6     ?
               Returns 6        1   6     ?,then 6


               multiply(6, 2)   n   m    ans
                                3   6     ?
               Returns 12       2   6     ?, then 12


               multiply(6, 3)   n   m    ans
               Returns 18       3   6     ?, then 18

Stacks are now empty.
```

Quick Check – Recursive Factorial

```c
int factorial(int n)
{
	int ans;

	if (n == 0)
		ans = 1;
	else
		ans = n * factorial(n - 1);

	return (ans);
}

int main(void)
{
	printf("Result is: %d", factorial(3));
}
```

FIGURE 9.11

Trace of fact = factorial(3);



Iterative Factorial

```
int factorial(int n)
{
    int i,              /* local variables */
        product = 1;

    /* Compute the product n x (n-1) x (n-2) x . . . x 2 x 1 */
    for (i = n; i > 1; --i) {
       product = product * i;
    }

    /* Return function result */
    return (product);
}
```

Fibonacci

- Sequence is 1, 1, 2, 3, 5, 8, 13, 21, 34, ...

- Fibonacci_1 is 1
- Fibonacci_2 is 1
- Fibonacci_n is Fibonacci_{n-2} + Fibonacci_{n-1} for $n > 2$

Quick Check: Write Fibonacci function in a recursive way

```
int fibonacci(int n)
{
        int ans;

        if (n == 1 || n == 2)
                ans = 1;
        else
                ans = fibonacci(n - 2) + fibonacci(n - 1);

        return (ans);
}
```

GCD

- greatest common divisor of two integers is the largest integer that divides them both evenly

Quick Check: Write GCD function in a recursive way

- gcd(*m*,*n*) is *n* if *n* divides m evenly
- gcd(*m*,*n*) is gcd(*n*, remainder of *m* divided by *n*) otherwise

```c
#include <stdio.h>

int gcd(int m, int n)
{
        int ans;

        if (m % n == 0)
                ans = n;
        else
                ans = gcd(n, m % n);

        return (ans);
}
int main(void)
{
        int n1, n2;

        printf("Enter two positive integers separated by a space> ");
        scanf("%d%d", &n1, &n2);
        printf("Their greatest common divisor is %d\n", gcd(n1, n2));

        return (0);
}
```

Example: Reverse a sentence from the user command prompt using recursion

```c
#include <stdio.h>
void reverseSentence();
int main() {
    printf("Enter a sentence: ");
    reverseSentence();
    return 0;
}

void reverseSentence() {
    char c;
    scanf("%c", &c);
    if (c != '\n') {
        reverseSentence();
        printf("%c", c);
    }
}
```

Output

```
Enter a sentence: margorp emosewa
awesome program
```

Pro’s and Con’s

- CONS:
  - **Recursion uses more memory** : function is added to the stack at each recursive call and keep the local variables until the call is finished.
  - **Recursion can be slow :** since it contains function calls . (iteration is more efficient)
- PROS:
  - **For certain problems, they are easier to solve using Recursion**
  - **and Recursion enables less complex, more clear and understandable code**
  - *(for instance, binary search problems, tree traversals etc.)*

Everything, written using Recursion can be implemented using Iteration and vice versa

References

1. Problem Solving & Program Design in C, Jeri R. Hanly & Elliot B. Koffman, Pearson 8. Edition, Global Edition