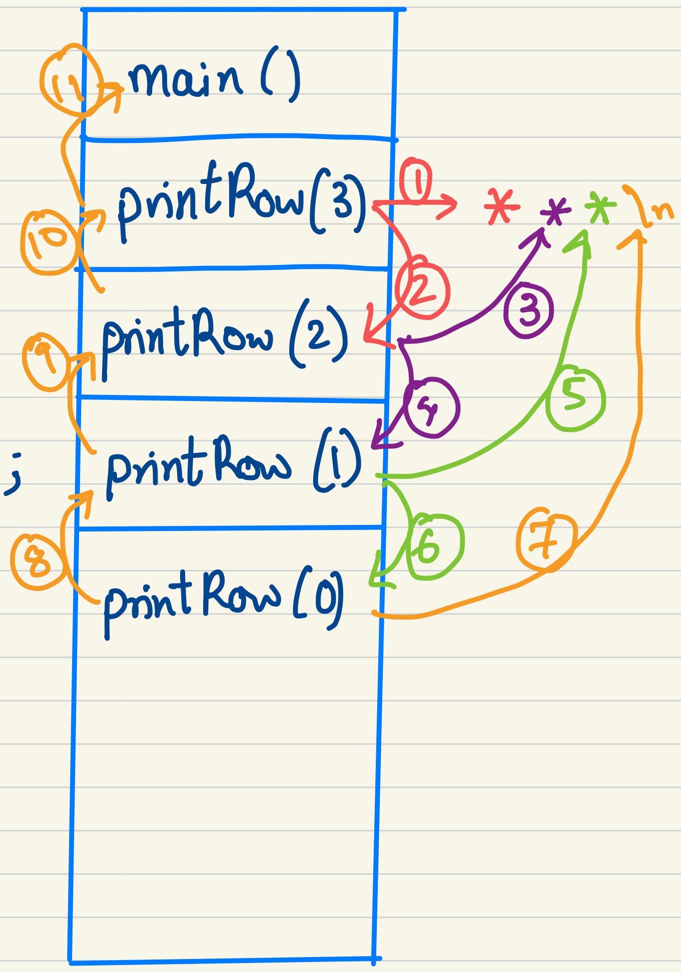
APS 105 Lecture 27 Notes

Last day: We discussed 2 more functions from string library strstr and strchar, and we introduced recursion.

Today: we continue discussing about recursion.

Recall:

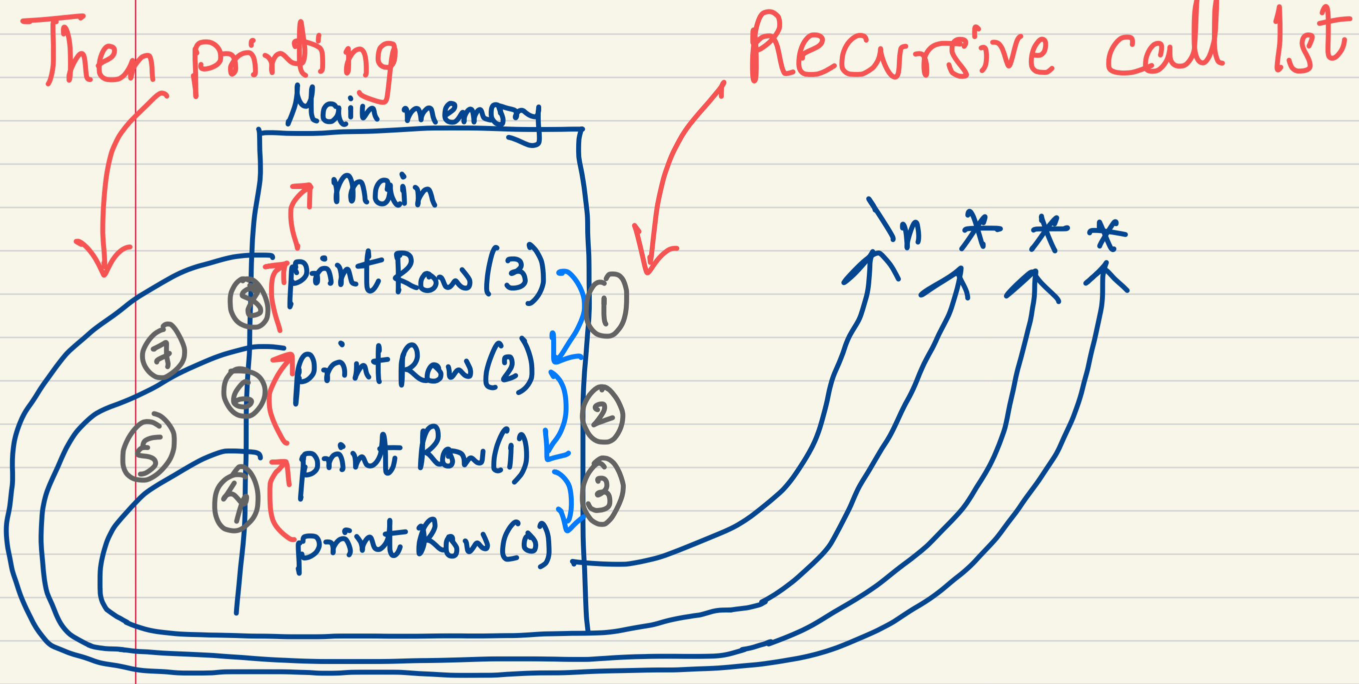
```
void printRow(int n){
        if(n==0)
                printf("\n");
        else {
                printf("*");
                printRow(n-1);
        }
}

int main(){
        int n= 3;
        printRow(n);
        return 0;
}
```



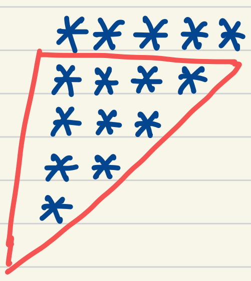
What happens if we switch order of printing and recursive call.

```
void printRow( int n) {
        if (n == 0  ){
              printf("\n");
        }
        else {
              printRow (n-1);
              printf("*");
        }
}
```

→ return happens here



Suppose we want to print a triangle recursively, like this



5 ⤳ print row of 5 * then
4 print triangle of 4
3
2
1

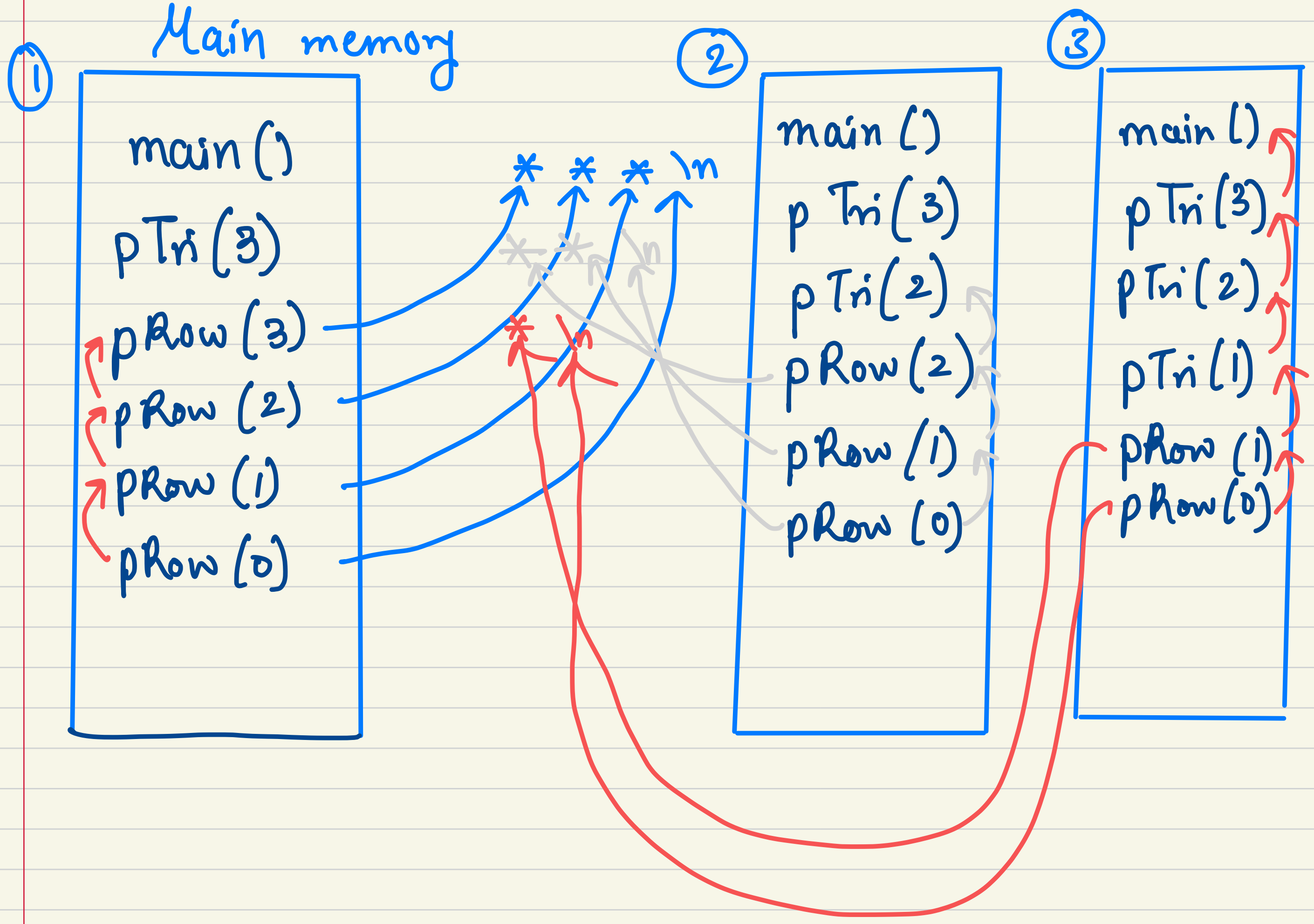
Recursive Thinking

```
void printTriangle(int n){
        if(n > 0) {
                printRow(n);
                printTriangle(n-1);
        }
        return;
}
```

Recursive call

Base case: n<=0 → do nothing then return

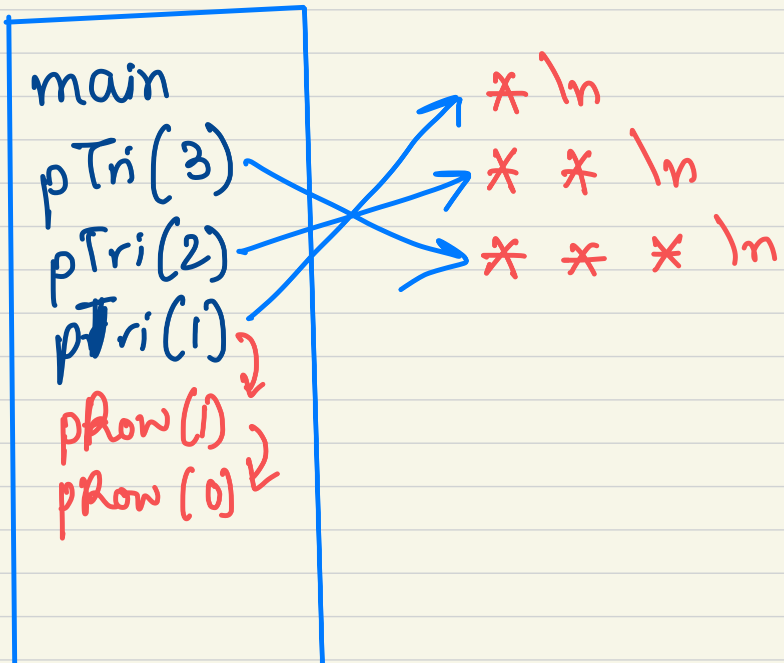
Main memory



How can we print the following pattern?
(Inverted)

```
*
* *
* * *
* * * *
* * * * *
```

```
void printTriangle(int n) {
        if(n > 0) {
                printTriangle(n-1);
                printRow(n);
        }
}
```

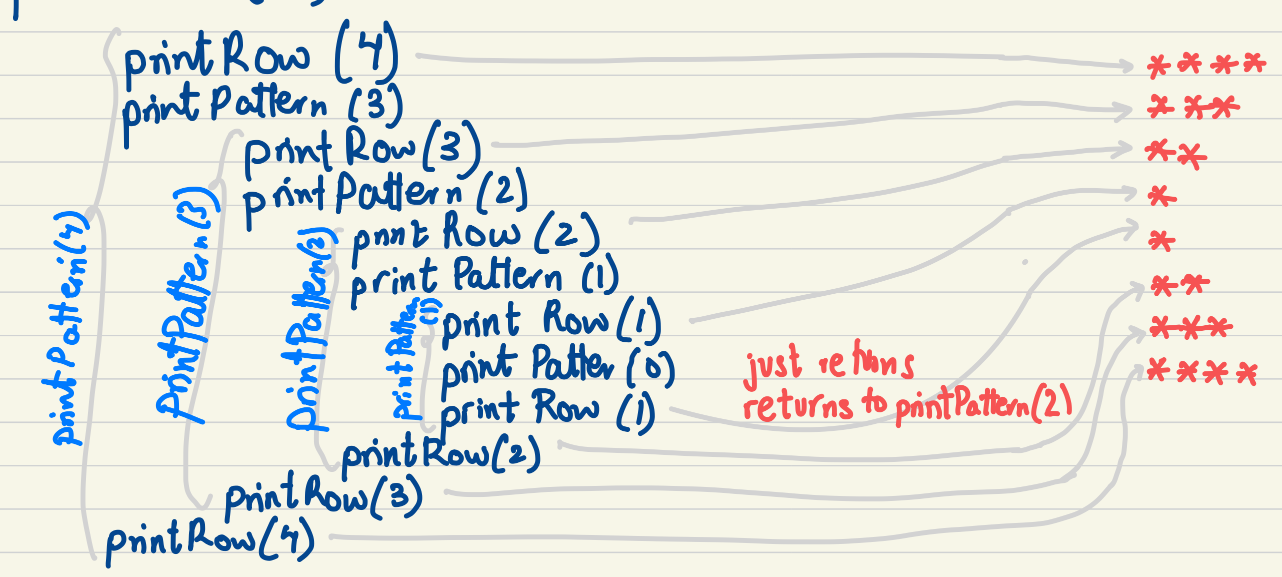


How can we print the following pattern?

```
* * * *
* * *
* *
*

*
* *
* * *
* * * *
```

```
void printTriangle(int n){
        if(n>0){
                printRow(n);
                printPattern(n-1);
                printRow(n);
        }
        return;
}
```

printPattern(4)



```
    printRow(4)                                  → ****
    printPattern(3)                              → ***
        printRow(3)                              → **
        printPattern(2)                          → *
            printRow(2)                          → *
            printPattern(1)                      → **
                printRow(1)                      → ***
                printPatter(0)                   → ****
                printRow(1)
            printRow(2)
        printRow(3)
    printRow(4)
```

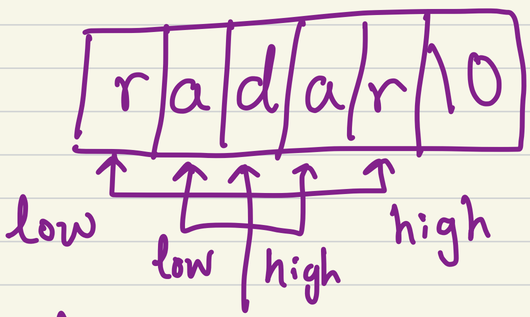
In real-life, we avoid recursion:

- Although it is easier to understand, recursion is not optimal $\longrightarrow$ consumes stack
  - Takes time to call function
  - Takes a lot of space
  - if problem is large, stack can overflow.

Recursion with strings

A String is an array of characters. To think of strings recursively, think of a string as a character followed by a string OR char preceded by a string OR two characters enclosing a string

Write a "recursive" function to determine if a string is a palindrome.



Check edges then the string enclosed is a smaller problem.

```
bool   isPalindromeRecursive(char *s, int low, int high){
          bool result;
          if(low == high)
                  result = true;
          else if( s[low] != s[high])
                  result = false;
          else
                  result = isPalindromeRecursive(
                             charPalindrome(s, low+1,
                                              high-1);
          return result;
}

int main(){
          char s[]= "level";
          printf("Is %s palindrome? %d", level,
                    isPalindromeRecursive(s,0, strlen(s)-1));
          return 0;
}
```