APS 105 Lecture 10 Notes

Last lecture: for loops, when to use which loop type, nested loops

This lecture: Another nested loop example, functions

Recall: nested loops is where we have loops within loops.

Last lecture we drew n rows of the following patter

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

prints triangle of stars

```
for (int row = 1; row <= 4; row++){
    for (int col = 1; col <= row ; col++)
        printf("*");
    printf("\n");
}
```

prints 1 line of stars

How about printing n rows of this pattern?

```
   1 2 3 4 5
1  ⊔⊔⊔⊔*
2  ⊔⊔⊔**
3  ⊔⊔***
4  ⊔****
5  *****
```

E.g. To print 5 rows

1st row: leave 4 spaces + 1 * = 5
2nd row: leave 3 spaces + 2 * = 5
3rd row: leave 2 spaces + 3 * = 5
4th row: leave 1 space + 4 * = 5
5th row: leave $\underbrace{0}_{5-\text{row}\#}$ spaces + $\underbrace{5}_{\text{row}\#}$ * = 5

We need 1 loop for rows, 1 loop for columns (as before)
Every row we need $\underbrace{\text{a number of spaces}}_{=5-\text{row}\#}$ then a $\underbrace{\text{number of stars}}_{=\text{row}\#}$

$\longrightarrow$

```
for (int row = 1 ; row <= 5 ; row++){
        for (int col = 1 ; col <= 5 ; col++){
              if ( col <= 5-row )
                     printf("  ");
              else
                     printf("* ");
        }
        printf("\n");
}
```

have 5-row# spaces before printing stars

Can also be written as col + row <= 5

Homework:

Write nested for loops to print the following pattern:

```
        *          1 *
      * * *        3 *
    * * * * *      5 *
  * * * * * * *    7 *
* * * * * * * * *  9 *
```

Remember we already printed in the previous example, i.e. number of spaces before printing stars is the same, you'll need to change how you'll print stars

Break Software into Manageable Pieces: Functions

- Want to build a complex product? It is difficult to handle all at once

- Better: break the big stuff into pieces and assemble them e.g. ikea

- In software, separate pieces are called functions (or subroutines, modules, procedures) → Modularity

- We can divide the work among different developers, each will create a separate piece or function

- Let's play a game.
- e.g. printf, scanf, rint, pow, rand

  You don't need to implement details of printf everytime you want to use it, just call the function

- Can test each function in isolation

- Avoid repeating code and making mistakes as you repeat.

Scanned with CamScanner

order of passing parameters matter

```
#include <stdio.h>

void printNChars (int numOfChars, char c){
        for (int count=1; count <=numOfChars; count++)
            printf("%c", c);

}

void printTriangle (int numOfRows){
        for(int row=1; row <=numOfRows; row++){
            printNChars( row , '*');
            //printTriangle "calls" printNChars

        }
}

int main (void){

    int n=5;
    printTriangle (n); //main "calls" printTriangle func.
    return 0;

}
```

didn't put type

Cannot switch order as a function should be having a prototype or implementation before calling it

ces.

* Main will always be executed 1st

* Functions can be either:

    (i) implemented before main (and then called in main) as in example.

    (ii) prototype of functions written before main, but functions are implemented after main.

    prototype gives i) return type, ii) function name, iii) parameter types

E.g.

```
#include <stdio.h>

//Function prototype        you can also give names (but unnecessary)
void printNChars( int , char);
                                        nothing is returned
void printTriangle(int);

int main (void) {

     int n= 5;
     printTriangle (n);
     return 0;
}                                        types of parameters
void printNChars (int numOfChars, char c){
                                        names of parameters
     for (int count=1; count <= numOfChars; count++)
          printf("%c", c);
}
void printTriangle ( int numOfRows){
     for (int row=1; row<= numOfRows; row ++)
          printNChars(row, '*');
}                 passed by its name in printTriangle
```

Order doesn't matter if implementation is after main

* When you call a function, pass the parameter by its name (not type)

* parameter passed to a function will take name of the parameter in the header of the function implementation

* Order of parameters passed when you call a function should be same as the order of parameters in function header.