APS105 Lecture 11 Notes

Last lecture: Example with nested loops and introduction to functions

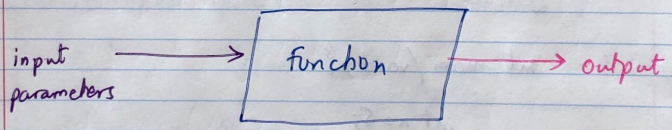
Today: More on how to declare functions, how to communicate with them

```
_____ _____ ( _____ )
```

return type – output type

function name

parameters to function – input



Recall:

```
void printStars ( int numOfStars ) {
    for (int count=1; count <= numOfStars; count++)
        printf("*");
    printf("\n");
}
```

In .c files, you have 2 ways to declare your functions, either before main or after main.

[ ] Before main (2 in 1 — declare & implement the function before main; main will be the last function)

```
#include <stdio.h>

void printStars(int numOfStars){

    for(int count=1; count<=numOfStars; count++)
        printf("*");
    printf("\n");
}

int main(void){

    int n=5;
    printStars(n);
    return 0;
}
```

2] Declare before main and implement after main - personal favorite

```
#include <stdio.h>

void printStars(int numOfStars);

int main(void) {

    int n=5;
    printStars(n);
    return 0;
}

void printStars(int numOfStars) {

    for(int count=1; count<=numOfStars; count++){
        printf("*");
    }
    printf("\n");
}
```

not necessary to name passed parameters

Forward Declaration

we call this a function prototype.

remember ;

Compiler will not complain when it sees your function in main, although it is not implemented yet.

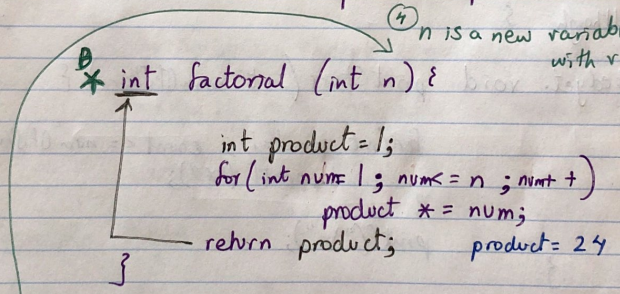
Can return type be non-void?

Yes, it can be any data type: int, double, char, bool

E.g. Recall factorial $n! = n * (n-1) * (n-2) \cdots 3 * 2 * 1$

Write a C function that returns factorial of an int type "argument"

↳ parameter passed to function



① Execution starts at main function

```
int main (void) {
    int value = 2, factValue;
A   factValue = factorial (value+2);
    printf("factorial of %d is %d", value+2, factValue);
    return 0;
}
```

⑤ evaluates to 24

⑥ factValue = 24

② value+2

③ call factorial with argument 4

When a function is called:

① evaluate argument, e.g value + 2

② Control goes from A * to B *

③ Parameters are assigned values only

The evaluated number from value + 2 is put into n in factorial function
↳ This is called "call by value"
↓
The value is passed to the function (value only)

④ The function executes and returns a value

assign it to the variable in main. Main proceeds normally.

Variable Scope:

Recall when in for

e.g.
```
for(int count = 1; count <= n; count++){
    printf("*");
}
```

not allowed as count is out-of-scope →
```
}
count = 7;
```

* Variable scope is where the variable is defined

Every function has its own variables, you can't use these variables outside the scope of the function.

↳ I can have variable n in main & variable n in function, but they are different variables.

E.g.
```
int divideByTwo(int n){ //copy 7 to divideByTwo::n
    n = n/2             // divideByTwo::n/2 = 3
    return n;           //return 3
}
```
3

One main takeaway from today's lecture

Very important:

n in main didn't change by change in divideByTwo!

```
int main (void){
    int result, n = 7;  //main::n = 7
    result = divideByTwo(n); // 3 is assigned to main::result
    return 0;
}
```