COMP20005 Workshop Week 5

```
1   discuss: functions, ex. 5.1, 5.2, 5.3
2   sample Quiz
3   implement 5.05 and 5.06
4   while (having_time()){
5     review/ask questions on quiz 1
6     implement("an exer. from grok C05");
    }
```

- x= 2.0, need to print out $\sqrt{x}$
- How to compute $\sqrt{x}$?

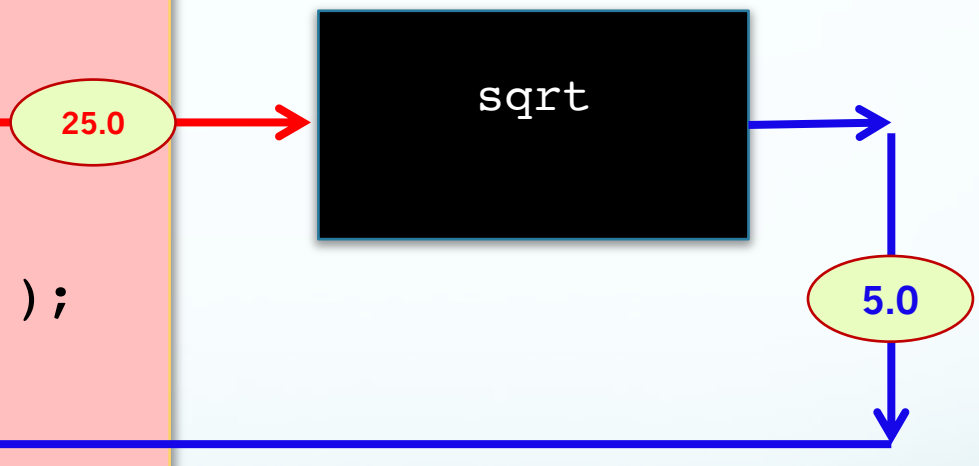
(pre-defined) functions are black boxes

my_prog.c

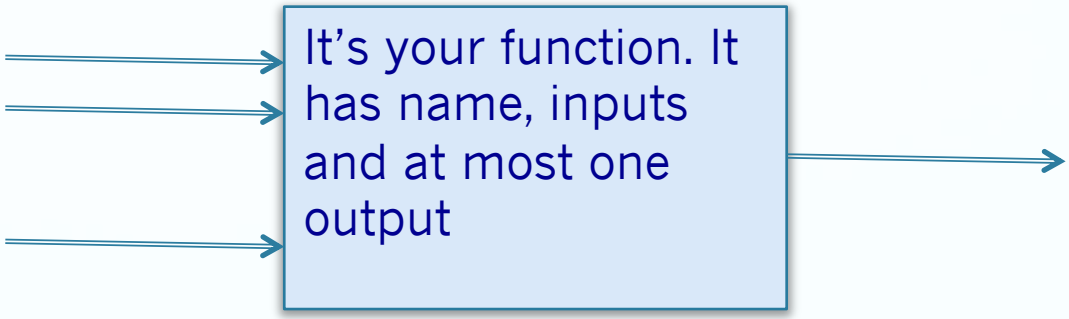
```
#include <stdio.h>
…
#include <math.h>
int main(...) {
   double x=3.0, z;

   z= sqrt( x*x + 4*4 );


   printf("z= %f\n", z);
   return 0;
}
```



we also write our own *user-defined functions*



User-defined functions & program structure

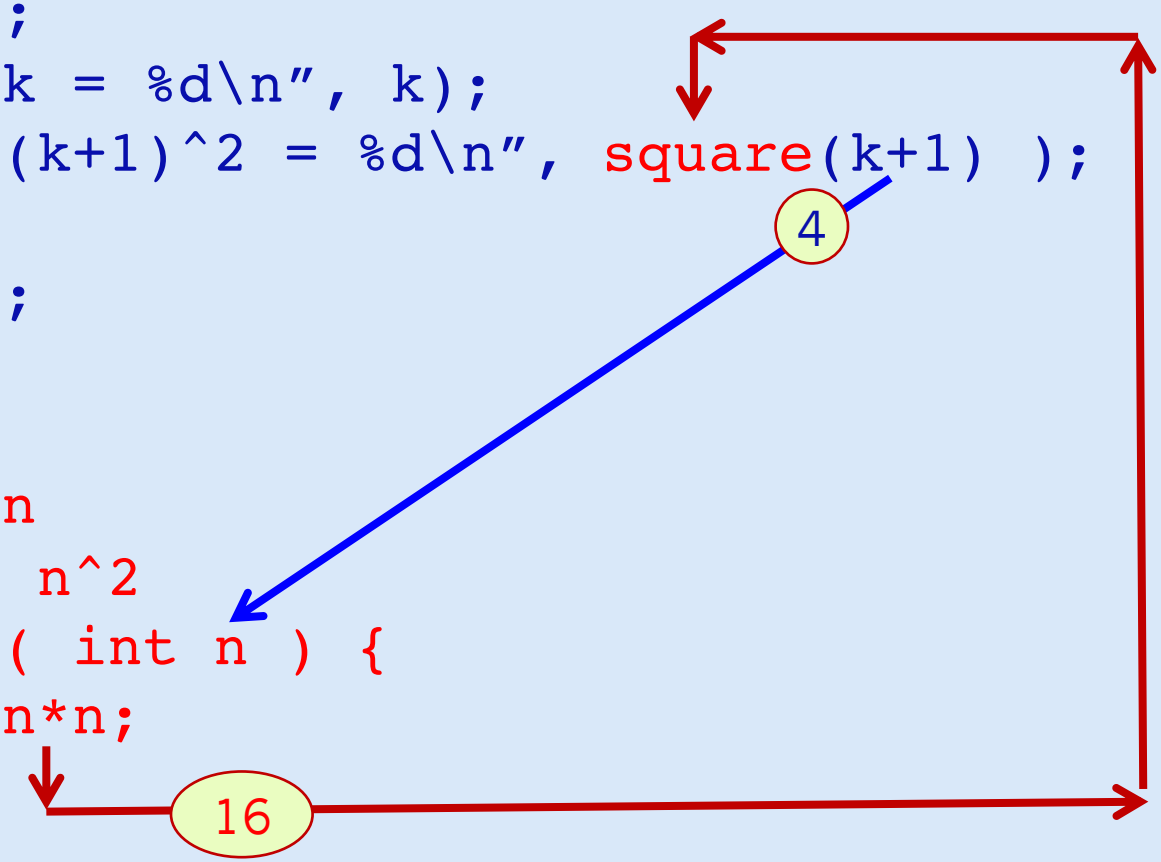
my_prog.c

```
#include <stdio.h>
int square(int);  // function prototype

int main(int argc, char *argv[]) {
   int k= 3;
   printf("k = %d\n", k);
   printf("(k+1)^2 = %d\n", square(k+1) );

   return 0;
}

// input: n
// output: n^2
int square( int n ) {
    return n*n;
}
```



how to write a function

- First, write the function header, therefore determine:

```
int        square      (  int n )
```

data type of function result such as: `int` `float`, `char`... or even `void`	name of the function	list of inputs (formal arguments): • How many inputs needed? • Make each input 1 formal argument of the function. • For each argument, decide its name and data type

- Then, design the algorithm and write the function body: use the *inputs*, compute and *return* the solution.

program structure

	my_program.c

#include & #define	`#include <stdio.h>`
	`#include <stdlib.h>`
	`#define…`

function prototypes	`int factorial( int n ) ;` `...`

main() function –start	`int main(int argc, char *argv[]) {`
	`  int n;`
	`  ...`
	`  printf( "%d! = %d\n", n, factorial(n));`
	`  ...`
	`  return 0;`

-end	`}`

implementation of declared functions	`int factorial( int k ) {`
	`  int i, ans=1;`
	`  for (i=1; i<=k; i++){`
	`  ans *= i;`
	`  }`
	`  return ans;`
	`}`
	`...`

write the *header* for functions

- A: to compute n!

  int factorial(int n)

- B: to compute sum $1^k + 2^k + 3^k + \ldots + n^k$

  int sum(int n, int k)

- C: to compute the volume of a cylinder

  double vol(double radius, double height)

- D: to print out a number of stars (ie the character *)

  void print_stars(int n)

Example: 5.04: write a function

- that returns

n!/(k!(n-

(the number of di
ways of selecting
from **n** distinct ite

- What if: **n!** i
  large

```
#define MAX 12
int prod(int a, b) {
    int c= a*b;
    if (if (a!=0 && c/a != b) {
        exit(…)
    }
    return c;
}

int fact(int n) {
    int ans= 1;
    for (i=1; i<=n; i++) {
      ans = prod(ans, n);
      ??
    }

     //…
    return ans;
}
// returns n!/ ( k! * (n-k)! )
int choices( int n, int k) {
    // biggest int: INT_MAX is about 2x10^9
```

Group Work: Exercises 1, 2, 3 from grok C05

note: write the functions only, without scaffold for testing

5.01 *Write a f* *and returns the*

5.02 *Write* *d returns the*

`5.03` *Writ* **`nt`** *and* **`n ≥`**

Spend a fe

WHAT

HOW

HOW

Zoom: In y

- one of *uss*
- if you h
- you car
- you car

5.01

Group Work: Exercises 1, 2, 3 from grok C05

note: write the functions only, without scaffold for testing

5.01 *Write a f* *and*
returns the
5.02 *Write* *d*
returns the
`5.03` *Writ* **`nt`**
and **`n ≥`**

5.01

Spend a fe
- WHAT
- HOW
- HOW

Zoom: In y
- one of *uss*
- if you h
- you car
- you car

break, then sample quiz

another trial [at home]

Write a function:

```
??? etox( ??? )
```

that returns the sum:

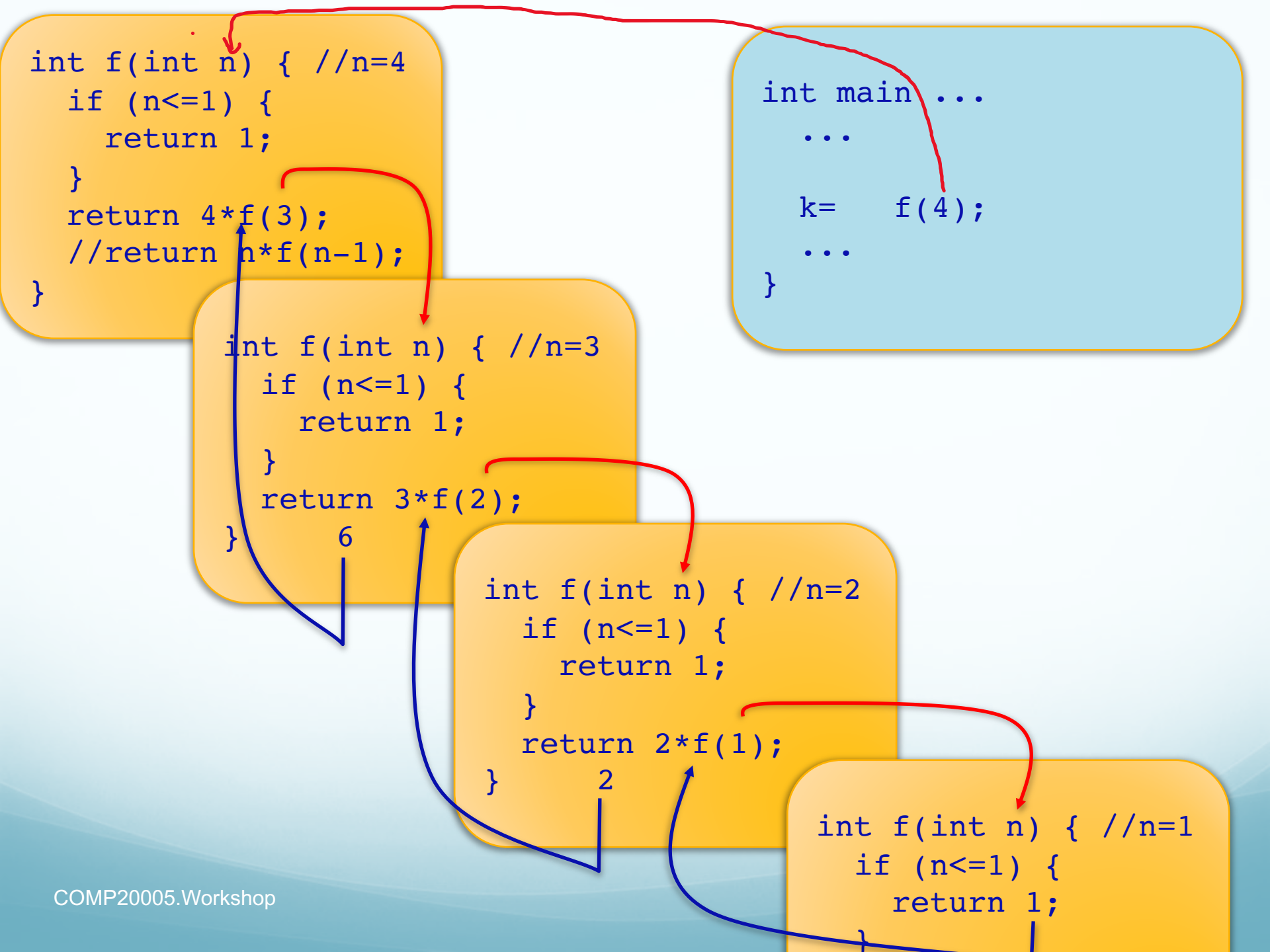
$1 + x/1! + x^2/2! + x^3/3! + \ldots + x^n/n!$

where `x` *is a real number,* `n` *is a positive integer.*

Recursive functions [time permitted]

Recursive function: function that calls itself

```
int factorial( int n ) {
  if (n) {
    return 1;
  }
  return n*factotial(n-1);
}
```


int f(int n) { //n=4
if (n<=1) {
return 1;
}
return 4*f(3);
//return n*f(n-1);
}
int main ...
...
k= f(4);
...
}
int f(int n) { //n=3
if (n<=1) {
return 1;
}
return 3*f(2);
}
6
int f(int n) { //n=2
if (n<=1) {
return 1;
}
return 2*f(1);
}
2
int f(int n) { //n=1
if (n<=1) {
return 1;

Recursive functions: How

- reduce the task of size n to the same tasks of smaller sizes (sometimes, to the same tasks of larger sizes)
- clearly describe the base cases where the solutions are trivial
- when writing code, start with solving the base cases first

Examples:

- factorial (n):
  - base case: when n==0 the solution is 1
  - general case: factorial(n) can be computed from factorial(n-1)

```
int factorial( int n ) {
  if (n==0) { // base case
    ???;
  }
  // general case    [note: else is not needed]
  ???
}
```

Examples

Write recursive function to compute

- $S = 1^2 + 2^2 + 3^2 + \ldots + n^2$

- x^n where x is a real number, $n \geq 0$ is an integer

Lab

- Implement 5.5 and 5.6 from grok C05
- Do not-yet-done exercises in grok C05
- do sample Quiz1 & review for Quiz1

5.05 & 5.06

5.05: *A number is perfect if it equal to the sum of its factors, excluding itself. An example is 6 (=1+2+3).*

- Write function **`int isperfect(int)`** *that returns true if its argument is a perfect number, and false otherwise.*

- Write function **`int nextperfect(int)`** *that returns the next perfect number greater than its argument.*

You need a **`main()`** *function that reads a number. If the number is perfect, print out it; if not, print out the next perfect number.*

WARNING: the first 6 perfect numbers are

6, 28, 496, 8 128, 33 550 336, 8 589 869 056

5.06: *Two numbers are an amicable pair if their factors (excluding themselves) add up to each other. The first such pair is 220, which has the factors [1, 2, 4, 5, 10, 11, 20, 22, 44,55, 110], adding to 284; and 284, which has the factors [1, 2, 4, 71, 142], the sum of which is 220. The next pairs are 1,184 and 1,210; and then 2,620 and 2,924.*

Write a function that takes two **`int`** *arguments and return true if they are an amicable pair. Then, test your function by writing a simple main program that inputs 2 integers and prints out if they are an amicable pair.*

Remember

- Review chapters 1-5 for the MST:
  - program structures, data types, expression, precedence order
  - `scanf` and `printf`
  - `if …,  switch ...`
  - loops: `for`, `while`, `do ... while`
  - functions

Good Luck with Quiz 1

More on Recursive Functions

- ***What:*** A function that calls itself.
- ***When:***
  - a task of size `n` can be reduced to a task (or tasks) of size `< n`, and
  - solution for small, trivial `n` can be easily found.
- Example: f(n) = compute n!, ie. 1*2*3*...*(n-1)*n
  - `f(n) = f(n-1)*n` and `f(0) = 1;`
- ***How*** to write a recursive function?
  - solve all trivial cases (*base cases*) first, then
  - solve the general case by calling the function itself.

```
int fact(int n){


}
```

Recursive functions: Example

- How to write a recursive function?
  - solve all trivial cases (*base cases*) first, then
  - solve the general case by calling the function itself.
- Example: write a recursive function to return the n-th Fibonacci number. The Fibonacci numbers are

```
1, 1, 2, 3, 5, 8, ...
```

where the 0-th and 1-st numbers are 1, and each subsequent number is the sum of its 2 immediate predecessors.

Recursive functions: Example

- How to write a recursive function?
  - solve all trivial cases (*base cases*) first, then
  - solve the general case by calling the function itself.
- Example: write a recursive function to return the n-th Fibonacci number.

  The Fibonacci numbers are

  1, 1, 2, 3, 5, 8, ...

where the 0-th and 1-st numbers are 1, and each subsequent number is the sum of its 2 immediate predecessors.

And so:

General case: f(n)= f(n-1)+f(n-2) when n>1

Base cases: f(0)= f(1) =1 when n=0, 1

Recursive functions: Example

Example: write a recursive function to return the n-th Fibonacci number. The Fibonacci numbers are **1, 1, 2, 3, 5, 8, …** where the 0-th and 1-st numbers are 1, and each subsequent number is the sum of its 2 immediate predecessors.

```
int fib(int n) {




}
```