COMP20005 Workshop Week 5

```
1    Functions. Discuss: Ex. 5.02, 5.03, 5.05
2    Recursive Functions, Ex 5.13

3    sample Quiz
4    implement 5.05, 5.06, 5.14

5    while (having_time()){
       review/ask questions on quiz 1
       implement("a new exercise from grok C05");
     }
```

	my_program.c
#include & #define function prototypes	`#include ...` `int factorial( int ) ;`

main() function –start
 	
-end
	`int main(int argc, char *argv[]) {`
	`  int n;`
	`  ...  // including scanf/set value for n`
	`  printf( "%d! = %d\n", n, factorial(n));`
	`  ...`
	`  return 0;`
	`}`

implementation of
declared functions
	`int factorial( int k ) {`
	`  ...`
	`  return ...;`
	`}`

Use grok for Ex 5.02, 5.03, 5.05

5.02 Write a function `max_4_ints` *that expects four* `int` *arguments, and returns the largest of them.*

`5.03` *Write a function* `int_pow(m,n)` *that returns* m^n *where* `m`, `n` *are* `int` *and* `n` $\geq$ `0`. *For example,* `int_pow(3,4)` *returns* 81.

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 `in` *perfect number, and false*

- Write function `in` *ber greater than its arg*

For each task, spend a few minutes *before writing* to discuss with your peers:

- WHAT are possible approaches?
- WHICH APROACH is simpler for us to write?

Recursive functions [time permits]

Recursive function: function that calls itself

```
int fib( int n ) {
  if (n==1) return 1;
  if (n==2) return 1;
  return fib(n-1) + fib(n-2);
}
```

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

```
int main ...
  ...

  k=   f(3);
  ...
}
```

```
int f(int n) { //n=3
  if (3<=1) return 1;

  return 3*f(2);
}
```

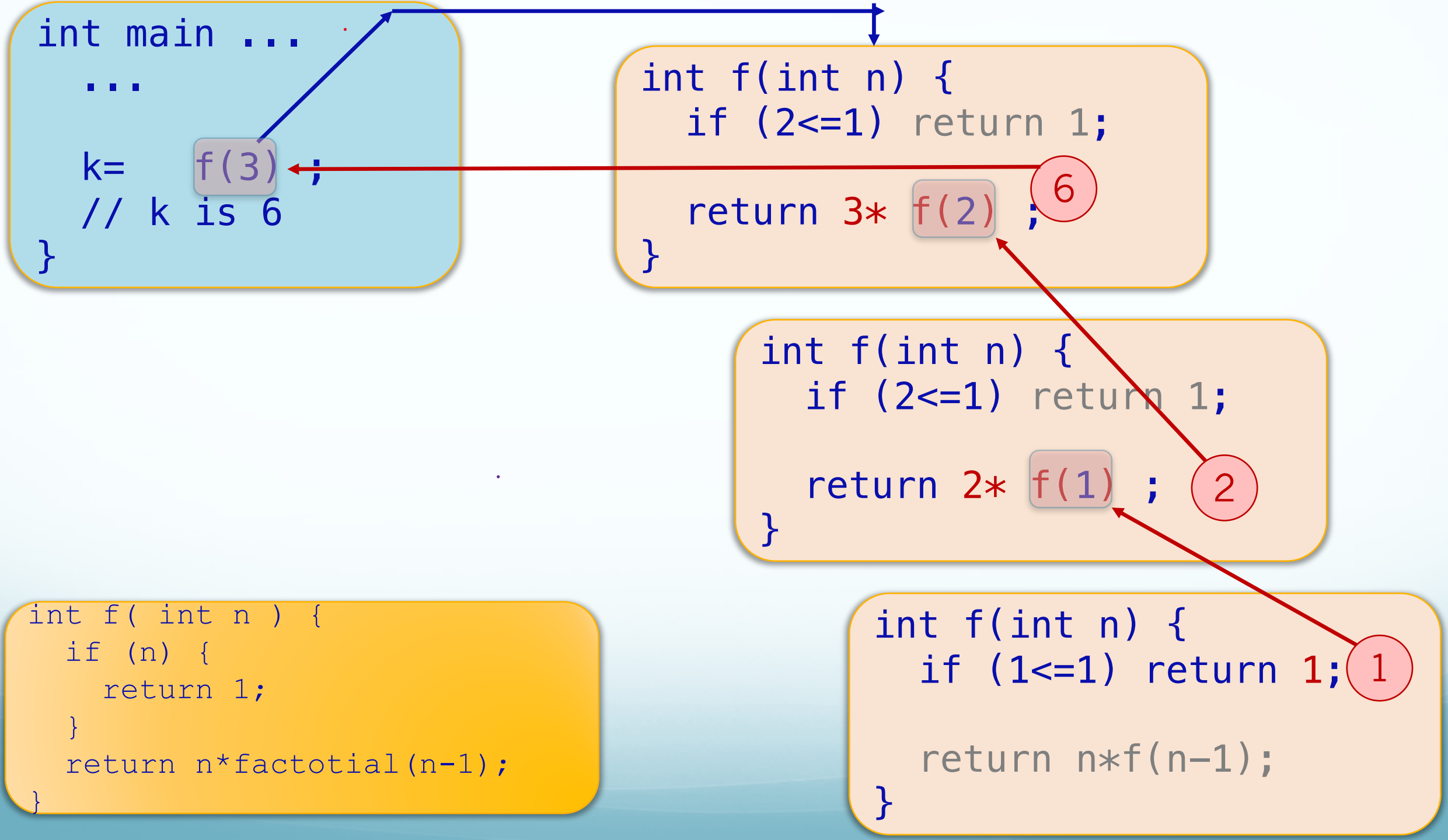
```
int f(int n) { //n=2
  if (2<=1) return 1;

  return 2*f(1);
}
```

```
int f(int n) { //n=1
  if (1<=1) return 1;

  return n*f(n-1);
}
```

```
int f( int n ) {
  if (n<=1) {
    return 1;
  }
  return n*f(n-1);
}
```


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

Review: 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 (base case).

Example: compute f(n) = n!

= 1*2*3*…*(n-1)*n

= (1*2*3*…*(n-1)) * n

- `f(n) = f(n–1)*n` (general case) and
- `f(1) = 1` (base case)

Recursive functions: How

- reduce the task of size n to the same tasks of smaller 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==1 the solution is 1
- general case: factorial(n) can be computed from factorial(n-1)

```
int factorial( int n ) {
  if (n==1) { // base case
    return 1;
  }
  // general case   [note: else is normally not needed]
  return   factorial(n-1)*n;
}
```

Examples: grok 5.13, 5.14

break, then sample quiz

```
if (   ((n1>=n2) && (n1<=n3)) ||(( n1>=n3) && (n1 <=n2)) return n1;


if (   ((n2>=n1) && (n2<=n3)) ||(( n2>=n3) && (n2 <=n1)) return n2;

return n3;


int gcd(int n1, int n1, int n3)
```

In the MST

- you have on average 3 min for each multiple-choice question, so
  - don‘t rush
  - when tracing code, use blank draft paper & pens to avoid mistakes
- You have around 15 minutes for the programming task, remember:
  - read carefully and check with the provided examples to make sure you understand the task correctly
  - spend time to think about possible approaches and stick with the one with is simple (even if it is inefficient)
  - write the code: if the question is to write a function, don’t write the main()
  - check your code for: semicolons, brackets, variable declaration, return, ...

Lab

- Review for Quiz1 & Ask Questions
- Implement `5.05` and `5.06`
- Extras:

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 ...`
- loops: `for ...` , `while ...`
- functions, recursive functions
- others, better not to use: `switch ...`, `do ... while`

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 (base case).

Example: compute f(n) = n! = 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) {



}
```