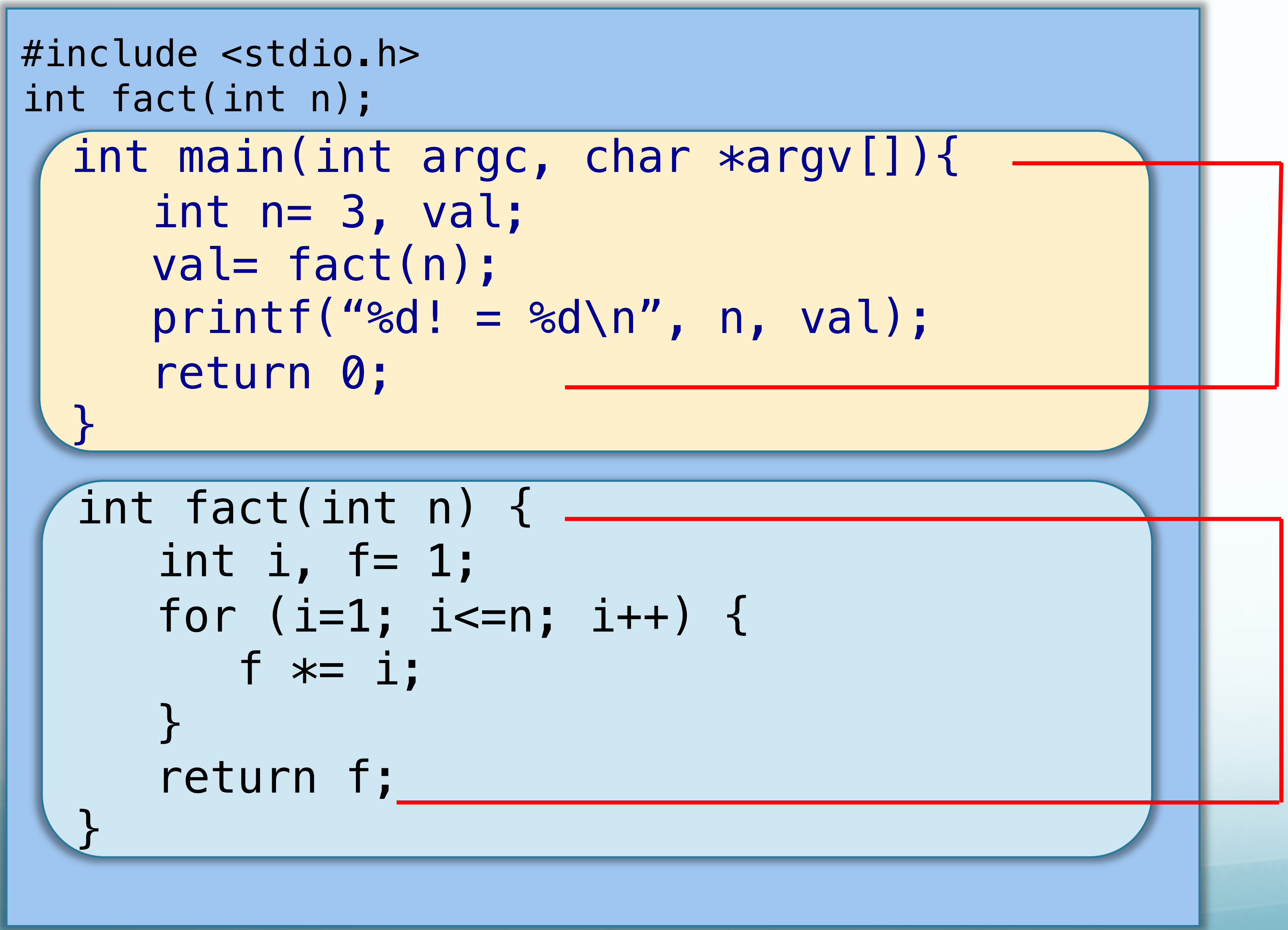
COMP20005 Workshop Week 6

1	Scopes, *exercise 6.2*
2	*Recursive Functions*
3	Pointers, Pointers as Function Arguments, *discuss: exercise 6.5, discuss exercise 6.9*
L A B	**6.5, 6.6**

Scopes: local variables



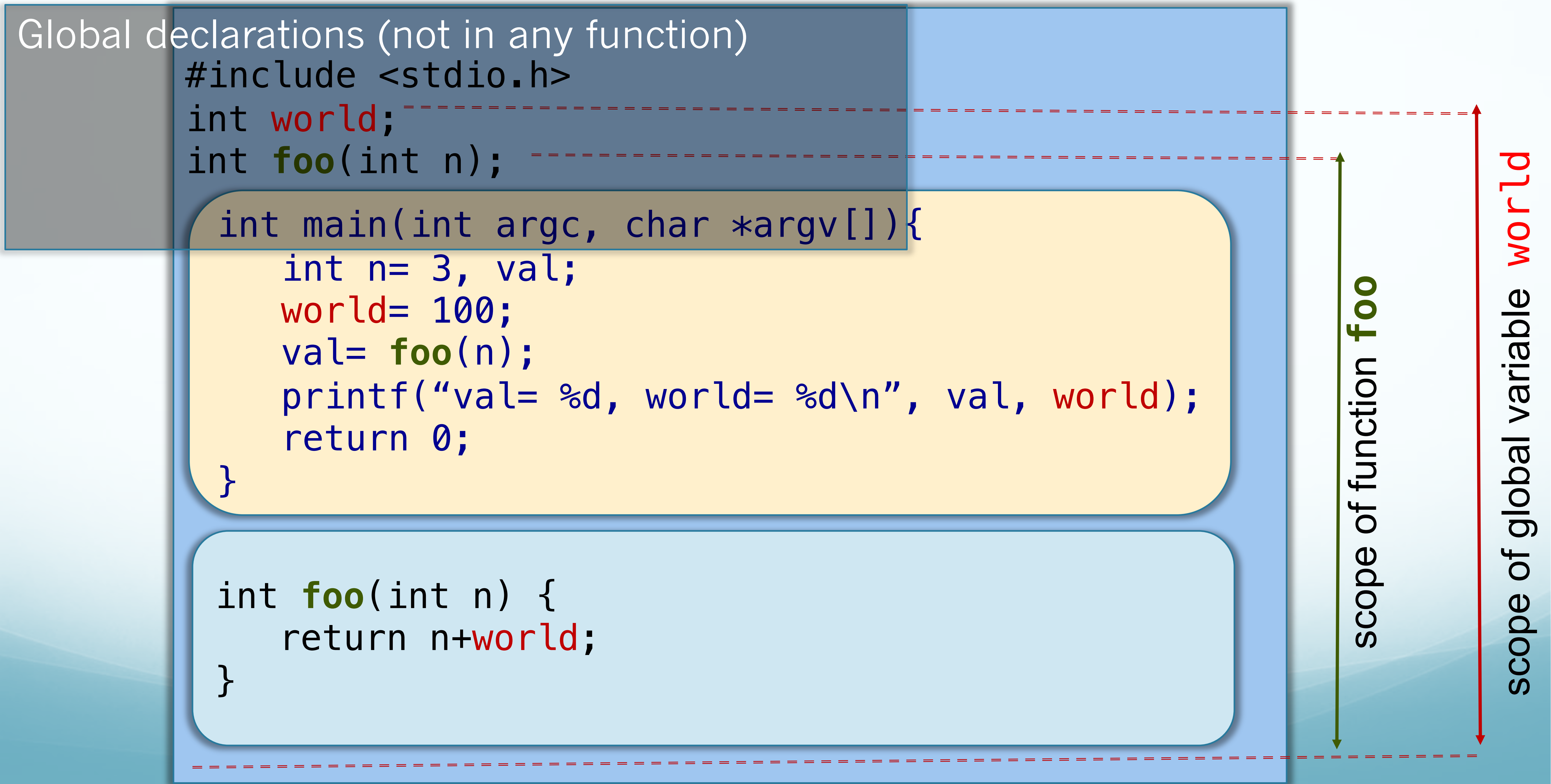
```
#include <stdio.h>
int fact(int n);
  int main(int argc, char *argv[]){
     int n= 3, val;
     val= fact(n);
     printf(“%d! = %d\n”, n, val);
     return 0;
  }

  int fact(int n) {
     int i, f= 1;
     for (i=1; i<=n; i++) {
        f *= i;
     }
     return f;
  }
```

`argc`, `argv`, `n`, and `val` available here

`n`, `i`, and `f` available here

All function variables and parameters are local and private.

Scopes: global variables (and functions)



Global declarations (not in any function)

```
#include <stdio.h>
int world;
int foo(int n);

int main(int argc, char *argv[]){
   int n= 3, val;
   world= 100;
   val= foo(n);
   printf("val= %d, world= %d\n", val, world);
   return 0;
}

int foo(int n) {
   return n+world;
}
```

scope of function foo

scope of global variable world

***6.02**:*

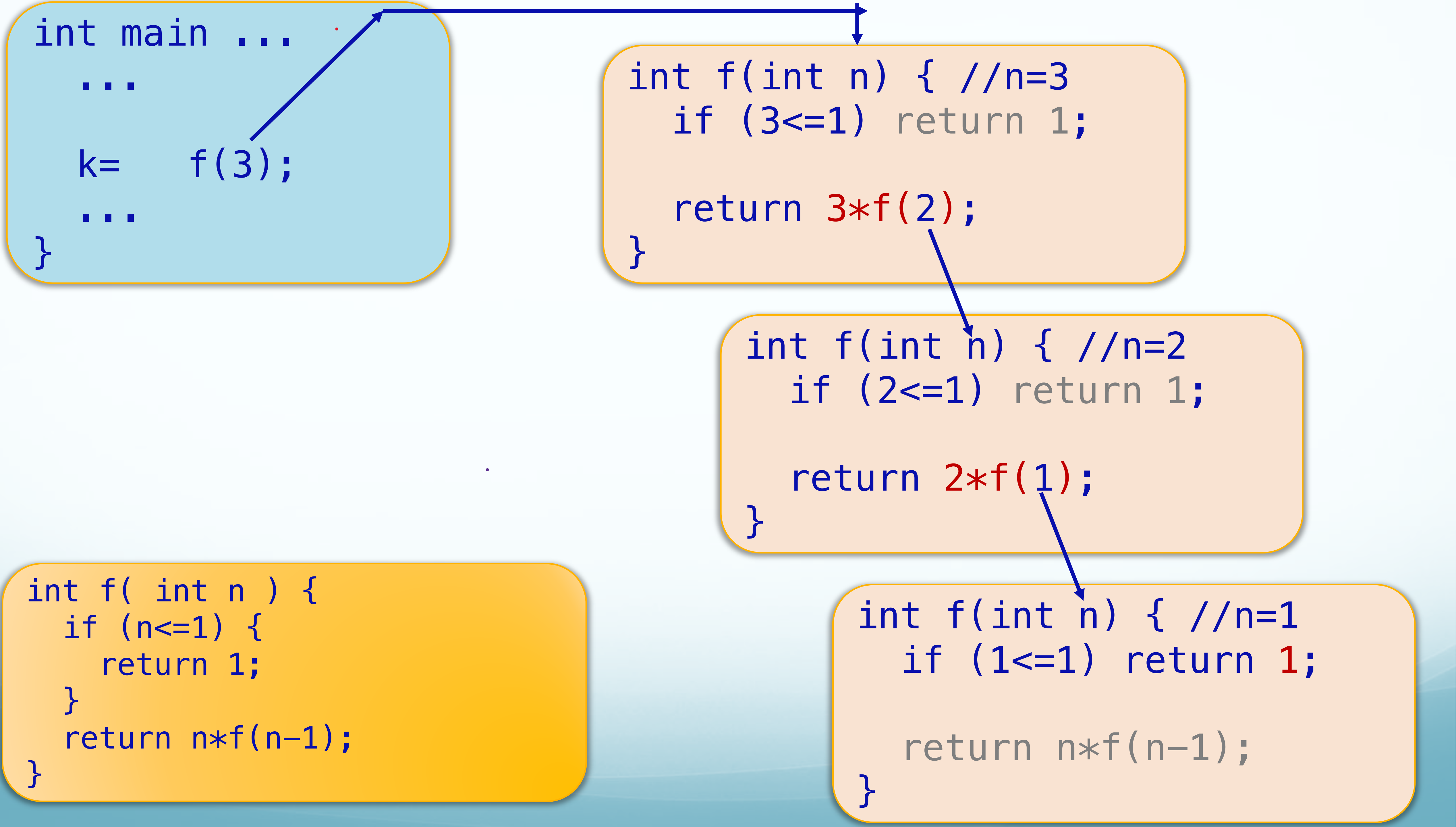
For each of the 3 marked points, write down a list of all of the program-declared variables and functions that are in scope at that point, and for each identifier, its type.

```c
int bill(int jack, int jane);
double jane(double dick, int fred, double dave);

int trev;

int main(int argc, char *argv[]) {
     double beth;
     int pete, bill;      /* -- point #1 -- */
     return 0;
}

int bill (int jack, int jane) {
     int mary;
     double zack;         /* -- point #2 -- */
     return 0;
}

double jane(double dick, int fred, double dave) {
     double trev;         /* -- point #3 -- */
     return 0.0;
}
```

Recursive functions [time permits]

Recursive function: function that calls itself

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



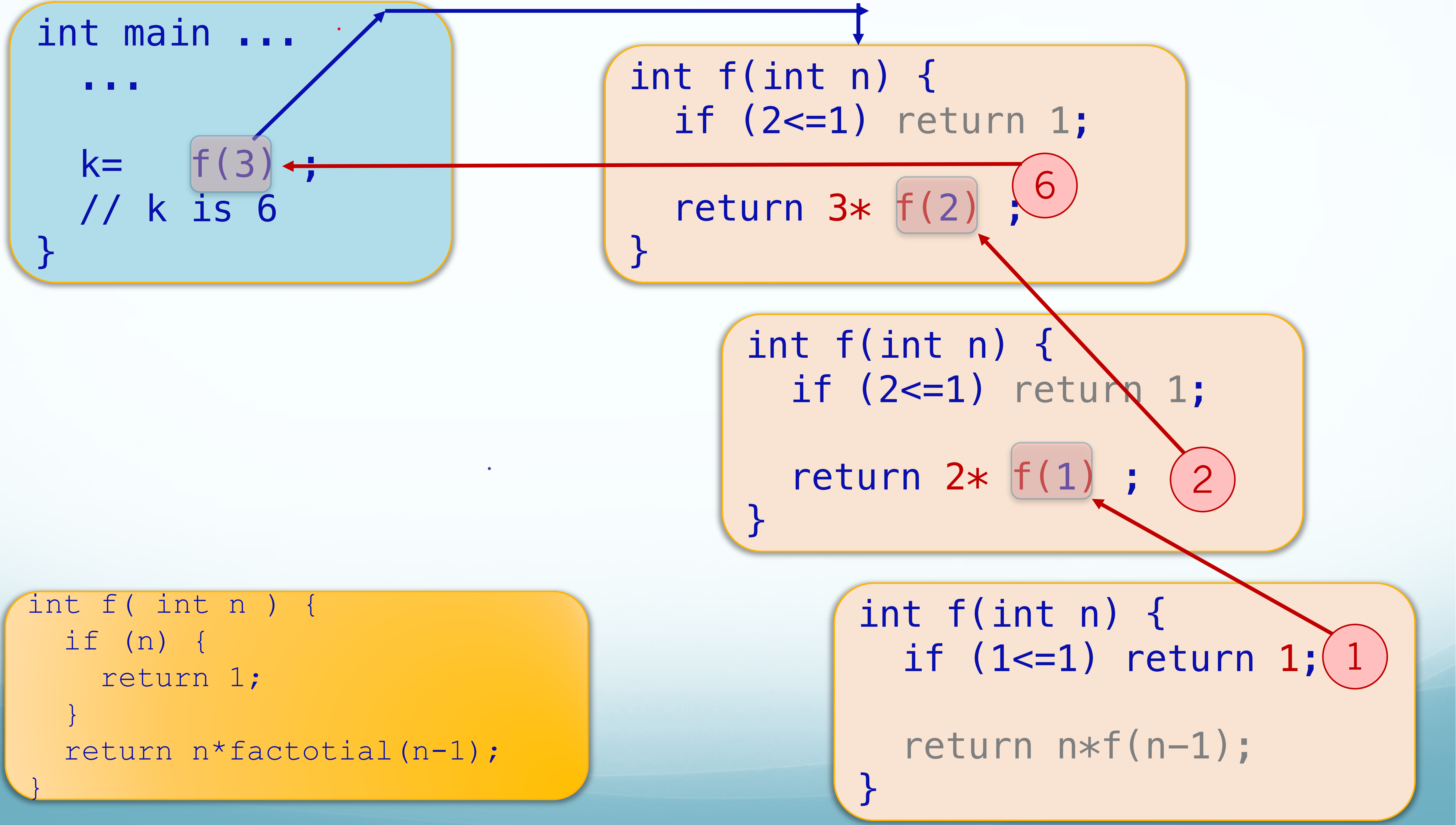
```
int main ...
  ...

  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) ;
}
```

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

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

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

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

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

= (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

Pointers: check your understanding

```
int a= 18
int *pa;
```

understood as:

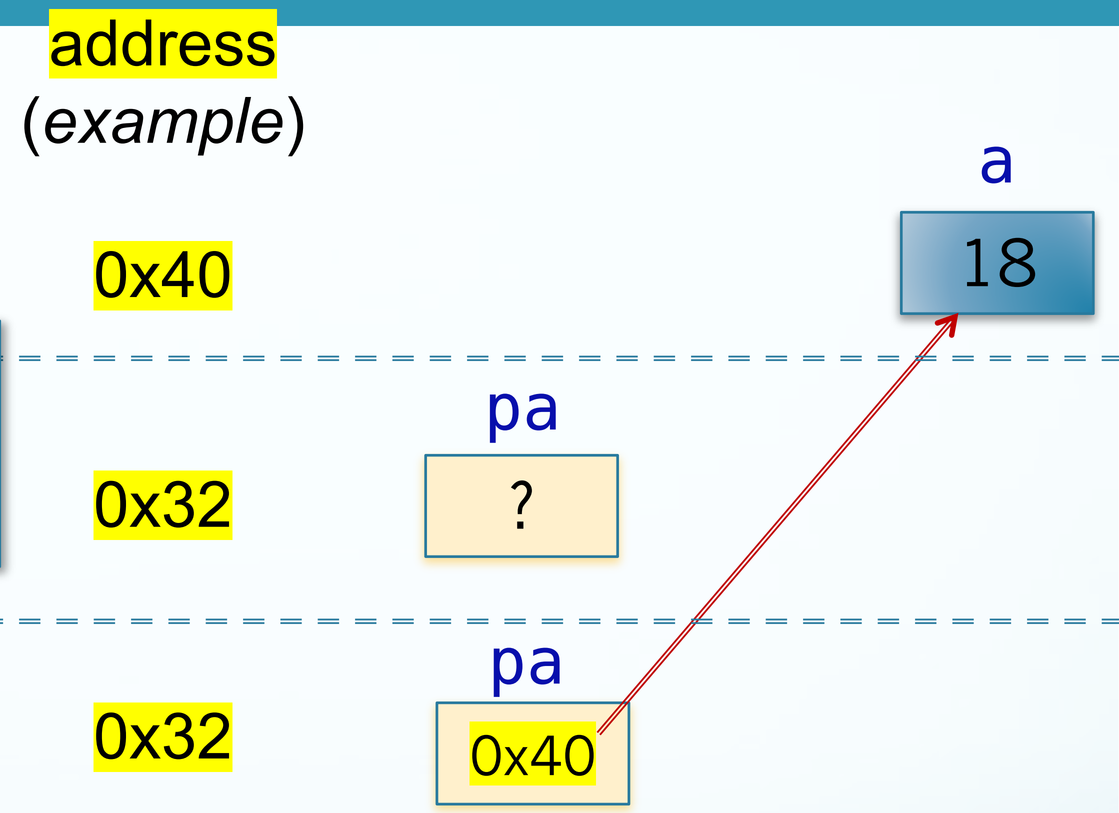
```
    int*     pa;
```

where pa, not *pa, is declared!

```
pa= &a;
```

```
// What is the value of a  and  pa  after:
*pa = (*pa) + 1;
```

here *pa, not pa, is used!
here * is a unary operator.



pointer/address supplies an alternative method for accessing memory

unary operators & and * : referencing and dereferencing

```
int n= 10;
int *pn;

pn= &n;
```

Check your understanding:

a) The datatype of pn is ______

b) If n is at the address 0x4444, then pn has the value of ______

c) The value of *pn is ______

d) After

```
*pn= 100;
```

the value of pn is __________, of n is ___________

e) What is the effect of:

```
*(&n) = 1;
```

a function problem: working as intended? why?

A Limitation of (naive) functions:

- can have only ONE output at most

Q: How to make sAndP change the value of **product**?

```
int main(...) {
  int a=2, b=4, sum= 0, product= 0;
       ...
  sum= sAndP(a, b, product);
  printf("sum=%d",        sum);
  printf("prod=%d”,  product);
     ...
}

// returns m+n and also computes prod= m*n
int sAndP(int m, int n, int prod ) {

  prod = m * n ;
  return m+n;

}
```

pointers as function parameters: an example

Using pointers as parameters, function can have multiple output!

Example: Line 4 leads to the change of **product**.

→ function sAndP effectively has 2 outputs!

```
1    int main(...) {
2       int a=2, b=4, sum, product;
3          ...
4       sum= sAndP(a, b, &product);
5
6       printf("sum=%d",             sum);
7       printf("prod=%d”,            product);
8         ...
9    }

11   void sAndP(int m, int n, int * pp ) {

12      *pp = m * n ;
13      return m+n;

14   }
```

Sending address **&var** to a function allows it to change **var**!

In this function, just use the integer *pp and not pp

another version:

Line 4 leads to the change of **sum** and **product**.

```
1    int main(...) {
2       int a=2, b=4, sum, product;
3           ...
4       sAndP(a, b, &sum, &product);
5
6       printf("sum=%d",                      sum);
7       printf("prod=%d”,                     product);
8          ...
9    }

11   void sAndP(int m, int n, int *ps, int *pp ) {

12      *ps = m + n ;
13
        *pp = m * n ;
14
     }
```

Here, *ps and *pp represent two outputs.

Pointers – application in function parameters

1	`int n=10;`	
2	`printf(“%d”, n);`	What sent to `printf` ? Can `printf` change the value of `n`?
3	`scanf(“%d”, &n);`	What sent to `scanf`? What `scanf` do to `&n`, to `n`?
4	`swap(&n, &m);`	What passed to `swap`? Can this call make change to `&n` or `&m`? Can this call make change to `m` or `n`?

5	`void int_swap(int *a, int *b){`
	`    ???`
	`}`	

Discuss, then do Exercise 6.05, 6.09

In the context of this `main()`, write a function `sort2`
so that it re-orders the values of `v1` and v2 in the increasing order.
For example:

- If `v1`, `v2` are `10`, `5` before calling `sort2`, they become `5`, `10` after calling.
- If `v1`, `v2` are `1`, `2` before calling `sort2`, they remain `1`, 2 after calling.

```
int main(int argc, char *argv[]) {
    ...

    printf("Before: v1 = %d, v2 = %d\n", v1, v2);
    sort2(???);
    printf("After:  v1 = %d, v2 = %d\n", v1, v2);

    return 0;
}

void sort2( ???  ) {


}
```

Lab

- Implement 6.9 (and Re-implement 6.5 if still in doubt)
- implement or write-solution-in-a-whiteboard for not-yet-done Exercises in grok C05
- *Do the exercise with* triangle.c *as described in LMS Week 6 Workshop Content and other exercises from C06*

Assignment 1 released Wed, discussed in Fri lecture!

- Do as much as you can by Week 7 Workshop
- Try to submit a few times
- Regularly use Discussion Forum
- Q&A in Week 7 & 8 Workshops

Additional Slides

Quiz 1

In executing the program:

```
int a=100, b=200;
void f(int a) {
   a++;
   print(“1: a= %d b= %d\n”, a, b) ;
}
int main(int argc, char *argv[]) {
  int a=5, b= 10;
  f(a);
  print(“2: a= %d b= %d\n”, a, b) ;
  return 0;
}
```

what will be printed out?:

A	1: a= 6 b= 200 2: a= 5 b= 10	B	1: a= 6 b= 200 2: a= 6 b= 10
C	1: a= 6 b= 10 2: a= 5 b= 10	D	1: a= 6 b= 10 2: a= 6 b= 10

Quiz 2

After executing the fragment:

```
int x= 10;
f(&x);
printf(“x= %d\n”, x);
```

the output is:

```
x= 0
```

which function has been used in the call `f(&x)` *?*

A:

```
int f(int n) {
    return 0;
}
```

B:

```
void f( int *n) {
   &n= 0;
}
```

C:

```
void f (int *n) {
    n= 0;
}
```

D:

```
void f( int *n) {
   *n= 0;
}
```

Quiz 3

Given function:

```
void f(int a, int *b) {
   a= 1;
   *b = 2;
}
```

Assuming the following fragment is in a valid main(). What will be printed out?

```
int m= 5;
int n= 10;
f(m, &n);
printf(“m= %d, n= %d\n”, m, n);
```

A) m= 5, n= 10	B) m= 1, n= 2
C) m= 5, n= 2	D) m=1, n= 10

A Rule: Never use global variables

```
#include <stdio.h>
~~int world;~~
int foo(int n);
int main(int argc, char *argv[]){
    int n= 3, val;
    int world= 100;
    val= foo(n, world);
    printf(“val= %d, world= %d\n”, val, world);
    return 0;
}

int foo(int n, int world) {
    return n + world;
}
```