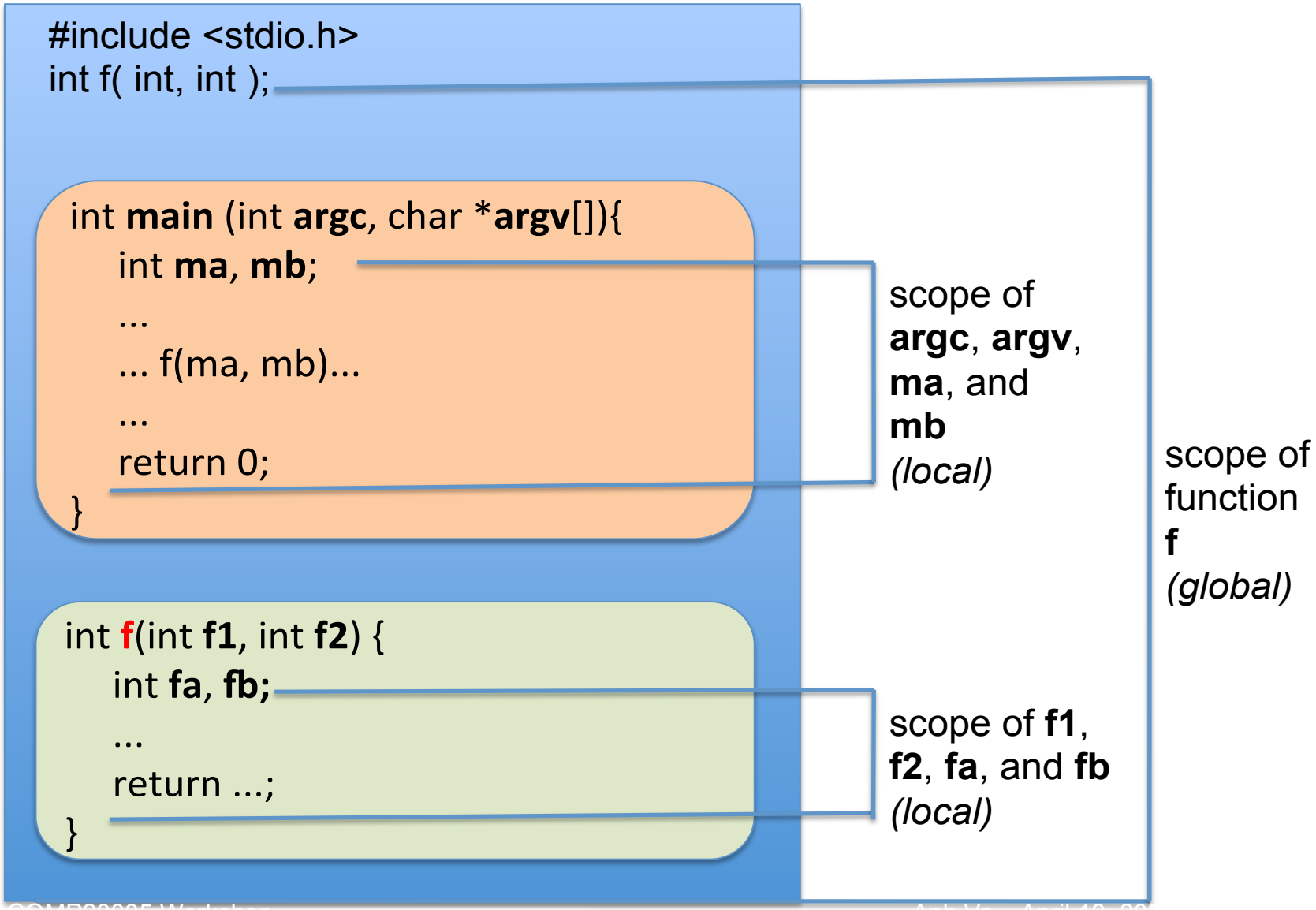
COMP20005 Workshop Week 6

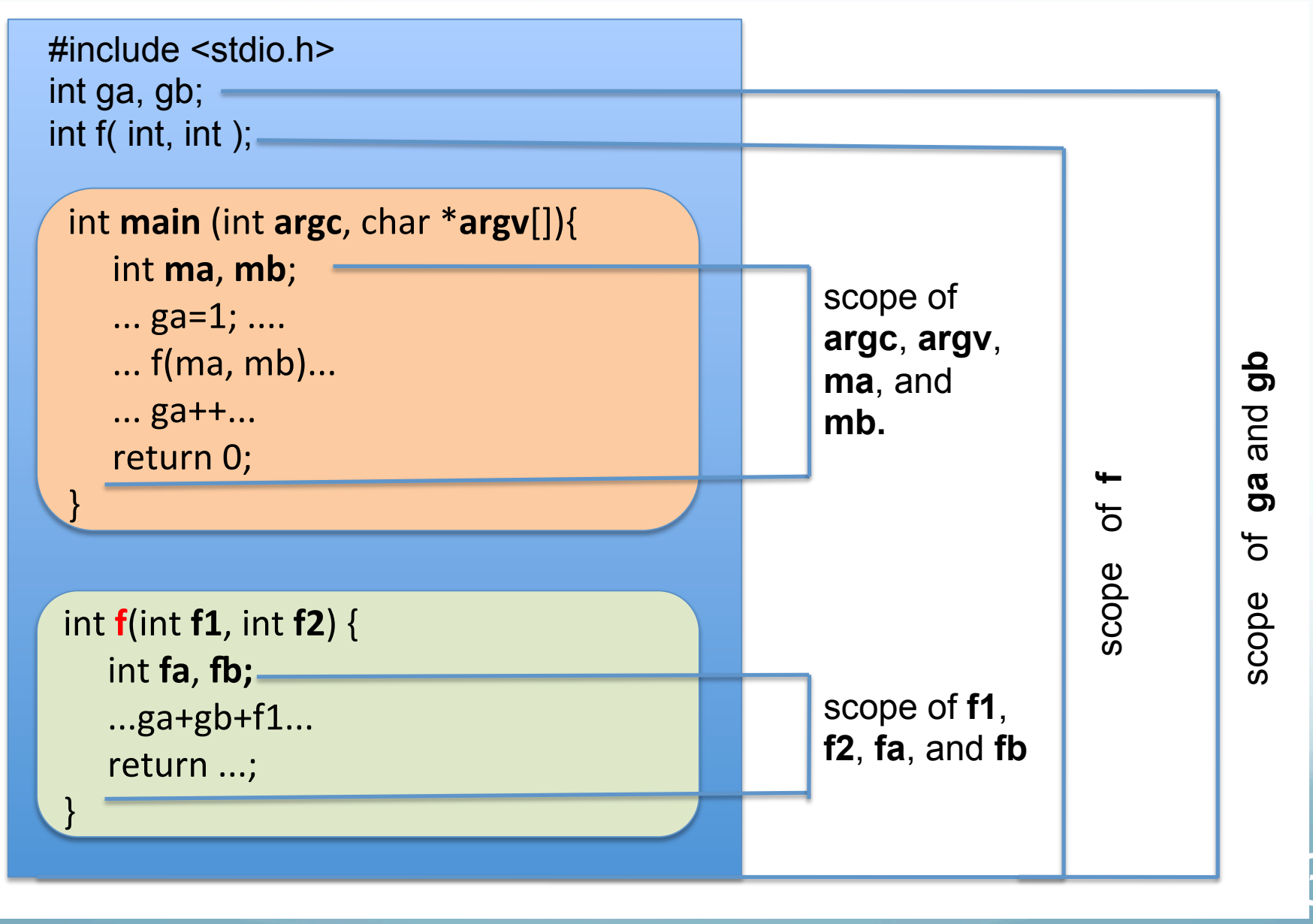
Scopes & Pointers

1. Scopes
2. Discuss 6.2, 6.3
3. Pointers, Pointers as Function Arguments
4. *Recursive Functions: When & How*
5. Design 6.9
6. Time for Fun: Quiz
7. Implement 6.9
8. Implement other tasks from LMS

Scopes, local & global variables



Scopes, local & global variables



Rule: No global variables

```
#include <stdio.h>
int ga, gb;
int f( int, int );

int main (int argc, char *argv[]){
    int ma, mb;
    ... ga=1; ....
    ... f(ma, mb)...
    ... ga++...
    return 0;
}

int f(int f1, int f2) {
    int fa, fb;
    ...ga+gb+f1...
    return ...;
}
```

scope of **argc**, **argv**, **ma**, and **mb.**

scope of **f1**, **f2**, **fa**, and **fb**

Group work: 6.2 & 6.3 (see grok), time limit: 8 min

***6.2**: 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. Don't forget* **main**, **argc**, **argv**. *Where there are more than one choice of a given name, be sure to indicate which one you are referring to.*

```
1  int bill(int jack, int jane);
2  double jane(double dick, int fred, double dave);
3
4  int trev;
5
6  int main(int argc, char *argv[]) {
7       double beth;
8       int pete, bill;      /* -- point #1 -- */
9       return 0;
10 }

11 int bill (int jack, int jane) {
12      int mary;
13      double zack;         /* -- point #2 -- */
14      return 0;
15 }

16 double jane(double dick, int fred, double dave) {
17      double trev;         /* -- point #3 -- */
18      return 0.0;
19 }
```

6.3

What's wrong with these functions?

Should we employ variable/ function names in this way?

```
1   int dumb(int dumb) {
2         return dumb+1;
3   }

11  int dumber (int dumber) {
12        int dumber= 6;
13        return dumb(dumber+1);
14  }
```

Never use a same identifier for more than one object/purpose!

Variable and address

```
1  int n;
2
3  scanf("%d", &n);
4  printf("%d", n);
```

What's the data type of `&n` ?
What were sent to `scanf`, to `printf` ?

Pointers

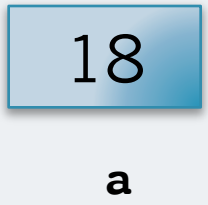
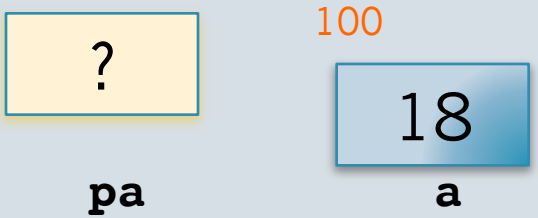
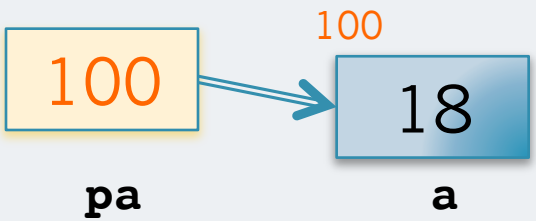
	Compare:	

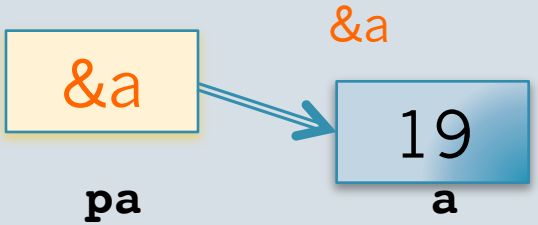
1
2
3
4
5
	`int n;`
	`int *p;`
	`p= &n;`
	`scanf("%d", p);`
	`printf("%d", *p);`
		`int n;`
		
		`scanf("%d", &n);`
		`printf("%d", n);`

	There are 2 possible interpretations for line 2 of the LHS:	
a)	`int *p;`	`*p` is a variable of type **`int`**, so: `*p= n;` ✔ `*p= &n;` ✖
b)	`int* p;`	**`p`** is a variable of type **`int*`** (or, type "**pointer to `int`**", "**address of `int`**"), so: `p= n;` ✖ `p= &n;` ✔

(&a)** *is equivalent to* **a

Pointers = references = addresses

`int a= 18;`		**a** is a location in the memory, interpreted as an **int,** with value of 18
`int *pa;`		**pa** is an **int pointer**, it can hold the address of an **int**
`pa= &a;`		**pa** now holds the address of **a**, or, it "points" to **a**. ***pa** is another method to access **a**

`*pa= *pa + 1;`		These statements are equivalent:
		`*pa = *pa + 1;`
		`a = a + 1;`
		`*pa = a + 1;`
		`a = *pa + 1;`

Pointers

```
1  int n;
2
3  scanf("%d", &n);
4  printf("%d", n);
```

What's the data type of `&n` ?

What were sent to `scanf`, to `printf` ?
Why is the difference ?

```
   void int_swap(int *a, int *b)
```

What's the input and output of this function?

pointers as function parameters

Pointers can be used to change the value of variables *indirectly*

Example: Function call in line 4 leads to the change of value 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 }
```

&sum and **&product** are kept unchanged, the changes happen to **sum** and **product**

Recursive functions?

- 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 found.
- Examples:
  - **`f(n) = n*f(n-1)`** and **`f(0) = 1;`**
  - **`f(n) = f(n-1)+f(n-2)`** and **`f(0) = f(1) = 1;`**
- How to write a recursive function?
  - solve all trivial cases (*base cases*) first, then
  - solve the general case by calling the function itself.

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?

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) {




}
```

Questions for the coming online test?

Lab: working in group with:
Doing exercises in grok W5, W5X (recursive), W6, W6X
Doing Quiz 1 - Quiz 4 in `github.com/anhvir/c205`

With the fragment:

```
int x= 10;
f(&x);
```

which function below will set `x` *to zero?*

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 2

Given function:

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

what are values of `m` *and* `n` *after the following fragment:*

```
m= 5;
n= 10;
f(m, &n);
```

A) 5 and 10	B) 1 and 2	C) 5 and 2	D) 1 and 10

Quiz 3

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 4

In executing the program:

```
#define N 3
int f(int);

int main(int argc, char *argv[]) {
  printf("%d \n", f(N));
  return 0;
}

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

what will be printed out?:

A 14	**B** 1
C 6	**D** 9

Ex 6.9 Part 1: Ex 3.6 revisited

Target: Read an integer amount of cents between 0 and 999. Print out the coin changes, using coins 200, 100, 50, 20, 10, 5 cents. For example, for cents = 293, the printout should be:

50, 20, 20, 5

Write & use functions:

```
try_one_coin

print_change(int cents)
```

Lab

- Design and implement a solution to Exercise 6.9, using an argument that is a pointer.
- Take the **`triangle.c`** program and add in some statements that use **`%p`** format to print out the address of the argument variable to each recursive call in **`t_rec()`**. Can you work out how big each stack frame is? Now add some more variable declarations including a static variable and print out their addresses too. Can you build a picture of whereabouts in memory things are getting located.
- Recursive Functions: Implement, or just write functions for: Ex 5.13, Ex 5.14
- Still got time on your hands? Read the first few pages of Chapter 7. Then further modify your solution to Exercise 6.9 so that you use an array of possible coin values, int coins[NUMCOINS]={200,100,50,20,10,5,2,1} and a loop to try out the coins one by one. Then look back at the solution to Exercise 5.6 that you did a couple of weeks ago, and go "wow".