COMP10002 Workshop Week 4: Functions & Pointers

1	Discussion1: Functions & Recursive Functions • Ex 6.2 • Design for Exercise 5.6
2	Discussion2: Pointers & Functions : What/Why/How • Pointers & Function Arguments • Design for Exercise 6.9
LAB	• Implement 5.6 and 6.9 • Review for Quiz 1
Note	Quiz 1: Thursday 26 August, 30 minutes, starting at 2:15pm
Requirements on LMS	• Discuss Exercise 6.2; then • Design and implement solutions to Exercises 5.6 and 6.9; then • Any other exercises from Chapters 5 and 6 that take your eye; and • In the lead-up to the Quiz, be sure to ask your tutors to explain any issues or topics from Chapters 1 to 6 that you haven't fully understood.

The Quiz Next Week (2:15PM Thursday 26 AUG)

Quiz 1. Some information in connection with Quiz 1:

It will be held on **Thursday 26 August**, starting at 2:15pm and finishing at 3:00pm, via the LMS Quizzes facility. You will be allowed 30 minutes within that 45 minute span to complete the questions, so you need to start before 2:30pm if you want the full 30 minutes.

- The Quiz consists of five multiple-choice questions and one "write a function" question.
- The Quiz covers Chapters 1 to 6 of the textbook, and lecture videos from lec01-a to lec04-l.

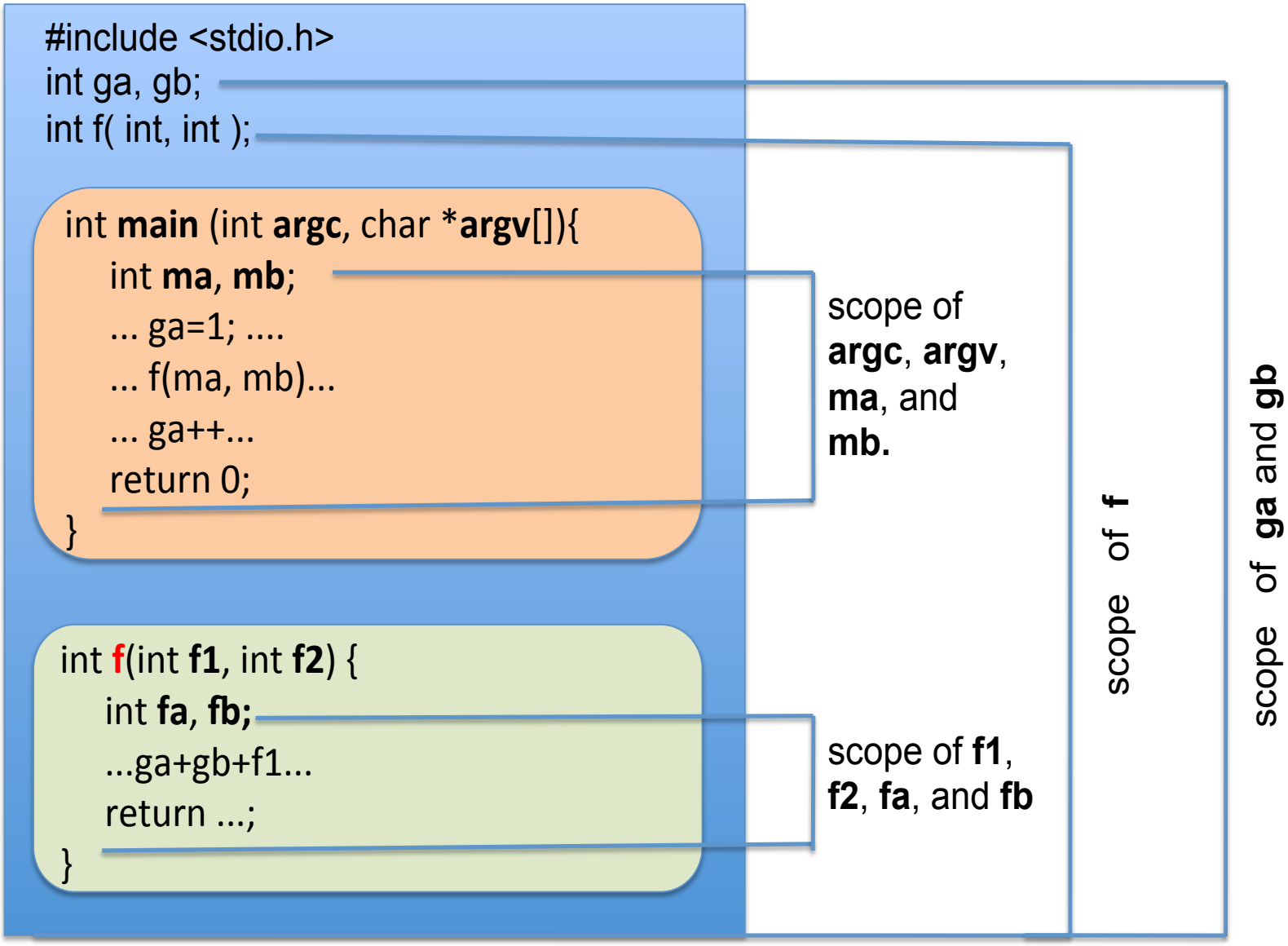
Look for "Quiz 1 Practice" at the end of Week 4 for a sample quiz that you can take up to five times (you'll see mostly different questions each time) to allow you to calibrate your expectations.

1. Getting Started: Do-It-Together on Functions

Use grok & grok's Playground for

- computing n!
- **W4X:** Exercise 5.4: Combinations
- Recursive Functions
- **Discussion on W04:** Exercise 5.6: Finding amicable pairs

Re-cap: function prototype, implementation, calls, local variables



Exercise 6.2

***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;   jane      /* -- 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 }
```

2a. Pointers

What are passed to functions?

Compare:

```
scanf("%d", &n);
```

and

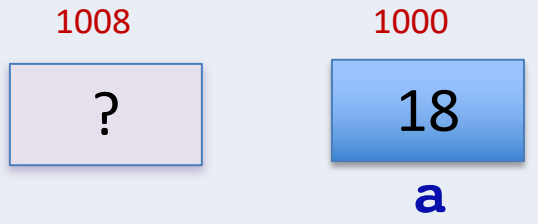
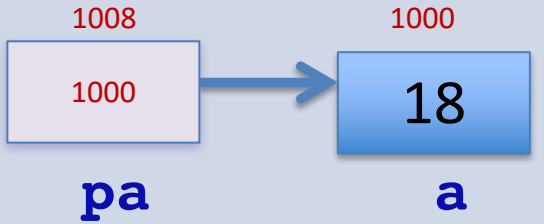
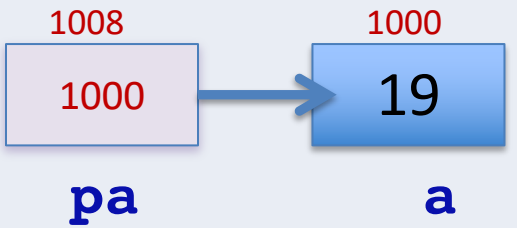
```
printf("%d", n);
```

What's the difference in the values passed to functions? Why?

Data type: `int *` ⇔ "pointer to int" ⇔ "address of int"

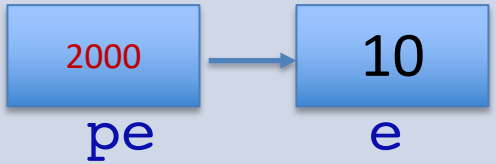
`int a= 18;`		**a** is a **int** cell in the memory, with value of 18. Suppose that the cell has address 1000. The address of a specifies the location of a in the computer's memory.
`int * pa;`		**pa** is an **int pointer**, it can hold the address of an **int** We supposed that pa has address 1008. int * === address of int pa= 1000; int int * double double * int * === pointer to int

Data type: `int *` ⇔ "pointer to int" ⇔ "address of int"

`int *pa;`		
`pa= &a;` `*pa`		**pa** now holds the address of **a**, or, it "points" to **a**. using the value of **pa** we can have *indirect access* to **a**
`(*pa)++;`		***pa*** is another name for **a** **`(*pa)++;`** is equivalent to: **`a++;`**

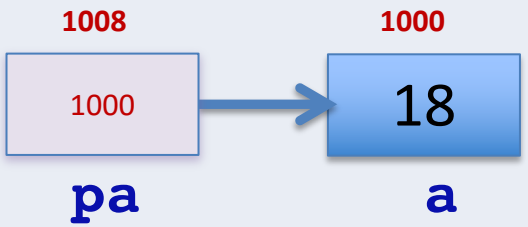
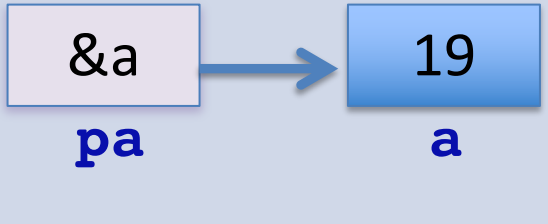
If `e` is a variable, the `&e` is the address of `e`, `&e` is a reference to `e`.
dereference of `&e` is written as `*(&e)` and give us back `e`, ie `*(&e)` is the same as `e`
If `p` is a pointer variable, `p` can hold and address, `p= &e` is valid
And `*p= *(&e)` is the same as `e`

`int e;` 	the envelop 
`e= 10;`	replace content of the envelop with $10

`int *pe;`
`pe = &e;`
 
	location= on top of the table

`*pe += 100;` 	add $100 to the thing that is at the above location

Data type: `int *` ⇔ "pointer to int" ⇔ "address of int"

`pa= &a;`		`Note: we don't need to know the address values 1000, 1008. Right?`
`(*pa)++;`		***pa*** is another name for **a** **`(*pa)++;`** is equivalent to: **`a++;`**

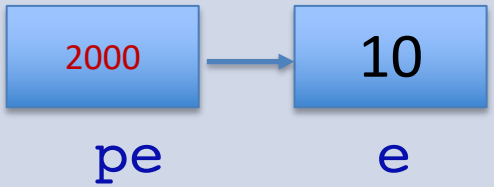
So what?

Suppose that function `f1` calls function `f2` with the call `b= f2(a);` then

- The value of a is passed to `f2`, there is no way function `f2` can make change to variable `a`

But if the call is `b= f2(&a);` function `f2` can change `a` to `10` with the command `*a=10;`

`int e;` 	the envelop 
`e= 10;`	replace content of the envelop with $10

`int *pe;`
`pe = &e;`
 
	location= on top of the table

`*pe += 100;` 	add $100 to the thing that is at the above location
`Sam(e);` `Bob(&e);`	tell Sam how much $ in the envelop ☺ tell Bob where the envelop is

pointers as function parameters

Function call in line 4 leads to the change of value of **sum** and **product**.

In this way, function sAndP can access local variables **sum** and **product** of main().

```
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 ) {
     m is 2, m is 4,  ps= &sum→ *ps is sum
12   *ps = m + n ;  // equivalent to sum= m + n;
13   *pp = m * n ;
14 }
```

ps= &sum, and so *ps is the same as sum

Quiz 1

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 printed after the following fragment:

```
m= 5;
n= 10;
f(m, &n); // m won't be changed
printf("%d and %d\n", m, n);
```

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

2b. A Discussion based on Exercise 6.9

The Task: Suppose that you're working at a shop's counter and need to return some money which is less than $10 to a customer, using only coins.
Write a program that reads an integer amount of cents between 0 and 999 then prints out the coin changes, using coins from $2 to 5c.
Note: valid coins are 200c, 100c, 50c, 20c, 10c, and 5c

Example of running the program:

```
./program
Enter amount in cents: 122
give 1 100c coins
give 1  20c coins
./program
Enter amount in cents: 533
give 2 200c coins
give 1 100c coins
give 1  20c coins
give 1  10c coins
give 1   5c coins
```

Discussion: Exercise 6.9

The Task:
Write a program that reads an integer amount of cents between 0 and 999 then prints out the coin changes, using coins from $2 to 5c.

Example:

```
./program
Enter amount in
cents: 122
give 1 100c coins
give 1  20c coins
```

Design, using functions when reasonable

```
- scanf value for cents
- exit if input invalid (not exist, <0, >999
- print_change(cents)
- end

??? print_change(???) {
```

Lab: Ex 6.9

Target: … For example, for cents = 393, the printout should be:
200c, 100c, 50c, 20c, 20c, 5c
(for simplicity, we print "200c" instead of "$2", but you can try "$2")

Write & use functions:

`int round_to_5(int cents)`
that returns the value rounded off to the nearest multiple of 5.

`int try_one_coin(int *cents, int coin)`
that reduces `cents` amount by the value of `coin` as many times as is possible, and returns the number of coins that should be issued.

`void print_change(int cents)`
that employs `try_one_coin` to print out the coins that should be issued for `cents`.

Lab Time

* Group+Individual Work in Break-out Rooms, Discuss with Friends
* Summon Anh when having questions/problems
* Also use grok for programming questions

Also ask questions related to Quiz1 materials

Pointer: &N for reference
***PN for de-reference**

Minimal Exercises:
W04: Exercise 5.6: Finding amicable pairs
W5X: Exercise 6.9: Calculating change using functions

Additional Exercises you might want to finish right now:
W04: Exercise 5.2: Find the largest among four integers
W04: Exercise 5.3: Compute an integer power
W4X: Exercise 5.13-same as 5.3: Compute an integer power using recursion

W4X: Exercise 5.11: Summing a real-valued sequence
W4X: Exercise 5.14: Recursive function for logstar
W04: Exercise 5.7: Functions isupper() and tolower(), normally defined via the ctype.h library

Other exercises:
W04: Exercise 5.1: Find the larger of two integers
W4X: Exercise 5.4: Combinations
W4X: Exercise 5.8: Computing near-equality for doubles
W4X: Exercise 5.15 (mutually recursive)

Suggested order: W04.5.2 W5X.6.9 W04.5.2 W04.5.3 W4X.5.13 W4X.5.11 W4X.5.14 W04.5.7

Not-for-use Slides

grok exercises this week: classification

W04: Exercise 5.1: Find the larger of two integers
W04: Exercise 5.2: Find the largest among four integers
W04: Exercise 5.3: see 5.13

W04: Exercise 5.7: Functions isupper() and tolower(), normally defined via the ctype.h library
W04: Exercise 5.6: Finding amicable pairs

With pointer parameters:
W5X: Exercise 6.9: Calculating change using functions (but we won't use arrays in this workshop)

Recursive
W4X: Exercise 5.13: Compute an integer power using recursion (=5.3)
W4X: Exercise 5.14: Recursive function for logstar
W4X: Exercise 5.15 (mutually recursive)

Others
W4X: Exercise 5.4: Combinations
W4X: Exercise 5.8: Computing near-equality for doubles
W4X: Exercise 5.11: Summing a real-valued sequence