COMP20005 Workshop Week 6

Plan & Notes:

1 Scopes, *exercise 6.2*

2 Pointers, Pointers as Function Arguments,
quiz
exercise 6.5 or equivalent
discuss exercise 6.9

3 Recursive Functions *(time permitted)*

4 Pre-Notes on Assignment 1:

- released: Wed evening this week – **W6**
- due: 6pm **Fri** 29 April – **W8** (the week after Easter)
- action: start early, finish major job before workshop week 7 because
  - you probably want a good holiday season,
  - questions in W8 might be too late, and
  - you might not have chance to ask questions in workshop **W8** due to Anzac Day holiday (**Monday** in **W8**).

5 ***Lab: 6.9, triangle.c, other ex from C05 and C06***

Scopes, local & global 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

function `fact` available here

Scopes, local & global variables

```
#include <stdio.h>
int ga, gb;
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;
}
```

scope of *local* variables `argc`, `argv`, `n`, and `val`

scope of *local* variables `n`, `i`, and `f`

scope of function `fact`

scope of global varables `ga` and `gb`

A Rule: Never use global variables

```
#include <stdio.h>
int ga, gb;
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;
}
```

scope of *local* variables `argc`, `argv`, `n`, and `val`

scope of *local* variables `n`, `i`, and `f`

scope of function `fact`

Discuss: exercise 6.02

***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 }
```

Quiz 1

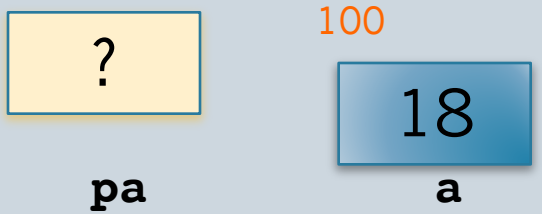
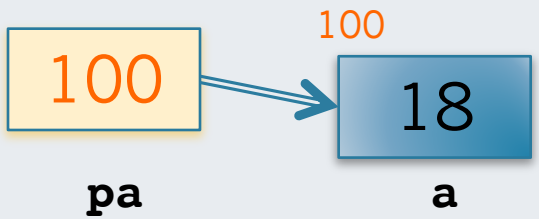
In executing the program:

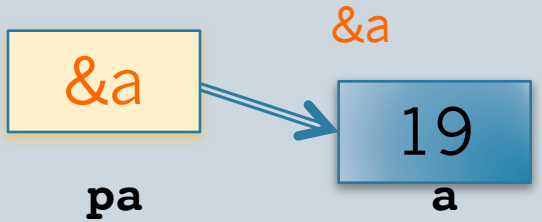
```
int a=100, b=200;
void f(int a) {
   a++;
   print("1: a= %d b= %d\n", a, b) ;
}
int main(int argc, char *argv[]) {
  int a=5, b= 10;
  f(a);
  print("2: a= %d b= %d\n", a, b) ;
  return 0;
}
```

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

Pointers: check your understanding

`int a= 18;`		**a** is a name, refers to a cell in the memory, capable for holding an **int**, now that cell contains the value of 18. Suppose that the cell is at address 100
`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 to:
		`*pa = *pa + 1;`
		and
		`a = a + 1;`
		Note: we, programmers, don't need to know the actual value of &a.

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 4444, 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's `&(n+5)`

f) What's `*n`

g) What's `*(n+5)`

Pointers – application in function parameters

1	`int n=10;`	
2	`printf("%d", n);`	What sent to `printf` ?
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){`
	`  ???`
	`}`	

example: 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**

Quiz 3

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 4

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

Quiz: Check your answers

- Q1: A
- Q2: A
- Q3: D
- Q4: C

Class Exercise (a variant of 6.05)

Suppose we have the following main() program with some missing parts ?:

```
#include <stdio.h>
?1
int main(int argc, char *argv[]) {
  int a=2, b=3, c= 1;
  printf("Original                    : a, b, c= %d, %d, %d\n");
  sort2(?2, ?3);
  printf("After sort2 for a and b: a, b, c= %d, %d, %d\n");
  sort2(?4, ?5);
  printf("After sort2 for b and c: a, b, c= %d, %d, %d\n");
  return 0;
}

// function to sort 2 integers in increasing order
?6 sort2 ( ?7 x, ?8 y) {
  ?9
}
```

Fill in all the missing parts ??? so that the output will be:

```
Original                    : a, b, c= 2, 3, 1
After sort2 for a and b: a, b, c= 2, 3, 1
After sort2 for b and c: a, b, c= 2, 1, 3
```

Recursive functions

Recursive function: function that calls itself

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

exercise 6.09
(a revision of exercise 3.06)
ALSO: using VS and gcc

(the coming) Assignment 1: dates & notes

Time	Likely Works
W6: Wed 06 to before the Workshop W7	• **Read spec! Watch the movie!** try to understand requirements & resources • Examine the supplied solution for previous year, focus on the `main()` function and imagine the structure of your program for assignment 1 • start your program (as advised in specs & video) right away • finish stage 1 and try "pre-submission testing" • do stage 2 and other stages • regularly check FAQ, the marking rubric, and the Discussion Forum
W7 workshop	• ask questions, make sure that you fully understand the requirements and the marking rubric • make sure that you know how to, and understand the value of pre-submission testing • try & make sure that you can submit • finish your implementation of at least stage1 and 2
W7, W8 and the Easter Week	• regularly check FAQ, the marking rubric, and the Discussion Forum • re-check your program against all the points in the marking rubric • do the pre-submission testing, carefully read the verify report, make sure that the report is clean (from any kind of warnings/errors) • do official submission after major changes
6:00PM Fri 29 APR	• enjoy your Friday drink with the friends who, like you, already submit.

Lab

- **Re-implement 6.5 if still in doubt**
- **Implement 6.9 if not done**
- *Do the exercise with triangle.c as described in LMS Week 6 Workshop Content*
- *Design and implement a solution to Exercise 5.5*: A number is perfect if its equal to the sum of its factors (including 1, but excluding itself), for example 6 (6=1+2+3). Write a function int isperfect(int n) that return true if n is perfect and false otherwise. Write a function int nextperfect(int n) that return the first perfect number greater than n. Write a main() that that prints all perfect numbers in the range 1.. 36,000,000.
- implement not-yet-done Exercises in grok C05 and C06

More on Recursive Functions

Recursive functions: yet another 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.

Base cases:

General case:

Fibonacci cnt.

- 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 $n=0, 1$

Fibocacci, the function.

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.

```
// note: this function is very inefficient!
int fib(int n) {









}
```