COMP20005 Workshop Week 6

Preparation:
- Download this .ppt (or .pdf if you don't have Powerpoint) from `github.com/anhvir/c205`
- Open `grok` and `LMS`

1	**Plan:** Scopes, *ex. 6.2 (discussion: scopes)*
2	Pointers, Pointers as Function Arguments: • Group exercise, • Class exercise 6.9
3	*Lab: 6.5, 6.9, grok W05*
4	**Assignment 1: how to start**
L M S	LMS: discuss 6.2, 6.5. Implement 6.9, 6.9 with functions. triangle.c and address of argument vars. exercise in W5

Scopes, local & global variables

```c
#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

```c
#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

```c
#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: 6.2 (W6)

| *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<br>2<br>3<br>4<br>5<br>6<br>7<br>8<br>9<br>10<br><br>11<br>12<br>13<br>14<br>15<br><br>16<br>17<br>18<br>19 | ```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;
}
``` |

Variable and address, operators & and *

What happens (what the system does) when:

```
int n=10;        is executed?
```

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

```
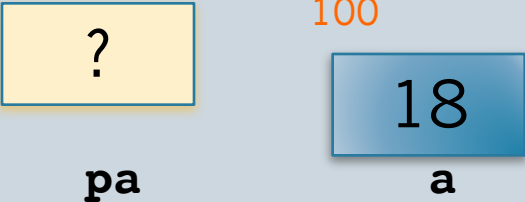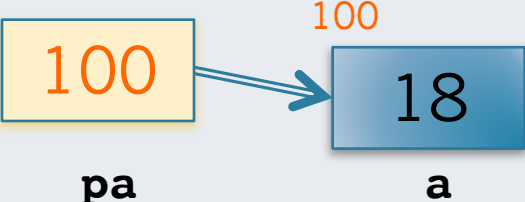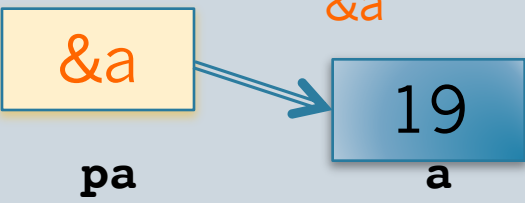*pn= 100;
```
  the value of pn is _____, of n is _____

Pointers: check your understanding

`int a= 18;`	**18** **a**	**a** is a location in the memory, interpreted as an **int,** with value of 18
`int *pa;`	**?** 100 **18** **pa** **a**	**pa** is an **int pointer**, it can hold the address of an **int**
`pa= &a;`	**100** 100 **18** **pa** **a**	**pa** now holds the address of **a**, or, it "points" to **a**. ***pa** is another method to access **a**

`*pa= *pa + 1;`	**&a** &a
	19
	pa **a**
		These statements are equivalent:
		`*pa = *pa + 1;`
		` a  =   a + 1;`
		
		`*pa = a + 1;`
		`a   = *pa + 1;`

Pointers – application in function parameters

1	`int n=10;`
2	`printf("%d", n);`
3	`scanf("%d", &n);`
4	`swap(&n, &m);`

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

What sent to `printf` ?

What sent to `scanf`?
What scanf do to `&n`, to `n`?

What passed to `swap`?
Can this call make change to `&n` or `&m?`
Can this call make change to `m` or `n`?

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 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

In executing the program:

```c
#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	B
14	1
C	**D**
6	9

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

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

```c
#include <stdio.h>
???
int main(int argc, char *a
  int a=2, b=1, c= 3;
  sort2(???1, ???);
  printf("After sort2 for
  sort2(???, ???);
  printf("After sort2 for
  return 0;
}

// function to sort 2 inte
??? sort2 ( ??? x, ??? y)
  ???
}
```

> Instructions:
> In your break-out room (of 3-4 students) discuss the solution to this exercise. One person should take a leading role by sharing the next page of this slide and do the filling in. Don't hurry! The main purpose is to have an *easy chat* while trying to understand pointers, &, and *.
>
> You will have 10 min for discussion.

Fill in all the missing parts ??? in /*1*/, /*2*/… /*5*/ so that the output will be:

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

Group Work

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

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

// function to sort 2 integers in increasing order
??? sort2 ( ??? x, ??? y) {        /* 3 */
  ???                              /* 4 */
}
```

*Fill in all the missing parts ??? in /*1*/, /*2*/... /*5*/ so that the output will be:*

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

do_together: 3.6 revisited
Top-Down Design Using Functions

Use 3.6 in grok W03: Suppose that coins are available in denominations of 50c, 20c, 10c, 5c, 2c and 1c. Write a program that reads an integer amount of cents between 0 and 99 (your program might check that the input value falls within this range) and prints out the coins necessary to make up that amount of money. *Use functions and top-down design!*

```
Enter amount in cents: 42
The coins required to make 42 cents are:
give a 20c coin
give a 20c coin
give a  2c coin
amount remaining: 0c
```

Note: Start with your current version of Ex. 3.6, but remove most of the body of the main() function, keeping only the input data part.

After finishing, copy this program and paste to Ex. 6.9, then do the extended requirements regarding the $2 and $1 coins, and change the output format as required by Mark, which looks like:

```
give 2 20-cent coins
give 1  2-cent coins
```

Ass1: deadline & notes

- Note: It would be easier and more convenient to use gcc/jEdit to do this assignment.

Time	Perhaps (the slowest & laziest schedule):
this week	• **Read spec! Watch the movie!** understand requirements & resources • start your program with the skeleton, sign the declaration • try and make sure that you can do "pre-submission testing" (see "Assignment Testing Instructions") • try to read and print out the original data • finish stage 1 and try "pre-submission testing"
next week	• make sure that you understand the requirement fully • Examine the supplied solution for 2020, and learn the way to break • finish your implementation of at least stage1 and 2, do the testing regularly • regularly check FAQ, the marking rubric, and the Discussion Forum • make sure that you can submit
by Thu 29/04	• check: declaration included and signed • check: comment "Programming is fun" is there at the required spot • check your program against marking rubric and test data • do the pre-submission testing, carefully read the verify report, make sure that the report is clean (from any kind of warnings/errors) • do final submission
5:00PM Fri 30	• enjoy your Friday drink with the friends who, like you, already submit.

Lab

- Re-implement 6.5 and 6.9 if still in doubt

- Implement 6.9

- *Do the exercise with triangle.c as described in LMS Week 6 Schedule*

- *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 W05