

# COMP20005 Workshop Week 7















1	Arrays
2	Discuss: ex 7.1, 7.2, 7.3
3	Assignment 1: <ul style="list-style-type: none"><li>• Q&amp;A</li><li>• Submission process</li><li>• Strategies</li></ul> Lab: Work on assignment 1

# Arrays






















# arrays: declaration & use

	statements	variables in memory ( <i>after</i> LHS statements)
1	<code>int i, A[5];</code> /* equivalent to declaring 6 variables, each is of data type <code>int</code> */	<div> <div>i</div> <div>A[0]</div> <div>A[1]</div> <div>A[2]</div> <div>A[3]</div> <div>A[4]</div> </div>
2	<code>A[0] = 10;</code> <code>i = A[0] * 2;</code>	
3	<code>i = 2;</code> <code>A[i] = 20;</code>	
4	<code>for (i=0; i&lt;5; i++) {</code> <code>A[i] = i*i;</code> <code>}</code>	
5	<code>for (i=0; i&lt;3; i++) {</code> <code>scanf ( "%d", &amp;A[i] );</code> <code>}</code> /* supposing that input from keyboard is <b>10 20 30</b> */	

# arrays...

	statements	variables in memory (after LHS statements)						
1	<code>int i, sum=0, A[5]= {0,1,2,3,4};</code>	i	sum	A[0]	A[1]	A[2]	A[3]	A[4]
								
2	<code>for (i=0; i&lt;5; i++) sum += A[i];</code>							

# arrays...

	statements	variables in memory (after LHS statements)						
1	<code>int i, sum=0, A[5]= {0,1,2,3,4};</code>	i	sum	A[0]	A[1]	A[2]	A[3]	A[4]
								
2	<code>for (i=0; i&lt;5; i++) sum += A[i];</code>							
3	<code>for (i=0; i&lt;4; i++) { A[i+1]= A[i]; }</code>							

# arrays...

	statements	variables in memory (after LHS statements)						
1	<code>int i, sum=0, A[5]= {0,1,2,3,4};</code>	i	sum	A[0]	A[1]	A[2]	A[3]	A[4]
		0	0	1	2	3	4	
2	<code>for (i=0; i&lt;5; i++) sum += A[i];</code>	5	10	0	1	2	3	4
3	<code>for (i=0; i&lt;4; i++) { A[i+1]= A[i]; }</code>	0	10	0	0	0	0	0

Notes: No operation with *whole arrays* is allowed. With declaration:

```
int A[3]={10,20,30}, B[3];
```

we cannot write:

```
B= A;
```

```
if (A==B) A= A+B;
```

# Arrays : ...

```
#define MAX_N 100
```

```
...
```

```
double X[MAX_N];
```

```
int n;
```

# Arrays as function arguments

- With a function prototype, say:

```
int change_array(int A[], int n);
```

we should note that:

- the formal parameter `A[]` is an array of `int`, but no size is specified in “`int A[]`”,
- instead, there is another parameter, `n`, which (normally) specifies the size (number of elements) of `A[]`,
- the array formal parameter `A[]` is *both input and output* of function `change_array`.
- With the call “`change_array(B, 10)`”:
  - the action in formal parameter `A[]` is actually happen to `B[]`.
- Why? Because the name `B` of array `B[]` is just an address...



# An Example

## Ex 7.1 a)

*Write a function*

```
int all_zero (int A[], int n)
```

*which returns true if the elements of  $A[0]$  to  $A[n-1]$  are all zero, and false if any of them are non-zero.*

## Ex 7.1 b)

*Write a function that return the minimal values of an array.*

# Insertion Sort

## Ex 7.2

*Modify so that the array of values is sorted into decreasing order.*

```
/* insertion sort: sorting elements A[0] to A[n-1]
    into non-decreasing order */
/* assume that A[0] to A[n-1] have valid values */
1  for (i= 1; i<n ; i++) {
2      /* swap A[i] left into correct position */
3      for (j= i-1; j>=0 && A[j+1]<A[j]; j--) {
4          /* not there yet */
5          int_swap( &A[j] , &A[j+1] );
6      }
7  }
/* and that's all there is to it */
```

## Ex 7.3

*Modify so that after the array has been sorted only the distinct values are retained in the array (with variable n suitable reduced) .*

```
10  for (i= 1; i<n ; i++) {  
20      for (j= i-1; j>=0 && A[j+1]<A[j]; j--) {  
30          int_swap( &A[j] , &A[j+1] );  
40      }  
50 }
```

Input : 1 8 15 3 17 12 4 8 4

Sorted: 1 3 4 4 8 8 12 15 17

Output: 1 3 4 8 12 15 17

# This week in github

- `ass1_notes.pdf`
- `array.c` : Simple examples on using arrays, can be used as skeleton for exercises 7.3 and 7.4.
- `structs.c` and `students.txt`: ...

# Assignment 1: the 4C process

- 1. CREATE:** Create a directory, say `ass1`, download all related files into `ass1`, then create `ass1/ass1.c` that satisfies the requirements 😊
- 2. COPY:** Copy the whole directory `ass1` to your university's drive `H:`. Note: if you work in lab computers, you don't need to do this step.
- 3. CHECK:** login into the server `dimefox.eng.unimelb.edu.au`, then on that server, check/test to make sure that your program is correct.
- 4. COMMIT:** while in `dimefox`, commit/submit your `ass1.c`, and verify.



# Assignment 1: the 4C process

**1. CREATE:** Create a folder, say `ass1`, download all related files into `ass1`, then create `ass1/ass1.c` that does the required job 😊

In `minGW` or `Terminal` window, when you are in your `COMP20005` (or similar) directory, do:

```
mkdir ass1
```

```
cd ass1
```

then, download all needed files from `LMS` → `Assignment1` to this directory. That includes 6 files listed in point 5 of

`LMS`→`Assignment1`:

`drones0.tsv`, `drones0-out-mac.txt`, `drones0-out-dos.txt`

`drones1.tsv`, `drones1-out-mac.txt`, `drones1-out-dos.txt`.

Now, it's time to build `ass1.c` (you can choose other name).

# Assignment 1: the 4C process

**2. COPY:** Copy the whole folder `ass1` to your university's drive `H:`.

Skip this step if you are working on a lab's PC. Otherwise, supposing that `ass1` is your current folder. In `MinGW` window (or Mac Terminal) :

<code>cd ..</code>	make parent of ass1 the current directory
<code>scp -r ass1 XXX@dimefox.eng.unimelb.edu.au:</code>	copy whole directory to your H:

Notes:

- replace `XXX` by your uni's login name
- there is a colon `:` at the end of `scp`
- if you do that outside uni, you need to run `VPN` first (instructions available in LMS)

# Assignment 1: the 4C process

## 3. CHECK:

login into **dimefox**: from your **minGW/Terminal**, type:

```
ssh XXX@dimefox.eng.unimelb.edu.au
```

You will see your prompt changed to

```
bash $
```

Now, use **ls** and **cd** to navigate into your **ass1** directory and compile:

```
bash $gcc -Wall -o ass1 ass1.c
```

Then, test your program against, say, **drones0.tsv**:

```
bash $./ass1 < drones0.tsv > out_0.txt
```

that will write output to **out\_0.txt**. Compare that with Alistair's one by:

```
bash $diff out_0.txt drones0-out-mac.txt
```

which lists the differences between 2 files. No output means the 2 files are exactly the same (bravo!).

# Assignment 1: the 4C process

**4. COMMIT:** while in `dimefox`, submit your `ass1.c`, and verify.

To submit you must be on `dimefox`, and `ass1` must be your current folder. Use the command (you cannot change the bolded part in the next 2 commands):

```
bash $ submit comp20005 ass1 ass1.c
```

After that you can verify your submission using:

```
bash $ verify comp20005 ass1 > receipt.txt
```

It's a good idea to open `receipt.txt` with `jEdit` to see its content. If you are not working with a lab's computer, to be able to use `jEdit`, on `receipt.txt`, you first need to copy that file to your own computer:

```
bash $ exit
```

```
$ scp XXX@dimefox.eng.unimelb.edu.au:ass1/receipt.txt .
```

# Assignment 1: the 4C process

## Today Work: minimal requirement

Create a simple `ass1.c` file (first, by copying the skeleton supplied in

`LMS→Assignment1→Point1→Link2`, and sign it), then try all other 3 steps. Make sure that you can submit, at least from a lab PC.

Then, incrementally **CREATE** your `ass1.c`, do **COPY-CHECK-COMMIT** after every major development.

# Assignments: advices

- *Be active in the subject's Discussion Forum!*
- *Visit **LMS→Assignment 1** frequently!*
- *Make as many submissions as you want, only the last one (before deadline) counts.*
- *To simplify, do commit at uni. If you want to commit from home, beware of **VPN**!*
- *Read the specifications carefully.*
- *Check your program carefully, at least with all supplied data. Do the testing on **dimefox**.*
- *Read the marking rubric carefully and try to maximize your marks!*
- *Read the **sample solution to 2015** (in LMS.Assignment1, point 6), focusing on **main( )**. You can learn something from there.*
- ***START EARLY, START RIGHT NOW!***