

# COMP20005 Workshop Week 7

## Preparation:

open [grok](#) and [VS](#) (or your favorite tools for editing and gcc)  
download this slide set ([ws7.pdf](#)) from [github.com/anhvir/c205](https://github.com/anhvir/c205) if you like

**1**      **Arrays:** understanding, Q&A, arrays and pointers

**2**      **Classwork based on:** Ex 7.01, Ex 7.04

**Assignment 1:** Q&A, marking rubric

### Lab:

- do your assignment 1 and test the submission, or
- do exercises in C07















**Important:** no workshop on ANZAC day which is Monday Week 8  
Arrangement for a replacement workshop?

# Arrays = ?

## arrays: declaration & use

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

## arrays...

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

# 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

# Arrays : using in C



```
#define SIZE 5
...
```

```
int X[SIZE]= {1, 2, 3};
int n= 3;    // n <= SIZE
// can we?
```

```
X[n]= 10; n++;
```

```
(X+2)[2]= 7;
```

```
X[5]= 100;
```

In computer memory, an array is stored as *a block of contiguous cells*, one cell for one array's element.

Essentially, an array is defined by 4 objects:

- **X**: the array's name, which is actually a pointer to the start of the memory block
- **int**: the data type of each element of the array
- **SIZE**: an **int** constant representing the array's capacity
- **n**: a buddy **int** variable, representing the number of elements that are currently employed

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

With declaration:

```
int A[3]={1,2,3}, B[3];
```

we cannot write:

```
B= A;
```

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

# Arrays as function arguments

- With a function prototype, say:

```
int sum_array(int A[]);
```

we should note that:

- the formal parameter `A[]` is an array of `int`, but no size or capacity is specified in “`int A[]`”, as such this parameter only specify the starting address of the array, it’s also equivalent to “`int *A`”.
- as such, there is no way for the above function to compute the sum of A

Array Name is a Pointer! a constant pointer!



# Arrays as function arguments

```
int sum(int A[], int n) {  
    int i, s= 0;  
    for (i=0; i<n; i++) {  
        s += A[i];  
    }  
    return s;  
}
```

- With the above function and the declarations:

```
int B[10]= {1,2,3,4,5,6,7,8,9};
```

For each of the following statements: valid? If yes, what's the output?

```
printf("%d\n", sum(B, 10));
```

```
printf("%d\n", sum(B, 5));
```

```
printf("%d\n", sum(&B[0], 5));
```

```
printf("%d\n", sum(B+0, 5));
```

```
printf("%d\n", sum(B+3, 2));
```

## DoltTogether: Exercise 7.1, then 7.4

PLEASE: If you want, use `VS` and `gcc` or similar tools instead of `grok`.

### NOTES:

**For 7.1:** (Write function `int all_zero(int A[], int n)` that returns 1 or 0)

Save your time by simplifying the main function to

```
int A[MAX_N], n;  
// add: reading the array  
printf("all_zero(A,3)= %d, all_zero(A,4)= %d\n",  
       all_zero(A+4,5), all_zero(A,MAX_N+1));  
/* should print out 0 and 1, why? */
```

and use data:

0 0 0 1 6 0 0 0 0 0 8 7 0

**Your task:** Start a new program on `VS` (or `grok playground` if you are too loyal to `grok`), copy and paste the above code into the `main()` function. Then continue the work together with Anh.

## Discussion 2: Exercise 7.4

*Write a program that reads as many as 1,000 integer values, and counts the frequency of each value in the input:*

`./program`

Enter as many as 1000 values, ^D to end

1 1 1 3 3 3 3 3 4 6 4 3 6 10 3 5 4 3 1 6 4 3 1

17 values read into array

Value Freq

1 3

3 5

4 4

5 1

6 3

10 1

*How?*

## Discussion 2: Exercise 7.4

*Write a program that reads as many as 1,000 integer values, and counts the frequency of each value in the input:*

*So we need to:*

- 1. Input value for an array*
- 2. Sort an array in increasing order*
- 3. Count and print out frequencies*

# Ass1: Q&A make sure that you understand the tasks

*Carefully read the spec and the marking rubric*

*Read the Discussion Forum and:*

- *answer if you are confident,*
- *post **new** questions if needed*

# Ass1: Maximize Your Mark

*Check your code against the marking rubric*

*Examine the 2020 sample solution: you don't need to understand, but you still can learn something from here*

*Questions on marking rubric?*

- *missing Authorship Declaration at top of program, -5.0;*
- *incomplete or unsigned Authorship Declaration at top of program, -3.0;*
- *use of external code without attribution (minor), -2.0;*
- *use of external code without attribution (major), -10.0;*
- *significant overlap detected with another student's submission, -10.0;*

# Marking Rubric: The importance of Style & Structure

Stage	Presentation	Structure	Execution	max accumulated mark
1	+5	+2	+1	8
2		+4	+4	16
3		+0	+4	20
all	5	6	9	

Failed code  
could even  
get 11

Absolutely  
correct program  
could get only 9

Correct and  
good Stage 1+2  
alone could get  
16

# ASS1 Marking Rubric: a few keys in Presentation

- use of magic numbers, -0.5;
- #defines not in upper case, -0.5;
- unhelpful #define
- bad choices for variable names, -0.5;
- bad choice for function names, -0.5;
- absence of function prototypes, -0.5;
- inconsistent bracket placement, -0.5;
- inconsistent indentation, -0.5;
- excessive commenting, -0.5;
- insufficient commenting, -0.5;
- lack of whitespace (visual appeal), -0.5;
- lines >80 chars, -0.5;
- use of constant subscripts in 2d arrays, -1.0;
- other issue: minor -0.5, major -1.0
- empty program, or helloworld, or etc, -5.0;
- comment at end that says "algorithms are fun", +0.5;
- overall care and presentation, +0.5;

use #define for meaningful constants  
#define-ed names should be in upper case  
variable names in lower case (except single-letter array name)  
variable names should be expressive

add function prototypes before main()  
no function implementation before main()!

you can use sample programs in lectures as the model

each function header should have a comment  
add comments for non-trivial code segment

DO IT !  
Easy way to get back 0.5 or 1 mark



# ASS1 Marking Rubric: a few keys in Structure

- **global variables, -1.0;**

Global variables are NOT allowed!

- **main program too long or too complex, -1.0;**
- functions too long or too complex, -0.5;
- overly complex function argument lists, -0.5;

- function should not be long
- and should not have too many arguments

- insufficient use of functions, -0.5;
- duplicate code segments, -0.5;

- when having a few line, or a complicated line similarly-duplicated, think about creating a new function!

- overly complex algorithmic approach, -0.5;

- avoid too many levels of nesting `if` and loops
- don't make the marker wonder too much to understand your code!

- unnecessary duplication/copying of data, -0.5;

- For example, think carefully before you copy an array!

- **other structural issue: minor -0.5, major -1.0;**

# ASS1 Marking Rubric: a few keys in Execution

- **failure to compile**, -3.0;
- unnecessary **warning messages** in compilation, -2.0;
- runtime segmentation fault on test1 with no output generated, -2.0;
- runtime segmentation fault on test2 with no output generated, -2.0;

Your program might be compiled OK in your computer, without any warning.  
But it might have compiler errors or warnings on the testing machine.

→ carefully read the submission report (compiler messages are at the beginning of the report)

- **numerically incorrect Stage 1 output on test1 or test2 (or both)**, -0.5;
- **different incorrect Stage 1 output on test1 or test2 (or both)**, -0.5;
- **numerically incorrect Stage 2 output on test1 or test2 (or both)**, -0.5;
- **different incorrect Stage 2 output on test1 or test2 (or both)**, -0.5;
- **numerically incorrect Stage 3 output on test1 or test2 (or both)**, -0.5;
- **different incorrect Stage 3 output on test1 or test2 (or both)**, -0.5;
- **notably different Stage 1/2/3 output for test1 or test2**, -0.5;
- **other execution-time issue not already covered**, -0.5;

Again, check the verification report!

When testing compare your outputs with expected outputs using command `diff`:

```
./ass1 < data1.txt > out1.txt  
diff out1.txt data1-S1-out.txt
```

Desirable outcome: EMPTY output from command `diff`.

If you have non-empty output:

- + The lines starting with `<` is from the first file of the `diff`
- + The lines starting with `>` is from the second file
- + You can just do the testing submit to see the diff

# Assignment 1: how to start

	using VS/gcc or equivalent	using grok
preparation	<ul style="list-style-type: none"><li>• make a new folder, say <code>ass1</code></li><li>• copy <code>ass1-skel.c</code> to, say, <code>ass1.c</code> in the <code>ass1</code> folder</li><li>• open <code>ass1.c</code> and sign in the Declaration</li><li>• download all files listed in Test data into folder <code>ass1</code> [or, for now, just <code>asx-5.tsv</code> and <code>asx-5-out.txt</code>]</li></ul>	using Assignment1
programing	fill in your code in <code>ass1.c</code>	fill in your code in <code>program.c</code>
testing	see demonstration	follow the instructions given in LMS
testing on dimefox	see demonstration	
submitting	LMS → Assignments → Assignment 1 → Start Assignment → (upload .c file)	

# Assignment 1: tersting

## TESTING IN YOUR COMPUTER

- *Using redirection when running/testing your program:*  
`./myass1 < asx-5.tsv > out_5.txt`
- *Your program's output must be the same as the expected, ie. the command*  
`diff out_5.txt asx-5-out.txt` *or*  
`diff out_5.txt asx-5-out-dos.txt` *(if using Windows)*  
*must give empty output (that is, no difference).*
- *Remember that your code might work well on the 2 supplied data sets, but fail on some other...*

## TESTING WITH `dimefox` :

- *Wait for the verification report*
- *Read the verification report carefully. Your program might work perfectly in your computer but fail in the testing computer(s).*
- *Try to submit early to avoid unexpected technical problems.*

# Compiler warnings: -2.0

overly complex algorithmic approach, -1.0;

Examples of overly complicated:

- sort the data when not actually required
- too many levels of nested loops/if

## duplicate code segments: turn similar segments into a function

- having 2 or more lines similar? Think if you should form a new function.

# duplicate code segments with boring repetitive fragment

```
#define RED 1  
#define BLUE 2  
#define BLACK 3
```

...

```
if (colour==RED) {  
    printf("colour is RED\n");  
} else if (colour==BLUE) {  
    printf("colour is BLUE\n");  
} else if (colour==BLACK) {  
    printf("colour is BLACK\n");  
}
```

→



# duplicate code segments with boring repetitive fragment

```
#define RED 1  
#define BLUE 2  
#define BLACK 3
```

...

```
if (colour==RED) {  
    printf("colour is RED\n");  
} else if (colour==BLUE) {  
    printf("colour is BLUE\n");  
} else if (colour==BLACK) {  
    printf("colour is BLACK\n");  
}
```

→

```
#define RED 1  
#define BLUE 2  
#define BLACK 3
```

...

```
char *names[4]= {"", "RED", "BLUE", "BLACK"};  
for (int i=1; i<=3; i++) {  
    printf("colour is %s\n", names[i]);  
}
```

→

# ASS1 Q&A:

# LAB: do Assignment 1 OR exercises in C07

## Notes:

- For assignment 1, you can use VS/gcc (preferable), or grok.
- There is an additional tool in grok: the Makefile. You can also copy that Makefile into your VS directory, make a change to “program”, and try.

# Assignment 1: A reasonable way to start with VS/gcc

	command/action	explanation
1	<code>cd ~</code>	set your home directory as your <i>current directory</i>
2	<code>mkdir ASS1</code>	<i>make a new directory, and of assignment files will be placed in that directory</i>
3	<code>cd ASS1</code>	<i>change current directory to ASS1</i>
4	<code>ls</code>	<i>list the content of the current directory, it should be empty</i>
5	navigate to the assignmen1 FAQ page and download file ass1-skel.c (2 <sup>nd</sup> link of point 1), and all the files listed in point 7. You should download the files to the ASS1 directory.	
6	<code>ls</code>	<i>now you should see the downloaded files</i>
7	<code>mv ass1-skel.c ass1.c</code>	<i>rename the skeleton file to your assignment</i>
8	using VS <code>editor</code> to do your assignment	
9	<code>gcc -Wall -o ass1 ass1.c</code>	<i>compile the program</i>
10	<code>./ass1 &lt;asx-5.tsv &gt;out5.txt</code>	<i>run program with redirection</i>
11	<code>diff out5.txt asv-5-out.txt</code>	<i>check if your output is the same as the expected</i>

# Additional Slides

## Discussion 3: typedef and struct for exercise 7.5

*Suppose that a set of "student number, mark" pairs are provided, one pair of numbers per line, with the lines in no particular order. Write a program that reads this data and outputs the same data, but ordered by student number. For example:*

823678 66

765876 94

864876 48

785671 68

854565 89

*On this input your program should output:*

Enter as many as 1000 "studnum mark" pairs, ^D to end

5 pairs read into arrays

studnum mark

765876 94

785671 68

823678 66

854565 89

864876 48

*Hint: use two parallel arrays, one for student numbers, and one for the corresponding marks. You may assume that there are at most 1,000 pairs to be handled.*

## Discussion 3: typedef and struct for exercise 7.5

*Use typedef to define a new data type. For example:*

```
typedef int integer;  
  
integer fact(integer n) {  
    ...  
}
```

## Discussion 3: typedef and struct for exercise 7.5

Use **typedef** to define a new data type.

Use **struct** to define a multi-component data type. For example:

```
typedef struct{  
    int stud_id;  
    double mark;  
} student_t;
```

```
/* return the average mark of n students,  
   the pairs (student_id, mark) are stored in array A[ ] */  
double average_mark(student_t A[ ], int n) {  
    ...  
}
```



## Discussion 3 :exampe of using typedef and struct

```
#include <stdio.h>
typedef struct{
    int stud_id;
    double mark;
} student_t;

int main(...) {
    student_t s1= {211111, 99.5), s2;
    student_t A[10];
    int i;
    s2= s1;
    s2.stud_id= 1000001;
    printf("id= %d mark=%f\n", s1.stud_id, s1.mark);
    for (i=0; i<10; i++) {
        scanf("%d %d", &(A[i].stud_id), &A[i].mark);
    }
    ...
}
```

## Discussion 3: typedef and struct for exercise 7.5

*Suppose that a set of "student number, mark" pairs are provided, one pair of numbers per line, with the lines in no particular order. Write a program that reads this data and outputs the same data, but ordered by student number. For example:*

823678 66

765876 94

*We can start with, for example:*

```
typedef struct{
    int stud_id;
    double mark;
} mark_t;

#define SIZE 30000
int main(...) {
    mark_t unimelb[SIZE];
    int n= 0;
    ...
}
```

And write functions to:

- input data to an array of `mark_t`
- sort an array of `mark_t`
- output data of an array of `mark_t`

**Remember to a) create a data file, and  
b) use redirection for inputting data.**