COMP10002 Workshop Week 7
stringing along

1. more on sorting with Ex 7.05 (sorting parallel arrays)?
2. strings and string processing, including
   - notes on typedef, argc, argv
   - discussing 7.12, 7.14, 7.15, 7.16
3. Implement 2 of them, also consider exercises in lec06.pdf

LAB:

- A1: Q&A
- Work on assignment 1

Exercise 7.05 – Sorting Parallel Arrays

Task 7.05

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**.

Sample input

```
823678  66
765876  94
864876  48
785671  68
854565  89
```

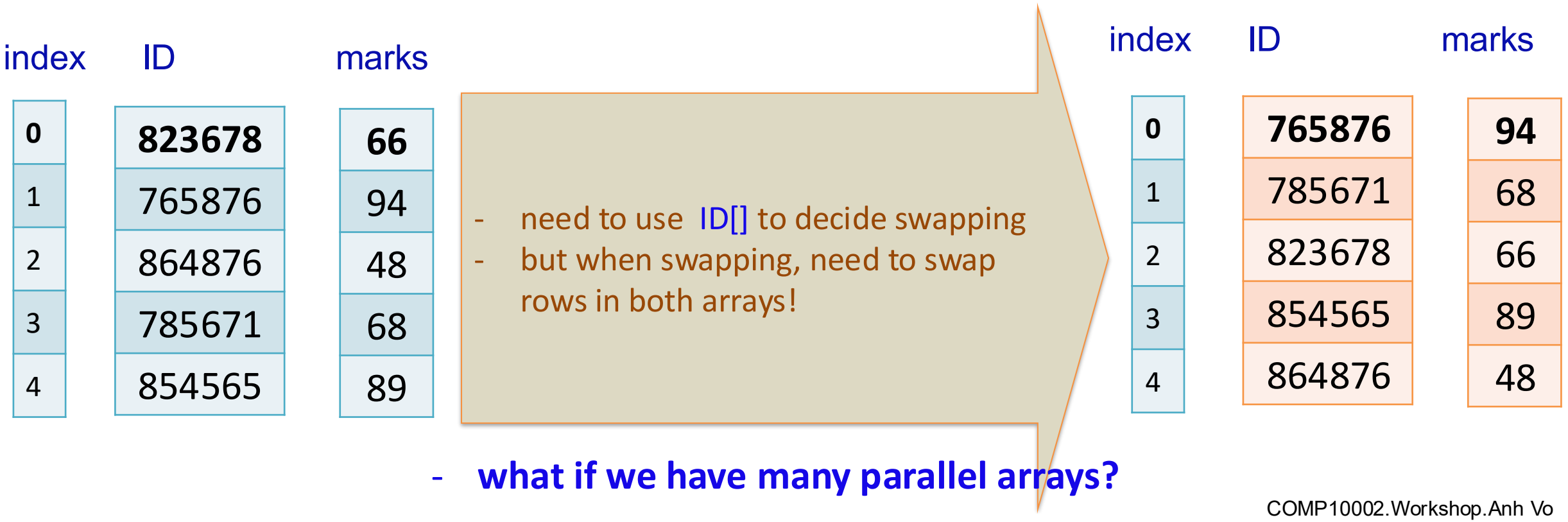
Sample output

```
765876  94
785671  68
823678  66
854565  89
864876  48
```

Sorting Parallel Arrays: Method 1 – performing swap in all arrays

Note: In this task, we have two parallel arrays:

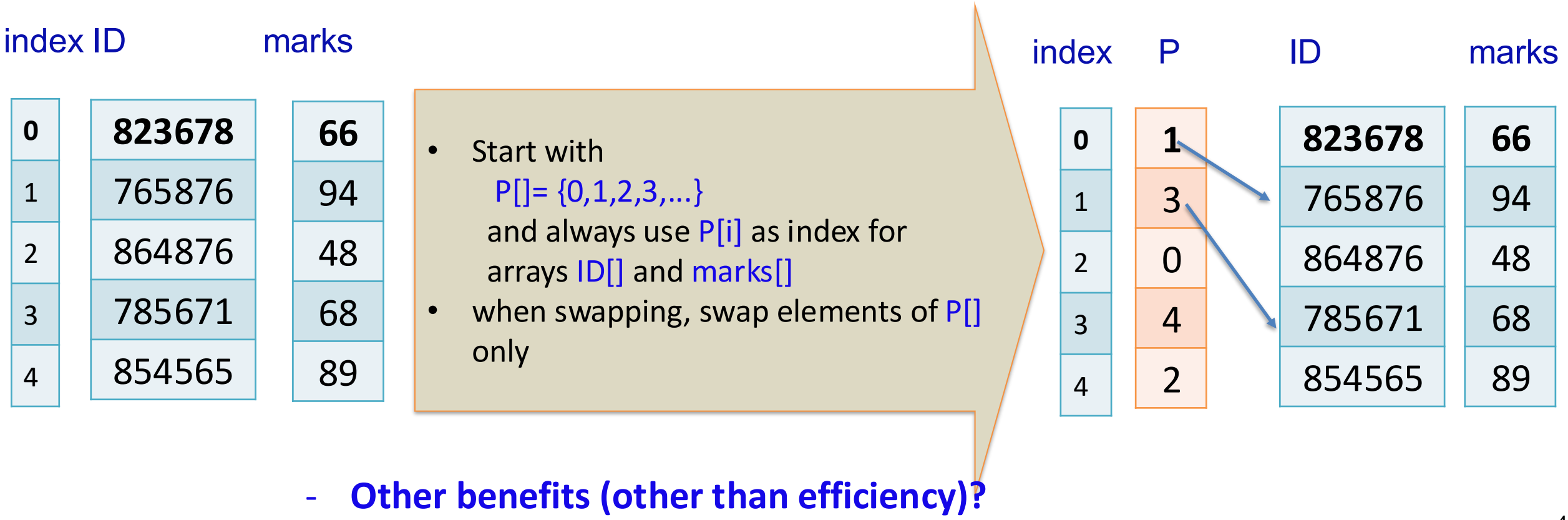
- ID[], where ID[i] is the student number of the i-th student
- marks[], where marks[i] is the corresponding mark of that student.



- **what if we have many parallel arrays?**

Sorting Parallel Arrays: Method 2 – indirect sort

Indirect sort uses an additional array to store sorted indices.
Aim: Building P[] as an array of sorted indices, ie:
P[i] is the position of ID[i] (and marks[i]) in the sorted order
➔ the sorted order is ID[P[0]], ID[P[1]], ID[P[2]], ...



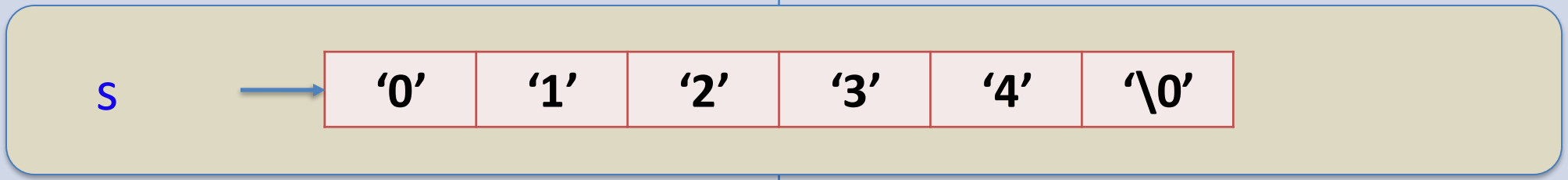
- **Other benefits (other than efficiency)?**

Strings: array notation & pointer notation

#define MAX 5

Array Notation	Pointer Notation
char s[MAX+1] = “01234”;	char *s = “01234”;

int n= strlen(s);
int i;
 	
for (i=0; i<n; i++) {
  printf(“%c”, s[i]);
}
	char *p;
	
	for (p=s; *p != ‘\0’; p++){
	  printf(“%c”, *p);
	}



#include <string.h>

```
#define MAX_STR_LEN 100
char s1[MAX_STR_LEN+1]= “123456789ABCDEF”, s2[MAX_STR_LEN+1];
```

```
printf (“String s contains %d characters\n, strlen(s1) );
if ( strcmp(s1, s2) == 0 )
        printf (“\”%s\” and \”%s\” are identical\n”, s1, s2);
```

IMPORTANT FUNCTIONS:

```
strlen(s)
strcpy(s2, s1);
strcmp(s1, s2)
```

Notes: lec06.pdf – Questions on ctype.h, typedef, argc, argv ?

- introduce ctype.h & string.h
- string1.c - printing the value of p and *p
- strcpy.c - implement strcpy using array & pointer notation

- getword.c - get next word using getchar()
  - a word is defined as a sequence of letters only

- words.c : same as getwords.c, but:
  - uses typedef char word_t[MAXCHARS+1];
  - includes a main() function

- progargs.c - demontration how to use argc, argv

Ex 7.12 (palindrome)

7.12: Write a function

int is_palindrome(char *)

that returns true if its argument string is a *palindrome*, that is, reads exactly the same forwards as well as backwards; and false if it is not a palindrome.

For example,

"rats live on no evil star"

is a palindrome according to this definition, while

"A man, a plan, a canal, Panama!"

is not. (But note that the second one is a palindrome according to a broader definition that allows for case, whitespace characters, and punctuation characters to vary.)

See `palindrome.net` for some interesting palindromes.

Write the function:

a) using array notation

b) using pointer notation

7.15 (anagram)

7.15: Write a function:

int is_anagram(char*, char*)

that returns true if its two arguments are an anagram pair, and false if they are not. An anagram pair have exactly the same letters, with the same frequency of each letter, but in a different order. For example, "luster", "result", "ulster", and "rustle" are all anagrams with respect to each other.

Rather more fun can be had if spaces can be inserted where required. A nice page at http://wordsmith.org/anagram discovered "programming is fun" can be transformed into both "prof margin musing" and "manuring from pigs".

Our assumptions: only consider *letters*, ignore all other characters.

Ex 7.14, 7.16: how-to?

7.14: Write a function

int atoi(char *)

that converts a character string into an integer value.

Discuss:

- How to convert a char digit like '3' to the numeric value (int 3 in this case)
- How to convert a string (a sequence of char), like "123", to the numeric value (here, 123)?

7.16: Modify a given program (that prints out distinct words contains words.c and getword.c) so that the frequency of each distinct word is listed too.

Strings - a quiz

```
char *s="123";
int n;
```

Which of the following fragments is correct and the best for computing n=atoi(s):

A.
```
for (int i=0; isdigit(s[i]); i++)
   n= n*10 + (s[i]);
```

B.
```
for (n=0; isdigit(*s); s++)
    n= n*10 + (*s-0)
```

C.
```
for (n=0; *s && isdigit(*s); s++)
    n= n*10 + (*s-'0');
```

D. none of the above

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*
  - *regular submit in Gradescope*
  - *allow time for, and do, check the verification report after submission*

Programming advices

- *KISS*
- *Incremental Development*
- *Keep the disciplines (indentation, bracket, #define, names...) from the start*
- *avoid unnecessary copies of arrays*
- *avoid code duplication*
- *avoid using too many level of nested loops and if*

Ass1: Maximize Your Mark

Check your code against the marking rubric

Examine the 2020 sample solution: you still can learn something from here even if you don't understand the task and the code.

Questions on marking rubric?

Section E: Academic Honesty

missing (or empty) Authorship Declaration at top of program, -5.0;

significant overlap detected with another student's submission, -10.0;

use of external code without attribution (minor), -2.0;

use of external code without attribution (major), -10.0;

Marking Rubric: The importance of Style & Structure

Stage	Presentation	Structure	Execution	max accumulated mark
1	+6	+4	+2	12
2	+1	+1	+2	16
3	+1	+1	+2	20
all	8	6	6	

Failed code could even get 14

Absolutely correct program could get only 6

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;
- bad choices for variable names, -0.5;
- bad choice for function names, -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

- absence of function prototypes, -0.5;

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

- inconsistent bracket placement, -0.5;
- inconsistent indentation, -0.5;

you can use sample programs in lectures as the model

- excessive commenting, -0.5;
- insufficient commenting, -0.5;

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

- lack of whitespace (visual appeal), -0.5;
- lines >80 chars, -0.5;

- other issue: minor -0.5, major -1.0

- comment at end that says "algorithms are fun", +0.5;
- overall care and presentation, +0.5;

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

ASS1 Marking Rubric: a few keys in Structure

- global variables, -0.5;

Global variables are NOT allowed!

- main program too long or too complex, -0.5;
- 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, -1.0;

- 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;

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.

ASS1 Marking Rubric: a few keys in Execution

failure to compile, -6.0;

unnecessary **warning messages** in compilation, -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 check with grok, and **check again when submitting**

incorrect Stage X layout or values in any test, -0.5 ;

different Stage X layout or values in any test, -0.5 ;

another different Stage X layout or values in any test, -0.5 ;

X can be: 1, 2, 3

Academic Honesty is taken seriously!

- Sign the Authorship Declaration, and don’t change the content.
- Write your own code, don’t “borrow” from any person or non-person identity. Also don’t let other person to “borrow” your code.
- When use lecture codes: give reference
- If use external code: give reference

Assignment 1: testing

COMPILING & TESTING IN Ed in your computer

Compiling

Using redirection when running/testing your program:

./myass1 < test0.txt > test0-myout.txt

Your program's output must be the same as the expected, ie. the command

diff test0-myout.txt test0-out.txt

must give empty output (that is, no difference).

Remember that your code might work well on the 3 supplied data sets, but fail on some other…

NOTE: It's important to check with Gradescope

Make sure to submit to Gradescope regularly, and carefully read the verification report after submitting.

Example of using diff

```
[user@sahara ~]$ ./myass1 < test1.txt > test1-myout.txt

[user@sahara ~]$ diff test1-myout.txt test1-out.txt
4,7c4,7
< read 5 candidates and 8 votes
< voter 7 preference...
<     rank  1: Allyx
<     rank  2: Chenge
---
> read 5 candidates and 7 votes
> voter 7 preferences...
>     rank  1: Cheng
>     rank  2: Allyx
13a14,77
> round 1...
>     Allyx           :  2 votes,  28.6%
>     Breyn           :  1 votes,  14.3%
```

not matched

line starting with < is from the left file (ie. our output)
here: our line is not identical to the expected

line starting with > is from the right file (ie. expected output)
here: we missed 1 blank line in our output

not found

NO corresponding lines in the left (our output)

ASS1 Q&A:

A2 Labs or 7.12, 7.14-7.16 + exercises from lec06.pdf

7.12 (palindrome, similar to but simpler than, Exercise 3 below),
7.14 (atoi),
7.15 (anagram),
7.16 (word frequencies)

Exercise 1 Write a function `is_subsequence(char *s1, char *s2)` that returns 1 if the characters in `s1` appear within `s2` in the same order as they appear in `s1`. For example, `is_subsequence("bee", "abbreviate")` should be 1, whereas `is_subsequence("bee", "acerbate")` should be 0.

Exercise 2 Ditto arguments, but determining whether every occurrence of a character in `s1` also appears in `s2`, and 0 otherwise. For example, ", `is_subset("bee"rebel")` should be 1, whereas `is_subset("bee", "brake")` should be 0.

Exercise 3 Write a function `is_anagram(char *s1, char *s2)` that returns 1 if the two strings contain the same letters, possibly in a different order, and 0 otherwise, ignoring whitespace characters, and ignoring case. For example, `is_anagram("Algorithms", "Glamor Hits")` should return 1.

Exercise 4 Write a function `next_perm(char *s)` that rearranges the characters in a string argument and generates the lexicographically next permutation of the same letters. For example, if the string `s` is initially `"51432"`, then when the function returns s should be `"52134"`.

Exercise 5 If the two strings are of length `n` (and, if there are two, `m`), what is the asymptotic performance of your answers to Exercises 1–4?

Additional Slides

char type, reading the input character-by-character

```
char c; // c has integer value from –128 to +127 inclusively
c= 'A';
printf("Character %c has int value of %d\n", c, c);
```

Equivalent procedure for reading the entire input, character-by-character:

```
int c;          //  can also be  char c;

while ( (c= getchar() )!=EOF ) {

     /*proccess  c  */

}

//  Note:  EOF  is a pre-defined constant for
"End Of File"
```

```
char c; // c  must be char for using in  scanf

while ( (scanf("%c",&c)==1 )  {

     /*proccess  c  */

}
```

#include <ctype.h>

```
c= getchar();
```

How to check if `c` is a lower-case or upper-case letter, or both?

```
if ( ? )
        printf ("%c is a lower-case letter\n", c);
```

```
if ( ? )
        printf ("%c is an upper-case letter\n", c);
```

```
if ( ? )
        printf ("%c is a letter\n", c);
```

In doubt on how to use a function, say `isalpha`?
Google it, or, *quicker*, in the Terminal type "`man isalpha`".
(not applied to `grok`'s terminal, should be OK in minGW)

#include <ctype.h>

```
int c;
c= getchar();
if ( isalpha(c) )
        printf ("%c is a letter\n", c);
```

Some useful functions in <ctype.h>

- isalpha(c)
- isdigit(c)
- isalnum(c)
- toupper(c)
- tolower(c)

In doubt on how to use a function, say `isalpha`?
Google it, or, *quicker*, in the Terminal type "`man isalpha`".
(not applied to `grok`'s terminal, should be OK in minGW)

isspace(c) : returns 1 if c is an (invisible) whitespace
such as ' ', '\t', '\n'

a string is an array of chars, but terminated with '\0'

```
#define MAX_N 5
```

```
char A[MAX_N];
int n= 0;

A[0]= '1';
A[1]= '2';
A[2]= '3';
n= 3;
```

```
char s[MAX_N];


s[0]= '1';
s[1]= '2';
s[2]= '3';
s[3]= '\0';
```

A is an array of maximal 5 characters

S can be viewed as a string that of at most 4 characters

Ex 7.16 – The Task

Use the program of Figures 7.13 and 7.14 of the textbook (`words.c` and `getword.c` on Page 4 of lec06.pdf).

Design and implement a program that reads text from `stdin`, and writes a list of the distinct words that appear, together with their frequencies.

First step:

Make sure you understand the task, that you can imagine what's the input and output.

Case Study & Ex 7.16 – Understanding The Task

Design and implement a program that reads text from stdin, and writes a list of the distinct words that appear, together with their frequencies.

Sample texts:

```
A cat in a hat!
+-abc 10e12 e 1abc #e#abc.abcdefghijklm=xyz
```

Input = ?

- How to get the input text?

Output= ?

- How to store output, which data structure?
- And how to produce output?

Assumptions/limits:

- What's a *word*?
- Other assumptions?

Understanding getword

```
int get_word(word_t W, int limit) {
    int c;
    while ((c=getchar())!=EOF && !isalnum(c)) { //  skip non alphanumerics
    }
    if (c==EOF) {
        return EOF;
    }

    *W = c;                 // ok, first character of next word has been found
    W += 1;
    limit -= 1;
    while (limit>0 && (c=getchar())!=EOF && isalnum(c)) {
        *W = c;             // another character to be stored
        W += 1;
        limit -= 1;
    }
    if (strchr(TERM_PUNCT, c) && (limit>0)) {// if next character trailingpunctuation
        *W = c;
        W += 1;
        limit -= 1;
    }

    *W = '\0';              // now close off the string
    return WORD_FND;
}
```