COMP10002 Workshop Week 8

	ASS1: **Progress?** Marking rubric – some specific topics, Q&A
	BREAK-OUT ROOMS • assignments 1 • Q&A in the 2nd hour MAIN ROOM • string searching & BMH • discuss 7.16 • Q&A on other exercises
LAB	Finish ass1 during this workshop or today! Submissions close at **6pm on Friday 17 September**.  Done ass1? Implement exercises from lec06 and/or 7.16, 7.14 7.15
LMS	**Workshops**: • ***Your most important goal this week is to complete Assignment 1.*** • Make sure you ***submit several times through the week***;
	• **Quiz 2 in Week 9.** Thursday 2:15pm to 3:00pm.

Your assignment1:

A- All done

B- finished stages 1+2

C- finished stage 1

D- none of the above

Marking Rubric: The importance of Style & Structure

Stage	Presentation	Structure	Execution	max mark of stage	max accumulated mark
1	+6	+3	+3	+12	12
2	+1	+1	+2	+4	16
3	+1	+1	+2	+4	20
all	+8	+5	+7	+20	

Failed code could even get 13

Absolutely correct program could get only 7

Correct and good Stage 1 alone could get 12

ASS2 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, -

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

- absence of function prototypes, -0.5;
- bad choice for function names, -0.5;

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

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

make sure that they are consistent with the skeleton code

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

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

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

- 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

ASS2 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`, or nesting loops
don't make the marker wonder too much to understand your code!

- unnecessary duplication/copying of data, -0.5;

Think carefully before you copy an array!

ASS2 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 read the submission report (compiler messages are at the beginning of the report)

- incorrect Stage 1 layout or values error in any test, -0.5;
- different Stage 1 layout or values error in any test, -0.5;
- inco
- di
- in
- di

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

Compiler warnings: -2.0

overly complex algorithmic approach; -1.5)

Examples of overly complicated

sort the data when not actually required

too many levels of nested loops/if

duplicate code segments: turn segment into a function

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

duplicate code segments with boring repetitive fragment

```
if (month==JAN) {
        printf("January: %d celebrate birth day\n", num);
} else if (month==FEB) {
        printf("February: %d celebrate birth day\n", num);
} else if (month==MAR) {
        printf("March: %d celebrate birth day\n", num);
} else if (month==APRIL) {
```

→

ASS2 Q&A:

Decide: to stay in the main room or to dfo ass2 in break-out rooms first, before the assignment Q&A time

String Searching

Input:

A (normally long) text T[0..n-1]. Example: T= “SHE SELLS SEA SHELLS”, with n=20

A (normally short) text pattern P[0..m-1]. Example: P=“HELL”, m=4.

Output:

index i such that T[i..i+m-1]=P[0..m-1], or NOTFOUND

Algorithms:

- Naïve: brute force, complexity O(nm)
  - max number of character comparisons = (n-m+1)*m
  - Note: strcmp also compares strings character-by-character
- BMH: also O(mn) but practically fast
- kmp: 2n+m iteration max → O(n+m)
-

Understand the BMH algorithm

a	b	a	b	y	a	y	b		a	a	b	a	b	c	a

a	b	a	b	c

SHIFT m because y not in P

First align the pattern P with the text T.

Loop:

Start from the right-hand end of the pattern, and work to the left, P[i] with the corresponding character in T

if mismatched:

shift pattern P to the right

else

FOUND

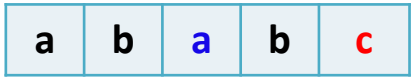
return NOTFOUND

Understand the BMH algorithm

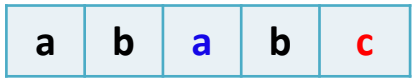
SHIFT m because y not in P

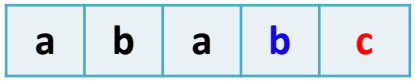
1 comp until mismatch
no matter where mismatch happens, the shift is totally decided by the rightmost examined char of T **y**

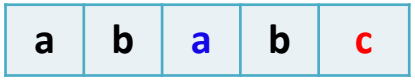
Shift the pattern P until having the first match of character on P with that rightmost **y**
(here, no match found)



SHIFT 2 = distance from last **a** to **c**







Horspool's Algorithm Review

The task: Seaching for a pattern `P` (such as "HELL" that has length m=5) in a text `T` (such as "SHE SELLS SEA SHELLS", having length n=20).

The Algorithm:

need to do a pre-processing of the pattern before performing the search

normally, $|P| \ll |T|$, this step doesn't affect the overall complexity

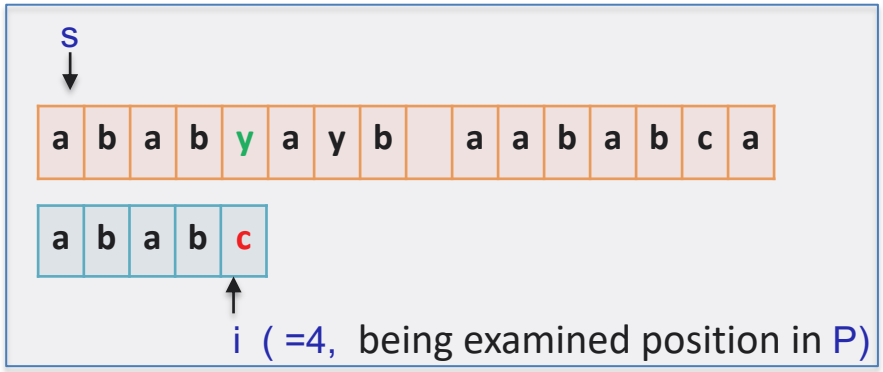
BMH: pre-processing

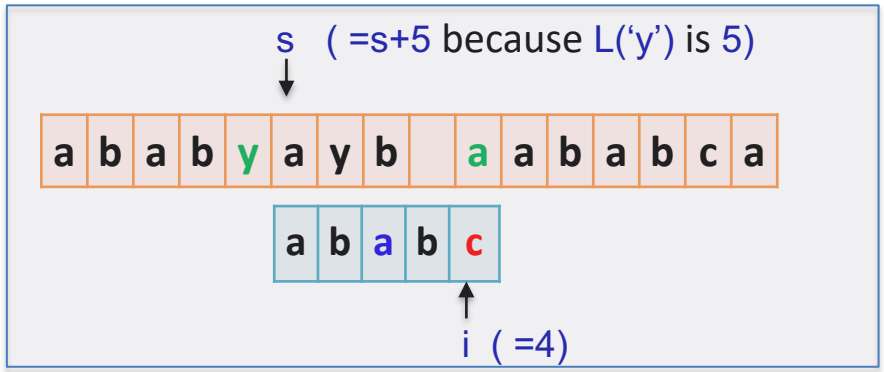
***Pre-processing**:* build `L[x]` for every possible character `x`, ie. for all x from the alphabet (in the lecture, the alphabet has σ symbols), by:

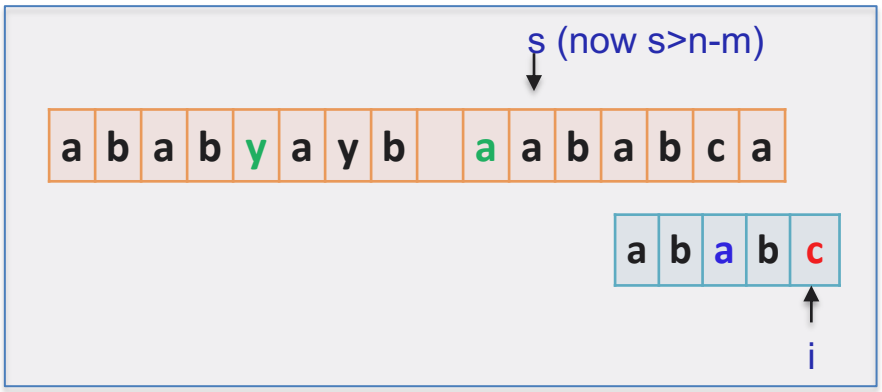
1. first, set `L[x]= m` for all `x`, then
2. for each character `x` in `P`, *except for the last one*: `L(x)` = distance from the last appearance of `x` to the end of `P`

for $v \leftarrow 0$ **to** $\sigma - 1$
    $L[v] \leftarrow m$
for $i \leftarrow 0$ **to** $m - 2$
    $L[P[i]] = m - i - 1$

BMH - searching








```
s=0;      // current start of P in T
i= m-1;   // current position in P
c= T[s+m-1]; // the pilot character

while (s+i not passing the end of T){
  if (mismatched at P[i]) {
    s = s + L[c];
    i = m-1;
    c= T[s+m-1];
  } else {
      if (i==0)
        return s;
      else
        i--;
  }
}
return NOTFOUND;
```

$s, i \leftarrow 0, m-1$

while $s \leq n - m$
  if $T[s+i] \neq P[i]$
    $s, i \leftarrow s + L[T[s+m-1]], m-1$
  else if $i = 0$
    return s
  else
    $i \leftarrow i - 1$

return *not_found*

BMP Algorithm: Exercises for your Brain

Problem: Use BMP algorithm to search for the pattern `GORE` in the string `ALGORITHM`.

Problem: How many character comparisons will be made by BMP algorithm in searching for each of the following patterns it the binary text of one million zeros?

(a) 01001 (b) 00010 (c) 01111

Problem: BMP Worst-Case Time Complextity: In BMP, what does a worst-case example look like (give an example)?

Labs? Other exercises from lec06.pdf

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?

Ex 7.16 and others

Combine Alistair's `getword.c` and `words.c` into one `.c` file, then change it to meet the requirement of Ex 7.16.

Case Study & 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?

Case Study & Ex 7.16 – Alistair's getword

```
int getword(char W[], int limit) {
  int c, len=0;
  /* first, skip over any non alphabetics */
  while ((c=getchar()) != EOF && !isalpha(c)) {
      /* 12+34 aWord ?-? is the first word */
  }
  if (c==EOF)  return EOF;

  /* ok, first character of next word has been found */
  W[len++] = c;
  while (len<limit && (c=getchar())!=EOF && isalpha(c)) {
    /* 12+34 aWord ?-? is the first word */
    W[len++] = c;
  }

  /* now close off the string */
  W[len] = '\0' ;   // W is the string aWord
  return 0;
}
```

Alistair's words.c

```
#define MAXCHARS 10
  /* Max chars per word */
#define MAXWORDS 1000
  /* Max distinct words */


typedef char word_t
            [MAXCHARS+1];
  /* word_t word; now is
     equivalent to
  char word [MAXCHARS+1];
  */

int getword(word_t W,
            int limit);

#include "getword.c"


int
main(int argc,
     char *argv[]) {
  word_t one_word, all_words[MAXWORDS];
  int numdistinct=0, totwords=0, i, found;

  while (getword(one_word, MAXCHARS) != EOF) {
    totwords = totwords+1;
    /* linear search in array of previous words...*/
    found = 0;
    for (i=0; i<numdistinct && !found; i++) {
      found = (strcmp(one_word, all_words[i]) == 0);
    }
    if (!found && numdistinct<MAXWORDS) {
      strcpy(all_words[numdistinct], one_word);
      numdistinct += 1;
    }
    /* NB - program silently discards words after
       MAXWORDS distinct ones have been found */
  }

  printf("%d words read\n", totwords);
  for (i=0; i<numdistinct; i++) {
    printf("word #%d is \"%s\"\n", i, all_words[i]);
  }
  return 0;
}
```

Some small, esay, but important topics

typedef

argc, argv

Program arguments

Write a program sum that accept two numbers and print out their sum. Example of execution:

```
$ ./sum 12 5
12.00 + 5.00 = 17.00
```

```
?: int main(int argc, char *argv[])
```

Program arguments: notes

Must check:

is the number of arguments as expected, and

when possible, is each argument valid.

Example: a program that accepts two positive numbers and print out their sum.

```
int main(int argc, char *argv[]) {
   double a, b;
   if (    argc != 3
        || (a=atof(argv[1]))<=0
        || (b=atof(argv[2]))<=0  ) {
      fprintf(stderr, "Usage: %s a b  where"
                 " a>0 and b>0\n",argv[0]);
       exit(EXIT_FAILURE);
   }
   printf("%.2f + %.2f = %.2f\n", a, b, a+b);
   return 0;
}
```