COMP10002 Workshop Week 7

1	char and strings
2	discuss 7.12, 7.14, 7.15, 7.16: what's required?
LAB	Implement at least two from 7.12 7.14 7.15, 7.16
ASS1	Progress so far? Q&A
LMS	**Workshop**: • Discuss the requirements of Exercises 7.12 (palindromes), 7.14 (atoi), 7.15 (anagrams), and 7.16 (frequency counting for words); then • Implement solutions to at least two of them (for some of them you will need to write simple main programs in order to test your functions; for some of them you can start with the insertionsort.c program and then alter it); plus • There are also exercises in the lec06.pdf slides that you shouldn't ignore. • **In the second half of the workshop, work on Assignment 1.**

Notes from lecture: lec06.pdf

- 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 nextword 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

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 ASCII value of %d\n", c, c);
```

```
char c;       // c must be char for using in scanf
while ( (scanf("%c",&c)==1 )  {
    /*proccess c */
}
```

```
int c;          // can also be char c;
while ( (c= getchar() )!=EOF ) {
    /*proccess c */
}
// Note: EOF is a pre-defined constant for "End Of File"
```

#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, use `jEdit`/`minGW`)

#include <ctype.h>

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

Some useful functions in <ctype.h>

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

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

strings: strings are arrays

```
#define MAX_N 5
```

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

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

strings are arrays of chars

```
#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 + 1];



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

Strings: array notation & pointer notation

```
char s[MAX+1] = “12345”;
int n= strlen(s);
int i;

for (i=0; i<n; i++) {
  printf(“%c”, s[i]);
}
```

Array Notation

```
char s[MAX+1] = “12345”;

char *p;

for (p=s; *p; p++){
  printf(“%c”, *p);
}
```

Pointer Notation

#include <string.h>

```
#define MAX_STR_LEN 100
char s1[MAX_STR_LEN+1]= "123456789ABCDEF", s2;
char s8[9]= "0123";
```

```
printf ("String \"%s\" contains %d characters\n,
                  s1,          strlen(s1) );
```

```
strcpy(s2, s1);
if ( strcmp(s1, s2) == 0 )
    printf ("\"%s\" and \"%s\" are identical\n", s1, s2);
printf("strcmp(\"%s\", \"%s\"%s) = %d\n",
                  s8,      s1,  strcmp(s8,s1));
```

BE CAREFUL:

```
strcpy(s8, s1);
```

is errorneous!

Ex 7.12 (palindrome)

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.

Preparation: If you (and you should) use jEdit/Terminal: download `string_examples_skel.c` from `github.com/anhvir/c102`

Extra home work: Change the function to satisfy the broader definition of palindrome.

Ex 7.12: Palindrome

Ex 7.14: atoi how-to?

Write a function

```
int atoi(char *)
```

that converts a character string into  er value.

Strings - a quiz

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

Which of the following fragments give the same result as `n=atoi(s)`:

A.
```
for (; isdigit(*s); s++)
    n= n*10 + (*s);
```

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

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

D. none of the above

Ex 7.15: how-to?

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.

HOW-TO approach the task?

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;
}
```

Ass1: The Marking Rubric: pay attention, get higher marks

LAB TIME

Assignment 1 Q&A

Do Assignment 1 here in the main room, or do it in the break-out rooms, but please remember:

- don’t show your code to your friends
- general discussion (ie. on the meaning of some parts in the spec, or on approaches to solve a problem) is OK

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.

Implement

- 7.12 (medium),
- 7.14 (easy),
- 7.15 (a bit hard).

Another choice: group work - doing 7.12, 7.14 or some other exercises on board/paper

Labs:

1. Implement **Ex 7.12** (palindrome), **7.14** (atoi), **7.15** (anagram), **7.16**

Labs:

1. Implement **Ex 7.12** (palindrome),
2. **7.14** (atoi),
3. **7.15** (anagram, same as lec06.E3),
4. **7.16**.

ASS2 Q&A

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?

Additional Slides

Strings

```
char s1[10]= “Hello”;
char *s2=“1234”;
```

Which of the following statements are OK:

```
1. s1++;
2. s2++;
3. s1= s2;
4. s2= s1 + 3;
```

Strings

```
typedef char word_t [11];
char *s="1234 abc9", *p= s;
word_t w, *q= w;
```

What the following fragments do?

1. `while (*p) *q++= *p++;`
2. `while (*q++= *p++);`
3. `while (isalpha(*p)) *q++= *p++;`
4.
```
while (!isalpha(*p)) p++;
  while (isalpha(*p)) *q++= *p++;
  *q= '\0';
```

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;
}
```