

COMP20005 Workshop Week 11

1

Assignment2 Q&A

2

Working on your assignment 2, OR (if you are 100% sure that you have submitted a perfect work for assignment 2):

2a

Optional Topics **for those who intend to take another algorithmic subject:**

- Strings: array notation and pointer notations
- Some string exercises: 7.13, 7.14, 7.12, 7.15

2b

OR, if not:

Do more array/struct exercises, AND/OR:

Do exercises from chapters 9:

Ex. 9.03 (simulation with Poker Hands)

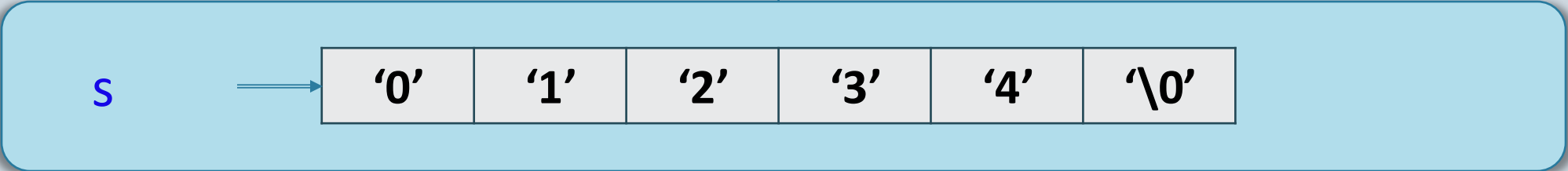
(**9.6, 9.7, 9.8, 9.9, 9.11**)

Assignment 2:

- learnt from errors, if any, in your assignment 1
- presentation: mark deduction even with one appearance of mistakes
- structure:
 - use struct and typedef
 - avoid duplication
 - avoid code that is overly complicated due to excessive nesting of control structures (loops, and if statements)
 - ...



Strings: array notation & pointer notation

#define MAX 5	
<pre>char s[MAX+1] = "01234";</pre>	<pre>char *s = "01234";</pre>
 <p>The diagram illustrates a memory layout for a string. A light blue rounded rectangle contains a pointer variable 's' on the left. An arrow points from 's' to a horizontal array of six boxes. The boxes contain the characters '0', '1', '2', '3', '4', and '\0' (the null terminator) in order from left to right.</p>	
<pre>int n= strlen(s); int i; for (i=0; i<n; i++) { printf("%c", s[i]); }</pre>	<pre>char *p; for (p=s; *p != '\0'; p++){ printf("%c", *p); }</pre>
<i>Array Notation</i>	<i>Pointer Notation</i>

#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): returns the length of s

strcpy(s2, s1): copies s1 to s2 (s2 must have enough space!)

strcmp(s1, s2): compare s1 and s2 lexicographically, and returns:

- **0:** if s1 and s2 are identical.
- **A negative value:** if s1 comes before s2 in lexicographical order (e.g., "apple" before "banana").
- **A positive value:** if s1 comes after s2 in lexicographical order (e.g., "zebra" after "yak").

Ex 7.13, 7.14: how-to

7.13: Write a function that calculates the length of a string, excluding the terminating null byte ('\0').

```
int my_strlen(char *str)
```

You cannot use `<string.h>` in your `my_strlen` implementation.

7.14: Write a function

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

that converts a character string into an integer value.

? How to convert a `char` digit like '3' to the numeric value (`int 3` in this case)

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: 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.

Strategy: Simulation

Simulate what done manually
Simulate physical processes

E 9.03: How to apply simulation here?

Write a program that deals four random five-card poker hands from a standard 52-card deck. You need to implement a suitable "shuffling" mechanism, and ensure that the same card does not get dealt twice. For example:

```
player 1: 3-S, Ac-C, Qu-D, 4-H, Qu-H  
player 2: 10-C, 2-H, 5-H, 10-H, Ki-H  
player 3: 2-C, 6-D, 10-D, Ki-D, 9-H  
player 4: 8-S, 9-S, 10-S, Qu-S, 4-D
```

Then modify your program to allow you to estimate the probability that a player in a four-person poker game obtains a simple pair (two cards with the same face value in different suits) in their initial hand. Compute your estimate using 40,000 hands dealt from 10,000 shuffled decks.

How about three of a kind (three cards of the same face value)?

And a full house (three of a kind plus a pair with the other two cards)?

e9.3: let's simulate a game

```
#define FACES    13
#define SUITS     4
#define CARDS (FACES*SUITS) /* number of cards */

#define PLAYERS   4
#define CARDSINHAND 5

const char *faces[FACES] = {"Ac", "2", "3", "4", "5", "6",
                             "7", "8", "9", "10", "Ja", "Qu", "Ki"};
const char suits[SUITS] = {'S', 'C', 'D', 'H'};

typedef struct {
    int face, suit;    // index to the above arrays
} card_t;

card_t players[PLAYERS][CARDSINHAND];
card_t deck[CARDS];
```

How to give each of the players random CARDINHAND cards?

e9.3: let's simulate a game

```
const char *faces[FACES] = {"Ac", "2", "3", "4", "5", "6",  
                             "7", "8", "9", "10", "Ja", "Qu", "Ki"};  
const char suits[SUITS] = {'S', 'C', 'D', 'H'};
```

```
typedef struct {  
    int face, suit;    // index to the above arrays  
} card_t;
```

```
card_t players[PLAYERS][CARDSINHAND];  
card_t deck[CARDS];
```

How to give each of the players random CARDINHAND cards?

- we start with a “brand new” deck that has some pre-defined order (for example from Ac to Ki, from S to H (easy, right?))
- we “shuffle” the deck by re-arranging its element in a *random* order (**how?**)
- then, we deliver the cards one-by-one to the players in the round-robin manner (easy, right?)

Assignment 2:

- learnt from errors, if any, in your assignment 1
- presentation: mark deduction even with one appearance of mistakes
- structure:
 - use struct and typedef
 - avoid duplication
 - ...



