COMP20005 Workshop Week 12

1	Problem Solving Strategies Discuss and then implement a solution to Exercise 9.7. Skimming: 9.3, 9.11
2	Implement at least one of the exercises 9.3, **9.7, 9.6, 9.8, 9.9, 9.10, 9.11, 9.12, 9.x1** Write a fun program to end the semester! **OR** **Do the missed exercises in C7 (arrays) and C8 (structs)**

Strategy: generate-and-test

Other names: brute-force search, exhaustive search.

The task: searching for a solution

Principle:

- *systematically enumerate all possible candidates for the solution, and*
- *check the candidates one-by-one until a solution found*

Ex 9.7

Devise a mechanism for determining the set of adjacent elements in an array of n elements that has the largest sum.

i :	0	1	2	3	4	5	6	7
A[i]:	2	1	3	−4	−5	2	5	3

Write a function named `max_subarray` for that.

What is the problem? Give a function prototype.

```
??? max_subarray( ??? )
```

How to apply generate-and-test?

e9.7: solution1 (brute force)

```
2 1 3 -4 -5 2 6 1 -8 9 2 3 1 5 -7 1 -2
-----       ------------------    --
```

```
int max_subarray(int A[], int n, int *left, int *right) {
  // ANY pair (i,j) (i<=j) can be a potential answer.



}
```

Strategy: Simulation

Simulate what done manually (especially in games)

Simulate physical processes

e9.7: solution2 – simulation what's done manually

```
 2  1  3 -4 -5  2  6  1 -8  9  2  3  1  5 -7  1 -2
 -----
```

What would we do when solving this problem manually?

```
0                5                10                15

2  1  3 -4 -5  2  6  1 -8  9  2  3  1  5 -7  1 -2
-----          -----     ----------     -        ???
```

e9.7: solution2

i	:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A[i]:		2	1	3	-4	-5	2	6	1	-8	9	2	3	1	5	-7	1	-2
		-																

```
start with maxsum= 0 (solution= empty subarray)
Checking: which pair (i,j) gives better maxsum?

Start with i=0, what's the best j?   j=
Best soln with i=0:  (i=0, j=2, maxsum= 6)
```

e9.7: solution2

```
i   :  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
A[i]:  2  1  3 -4 -5  2  6  1 -8  9  2  3  1  5 -7  1 -2
       -------        -------------------------     --
```

```
start with maxsum= 0, *left=0, *right=-1 (empty subarray)
Checking: which pair (i,j) gives better maxsum?

i= 0; j=-1; sum=0;  // Start with i=0
while (i<n && j<n) {
   // advance j with j++ and add A[j] to sum
   if (sum>maxsum) {...} // update *left, *right, maxsum
   if (sum<0) {...} // update i= ?
}
```

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)?

Using random numbers in simulation

Problem: some simulation (like gaming) requires to use random numbers

Solution: use pseudo-random number generators

In C (stdlib):

- rand(): generates an unsigned pseudo-random values
- srand(int): change the seed for generating random values

Typical use:

Examples:

use grok 9.3

```
#include <stdlib.h>
#include <time.h>

srand(time(NULL));

// use: MIN+rand()%(MAX-MIN+1)
//      to generate random int from MIN to MAX
```

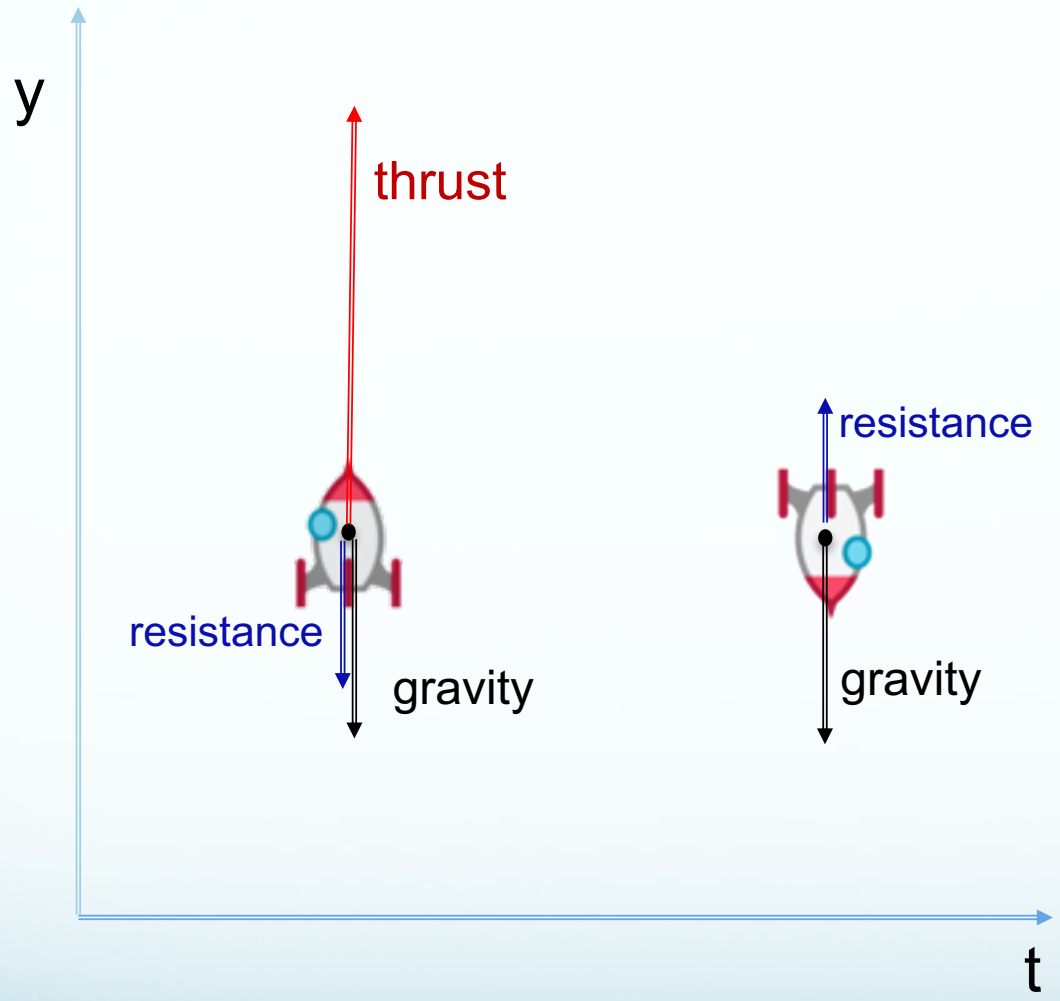
e9.3: let's simulate a game

```
#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’};


int players[PLAYERS][CARDSINHAND]; ???
int deck[CARDS];                   ??? should a card be a struct instead?
```

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 0 to 51)
- 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?)

Ex 9.11: See grok for 9.11 exercise and beautiful output



High School Physics :-)

F= sum of all forces
m= total weight

loop:
  a= F/m
  v= v0 + a Δt
  y= y0 + v Δt

  t= t0 + Δt
  m= ???

--- for this task ---
m= W_{rocket} + W_{fuel}
W_{rocket} = 10
W_{fuel}= 8 – t x rate
rate= 0.8 (kg/s)
You can choose Δt as a parameter

Our last hour:

- Write a fun program to end the semester OR review arrays & structs!
- Questions on possible enrolment in ADS (COMP20003) in semester 2

fun programs?	arrays & structs
Implement at least one of the exercises **9.x1, 9.3, 9.7, 9.11, 9.6, 9.8, 9.9** Recommended: 9.11 if you love high school physics 9.3 if you like playing poker 9.8: more with array – k-th largest value	arrays: • 7.04 – 7.11 • 3.06-X (in C7) • 7.x1 – 7.x3 structs: • 8.02 – 8.05 arrays of structs: • 8.06 – 8.09, 8.x1 9.06 more with numerical method: Newton-Raphson method

Thank You
&
Good Luck