# COMP20005 Workshop Week 10

**1**

## Discussion 1: Root finding

- Exercise 9.5

## Discussion 2: struct

- Ex. 8.1
- Ex. 8.2-8.3

**2**

Assignment 2:

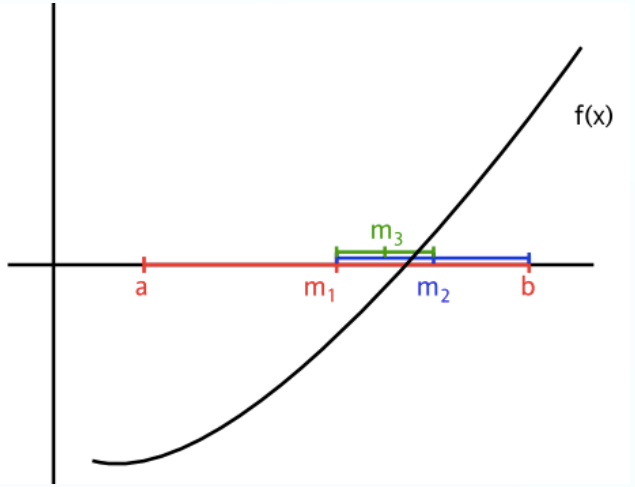
- Read specs & watched the assignment lecture?
- Q&A
- Working on assignments

**LAB**

# Discussion 1: Roof Finding for f(x)=0: bisection method

`mid= (a+b)/2` `(f(a)f(b)<0)`

Transition to next iteration: a=mid or b=mid depending on `af(mid)<0` or `bf(mid)<0`



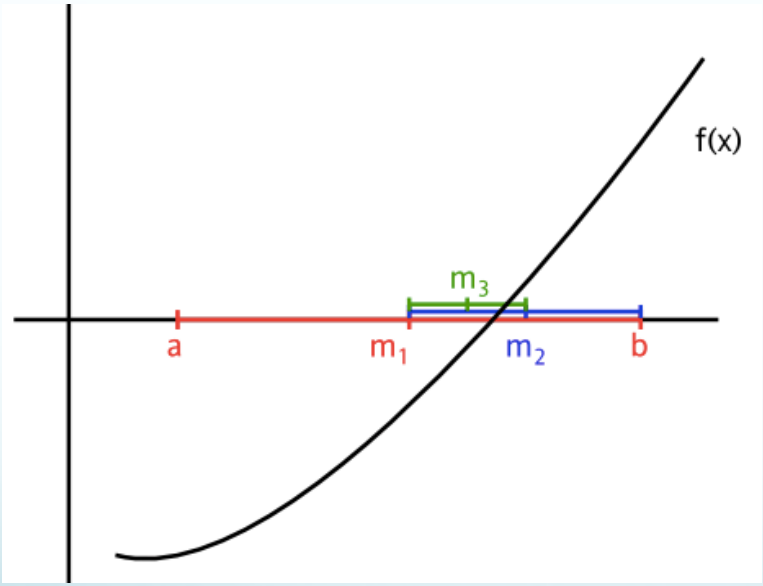

**Ex. 9.5**: The square root of *2* is the of equation $f(x) = x^2 - 2$. *Using bisection method*
start with a= 1 and b= 3. Stop when the length of the interval is 0.1 or less.

| a | b | mid | f(a) | f(b) | f(mid) | b-a |
|---|---|---|---|---|---|---|
| 1.00 | 3.00 | 2.00 | – | + | + | > 0.1 |
| 1.00 | 2.00 | 1.50 | | | + | |

Do it manually and check your answer with me after 3 min

# Check: Roof Finding for f(x)=0: bisection method

`mid= (a+b)/2` `(f(a)f(b)<0)`
Transition to next iteration: a=mid or b=mid depending on `f(mid)<0` or `f(mid)<0`




**Ex. 9.5**: The square root of *2* is the of equation $f(x) = x^2 - 2$. *Using bisection method*
start with a= 1 and b= 3. Stop when the length of the interval is 0.1 or less.

| a | b | mid | f(a) | f(b) | f(mid) | b-a |
|---|---|---|---|---|---|---|
| 1.00 | 3.00 | 2.00 | – | + | + | > 0.1 |
| 1.00 | 2.00 | 1.50 | | | + | |
| 1.00 | 1.50 | 1.25 | | | – | |
| 1.25 | 1.50 | 1.375 | | | – | |
| 1.375 | 1.50 | 1.44 | | | + | > 0.1 |
| 1.375 | 1.44 | 1.41 | | | – | < 0.1 |

# Structs

Arrays: for combining a collection of same-type data into a same roof

Structs: for combining different aspects (normally of different types) of the same object into a same roof

# struct How to use struct

```
typedef struct {
    int dd, mm, yyyy;
} date_t;

date_t dob={28,02,2001}, my_date;

printf("dob= %02d/%02d/%04d\n", dob.dd, dob.mm, dob.yyyy);

// change dob to {31,12,2001}
dob.dd= 31;
dob.mm= 12;


my_date= dob;
```

dob

| dd | mm | yyyy |
|---|---|---|
| 28 | 2 | 2001 |

The dot . operator is used to access a *component of a* struct

Assignment of whole struct is OK!
(*but comparison is* ***not***)

# pointers to struct

```
typedef struct {
    int dd, mm, yyyy;
} date_t;

date_t dob={28,02,2001};

date_t *p;
p = &dob;   // *p == dob   (*p).dd
// change dob to {31,12,2002}
(*p).dd= 31;        // better: p->dd= 31;
(*p).mm= 12;        // better: p->mm= 12;
p->yyyy= 2002;      //                  same as (*p).yyyy = 2022;
```

| | dd | mm | yyyy |
|---|---|---|---|
| dob | 28 | 2 | 2001 |

p → &dob

The arrow `->` operator is used to access a *component of* the struct that a *pointer* points to

`pointer->` is a convenient shorthand for `(*pointer).`

```
define NAMESTRLEN 39
#define MAXSUBJECTS 8
typedef char namestr_t[NAMELEN+1];

typedef struct {
   namestr_t first, others, family;
} fullname_t;

typedef struct {
   int dd, mm, yyyy;
} date_t;

typedef struct {
  int subjectcode, status, finalmark;  // ?
  date_t enrolled;                     // ?
} subject_t;

typedef struct {
  fullname_t name;
  date_t dob, datecommenced;
  int id, status, salary;
} staff_t;

typedef struct {
  int id;                 // ?
  fullname_t name;        //
  date_t dob;             //
  int nsubjects;          //
  subject_t subjects[MAXSUBJECTS];
} student_t;

staff_t alice;
student_t bob;
staff_t allstaff[1000];
student_t allstudents[10000];
```

## Discussion 2: struct Exercise 8.1:

With the declaration on the LHS, how many bytes each of the variables consume?

```
alice       :
bob         : 332
allstaffs   :
allstudents:
```

Discuss your solutions with friends, then compare with the expected solution (shown next).

```
define NAMESTRLEN 39
#define MAXSUBJECTS 8
typedef char namestr_t[NAMELEN+1];

typedef struct {
   namestr_t first, others, family;
} fullname_t;

typedef struct {
   int dd, mm, yyyy;
} date_t;

typedef struct {
  int subjectcode, status, finalmark;
  date_t enrolled;
} subject_t;                  // 24

typedef struct {
  fullname_t name;
  date_t dob, datecommenced;
  int id, status, salary;
} staff_t;

typedef struct {
  int id;                     // 4
  fullname_t name;            // 120
  date_t dob;                 // 12
  int nsubjects;              // 4
  subject_t subjects[MAXSUBJECTS];  // 8*24
} student_t;                  // 332

staff_t alice;
student_t bob, x;
staff_t allstaff[1000];
student_t allstudents[10000];
```

## Exercise 8.1:

```
alice       : 156
bob         : 332
allstaffs   : 14,600
allstudents: 332x10^4
```

Messages:

- a struct could be very large
- a struct can contain other structs and even array,
- but assignment like `x= bob` is still possible (when *array assignment is not allowed*)

# Function for input, version 1 (The Bad & The Ugly)

```
student_t read_student() {
  student_t s;
  scanf("%d", &(s.id));
  scanf("%s %s %s", s.name.first, ...);
  scanf("%d/%d/%d", &s.dob.dd, &s.dob.mm, &s.d
  ...
  return s;
}
```

```
typedef struct {
   namestr_t first,
       others, family;
} fullname_t;

typedef struct {
  int id;
  fullname_t name;
  date_t dob;
  ...
} student_t;
```

*What's bad?*

```
student_t stud= read_student(); // 332 bytes transferred
```

(recall that a `student_t` variable consumes 332 bytes)

# Function for input, version 2 (The Good)

```
typedef struct {
  int id;
  fullname_t name;
  date_t dob;
  ...
} student_t;
```

```
void read_student(student_t *p) {
  scanf("%d", &(p->id));
  scanf("%s %s %s", p->name.first, p->name.other, p->name.family);
  scanf("%d/%d/%d", &((p->dob).dd), &(p->dob.mm), &p->dob.yyyy);
  ...
}
```

*How to use read_stud? How good is this vesrsion?*

```
student_t stud;
read_stud(&stud)      // 8 bytes transferred instead of 132 !!!
```

# Structures: important rules

When a struct is large (say >16 bytes), and in general,

DON'T:

- use the struct as a function argument
- return the struct from a function

DO:

- use a *pointer to struct* as a function argument, for both input and output of a function
- not to return a struct

Reason: save memory, and save time for copying whole structs

# Discuss: Ex 8.02-8.03 for structs & arrays of structs

*8.2:* Define a structure vector_t that could be used to store points in two dimensions x and y (such as on a map).
Then write a function double distance(vector_t p1, vector_t p2) that returns the Euclidean distance between p1 and p1.

*8.3:* Suppose that a closed polygon is represented as a sequence of points in two dimensions. Give suitable declarations for a type poly_t in which it is assumed that no polygon contains more than 100 points.
Then write a function double perimeter(poly_t P) that returns the length of the perimeter of polygon P represented in your format.
Write a main function to test your perimeter function that scans in points from input until an EOF is read (or when scanf fails to read a point), that prints the perimeter of that polygon.
→ BETTER TO CHANGE TO double perimeter(poly_t *P)

# ass2: Q&A

# assignment 2: new items in rubric (section Structure)

duplicate code segments, -0.5;

main program too long or too complex, -1.0;

...

avoidance of structs, -2.0;

avoidance of typedefs, -2.0;

other structural issue (minor), -0.5;

other structural issue (major), -1.0;

# Assignment 2

start your `ass2.c` by using grok or:

- copying `ass2_skel.c` to `ass2.c`
- copying data and expected output files
- copying Makefile from grok

Then

- sign the declaration
- implementing  Stage 1 [*should be quick, if not done then do it now*]
- read, understand Stage 2 and implement it
- then, move on with stage 3

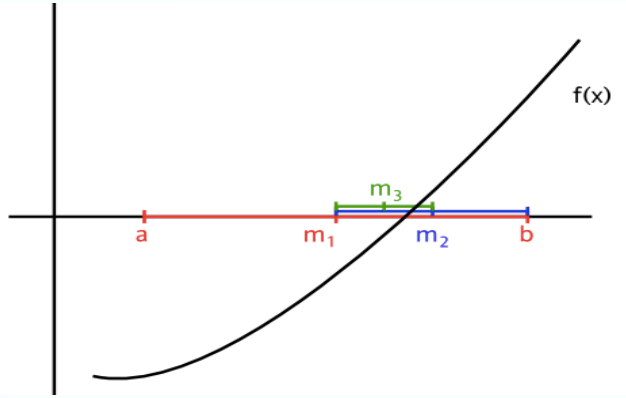
*ask questions, discuss with Anh and everybody, but do not show code to your friends*

submit test today (and see if the compiler complains)

# Other exercise required in LMS

Re-examine the cube_root() function on page 77 of the textbook, croot.c. What method does it use? Explore what happens if: (a) very large numbers are provided as input; (b) very small (close to zero) numbers are provided; and (c) CUBE_ITERATIONS is made larger or smaller.

# Review: Methods for Roof Finding for f(x)=0




Bracketing Methods `[a mid b]`

Bisection: `mid= (a+b)/2`

False position:

`mid= (a*f(b)-b*f(a))/(f(b)-f(a))`

Transition to next iteration: `a=mid` or `b=mid` depending on `b*f(mid)<0` or `a*f(mid)<0`

Methods that build series $x_1, x_2, ..., x_n, ...$

Secant:

$$x_{k+1} = (f(x_k))x_{k-1} - f(x_{k-1})x_k) / (f(x_k)-f(x_{k-1}))$$

Newton-Raphson:

$$x_{i+1} = x_i - f(x_i)/f'(x_i)$$

*Root finding*: see `numericB.pdf` for methods such as: *bisection,* , *false position*, *fix-point iteration*, *Newton-Raphson*, *secant*