COMP10002 Workshop Week 9

1	Struct, pointers to struct, arrays of struct. Discuss Ex1,2,3 of lec07 Discuss: approaches for 8.02,8.03
2	**Break-Out Rooms** • implement 8.02-8.03, 8.05, 8.07 • start looking at Exercises 4, 5, 6, and 7 in lec07.pdf For the 2:15PM quiz: You can leave early if you prefer
From LMS	**Workshops:** • Exercises 8.02, 8.03, 8.05, and 8.07. • Exercises 1, 2, and 3 in lec07.pdf • Then, if you still have time, start looking at Exercises 4, 5, 6, and 7 in lec07.pdf **Quiz 2.** This will take place at 2:15pm on Thursday 30 September. Quiz 2 will cover all topics through until the end of the lec06 slides and videos (end of Week 7). **Assignment 2:** Due 6pm Friday 15 October

Toward using multi-component objects…

Supposing that I need to keeps MST scores of students of this class together with names and ID. Supposing that each name has maximum 30 characters.

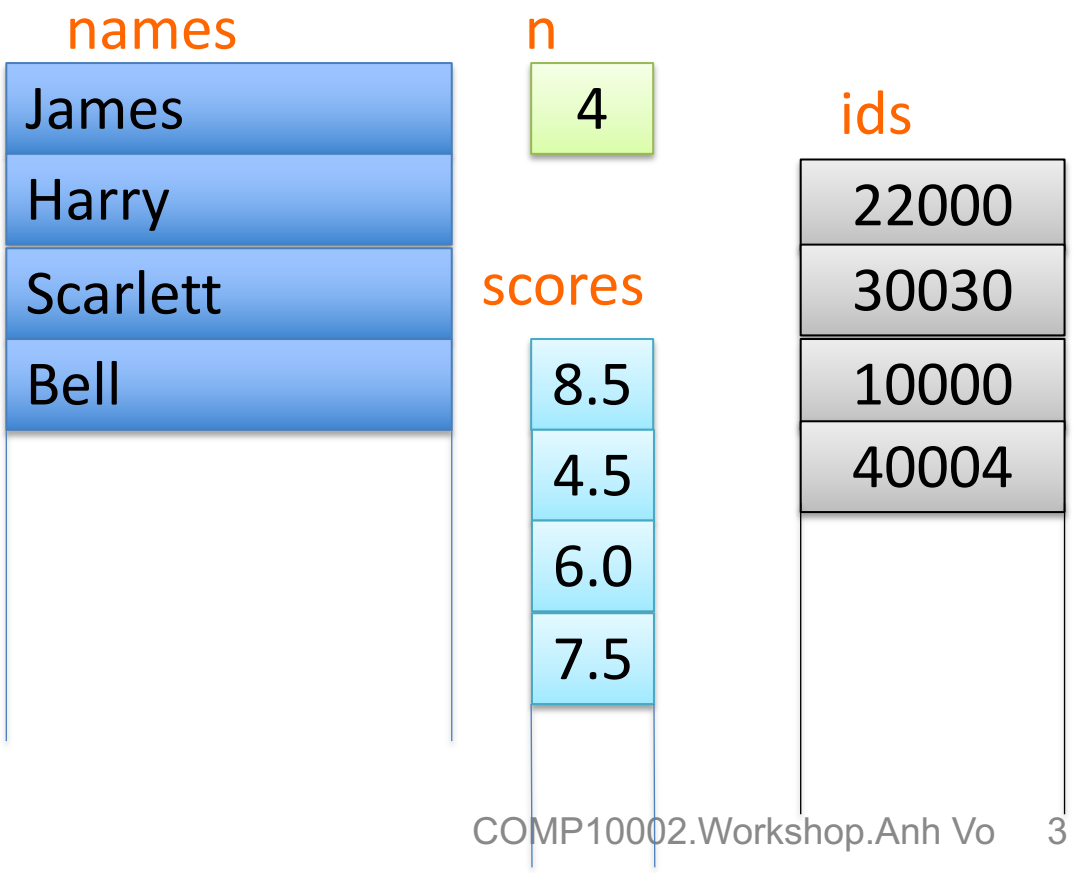
```
#define MAX_S 25
#define MAX_NAME 30
…

char names[MAX_S][MAX_NAME+1];
int ids[MAX_S];
float scores[MAX_S];
int n= 0;     // current number of students
```

Is that Good?

Toward using objects…

```
while (n<4) {
    scanf("%s %d %f ", names[n], &ids[n], &scores[n]);
}
printf("…", names[0], ids[0], scores[0]); //display the 1st stud
```



Hmmm…

Is that a *correct* design?

Is that a *good* design?

Let's define a compound data type…

```
typedef struct {
   char name [MAX_NAME+1];
   int id;
   float score;
} student_t;
```

```
student_t bob= {"Bob", 40004, 7.5};
```


```
printf("name= %s,   id= %d,  score= %f\n",
       bob.name, bob.id,  bob.score );
```

Toward using objects…

```
typedef struct {
    ...
} student_t;
student_t studs[MAX_N]= {...};
for (n=0; n<4;n++) ...      // now n= 4
```

name	ids	score
James	22000	8.5
Harry	30030	7.5
Scarlett	10000	9.0
Bell	40004	7.5

n

4

Memo: Processing Structures

Initialising `struct` just like initialising arrays:

```
student_t s1= { "Bob" , 1234 , 97.75 };
```

Processing `struct` by doing so with each component (like arrays):

```
scanf("%s %d %f", s1.name, &s1.id, &s1.score);
printf("name= %s, id= %d, score= %.1f\n",
        s1.name, s1.id, s1.score);
```

But, unlike arrays, we can:

- make assignment: `s1= s2;`
- and hence, `struct` can be the output of a function:

  ```
  student_t best_student(student_t s[], int n)
  ```

- but note: don't compare `struct`:

  `if (s1==s2)` makes nonsense

Memo: Pointers to Structures

```
typedef struct {
    char name[31];    // note that "name" is an array
    int id;
    float score;
} student_t;  double *  student_t *

student_t s, *ps;   // ps is a pointer to student_t
ps= &s;             // now *ps and s are equivalent
```

The following 2 lines are equivalent:

```
scanf("%s %d %f",   s.name,   &s.id,  &(*ps).score);
scanf("%s %d %f", ps->name, &ps->id,   &ps->score);
```

`ps->name` *is just a shorthand* for `(*ps).name`
an excellent shorthand!

Arrays of structs are popular & powerful. Example

- a list of student records is an array:

```
student_t class[MAX_S];
int n=0;
```

- selection-sort the class by name

```
void sort_by_name(student_t A[], int n) {
    if (n<=1) return;
    int i, imax= 0;
    // set imax= index of largest element in A[]
    for (i=0; i<n; i++) {
        if (strcmp(A[i].name, A[imax].name)>0) {
            imax= i;
        }
    }
    // swap A[n-1] & A[imax]
    student_t tmp= A[n-1]; A[n-1]= A[imax]; A[imax]= tmp;
    // recursive call
    sort_by_name(A, n-1);
}
```

Memo: Passing/returning struct with functions is INNEFFICIENT! Use pointers instead!

Compare:

```
float bad_get_score(student_t s) {
    return s.score;
}
```

and:

```
void get_score(student_t *ps) {
    return ps->score;
}
```

In each case, how many bytes is passed by one function call:

`bad_get_score(s1)` or `get_score(&s1)` ? Recall that:

```
#define MAX_NAME 50
typedef struct {
  char name [MAX_NAME+1];
  int id;
  float score;
} student_t;
```

from lec07.pdf: Scenario for ex 1-3 [grok W08]

People have titles, a given name, a middle name, and a family name, all of up to 50 characters each. People also have dates of birth (dd/mm/yyyy), dates of marriage and divorce (as many as 10 of each), and dates of death (with a flag to indicate whether or not they are dead yet). Each date of marriage is accompanied by the name of a person. Assuming that people work for less than 100 years each, people also have, for each year they worked, a year (yyyy), a net income and a tax liability (both rounded to whole dollars), and a date when that tax liability was paid.

Countries are collections of people. Australia is expected to contain as many as 30,000,000 people; New Zealand as many as 6,000,000 people.

lec07.E1: Give declarations that reflect the data scenario that is described.

```
#define MAXL 50
typedef char nstr[MAXL+1];
```

People have titles, a given name, a middle name, and a family name, ***all of up to 50 characters each****.* People also have dates of birth (dd/mm/yyyy), dates of marriage and divorce (as many as 10 of each), and dates of death (with a flag to indicate whether or not they are dead yet). Each date of marriage is accompanied by the name of a person. Assuming that people work for less than 100 years each, people also have, for each year they worked, a year (yyyy), a net income and a tax liability (both rounded to whole dollars), and a date when that tax liability was paid.

Countries are collections of people. Australia is expected to contain as many as 30,000,000 people; New Zealand as many as 6,000,000 people.

lec07.E1: Give declarations that reflect the data scenario that is described.

```
#define MAXL 50
typedef char nstr[MAXL+1];
typedef struct {
  nstr given, mid, fam;
} name_t;
typedef struct {
  int dd,mm,yyyy;
} date_t;

typedef struct {
  name_t spouse;
  date_t start, end;
} mary_t;
```

People have titles, a given name, a middle name, and a family name, all of up to 50 characters each. People also have dates of birth (dd/mm/yyyy), dates of marriage and divorce (as many as 10 of each), and dates of death (with a flag to indicate whether or not they are dead yet). Each date of marriage is accompanied by the name of a person. Assuming that people work for less than 100 years each, people also have, for each year they worked, a year (yyyy), a net income and a tax liability (both rounded to whole dollars), and a date when that tax liability was paid.

Countries are collections of people. Australia is expected to contain as many as 30,000,000 people; New Zealand as many as 6,000,000 people.

```
…
typedef struct {
  nstr given, mid, fam;
} name_t;
typedef struct {
  int dd,mm,yyyy;
} date_t;
typedef struct {
  name_t spouse;
  date_t start, end;
} mary_t;
typedef struct {
  int yyyy;
  int income, tax;
  date_t taxdate;
} work_t;
typedef struct {
  name_t name;
  date_t dob;
  date_t dod;
  int is_dead;
  mary_t mary[10]; //chan
  int nm;
  work_t work[100]; //cha
  int nw;
} person_t;
```

lec07.E2: Write a function that calculates the average age of death for a country. Do not include people that are not yet dead.

Countries are collections of people. Australia is expected to contain as many as 30,000,000 people;

```
#define MAX_PEOPLE 30000000
persont_t aus[MAX_PEOPLE];
```

New Zealand as many as 6,000,000 people.

```
???  avg_longevity(???  ) {

}
```

E3: Write a function that calculates, for a country, the total taxation revenue in a specified year.

```
typedef char nstr[MAXL+1];
typedef struct {
  nstr given, mid, fam;
} name_t;
typedef struct {
  int dd,mm,yyyy;
} date_t;
typedef struct {
  name_t spouse;
  date_t start, end;
} mary_t;
typedef struct {
  int yyyy;
  int income, tax;
  date_t taxdate;
} work_t;
typedef struct {
  name_t name;
  date_t dob;
  date_t dod;
  int is_dead;
  mary_t mary[10]; //cha
  int nm;
  work_t work[100]; //cl
  int nw;
} person_t;
```

```
tax_revenue(   ) {

}
```

Discuss HOW-TO:
The Task (adapted from Exercises 8.2 and 8.3)

Define a structure `vector_t` *that could be used to store points in two dimensions* `x` *and* `y` *(such as on a map).*

Give suitable declarations for a type `poly_t` *that could be used to store a closed polygon, which is represented as a sequence of points in two dimensions. Assume that no polygon contains more than 100 points. Then:*

a) *Write a function* `double distance(vector_t *p1, vector_t *p2)` *that returns the Euclidean distance between* `*p1` *and* `*p2`*. (note: not the same as the grok's exercise 8.02)*

b) *Write a function that returns the length of the perimeter of a polygon. Remember: not to pass or return struct (not the same as graok's 8.03)*

c) *[Homework, Challenging] Write a function that returns the area of a polygon.*

d) *[Homework] Define a data type for a line segment (on a map) and write a function that computes the midpoint of a segment.*

Some reviews for quiz2 [Wed & Thu 1PM classes]

Main Room

- a few remarks/Q&A on quiz 2
- do together: 8.02-8.03

Break-Out Rooms

- self-study for quiz 2
- implement 8.02-8.03, 8.05, 8.07

Data type of pointers

```
int n= 10;
double z= 1.5;
int *pi;
double *pz;
```

Valid? Invalid?

```
pi = &n;
pi = n;
pi = z;
pi = &z;
pz = &z;   pi= pz;
```

Pointer type is quite privilege! In:

```
int *p;
```

`p` cannot be mixed up with any other type, including with `double*` , `float*` , `char*`.

Consider the following declarations:

```
int n=44;   int c='a';  double z=2.5;
int *pn=&n; int *pc=&c; double *pz=&z;
```

In the context of the declarations, which of the following expressions contains a type incompatibility (and hence is not valid):

```
A.  pc == pz
B.  *pn == *pc
C.  *pn == *pz
D.  n == z
E.  &pn == &pc
```

Array name is a pointer constant!

Suppose that `A` is an array of int, and that `B` is declared to be a pointer of type `int *`. Which of the following is not a valid assignment statement, assuming that both `B` and the elements of `A` have been initialized to suitable values:

```
1. B = &(*A);
2. B = A + *B;
3. *A = A[1];
4. B = *(*A);
5. *B = A[1];
6. A= &A[2];
7. B= &A[2];
```

Constants can not be changed

The string on the RHS below are constants. Changing the value of constants is unreasonable and leads to runtime errors.

```
1. char *ps= "ABBA"
2. char s[10]= "The Best";
```

Here:

1. `ps` points to the constant string "`ABBA`", and commands like:

```
*ps= 'B';
strcpy(ps, "1234");
```

cause run-time error.

2. `"The Best"` is a constant, but array `s[]` is not. In line 2, the constant string is just copied to `s[]`. So the followings are valid:

```
*s= 'A';
strcpy(s, "ABBA!");
```

if A is an 1D array, i is an int, A[i] is valid, but might be undefined!
if A is an 2D array, A[i][j] is valid, but might be undefined!

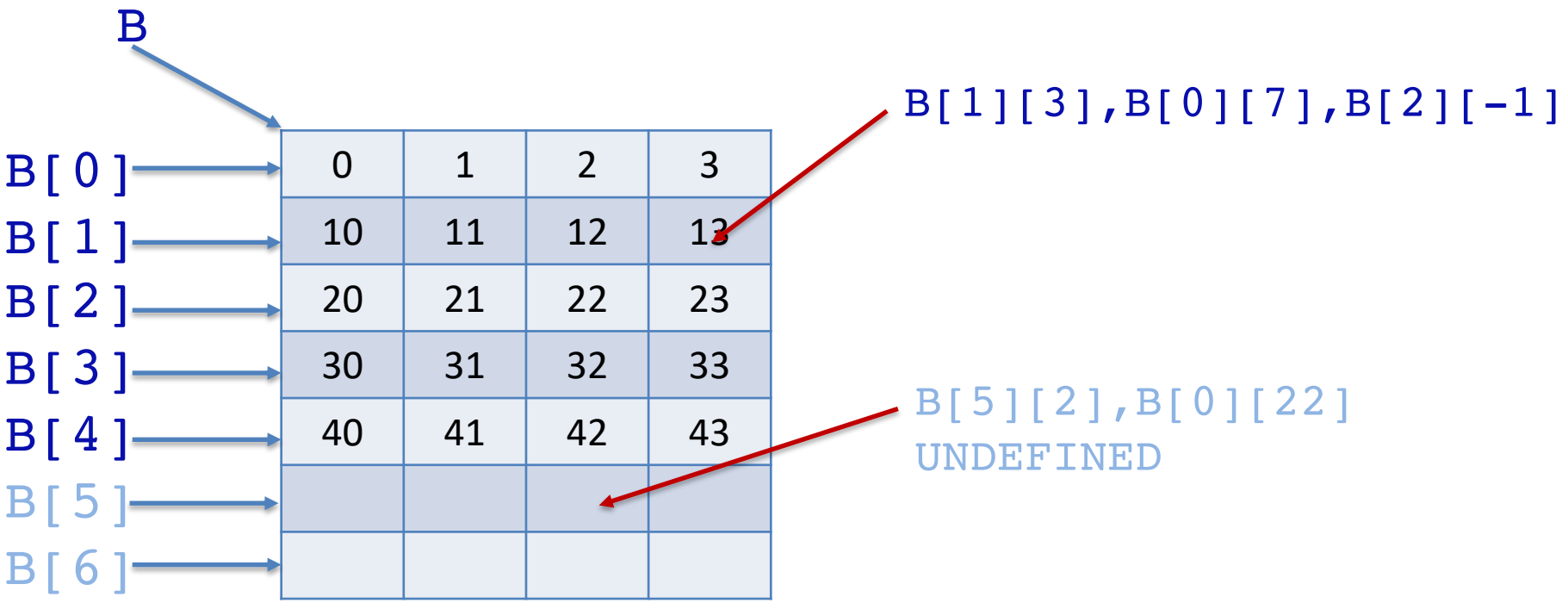
`int B[5][4]={{0,1,2,3},{10,11,12,13},…};`

`B` points to the location of `B[0][0];`

`B[i]` points to `B[i][0],` ie `B[i]` has the value of `&B[i][0];`

`B[i]+j` points to the `j`-th element after `B[i][0].` Depending on the value of `j`, this element can be at any position and can be undefined.

`B[i][j]` is valid, but might not be defined.



```
int A[5]={4,3,5,2,1};
int B[3][5]
   ={{1,12,3,14,5},{15,4,6,2,13},{10,7,8,11,9}};
```

What's the value of

1. `B[A[4]][A[1]]`
2. `B[A[1]][A[2]]`
3. `B[A[4]][A[3]]`
4. `B[A[4]][A[2]]`

Discuss HOW-TO:
The Task (adapted from Exercises 8.2 and 8.3)

Define a structure `vector_t` *that could be used to store points in two dimensions* `x` *and* `y` *(such as on a map).*

Give suitable declarations for a type `poly_t` *that could be used to store a closed polygon, which is represented as a sequence of points in two dimensions. Assume that no polygon contains more than 100 points. Then:*

a) *Write a function* `double distance(vector_t *p1, vector_t *p2)` *that returns the Euclidean distance between* `*p1` *and* `*p2`*. (note: not the same as the grok's exercise 8.02)*

b) *Write a function that returns the length of the perimeter of a polygon. Remember: not to pass or return struct (not the same as graok's 8.03)*

c) *[Homework, Challenging] Write a function that returns the area of a polygon.*

d) *[Homework] Define a data type for a line segment (on a map) and write a function that computes the midpoint of a segment.*