COMP10002 Workshop Week 9

Structs

Struct, pointers to struct, arrays of struct.
Discuss: 8.02,8.03
Discuss: Ex1,2,3 of lec07

Lab Time:

- 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

Warnings on Assignment 2 (due Friday Week 11) :

- likely challenging
- be prepared before Workshop Week 10 [at least understand & finish the 1st stage]

struct: a group of related variables

isolated variables	grouped into a struct

char hName[20]= “Harry Potter”,
dName[20]= “Draco Malfoy”;
int hId = 10001, dId = 10108;
float hScore= 7.5, dScore= 8.0;
 	
printf(“Student Harry:\n”);
printf(“name = %s\n”, hName);
printf(“id = %d]n”, hId);
printf(“score= %f\n”, hScore);
	typedef **struct {**
	char name[20];
	int id;
	float score;
	} student_t;
	
	student_t
	harry= {“Harry Potter”, 10001, 7.5},
	draco= {“Draco Malfoy”, 10108, 8.0};
	
	printf(“Student Harry:\n”);
	printf(“name = %s\n”, harry.name);
	printf(“id = %d]n”, harry.id);
	printf(“score= %f\n”, harry.score);

How best to organise collections of data?

Scenario

Data:
<= 1000 student records, each has ID, name, MST score.

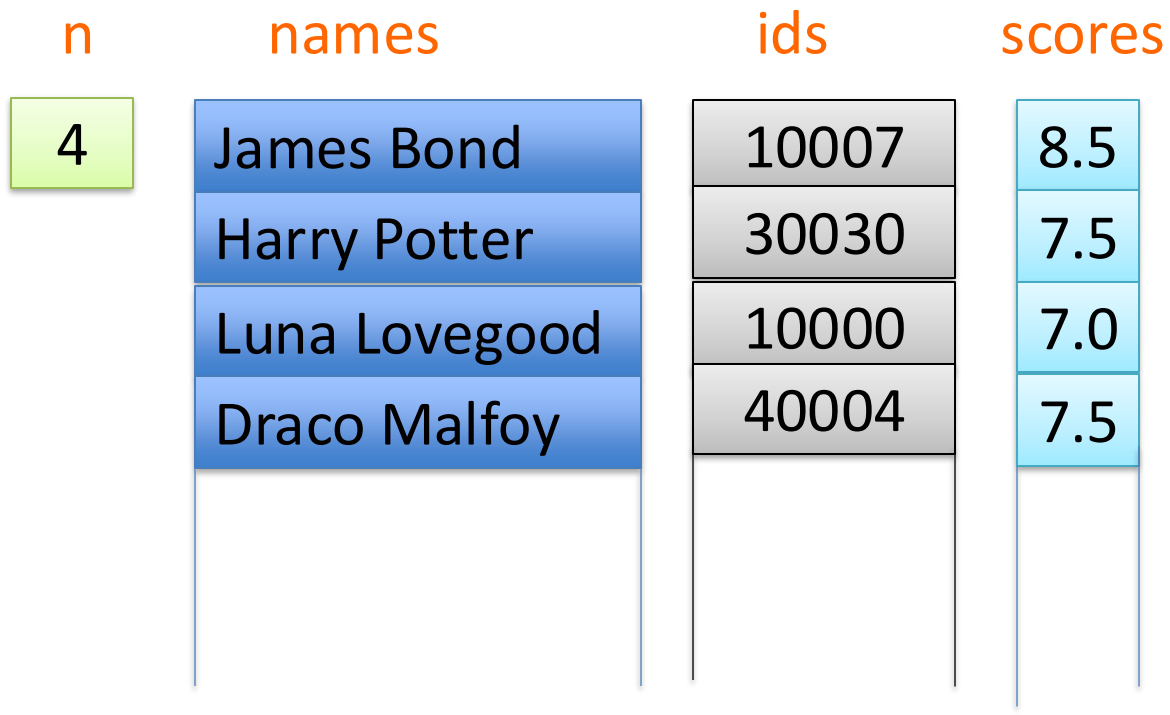
Some of the tasks:
1. searching: find a student and score, knowing the ID
2. sorting: sort data in ID order
3. . . .

Data structure Design 1 : 3 Parallel Arrays

```
char names[SIZE][MAX_NAME+1];
int ids[SIZE];
float scores[SIZE];

int n= 0;    // current length
```

- How to do the searching/sorting?
- Is that a *good* design?



struct

student_t		
name	id	score
James Bond	10007	8.5
Harry Potter	30030	7.5
Luna Lovegood	10000	7.0
Draco Malfoy	40004	7.5

n

4

Data structure Design 2 : 1 array of structs

```
typedef struct {
  char name[MAX_NAME+1];
  int id;
  double score;
  // other fields
} student_t;

student_t studs[SIZE]= {...};
int n= 0;   // current length
```

Searching: operates in just one array, return one data!

```
student_t search(student_t A[], int n, int id ) {
  ...if (A[i].id == id) return A[i];
}
```

Sorting: operates in a single array, need a single swap each time

```
void sort_by_name(student_t A[], int n) {
  ... if ( strcmp(A[j].name, A[j-1].name) < 0) {
       tmp= A[j]; A[j]= A[j-1];  A[j-1]= tmp;   // swap
  ...
}
```

Memo 1: Processing structs

```
typedef struct {
  char name [MAX_NAME+1];   // note that "name" is an array
  int id;
  float score;
} student_t;
```

Initialising :

```
student_t bob= { "Bob Taylor" , 1234 , 97.75 };
```

Processing `struct` by processing each member, using the dot operator:

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

But, unlike arrays, we can:

- make assignment: s= bob;
- and hence, struct can be the output of a function:

  student_t best_student(student_t s[], int n)
- Note: we cannot compare 2 struct-s : if (s==bob) is invalid

Discuss HOW-TO: 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).*

> ***8.02**: Write* `double distance(vector_t p1, vector_t p2)` *that returns the Euclidean distance between* **p1** *and* **p2**.

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 (in counter-clockwise order). Assume that no polygon contains more than 100 points.*

Memo 2: Pointers to structs

```
typedef struct {
        char name[31];
        int id;
        float score;
} student_t;

student_t s, *ps;
ps= &s;          // now *ps and s are equivalent
```

The following 2 lines are equivalent:

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

ps->name *is just a shorthand* for (*ps).name, *a great shorthand!*

Discuss HOW-TO: The Task (adapted from Exercises 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 (in counter-clockwise order). Assume that no polygon contains more than 100 points. Then:*

8.03: Write `double perimeter(poly_t P)` *that returns the length of the perimeter of a polygon.*

8.03-Extended:

- *how many bytes passed around when we make a call*

```
peri= perimeter(poly_t P);
```

?

- *can we reduce that amount?*

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 (in counter-clockwise order). Assume that no polygon contains more than 100 points. Then:*

a) *8.02: Write* `double distance(vector_t p1, vector_t p2)` *that returns the Euclidean distance between* **p1** *and* **p2**.

b) *8.03: Write* `double perimeter(poly_t P)` *that returns the length of the perimeter of a polygon.*

c) *8.03-improved: Write a function that returns the length of the perimeter of a polygon, but avoiding copying a whole struct. What the advantages in comparison with soln of b)?*

d) *[Homework, need Internet exploring] Write a function that returns the area of a polygon.*

$$A = \frac{1}{2}(x_0y_1 - x_1y_0 + \ldots + x_{n-2}y_{n-1} - x_{n-1}y_{n-2} + x_{n-1}y_0 - x_0y_{n-1}).$$

Memo 3: Passing/returning large struct-s in functions is INNEFFICIENT!
Use pointers instead!

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

Compare:

```
student_t best_student(student_t A[], int n) {
  ...
    return A[i];
}
```

and:

```
student_t *best_student(student_t A[], int n) {
  ...
    return &A[i];
}
```

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

best_student(A, n) ?

Ed 8.X1: from lec07.pdf: Scenario for ex 1-3

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.

a person:
- name (title, given, middle, family),
- dob (dd/mm/yy),
- dod,
- is_dead,
- number_of_marriages,
- marriages= {d_start, d_end, spouse(...)} x 10
- number_of_work_years
- work_years= {yyyy, income, tax, d_tax} x 100

Ed 8.X1: from lec07.pdf: Scenario for ex 1-3

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 {
  name
  dob, dod
  is_dead
  number_of_marriages
  marriages[10]
  number_of_work_years
  works[100]
} person_t;
```

lec07.E1: Give declarations that reflect :

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 date 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.

```
typedef struct {
  name,
  dob, dod
  is_dead
  number_of_marriages
  marriages[10]
  number_of_work_years
  works[100]
} person_t;
```

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

typedef struct {
  nstr title, 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 char
        nstr[MAXL+1];

typedef struct {
  nstr giv, 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 marys[10]; //need #define for 10
  int nmary;
  work_t works[100]; //need #define for 100
  int nwork;
} 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(??? ) {

}
```

```
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]; //change!
  int nm;
  work_t work[100]; //change!
  int nw;
} person_t;
```

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

```
???   tax_revenue(  ) {

}
```

LAB TIME

Lab Time:

- Exercises 8.02, **8.03 (using pointer parameter!)**, 8.05, and 8.07.
- Exercises 1, 2, and 3 in lec07.pdf (same as Ed exercise 8.X1)
- Then, if you still have time, start looking at Exercises 4, 5, 6, and 7 in lec07.pdf

- **Questions, including questions on dynamic memory**

Warnings on Assignment 2 (due Friday Week 11) :

- likely challenging
- be prepared before Workshop Week 10 [perhaps *at least* finish the first stage?]
- depending on your study plan, you might want to finish A2 during the break!

Additional Slides

Toward using multi-component objects…

Scenario

Data:
<= 1000 student records, each has ID, given name, MST score (and possibly some other fields).

Some of the tasks:
1. input: input all the data
2. searching: find a student knowing the ID
3. sorting: sort data in ID order
4. . . .

Data structure Design 1: Parallel Arrays

```
char names[SIZE][MAX_NAME+1];
int ids[SIZE];
float scores[SIZE];
// arrays for other fields
int n= 0;     // current length
```

Searching: how to return name, id, score?

```
? search( <the 3 arrays>, int n, int id ) {
    ...
}
```

Sorting: how to simplify the swaps?

```
void sort( <the 3 arrays>, int n) {
...
    swap names[?] with names[??]
    swap ids{?] with ids[??]
    swap scores[?] with scores[??]
...
}
```

example: insertion sort for an array of struct

```
// insertion sort: sorting elements A[] in ascending order
void ins_sort(int A[], int n) {
  int i;
  for (i= 1; i<n ; i++) {
    // swap A[i] left into correct position
    for (j= i-1; j>=0 && A[j+1]<A[j]; j--) {
      int_swap( &A[j] , &A[j+1] );
    }
}
```

```
void int_swap( int *a, int *b ) {
  int tmp= *a;
  *a= *b;
  *b= tmp;
}
```

```
// insertion sort: A[] in ascending order of student id
void ins_sort_stud(student_t A[], int n) {
  int i;
  for (i= 1; i<n ; i++) {
    // swap A[i] left into correct position
    for (j= i-1; j>=0 && A[j+1]<A[j]; j--) {
      int_swap( &A[j] , &A[j+1] );
    }
}
```

```
void swap( ?? a,  ?? b ) {

}
```