Disclaimer

These slides have been created by Liam Saliba as teaching material for his COMP10002 Foundations of Algorithms workshops.

While the best effort has been made to ensure the information given is accurate, there are no guarantees.

The lecture slides, textbook, and information published on the LMS must be considered as authoritative resources, and if any information clashes between this resource and those, you should consider those as correct. This material should only be considered as **supplementary material**, and does not replace the other materials listed here.

If you find any mistakes, differences, or if the work is not clear enough, do not hesitate to email Liam at liam.saliba@unimelb.edu.au. Thank you for contributing to the continued development of this resource.

Structs

Structs

Structs represent some real world structure.
Structs store possibly **different** data types in a contiguous block of memory. Each field is a fixed offset from the start.

```
struct {
    int id;
    char name[21];
    int year_opened;
    double balance;
} account;
```

Structs

Structs represent some real world structure.
Structs store possibly **different** data types in a contiguous block of memory. Each field is a fixed offset from the start.

```
struct {
    int id;
    char name[21];
    int year_opened;
    double balance;
} account;
```

```
struct account accounts[10];
int n = 0;
struct account acc = {
    .id=12,
    .name = "Liam",
    .balance = 10.04,
    .year_opened = 2023,
};
accounts[0] = acc;
```

Structs

```
struct account {
    int id;
    char name[21];
    int year_opened;
    double balance;
};
```

```
// create account a
struct account account1;
account1.id = 12;
strcpy(“Liam”, account1.name);
account1.year_opened = 2023;
account1.balance = 0.01;

struct account account2;
account2.id = 13;
strcpy(“Anon”, account2.name);
account2.year_opened = 2020;
account2.balance = 42069.42;
// verbose
```

Structs

```
typedef struct {
    int id;
    char name[21];
    int year_opened;
    double balance;
} account_t;

// “anonymous struct”
```

```
// create account a
account_t account1;
account1.id = 12;
strcpy(“Liam”, account1.name);
account1.year_opened = 2023;
account1.balance = 0.01;

account_t account2;
account2.id = 13;
strcpy(“Anon”, account2.name);
account2.year_opened = 2020;
account2.balance = 42069.42;
// verbose
```

Structs

```
typedef struct {
    int id;
    char name[21];
    int year_opened;
    double balance;
} account_t;
```

```
account_t new_account(int id, char name[21],
int year_opened, double balance) {
    account_t account1;
    account1.id = id;
    strcpy(name, account1.name);
    account1.year_opened = year_opened;
    account1.balance = balance;
    return account1;
}
```

```
account_t a1 = new_account(12, “Liam”, 2023, 0.01);
```

Caution: this creates a struct within scope of new_account, and (shallow) copies the whole memory block to the returned scope. Not a good idea if this is a large struct.

Structs

```
typedef struct {
    int id;
    char name[21];
    int year_opened;
    double balance;
} account_t;
```

This is all you need!

```
account_t a1 = {.id=12, .name="Liam",
                .year_opened=2023, .balance=0.01};
account_t a2 = a1;
a2.balance += 2;

printf("%.2lf %.2lf", a1.balance, a2.balance);
```

Structs

```
typedef struct {
    int id;
    char name[21];
    int year_opened;
    double balance;
} account_t;

account_t a1 = {12, “Liam”, 2023, 0.01};
account_t a2 = a1;  // (shallow) copies!

a2.balance += 2;

printf(“%.2lf %.2lf”, a1.balance, a2.balance);  // 0.01 2.01
```

Structs – Shallow copy

```
struct mystruct {
    int val;
    char *name;
};

char name[] = “Test”;
struct mystruct s1 = {12, name};
struct mystruct s2 = s1;

s1.name[0] = ‘t’;
S2.val = 2;

printf(“%d %d”, s1.val, s2.val};
printf(“%s %s”, s1.name, s2.name);
```

Structs - Shallow copy

```
struct mystruct {
    int val;
    char *name;
};

char name[] = “Test”;
struct mystruct s1 = {12, name};
struct mystruct s2 = s1;

s1.name[0] = ‘t’;
S2.val = 2;

printf(“%d %d”, s1.val, s2.val};   // 12 test
printf(“%s %s”, s1.name, s2.name); // 2 test
```

Need to copy whatever is being stored as a pointer -- deep copy.

Structs - Motivation

```
typedef struct {
    int id;
    char name[21];
    int year_opened;
    double balance;
} account_t;

int cmp_account_id(void * a1, void *a2) {
    account_t *p1, *p2;
    return p1->id - p2->id;
}

account_t accounts[NUM_ACCTS] = …;

qsort(accounts, NUM_ACCTS, sizeof(account_t), cmp_account_id);
// now accounts are sorted by id!
```

We can now sort by any of these fields.

Structs - Motivation

```
typedef struct {
    int id;
    char name[21];
    int year_opened;
    double balance;
} account_t;

int cmp_account_balance(void* a1, void* a2) {
    account_t *p1, *p2;
    return p1->balance - p2->balance;
}

account_t accounts[NUM_ACCTS] = …;

qsort(accounts, NUM_ACCTS, sizeof(account_t), cmp_account_balance);
// now accounts are sorted by balance!
```

We can now sort by any of these fields.

Hierarchical struct

```
typedef struct {
    int id;
    char name[21];
    int opened_year, opened_month, opened_date;
    int closed_year, closed_month, closed_date;
    double balance;
} account_t;

… more repetition.
Solution?
```

Hierarchical struct

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

typedef struct {
    int id;
    char name[21];
    date_t opened, closed;
    double balance;
} account_t;


account_t liam = {42, "Liam Saliba", {20,9,2022}, {-1,-1,-1}, 0};
liam.opened.yyyy
```

Hierarchical struct

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

typedef struct {
    int id;
    char name[21];
    date_t opened, closed;
    double balance;
} account_t;
```

```
void print_date(date_t date) {
    printf("%d/%d/%d", date.dd, date.mm, date.yyyy);
}


// then
account_t account = {12, "Liam", {11,12,2020},
{12,12,2020}, 0.01};

print_date(account.opened);

print_date(account.closed);
```

Structs

Structs are not arrays.

```
typedef struct {
    int x, y;
} point_t;
```

!=

```
typedef int point_t[2];
```

```
// both can be declared like this
point_t p = {1, 2};

// struct
printf("(%d, %d)", p.x, p.y);

// array
printf("(%d, %d)", p[0], p[1]);

// (extension:)
// point_t p = {.y = 2, .x = 1};
```

Arrays ...

```
// Calculate the Euclidean distance √((x1 − x2)² + (y1 − y2)²). between two
   points p1=(x1, y1) and p2=(x2, y2).

#include <math.h>
double distance(int x1, int y1, int x2, int y2) {
    return sqrt( (x1-x2) * (x1-x2) + (y1-y2) * (y1-y2) );
}
```

Structs ...

```
// Calculate the Euclidean distance √((x1 − x2)² + (y1 − y2)²). between two
   points p1=(x1, y1) and p2=(x2, y2).

#include <math.h>
typedef struct {int x, y;} point_t;

double distance(point_t p1, point_t p2) {
    int xs = p1.x - p2.x, ys = p1.y - p2.y;
    return sqrt( xs*xs + ys*ys );
}
```

Arrays of structs

```
typedef struct {int x, y;} point_t;

// “highest” -- maximum y value
point_t highest_point(point_t points[], int n) {
    assert(n > 0);
    point_t highest = points[0];
    for (int i = 1; i < n; i++) {
        if (points[i].y > highest.y) {
           highest = points[i];
        }
    }
    return highest;
}
```

Arrays of structs

```
// “highest” -- maximum y value
// arguably less readable, and we can’t return the entire point
int highest_point(int points[][2], int n) {
    assert(n > 0);
    int highest = points[0][1];
    for (int i = 1; i < n; i++) {
        if (points[i][1] > highest) {
           highest = points[i][1];
        }
    }
    return highest;
}
```

Arrays of structs

```
// “highest” -- maximum y value
// arguably less readable now returning index
int highest_point(int points[][2], int n) {
    assert(n > 0);
    int highest_i = 0;
    for (int i = 1; i < n; i++) {
        if (points[i][1] > points[highest_i][1]) {
           highest_i = i;
        }
    }
    return highest_i;
}
```

Arrays of structs

```
struct pair_pt {
    point_t p1, p2;
    double dist;
}; // not typedef'd as to not pollute types

// Find the closest pair of points, and their distance
struct pair_pt closestpair(point_t points[], int n) {
    assert( n >= 2);
    struct pair_pt closest_pair = {0, 1, distance(points[0], points[1]) }
    for (int i = 0; i < n - 1; i++) {
        for (int j = 1; j < n; j++) {
            double dist = distance(points[i], points[j]);
            if (dist < closest_pair.dist) { // why can't we do = {points[i], points[j], dist} ?
                closest_pair.p1 = points[i];
                closest_pair.p2 = points[j];
                closest_pair.dist = dist;
            }
        }
    }
    return closest_pair;
}
```

Structs - pointers

```
// sets a point p to (x,y)
point_t* set_point(point_t *p, int x, int y) {
    (*p).x = x;
    (*p).y = y;
    return p;
}
```

Structs - pointers

```
// sets a point p to (x,y)
point_t* set_point(point_t *p, int x, int y) {
    p->x = x;
    p->y = y;
    return p;
}
```

Arrays of structs

```
typedef struct {int x, y;} point_t;

// “highest” -- maximum y value
point_t highest_point(point_t points[], int n) {
    assert(n > 0);
    point_t highest = points[0];
    for (int i = 1; i < n; i++) {
        if ((points+i)->y > highest.y) {
            highest = points[i];
        }
    }
    return highest;
}
```

Recursive struct definition

```
typedef char name_t[NAME_LEN + 1];

// family tree

typedef struct {
    name_t name;
    person_t *mother;
    person_t *father;
    person_t *spouse;
} person_t;

// this won't work.  Can't use person_t when not yet defined.
```

Recursive struct definition

```
typedef char name_t[NAME_LEN + 1];

// family tree
typedef struct person person_t;

struct person {
    name_t name;
    person_t *mother;
    person_t *father;
    person_t *spouse;
};
```

Coming up

- This definition is very powerful.

- Used for:
  - Graphs
  - Linked lists
  - Trees
  - … among others

Final notes on structs

- Use structs for…
  - Store hierarchically arranged data
  - Minimise repeated variable declarations (int year1, year2, year3 …)
  - Simplifying function calls
  - Multiple return values from functions (less messy than pointers)

- Be careful...
  - Structs are copied in their entirety when passed to a function
  - For large structs, this is an expensive operation
  - We may instead copy a pointer to the structure instead
    - Every struct.parameter becomes a struct->parameter

Agenda

- **Grok Ex9-1 Structs**
  - Not convinced by structs yet? Try this ex.
- Grok Ex7-12
- Grok Ex7-16
- Grok Ex8-07

Chat about malloc

Completed Exercise 9-1

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <assert.h>

// defines the bool type (_Bool)
// true (1) and false (0)

#define MAX_TOKEN_LEN 50
#define MAX_MARRIAGES 10
#define MAX_WORK_YEARS 100

#define NAME_TOKENS 4

#define POPULATION_AU 30000000
#define POPULATION_NZ 6000000
#define INITIAL_POPULATION 1000

typedef char name_token_t[MAX_TOKEN_LEN + 1];

typedef int dollars_t;

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