CPSC 259: Data Structures and Algorithms for Electrical Engineers

Structs (Records)

Textbook References:
(a) Etter: start of Chapter 7
(b) Thareja (*first edition*): 7.1 – 7.4
(c) Thareja (*second edition*): 5.1 – 5.4

Learning Goals

- Define and use records (e.g., structs in C) in an implementation with dynamic memory allocation.
- Become more familiar with addresses and pointers in C.

Records (Structures)

- Often, we need to deal with related data (i.e., several attributes) about a specific entity. For example:
  - an **employee** is identified by a unique employee number, and has the following additional (possibly non-unique) attributes: name, street address, city, province, postal code, salary, job title, etc.
- A structure is declared using the keyword struct followed by a structure name. All the variables of a structure are declared within the structure. A structure type is defined by using the given syntax.

```
struct Employee {
  int empNum;
  char   name[MAXLEN];
  double salary;
};
```

Records (Structures)

- The structure definition does not allocate any memory. It just gives a template that conveys to the C compiler how the structure is laid out in memory and gives details of the member names.
- Memory is allocated for the structure when we declare a variable of the structure. For example, we can define a variable of an employee by writing

```
struct Employee {
  int empNum;
  char    name[MAXLEN];
  double salary;
};
```

```
struct Employee boss1;
```

Typedef

- We can define a structure as a type so then we can declare it without using the `struct` keyword.

struct Employee boss1;  *Employee boss1;*

Method 1

```
typedef struct{
   int      empNum;
   char   name[MAXLEN];
   double   salary;
} Employee;
```

Method 2

```
struct Employee{
   int      empNum;
   char   name[MAXLEN];
   double   salary;
} ;

typedef struct Employee Employee
```

Initialization of Structures

- Initializing a structure means assigning some constants to the members of the structure.
- The initializers are enclosed in braces and are separated by commas. Note that initializers match their corresponding types in the structure definition.

- When the user does not explicitly initialize the structure then C automatically does that. For int and float members, the values are initialized to zero and char and string members are initialized to the '\0' by default.

```
Employee    former_boss = {5000, "Derek", 99250.75};
```

Accessing the Members of a Structure

- Each member of a structure can be used just like a normal variable, but its name will be a bit longer. A structure member variable is generally accessed using the '.' (dot operator).

- The syntax of accessing a structure member:

```
new_boss.empNum = 1000;
strcpy(new_boss.name, "Ralph");
new_boss.salary = 125750.99;
```

Arrays of Structures

- The general syntax for declaring an array of structure can be given as:

```
Employee     staff_junior[20];
```

- Now, to assign values to the ith staff, we will write:

```
staff_junior[0].empNum = 2000;
strcpy(staff_junior[0].name, "Susan");
staff_junior[0].salary = 50000.00;
```

Declaring a Stand-alone Structure (pointers)

- Like in other cases, a pointer to a structure is never itself a structure, but merely a variable that holds the address of a structure. The syntax to declare a pointer to a structure can be given as

```
Employee *   vice_president;
vice_president = (Employee *) malloc( sizeof(Employee) );
```

- To access the members of the structure, one way is to write **/* get the structure, then select a member */**

```
(*vice_president).salary += 10000.00; /* one way */
```

- An alternative to the above statement can be used by using 'pointing-to' operator (->)

```
vice_president->empNum = 1; /* another way */
vice_president->salary = 105000.00;
```

Declaring an Arrays of a Structure

```
 Employee *  staff_senior;
 staff_senior  = (Employee *) malloc(num_staff_senior *
sizeof(Employee));

/* Accessing the data using arrays */
 staff_senior[i].empNum = 100 +i;

/* another way of accessing the data, via pointer arithmetic */
(staff_senior +i)->salary = 80000;/* parentheses needed */
(*(staff_senior+i)).salary  *= 1.05;    /* 5% pay increase */
```

Nested Structs

- A structure can be placed within another structure;

```
typedef struct{
    int      dd;
    int      mm;
    int      yy;
} Date;
```

```
typedef struct{
    int      empNum;
    char     name[MAXLEN];
    double   salary;
    Date     dob;
} Employee;
```

```
int main(void){
    Employee instructor;
    instructor.empNum = 100;
    instructor.dob.dd = 10;
    instructor.dob.mm = 11;
    instructor.dob.yy = 1962;
}
```

Passing a structure to a function call by value

- When a structure is passed as an argument, it is passed using call by value method. That is a copy of each member of the structure is made.

```
printEmp(new_boss);
```

```
void printEmp(Employee emp){
      printf("Employee Number: %d\n",      emp.empNum);
      printf("Employee Name: %s\n",     emp.name);
      printf("Employee Salary: $%.2f\n\n", emp.salary);
}
```

Passing a structure to a function call by reference

- This is a very inefficient method especially when the structure is very big or the function is called frequently. Therefore, in such a situation passing and working with pointers may be more efficient.

```
printEmp_ptr(&new_boss);
```

```
void printEmp_ptr(Employee* emp){
	printf("Employee Number: %d\n",	(*emp).empNum);
	printf("Employee Name: %s\n",	(*emp).name);
	printf("Employee Salary: $%.2f\n\n", (*emp).salary);
}
```

Please see employee_records.c

Clicker question

What is the size of the Employee struct given sizeof(int) =4, sizeof(char*)=8, and sizeof(double)=8?

```
typedef struct{
 int      empNum;
 char*    name;
 double   salary;
} Employee;
```

A. 12 bytes
B. 16 bytes
C. 20 bytes
D. 32 bytes
E. We can't estimate the size since we don't know how many characters are in the name field.

Clicker question

What is the size of the Employee struct given sizeof(int) = 4, sizeof(char*)=8, and sizeof(double)=8?

```
typedef struct{
 int      empNum;
 char*    name;
 double   salary;
} Employee;
```

A. 12 bytes
B. 16 bytes
C. 20 bytes
D. 32 bytes
E. We can't estimate the size since we don't know how many characters are in the name field.

Clicker question

- What is stored in the "name" field in boss? Choose the best answer

```
typedef struct{
  int      empNum;
  char*    name;
  double   salary;
} Employee;
...
Employee boss;
```

A. The name field eventually contains a character string of some currently unknown length, so the size of "boss" will change.

B. The name field eventually contains a character string of some currently unknown length, but the size of "boss" will not change.

C. The name field is a pointer to another area of memory that eventually holds a character string of some currently unknown length, but the size of "boss" will not change.

Clicker question

- What is stored in the "name" field in boss? Choose the best answer

```
typedef struct{
  int      empNum;
  char*    name;
  double   salary;
} Employee;
...
Employee boss;
```

A. The name field eventually contains a character string of some currently unknown length, so the size of "boss" will change.

B. The name field eventually contains a character string of some currently unknown length, but the size of "boss" will not change.

C. The name field is a pointer to another area of memory that eventually holds a character string of some currently unknown length, but the size of "boss" will not change.

A running example

- Example of an Airplane structure.

```
struct Airplane {
  int    flight_number;
  char   source[32];
  char   destination[32];
};
```

- Declare and initialize a local record using the Airplane structure

```
struct Airplane AC={101, "Vancouver", "Calgary"};
```

```
struct Airplane AC;
AC.flight_number = 101;
strcpy(AC.source, "Vancouver");
strcpy(AC.destination, "Calgary");
```

A running example

```
struct Airplane {
  int    flight_number;
  char   source[32];
  char   destination[32];
};
```

- Declare and initialize a local array of records using the Airplane structure

```
struct Airplane     WJ[10];
```

```
WJ[5].flight_number = 201; /* WJ */
strcpy(WJ[5].source, "Vancouver");
strcpy(WJ[5].destination, "Edmonton");
```

A running example

```
struct Airplane {
  int    flight_number;
  char   source[32];
  char   destination[32];
};
```

- Use a pointer to declare and assign values to exactly one plane

```
struct Airplane *   dynamic_AC;
```

```
dynamic_AC =(struct Airplane *) malloc(sizeof(struct Airplane )) ;
dynamic_AC->flight_number = 301;
strcpy(dynamic_AC->source, "Montreal");
strcpy(dynamic_AC->destination, "Toronto");
```

A running example

- Dynamically allocate more than one plane, but still use only one pointer.

```
struct Airplane {
  int    flight_number;
  char   source[32];
  char   destination[32];
};
```

```
dynamic_AC2 =(struct Airplane*) malloc( 3 * sizeof (struct Airplane));
```

```
dynamic_AC2[1].flight_number = 402;
strcpy(dynamic_AC2[1].source, "Toronto");
strcpy(dynamic_AC2[1].destination, "San Francisco");
```

A running example

```
struct Airplane {
  int    flight_number;
  char   source[32];
  char   destination[32];
};
```

- Consider the following code

```
dynamic_AC2 = (struct Airplane *) malloc( 3 * sizeof(struct Airplane) );

dynamic_AC2[3].flight_number = 404;
strcpy(dynamic_AC2[3].source, "Toronto");
strcpy(dynamic_AC2[3].destination, "Honolulu");
```

A: This code is safe to be executed

B: This code might crash the program

C: I don’t know

A running example

```
struct Airplane {
  int    flight_number;
  char   source[32];
  char   destination[32];
};
```

- Consider the following code

```
dynamic_AC2 = (struct Airplane *) malloc( 3 * sizeof(struct Airplane) );

dynamic_AC2[3].flight_number = 404;
strcpy(dynamic_AC2[3].source, "Toronto");
strcpy(dynamic_AC2[3].destination, "Honolulu");
```

A: This code is safe to be executed

B: This code might crash the program

C: I don’t know

A running example

Example 5: Use an array of pointers to airplane struct EACH pointer can point to zero, one, or more dynamically allocated airplane structs.

```
struct Airplane   * dynamic_WJ[10];
dynamic_WJ[0] = NULL;   /* zero planes */
dynamic_WJ[1] = NULL;
dynamic_WJ[7] = (struct Airplane *) malloc( 5 * sizeof(struct Airplane) );
dynamic_WJ[8] = (struct Airplane *) malloc( 1 * sizeof(struct Airplane) );
dynamic_WJ[9] = (struct Airplane *) malloc( 100 * sizeof(struct Airplane) );
```

See pointers_airplanes_dma_handout.pdf and pointers_airplanes_dma.c

```
struct Airplane ** dynamic_BA;
dynamic_BA = (struct Airplane **)malloc(20 * sizeof(struct Airplane *));
dynamic_BA[0] = (struct Airplane *)malloc(5 * sizeof(struct Airplane));
```

Learning Goals revisited

- Define and use records (e.g., structs in C) in an implementation with dynamic memory allocation.
- Become more familiar with addresses and pointers in C.