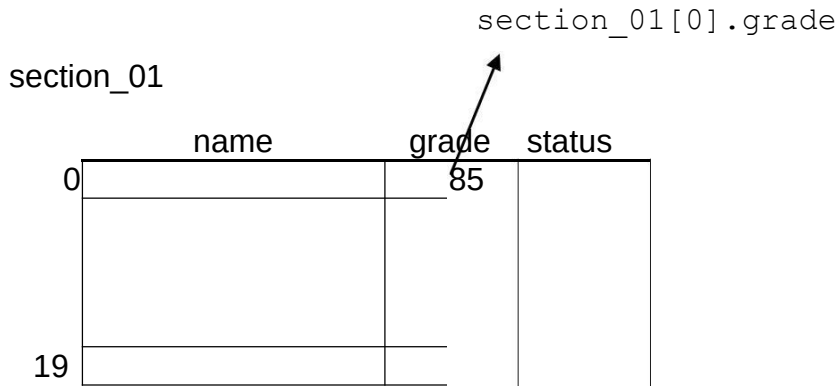
Arrays Of Structures

- We can declare arrays of structures as shown below:

Example:

```
struct stu_t {
     char    name[25];
     int     grade;
     char    status;
} section_01[20], section_02[20];

section_01[0].grade = 85;
// (*section_01).grade = 85;
// section_01->grade = 85;
```



Example: Display the grades of the students in Section 2.

```
for (i = 0; i < 20; i++)
     printf("%d\n", section_02[i].grade);
```

or

```
     printf("%d\n", (section_02 + i)->grade);
```

or

```
     printf("%d\n", (*(section_02 + i)).grade);
```

Example: Display the first 5 letters of the student names in Section 1.

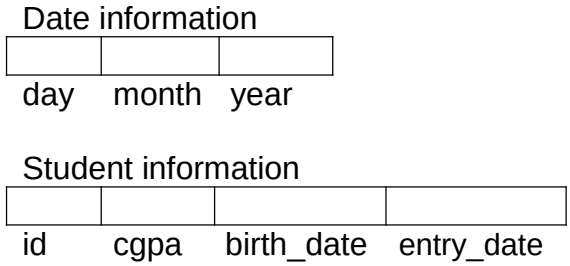
```
for (i = 0; i < 20; i++)
{
     for (j = 0; j < 5; j++)
         printf("%c",
     section_01[i].name[j]); printf("\n");
}
```

Nested Structures

- A structure cannot have a member with the same type as itself. For example, none of the elements of `stu_t` may have the type `stu_t`.

- However, it is possible for a whole structure to be a member of another structure. These are named as *nested structures*. In such cases, the member structure must be defined first.

- For example, let's define another student structure that consists of the ID, CGPA, birthdate and entrance date of a student. Notice that both dates should consist of three members: day, month, year. Thus, we can first define a date type and then use that type for both dates in the structure type.




```
enum month_t { JAN = 1, FEB, MAR, APR, MAY, JUN, JUL,
               AUG, SEP, OCT, NOV, DEC };

typedef struct {
     int     day;
     month_t month;
     int     year;
} date_t;

typedef struct {
     int     id;
     double  cgpa;
     date_t  birth_date, entry_date;
} stu_t;
...
stu_t ceng_stu;
```

- In this case, to refer to the month when the CENG student was born, and the year s/he entered the university, we need to write

```
ceng_stu.birth_date.month = APR;
ceng_stu.entry_date.year = 2009;
```

- However, to refer to the ID or CGPA of that student, we still write

```
ceng_stu.id = 20902155;
ceng_stu.cgpa = 1.88;
```

  because `id` and `cgpa` are still direct members of student, where the others are indirect members.

- If we want to store information about **n** students what should we do?

```
stu_t *std;
std = (stu_t *) malloc (n * sizeof(stu_t));
```

- Now, we can assign the birthdate of the first student using the following statements:

```
std[0].birth_date.day = 7;
(*std).birth_date.month = JAN;
std->birth_date.year = 1992;
```

- In general, to reach to the information about the `k+1`st student, we may use one of the following prefixes:

```
std[k].
(*(std+k)).
(std+k)->
```

Home Exercise: Write a main that asks the number of students to the user and reads information about that many students from a file and displays the information about the student with the maximum CGPA.

- Using an array
- Without using any arrays (store only the last read student and the student with the maximum cgpa in two structure variables istead of array of structures)

Structures as Function Parameters

- Structures may be passed to functions by passing individual structure members, by passing an entire structure or by passing a pointer to a structure.

- Arrays of structures, like all other arrays, are automatically passed call by reference.

Example: Write a modular program that reads a time (as hour min sec) and a duration in seconds, and displays that time and the time after the duration.

> **Example Run:**
> Enter a time (as hours minutes seconds): **07 58 32**
> Enter the duration in seconds: **97**
>
> The current time: **07:58:32**
> The time after 97 seconds: **08:00:09**

- First, let's define the time as a structure:

```
typedef struct {
     int hour, min, sec;
} time_t;
```

- Now, let's write a function to get a time as input:

```
time_t get_time(void) {
     time_t t;
     scanf("%d %d %d",&t.hour, &t.min, &t.sec);
     return(t);
}
```

- We can also write it as follows:

```
void get_time(time_t *t) {
     scanf("%d %d %d", &t->hour, &t->min, &t->sec);
     //scanf("%d %d %d", &(*t).hour, &(*t).min, &(*t).sec);
}
```

- The first one should be called as

```
time_t time;
time = get_time();
```

- The second one should be called as `get_time(&time);`

- The second one uses less memory, and makes no data transfer; so more efficient.

Home Exercise: Modify the `get_time` function so that it validates the time.

- Now, let's write a function that displays a time.

```
void display_time(time_t t) {
     printf("%d:%d:%d\n", t.hour, t.min, t.sec);
}
```

- Since the function uses extra space for it and the actual time will be copied to that space when the function is called, this usage is not very efficient. To increase the efficiency, even though `t` is not an output parameter, you may pass it by reference.

```
void display_time(time_t *t) {
printf("%d:%d:%d\n", t->hour, t->min, t->sec);
}
```

Home Exercise: Modify the above function so that it displays the time as **08:00:09** instead of **8:0:9**.

- Now, let's write a function that returns time updated based on the given time and number of seconds.

```
time_t new_time (time_t t, int dur)
{int new_hour, new_min, new_sec;
     new_sec = t.sec + dur;
     t.sec = new_sec % 60;
     new_min = t.min + new_sec /60;
     t.min = new_min % 60;
     new_hour = t.hour + new_min / 60;
     t.hour = new_hour % 24;
     return (t);
}
```

- The three local variables `new_hour`, `new_min`, and `new_sec` represent a new time. Notice that, we could replace them with a single variable with the `time_t` type. However, since `new` is a reserved word, we can not use it as our variable's name.

- The function can also be written as a void function:

```
void new_time(time_t *t, int dur) {
     time_t nev;
     nev.sec = t->sec + dur;
     t->sec = nev.sec % 60;
     nev.min = t->min + nev.sec / 60;
     t->min = nev.min % 60;
     nev.hour = t->hour + nev.min / 60;
     t->hour = nev.hour % 24;
}
```

- The second one is more efficient, because time will not be transferred.

- Now, we are ready to write the main:

```
int  main(void)  {
     time_t  time;  // (input/output) the given and new time
     int duration;  // (input) duration in seconds

     // Get the time and duration
     printf ("Enter a time (as hours minutes seconds): ");
     get_time (&time);   // time = get_time ();
     printf ("Enter the duration in seconds:
     "); scanf ("%d", &duration);
     // Display the time
     printf ("\nThe current time: ");
     display_time (time);
     // Find and display the time after
     duration new_time (&time, duration);
     // time = new_time (time, duration);
     printf ("The time after %d seconds: ", duration);
     display_time (time);
     return(0);
}
```

Example: Given the IDs and midterm1, midterm2 and final exam grades of 75 students taking a course within the file **grades.txt**, calculate the overall grade of each student using 0.25, 0.35 and 0.4 as the weight of each exam, respectively, write the student IDs and overall grades to the file **overall.txt**, and then display (with proper messages) the average of each exam, the maximum overall grade, and the ID of the student taking that grade.

- We can solve this problem using parallel arrays. We can store the student IDs in a one-dimensional integer array, the exam grades and overall grades in a two-dimensional double array. We can not store the student IDs and the exam grades in the same two dimensional array, because their data types are different.

- However, we can define a structure which consists of one integer and four double members, as follows:

```
typedef struct {
     int    id;
     double mt1, mt2, fin, overall;
} student;
```

  and store all information about the students in a one dimensional array of type `student`.

- We can declare 6 separate variables as `sum1`, `sum2`, `sum3`, `avg1`, `avg2`, `avg3` to calculate the sum and average of the exams. Or, we can define a structure consisting of three double variables, one for each exam, and declare only one `sum` and one `avg` variable using that data type:

```
typedef struct {
     double mt1, mt2, fin;
} exams;
...
exams sum, avg;
```

- Notice that our `student` structure also contains three double members corresponding to each exam grade, and `exams` is a structure consisting of three such members. Therefore, we could include `exams` structure in our `student` structure, making sure that `exams` is defined before `student`, as follows:

```
typedef struct {
     double mt1, mt2, fin;
} exams;

typedef struct {
     int    id;
     exams  grd;
     double overall;
} student;
```

- We can define a function which calculates the overall grade of one student, given the exam grades of the student, and the percentage of each exam.

```
double overall_grd(exams st_grd,
                   double per_mt1, double per_mt2, double per_fin) {
    return (st_grd.mt1 * per_mt1 + st_grd.mt2 * per_mt2 + st_grd.fin
            * per_fin);
}
```

- We can call this function in our main as follows:

```
std[k].overall = overall_grd(std[k].grd, 0.25, 0.35, 0.4);
```

- Although st_grd is an input parameter, to increase efficiency, we could define this function also as follows:

```
double overall_grd(exams *st_grd,
                   double per_mt1, double per_mt2, double per_fin)
    { return (st_grd->mt1 * per_mt1 + st_grd->mt2 * per_mt2 +
            st_grd->fin * per_fin);
}
```

- We can call this function in our main as:

```
std[k].overall = overall_grd(&std[k].grd, 0.25, 0.35, 0.4);
```

- Now, we are ready to write the main:

```
int main(void) {
    student std[MAX];          // (input) student info
    exams sum = {0}, avg;      // sum and average of each exam
    int k, max_std;
    FILE *grd_file, *ovr_file;

    // Open the input file and check if exists
    grd_file = fopen("grades.txt", "r");
    if (grd_file == NULL)
       printf ("File can not be opened!\n");
    else {
       // Open the output file
       ovr_file = fopen("overall.txt", "w");
       for (k = 0; k < MAX; k++) {
          // Read student information
          fscanf(grd_file, "%d %lf %lf %lf", &std[k].id,
              &std[k].grd.mt1, &std[k].grd.mt2, &std[k].grd.fin);
          // Calculate the overall grade of each student
          std[k].overall = overall_grd(&std[k].grd,0.25,0.35,0.4);
          // Write the id and overall grade of the student
          fprintf(ovr_file,"%d %f\n", std[k].id, std[k].overall);
       }
       // Close the files
       fclose(grd_file);
       fclose(ovr_file);

       // Find and output the average of each exam */
       for (k = 0; k < MAX; k++) {
          sum.mt1  +=  std[k].grd.mt1;
          sum.mt2  +=  std[k].grd.mt2;
          sum.fin += std[k].grd.fin;
       }

       avg.mt1 = sum.mt1 / MAX;
       avg.mt2 = sum.mt2 / MAX;
       avg.fin = sum.fin / MAX;
       printf ("Average of Midterm 1 = %0.2f\n", avg.mt1);
       printf ("Average of Midterm 2 = %0.2f\n", avg.mt2);
       printf ("Average of Final Exam = %0.2f\n", avg.fin);
```

```
        // Find and output the maximum overall grade
        max_std = 0;
        for (k = 0; k < MAX; k++)
           if (std[k].overall >
               std[max_std].overall) max_std = k;
        printf ("Maximum overall grade = %0.2f\n",
                std[max_std].overall);
        // Output the id of the student taking that grade
        printf ("Student with ID %d got that grade.\n",
                std[max_std].id);
    }
    return (0);
}
```

- Notice that the percentages of the three exams are also double variables. So, we could declare only one percentage variable with the `exams` type, initialize it to the percentages, and use it as the parameter of the `overall_grd` function.

```
double overall_grd(exams *st_grd, exams *per) {
    return (st_grd->mt1 * per->mt1 + st_grd->mt2 * per->mt2
            + st_grd->fin * per->fin);
}

int main(void) {
    ...
    exams per = {0.25, 0.35, 0.4}; // percentage of each exam
    ...
          // Calculate the overall grade of each student
          std[k].overall = overall_grd(&std[k].grd, &per);
```

- It is possible to solve the above problem without using any arrays. Try it as a **home exercise**.

Home Exercise: Modify the program so that

- It will include a function that returns the average of each exam.
- It will include a function that returns the index of the student who got the maximum grade.
- It will not use subscript notation.
- It will ask the number of students and decide the array size accordingly.

➢ *READ Sec. 10.1 - 10.6 (pg 381 – 386) from Deitel & Deitel.*

- As you know, arrays are automatically passed to a function by reference.

Example:

```
void funct (int arr[]) {
     arr[0]++;
}

int main (void) {
     int a[10];
     a[0] = 0;
     funct(a);
     ...
```

- Both `funct` and `main` use the same memory area for the array. So `a[0]` is incremented and becomes 1.

- To pass an array by value, you can create a structure with the array as a member, and pass that structure to the function.

```
typedef struct {
     int member[10];
} arr_t;

void funct (arr_t arr) {
     arr.member[0]++;
}

int main (void) {
     arr_t a;
     a.member[0] = 0;
     funct (a);
     ...
```

- `funct` and `main` use different memory areas for the array. So `a.member[0]` is not incremented and stays 0.