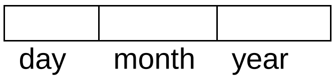
Nested Structures

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

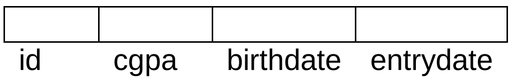
- However, it is possible for a whole structure to be a member of another structure. These are named as *nested structures*. In such cases, the member structure must be defined first.

- For example, let's define another student structure that consists of the <u>ID</u>, <u>CGPA</u>, <u>birthdate</u> and <u>entrance date</u> 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.

Date information



Student information



```
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   birthdate, entrydate;
} stu_t;
...
stu_t ctisstu;
```

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

```
ctisstu.birthdate.month = APR;
ctisstu.entrydate.year = 2009;
```

- 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].birthdate.day = 7;
(*std).birthdate.month = JAN;
std->birthdate.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)->
```

<u>Home Exercise:</u> 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 instead 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 the time after the duration.

Example Run:
Enter a time (as hours minutes seconds): **07 58 32**
Enter the duration in seconds: **97**

The time after 97 seconds: **08:00:09**

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

tm_t getTime(void) {
     tm_t t;
     scanf("%d %d %d", &t.hour, &t.min, &t.sec);
     return(t);
}

void getTime(tm_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

```
tm_t time;
time = getTime();
```

- The second one should be called as

```
getTime(&time);
```

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

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

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

```
void display_time(tm_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 displayTime(tm_t *t) {
     printf("%02d:%02d:%02d\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.

```
tm_t newTime (tm_t t, int dur)
{    tm_t temp;
     temp.sec = t.sec + dur;
     t.sec = temp.sec % 60;
     temp.min = t.min + temp.sec / 60;
     t.min = temp.min % 60;
     temp.hour = t.hour + temp.min / 60;
     t.hour = temp.hour % 24;
     return (t);
}
```

- The function can also be written as a void function, which is more efficient, because time will not be transferred:

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

- Now, we are ready to write the main:

```
int main(void) {
     tm_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): ");
     getTime (&time);           // time = getTime ();
     printf ("Enter the duration in seconds: ");
     scanf ("%d ", &duration);
     // Find and display the time after duration
     newTime (&time, duration); // time = newTime (time, duration);
     // Display the time
     printf ("The time after %d seconds: " , duration);
     displayTime (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.

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

- 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;
} exam_t;
...
exam_t sum, avg;
```

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

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

typedef struct {
     int    id;
     exam_t  grd;
     double overall;
} stu_t;
```

- 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 overallGrd(exam_t stgrd,exam_t per ) {
    return (stgrd.mt1 * per.mt1 + stgrd.mt2 * per.mt2 + stgrd.fin * per.fin);
}
```

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

```
std[k].overall = overallGrd(std[k].grd, percentage);
```

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

```
double overallGrd(exam_t *stgrd, exam_t *per) {
    return (stgrd->mt1*per->mt1 + stgrd->mt2*per->mt2 + stgrd->fin*per->fin);
}
```

- We can call this function in our main as:

```
     std[k].overall = overallGrd(&std[k].grd, &percentage);
```

- Now, we are ready to write the main:

```
int main(void) {
   stu_t std[MAX];                            // (input) student info
   exam_t sum = { 0 }, avg;                   // sum and average of each exam
   exam_t percentage = { 0.25, 0.35, 0.40 };      // percentage of each exam
   int k, maxstd;
   FILE *grdfile, *ovrfile;

   // Open the input file and check if exists
   grdfile = fopen("grades.txt", "r");
   if (grdfile == NULL)
        printf("File can not be opened!\n");
   else {
        // Open the output file
        ovrfile = fopen("overall.txt", "w");

        for (k = 0; k < MAX; k++) {

             // Read student information
             fscanf(grdfile, "%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 = overallGrd(&std[k].grd, &percentage);

             // Write the id and overall grade of the student
             fprintf(ovrfile, "%d %f\n", std[k].id, std[k].overall);
        }

        // Close the files
        fclose(grdfile);
        fclose(ovrfile);
        // Find the average of each exam and the maximum overall grade
        maxstd = 0;
        for (k = 0; k < MAX; k++)
        {
             sum.mt1 += std[k].grd.mt1;
             sum.mt2 += std[k].grd.mt2;
             sum.fin += std[k].grd.fin;
             if (std[k].overall > std[maxstd].overall)
                  maxstd = k;
        }

        avg.mt1 = sum.mt1 / MAX;
        avg.mt2 = sum.mt2 / MAX;
        avg.fin = sum.fin / MAX;

        // Output the average of each exam and the maximum overall grade
        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);
        printf("Maximum overall grade = %0.2f\n", std[maxstd].overall);

        // Output the id of the student taking that grade
        printf("Student with ID %d got that grade.\n", std[maxstd].id);
   }
   return (0);
}
```

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

<u>Home Exercise:</u> 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 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.