

# PROGRAMMING IN C: STRUCTURES

COMP1206 - PROGRAMMING II

Enrico Marchioni  
e.marchioni@soton.ac.uk  
Building 32 - Room 4019

- ▶ What is a structure and how to define it.
- ▶ Structure as a type.
- ▶ Defining new types/type names with `typedef`.
- ▶ Structures as input and output of functions.
- ▶ Arrays of structures and structures with arrays.
- ▶ Pointer to structures and structures with pointers.

# STRUCTURES

- ▶ An array can be seen as a tool to group elements of the same type together.
- ▶ Structures are a tool that allows you to group together elements of different types.
- ▶ Imagine you need to store the time. You can declare and initialize three variables:

```
int hour = 14, min = 23, sec = 38;
```

- ▶ If you need to store another time, you need to declare three more variables:

```
int hourNEW = 15, minNEW = 20, secNEW = 11;
```

- ▶ Very inefficient: you have to keep track of more and more variables.
- ▶ You can avoid this by using structures.

- You can define a structure to store time as follows

```
struct time
{
    int hour;
    int min;
    int sec;
}
```

- The structure `time` contains three members `hour`, `min` and `sec` all of the same type (integers).
- The general format to define a structure is the following

```
struct structNAME
{
    type member1;
    type member2;
    type member3;
    .
    .
}
```

- ▶ When you define a new structure you are, in a certain sense, defining a new type.
- ▶ Defining a structure gives you a sort of data template.
- ▶ After defining the structure `structNAME`, you can use it to declare other variables of the same type:

```
struct structNAME varNAME;
```

- ▶ `struct structNAME` can be used as type similar to `int`, `float`, etc.
- ▶ For instance, you can use `time` to declare other variables of the same type:

```
struct time meeting1, meeting2;
```

- ▶ `meeting1` and `meeting2` will be two structure variables both with members `hour`, `min` and `sec`.

- In general, after defining a structure `structNAME` and declaring a variable

```
struct structNAME varNAME
```

you can initialize one of its members with

```
varNAME.member1 = VALUE;
```

- So after declaring a structure variable `meeting1` of type `time` you can initialize its members as follows:

```
meeting1.hour = 14;  
meeting1.min = 23;  
meeting1.sec = 38;
```

- The member operator `"."` allows you to access a specific member in a structure.

- ▶ You can also initialize structures similar to arrays.
- ▶ You can initialize the whole structure:

```
struct time meeting1= { 14, 23, 38};
```

- ▶ Writing

```
struct time meeting1= { 14, 23};
```

initializes only the first two members of the structure, i.e. hour, min, and leaves sec uninitialized.

- ▶ Writing

```
struct time meeting1= { .hour = 14, .sec = 38};
```

initializes only the specified members.



```
#include <stdio.h>           //DEFINE A STRUCTURE FOR TIME AND INITIALIZE AN INSTANCE

int main (void)
{
    struct time // Define structure
    {
        int hour;
        int min;
        int sec;
    };

    struct time meeting; // Declare variable for structure

    meeting.hour = 13;    // Initialize members
    meeting.min = 25;
    meeting.sec = 0;

    printf ("The next meeting is at %.2i:%.2i:%.2i.\n", meeting.hour, meeting.min,
            meeting.sec);

    return 0;
}
```

```
The next meeting is at 13:25:00.
Program ended with exit code: 0
```

- Note: format characters `%.2i` are used to specify that two integer digits are to be displayed with zero fill.

# typedef

- ▶ typedef allows you to assign an alternate name to data types.
- ▶ For instance

```
typedef int counter;
```

defines the name `counter` to be equivalent to `int`.

- ▶ You can then define variables of type `counter` as follows:

```
counter x, y;
```

- ▶ To define a pointer to integer type you can use

```
typedef int* point;
```

- ▶ Writing

```
point x, y, z;
```

becomes equivalent to writing

```
int *x, *y, *z;
```

- To define a new type name with `typedef`, follow these steps:
  1. Write the statement as if a variable of the desired type were being declared:

```
int a [5];
```

2. Where the name of the declared variable would normally appear, substitute the new type name:

```
int ARRAY5 [5];
```

3. In front of everything, place the keyword `typedef`:

```
typedef int ARRAY5 [5];
```

- `ARRAY5` is now the name of a new type for 5 element arrays.
- So, writing

```
ARRAY5 newarray;
```

makes `newarray` of the type `ARRAY5`, i.e. makes it an array of 5 elements.

- ▶ You can use `typedef` with structures in two ways.
- ▶ First, after defining the `time` structure

```
struct time
{
    int hour;
    int min;
    int sec;
}
```

we can make a new type:

```
typedef struct time TIMETYPE;
```

- ▶ Alternatively, you can define the type along with the structure definition as follows:

```
typedef struct
{
    int hour;
    int min;
    int sec;
} TIMETYPE;
```

# FUNCTIONS AND STRUCTURES

- ▶ Functions can have structures as arguments and return another structure.
- ▶ We define a function that takes the current time as an argument and returns a new time updated by a second.
- ▶ We start by defining a new type

```
typedef struct
{
    int hour;
    int min;
    int sec;
} TIMETYPE;
```

- ▶ We then define an update function:

```
TIMETYPE update (TIMETYPE now) { }
```

- ▶ Both input and output of update are structures of type TIMETYPE.
- ▶ Alternatively, we could also have defined the function without TIMETYPE using struct time instead:

```
struct time update (struct time now) { }
```

```
#include <stdio.h>           //PROGRAM ASKS USER FOR CURRENT TIME AND RETURN UPDATED TIME

typedef struct               //Define Structure and Type Name TIMETYPE
{
    int hour;
    int min;
    int sec;
} TIMETYPE;

//Define update function - Argument is structure for current time.
//Output is structure with current time + 1 second
TIMETYPE update (TIMETYPE now)
{
    ++now.sec;    //Add 1 second

    if ( now.sec == 60 )    //If seconds=60, then set seconds=0 and add 1 minute
    {
        now.sec = 0;
        ++now.min;

        if ( now.min == 60 )    //If minutes=60, then set minutes=0 and add 1 hour
        {
            now.min = 0;
            ++now.hour;

            if ( now.hour == 24 )    //If hours=24, then set hours=0
            {
                now.hour = 0;
            }
        }
    }

    return now;    //Return updated structure
}
```



```
int main()
{
    TIMETYPE update (TIMETYPE now);    //Declare update function

    TIMETYPE current, next;    //Define variables for structures of type TIMETYPE

    printf ("Enter the time (hh:mm:ss): \n");    //Ask for current time

    scanf ("%i:%i:%i", &current.hour,
           &current.min, &current.sec);    //Enter current time

    next = update (current);    //Call update function and assign output to next
                                structure

    printf("Updated time is %.2i:%.2i:%.2i\n", next.hour, next.min, next.sec);    //
        Print updated time with values of memebers of next structure

    return 0;
}
```

```
Enter the time (hh:mm:ss):
23:59:59
Updated time is 00:00:00
Program ended with exit code: 0
```

- Note: specifying a nonformat character, such as ":" in a format string signals to the `scanf` function that the particular character is expected as input.

# ARRAYS AND STRUCTURES

- ▶ We can store and manage multiple times with arrays of structures.
- ▶ An array of structure is declared in the following form

```
struct structNAME arrayNAME[n];
```

so, arrayNAME is an array of n structures that conform to the template of structNAME.

- ▶ Following our example

```
struct time testTimes[5];
```

declares an array of 5 time structures.

- ▶ We can do the same with:

```
TIMETYPE testTimes[5];
```

- To initialize an array of structures, you can initialize the whole array, similar to a multidimensional array:

```
TIMETYPE testTimes[5] = {  
    {11, 59, 59}, {12, 0, 0}, {1,29,59},  
    { 23, 59, 59 }, { 19, 12, 27 }  
};
```

or

```
TIMETYPE testTimes[5] = { 11, 59, 59,12, 0, 0, 1,29,59, 23,  
    59, 59, 19, 12, 27};
```

- To initialize a member of a specific structure in the array, use:

```
testTimes[3].hour = 14;
```

This assigns value 14 to the member `hour` in the fourth structure of the array.

- The next program defines an array of 5 `TIMETYPE` structures and calls the `update` function for each array element.

```
#include <stdio.h>           //Define array of 5 time structures and update each time

// Insert definition of TIMETYPE here

// Insert definition of update function here

int main()
{
    TIMETYPE update (TIMETYPE now);    //Declare update function

    TIMETYPE testTimes[5] =           //Initialize array of structures
    {{11, 59, 59}, {12, 0, 0}, {1,29,59}, { 23, 59, 59 }, { 19, 12, 27 }};

    for (int x = 0; x < 5; x++)
    {
        printf("Time is %.2i:%.2i:%.2i", testTimes[x].hour,
               testTimes[x].min, testTimes[x].sec);    //Print current time

        testTimes[x] = update(testTimes[x]);    //Update current time

        printf("...one second later it's %.2i:%.2i:%.2i\n", testTimes[x].hour,
               testTimes[x].min, testTimes[x].sec);    //Print updated time
    }
    return 0;
}
```

```
Time is 11:59:59...one second later it's 12:00:00
Time is 12:00:00...one second later it's 12:00:01
Time is 01:29:59...one second later it's 01:30:00
Time is 23:59:59...one second later it's 00:00:00
Time is 19:12:27...one second later it's 19:12:28
Program ended with exit code: 0
```

testTimes[0]	{	.hour	11
		.min	59
		.sec	59
testTimes[1]	{	.hour	12
		.min	0
		.sec	0
testTimes[2]	{	.hour	1
		.min	29
		.sec	59
testTimes[3]	{	.hour	23
		.min	59
		.sec	59
testTimes[4]	{	.hour	19
		.min	12
		.sec	27

- ▶ Structures can also have arrays as members.
- ▶ For instance, define a structure `student` to store a student ID and the student's marks.

```
struct student
{
    int ID;
    int marks[5];
}
```

- ▶ You can now define a variable for each student:

```
struct student Lucy, Robert;
```

- ▶ To initialize the members of the structure, you can do the following:

```
Lucy.ID = 123456;
Lucy.marks[0] = 65;
Lucy.marks[1] = 70;
```

- ▶ Notice the use of the “.” operator and how it is used to access the elements of the array member.

# STRUCTURES AND POINTERS



- ▶ Pointers can also point to structures.
- ▶ To define a variable `pointNAME` to be pointer to a structure `structNAME`, you can use the format

```
struct structNAME *pointNAME;
```

- ▶ Following our example of the `time` structure, we can define a pointer variable `p_time` as follows:

```
struct time *p_time;
```

or, similarly

```
TIMETYPE *p_time;
```

- ▶ After declaring a structure variable

```
struct time now;    (TIMETYPE now;)
```

we can initialize a pointer

```
p_time = &now;
```

- To indirectly access members of a structure with a pointer you can use:

```
(*pointNAME).member
```

- In our example, to initialize a member of the `time` structure:

```
(*p_time).hour = 14;
```

- The parentheses are required because the structure member operator `.` has higher precedence than the dereference operator `*`.
- To access a member of a structure with a pointer you can also use the special structure pointer operator `->`:

```
pointNAME->member
```

- So,

```
(*p_time).hour = 14;
```

is the same as

```
p_time->hour = 14;
```

- ▶ You can dynamically allocate memory to pointers to a structure with `malloc`.
- ▶ After defining a structure `structNAME`, you can allocate memory to a pointer `p` that points to this structure as follows:

```
struct structNAME *p = malloc(sizeof(struct structNAME));
```

- ▶ In the case of the `time` structure, you can use

```
struct time *p = malloc(sizeof(struct time));
```

or, using the type `TIMETYPE`:

```
TIMETYPE *p = malloc(sizeof(TIMETYPE));
```

- ▶ You can then free the allocated space as usual:

```
free(p);
```

- ▶ The next program shows how you can allocate space for two pointers to `TIMETYPE` structures, use these pointers to initialize their members and update the time with the `update` function.

```
#include <stdio.h>
#include <stdlib.h>

// Insert definition of TIMETYPE here

// Insert definition of update function here

int main() {

    TIMETYPE *p = malloc(sizeof(TIMETYPE));    //Allocate memory to pointers
    TIMETYPE *u = malloc(sizeof(TIMETYPE));

    p->hour=14;    //Initialize structure members with pointer
    p->min=22;
    p->sec=38;

    *u = update(*p);    //Update structure by dereferencing the pointer

    printf("The updated time is: %.2i:%.2i:%.2i\n", u->hour, u->min, u->sec);
    //Print updated time

    free(p);    //Deallocate memory
    free(u);
    return 0;
}
```

```
The updated time is: 14:22:39
Program ended with exit code: 0
```

- Pointers can also be members in your structures.

```
struct pointertime
{
    int *hour;
    int *min;
    int *sec;
}
```

- You can define a new type

```
typedef struct pointertime PTIMETYPE
```

- You can then define a new structure variable and initialize its members:

```
PTIMETYPE newtime;
int newhour;
newtime.hour = &newhour;
*newtime.hour = 14;
```

- The next program is very similar to the program we previously introduced to define a time structure and update it with a function. The difference is that our structure now contains pointers to an integer in place of integers.

```
#include <stdio.h>

typedef struct    //Define a new structure with pointer members
{
    int *hour;
    int *min;
    int *sec;
} PTIMETYPE;

//Define a new update function with new type. The function behaves the same as old
//update function, but uses pointers.

PTIMETYPE pointupdate (PTIMETYPE now)
{
    ++(*now.sec);

    if ( *now.sec == 60 )
    {
        *now.sec = 0;
        ++(*now.min);

        if ( *now.min == 60 )
        {
            *now.min = 0;
            ++(*now.hour);

            if ( *now.hour == 24 )
            {
                *now.hour = 0;
            }
        }
    }

    return now;
}
```

```
int main()
{
    PTIMETYPE newtime, nexttime;    //Declare structure variables

    int newhour, newmin, newsec;    //Declare variables for storing time

    newtime.hour = &newhour;    //Initialize members of structure variable newtime
    newtime.min = &newmin;
    newtime.sec = &newsec;

    printf ("Enter the time (hh:mm:ss): \n");    //Input time to be stored in
        structure variable newtime
    scanf ("%i:%i:%i", newtime.hour, newtime.min, newtime.sec);

    nexttime = pointupdate(newtime);    //Update time by calling pointupdate function
        and print result
    printf ("Updated time is %0.2i:%0.2i:%0.2i\n", *nexttime.hour, *nexttime.min, *
        nexttime.sec);

    return 0;
}
```

```
Enter the time (hh:mm:ss):
23:59:59
Updated time is 00:00:00
Program ended with exit code: 0
```