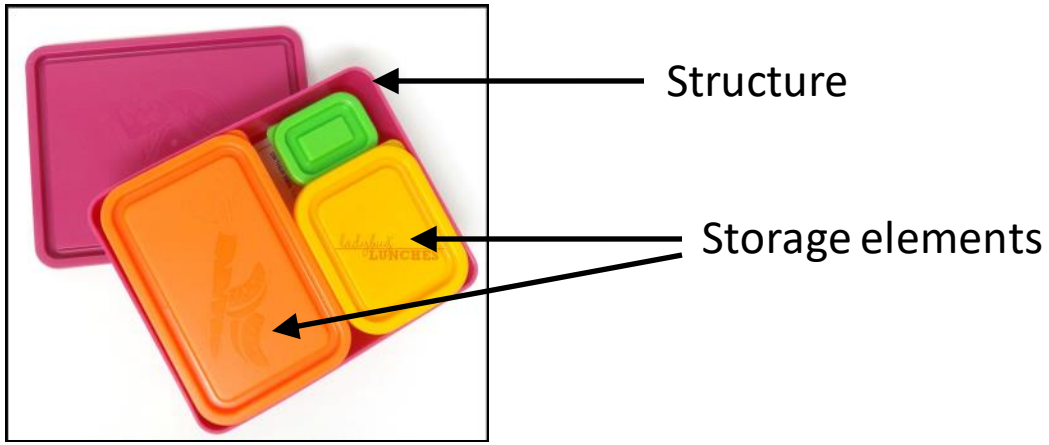
Structures

Outline

- Introduction
- Syntax
  - Defining a structure
  - Accessing members of a structure
- `typedef`
- Syntax
  - Passing structures to a function
  - Returning a structure from a function

1. Introduction

- A ***structure*** is a collection of related storage elements under a single name.



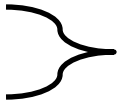
- The elements in a structure can be of different types.

- All elements in a structure typically related semantically.

Image src: http://www.ladybug-lunches.com/2013/07/bento-buddies-review.html

1.1. Defining a Structure Type (*Syntax*)

```
struct struct_name {
  type1 member1;
  type2 member2a, member2b;
  …
  typeN memberN;
};
```



Define what kinds of data to hold.

- A structure type is a defined using the keyword `struct`.
- We need to define a structure type before we can declare variables of that type to store values.

1.1. Defining a Structure Type

- Another example:

```
/* define a NEW type for storing data LATER */
struct student {
    char    id[11];
    char    name[40];
    double  gpa;
};
```

Note the semicolon (;) at the end.

- This defines a new data type called `struct student`, which consists of three *related* ***members***: `id`, `name`, and `gpa`.

1.1. Defining a Structure Type

```
/* define a NEW type for storing data LATER */
struct student {
    char    id[11];
    char    name[40];
    double  gpa;
};
```

Important:

- This defines only a new data type called `struct student`.
- *No* variable storage has YET been allocated.
- This is just the *blueprint* (*design*) of the new type.

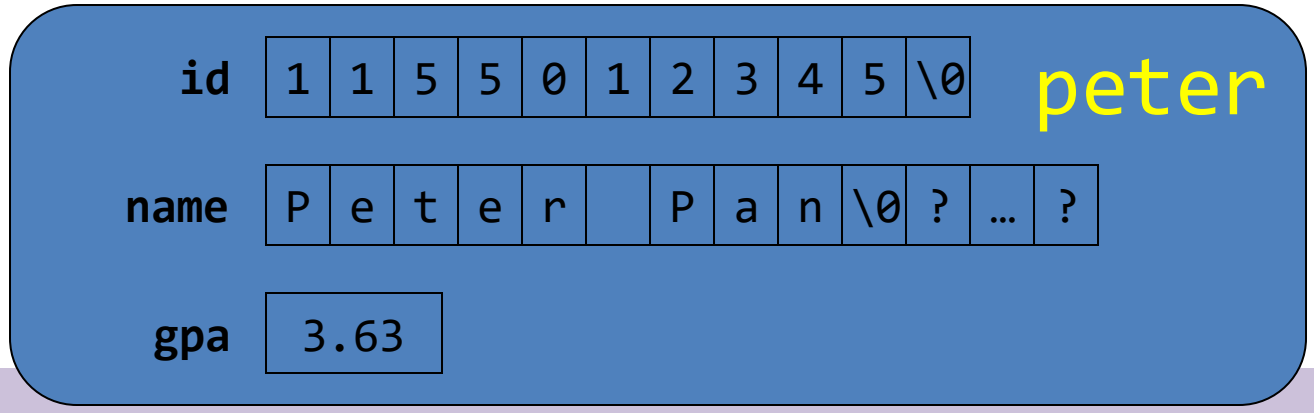
1.1. Defining a Structure Type

- *Structure variables* can then be declared just like variables of simple data types.
- E.g.,

```
struct student  peter;
```

This is the type name.

This is the variable name.



1.1. Altogether: Global Structure Type

- A struct type is usually defined *outside* any functions.

Global scope

Local variable

```
#include <stdio.h>

/* define a NEW type for storing data LATER */
struct student {
    char   id[11];
    char   name[40];
    double gpa;
};

int main() {
    /* a variable declaration using the NEW type*/
    struct student  peter;
    int             no_of_students;
    ...
}
```

1.2. Another Example

```
1   struct date {
2       int day, month, year;
3   };
4
5   int main(void) {
6       struct date d1, d2;
7
8       // Assign 10 to the
9       // member "day" of d1
10      d1.day = 10;
11
12      // Assign 2014 to the
13      // member "year" of d2
14      d2.year = 2014;
15
16      return 0;
17  }
```

Defining a new structure type named `date`.

In this definition, we specify that each "value" of this type contains three members (`day`, `month`, and `year`) which are of type `int`.

1.2. Another Example

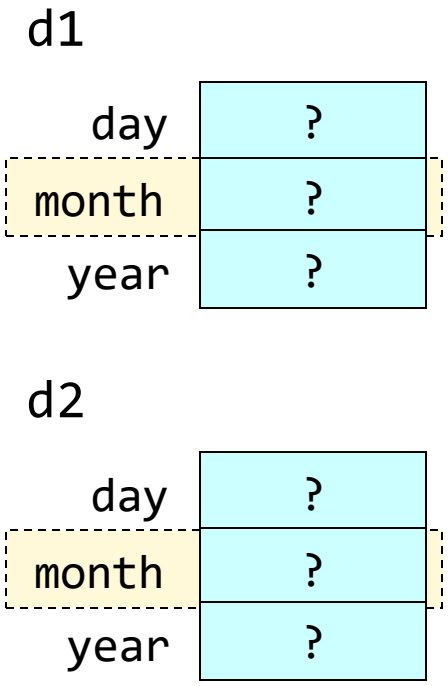
```
1   struct date {
2       int day, month, year;
3   };
4
5   int main(void) {
6       struct date d1, d2;
7
8       // Assign 10 to the
9       // member "day" of d1
10      d1.day = 10;
11
12      // Assign 2014 to the
13      // member "year" of d2
14      d2.year = 2014;
15
16      return 0;
17  }
```

"`struct date`" is the name of the newly defined type.

At line 6, we declare two variables, `d1` and `d2`, of type "`struct date`".

1.2. Another Example

```
1   struct date {
2       int day, month, year;
3   };
4
5   int main(void) {
6       struct date d1, d2;
7
8       // Assign 10 to the
9       // member "day" of d1
10      d1.day = 10;
11
12      // Assign 2014 to the
13      // member "year" of d2
14      d2.year = 2014;
15
16      return 0;
17  }
```

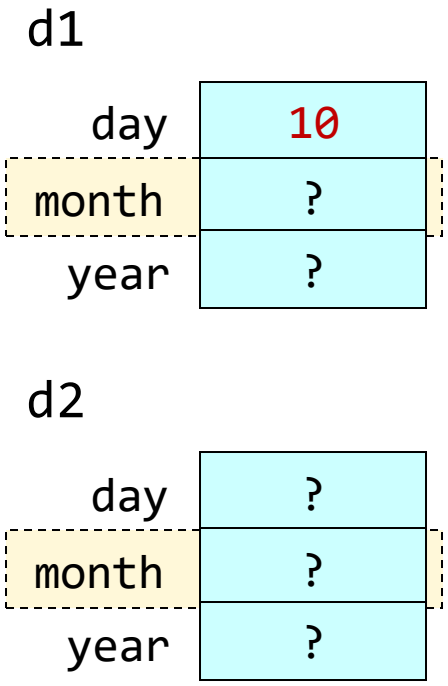


Each variable of type "`struct date`" has its own members.

Initially the members are uninitialized.

1.2. Another Example

```
1   struct date {
2       int day, month, year;
3   };
4
5   int main(void) {
6       struct date d1, d2;
7
8       // Assign 10 to the
9       // member "day" of d1
10      d1.day = 10;
11
12      // Assign 2014 to the
13      // member "year" of d2
14      d2.year = 2014;
15
16      return 0;
17  }
```

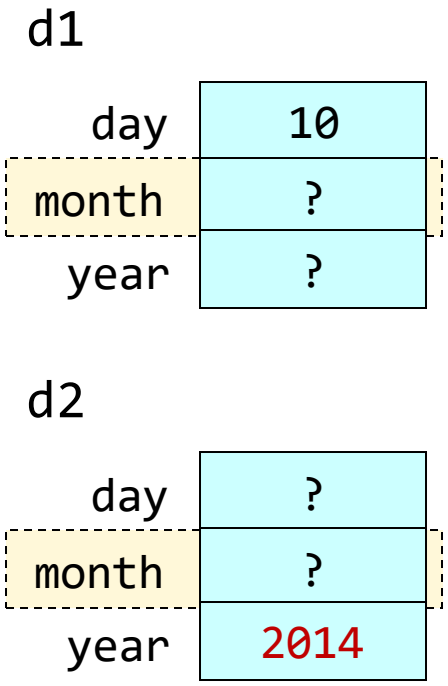


The *dot operator* (`.`) is called a *member selection* operator.

`d1.day` means "select the member `day` of `d1`".

1.2. Another Example

```
1   struct date {
2       int day, month, year;
3   };
4
5   int main(void) {
6       struct date d1, d2;
7
8       // Assign 10 to the
9       // member "day" of d1
10      d1.day = 10;
11
12      // Assign 2014 to the
13      // member "year" of d2
14      d2.year = 2014;
15
16      return 0;
17  }
```



A member of type `int` is just like a regular variable of type `int`. Any syntax that is valid for a variable of type `int` is also valid for a member of type `int`.

1.3. More Structure Examples

- 2D Coordinates

```
struct coord2D {
    double  x;
    double  y;
};
```

- Employee record

```
struct employee {
    char    name[50];
    double  salary[12];
    double  MPF_contrib;
};
```

- Quadratic form Ax^2+Bx+C

```
struct quad_form {
    double  A, B, C,;
};
```

- Polynomial $Ax^n+Bx^{n-1}+\ldots+E$

```
struct polynomial {
    int    degree;
    double coeff[10];
};
```

1.4. Accessing struct Members

- The Member Operator:
  - Members of a struct variable can be accessed using the member operator,

    i.e. the DOT `.`
  - For example,

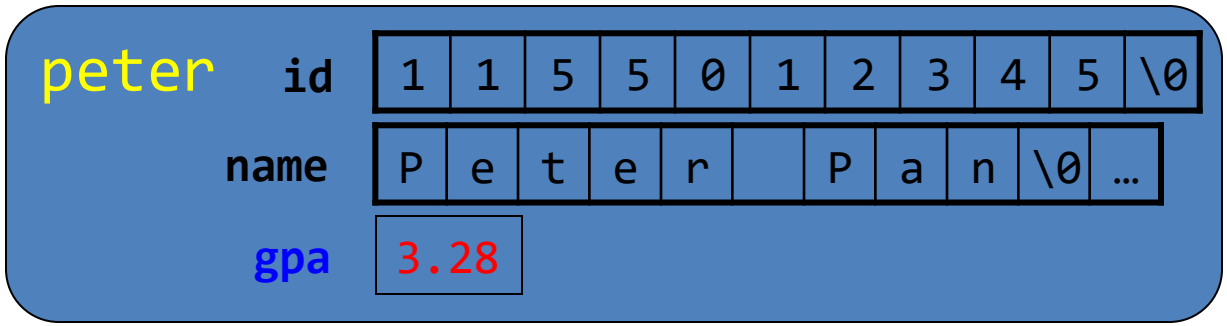
    ```
    variable.member = 999;
    ```

    where `variable` is a `struct` variable and `member` is a defined component of the `struct`.

1.4. Accessing struct Members

- The previous example:

```
…
int main() {
    struct student  peter;
    peter.gpa = 3.28;
    …
}
```



1.4. Accessing `struct` Members (Another Example)

```
1   struct date { int day, month, year; };
2
3   int main(void) {
4      struct date today, dob;   // Declare 2 variables
5
6      today.day = 19;
7      today.month = 11;
8      today.year = 2014;
9
10     printf("Date of birth (dd mm yyyy)? ");
11     scanf("%d%d%d", &dob.day, &dob.month, &dob.year);
12
13     if (today.month > dob.month ||
14         (today.month == dob.month && today.day >= dob.day))
15        printf( "Age = %d\n", today.year - dob.year );
16     else
17        printf( "Age = %d\n", today.year - dob.year - 1 );
18     return 0;
19  }
```

1.5. Initializing Structures

- `struct` variables can be initialized as follows:

```
struct date {
    int day, month, year;
};
int main() {
    struct date today = {25, 12, 1997};
    …
}
```

- If there are not enough initializers, values of the *remaining members are assigned **zero** by default*.

```
struct date   today = {25, 12};
/* today.year is assigned 0 implicitly */
```

1.5. Initializing Structures

- The initializers should be constant values or constant expressions.
  - No variables should be involved in the initializer expressions.

```
struct date {
    int day, month, year;
};
int main() {
    int i = 1997;
    struct date today = { 25, 12, i }; /* Wrong */
    …
}
```

1.5. Initializing Structures (More Examples)

```
struct employee {
  char    name[50];
  double  salary[12];
  double  MPF_contrib;
};

struct employee   peter = {
  "Peter Pan",
  { 9500,  9500, 10000, 11000, 12000, 12050,
   12100, 13000, 13000, 13000, 14000, 14000},
  0.05
};
```

1.6. Assignment of Structures

```
struct date d1, d2 = { 1, 1, 2014 };

d1 = d2;  // Copy d2 to d1 (all data byte by byte)
```

- A struct value can be copied using the assignment operator.

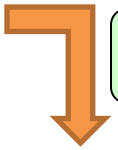
1.6. Assignment of Structures

- The assignment operator = can be applied to structure variables.
- Individual member of the *source structure* is **copied** to the corresponding member of the *target structure*.
- This applies even to array members in the structure, i.e., it also makes a copy of the array member inside the structure.

Structure assignment (COPYING and PASSING) consumes computing TIME, but it's CONVENIENT!

1.6. Assignment of Structures

```
1  #include <stdio.h>
2  #define NAMEMAX 30
3
4  struct person_t {
5    char surname[NAMEMAX+1], forename[NAMEMAX+1];
6    int age;                    /* Two arrays in the structure */
7  };
8
9  int main() {
10   struct person_t computer1;
11   struct person_t computer2 = { "Machine", "Computing", 50 };
12
13   computer1 = computer2;           /* Structure assignment */
14
15   printf("computer 1: %s %s\n", computer1.forename,
16                                  computer1.surname);
17   printf("computer 2: %s %s\n", computer2.forename,
18                                  computer2.surname);
19   return 0;
20 }
```



This is a trick to have changeable array size!

```
computer 1: Computing Machine
computer 2: Computing Machine
```

1.7. Defining an alias to an existing data type

- We can introduce an alias (別名) to an existing data type using `typedef`.

- Syntax

```
typedef existing_type_name alias;
```

- After the declaration, both `alias` and `existing_type_name` refer to the same data type.

1.7. Defining an alias to an existing data type

```c
struct date {
    int day, month, year;
};

int main(void) {
    struct date d1, d2;
    ...
}
```

```c
struct date {
    int day, month, year;
};

typedef  struct date  Date;
// From this point onward,
// "Date" becomes an alias of
// "struct date"

int main(void) {
    Date d1, d2;
    struct date d3;

    // The types of d1, d2, and
    // d3 are the same
    ...
}
```

Do not add the keyword `struct` in the front.

1.7. Combining `typedef` with structure definition

```
struct date {
    int day, month, year;
};

typedef struct date Date;
```

equivalent to

```
typedef struct date {
    int day, month, year;
} Date;
```

almost equivalent to

```
typedef struct {
    int day, month, year;
} Date;
```

Defining a `struct` type named "`date`" and defining an alias to the struct type in two separate declarations.

Defining a `struct` type named "`date`" and defining an alias to the struct type in one declaration.

Defining a `struct` type with no name and defining an alias to the struct type in one declaration.

With the first two approaches, the type can be referred to in the program as "`struct date`" and "`Date`". With the 3rd approach, the type can only be referred to in the program as "`Date`".

1.7. More typedef examples

```
struct student {
  char    id[11];
  char    name[40];
  double  gpa;
};   /* A structure declaration */

typedef  struct student  STUDENT_t;  /* An alias */
...
STUDENT_t  peter; /* Declare variable peter */
```

- Define STUDENT_t as an *alias* to struct student. This allows shorter type names.

- STUDENT_t is also called a *Type-Defined Structure*.

2. Structures and Functions

2.1. Structures as Function Parameters

```
void func( struct rational  rat )
```

2.2. Structures as Function Return Values

```
struct rational  func( )
```

2.3. Structures as BOTH Function Parameters and Function Return Values

```
struct rational  func( struct rational  rat )
```

2.1. Structures as Function Parameters

- Rational (Fractional) Number
  - Form of $\frac{\text{numerator}}{\text{denominator}}$, or simply $\frac{\text{num}}{\text{den}}$, where *num* and *den* are integers.
  - To simplify the example, we assume both *num* and *den* are positive integers.
  - The Greatest Common Divisor (GCD) is usually used to simplify a rational number.
  - We further assume that we have already had a function `gcd()` for finding the GCD of 2 integers.

2.1. Structures as Function Parameters

```c
#include <stdio.h>

struct rational {          /* Define the structure "rational" */
   int num, den;
};

void r_Print( struct rational  rat ) {  // A function for
   printf("%d / %d", rat.num, rat.den); // printing rational
}

double r_Real( struct rational  rat ) { // A function for
   return (double)rat.num / rat.den;    // obtaining real number
}

int main() {
   struct rational  three_over_four = {3, 4};
   double real_num;

   r_Print( three_over_four );        // Function call
   real_num = r_Real( three_over_four );   // Function call
   printf(" = %f\n", real_num);

   return 0;
}
```

```
3 / 4  = 0.750000
```

2.2. Structures as Function Return Values

```
1   …… /* struct definition and r_Print from Program 2.1 */
2
3   struct rational  r_Create( int n, int d ) {
4      struct rational  rat;
5      int              g;
6
7      g = gcd(n, d);   /* Assume existence of gcd() */
8      rat.num = n / g;
9      rat.den = d / g;
10
11     return rat;
12  }
13
14  int main() {
15     struct rational  rat_68 = {6, 8};  /* struct initialization */
16     r_Print( rat_68 );
17     printf(" --> ");
18     struct rational  rat_34;
19     rat_34 = r_Create( 6, 8 );         /* struct assignment */
20     r_Print( rat_34 );
21
22     return 0;
23  }
```

Properly assign the members before returning a `struct` value

```
6 / 8  -->  3 / 4
```

2.3. Structures as BOTH Parameters and Return Values

```
1   … /* struct definition and other functions same as before */
2   typedef  struct rational  Rational;   /* alias */
3
4   Rational  r_Add( Rational rat1, Rational rat2 ) {
5      int num, den;
6      num = rat1.num * rat2.den + rat2.num * rat1.den;
7      den = rat1.den * rat2.den;
8      return r_Create( num, den );
9   }
10
11  int main() {
12     Rational six_over_eight, one_over_two;
13
14     six_over_eight = r_Create( 6, 8 );
15     r_Print( six_over_eight );
16     printf(" + ");
17     one_over_two   = r_Create( 1, 2 );
18     r_Print( one_over_two );
19     printf(" = ");
20     r_Print( r_Add( six_over_eight, one_over_two ) );
21
22     return 0;
23  }
```

$$\frac{n_1}{d_1} + \frac{n_2}{d_2} = \frac{n_1 d_2 + n_2 d_1}{d_1 d_2}$$

```
3 / 4  + 1 / 2  = 5 / 4
```

3. **[Optional]** More Structures

3.1. Array in Structures

3.2. Array of Structures

```
struct student  student_list[100];
```

3.3. Nested Structures

```
struct student {
  char          name[30];
  struct date   dob;
}
```

3.1. Array Members of Structures

```c
#include <stdio.h>
#define NAMEMAX 100

struct person {
   char surname[NAMEMAX+1];
   char forename[NAMEMAX+1];
   int  age;
};

int main() {
   struct person computer = { "Machine", "Computing", 50 };
   struct person user;

   printf("Your surname? ");
   scanf("%s",user.surname);
   printf("Your forename? ");
   scanf("%s",user.forename);
   printf("Dear %c. %s\n", user.forename[0], user.surname);
   printf("I am %s.\n", computer.forename);

  return 0;
}
```

```
Your surname? WAY↵
Your forename? Peter↵
Dear P. WAY
I am Computing.
```

3.1. Array Members of Structures

Entities	Meaning/ Description	Type/ Value
`computer`	The whole structure	`struct person`
`computer.surname`	The whole string member `surname`	`char []` `"Machine"`
`computer.surname[0]`	The 1st character of string `computer.surname`	`char` `'M'`
`computer.forename[4]`	The 5th character of `computer.forename`	`char` `'u'`
`computer.age`	The member age	`int` `50`

```
struct person {
   char surname[NAMEMAX+1];
   char forename[NAMEMAX+1];
   int  age;
};
```

3.2. Array of Structures

- Structures may occur as an array.
- Example:

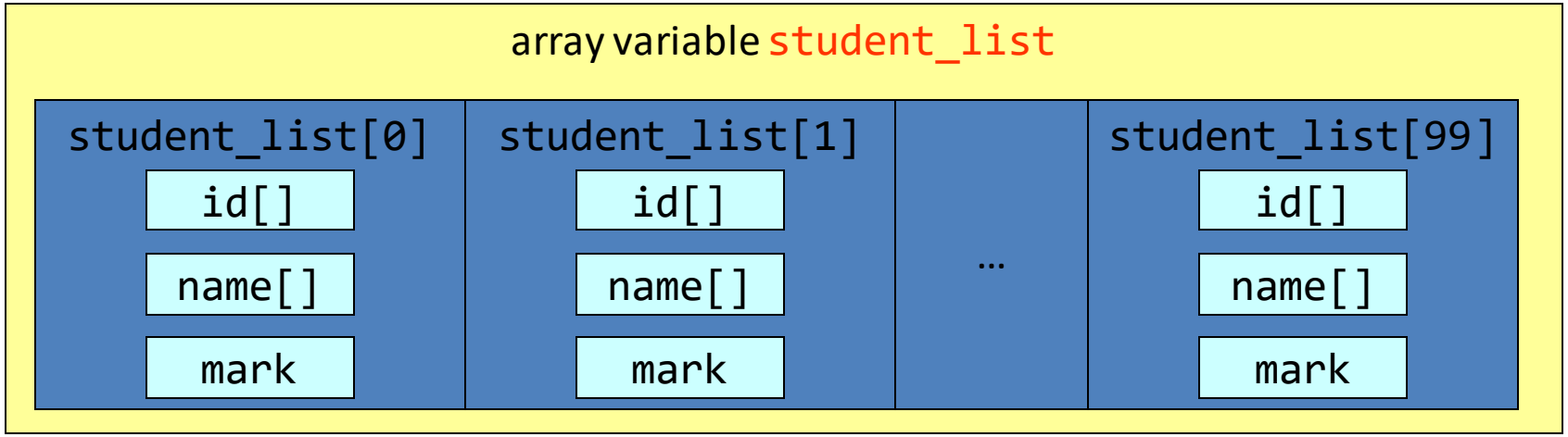
```
struct student {
   char id[11];
   char name[40];
   int  mark;
};

int main(void) {
   struct student   student_list[100];

   student_list[5].mark = 98;

   if ( student_list[6].id[1] == '0' )
      printf("Admitted before 2010/11");
}
```

3.2. Array of Structures



Entities	Meaning/ Description
`student_list`	An array of `struct student`
`student_list[2]`	The 3rd element of the array, a variable of type `struct student`
`student_list[i].name`	The name of the (i+1)th student in the array, a string
`student_list[i].name[j]`	The (j+1)th character of the name of the (i+1)th student in the array, a single `char`

3.3. Nested Structures

```
1  #include <stdio.h>
2  #include <string.h>
3
4  struct date {
5     int  day, month, year;
6  };
7  struct book {
8     char         author[30], title[50], publisher[30];
9     int          edition;
10    struct date  date_of_pub;
11 };
12
13 int main(void) {
14    struct book  booklist[100];
15
16    strcpy( booklist[10].author,    "Al Kelley, Ira Pohl" );
17    strcpy( booklist[10].title,     "C By Dissection" );
18    strcpy( booklist[10].publisher, "Addison-Wesley" );
19
20    booklist[10].edition = 4;
21    booklist[10].date_of_pub.day   = 1;
22    booklist[10].date_of_pub.month = 10;
23    booklist[10].date_of_pub.year  = 2000;
24    ...
25    return 0;
26 }
```

Multiple levels of dots!

Summary

- A structure is a means for a programmer to group related variables into one "container".
- Each member of a structure is like a regular variable. Their main difference is in the syntax.
- Syntax which you should remember
  - Defining a structure (with and without `typedef`)
  - Using structure and accessing members of a structure by the member operator "`.`".
- Structures can be members of other structures. (Nested structures)
- Structure assignment can potentially be expensive.
- Structure values can be passed as function parameters and also returned from functions.

Reading Assignment

- C: How to Program, 8th ed, Deitel and Deitel
- Chapter 10 C Structures
  - Sections 10.1 – 10.4: Structure Basics
  - Section 10.5: Using Structures with Functions
  - Section 10.6: typedef
  - Section 10.7: A Practical Example