Overview
Data types of C
structs within structs
Arrays of structs
 structs containing arrays
Sorting an array of structs

Data Types of C
Simple data types
int, double, char
Atomic chunks of data - cannot be pulled apart into components
Composite data
Arrays
Structs
For many problems, an array or a struct still not sufficient

Composite Data
Arrays
Sequence of variables all of the same type
structs
Collection of fields of possibly different types
Key point: variables of any type can be a component of an array or struct...
including an array or struct!

Nested structs - Example

typedef struct { /* a single point */
    double x, y;
} point;

typedef struct { /* a size */
    double width, height;
} dimension;

typedef struct { /* description of rectangle */
    dimension size;
    point lower_left;
    int line_color, fill_color;
} rectangle;

/* variable declaration */
rectangle r;

Nested struct Layout
Field Selection

Use the . operator to select a field.
If the field is itself a struct, use . again to select its components
r
r.lower_left
r.lower_left.x

QUIZ: Calculating Types

typedef struct {
  double x, y;
} point;
typedef struct {
  double width, height;
} dimension;
typedef struct {
  dimension size;
  point lower_left;
  int line_color, fill_color;
} rectangle;

Structures and Arrays

A struct represents a single record
Typically, computer applications have to deal with collections of such records
Examples: student records, employee records, customer records, parts records
In each case we will have multiple instances of one record (struct) type

Arrays of structs are the natural way to do this

Components in struct Arrays

point pentagon[5];

Arrays in structs

The fields in a struct can themselves be an array
Common example: strings (arrays of char)

#define MAX_NAME 40
typedef struct {
  char name [MAX_NAME+1];
  int id;
  double grade;
  int hw, exam;
} student_record;

Component Access

Given a data structure,
If it's an array, use subscripts ([]) to access an element
If it's a struct, use . to access a field
If the result is itself an array or struct, use . or [] to access components, as appropriate

student_record cse_142[MAX_STUDENTS];
What is student 0's hw?

answer: cse_142[0].hw
Using Arrays of structs

```
using Arrays of structs

student_record class[MAX_STUDENTS] ;
...
/* read student hw and exams and calculate grade */
for ( i = 0 ;  i < nstudents ;  i = i + 1 )
{
    scanf("%d %d", &class[i].hw, &class[i].exam) ;
    class[i].grade =
        (double) (class[i].hw + class[i].exam) / 50.0 ;
}
```

Type Quiz

```
typedef struct {
    char name [MAX_NAME+1];
    int id ;
    double score ;
} StudentRecord ;
StudentRecord a [MAX_STUDENTS];
/*What is the type of each?*/
a        a[0]                           a[5].name
a[4].id           &a[6].score              a[2].name[1]
a.score[0]        StudentRecord[1]
```

Review: structs as Parameters

```
A single struct is passed by value
all of its components are copied from the argument (actual parameter) to initialize the (formal) parameter, even if they are arrays (unless you use pointers explicitly)
point midpoint (point a, point b) {...}
int main (void) {
    point p1, p2, m;            /* declare 3 points */
    ...
    m = midpoint ( p1, p2);    /* declare 3 points */
}
```

Passing Arrays of structs

```
Passing Arrays of structs

An array of structs is an array.
When any array is an argument (actual parameter), it is passed by reference (not copied)
The parameter is an alias of the actual array argument
int avg (student_rec class_db[MAX_N]) {...}
int main (void) {
    student_rec cse_142[MAX_N];
    int average;
    ....
    average = avg ( cse_142 ); /* by reference */
}
```

Sorting Arrays of structs

```
Sorting Arrays of structs

<table>
<thead>
<tr>
<th>Bill 920915 2.9</th>
<th>Will 901028 4.0</th>
<th>Jill 900317 3.9</th>
<th>Phil 920914 2.8</th>
<th>Jill 910607 3.6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phil 920914 2.9</td>
<td>Bill 920915 2.9</td>
<td>Jill 910607 3.6</td>
<td>Gill 900317 3.9</td>
<td>Will 901028 4.0</td>
</tr>
</tbody>
</table>

typedef struct {
    char name [MAX_NAME + 1] ;
    int id ;
    double score ;
} StudentRecord ;
```
Review: Selection Sort

/* Sort b[0..n-1] in non-decreasing order
   (rearrange elements in b so that
   b[0]<=b[1]...<=b[n-1]) */

void sel_sort (int b[], int n)
{
    int k, m;
    for (k = 0; k < n - 1; k = k + 1)
    {
        m = min_loc(b, k, n);
        swap(&b[k], &b[m]);
    }
}

Helper for Selection Sort

/* Find location of smallest element in b[k..n-1] */
/* Returns index of smallest, does not return the
   smallest value itself */
int min_loc (int b[], int k, int n)
{
    int j, pos; /* b[pos] is smallest element */
    pos = k; /* found so far */
    for (j = k + 1; j < n; j = j + 1)
    {
        if (b[j] < b[pos])
        {
            pos = j;
            return pos;
        }
    }
    /* Interchange values */
    void swap (int * x, int * y);
}

Modifying for Array of StudentRecord

1. Decide which field to sort by: the
   "sort key"
   Let's sort by score
2. Change array types to StudentRecord
3. Change comparison to pull out sort
   key from the structs
4. Write a "swap" for StudentRecord

Selection Sort Modified

/* Sort b[0..n-1] in non-decreasing order
   (rearrange elements in b so that
   b[0]<=b[1]...<=b[n-1]) */

void sel_sort (StudentRecord b[], int n)
{
    int k, m;
    for (k = 0; k < n - 1; k = k + 1)
    {
        m = min_loc(b, k, n);
        swap(&b[k], &b[m]);
    }
}

Selection Sort Helper Modified

/* Sort b[0..n-1] in non-decreasing order
   (rearrange elements in b so that
   b[0]<=b[1]...<=b[n-1]) */

int min_loc (StudentRecord b[], int k, int n)
{
    int j, pos; /* b[pos] is smallest element */
    pos = k; /* found so far */
    for (j = k + 1; j < n; j = j + 1)
    {
        if (b[j].score < b[pos].score)
        {
            pos = j;
            return pos;
        }
    }
    /* Interchange values */
    void swap (StudentRecord * x, StudentRecord * y);
}

Alphabetical Order

typedef struct
{
    char name[MAX_NAME + 1];
    int id;
    double score;
} student_record;

Need a function to compare two strings!
Review: String Comparison

“Alice” is less than “Bob”
“Dave” is less than “David”
“Rob” is less than “Robert”

```c
#include <string.h>
int strcmp (char str1[], char str2[])
returns negative integer if str1 is less than str2
      0 if str1 equals str2
      positive integer if str1 is greater than str2
```

Modified to Sort by Name

The only change from sorting by score is in the function min_loc

```c
int min_loc (StudentRecord b[], int k, int n) {
    int j, pos;     /* b[pos] is smallest element */
    pos = k;        /* found so far */
    for (j = k + 1; j < n; j = j + 1)
        if (0 > strcmp(b[j].name, b[pos].name))
            pos = j;
    return pos;
}
```

Data Structures: What If...

...you wanted to keep information about one song on the computer.
What pieces of data would you want?
How would you organize them?
How would it look in C?
And then...
What if you wanted information about an entire CD of songs?
And then... how about a whole collection of CD’s?

Summary

- Arrays and structs can be combined and nested – to any level
- The separate rules for arrays and structs are followed – even when the two ideas are combined
- 2-D arrays and strings can be used, too
- An infinite number of data structures can be created! – design a structure appropriate to a particular programming problem