University of Washington Computer Programming I

Lecture 20: Nested Data Structures

Overview

Data representation in C

Arrays of structs

structs containing arrays

Sorting an array of structs

Data Representation in C

Simple data types
- int, double, char
- Atomic chunks of data - cannot be pulled apart into components

Composite data
- Arrays
- Structs

Composite Data

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!

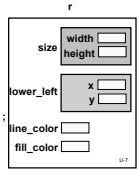
Nested *structs* - Example

```
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 ;
```


Nested *struct* Layout

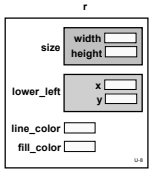
```
typedef struct {
    double  x, y ;
}  point ;
typedef struct {
    double  width, height ;
} dimension ;
typedef struct {
    dimension size ;
    point          lower_left ;
    int            line_color, fill_color ;
} rectangle ;
```

/* variable declaration */
rectangle r;



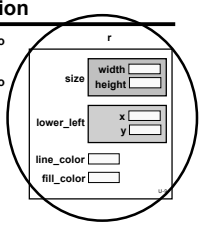
Field Selection

Use the . operator to select a field.
If the field it itself a struct, use . again to select its components



Field Selection

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

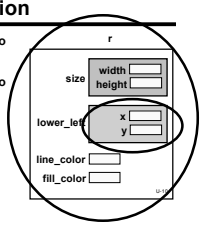


Field Selection

Use the . operator to select a field.
If the field it itself a struct, use . again to select its components

r

r.lower_left



U-11

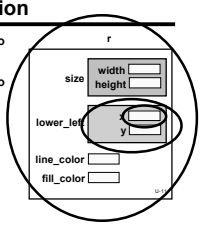
Field Selection

Use the . operator to select a field.
If the field it itself a struct, use . again to select its components

r

r.lower_left

r.lower_left.x



U-12

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 ;
```

U-12

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 ;
```

```
rectangle R;
rectangle * rp;
```

U-13

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 ;
```

```
rectangle R;
rectangle * rp;

R.size
R.lower_left
R.fill_color
R.lower_left.x
&R.lower_left.y
rp -> size
&rp -> lower_left
*rp.line_color
R  -> size
rp -> size -> width
```

U-14

Structures and Arrays

U-15

Structures and Arrays

A *struct* represents a single record

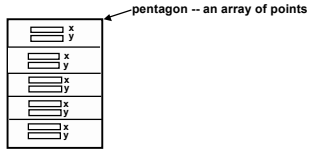
Typically, computer applications have to deal with collections of such records

- 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

U-16

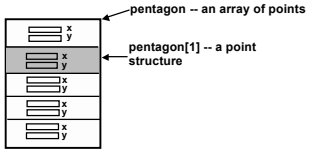
Components in *struct* Arrays



point pentagon[5];

U-17

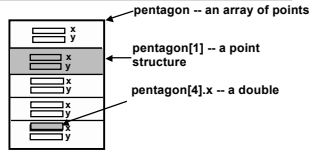
Components in *struct* Arrays



point pentagon[5];

U-18

Components in *struct* Arrays



point pentagon[5];

U-19

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  score ;
} student_record ;
```

U-20

Component Access

```
student_record cse_142[MAX_STUDENTS];
```

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

U-21

Using Arrays of *structs*

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

U-22

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);
}
```

U-23

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*

Bill	Will	Gill	Phil	Jill
920915	901028	900317	920914	910607
2.9	4.0	3.9	2.8	3.6

U-25

Sorting Arrays of *structs*

Bill	Will	Gill	Phil	Jill
920915	901028	900317	920914	910607
2.9	4.0	3.9	2.8	3.6

Phil	Bill	Jill	Gill	Will
920914	920915	910607	900317	901028
2.8	2.9	3.6	3.9	4.0

U-26

Sorting Arrays of *structs*

Bill	Will	Gill	Phil	Jill
920915	901028	900317	920914	910607
2.9	4.0	3.9	2.8	3.6

Phil	Bill	Jill	Gill	Will
920914	920915	910607	900317	901028
2.8	2.9	3.6	3.9	4.0

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

U-27

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(&a[k], &b[m]);
   }
}
```


Helper for Selection Sort

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 */
```


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;
}
```


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

Modifying for Array of StudentRecord

- Decide which field to sort by: the "sort key"
  - Let's sort by *score*
- Change array types to StudentRecord
- Change comparison to pull out sort key from the structs
- Write a "swap" for StudentRecord

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] ) */

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(&a[k], &b[m]);
   }
}
```


Selection Sort Modified

Selection Sort Modified

```
/* Find location of smallest element in b[k..n-1] */
/* Returns index of smallest, does not return the
    smallest value itself */
```


Selection Sort Modified

```
/* Find location of smallest element in b[k..n-1] */
/* Returns index of smallest, does not return the
    smallest value itself */
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;
}
```


Selection Sort Modified

```
/* Find location of smallest element in b[k..n-1] */
/* Returns index of smallest, does not return the
    smallest value itself */
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

Phil	Harry	Susan	David	Sarah
920914	910607	901028	920915	900317
2.8	3.6	4.0	2.9	3.9

Alphabetical Order

Phil	Harry	Susan	David	Sarah
920914	910607	901028	920915	900317
2.8	3.6	4.0	2.9	3.9

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


Need a function to compare two strings!

Alphabetical Order

Phil	Harry	Susan	David	Sarah
920914	910607	901028	920915	900317
2.8	3.6	4.0	2.9	3.9

David	Harry	Phil	Sarah	Susan
920915	910607	920914	900317	901028
2.9	3.6	2.8	3.9	4.0

```
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"

#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

Modified to Sort by Name

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

Modified to Sort by Name

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

```
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;
}
```


Modified to Sort by Name

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

```
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;
}
```


Type Quiz

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


Type Quiz

```
typedef struct {
    char name [MAX_NAME+1];
    int id ;
    double  score ;
} StudentRecord ;
StudentRecord a [MAX_STUDENTS];
```


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]	

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 where appropriate, too

An infinite number of data structures can be created, each one appropriate to a particular programming problem