CSE / ENGR 142 Programming I

Structures

Chapter 11

Read 11.1-11.3, 11.5, & 11.7

- 11.1: Structure types
- 11.2: Structures as parameters
- 11.3: Structures as return values
- 11.5: Arrays of structures

Optional examples; skim or read:

- 11.4: Complex numbers

Heterogeneous Structures

- Collection of values of possibly differing types.
- Name the collection; name the components.
- Example: student record

```
harvey
   name   "Harvey S."
   id     9501234
   hw     87
   exams  74
   grade  3.1
```

C expressions:

harvey.hw is 87

harvey.name is "Harvey S."

*2*harvey.exams* is 148

Defining *structs*

```
#define MAX_NAME  40
typedef struct {  /* typedefs go at the top of the program */
   char   name [MAX_NAME + 1] ;
   int    id ;
   int    hw, exams ;
   double grade ;
} student_record ;
```

Defines a new data type called student_record. Does not declare (create) a variable. No storage is allocated.

Terminology

- A "*struct*" is sometimes called a "record" or "structure".
- Its "components" are sometimes called "fields" or "members".

Structs are the basis of classes in C++ and Java, which are the fundamental building-blocks for object-oriented programming.

Declaring *struct* Variables

```
… /*typedef structs go at top of program */
…
int     i1 ;          /* int decls. and initializers */
int     count = 0 ; /*nothing new */
char    c1[5] ;       /*array decls.  */

student_record   s1 ;
student_record   harvey ;
```

/*student_record is a type; *s1* and *harvey* are variables. */

Things You Can and Can't Do

- You can
  - use = to assign whole *struct* variables
- You can
  - have a *struct* as a function return type
- You can't
  - use == to directly compare *struct* variables; can compare fields directly
- You can't
  - directly *scanf* or *printf structs*; can read fields one-by-one

struct initializers

```
… /*typedef structs go at top*/

int     i1 ;          /* int decls. and initializers */
int     count = 0 ; /*nothing new */
char    c1[5] ;       /*array decls. and initializers */
char    pet[5] = "lamb" ;      /*string initializer*/

student_record   harvey = {"Harvey S.", 9501234,
                           87, 74, 3.1} ;
```


Using Components of *struct* Variables

```
student_record s1;
s1.hw     = 90 ;
s1.exams= 80 ;
s1.grade = (double)(s1.hw + s1.exams) / 50.0 ;
printf("%d: %f", harvey.id, harvey.grade) ;
scanf("%d", &s1.id) ;
```


Assigning Whole *structs*

```
s1 = harvey ;
   equivalent to

s1.id      = harvey.id ;
s1.hw      = harvey.hw ;
s1.exams     = harvey.exams ;
s1.grade= harvey.grade ;
strncpy(s1.name, harvey.name, MAX_NAME + 1) ;
   /* string copy -- we'll talk about this shortly */
```


Why use *structs*?

- Collect together values that are treated as a unit (for readability, maintainability).

```
typedef struct {
   int  dollars, cents ;
} money ;
```

```
typedef struct {
   int hours, minutes ;
   double  seconds ;
} time ;
```

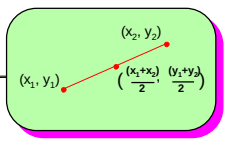
- Functions can return *structs*
  - another way to have multiple return values.
- Files and databases: collections of *structs* (e.g., 200 student records).

Recall Midpoint Example

```
/* Given 2 endpoints of a line, "return" coordinates of midpoint */

void set_midpoint(
   double x1, double y1,          /* 1st endpoint      */
   double x2, double y2,          /* 2nd endpoint      */
   double *midxp, double *midyp ) /* Pointers to midpoint */
{
   *midxp = (x1 + x2) / 2.0;
   *midyp = (y1 + y2) / 2.0;
}

double ax, ay, bx, by, mx, my;
…
set_midpoint(ax, ay, bx, by, &mx, &my);
```



Points as *structs*

```
typedef struct {
    double  x, y ;
} point ;
...
point  a = {0.0, 0.0},  b = {5.0, 10.0} ;
point  m ;
m.x = (a.x + b.x) / 2.0 ;
m.y = (a.y + b.y) / 2.0 ;
```


Midpoint function via *structs*

```
point midpoint (point  pt1,  point  pt2)
{
    point  mid;
    mid.x = ( pt1.x + pt2.x ) / 2.0 ;
    mid.y = ( pt1.y + pt2.y ) / 2.0 ;
    return (mid);
}

point  a = {0.0, 0.0}, b = {5.0, 10.0}, m;
                    /* struct declaration and initialization */

m = midpoint (a, b) ;              /* struct assignment */
```

structs passed by value (copied)

Midpoint with pointers

```
void set_midpoint (point  pt1,  point  pt2, point  *mid)
{
    (*mid).x = ( pt1.x + pt2.x ) / 2.0 ;
    (*mid).y = ( pt1.y + pt2.y ) / 2.0 ;
}                          /* . has high precedence than * */

point  a = {0.0, 0.0}, b = {5.0, 10.0}, m;
set_midpoint (a, b, &m) ;
```

- Structs behave like all non-array types when used as parameters.

Pointer Shorthand: ->

```
void set_midpoint (point  pt1,  point  pt2, point  *mid)
{
    mid->x = ( pt1.x + pt2.x ) / 2.0 ;
    mid->y = ( pt1.y + pt2.y ) / 2.0 ;
}
```

structp - > component means (*structp).component

-> is called the "indirect component selection operator"

Testing Equality of *structs*

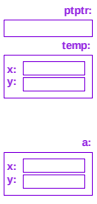
```
if (pt1 = = pt2) { ... }     /* Doesn't work */
```

```
int points_equal(point pt1, point pt2)
{
    return (pt1.x = = pt2.x   &&
            pt1.y = = pt2.y ) ;
}
```

```
if (points_equal(pt1, pt2)) { ... }   /* OK */
```


Do-it-yourself *struct* I/O

```
void print_point (point p)  {
    printf ("(%f,%f)", p.x, p.y) ;
}
void scan_point (point *ptptr)  {
    point temp ;
    scanf ("%lf %lf", &temp.x, &temp.y) ;
    *ptptr = temp ;
}
```




```
point a ;
scan_point (&a) ;
print_point (a) ;
```

Alternative *scan_point*

```
void scan_point (point *ptptr)  {
    scanf ("%lf %lf",  & ptptr->x,  & ptptr->y) ;
}
```

ptptr:

```
point a ;
scan_point (&a) ;
print_point (a) ;
```

a:
x:
y:

scan_point without ->

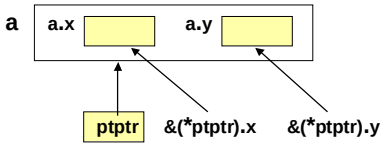
```
void scan_point (point *ptptr)  {
    scanf ("%lf %lf", &(*ptptr).x, &(*ptptr).y) ;
}
```

ptptr:

```
point a ;
scan_point (&a) ;
print_point (a) ;
```

a:
x:
y:

Pointers and *structs*



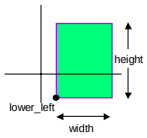
expression	type
ptptr	point *
*ptptr	point
(*ptptr).x	double
&(*ptptr).x	double *

Hierarchical *structs*

```
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 ;
```



Using *structs* within *structs*

```
point origin = { 0.0, 0.0 } ;
rectangle  a = { {5.0, 10.0}, {1.0, -2.0}, BLUE, CYAN } ;
rectangle  b ;

b.fill_color    = BLUE ;
b.lower_left    = origin ;          /* place at origin */
b.lower_left.y = 15.0 ;             /* move up 15      */
...
b.size.width  = 2.0 * b.size.width ;   /* stretch in x   */
b.size.height = 4.0 * b.size.height ;  /* stretch in y   */
```


QUIZ: Calculating Types

```
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
```