structs

C Programming and Software Tools
N.C. State Department of Computer Science
The Derived Data Types

- Arrays
- Pointers
- Structs
  - ( Enums )
  - ( Unions )
**structs**

- Example: a person has multiple attributes
  - name
  - weight
  - height
  - gender
  - ID number
  - age
  - etc.

- To indicate these are all part of the same entity, we define a `struct` data type for persons
Declaring Structure Tag

- Makes more sense than simply defining these fields individually, not indicating how they are related.
Compared with Java

• members of a **struct** in C are very similar to instance fields of a **class** in Java
  – but there is no access specifier (**public**, **private**) for members of a **struct** (i.e., they are all **public**)

• Syntax for referring to both is the same

```c
struct person person1;
person1.height = 72;
person1.weight = 180;
person1.gender = 'M';
...
```
Declaring structs

- A struct **may** have a name
- A struct **may** have instances declared when it’s defined
- An instance **may** be initialized with values
- A struct **may** have members
  (but if it doesn’t, you’re probably dumb)
Example declarations

```c
#define LEN 32

struct student {
    char name[LEN];
    float gpa;
};

struct {
    char name[LEN];
    float gpa;
} bob;

struct student {
    char name[LEN];
    float gpa;
} bob="Bob Studentface", 3.5;  
```

- Named struct
- Anonymous struct with one instance
- Named struct with one instance, initialized.
**structs in Memory**

- **struct members** stored in memory in order declared
- Each member is allocated the amount of memory appropriate to its type
- Members are in same memory block
  - May be offsets

<table>
<thead>
<tr>
<th>name</th>
<th>height</th>
<th>weight</th>
<th>gender</th>
<th>idnum</th>
<th>age</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**struct Name Space**

- A **struct** is a new scope
- Two different **structs** can have members with the same names

```c
struct person {
    char name[LEN];
    int weight;
    int height;
    ...
};
```

```c
struct student {
    char name[LEN];
    char class;
    int creditHours;
    ...
};
```

No conflict!
Initializing Named structs

Uninitialized

```c
struct person person1;
```

Fully initialized

```c
struct person person1 =
    {"Fred", 72, 180, 'M', 12345, 20};
```

Partly initialized (version 1)

```c
struct person person1 =
    {"Fred", 72, 180, 'M'};
```
Partly initialized (version 2)

```c
struct person person1 =
    {.name = "Fred",
     .height = 72,
     .gender = 'M',
     .idnum = 12345};
```
Exercise 16a
Hello, Struct!

• Declare a struct named position with integer members x, y, and z.

• Write the statement to initialize the struct to contain the coordinates (2,5,-3).

• Print the position to the console with the format “(%d,%d,%d)”.

Reminder: Go to course web page for link to exercise form. Paste code into ideone.com and submit the link.
Referring to structs and members

Simple assignment to a struct member

```c
person3.weight = 200;
```

Assignment to an entire struct (version 1)

```c
person2 = person1;
```

Assignment to an entire struct (version 2)

```c
person4 = (struct person)
    { "Mary", 
      66, 
      125, 
      'F', 
      98765, 
      21 };
```

If setting a struct after it’s declared, you need to cast the braced stuff to the correct struct type.
structs can contain structs

One struct...

struct date {
   unsigned short month;
   unsigned short day;
   unsigned int year;
};

Contained in another struct...

struct person-with-start {
   struct date start;
   char name[LEN];
   int height;
   int weight;
   char gender;
   int idnum;
   short age;
   ...
};
structs can contain... (cont'd)

Referencing a struct within a struct

```c
struct person-with-start p1;
...
p1.start.month = 8;
p1.start.day = 16;
p1.start.year = 2009;
```
Exercise 16b

Structs with structs

• Create a struct named box with members:
  – itemnum (int),
  – color (char * or char[25]),
  – p (struct position),
  – height, width, and depth (all ints).

• Write a statement to initialize a struct with values of 3 for itemnum, “red” for color, (1,2,3) for position, 3 for height, 2 for width, and 5 for depth.

• Print the strict with the format:
  “Item #%d (%s) POS=(%d,%d,%d) DIMS=(%d,%d,%d)”
Arrays of structs

Example

```c
... int main () {
    struct person persons[100];
    persons[1] = getstruct(“Liz”);
    (persons[2]).idnum = 23456;
    ...
}
```

Parentheses needed? \textbf{No.}
## Reminder: C Operator Precedence

<table>
<thead>
<tr>
<th>Tokens</th>
<th>Operator</th>
<th>Class</th>
<th>Prec.</th>
<th>Associates</th>
</tr>
</thead>
<tbody>
<tr>
<td>a[k]</td>
<td>subscripting</td>
<td>postfix</td>
<td>16</td>
<td>left-to-right</td>
</tr>
<tr>
<td>f(...)</td>
<td>function call</td>
<td>postfix</td>
<td></td>
<td>left-to-right</td>
</tr>
<tr>
<td>.</td>
<td><strong>direct selection</strong></td>
<td>postfix</td>
<td>16</td>
<td>left-to-right</td>
</tr>
<tr>
<td>-&gt;</td>
<td><strong>indirect selection</strong></td>
<td>postfix</td>
<td></td>
<td>left to right</td>
</tr>
<tr>
<td>++ --</td>
<td>increment, decrement</td>
<td>postfix</td>
<td></td>
<td>left-to-right</td>
</tr>
<tr>
<td>(type){init}</td>
<td>literal</td>
<td>postfix</td>
<td></td>
<td>left-to-right</td>
</tr>
<tr>
<td>++ --</td>
<td>increment, decrement</td>
<td>prefix</td>
<td></td>
<td>right-to-left</td>
</tr>
<tr>
<td>sizeof</td>
<td>size</td>
<td>unary</td>
<td></td>
<td>right-to-left</td>
</tr>
<tr>
<td>~</td>
<td>bit-wise complement</td>
<td>unary</td>
<td></td>
<td>right-to-left</td>
</tr>
<tr>
<td>!</td>
<td>logical NOT</td>
<td>unary</td>
<td></td>
<td>right-to-left</td>
</tr>
<tr>
<td>- +</td>
<td>negation, plus</td>
<td>unary</td>
<td></td>
<td>right-to-left</td>
</tr>
<tr>
<td>&amp;</td>
<td>address of</td>
<td>unary</td>
<td></td>
<td>right-to-left</td>
</tr>
<tr>
<td>*</td>
<td>Indirection (dereference)</td>
<td>unary</td>
<td></td>
<td>right-to-left</td>
</tr>
</tbody>
</table>
Arrays of... (cont'd)
Example of an array of structs, each containing an array of structs...

```c
struct phonenumber {
    short areacode;
    short exchange;
    short number;
    char type;
};
struct person {
    ...
    struct phonenumber pno[4];
};
struct person persons[MAXPERSONS];
```
Initializing Arrays of structs

Example

```c
struct person persons[100] = {
    { "Fred", 72, 180, 'M', 0, 20 },
    { "Liz", 63, 115, 'F', 33333, 19 },
    { "Mary", 76, 180, 'F', 44444, 25,
        {{919, 515, 2044, 'W'},
         {919, 555, 6789, 'H'}}
    },
    [10] = {.name = "Bill", .height = 70,
            .gender = 'M'}
};
```
Referencing Arrays of structs

```c
if (persons[4].pno[2].areacode == 919)
...
```
Exercise 16c

Array of structs

- Declare an array of 100 boxes.
- Initialize a box at indexes 0 and 1 (your choice of values)
- Console output optional
structs as Input Parameters

void printname ( struct person );

int main() {
    struct person person1 = {...};
    printname(person1);
    ...
}

void printname( struct person p )
{
    printf("Name: %s\n", p.name);
}

Structs are passed by value, as usual
    - i.e., a copy is made and passed to the function
structs as Return Values

- (finally!) The answer to how functions can return multiple results
  - one struct (with multiple members) = one result
structs as Return Values

```c
struct person getstruct(char * name) {
    struct person new;
    new.name = name;
    printf("Enter height and weight for %s: ", name);
    scanf("%d %d", &new.height, &new.weight);
    return (new);
}

int main() {
    ...
    struct person person1 = getstruct("Bob");
    ...
}
```

Parentheses needed? No.
Exercise 16d

Return a struct

• Write a function which when given two structs (box), returns the one with the greater volume (but with position at 0, 0, 0 and color = “green”).

Reminder: Go to course web page for link to exercise form. Paste code into ideone.com and submit the link.
structs Can Contain Pointers

```c
struct person {
    char *name;
    ...
} person1;

person1.name = "Donna";
printf(“Name is %s\n”, person1.name);
char initial = *person1.name;
```

Parentheses needed? No.

Be careful when assigning string values from another function.
Pointers to Structs

```c
struct person {
    ...
} person1, *p;

p = &person1;

(*p).name = "Donna";
(*p).height = 65;
printf("Name is %s\n", (*p).name);
char initial = *(*p).name;
printf("Height is %d\n", (*p).height);
```

Parentheses needed? **YES.**
Does it suck that I need parentheses? **ALSO YES.**
Wouldn’t it be cool if I didn’t need them? **TOTALLY.**
A New Operator

• Unfortunately, \(*p\).height != (*p).height
  the value pointed to by
  the member pp.height
  the height of the person
  pointed to by pp

• A new operator (for convenience):
  \((*a) . b\) can be replaced by \(a->b\)

... 

\(\text{p} = &\text{person1;}\)
\(\text{p->name} = \text{“Donna”;}\)
\(\text{p->height} = 65;\)
\(\text{printf(“Name is %s\n”, p->name);}\)
\(\text{char initial} = *\text{p->name;}\)
\(\text{printf(“Height is %d\n”, p->height);}\)
A New Operator... (cont’d)

• How about pointer to a struct containing pointer to a struct containing...? No problem!

```c
struct person {
    ...
    struct person *father;
    struct person *mother;
} persons[100], *p;
p = &persons[1];
p->father = &persons[22];
p->mother = &persons[45];

if ( p->father->age >= 65 )
    ...
printf("Mother: %s\n", p->mother->name);
```

Parentheses needed? No.
Exercise 16e

Pointers and structs

• Write a function that given two pointers to box structs, will update the one with the greater volume to position 0, 0, 0, and color “green”. The function should be void.

• Write a main that:
  – Creates two box structs with reasonable test values
  – Prints all the members of both structs
  – Calls your function
  – Prints all the members of both structs again
Any Questions?