structs

C Programming and Software Tools

N.C. State Department of Computer Science

Computer Science
NC STATE UNIVERSITY

The Derived Data Types

- ✓ Arrays
- ✓ Pointers
- ➤ Structs
- (Enums)
- (Unions)

Computer Science
NC STATE UNIVERSITY

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

```
struct person {
   char name[LEN];
   int height;
   int weight;
   char gender;
   int idnum;
   short age;
   …
};
struct person
      persons[MAXP];
```

```
char *name[MAXP];
int height[MAXP];
int weight[MAXP];
char gender[MAXP];
int idnum[MAXP];
short age[MAXP];
   …
```

- 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

```
struct person person1;
person1.height = 72;
person1.weight = 180;
person1.gender = 'M';
…
```


Declaring Structs

```
struct {
   char name[LEN];
   int height;
   int weight;
   char gender;
   int idnum;
   short age;
   …
} person1, person2;
```

Unnamed **`struct`**

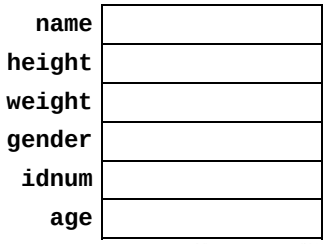
`struct` variables

```
struct {
   char name[LEN];
   int height;
   int weight;
   char gender;
   int idnum;
   short age;
   …
} person1 = {"Bob", 70,
185, 'M', 5, 27},
person2 = {…};
```

Initialized **`struct`** variables

`structs` in Memory

- **`struct`** *members* stored in memory in order declared
- Each member is allocated the amount of memory appropriate to its type
- There are no gaps between members in memory



Computer Science NC STATE UNIVERSITY

`struct` Name Space

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

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

No conflict!

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


Computer Science NC STATE UNIVERSITY

Initializing Named `structs`

Unitialized

```
struct person person1;
```

Fully initialized

```
struct person person1 =
   {“Fred”, 72, 180, ‘M’, 12345, 20};
```

Partly initialized (version 1)

```
struct person person1 =
    {“Fred”, 72, 180, ‘M’};
```


…Initializing (cont’d)

Partly initialized (version 2)

```
struct person person1 =
   {.name = “Fred”,
    .height = 72,
    .gender = ‘M’,
    .idnum = 12345};
```

Referring to **structs** and members

Simple assignment to a **struct** member

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

Assignment to an entire **struct** (version 1)

```
person2 = person1;
```

Assignment to an entire **struct** (version 2)

```
person4 = (struct person)
    {“Mary”,
      66,
      125,
     ‘F’,
      98765,
      21};
```

Computer Science
NC STATE UNIVERSITY

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

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


Arrays of `structs`

Example

```
…
int main () {
   struct person persons[100];

   persons[1] = getstruct("Liz");
   persons[2] = getstruct("Jim");
   (persons[2]).idnum = 23456;
   …
}
```

Parentheses needed?

Reminder: C Operator Precedence

Tokens	Operator	Class	Prec.	Associates
`a[k]`	**subscripting**	**postfix**	16	left-to-right
`f(...)`	function call	postfix		left-to-right
`.`	**direct selection**	**postfix**		left-to-right
`->`	indirect selection	postfix		left to right
`++ --`	increment, decrement	postfix		left-to-right
`(type){init}`	literal	postfix		left-to-right
`++ --`	increment, decrement	prefix	15	right-to-left
`sizeof`	size	unary		right-to-left
`~`	bit-wise complement	unary		right-to-left
`!`	logical NOT	unary		right-to-left
`- +`	negation, plus	unary		right-to-left
`&`	**address of**	**unary**		right-to-left
`*`	Indirection (*dereference*)	unary		right-to-left

Arrays of… (cont'd)

Example of an array of structs, each containing an array of structs…

```
struct person {
   …
   struct phonenumber pno[4];
};
struct person persons[MAXPERSONS];
```

```
struct phonenumber {
    short areacode;
    short exchange;
    short number;
    char type;
};
```

Initializing Arrays of `struct`s

Example

```
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 `struct`s

```
if (((persons[5]).pno[2]).areacode == 919)
   …
```

Parentheses needed?

`struct`s as Input Parameters

```
void printname ( struct person );

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

void printname ( struct person p )
{
    (void) 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

CSC230: C and Software Tools © NC State Computer Science Faculty

Computer Science NC STATE UNIVERSITY

structs as Return Values

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

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

Parentheses needed?

structs Can Contain Pointers

```
struct person {
   char *name;
   …
} person1;

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

Parentheses needed?

Be careful when assigning string values from another function.

Pointers to Structs

```
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?

common source of bugs: failure to use parens around (*p).m

A New Operator

common source of bugs: failure to use parens around (*p).m

- Unfortunately, `*p.height` != `(*p).height`

  `*p.height`: the value pointed to by the member pp.height

  `(*p).height`: the height of the person pointed to by pp

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

```
…
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);
```

*What does * dereference?*

A New Operator… (cont'd)

- How about pointer to a **`struct`** containing pointer to a **`struct`** containing…? No problem!

```
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?