structs

ITSC 2181: Introduction to Computer Systems
UNC Charlotte
College of Computing and Informatics

**COLLEGE OF COMPUTING
AND INFORMATICS**

`struct`s

- 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

COLLEGE OF COMPUTING
AND INFORMATICS

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

COLLEGE OF COMPUTING AND INFORMATICS

Declaring Structs

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

Unnamed **struct**

struct variables

Initialized **struct** variables

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

COLLEGE OF COMPUTING AND INFORMATICS

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

name	
height	
weight	
gender	
idnum	
age	

COLLEGE OF COMPUTING
AND INFORMATICS
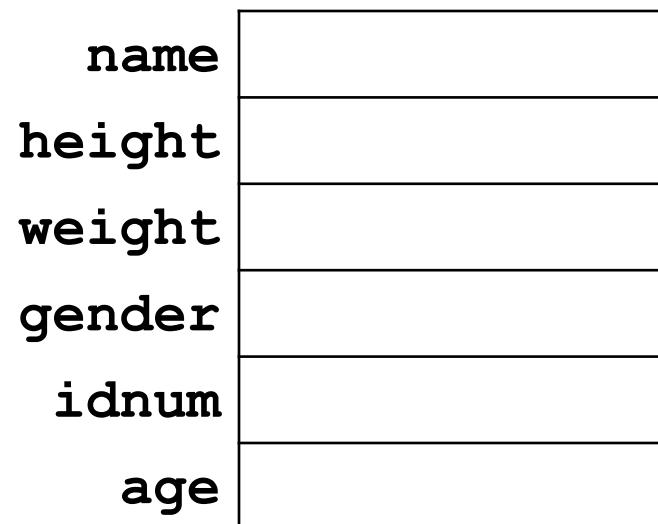
struct Name Space

- 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;
    …
};
```

COLLEGE OF COMPUTING
AND INFORMATICS

Initializing Named **structs**

Unitialized

```
struct person person1;
```

Fully initialized

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

Partially initialized (version 1)

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

(see **struct_initialization.c** in *Code samples and Demonstrations* in *Canvas*)

COLLEGE OF COMPUTING AND INFORMATICS

…Initializing (cont'd)

Partially initialized (version 2)

```
struct person person1 =
    {.name = "Fred",
     .height = 72,
     .gender = 'M',
     .idnum = 12345};
```

(see `struct_initialization.c` in *Code samples and Demonstrations* in *Canvas*)

COLLEGE OF COMPUTING AND INFORMATICS

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};
```

This code uses a **compound literal**.

COLLEGE OF COMPUTING AND INFORMATICS

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;
    …
};
```

COLLEGE OF COMPUTING
AND INFORMATICS

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;
```

COLLEGE OF COMPUTING
AND INFORMATICS

Arrays of **struct**s

Example

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

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

    …
}
```

Are parentheses needed?
No

(see **struct_array1.c** in *Code samples and Demonstrations* in *Canvas*)

COLLEGE OF COMPUTING AND INFORMATICS

Reminder: C Operator Precedence

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

LEGE OF COMPUTING
INFORMATICS

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;
};
```

COLLEGE OF COMPUTING AND INFORMATICS

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'}
};
```

(see `inventory.c` in *Code samples and Demonstrations* in *Canvas*)

COLLEGE OF COMPUTING AND INFORMATICS

Referencing Arrays of `struct`s

```
if (((persons[4]).pno[2]).areacode == 919)
    …
```
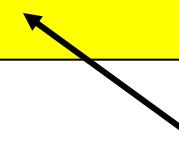
Are parentheses needed?

No

(see **struct_array2.c** and **inventory.c**
in *Code samples and Demonstrations* in *Canvas*)

**COLLEGE OF COMPUTING
AND INFORMATICS**

structs 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

COLLEGE OF COMPUTING
AND INFORMATICS

structs as Return Values

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

COLLEGE OF COMPUTING
AND INFORMATICS

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");
    …
}
```

Are parentheses needed? No

(see **struct_return.c** in *Code samples and Demonstrations* in *Canvas*)

COLLEGE OF COMPUTING AND INFORMATICS

structs Can **Contain** Pointers

```
struct person {
    char *name;

    …
} person1;


person1.name = "Donna";
printf("Name is %s\n", person1.name);
char initial = *person1.name;
```
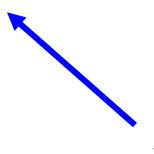
Are parentheses needed? No

Be careful when assigning string values from another function.
Use the **strcpy()** or **strncpy()** function.

COLLEGE OF COMPUTING
AND INFORMATICS

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 );
```

(see `struct_pointer1.c` in *Code samples and Demonstrations* in *Canvas*)

Are parentheses needed?
Yes!

☠ *common source of bugs* ☠
failure to use parenthesis around `(*p).m`

COLLEGE OF COMPUTING AND INFORMATICS

A New Operator: ->

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

  the value pointed to by
  the member **p.height**

  the height of the person
  pointed to by **p**

  ☠ *common source of bugs* ☠
  **failure to use parenthesis
  around (*p).m**

- 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);
```

(see **struct_pointer2.c** in *Code samples
and Demonstrations* in *Canvas*)

*What does *
dereference?*

COLLEGE OF COMPUTING
AND INFORMATICS

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?

COLLEGE OF COMPUTING AND INFORMATICS

References

- S. J. Matthews, T. Newhall and K. C. Webb, *Dive into Systems*, Version 1.2. Free online textbook, available at: https://diveintosystems.org/book/

- K. N. King, *C Programming: A Modern Approach*, 2nd Edition. W. W. Norton & Company. 2008.

- D.S. Malik, *C++ Programming: From Problem Analysis to Program Design*, Seventh Edition. Cengage Learning. 2014.

COLLEGE OF COMPUTING AND INFORMATICS