CIS212 Lab Week #6

Creating Your Own Data Type – Structs & Debugging Structs

UNIVERSITY OF OREGON

CIS212 Lab Week #6

Creating Your Own Data Type – **Structs** & Debugging Structs

UNIVERSITY OF OREGON

What is a Struct?

- A struct is a user defined data type.
  - "*Composite*" data type
  - Allows the user to group items of different types into one type.
  - Why would we want that?

- Just imagine having to do project 2G without structs!

Uses of Structs in Functions

- There would be no point in creating a struct if you don't use it in your functions!
- Some good ways to know whether you truly need a struct:
  - Are the attributes the same type?
  - Are you essentially making a **Dictionary** entry?
  - Do you want to make your life easier?

Defining a Struct (Option 1)

```
struct <StructName>
{
  <type1> var1;
  <type2> var2;
        .
        .
        .
};
```

- One possible way to define a struct! (Not the one Hank uses in 2G. Why?)

Defining a Struct (Option 2)

```
typedef struct
{
  <type1> var1;
  <type2> var2;
        .
        .
        .
} <StructName>;
```

- Allows you to define this `struct` as a type and use it as a type, without having to say `struct` all the time.

Some Examples!

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <stdbool.h>
4
5  int main()
6  {
7
8      typedef struct
9      {
10         int UOid;
11         char name[100];
12         int year;
13         bool graduated;
14     } Undergrad;
15
16     return 0;
17 }
```

What is the scope of the struct?

Some Examples!

```c
#include <stdio.h>
#include <stdlib.h>

int main()
{

    typedef struct
    {
        char name[50];
        char street[100];
        char city[50];
        char state[20];
        int zip;
    } address;

    return 0;
}
```

Defining a Struct

- Two primary ways to define a struct.

Option #1

```
5       struct Point
6       {
7           int x, y;
8       };

10      struct Point p1;
```

Option #2

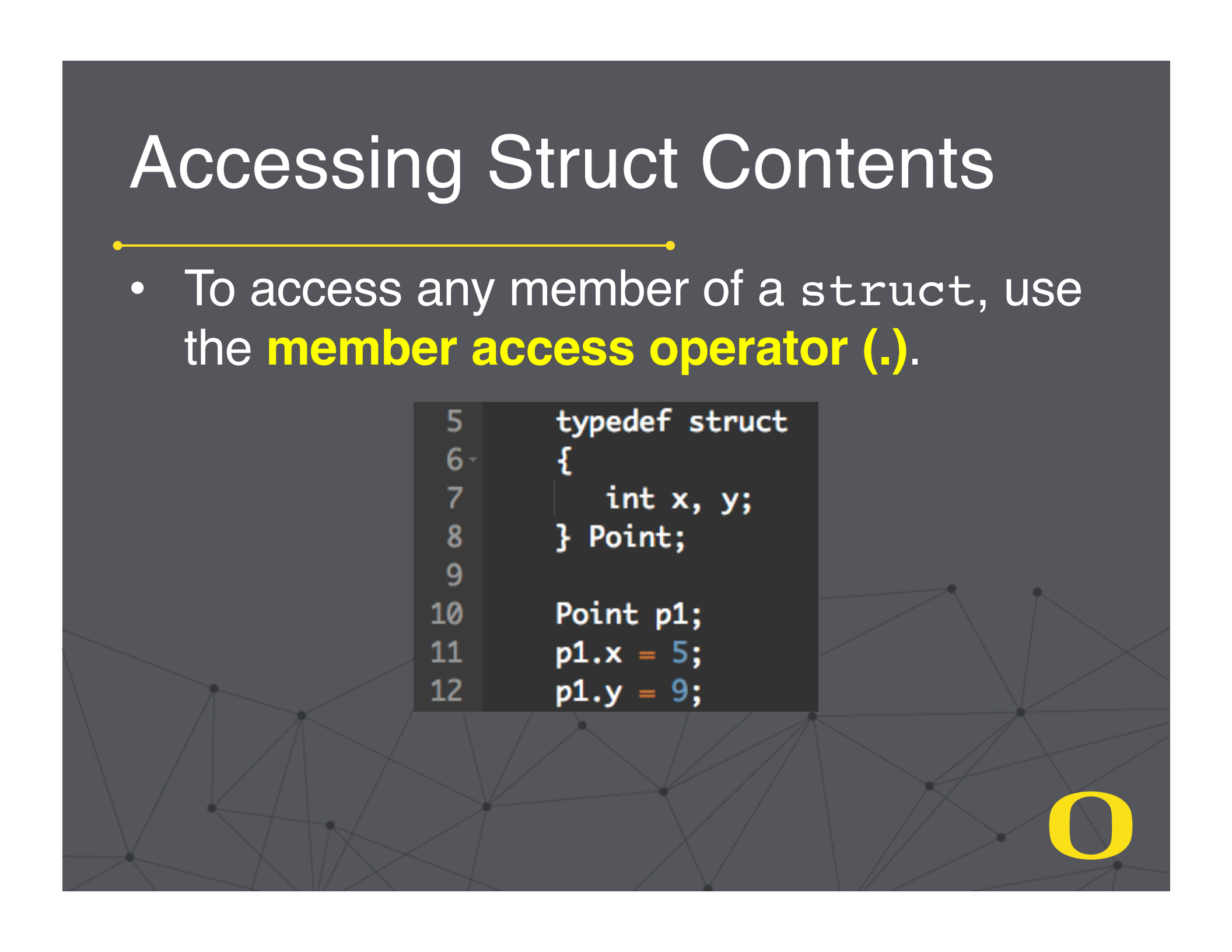
```
5       typedef struct
6       {
7           int x, y;
8       } Point;

10      Point p1;
```

Option #2 is generally the better approach

Accessing Struct Contents

- To access any member of a `struct`, use the **member access operator (.)**.

```
5     typedef struct
6     {
7         int x, y;
8     } Point;
9
10    Point p1;
11    p1.x = 5;
12    p1.y = 9;
```



Pointing to a Struct

- We can create a pointer to a struct.

```
10      Point p1 = {1, 2};
11      Point *p2 = &p1;
```

- Also requires some special symbols to access the members! (Why?)

```
12
13      printf("%d %d", p2->x, p2->y);
14
```

Assignment and Equality with Structs

```
#include <stdio.h>
/* Define a type point to be a struct with integer members x, y */
typedef struct {
    int x;
    int y;
} Point;

int main() {
    Point p = { 1, 3 };
    Point q;
    /* Set q "equal" to p */
    q = p;

    /* Check struct equality */
    if (p == q)
        printf("p equals q\n");

    /* Change the member x of q to have the valu
    q.x = 3;
    if (p.x != q.x)
        printf("The members are not equal! %d != %d", p.x, q.x);
}
```

Copies that **value** of all fields

No equality operator for `structs`

How C Stores a `struct` in Memory

```
typedef struct
{
    <type1> var1;
    <type2> var2;
          .
          .
          .
} <StructName>;
```

- A single, contiguous piece of memory
- Determining the size (in bytes) of a `struct`:

```
sizeof(<StructName>) = sizeof(type1) + sizeof(type2) + …
```

- *No storage overhead*

Tricks (& Treats!)

- Assign values all at once at declaration:

```
5       typedef struct
6       {
7           int x, y, z;
8       } Point;
9
10      Point p1 = {.y = 0, .z = 1, .x = 2};
11      Point p2 = {.x = 20};
```

- Create an array of structs:

```
10      Point arr[10];
11
12      arr[0].x = 10;
13      arr[0].y = 20;
```

Let's Play a Game...

Pick the right type

- I will name something I am trying to represent in my program.
- You tell me the type you would use to represent it
- Valid Choices: struct, primitive, array of <X> objects, etc.

Picking the Right Type

- Book

  `struct`

- Bookshelf

  Array of `Book` objects

- Array index

  `unsigned int`

- Library

  `struct`

CIS212 Lab Week #6

Creating Your Own Data Type – Structs & **Debugging Structs**

UNIVERSITY OF OREGON

Seriously, GDB Again?

- Despite the fact that structs are useful and make life easier, a lot can go wrong fast.
- Possible issues you may encounter:
  - Segmentation faults
  - Not populating of the `struct`'s fields
  - Writing to the wrong field
- So, time for some GDB again!

Quick Review of Commands

- Start your (loaded) program
  `run <args>` or `r <args>`
- Restart the program's execution
  `r` then `y`
- Step (into) line by line (*your code only*)
  `step` or `s`
- Step (over) line by line
  `next` or `n`
- Run to the next breakpoint
  continue or `c`
- Step (out) of the function, i.e., run to the function's end:
  `finish` (not `f` though)

Additional GDB Commands

- Display the current line and function and the stack of calls

  `where`

- Print the value of a variable

  `print <expr>` or `p <expr>`

- Change a variable's value

  `print <varName> = <newValue>`

GDB with Structs

```
5     typedef struct
6     {
7         int x, y;
8     } Point;
9
10    Point p1;
11    p1.x = 5;
12    p1.x = 9;
13
14    Point* p2 = &p1;
```

- We can print out the values of all fields in the struct:

  `print p1` or `p p1`

  {x = 5, y = 9}

- The instance of the `struct` is accessed via a pointer, the GDB command changes slightly:

  `print *p2` or `p *p2`

GDB with Structs

```
5     typedef struct
6     {
7         int x, y;
8     } Point;
9
10    Point p1;
11    p1.x = 5;
12    p1.x = 9;
13
14    Point* p2 = &p1;
```

- Printing the value of a specific field in the `Struct`:

  `print p1.x` or `p p2->y`

- Changing the specific field's value:

  `print p1.x = 10` or `p p2->y = 6`

Other GDB Commands

- Print the `struct` definition of the struct:

```
ptype <TypeName>
```

- Get information about a variable including :
  - Variable type including whether it is a pointer
  - List of `struct` fields, including each field's types and values

```
explore <varName>
```

Start the Exercise…

Do not use `printf` to answer the lab questions. Use solely GDB and `man`.

UNIVERSITY OF OREGON