SYSC 2006
Fall 2017

Carleton
UNIVERSITY
Canada's Capital University

C Structures

- some examples adapted from "The C Programming Language", Kernighan & Ritchie

Last edited by C.-H. Lung, Sept. 25, 2019

What is a Structure?

- One or more variables grouped together under a single name
  - variables can have different types
- Allows a group of related variables to be treated as a unit
  - Able to deal with a group at the same time

Structure Declarations

- Example: a 2-D point

```
struct point {
    int x;
    int y;
};
```

- `x` and `y` are the *members* of the structure
- A structure declaration doesn't allocate memory
  - `point` is a *tag*, not a variable name
  - The key word *struct* is needed.

Structure Instances

- A structure declaration defines a **type**

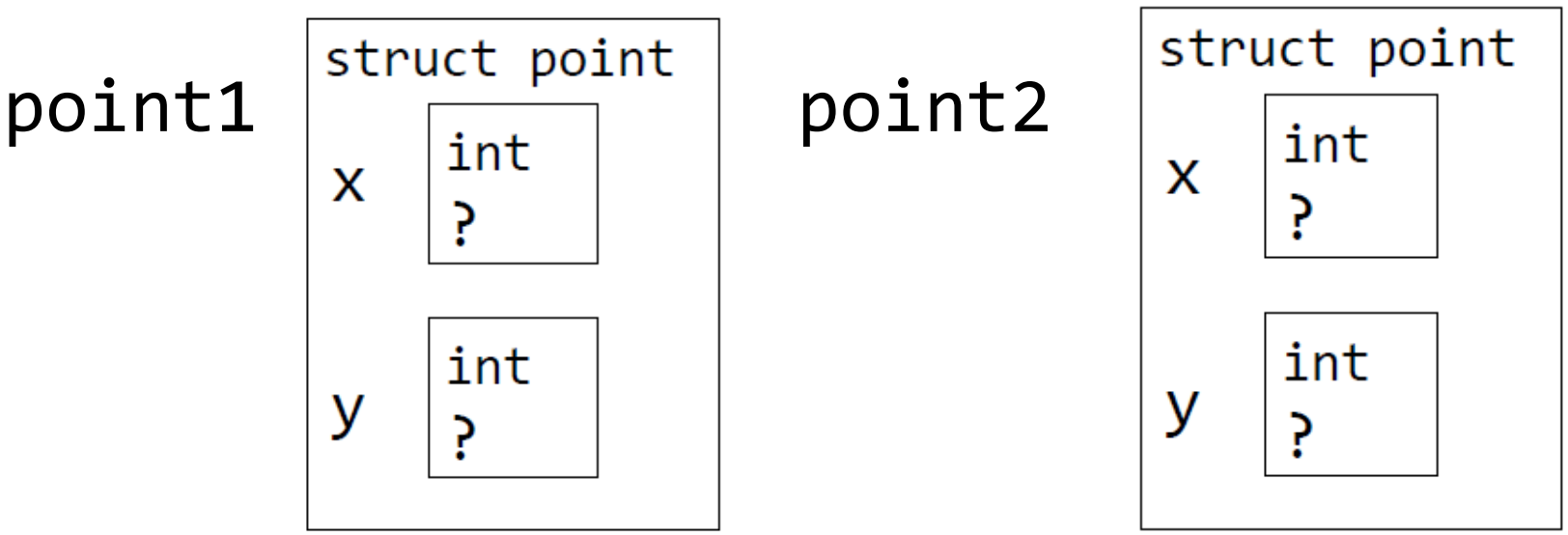
- Given the declaration of `point` (see previous slide), the statement

```
struct point point1, point2;
```

declares variables `point1` and `point2` as *instances* of type `struct point`

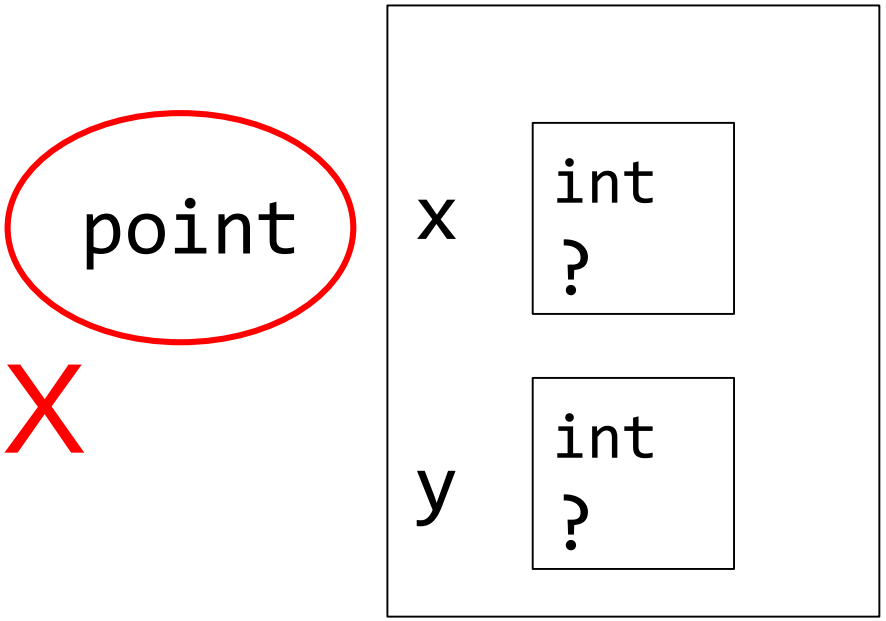
Memory Diagram

- Visualize the two variables this way
  - C Tutor uses a similar notation



Memory Diagram

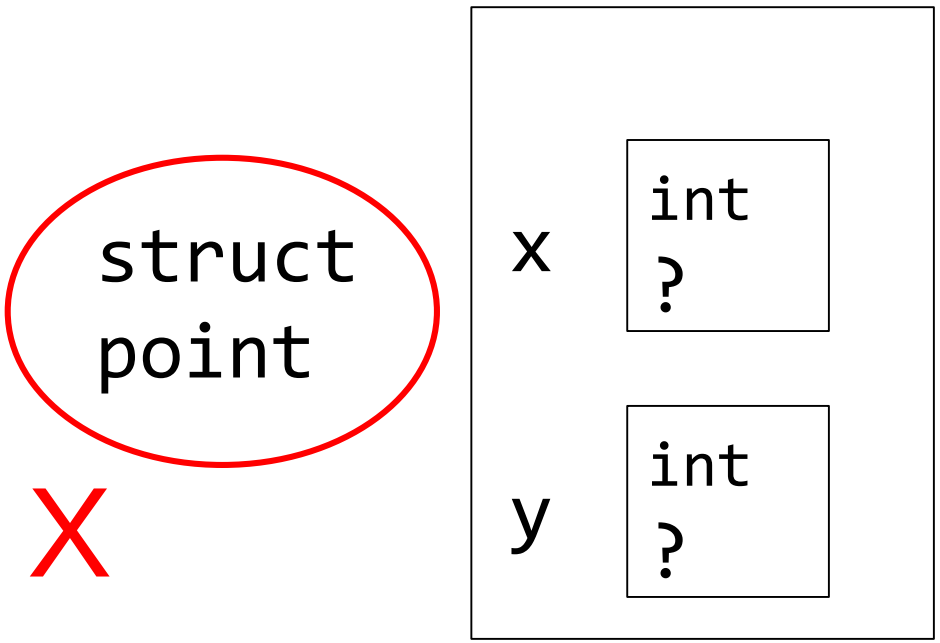
- Don't visualize the structures this way:



- `point` is a structure tag, not a variable name

Memory Diagram

- Don't visualize the structures this way:

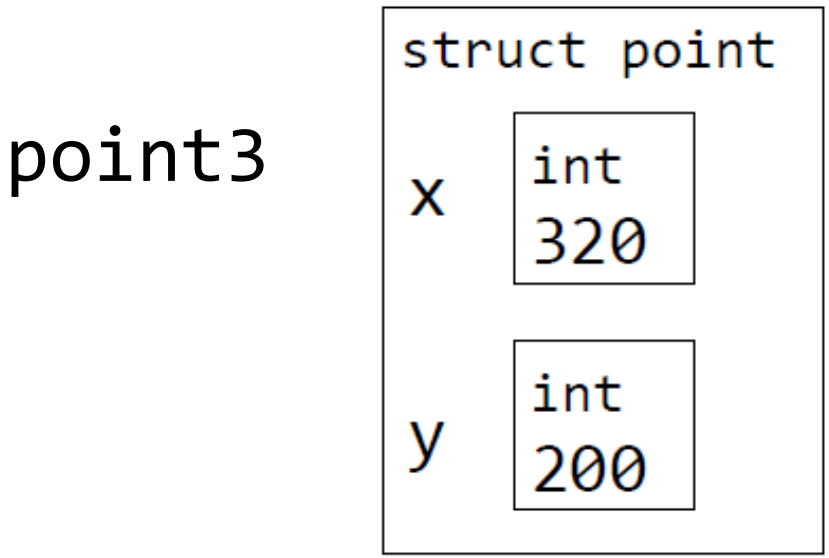


- `struct point` is a type, not a variable name
- `struct` declarations don't allocate memory

Initialization

- Members of a structure can be initialized with constant expressions as part of the variable declaration

```
struct point point3 = {320, 200}; // OK
```



Initialization

- Structure members can also be initialized after the variable declaration
- This doesn't work:

```
struct point point4;
point4 = {320, 200};  // No
```

- Must ***cast*** the expression `{320, 200}` to type `struct point`

```
point4 = (struct point) {320, 200};  // OK
```

typedef

- This statement:

```
typedef struct point point_t;
```

  declares `point_t` as a **synonym** for `struct point`

- It does not declare a variable `point_t` of type `struct point`

typedef

- You can use `point_t` anywhere you would use `struct point`

- Examples:

```
point_t point1, point2;
point_t point3 = {320, 200};
point_t point4;
point4 = (point_t) {320, 200};
```

typedef

- We often combine the structure and `typedef` declarations into a single declaration:

```
typedef struct point {
    int x;
    int y;
} point_t;

point_t point1, point2;
point_t point3 = {320, 200};
point_t point4;
point4 = (point_t) {320, 200};
```

typedef

- When we do this, we can eliminate the structure tag:

```
typedef struct {
    int x;
    int y;
} point_t;

point_t point1, point2;
point_t point3 = {320, 200};
point_t point4;
point4 = (point_t) {320, 200};
```

Structure Operations

- Structure members can be accessed individually
- Structures can be copied and assigned
- Structures can be passed as arguments to functions (pass by value semantics)
- Structures can be returned from functions
- The address of a structure can be calculated (more about this when we cover pointers)

Accessing Members

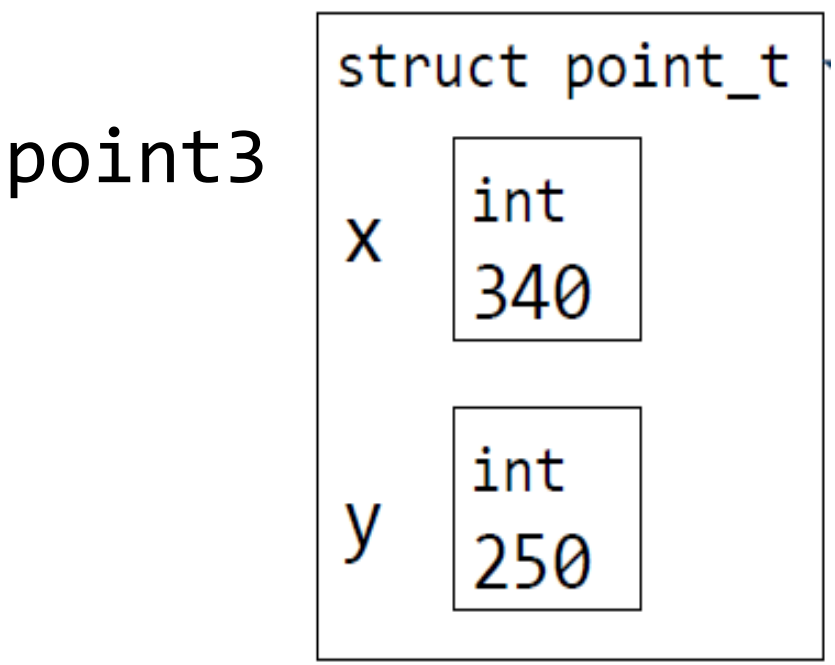
- To access members of a structure, use the dot operator:

```
instance_name.member_name
```

Accessing Members

- Example: move `point3` by 20 units along the x axis and 30 units along the y axis:

```
point3.x = point3.x + 20;
point3.y += 30; // op = works with
                // struct members
```



Notation: `point_t` would be suffice because it is a `typedef`, but C Tutor uses `struct point_t` when it depicts structures.

Accessing Members

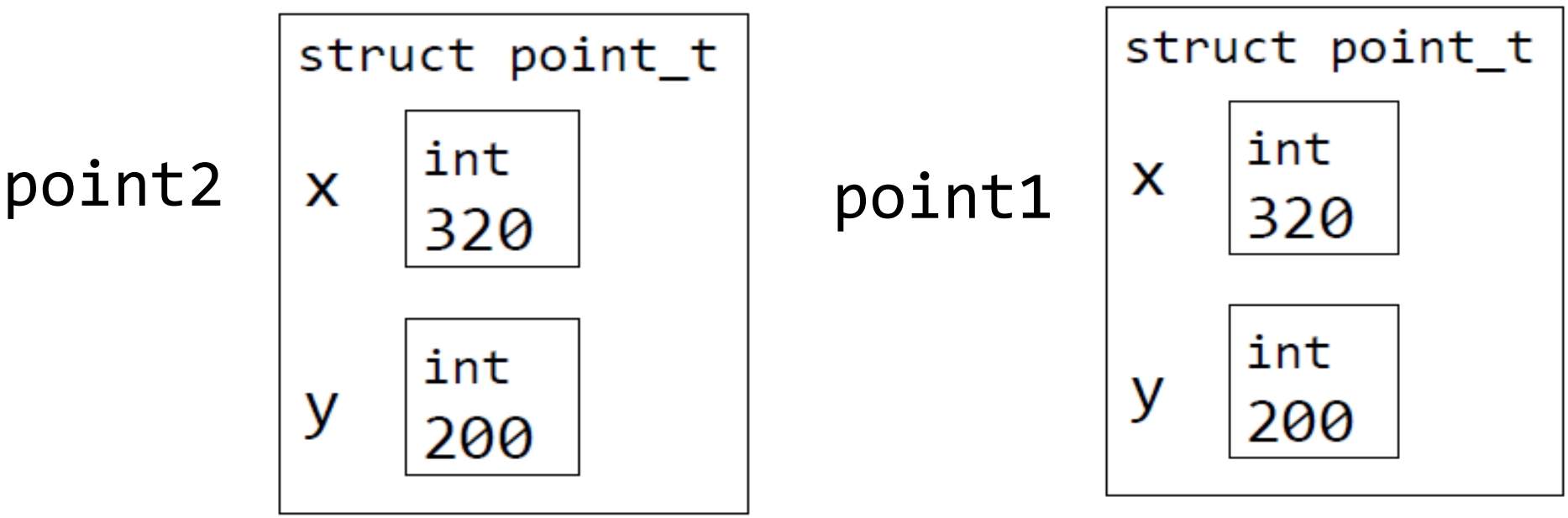
- The [ ] operator cannot be used to access members by name or position

```
point1[x] = 200;    // No!
point1[0] = 200;    // No!
```

Structure Assignment

```
point1.x = 320;
point1.y = 200;
point2 = point1;
```

- After `point2 = point1;` is executed:



Structures and Functions

- A function that takes 2 integers and returns an initialized `point_t` structure

```
point_t makepoint(int x, int y)
{
    point_t temp;

    temp.x = x;
    temp.y = y;
    return temp;
}
```

Structures and Functions

- Typical call:

```
int a, b;
point_t point1;
...
a = 320;
b = 200;
...
point1 = makepoint(a, b);
```

Memory Diagram Exercise

- Use C Tutor to visualize the execution of a program that calls `makepoint`

- Make sure you understand the program's activation frames
  - just before `return temp;` is executed
  - after the statement

    ```
    point1 = makepoint(a, b);
    ```

    is executed

Structures and Functions

- A more concise implementation of `makepoint`

```
point_t makepoint(int x, int y)
{
    return (point_t) {x, y};
}
```

Structures and Functions

- A function that is passed two `point_t` structures and returns their sum

```
point_t addpoints(point_t pt1,
                  point_t pt2)
{
    point_t temp;
    temp.x = pt1.x + pt2.x;
    temp.y = pt1.y + pt2.y;
    return temp;
}
```

Structures and Functions

- We can eliminate local variable `temp`

```
point_t addpoints(point_t pt1,
                  point_t pt2)
{
    pt1.x = pt1.x + pt2.x;
    pt1.y = pt1.y + pt2.y;
    return pt1;
}
```

- Modifying `pt1` does not modify the corresponding argument - why?

Structures and Functions

- Typical call:

```
int a = 320;
int b = 200;

point_t point1 = makepoint(a, b);
point_t point2 = makepoint(30, 40);
point_t sum = addpoints(point1, point2);
```

Memory Diagram Exercise

- Use C Tutor to visualize the execution of a program that calls `addpoints`
- Make sure you understand the program's activation frames
  - just before `return pt1;` is executed
  - after the statement

    ```
    sum = addpoints(point1, point2);
    ```

    is executed

Structures and Functions

- A more concise implementation of `addpoints`

```
point_t addpoints(point_t pt1,
                  point_t pt2)
{
    return (point_t) {pt1.x + pt2.x,
                      pt1.y + pt2.y};
}
```

Structures and Functions

- Yet another implementation of `addpoints`
  - calls `makepoint` to build & return the structure containing the sum

```
point_t addpoints(point_t pt1,
                  point_t pt2)
{
    return makepoint(pt1.x + pt2.x,
                     pt1.y + pt2.y);
}
```

Memory Diagram Exercise

- Use C Tutor to visualize the execution of the revised implementations of `addpoints`