# Structures

# Connecting Data

- Problem: Certain variables naturally fit together
- Examples:
  - test1_grade, test2_grade, test3_grade
  - x_coordinate, y_coordinate
  - Width, Height, Depth (of a cube)
  - year, month, dom
  - experiment_id, exp_temp, exp_pressure
  - First_Name, Last_Name, Middle_Initial, ID_Number, Age
  - Artist, Album, Track, Title, Duration, Date_of_Publication

# Connect with Arrays?

- test1_grade, test2_grade, test3_grade

```
float grades[3];
const char test1=0, test2=1, test3=2;

grades[test1]=90.0;
…
printf("Test 3 grade is %f\n",grades[test3]);
```

# Connect with Arrays?

- year, month, dom

```
int date[3];
const char year=0, month=1, dom=2;

date[year]=currentYear();


…
if (date[month]>12) {
    date[year]++;
    date[month]-=12;
}
```

# Connect with Arrays?

- Works when all associated variables are the same type
- How do we put different types into an array?
- e.g. First_Name, Last_Name, Middle_Initial, ID_Number, Age

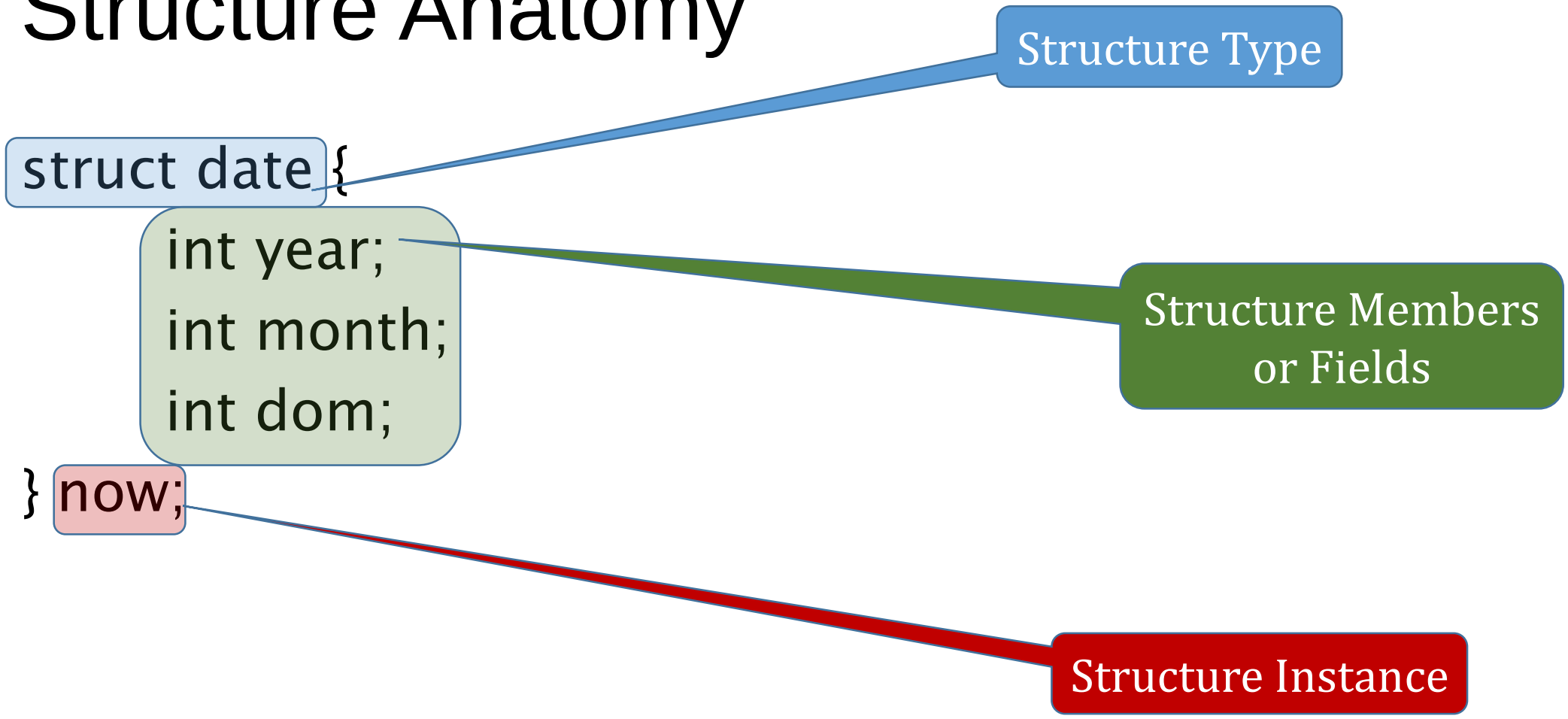
# C Structures

- Method to group like variables
- Allows each variable to have its own name
- Allows each variable to have its own type
- Gives a name (and type) to the entire group

# Example Structure

```
struct date {
        int year;
        int month;
        int dom;
} now;
```

# Structure Anatomy



# Structure Type

- Once a structure is declared, you can use the structure type as a data type

```
struct date nextDay(struct date today) {
        struct date tomorrow;
        tomorrow=today;
        tomorrow.dom++;
        …
        return tomorrow;
}
```

# Structure Members

- Look like variable declarations
- May have the same name as real variables
- May be of any type
  - int, float, char, pointers, arrays, even other structures!

# Structure Instance

- The instance name is a variable name
- Space is reserved in memory for a structure instance
  - At least enough to hold all the members of the structure
  - Sometimes, extra space is added… "Padding" to make everything line up.
- Instance name must be a unique variable name

# Using Structure Instances

- Access individual members using:
  <Instance_Name> . <Member_Name>

```
struct date today;
today.year=currentYear();
today.month=currentMonth();
toda.dom=currentDom();
printf("This year is %d\n",today.year);
```

# Structure Initialization

- You may provide a comma separated list of initial values as initialization values in a structure instance declaration.
- Members of the structure are initialized in the order in which they appear when the structure is defined

```
struct date {
        int year; int month; int dom;
} today={2015,10,28};
```

# Structure Copy

- You may assign one structure instance to another structure instance if they are instances of the same structure.
- This is the same as assigning each of the members.

```
struct date today={2015,10,28};
struct date tomorrow;
tomorrow=today; // Copy today's date to tomorrow
```

- You may not compare two structure instances.

# Example Comparison

```
int compDate(struct date a, struct date b) {
        if (a.year < b.year) return 1;
        if (a.year > b.year) return -1;
        if (a.month < b.month) return 1;
        if (a.month > b.month) return -1;
        if (a.dom < b.dom) return 1;
        if (a.dom > b.dom) return -1;
        return 0;
}
```

# Type Definition vs. Variable Declaration

- A typical structure definition does two things:
  - Defines a new type
  - Creates/Declares a new variable

```
struct date {
        int year; int month; int dom;
} ;
```

Defines type: struct date

```
struct date today;
```

Creates variable: today of type: struct date

# Un-named Structure Types

- Type name not required, but without a type name, impossible to create other instances of the same type

```
struct {
        int x; int y;
} origin={0,0};
```

# Pointers to Structures

- Get the location of a structure with &

```
struct date today={2015,10,28};
struct date *dptr=&today;
```

- Can use * indirection: (*dptr).dom++;
- Or, use C “pointer” shorthand: dptr->dom++;

# Big Structures

```
struct song {
        char artist[100];
        char album[100];
        int track;
        char title[100];
        float duration;
        struct date publication;
} dearPrudence={“Beatles”,”White Album”,2,”Dear Prudence”, 380.6,{1968,11,22}};
```

# Problem: Data Size

- If I pass a big structure as an argument, C copies lots of data
- Solution: Pass a pointer to the structure
- Also enables structure update!

# Example of Structure Pointer Usage

```
void setTitle(struct song *sptr,char * title) {
        if (strlen(title) > sizeof(sptr->title)) {
                printf("Title too large… will be truncated.\n");
                title[sizeof(sptr->title)-1]=x00;
        }
        strcpy(sptr->title,title);
}
```

# Resources

- Programming in C, Chapter 7
- Wikipedia Record https://en.wikipedia.org/wiki/Record_(computer_science)
- Structure Tutorial: http://www.tutorialspoint.com/cprogramming/c_structures.htm