iClicker Attendance

Please click on A if you are here:

A. I am here today.

Structures

Connecting Data

- Problem: Certain variables naturally fit together
- Examples:
  - test1_grade, test2_grade, test3_grade
  - Width, Height, Depth (of a box)
  - year, month, dom (day of month)
  - experiment_id, exp_temp, exp_pressure
  - Pin name, module number, instance, value, direction, net
  - 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 now[3];
const char year=0, month=1, dom=2;

now[year]=currentYear();


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

Connect with Arrays?

- May be acceptable if all associated variables are the same type
- How do we put different types into an array?

  e.g. Pin name, module number, instance, value, direction, net

  - Pin name – char * string
  - Module number – int
  - Instance number – int
  - Value – int
  - Direction – char
  - net - int

C Structures

- Method to group 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

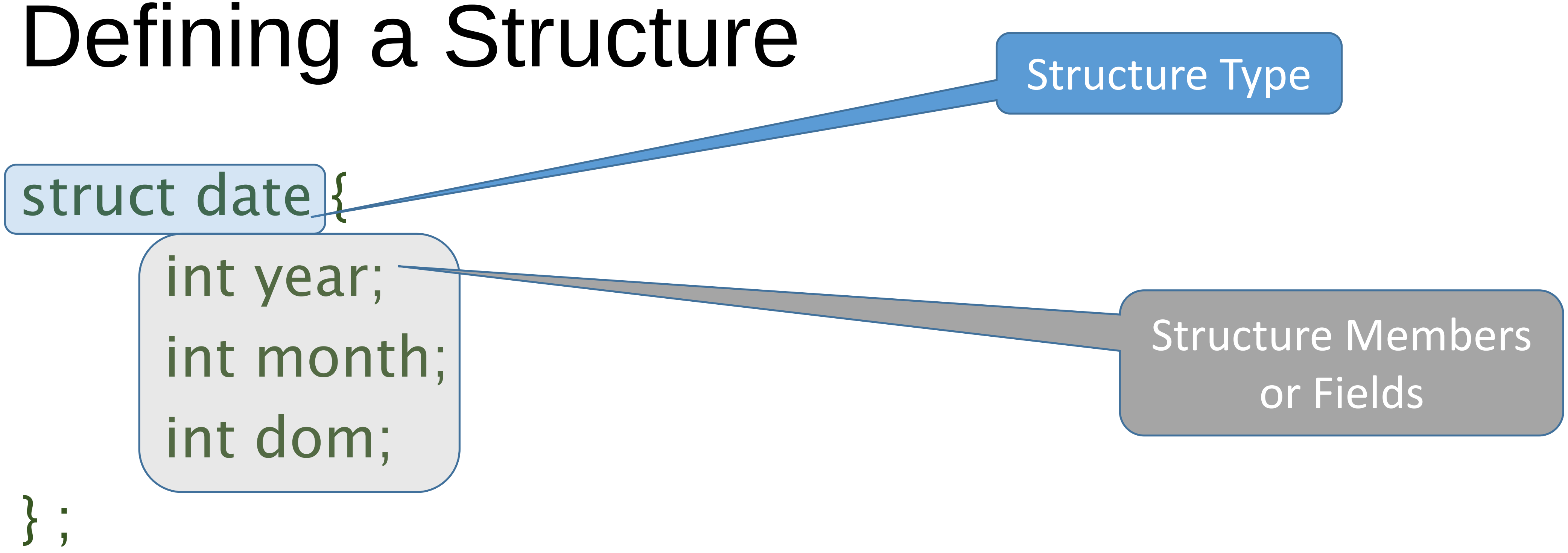
Example Structure

```
struct date {
        int year;  // Like 2017
        int month; // Like 10 for October
        int dom; // Like 23 for October 23
} ;
```

Structure Family of Data Types

- There is only one data type called “int”
  - A 32 bit two’s complement binary number
- There’s only one float, char, double, etc.
- A “struct” is not a single data type, but a FAMILY of data types
  - The family of all data types that are made up of multiple components
- To further qualify a structure, we must
  - Define the structure – tell the compiler what the components are
  - Tag our newly defined structure with a name; a “tag”
- The full data type is a *combination* of the keyword, struct; and the tag

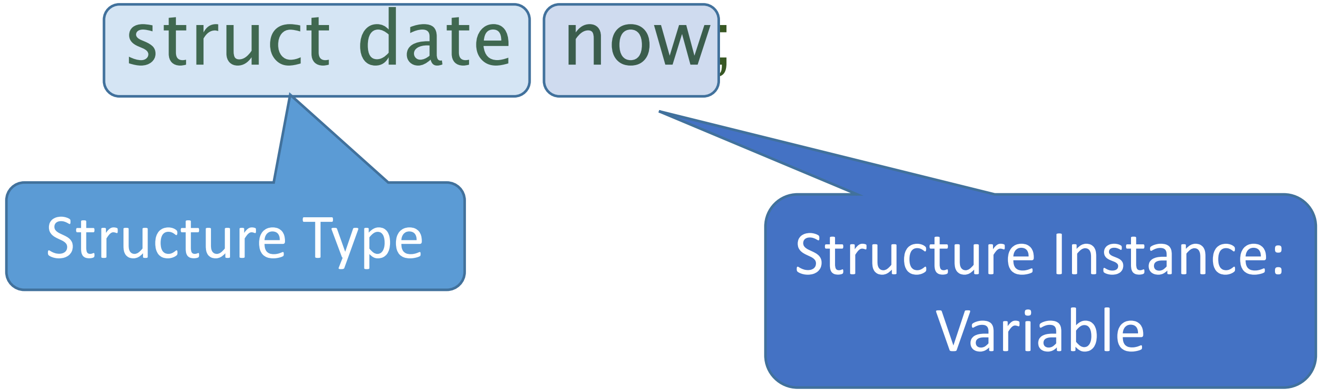
Defining a Structure



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

- Creates a new derived data type called “struct date”
- Every variable with this type has sub-fields: year, month, dom

Declaring a Structure



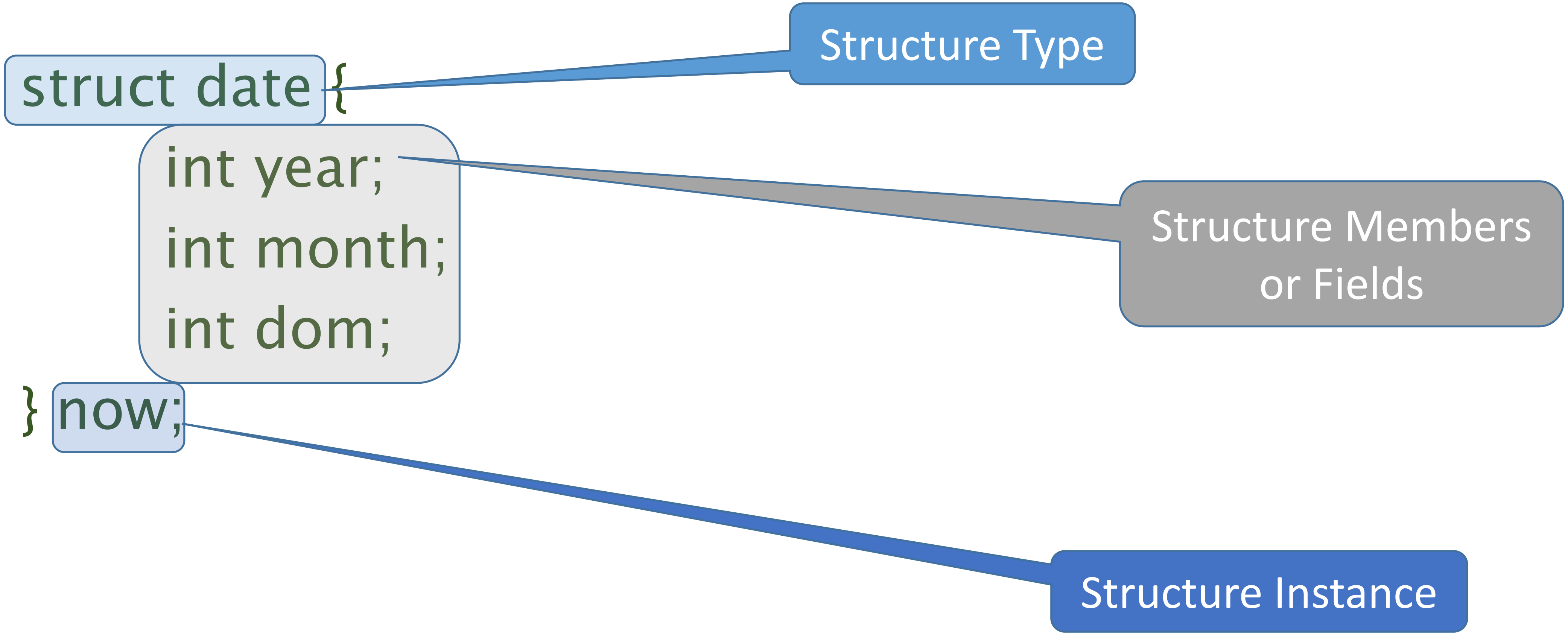
```
struct date now;
```

Structure Type

Structure Instance: Variable

- Structure type must be defined previously
- Creates a variable called “now” which has sub-fields
  - Year, month, and day of month (dom)
- Reserves memory for the sub-fields
- Terminology: now is an ***instance*** of the date structure

Define and Declare shorthand



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

Structure Type

Structure Members or Fields

Structure Instance

Structures may only be defined once

Mistake

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

Error, structure “date” is already defined!

Correct

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

Use the "date" structure defined previously

Structure Type

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

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

Structure Members

- Look like variable declarations
- May be of any type
  - int, float, char, pointers, arrays, even other structures!
- May *not* be initialized in the structure definition
- May have the same name as real variables

```
int dom=21;
struct date { int year; int dom; int mon; } today ={2017,22,10};
```

Refer to as "dom"

Refer to as "today.dom"

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
    - even though fields do not have to be unique

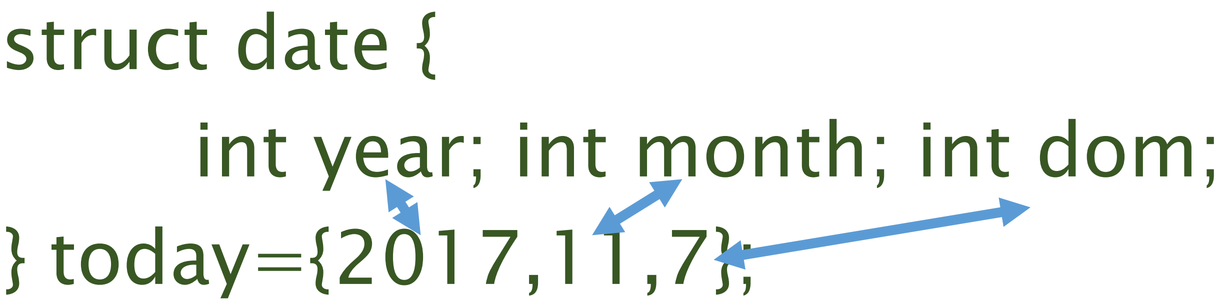
Using Fields in Structure Instances

Access fields using: *Instance_Name.Field_Name*

```
struct date { int year; int month; int dom;} today;
today.year=currentYear();
today.month=currentMonth();
today.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={2017,11,7};
```

Anonymous 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};
```

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={2017,10,23};
struct date tomorrow;
tomorrow=today; // Copy today’s date to tomorrow
```

- You may ***not*** compare two structure instances.
  - C doesn’t know which fields should be compared or which are more important

Structures as Arguments

- If you specify a structure as a parameter for a function, when that function is invoked, the argument must be an instance of that structure.
- C will create a NEW instance of the structure for the parameter, and copy the argument into the parameter (field by field!)
  - For big structures, this could be time consuming!

```
struct date today={2019,11,18};
struct date newyear={2020,1,1};
if (compDate(today,newyear)<0) printf("New year is coming!");
```

Example Comparison

```
int compDate(struct date a, struct date b) {
        if (a.year < b.year) return –1; // a is smaller
        if (a.year > b.year) return 1; // b is smaller
        // Year matches… compare month
        if (a.month < b.month) return –1;
        if (a.month > b.month) return 1;
        // Month matches.. compare day
        if (a.dom < b.dom) return –1;
        if (a.dom > b.dom) return 1;
        return 0; // dates are the same
}
```

Example Comparison (version 2)

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

Example: days in …

```
int daysInYear(int y) {
   return 365 + isLeapYr(y);
}
```

```
int isLeapYr(int y) {
   if (!y%400) return 1;
   if (!y%100) return 0;
   if (!y%4) return 1;
   return 0;
}
```

```
int daysInMonth(struct date d) {
   static const int dim[]
={31,28,31,30,31,30,31,31,30,31,31,30}
// jn,fb,mr,ap,my,jn,jl,au,sp,oc,no,de
   if(d.mon!=2) return dim[d.mon-1];
   return 28+ isLeapYr(d.year);
}
```

Example: daysInMillenium

```
int daysinMillenium(struct date d) {
  assert(d.year>=1900);
  int dys=0;
  for(int yrs=d.year;yrs>1900;yrs--) dys+=daysInYear(yrs);
  while(d.month > 0) {
    dys+=daysInMonth(d);
    d.month--;
  }
  return dys+d.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}};
```

Big Structures as Arguments

```
boolean isGoldie(struct song s) {
   if (s.published.year < 2006) { return true; }
   return false;
}
```

- Copies entire “song” structure to activation record… 308 bytes!

Structures as Return Types

- A C function may return a structure.
- If you assign the result of a function to a structure, the fields will be copied into the instance of the structure you provide as the target to the assignment.
    - Allows return of multiple values!
    - Allows function to build a structure in a local variable and return it!
    - Compare to arrays – no array copy is performed on return of address

Example of a structure return type

Assumes "struct date" already defined

```
struct date newDate(int year,int month,int dom) {
        struct date n;
        n.year=year;
        n.month=month;
        n.dom=dom;
        reutrn n;
}
…
struct date xmas=newDate(2019,12,25);
```

Local instance "n"

"n" stays around just long enough for C to copy n.year to xmas.year, n.month to xmas.month, and n.dom to xmas.dom

C Pitfall… Cannot return local array!

```
int * sequence(int s,int n) {
        int nums[n]; // array of integers with n elements
        for(int i=0;i<n);i++) nums[i]=s+i;
        return nums;
}
…
int seq[5]=sequence(10,5);
```

Local instance "nums"

Returns &nums[0]
Then gives back the memory for nums!

Allocates space for 5 ints, but
then replace the address of that memory
with the address of nums
which is NO LONGER VALID!

Alternatives for Returning Arrays

- Make the array static
  - but then someone else might invoke the function and change it
- Make the return array an argument
  - Require your use to give you the space to write to
  - Makes your user responsible for providing (and cleaning up) the space
- Dynamically allocate space for the array (coming soon)
  - Makes your user responsible for cleaning up the space

Pointers to Structures

- It is so useful to pass structures by reference, we almost always do so
- C programmers got tired of coding: (*todayRef).year
- Pointer to structure shorthand…

(*todayRef).year

is the same as….

todayRef->year

C Pitfall – Returning *local structure

```
struct date * newDate(int year,int month,int dom) {
        struct date n;
        n.year=year;
        n.month=month;
        n.dom=dom;
        return &n;
}
…
struct date *today=newDate(2012,11,18);
```

Local instance "n"

Returns &n
Then gives back the memory for n!

today ->n
which is NO LONGER VALID!

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