Today’s Goals

- Structures
  - Types and variables
  - **`typedef`**
  - **`struct`**s and pointers
- Unions
- Enumerations

Section 1

Structures

- To group multiple (heterogeneous) variables
- Similar to Java classes, but not as powerful
  - A structure has only *data* members
  - All members are *public*

Structure Operations

- Structure type declaration
- Structure variable declaration
- Member assignment/reference
- Structure initialization
- Structure assignment

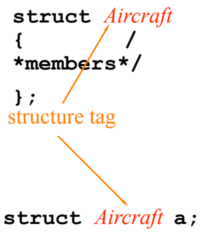
Structure Type Declaration

- Pattern
  - **`struct`** *StructType* **`{ /* members */`** ⚠ **`};`**
  - *Typ*ically global
- Members
  - Analogous to data declaration

```
struct Aircraft{
  char id[10];
  int x;
  int y;
  int z;
  int prevZ;
  int heading;
  int verticalSpeed;
  int speed;
};

int main() {
  /* skipped */
}
```

Struct Instance

- Aircraft identifies a structure type, also known as a structure tag.
- **`a`** is an instance of the structure type Aircraft
- Keyword **`struct`** may not be dropped



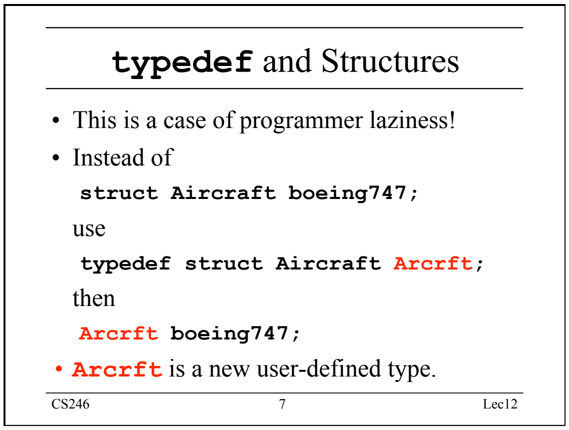
```
struct Aircraft
{         /
*members*/
};
```

structure tag

```
struct Aircraft a;
```

`typedef`

- A way to define a synonym for existing (complicated) types.
  - **`typedef int Bool;`**
  - **`typedef int*** Intptr3;`**
- **`typedef`**ed type names by convention have the first letter in uppercase.
- Besides programmer laziness, **`typedef`** does contributes to portability (**`size_t`**)
  - **`typedef long Myint;`** – others
  - **`typedef int Myint;`** – machines with 32-bit **`int`**



`typedef` and Structures

- This is a case of programmer laziness!
- Instead of

```
struct Aircraft boeing747;
```

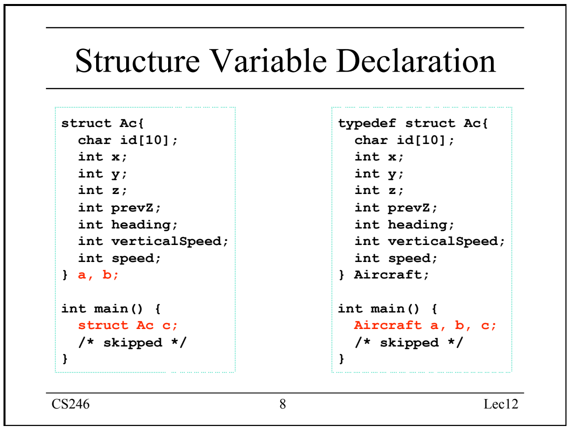
use

```
typedef struct Aircraft Arcrft;
```

then

```
Arcrft boeing747;
```

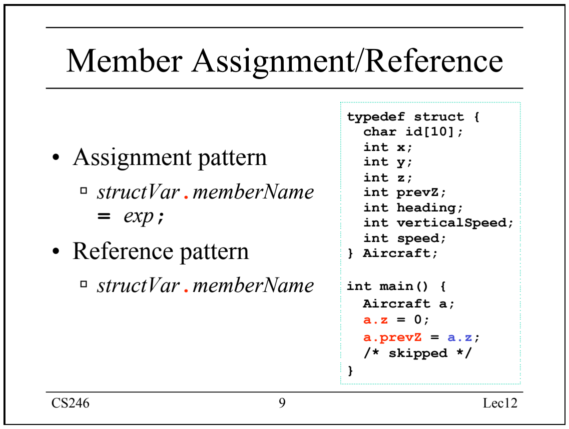
- `Arcrft` is a new user-defined type.



Structure Variable Declaration

```
struct Ac{
  char id[10];
  int x;
  int y;
  int z;
  int prevZ;
  int heading;
  int verticalSpeed;
  int speed;
} a, b;

int main() {
  struct Ac c;
  /* skipped */
}
```

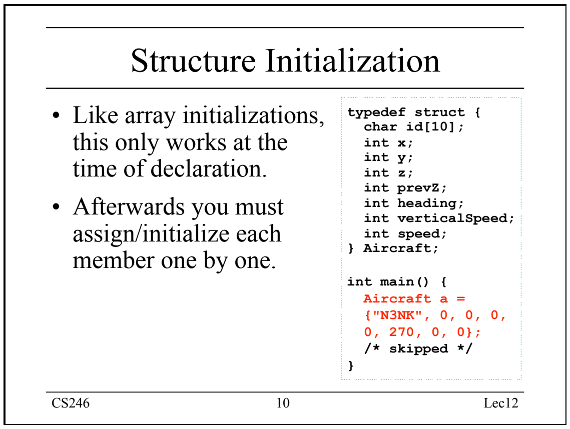
```
typedef struct Ac{
  char id[10];
  int x;
  int y;
  int z;
  int prevZ;
  int heading;
  int verticalSpeed;
  int speed;
} Aircraft;

int main() {
  Aircraft a, b, c;
  /* skipped */
}
```



Member Assignment/Reference

- Assignment pattern
  - *structVar*`.`*memberName* `=` *exp*`;`
- Reference pattern
  - *structVar*`.`*memberName*

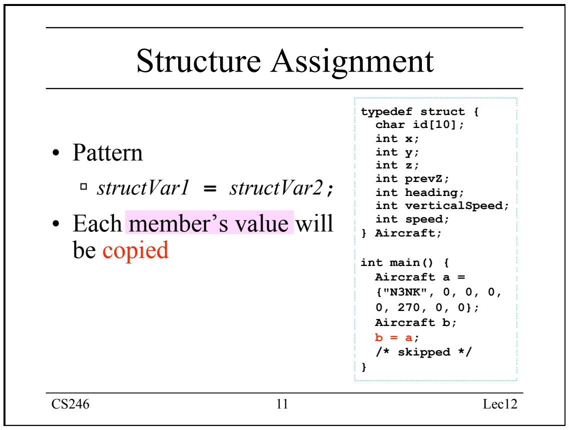
```
typedef struct {
  char id[10];
  int x;
  int y;
  int z;
  int prevZ;
  int heading;
  int verticalSpeed;
  int speed;
} Aircraft;

int main() {
  Aircraft a;
  a.z = 0;
  a.prevZ = a.z;
  /* skipped */
}
```



Structure Initialization

- Like array initializations, this only works at the time of declaration.
- Afterwards you must assign/initialize each member one by one.

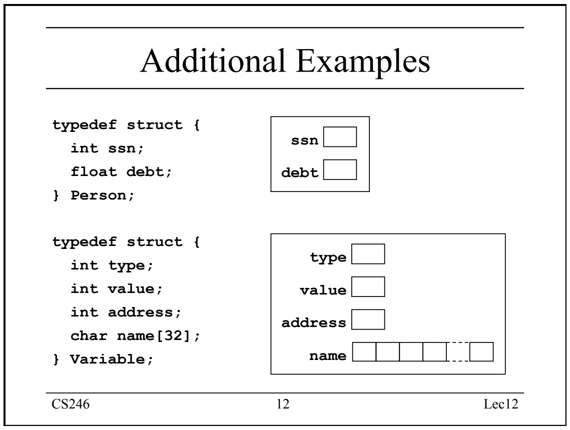
```
typedef struct {
  char id[10];
  int x;
  int y;
  int z;
  int prevZ;
  int heading;
  int verticalSpeed;
  int speed;
} Aircraft;

int main() {
  Aircraft a =
  {"N3NK", 0, 0, 0,
  0, 270, 0, 0};
  /* skipped */
}
```



Structure Assignment

- Pattern
  - *structVar1* `=` *structVar2*`;`
- Each member's value will be copied

```
typedef struct {
  char id[10];
  int x;
  int y;
  int z;
  int prevZ;
  int heading;
  int verticalSpeed;
  int speed;
} Aircraft;

int main() {
  Aircraft a =
  {"N3NK", 0, 0, 0,
  0, 270, 0, 0};
  Aircraft b;
  b = a;
  /* skipped */
}
```



Additional Examples

```
typedef struct {
  int ssn;
  float debt;
} Person;
```

ssn
debt

```
typedef struct {
  int type;
  int value;
  int address;
  char name[32];
} Variable;
```

type
value
address
name

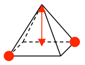
Section 2

Complex Structures

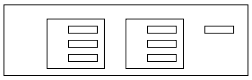
- Various structure members
  - Basic types: **int**, **double**, **char**, etc.
  - Arrays
  - Pointers
  - Structures
- Arbitrary combination possible

```
typedef struct {
  int x;
  int y;
  int z;
} Position;

typedef struct {
  char id[10];
  Position pos;

  int prevZ;
  int heading;
  int verticalSpeed;
  int speed;
} Aircraft;
```

Another Example

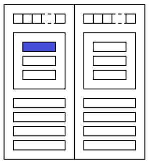


```
typedef struct {
  Position northeast_corner;
  Position southwest_corner;
  int height;
} Mountain;
```



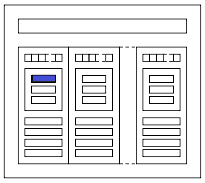
Array of Structures

```
typedef struct {
  char id[10];
  Position pos;
  int prevZ;
  int heading;
  int verticalSpeed;
  int speed;
} Aircraft;

int main() {
  Aircraft aircrafts[2] =
     { { init list for elem 0 },
       { init list for elem 1 } };

  aircrafts[0].pos.x = 0;
}
```



Structure with Array of Structures

```
typedef struct {
  char id[10];
  Position pos;
  int prevZ;
  int heading;
  int verticalSpeed;
  int speed;
} Aircraft;

typedef struct {
  int numOfAircrafts;
  Aircraft aircrafts[100];
} Radar;

int main() {
  Radar r;
  r.aircrafts[0].pos.x = 0;
}
```



Section 3

Structure Arguments

```
void updateStatus(Aircraft b) {
  b.heading += 90;
}

int main() {
  Aircraft a = initialization;
  updateStatus(a);
  return 0;
}
```

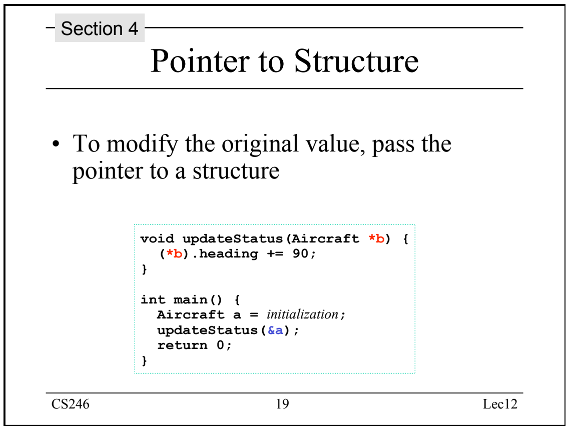
- The argument variable **b** is a copy of the original variable **a**.
- Analogous to basic variables, different from arrays
- Cannot change the original variable **a**

Section 2

Structure Return

```
Aircraft updateStatus(Aircraft b) {
  b.heading += 90;
  return b;
}

int main() {
  Aircraft a = initialization;
  a = updateStatus(a);
}
```

- The local variable **b** is modified and returned.
- The returned **b** can be assigned (copied) to the original **a**.

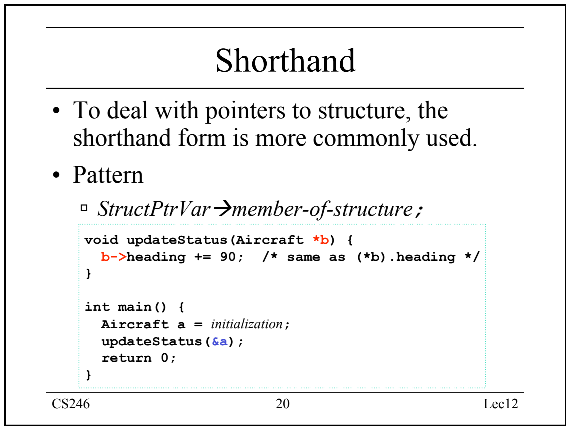


Section 4

Pointer to Structure

- To modify the original value, pass the pointer to a structure

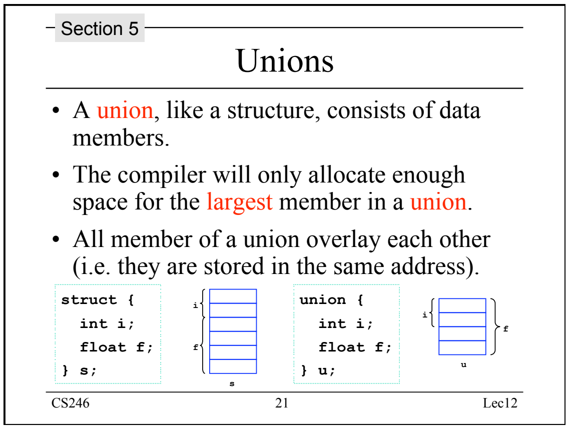
```
void updateStatus(Aircraft *b) {
   (*b).heading += 90;
}

int main() {
   Aircraft a = initialization;
   updateStatus(&a);
   return 0;
}
```



Shorthand

- To deal with pointers to structure, the shorthand form is more commonly used.
- Pattern
  - *StructPtrVar→member-of-structure***;**

```
void updateStatus(Aircraft *b) {
   b->heading += 90;  /* same as (*b).heading */
}

int main() {
   Aircraft a = initialization;
   updateStatus(&a);
   return 0;
}
```



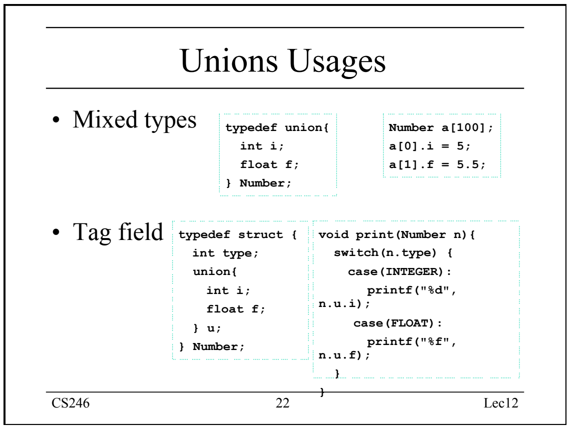
Section 5

Unions

- A union, like a structure, consists of data members.
- The compiler will only allocate enough space for the largest member in a union.
- All member of a union overlay each other (i.e. they are stored in the same address).

```
struct {
  int i;
  float f;
} s;
```

```
union {
  int i;
  float f;
} u;
```



Unions Usages

- Mixed types

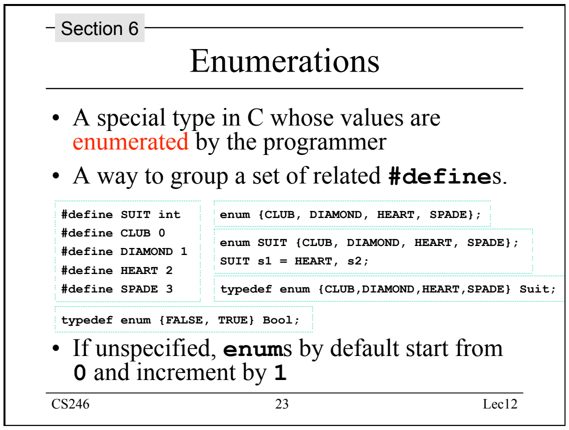
```
typedef union{
  int i;
  float f;
} Number;
```

```
Number a[100];
a[0].i = 5;
a[1].f = 5.5;
```

- Tag field

```
typedef struct {
  int type;
  union{
    int i;
    float f;
  } u;
} Number;
```

```
void print(Number n){
  switch(n.type) {
    case(INTEGER):
      printf("%d",
n.u.i);
     case(FLOAT):
      printf("%f",
n.u.f);
  }
}
```



Section 6

Enumerations

- A special type in C whose values are enumerated by the programmer
- A way to group a set of related **`#define`**s.

```
#define SUIT int
#define CLUB 0
#define DIAMOND 1
#define HEART 2
#define SPADE 3
```

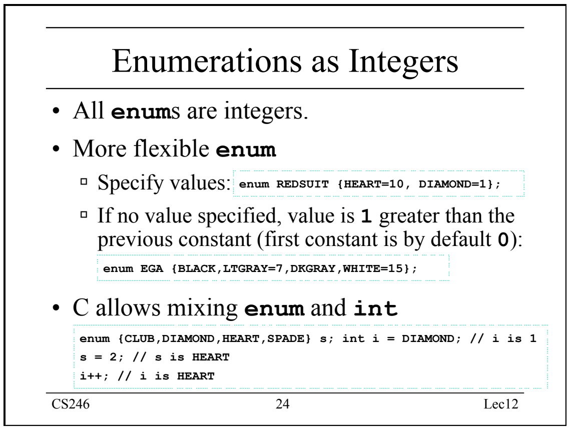
```
enum {CLUB, DIAMOND, HEART, SPADE};
```

```
enum SUIT {CLUB, DIAMOND, HEART, SPADE};
SUIT s1 = HEART, s2;
```

```
typedef enum {CLUB,DIAMOND,HEART,SPADE} Suit;
```

```
typedef enum {FALSE, TRUE} Bool;
```

- If unspecified, **`enum`**s by default start from **`0`** and increment by **`1`**



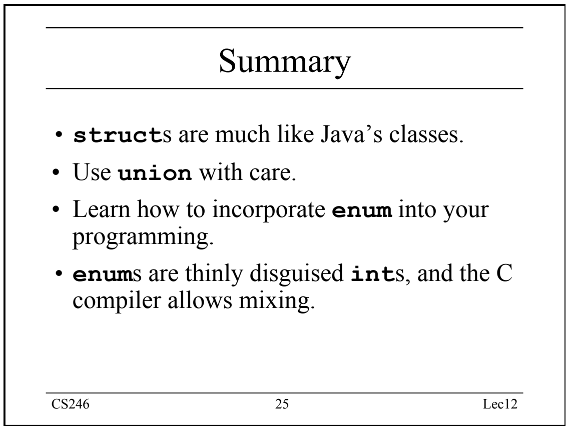
Enumerations as Integers

- All **`enum`**s are integers.
- More flexible **`enum`**
  - Specify values: `enum REDSUIT {HEART=10, DIAMOND=1};`
  - If no value specified, value is **`1`** greater than the previous constant (first constant is by default **`0`**):

```
enum EGA {BLACK,LTGRAY=7,DKGRAY,WHITE=15};
```

- C allows mixing **`enum`** and **`int`**

```
enum {CLUB,DIAMOND,HEART,SPADE} s; int i = DIAMOND; // i is 1
s = 2; // s is HEART
i++; // i is HEART
```



Summary

- **`struct`**s are much like Java’s classes.
- Use **`union`** with care.
- Learn how to incorporate **`enum`** into your programming.
- **`enum`**s are thinly disguised **`int`**s, and the C compiler allows mixing.