Today’s Goals

- Structures
  - Types and variables
  - typedef
  - structs and pointers
- Unions
- Enumerations

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 */
    
    /*Typically global*/
  }
  ` Members
  - Analogous to data declaration

typedef

- A way to define a synonym for existing (complicated) types.
  - typedef int Bool;
  - typedef int*** Intptr3;
- typedefed 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

Struct Instance

- Aircraft identifies a structure type, also known as a structure tag.
- `struct Aircraft {
    /* members */
    
    /*Typically global*/
  }
  ` Structure tag
- a is an instance of the structure type Aircraft
- Keyword struct may not be dropped
- `struct Aircraft a;`
**typedef and Structures**

- This is a case of programmer laziness!
- Instead of
  ```c
  struct Aircraft boeing747;
  ```
  use
  ```c
  typedef struct Aircraft Arcrft;
  ```
  then
  ```c
  Arcrft boeing747;
  ```
- `Arcrft` is a new user-defined type.

**Structure Variable Declaration**

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

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

**Member Assignment/Reference**

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

**Structure Initialization**

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

**Structure Assignment**

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

**Additional Examples**

```c
typedef struct {
    char id[10];
    int ssn;
    float debt;
} Person;
```

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

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

Another Example

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

Array of Structures

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

typedef struct {
    Aircraft aircrafts[2] = {
        { init for elem 0 },
        { init for elem 1 }
    };
    Aircraft 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;

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

Structure Arguments

- 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`

Structure Return

- 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.

```c
void updateStatus(Aircraft *b) {
    (*b).heading += 90;
}
int main() {
    Aircraft a = initialisation;
    updateStatus(&a);
    return 0;
}
```

- **Shorthand**
  - To deal with pointers to structure, the shorthand form is more commonly used.
  - Pattern

```
StructPtrVar member-of-structure;
```

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

```c
struct {
    int i;
    float f;
} s;
union {
    int i;
    float f;
} u;
```

**Section 6**

**Enumerations**

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

```c
#define SUIT int
#define CLUB 0
#define DIAMOND 1
#define HEART 2
#define SPADE 3
enum {CLUB, DIAMOND, HEART, SPADE};
typedef enum {CLUB, DIAMOND, HEART, SPADE} Suit;
enum {FALSE, TRUE} Bool;
```

- If unspecified, enums by default start from 0 and increment by 1.

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

- All enums are integers.
- More flexible enum
  - Specify values:
  ```c
  enum REDSUIT (HEART=10, DIAMOND=11);
  ```
  - If no value specified, value is 1 greater than the previous constant (first constant is by default 0):
  ```c
  enum ESA (BLACK, LTGRAY=7, DKGRAY, WHITE=15);
  ```
- C allows mixing enum and int
  ```c
  enum (CLUB, DIAMOND, HEART, SPADE) s; int i = DIAMOND; // i is 1
  int s = 2; // s is HEART
  i++; // i is HEART
  ```
Summary

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