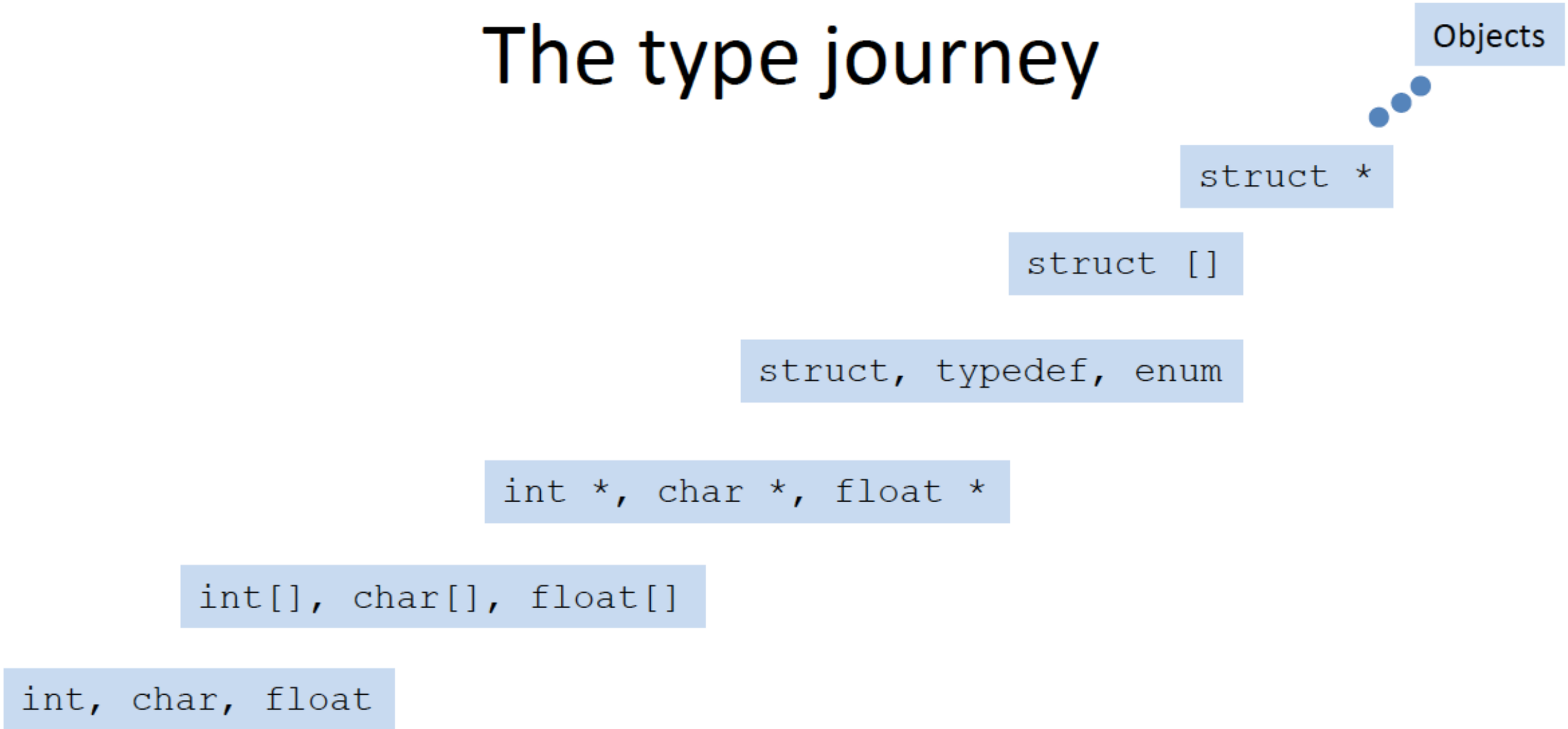
ECE 220: Computer Systems & Programming

Lecture 15: Data Structures

Thomas Moon

March 7, 2024

The type journey



Data Types

- 3 fundamental data types
  - `int`: integer number
  - `float/double`: real number
  - `char`: character

- Array: A list of values (*homogeneous*) arranged sequentially in memory

What if we want to group different types of items together?

Structures

- A `struct` allows user to define a new data that can be used to group items of *different types*.

Example: We want to represent an aircraft

```
char flightName[20];
int altitude;
int longitude;
int latitude;
int heading;
double airSpeed;
```

Use a `struct` to *group* them for each plane.

Defining a Struct

- First, define a new data type for the compiler and tell it what our struct looks like.

```
struct flightType
{
    char flightName[20];
    int altitude;
    int longitude;
    int latitude;
    int heading;
    double airSpeed;
};
```

This tells the compiler

1. How big the struct is
2. How the different data items ("members") are laid out in memory.

But it does not allocate any memory yet.

Declaring and Using a Struct

- To allocate memory for a struct, we declare a variable using the new data type.

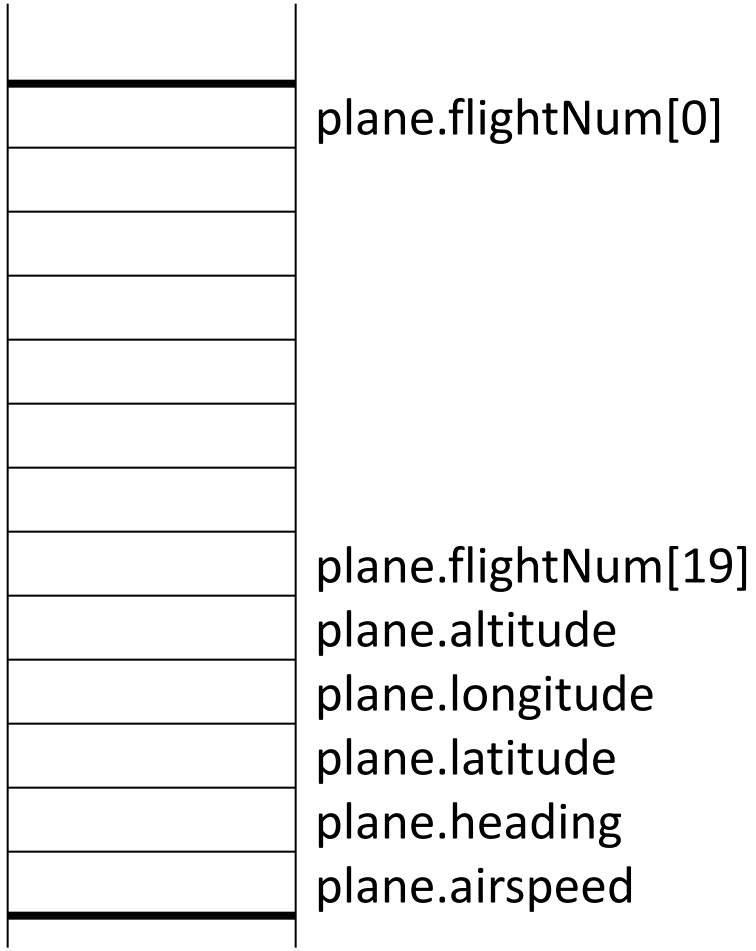
```
struct flightType plane;
```

- Member access operator (.):
  Once memory is allocated, we can access individual members of this variable.

```
plane.altitude = 1000;
plane.airspeed = 800.0;
```

- You can also initialize a struct.

```
struct flightType plane = {"DL2034", 0, 0};
```



Using `typedef`

- C provides a way to define a data type by giving a new name to a predefined type.
- syntax: `typedef <type> <name>;`
- Example:

```
typedef int Color;
Color pixel[100];
```

```
typedef struct flightType{
        …
}Flight;

Flight plane1;
```

```
struct flightType{
        …
};
typedef struct flightType Flight;

Flight plane1;
```

sizeof

- Use sizeof operator

```
sizeof(type)
```

to evaluate the number of bytes occupied by the type.

- It is useful to calculate the size of structure.
  (It's not a good idea to calculate a size by yourself,
  because compilers insert some paddings to ensure alignment in memory)

```
struct struct1{
    char c;
    int x;
    double y;
}Struct1;
```

```
sizeof(Struct1) = 16 > 1+4+8
```

```c
#include <stdio.h>
#include <math.h>

typedef struct PointType{
    double x,y;
}point;

int main(){
    // Initialize p1, p2 by the origin
    ____________________________________;
    double dis = ___________________;

    printf("Initial distance: %lf\n", dis);

    printf("Enter x and y for p1:");
    scanf("%lf %lf", ______, _____);

    printf("Enter x and y for p2:");
    scanf("%lf %lf", ______, _____);

    dis = sqrt( ____________________________ );
    printf("New distance: %lf\n", dis);
}
```

Other User Defined Data Types

- Enumeration (`enum`): User defined data type assigning names to integer constants (starting from 0)
  - syntax: `enum [tag] {enumerator list};`

```
enum weekday {SUN,MON,TUE,WED,THU,FRI,SAT};
```
<- definition

```
enum weekday today;
today = TUE;
```
<- declaration

  - Value is implicitly set, but we can set it explicitly.

```
enum weekday {SUN=7,MON=1,TUE,WED,THU,FRI,SAT};
```

Union

- Similar to struct, but the members of the union shares the same memory location.

```
typedef union union1{
    char c;
    int x;
    double y;
}Union1;

Union1 u1;
u1.c = 'a';
u1.x = 10;
u1.y = 1.5;
```

```
typedef struct struct1{
    char c;
    int x;
    double y;
}Struct1;

Struct1 s1;
s1.c = 'a';
s1.x = 10;
s1.y = 1.5;
```

size of union vs struct?

```
sizeof(union1) = 8, sizeof(struct1) = 16
```

contents of union vs struct?

```
u1: , 0, 1.500000
s1: a, 10, 1.500000
```

Array of Struct

- Can declare an array of structs:

```
Flight planes[100];
```

- Access each element of array

```
planes[0]
planes[1]
…
```

- To access member of each element:

```
planes[0].altitude = 10000;

planes[0].flightName[0] = 'D';
planes[0].flightName[1] = 'L’;
planes[0].flightName[2] = ‘\0';

strncpy(planes[0].flightName, "DL", sizeof(planes[0].flightName));
strcpy(planes[0].flightName, "DL");
```

```c
#define BUF_SIZE 100
typedef struct StudentStruct{
    int UIN;
    char netid[BUF_SIZE];
    float GPA;
}student;

int main(){
    student s[100];

    //add Bob
    ____________________;
    ____________________;
    ____________________;

    //add Alice
    ____________________;
    ____________________;
    ____________________;

    //Print out
    ____________________________;
    ____________________________;
```

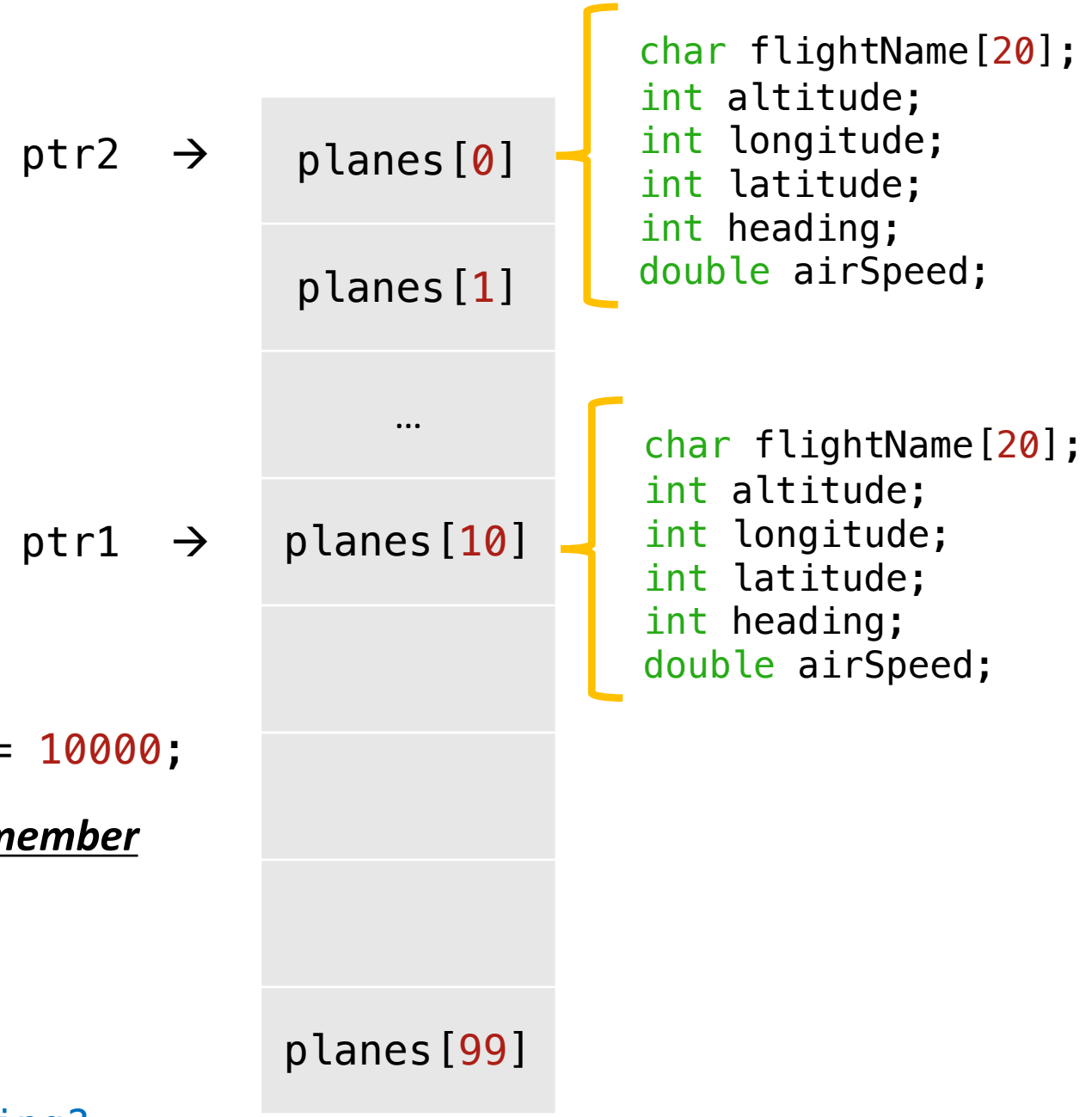
```
UIN netid GPA
1 Bob 3.0
2 Alice 3.5
```

Pointer to Struct

```
Flight planes[100];
Flight *ptr1;

ptr1 = &planes[10];

Flight *ptr2;
ptr2 = planes;
```



→ *How to access a struct member via pointer?*

method1:

```
(*ptr1).altitude = 10000;
```

method2:

```
ptr1->altitude = 10000;
```

C provides special syntax -> for *accessing a struct member* through a pointer:

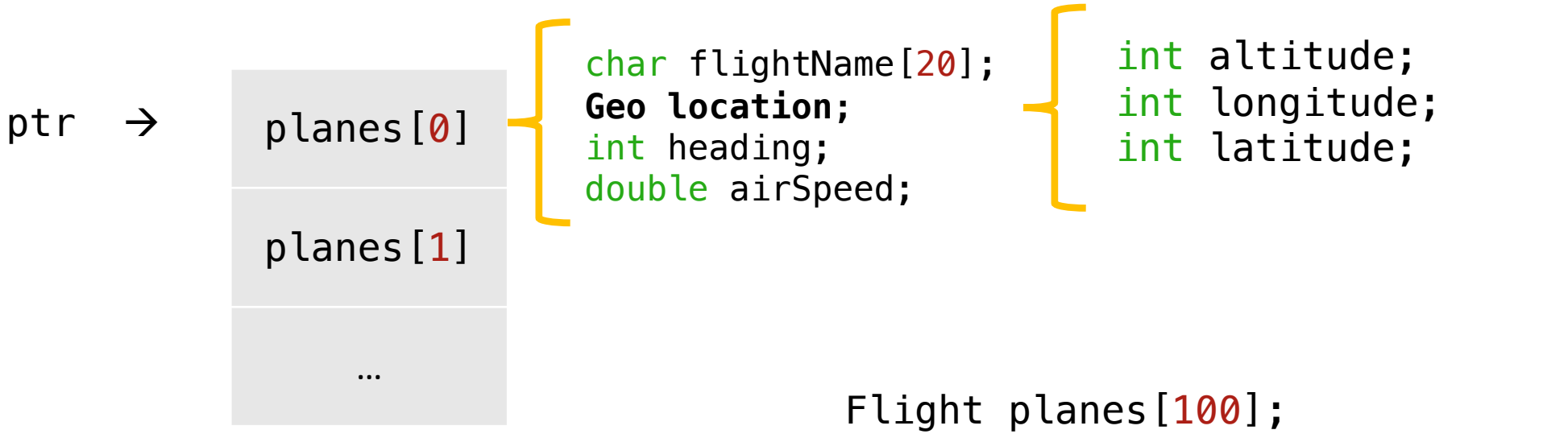
```
ptr2->altitude = 15000;
strcpy(ptr2->flightName, "AZ");

ptr2++;   // where is ptr2 pointing?
```

Struct in Struct

```
typedef struct GeoCoordinate{
    int altitude;
    int longitude;
    int latitude;
}Geo;
```

```
typedef struct flightType{
    char flightName[20];
    Geo location;
    int heading;
    double airSpeed;
}Flight;
```




```
Flight planes[100];
Flight *ptr;
ptr = planes;

ptr->location.altitude = 10000;
```

```c
#define BUF_SIZE 100
typedef struct StudentStruct{
    int UIN;
    char netid[BUF_SIZE];
    float GPA;
}student;

int main(){
    student s[100];

    student *ptr;
```

Use a student pointer to make the change

```
UIN netid GPA
1 Bob 3.0
2 Alice 3.5
```

```
UIN netid GPA
100 Bruno 3.8
2 Alice 3.5
```

Pass Structs as Arguments

A.

```
void print_flightName(Flight plane)
{
    printf("flight name: %s\n", plane.flightName);
}
```

vs

which one is more efficient?

B.

```
void print_flightName(Flight *plane)
{
    printf("flight name: %s\n", plane->flightName);
}
```

A. Passing by value will push the entire struct members onto the run-time stack.

B. Pass only one pointer.

Example: Student Record Management

```
UIN netid GPA
100 Bruno 3.8
2 Alice 3.5
```

```
int main(){
    student s[100];

    printStudents(_________);



void printStudents(________, int num){
    printf("UIN netid GPA\n");
    for(int i=0;i<num;i++)
        printf("%d %s %f\n",_____,______,______);
}
```

Example: Student Record Management

```
UIN netid GPA
100 Bruno 3.8
2 Alice 3.5
```

```
UIN netid GPA
2 Alice 3.5
100 Bruno 3.8
```

```
int main(){
    student s[100];

    swapStudent(____,______)



 void swapStudent(__________, ____________){
     ____________________;
     ____________________;
     ____________________;
     ____________________;
 }
```

Example: Student Record Management

```
UIN netid GPA
100 Bruno 3.8
2 Alice 3.5
```

```
UIN netid GPA
2 Alice 3.5
100 Bruno 3.8
```

```
int main(){
    student s[100];

    swapStudent(____,_____)
```

Modify bubbleSort to sort the students by GPA

```
void bubbleSort(int array[], int n){
    int is_swap, i;
    do{
        is_swap = 0;
        for(i=0; i < n-1; i++){
            if(array[i] > array[i+1]){
                swap(&array[i], &array[i+1]);
                is_swap = 1;
            }
        }
    }while(is_swap != 0);
}
```