CSc 352

Structs

Benjamin Dicken

What is a Struct?

- A **struct** is a feature of the C languages that allows us to cluster together multiple values into a single **object**
  - In C, **object** does not mean an instance of a class. Instead it just mean a chunk of memory
- As you should already know, in programming, it often makes sense to group multiple values together

```c
#include <stdio.h>

struct testing {
  int number;
  float other_number;
  char character;
};

int main() {
  struct testing x;
  x.number = 25;
  x.other_number = 1.1;
  x.character = 'c';
  printf("%d %f %c\n", x.number, x.other_number, x.character);
  return 0;
}
```

```c
#include <stdio.h>

struct testing {
  int number;
  float other_number;
  char character;
};

int main() {
  struct testing x;
  x.number = 25;
  x.other_number = 1.1;
  x.character = 'c';
  printf("%d %f %c\n", x.number, x.other_number, x.character);
  return 0;
}
```

Defines a new struct type, containing three values

Create an instance of the struct, on the stack

Use dot notation to set / get values

Try Running!

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

struct testing {
  int number;
  float other_number;
  char character;
};

int main() {
  struct testing * x = malloc(sizeof(struct testing));
  x->number = 25;
  x->other_number = 1.1;
  x->character = 'c';
  printf("Size: %lu\n", sizeof(struct testing));
  printf("%d %f %c\n", x->number, x->other_number, x->character);
  printf("%d %f %c\n", (*x).number, (*x).other_number, (*x).character);
  return 0;
}
```

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

struct testing {
  int number;
  float other_number;
  char character;
};

int main() {
  struct testing * x = malloc(sizeof(struct testing));
  x->number = 25;
  x->other_number = 1.1;
  x->character = 'c';
  printf("Size: %lu\n", sizeof(struct testing));
  printf("%d %f %c\n", x->number, x->other_number, x->character);
  printf("%d %f %c\n", (*x).number, (*x).other_number, (*x).character);
  return 0;
}
```

Can also create a pointer to dynamically allocated memory of appropriate size

How big is this?

The arrow is equivalent to a dereference and a dot

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

typedef struct testing {
  int number;
  float other_number;
  char character;
} testing;

int main() {
  testing * x = malloc(sizeof(testing));
  x->number = 25;
  x->other_number = 1.1;
  x->character = 'c';
  printf("Size: %lu\n", sizeof(testing));
  printf("%d %f %c\n", x->number, x->other_number, x->character);
  printf("%d %f %c\n", (*x).number, (*x).other_number, (*x).character);
  return 0;
}
```

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

typedef struct testing {
  int number;
  float other_number;
  char character;
} testing;

int main() {
  testing * x = malloc(sizeof(testing));
  x->number = 25;
  x->other_number = 1.1;
  x->character = 'c';
  printf("Size: %lu\n", sizeof(testing));
  printf("%d %f %c\n", x->number, x->other_number, x->character);
  printf("%d %f %c\n", (*x).number, (*x).other_number, (*x).character);
  return 0;
}
```

Use a typedef

Notice, we can now say just “testing” instead of “struct testing” at these places.

Structs

A struct is kind of like an instance of a class from Java/Python (an object), but without methods and private elements

Why Structs?

Come up with a few examples of where a struct would be practically useful in C programming.

```
typedef struct Coordinate2D {
  float x;
  float y;
} Coordinate2D;
```

```
typedef struct Coordinate3D {
  float x;
  float y;
  float z;
} Coordinate3D;
```

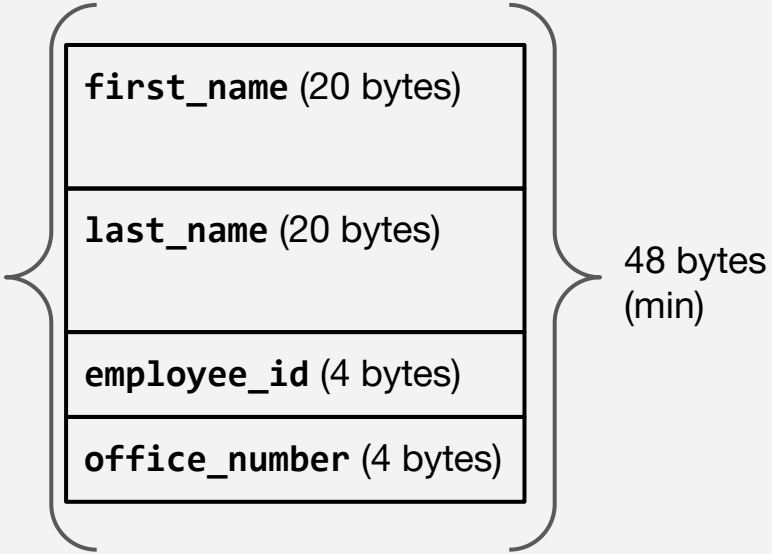
```c
typedef struct Employee {
  char first_name[20];
  char last_name[20];
  int employee_id;
  int office_number;
} Employee;
```

What is the difference?

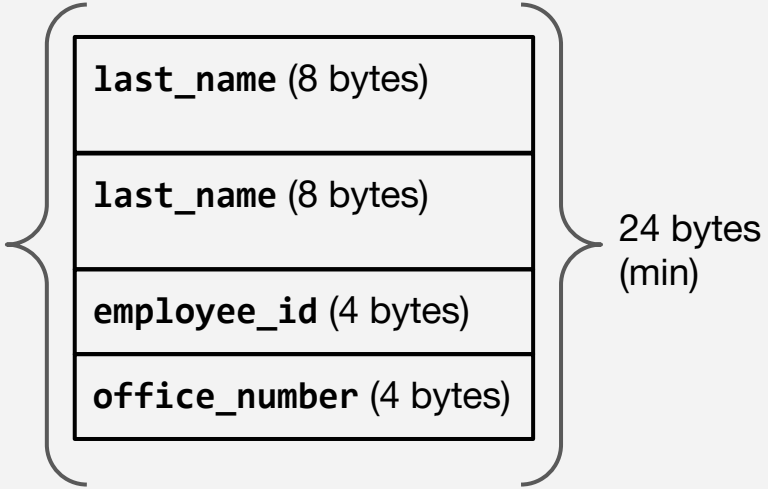
```c
typedef struct Employee {
  char first_name[20];
  char last_name[20];
  int employee_id;
  int office_number;
} Employee;
```

```c
typedef struct Employee {
  char* first_name;
  char* last_name;
  int employee_id;
  int office_number;
} Employee;
```

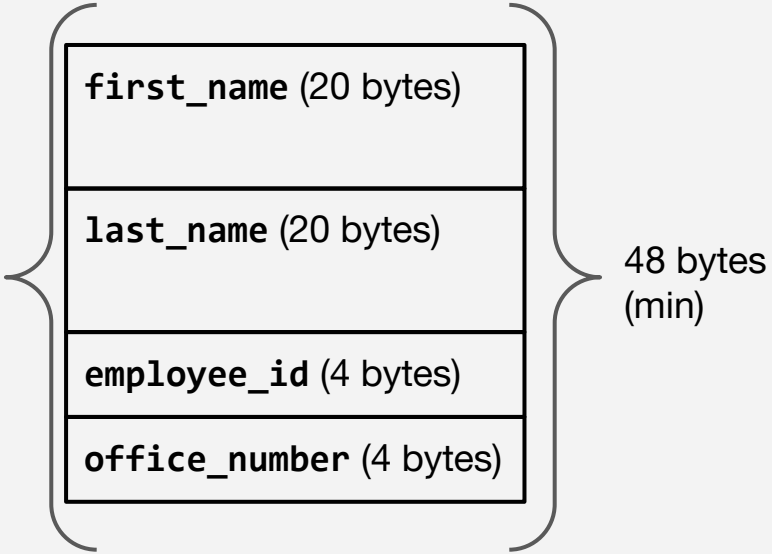
```c
typedef struct Employee {
  char first_name[20];
  char last_name[20];
  int employee_id;
  int office_number;
} Employee;
```



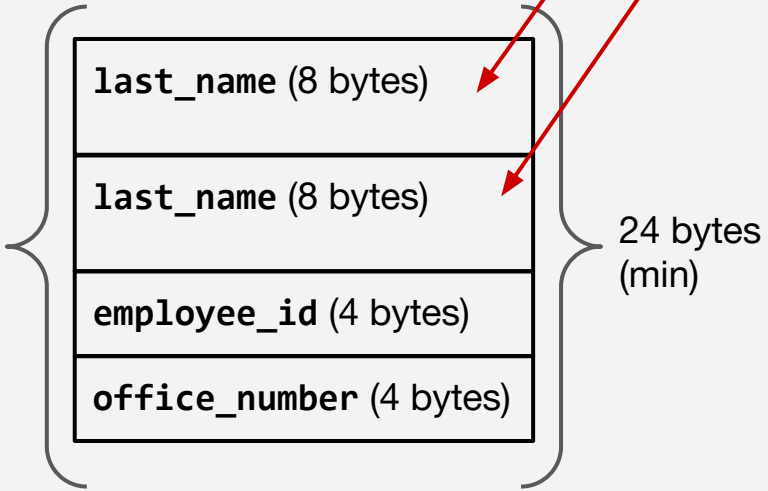

```c
typedef struct Employee {
  char* first_name;
  char* last_name;
  int employee_id;
  int office_number;
} Employee;
```



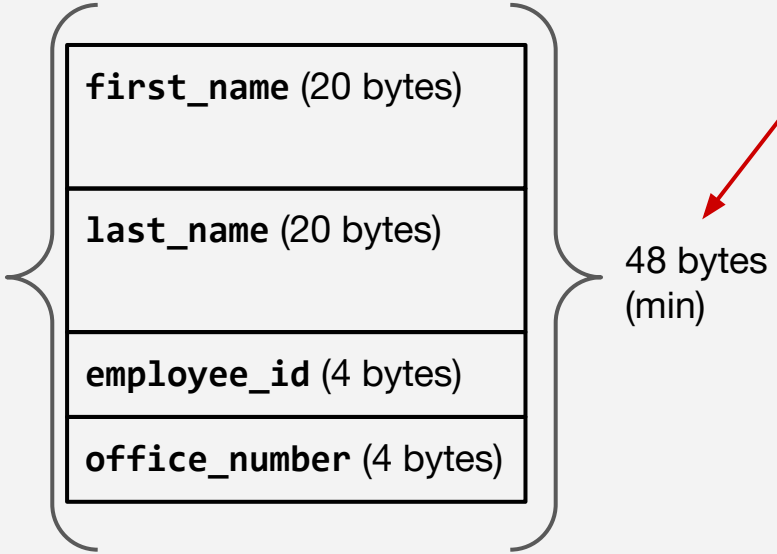
```c
typedef struct Employee {
  char first_name[20];
  char last_name[20];
  int employee_id;
  int office_number;
} Employee;
```




```c
typedef struct Employee {
  char* first_name;
  char* last_name;
  int employee_id;
  int office_number;
} Employee;
```



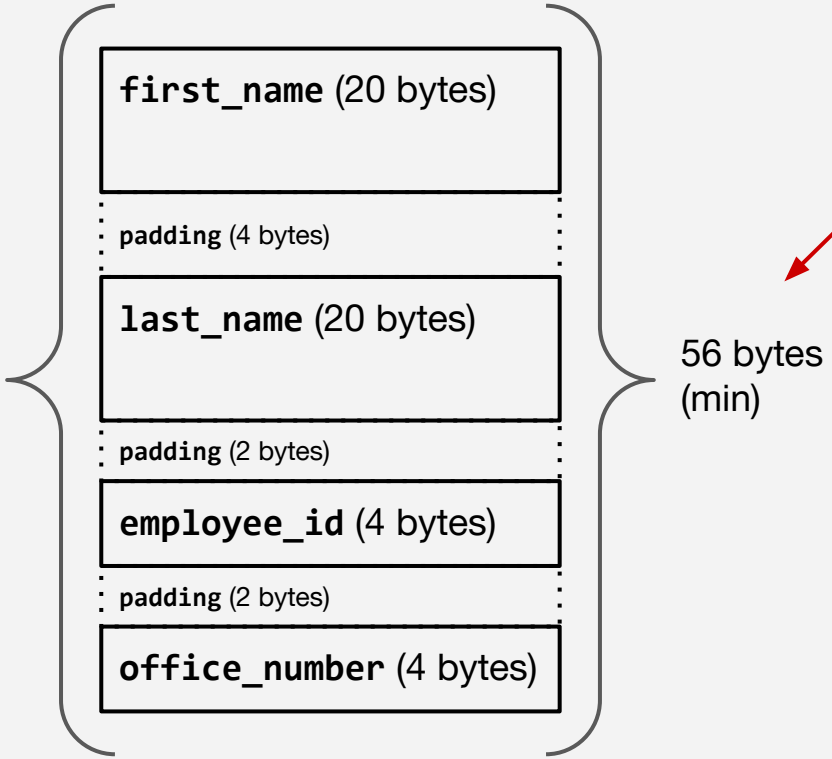
```c
typedef struct Employee {
  char first_name[20];
  char last_name[20];
  int employee_id;
  int office_number;
} Employee;
```



Not guaranteed to be exactly 48 bytes

According to the C standard, padding may be added between elements (but not at beginning)

For example


first_name (20 bytes)
padding (4 bytes)
last_name (20 bytes)
padding (2 bytes)
employee_id (4 bytes)
padding (2 bytes)
office_number (4 bytes)
56 bytes
(min)

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

typedef struct Coordinate2D {
  int x;
  int y;
} Coordinate2D;

void changeCoordinates(Coordinate2D c) {
  c.x = 25;
  c.y = 100;
}

int main() {
  Coordinate2D m = {0, 0};
  printf("Coordinates: %d,%d\n", m.x, m.y);
  changeCoordinates(m);
  printf("Coordinates: %d,%d\n", m.x, m.y);
  return 0;
}
```

What would this print?

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

typedef struct Coordinate2D {
  int x;
  int y;
} Coordinate2D;

void changeCoordinates(Coordinate2D c) {
  c.x = 25;
  c.y = 100;
}

int main() {
  Coordinate2D* m = calloc(sizeof(Coordinate2D), 1);
  printf("Coordinates: %d,%d\n", m->x, m->y);
  changeCoordinates(*m);
  printf("Coordinates: %d,%d\n", m->x, m->y);
  return 0;
}
```

What would this print?

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

typedef struct Coordinate2D {
  int x;
  int y;
} Coordinate2D;

void changeCoordinates(Coordinate2D* c) {
  c->x = 25;
  c->y = 100;
}

int main() {
  Coordinate2D m = {0, 0};
  printf("Coordinates: %d,%d\n", m.x, m.y);
  changeCoordinates(&m);
  printf("Coordinates: %d,%d\n", m.x, m.y);
  return 0;
}
```

What would this print?

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

typedef struct Coordinate2D {
  int x;
  int y;
} Coordinate2D;

void changeCoordinates(Coordinate2D* c) {
  c->x = 25;
  c->y = 100;
}

int main() {
  Coordinate2D* m = calloc(sizeof(Coordinate2D), 1);
  printf("Coordinates: %d,%d\n", m->x, m->y);
  changeCoordinates(m);
  printf("Coordinates: %d,%d\n", m->x, m->y);
  return 0;
}
```

What would this print?