Lebanese University
Faculty of Science
BS Computer Science
2nd Year – S3

I2204 - Imperative Programming

Dr Siba Haidar

Lebanese University
Faculty of Science
BS Computer Science
2nd Year – S3

Structures

Chapter 3

Structures

1. Definition, Use and the Dot Operator
2. typedef & sizeof
3. Passing Structures
4. Nested Structures
5. Pointers to Structure

Structures

1. Definition, Use and the Dot Operator
2. typedef & sizeof
3. Passing Structures
4. Nested Structures
5. Pointers to Structure

Structure Definition

A struct is a composite data type (or record) declaration that defines a physically grouped list of variables under one name in a block of memory.

Dr. Seuss' belongings:

- a fish
- an umbrella
- a coffee cup
- a book

Structure Definition

- keyword **struct**
- a structure declaration is a template that may be used to create structure variables | objects | instances
- members are the internal variables that make up the structure, also called elements | fields
- a structure provides convenient means of keeping related information together
- members of a structure **are logically related**

Structure Declaration

```
struct struct_type_name {
  type member_name;
  type member_name;
  type member_name;
  ...
} struct_variables;
```

- **either** *struct_type_name* ***or*** *structure_variables may be omitted but not both*

- example: employee's info
  - name: char[30]
  - salary: float
  - phone : unsigned long

```
struct employee {
  char name[30];
  float salary;
  unsigned long phone;
};
```

Structure Declaration

1. can declare the data type alone , then declare variables of this type

```c
struct employee {
  char name[30];
  float salary;
  unsigned long phone;
};

struct employee  e;
```

2. can declare struct type + variables at once

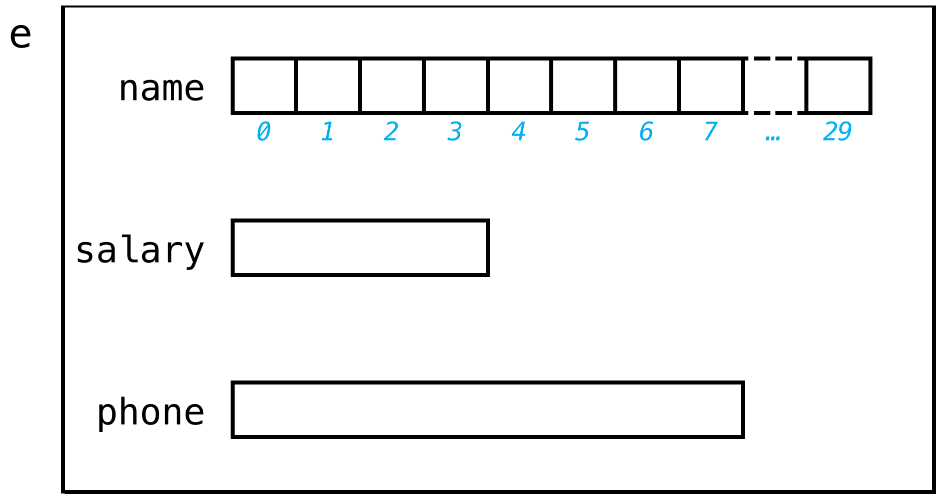
```c
struct employee {
  char name[30];
  float salary;
  unsigned long phone;
} a, b, c;
```

3. can declare anonymous struct type + variables of this type

```c
struct {
  char name[30];
  float salary;
  unsigned long phone;
} a, b, c;
```

Structure Representation in the Memory

```
struct employee {
  char name[30];
  float salary;
  unsigned long phone;
} e;
```



Structure Initialization

```c
struct employee {
  char name[30];
  float salary;
  unsigned long phone;
};
```

1. *C89-style initializer*

```c
struct employee e = {"Alix",
1590.5, 96170123456};
```

2. *designated initializer* (not supported by some compilers)

```c
struct employee e = {.salary =
1590.5, .name = "Alix"};
```

- omitted elements are initialized to their default values

The . (dot) Operator

```
struct employee {
  char name[30];
  float salary;
  unsigned long phone;
} e;
```

- to access the members of a structure variable, use the . (*dot*) operator
- examples

```
e.phone = 70123456;

printf("%lu\n", e.phone);

fgets (e.name, 30, stdin);
```

  - also name can be addressed as an array of characters as usual

Exercise: struct Student

- Write a C program in which,
  - you define a structure type for student containing a name, an ID, and grades for 6 courses.
  - declare a structure variable of type student and initialize it.
  - then calculate and display the student’s average.

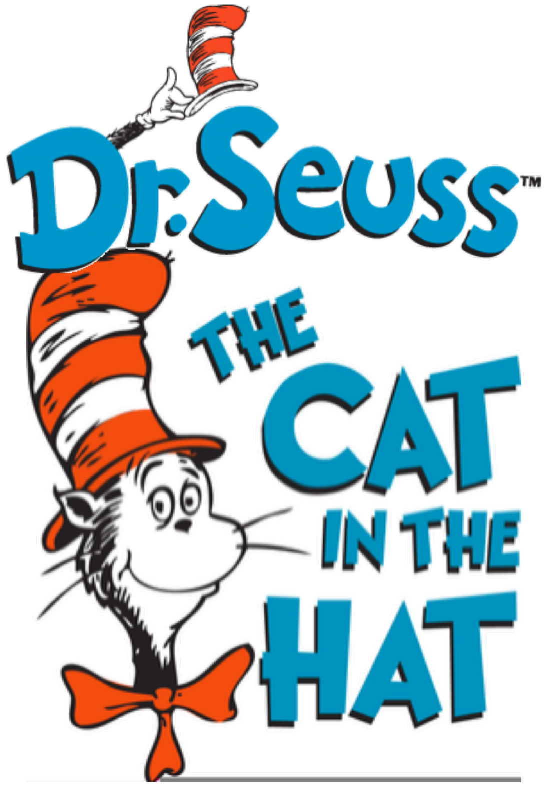
```c
# include <stdio.h>

int main(){

  struct student{
    char name[20];
    int id;
    float grades[6];
  };

  int i;
  float sum = 0.0;
  struct student s = {"Dr Seuss", 123, 90.0, 99.9, 
80.0, 87.5, 100.0, 75.0};
  for(i=0;i<6;i++)
  sum += s.grades[i];

  printf("average = %.1f\n", sum /6 );

  return 0;
}
```

Structures

The *typedef* Keyword

- typedef is a keyword used in C language to assign alternative names to existing datatypes.
  - not actually creating a new data type

```
typedef existing_type new_name;

typedef unsigned long ulong;

typedef unsigned int  unit;
```



typedef and Structures

- typedef can be used to give a name to user defined data type as well.

```c
typedef struct employee {
  char name[30];
  float salary;
  unsigned long phone;
} emp;

emp e;
```

- can use same type_name to get rid of keyword struct in variable declarations!

```c
typedef struct employee {
  char name[30];
  float salary;
  unsigned long phone;
} employee;

employee e;
```

typedef and Structures

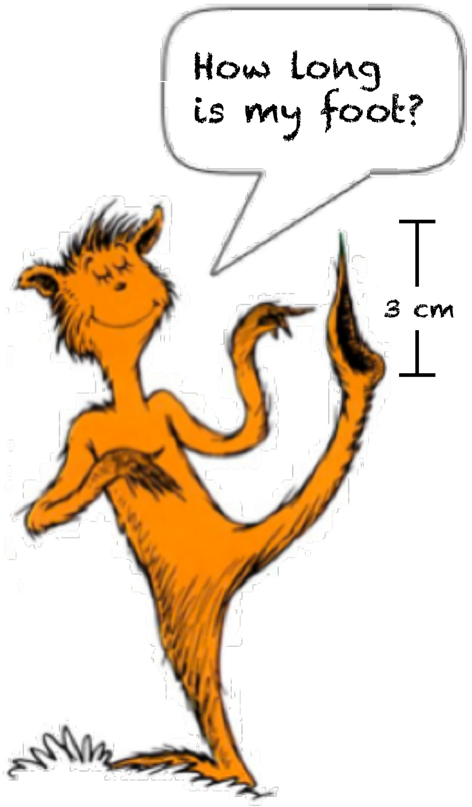
- can also use it with anonymous type declarations!

```
typedef struct {
   char name[30];
   float salary;
   unsigned long phone;
} employee;

employee e;
```

- beware when typedef keyword is present you cannot declare structure variables at the same time of structure declaration
  - employee is new type name for the anonymous defined struct

The *sizeof* Operator

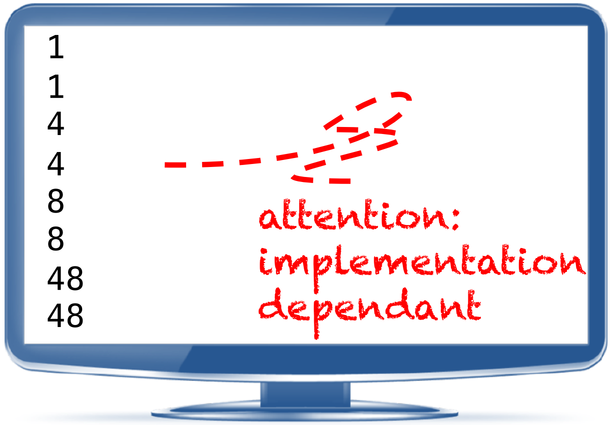
- sizeof is a unary operator that generates the storage size of an expression or a data type, measured in the number of char-sized units.
  - sizeof (char) is guaranteed to be 1
- return type is *size_t*
  - unsigned integer *(typedef implemention dependant)*
- single operand, either an expression or a data type cast
  - a cast is a data type enclosed in parenthesis



Demo: The *sizeof* Operator

```c
#include <stdio.h>
#include <string.h>
typedef struct  {
  char name[30];
  float salary;
  unsigned long phone;
} employee;
int main(void){
  char n[30]="Alix";
  float s=1000.0;
  unsigned long p = 96170123456;
  employee e;
  strcpy(e.name, n);
  e.salary = s;
  e.phone = p;
  printf("%lu\n", sizeof n[0]);
  printf("%lu\n", sizeof(char));
  printf("%lu\n", sizeof s);
  printf("%lu\n", sizeof(float));
  printf("%lu\n", sizeof p);
  printf("%lu\n", sizeof(unsigned long));
  printf("%lu\n", sizeof e);
  printf("%lu\n", sizeof(employee));
  return 0;
}
```

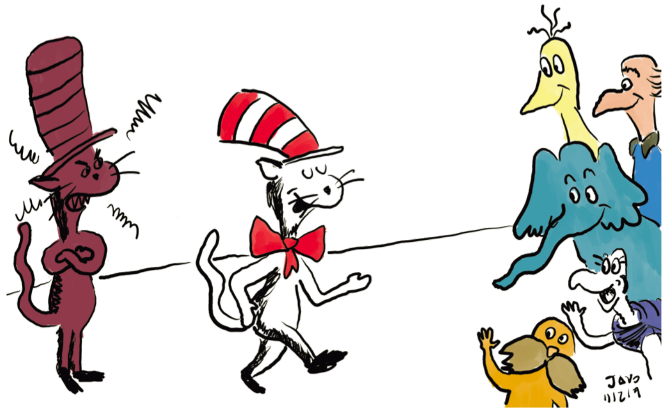
$$48 = sizeof(e) \geq \sum sizeof\ its\ fields = 42$$



```
1
1
4
4
8
8
48
48
```

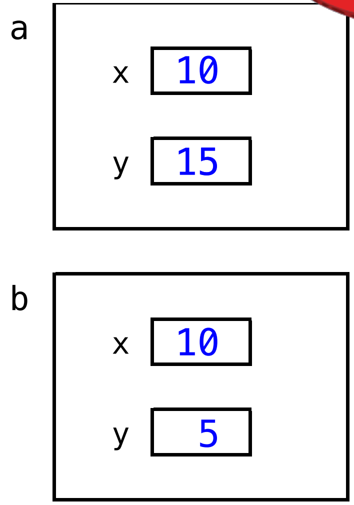
attention: implementation dependant

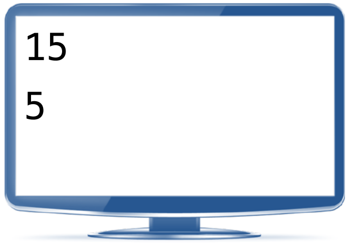
Structures



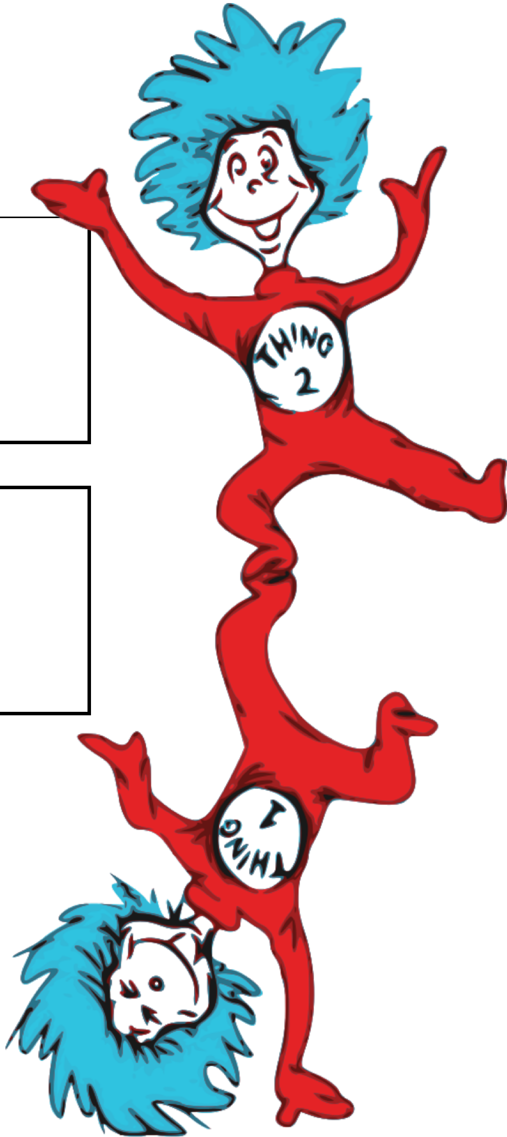
Structure Assignment

```c
#include <stdio.h>

int main(void){

   typedef struct { int x, y;} Point;

   Point a = {10, 5}, b;

   b = a;

   a.y += 10;

   printf("%d\n", a.y);
   printf("%d\n", b.y);
   return 0;
}
```



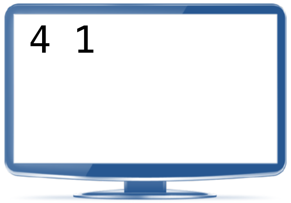


```
15
5
```

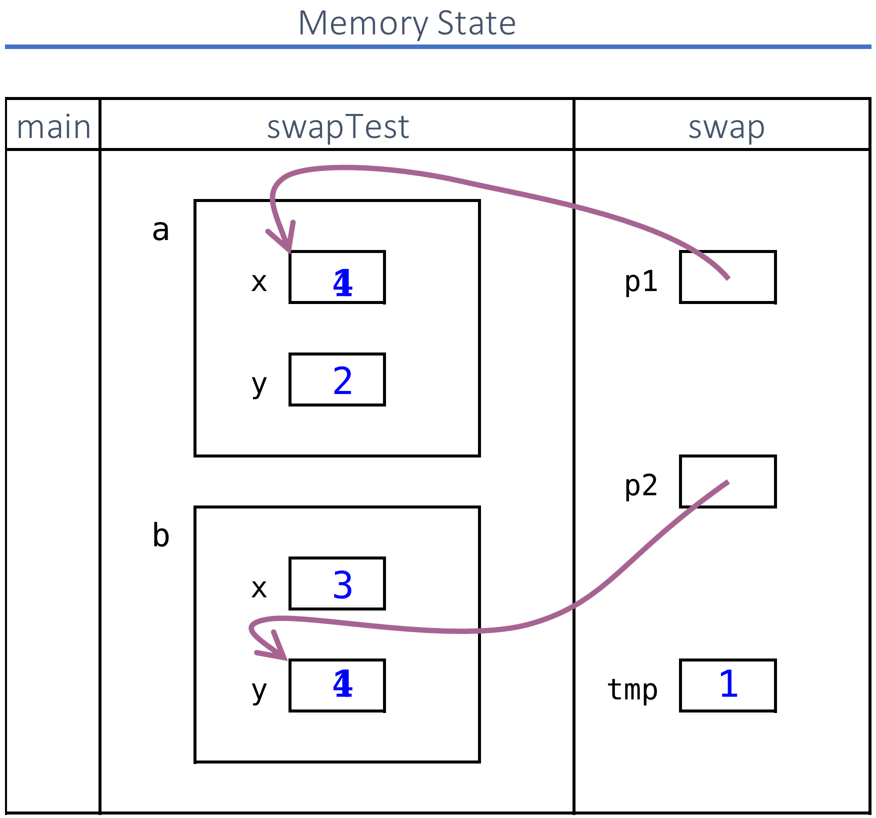


Passing Structure Members

```c
#include <stdio.h>
void swap(int *p1, int *p2){
   int tmp = *p1;
   *p1 = *p2;
   *p2 = tmp;
}
void swapTest(){
   typedef struct { int x, y;} Point;
   Point a = {1, 2}, b = {3,4};
   swap(&a.x, &b.y);
   printf("%d %d\n", a.x, b.y);
}
int main(){
   swapTest();
   return 0;
}
```



```
4 1
```



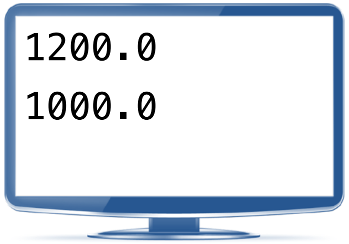
Passing Structures

- recall that, passing data from argument in the call to the parameter in the function behaves exactly like an assignment operation.
- type of argument must match type of parameter
- to be visible from both caller and called functions, you must make global the declaration of the structure type

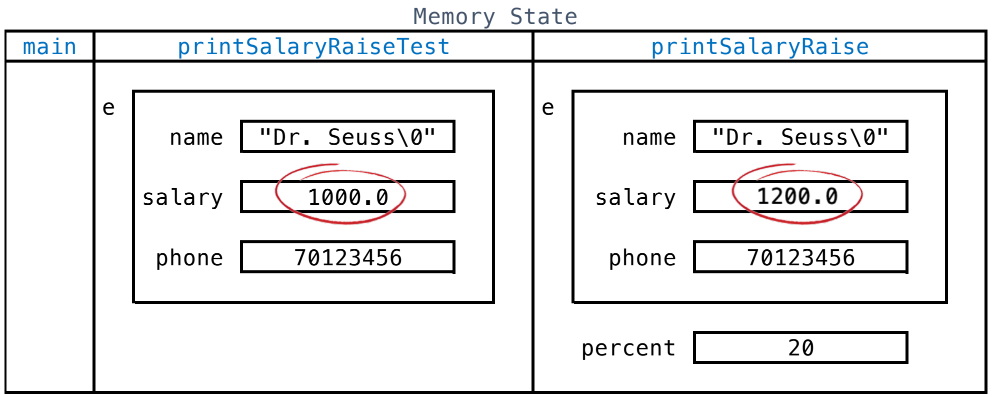
Example: Passing Structures

```c
#include <stdio.h>
typedef  struct  {
  char name[30];
  float salary;
  unsigned long phone;
} employee;

void printSalaryRaise(employee e, int percent){
  e.salary += e.salary * percent / 100;
  printf("%.1f\n", e.salary);
}

void printSalaryRaiseTest(){
  employee e = {"Dr. Seuss", 1000.0,
               70123456};
  printSalaryRaise(e, 20);
  printf("%.1f\n", e.salary);
}
```

```c
int main(){
  printSalaryRaiseTest();
  return 0;
}
```



```
1200.0
1000.0
```



Find the Mistake

```
#include <stdio.h>
struct type1{
   int a, b;
   char ch;
};

struct type2{
   int a, b;
   char ch;
};
```

```
void f1(struct type2 parm){
   printf("%d\n", parm.a);
}

int main(void){
   struct type1 arg;
   arg.a = 1000;
   f1(arg);
   return 0;
}
```

type mismatch:
Passing 'struct type1' to
parameter of incompatible type
'struct type2'

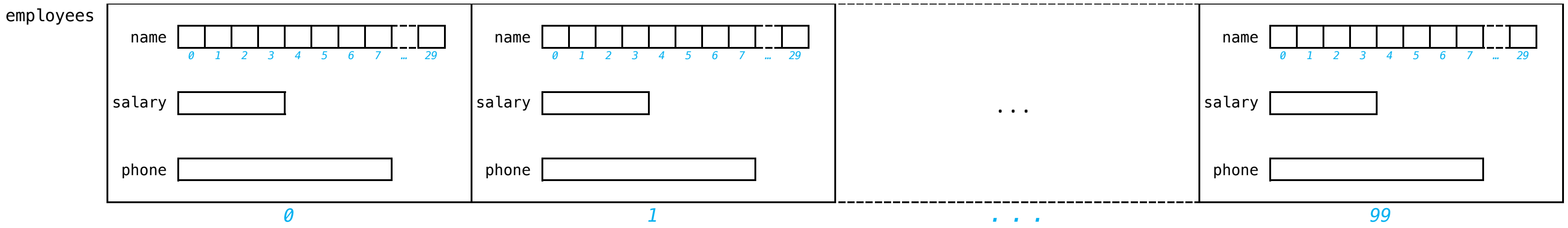
Structures

Arrays of Structures

- most common usage
- declare array of structures
  - define a structure
  - declare an array variable of that type
- example
  - in a company, there is more than one employee
  - declare 100-element array of structures of type employee

```
#include <stdio.h>
typedef struct  {
  char name[30];
  float salary;
  unsigned long phone;
} employee;
int main(void){
  employee employees[100];






  return 0;
}
```

Arrays of Structures



- example
  - in a company, there is more than one employee
  - declare 100-element array of structures of type employee

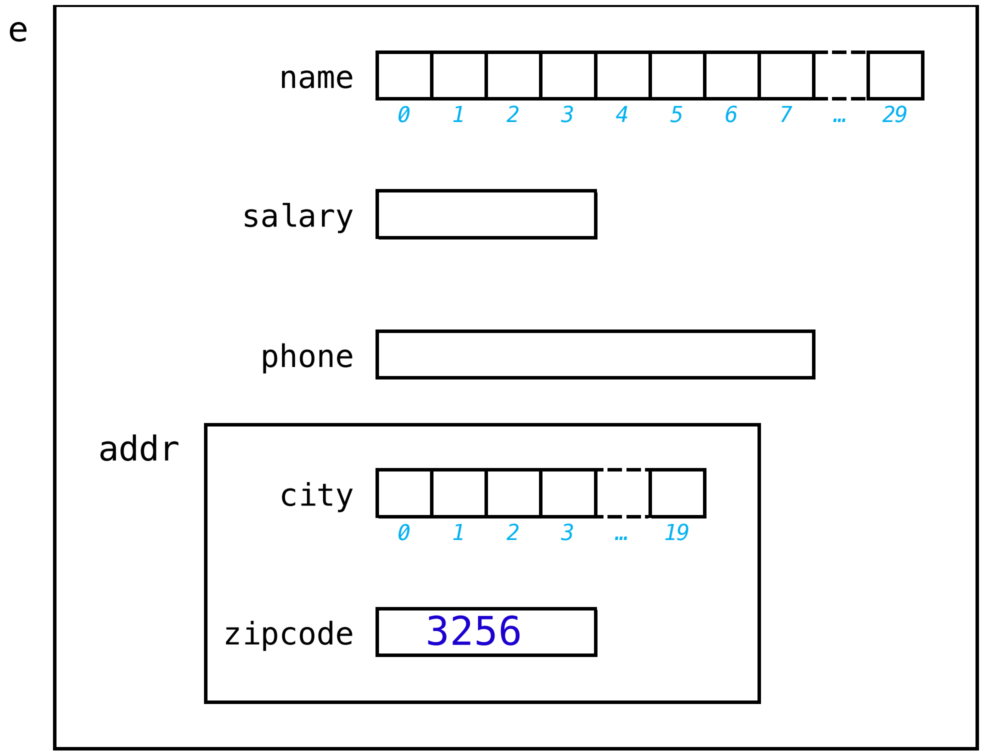
```
int main(void){
  employee employees[100];
  int i;
  // ...
  //print list of names + salaries:
  for (i=0;i<100;i++){
    printf("%s's salary: ", employees[i].name);
    printf("%.1f\n", employees[i].salary);
  }
  return 0;
}
```

Nested Structures

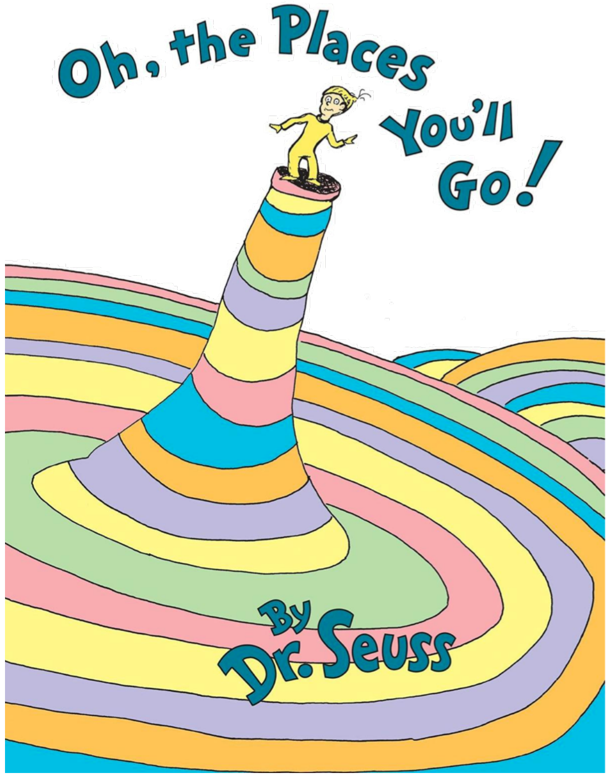
- members of structure may be of
  - simple type, or
  - compound type: 1D arrays, multidimensional arrays, other data types

Nested Structures

```
typedef struct  {
  char city[20];
  int zipcode;
} address;

typedef struct  {
  char name[30];
  float salary;
  unsigned long phone;
  address addr;
} employee;

//...
employee e;
e.addr.zipcode = 3256;
```

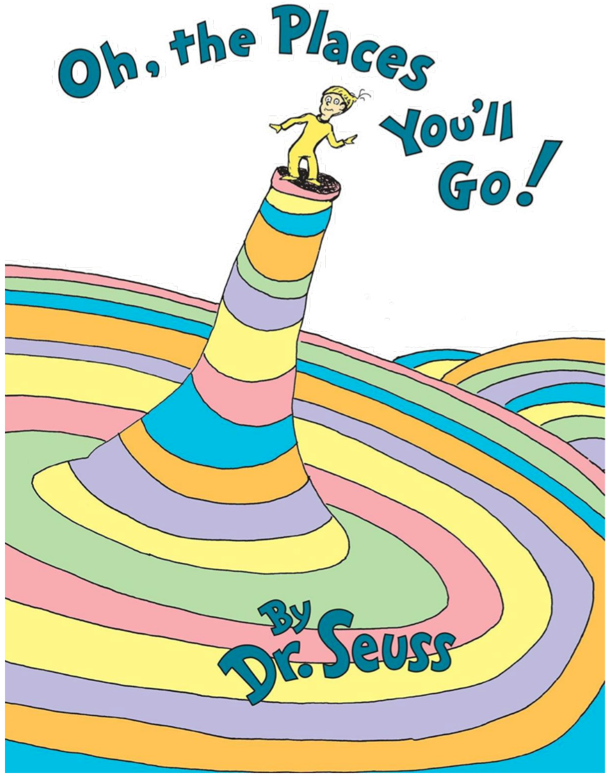


Structures



1. Definition, Use and the Dot Operator
2. typedef & sizeof
3. Passing Structures
4. Nested Structures
5. Pointers to Structure

Pointers to Structures



- C allows pointers to structures just as it allows pointers to any other type of variable

- example

```
employee * pe;
```

- 2 primary uses for structure pointers
  - pass a structure to a function using call by reference
  - create linked lists and other dynamic data structures that rely on dynamic allocation

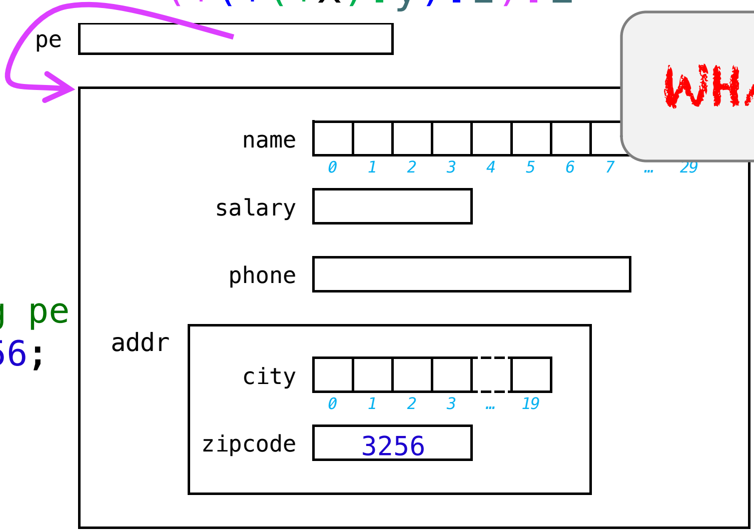
The -> (Arrow) Operator

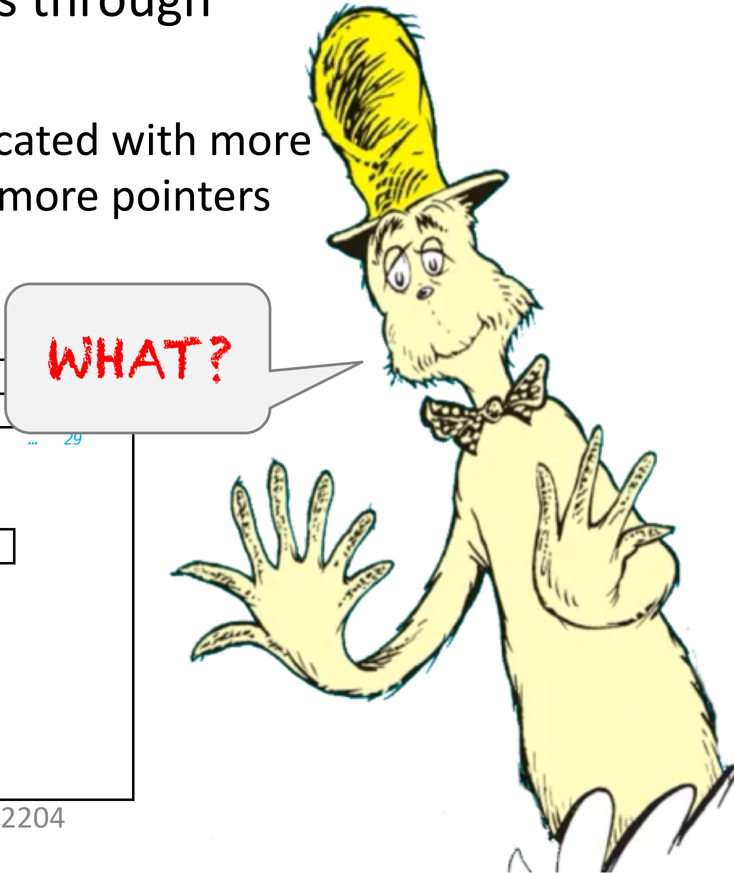
```
typedef struct  {
   char city[20];
   int zipcode;
} address;

typedef struct  {
   char name[30];
   float salary;
   unsigned long phone;
   address addr;
} employee;

employee e = ...;
employee * pe = &e;

//access zipcode using pe
(*pe).addr.zipcode=3256;
```

- accessing members through pointers

becomes more complicated with more nested structures and more pointers

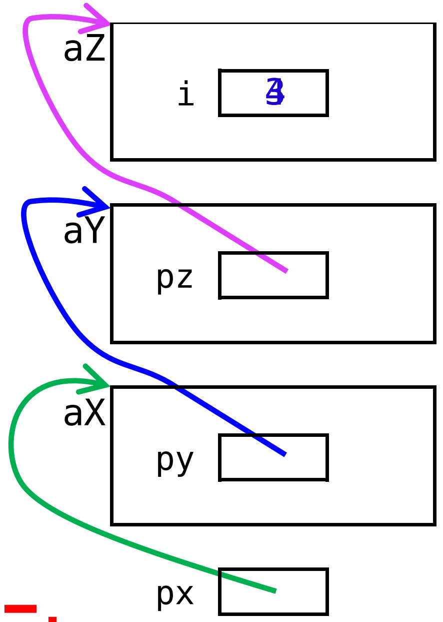
```
(*(*(*x).y).z).i
```





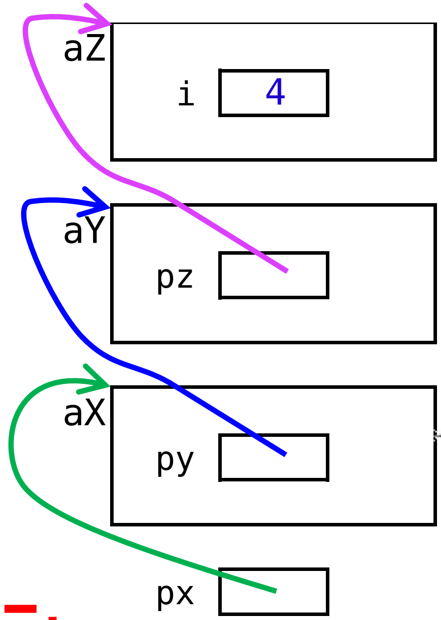
The -> (Arrow) Operator

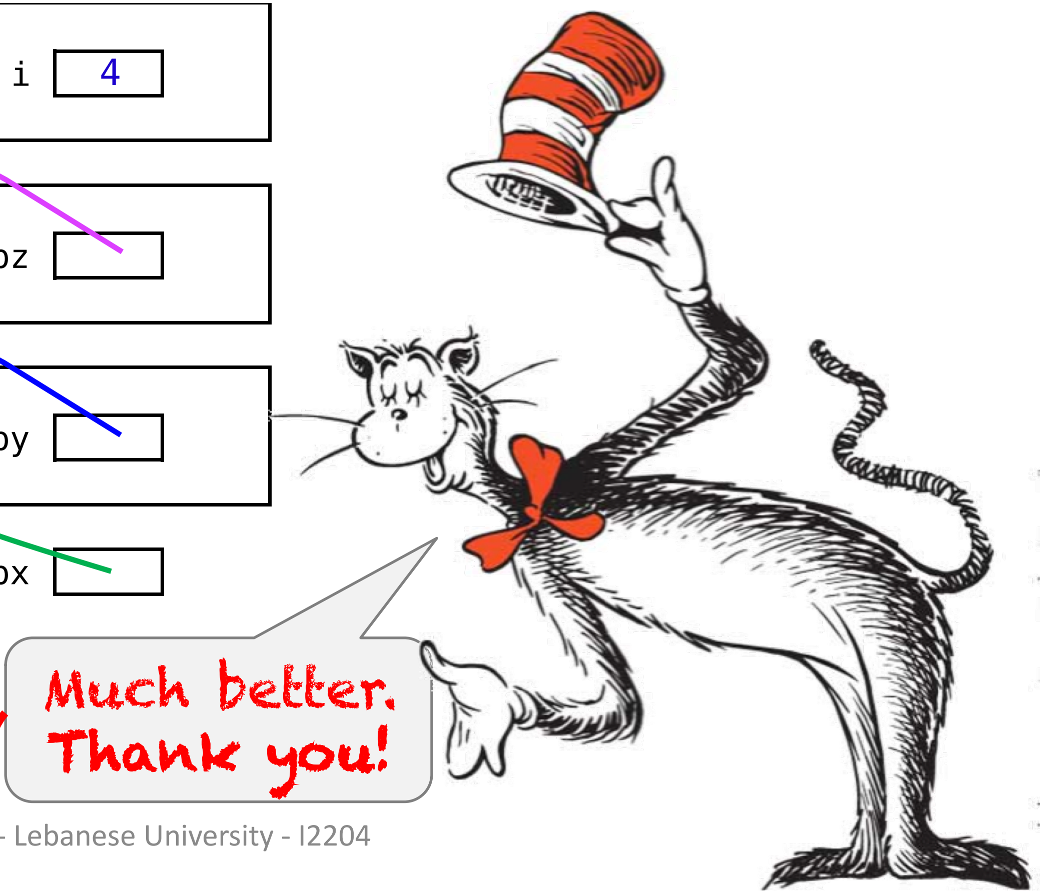
```c
#include <stdio.h>
typedef struct {int i;} Z;
typedef struct {Z * pz;} Y;
typedef struct {Y * py;} X;
int main(void){
  Z aZ = {3};
  Y aY = {&aZ};
  X aX = {&aY};
  X *px = &aX;
  printf ("%d\n", aZ.i);
  // via pX put 4 instead 3
  (*(*(*px).py).pz).i = 4;
  printf ("%d\n", aZ.i);
  return 0;
}
```



The -> (Arrow) Operator

```c
#include <stdio.h>
typedef struct {int i;} Z;
typedef struct {Z * pz;} Y;
typedef struct {Y * py;} X;
int main(void){
  Z aZ = {3};
  Y aY = {&aZ};
  X aX = {&aY};
  X *px = &aX;
  printf ("%d\n", aZ.i);
  // via pX put 4 instead 3
   px->py->pz->i = 4;
  printf ("%d\n", aZ.i);
  return 0;
}
```

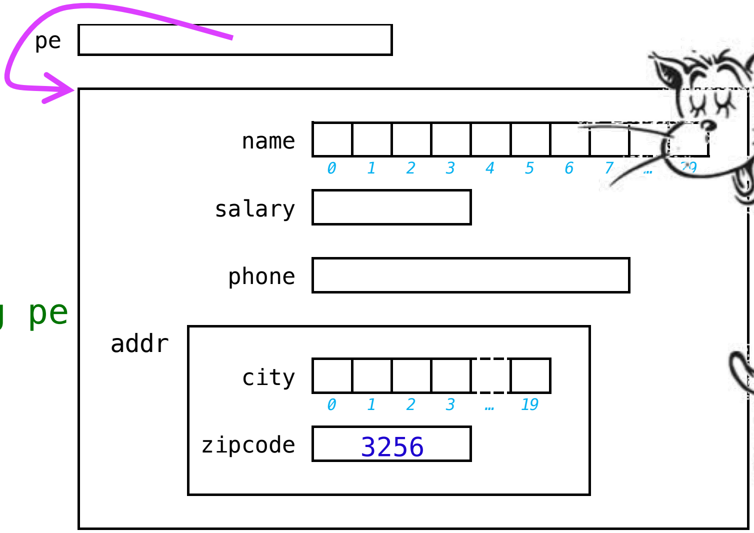


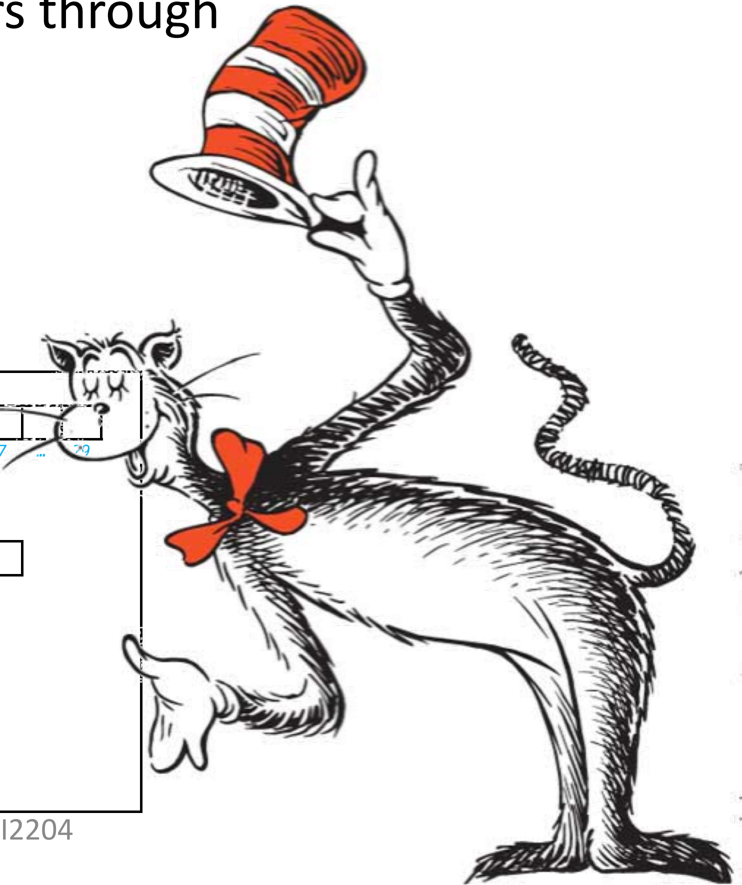


The -> (Arrow) Operator

```
typedef struct  {
  char city[20];
  int zipcode;
} address;

typedef struct  {
  char name[30];
  float salary;
  unsigned long phone;
  address addr;
} employee;

employee e = ...;
employee * pe = &e;

//access zipcode using pe
pe->addr.zipcode=3256;
```

- accessing members through pointers

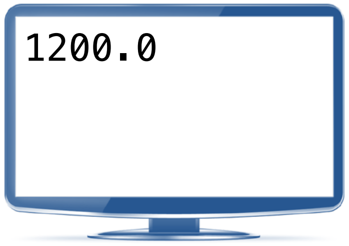




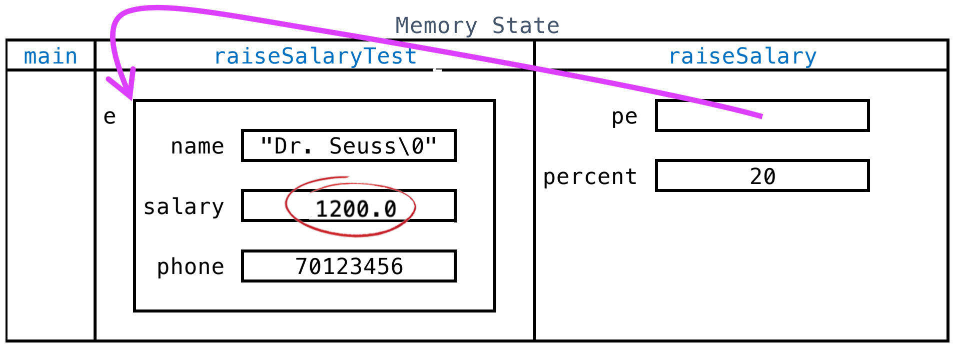
Example: Passing Structure Reference

```
#include <stdio.h>
typedef  struct  {
  char name[30];
  float salary;
  unsigned long phone;
} employee;

void raiseSalary(employee *pe, int percent){
  pe->salary += pe->salary * percent / 100;
}

void raiseSalaryTest(){
  employee e = {"Dr. Seuss", 1000.0,
                              70123456};
  raiseSalary(&e, 20);
  printf("%.1f\n", e.salary);
}
```

```
int main(){
  raiseSalaryTest();
  return 0;
}
```



```
1200.0
```



Recursive Structures

- what is the meaning of

struct rec {int i; ~~struct rec r;~~};

  - it is impossible to allocate a variable of this type in the memory
  - so, without pointer → not allowed

- with pointer

```
struct rec {int i; struct rec *r;};
```

  - of course allowed
  - → next chapter

typedef and Pointers

- typedef can be used to give an alias name to pointers also

```
int* x, y;
```

– declares `x` of type `int*`, however `y` of type `int`

```
typedef int* IntPtr;
IntPtr x, y, z;
```

– declare any number of pointers in a single statement

```
typedef struct t{ int a,b; } * u;
```

– declares u as an alias name for `struct t*`

Exercise: Use of (->) Memory State

```
# include <stdio.h>
typedef struct s{
  int i;
  struct s* s1;
} t;
int main(){
  t a, b, *c;
  a.i = 10;
  b.i = 5;
  a.s1 = b.s1 = &b;
  a.s1->i = 3;
  printf("%d\n", b.i);
  c = &a;
  printf("%d\n", c->s1->i);
  return 0;
}
```

- Can you draw the memory state?