# Warning

BU VIDEO TÜMÜYLE AŞAĞIDA BELİRTİLMİŞ LİSANS ALTINDADIR. THIS VIDEO, AS A WHOLE, IS UNDER THE LICENSE STATED BELOW.

#### Türkçe:

Creative Commons Atıf-GayriTicari-Türetilemez 4.0 Uluslararası Kamu Lisansı https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode.tr

#### English:

Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International Public License https://creativecommons.org/licenses/by-nc-nd/4.0/legalcode

LISANS SAHIBI ODTÜ BILGISAYAR MÜHENDISLIĞI BÖLÜMÜDÜR.
METU DEPARTMENT OF COMPUTER ENGINEERING IS THE LICENCE OWNER.

#### LİSANSIN ÖZÜ

Alıntı verilerek indirilebilir ya da paylaşılabilir ancak değiştirilemez ve ticari amaçla kullanılamaz.

#### LICENSE SUMARY

Can be downloaded and shared with others, provided the licence owner is credited, but cannot be changed in any way or used commercially.

# **CEng 140**

Structures

#### Structures

 A structure is a collection of logically related data items grouped together under a single name, i.e., a structure tag.

- Why do we need them?
  - 1. Grouping
  - 2. Modularity
  - 3. Flexibility
  - 4. ...

#### Structures

 A structure is a collection of logically related data items grouped together under a single name, i.e., a structure tag.

 Data items of a structure are called its members, components, or fields; and can be of different types.



# A structure defines a **new type**







# **Defining Structures**

```
name of the structure (optional)
struct [tag]
{ variable declarations
};

type and name declarations
for member data items
```

# Example

```
struct person
                       struct date
                        { int day, month, year; };
   int tc-id;
   int age;
                       struct course
  double weight;
                       { int course id;
  char gender;
                        int student no;
 };
                        double avg_grade;
                       };
```

# Naming Structure Tags & Member Variables

 Name of a member variable can be the same as its tag (as they will be differentiated by the context).

```
struct person
{
  int person; /*no confusion with the tag name */
  int age;
  double weight;
  char gender;
};
```

# Naming Structure Tags & Member Variables

 Name of a member variable or a tag can be the same as that of some **non-member** variable.

```
struct person
{ int person; /*no confusion with the tag name */
  int age;
  double weight;
  char gender; };

int person;
double weight;
```

# Naming Structure Tags & Member Variables

• Two member variables in different structures can have the same name.

```
struct person
{ int person;
  int age;
  double weight;
  char gender; };

int person;

double weight;
  char gender; };
struct human
{ int human;
  double weight;
  double height; };
```



# A structure defines a **new type**







- Defining a structure defines a <u>new type</u>
- Variables of this type can be declared as:

```
struct date
{
  int day, month, year;
} order_date, arrival_date;

variables of type struct date
```

```
struct date
{ int day, month, year; };
struct date order_date;
struct date arrival_date;
```

variables of type struct date

```
struct date
{
  int day, month, year;
} order_date, arrival_date;
```

day
order\_date month
year

day
arrival\_date month
year

 Tag may be omitted, but then all variables of this type should be declared when the structure is defined:

```
struct tag omitted
{ float r;
    float theta;
} polar1, polar2;

Then, we cannot later subsequently define a variable polar3 with the same type of polar1 and polar2!
```

• Each occurrence of a <u>structure definition</u> introduces a <u>new</u> structure type that is <u>neither</u> the same nor equivalent to any other type!

```
struct { char c; int i; } u;
struct { char c; int i; } v;
struct s1 { char c; int i; } w;
struct s2 { char c; int i; } x;
struct s2 y;
```

Which of these variables are the same type?

- a) ALL
- b) NONE
- c) u and v
- d) w, x and y
- e) x and y

 Each occurrence of a <u>structure definition</u> introduces a <u>new</u> structure type that is <u>neither</u> the same nor equivalent to any other type!

```
struct { char c; int i; } u;
struct { char c; int i; } v;
struct s1 { char c; int i; } w;
are all different!
struct s2 { char c; int i; } x;
struct s2 y;
```

Types of the variables **x** and **y** are the **same**!

 A structure definition does <u>not</u> allocate any storage; it merely describes the type!

 Storage is allocated only when a variable of the corresponding type is declared!

## Initialization of structure variables

```
struct date
   int day, month, year;
 } childrensDay = {23, 4, 1920};
struct date republicDay = {29, 10, 1923};
                                                 day
                                                            23
                                                 month
                                      childrensDay
                                                 year
                                                           1920
                                                            29
                                                  day
                                       republicDay
                                                            10
                                                 month
                                                           1923
                                                 year
```

#### Initialization of structure variables

```
struct date
{
  int day, month, year;
} childrensDay = {23, 4, 1920};
struct date republicDay = {29, 10, 1923};
```

 If there are fewer initializors, remaing ones are set to zero

```
struct date mybDay = {21};
```

If there are more initializors, error!

# Assignment of structure variables

 A structure variable may be assigned to another structure variable of the same type.

```
struct date
{
  int day, month, year;
} childrensDay = {23, 4, 1920};
struct date nationalDay, republicDay = {29, 10, 1923};
nationalDay = republicDay;
```

# Assignment of structure variables

```
struct date
   int day, month, year;
 } childrensDay = {23, 4, 1920};
struct date national Day;
                                                 day
                                                            23
nationalDay = childrensDay;
                                                 month
                                      childrensDay
                                                 year
                                                           1920
                                                            23
                                                 day
                                       nationalDay
                                                 month
                                                           1920
                                                 year
```

## Accesing Structure Members

 For accessing the members of a structure, we use the dot operator:

```
structure_var.member_name
struct date nationalDay;
nationalDay.day = 29;
nationalDay.month=10;
nationalDay.year= 1923;
```

- The dot operator has the same precedence with function call (), array subscript [], and arrow operator -> (but higher than any other C operator)
- And, it is left-to-right associative.

# Reminder

Operator	Туре	Associativity
Fucntion call: () Array subscript: []  Dot operator: Arrow operator:		Left to right
(type) + - ++ ! & * sizeof	Unary	Right to left
* / %	Binary	Left to right
+ -	Binary	Left to right
< <= > >=	Binary	Left to right
== !=	Binary	Left to right
&&	Binary	Left to right
	Binary	Left to right
= *= /= %= += -=	Binary	Right to left
,		Left to right

### Size of a struct

- You can use the size of operator on structures.
- The size of a struct may be more than the sum of the sizes of its members.
- For example:

```
struct st_type {char a; int b;} var1;
```

- → The size of var1 is probably more than 5 (due to data alignment with memory words)
- However, the following is probably 2 times the size of an int:

```
struct st_type2 {int a; int b;} var2;
```

#### Nested structures

You can use one structure within another:

```
struct date
 { int day, month, year; };
struct project
 { int no;
   struct date start date;
   struct date end date;
   float budget;
   int year; };
struct project myproject = {10, {1,1,2019}, {1,12,2019}, 25000, 2018};
```

### Nested structures

```
struct date
 { int day, month, year; };
struct project
  { int no;
                                            no
                                                                10
   struct date start_date;
                                                     day
                                            start_date
                                                     month
   struct date end_date;
                                   myproject
                                                     year
                                                              2019
   float budget;
                                                     day
   int year; };
                                           end_date
                                                     month
                                                                12
                                                     year
                                                              2019
struct project myproject =
                                            budget
                                                              25000
{10, {1,1,2019}, {1,12,2019}, 25000, 2018};
                                                              2018
                                            vear
```

#### Nested structures

- No limit on the depth of nesting!
- A member inside a nested structure can be accessed by repeatedly applying the dot operator

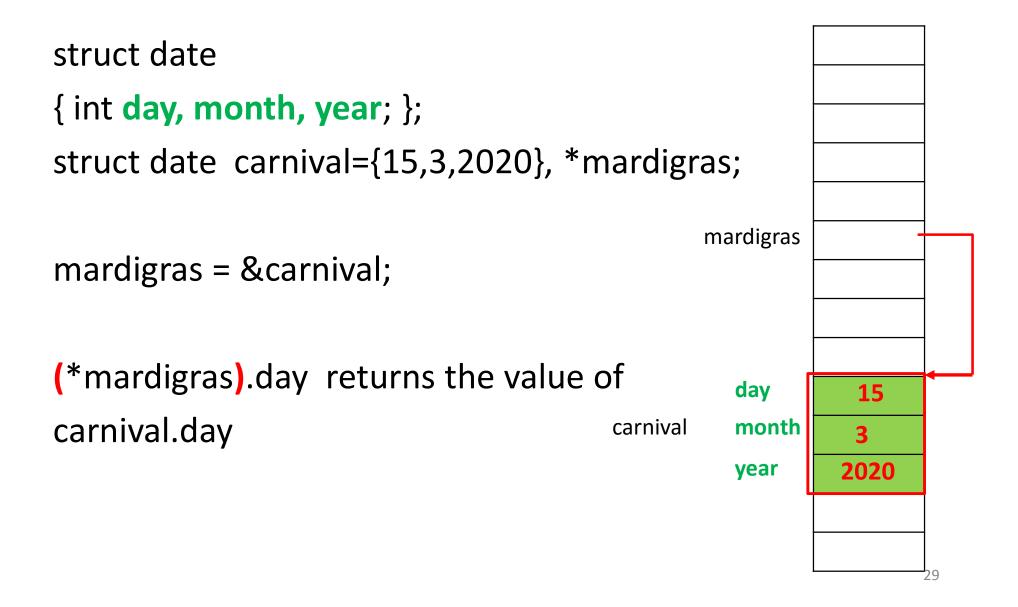
```
struct project myproject = {10, {1,1,2019}, {1,12,2019}, 25000, 2018}
myproject.no = 500;
myproject.start_date.month = 6;
struct date new_end_date = {1, 12, 2021};
myproject.end_date = new_end_date;
```

- A structure cannot be nested within itself (but it is possible to have a pointer to itself – LATER)
- A pointer to a structure var is created in the same way to a simple data type:

```
struct date carnival, *mardigras;

Pointer to a variable of type struct date

struct date
```



- A structure cannot be nested within itself (but it is possible to have a pointer to itself – LATER)
- A pointer to a structure var is created in the same way to a simple data type:

```
struct date carnival, *mardigras;

Pointer to a variable of type struct date

struct date
```

(\*mardigras).day returns the value of carnival.day

Parantheses are needed as dot op has higher precedence then deferenceing op \*. Without parantheses, meaning is \*(mardigras.day), which is an error!

(\*mardigras).day

Arrow operator: A special operator for accessing members of a structure variable pointed to by a pointer!

pointer\_name → member\_name

Instead of (\*mardigras).day we better write: mardigras \rightarrow day

→ has the same predence as dot operator and left-assoc.

 A pointer can point into the middle of a structure:

```
struct date *complete_date;
complete_date = &(myproject.end_date);
```

Parantheses **not** needed; dot operator has higher precedence

It is allowed to take the addresses of member vars of a structure variable

See the example at p.285 of your textbook!

```
struct date
                                                complete_date
 { int day, month, year; };
struct project
 { int no;
   struct date start_date;
                                             no
                                                                 10
                                                      day
   struct date end_date;
                                            start date
                                                      month
   float budget;
                                   myproject
                                                      year
                                                                2019
   int year; };
                                                      day
struct project myproject =
                                            end_date
                                                      month
                                                                 12
{10, {1,1,2019}, {1,12,2019}, 25000, 2018};
                                                      year
                                                                2019
                                            budget
                                                                25000
struct date *complete date;
                                                                2018
complete_date = &(myproject.end date);
```

```
int *month;
struct date *complete date;
                                               complete_date
complete date = &myproject.end_date;
                                                     month
month=&myproject.start date.month;
                                            no
                                                                 10
                                                     day
                                            start_date
                                                     month
/* print start month using ptr*/
                                   myproject
                                                               2019
                                                     year
printf("%d", *month);
                                                     day
/*print end month*/
                                            end_date
                                                     month
printf("%d", (*complete date).month);
                                                     year
                                                               2019
printf("%d", complete date \rightarrow month);
                                            budget
                                                               25000
                                                               2018
                                            year
```

- Using pointers to structures is better/faster than using structures directly especially in the case of function calls (for passing parameters or returning values).
- Using pointers to structures allows us to create sophisticated data structures.

# Reminder

Operator	Туре	Associativity
Fucntion call: () Array subscript: []  Dot operator: →  Arrow operator: →		Left to right
(type) + - ++ ! & * sizeof	Unary	Right to left
* / %	Binary	Left to right
+ -	Binary	Left to right
< <= > >=	Binary	Left to right
== !=	Binary	Left to right
&&	Binary	Left to right
II	Binary	Left to right
= *= /= %= += -=	Binary	Right to left
,		Left to right

#### Structures & Functions

#### Scope of a structure definition

The scoping rules for variable names apply also for struct definitions:

- With a <u>local</u> struct definition, only <u>local</u> variables of struct type can be defined.
- With a global definition (with a tag), you can have global & local variables of this struct type

## Example

```
struct global_date {int day, month, year; };
void f()
{ struct date {int d, m, y; };
 struct date valid = {01, 01, 2018}; /* var decl & init*/
 struct global date next = {02, 01, 2018}; }
struct global_date my_date = {20, 2, 2020};
void g()
{ struct date today; /* illegal!!! */
 struct global_date tomorrow; }
```

## Structures as Function Arg.s

 You can pass structures as parameters to a function, in 3 ways:

## Structures as Function Arg. (1)

1) Supply structure members as arguments in a function call separately; i.e., treat as non-structs

```
struct point { float x, y; };
struct circle { float r; struct point o; };
int contains (float cr, float cx, float cy, float px, float py)
{ return sqr(cx-px) + sqr(cy-py) > sqr(cr) ? 0 : 1 ; }
```

#### In main:

```
struct circle c = \{2, \{1, 1\}\}; struct point p = \{2, 2\}; contains(c.r, c.o.x., c.o.y, p.x, p.y); // function call
```

```
int contains (float cr, float cx, float cy, float px, float py)
{ return sqr(cx-px) + sqr(cy-py) > sqr(cr) ? 0 : 1 ; }
                                                               2
                                                        cr
                                                        CX
                                                        CY
                                                        рх
                                                        py
                                                      p
int main(void) {
struct circle c = \{2, \{1, 1\}\};
                                                   C
struct point p = \{2,2\};
                                                       0
contains(c.r, c.o.x., c.o.y, p.x, p.y); }
```

## Structures as Function Arg. (2)

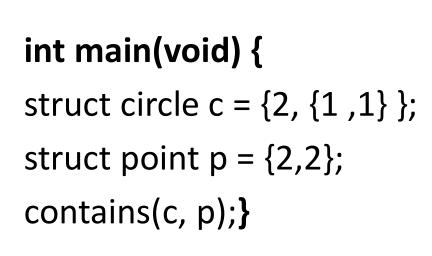
2) Pass the complete structure by simply providing the name of the structure var as the argument in the function call

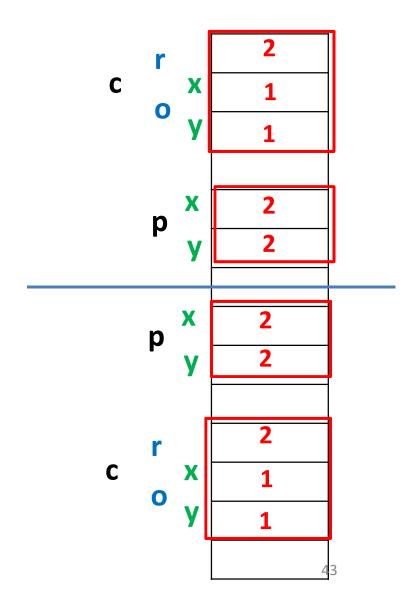
```
int contains (struct circle c, struct point p)
{ return sqr(cx-px) + sqr(cy-py) > sqr(cr) ? 0 : 1 ; }
{ return sqr(c.o.x-p.x) + sqr(c.o.y-p.y) > sqr(c.r) ? 0 : 1 ; }
```

#### In main:

```
struct circle c = {2, {1,1}}; struct point p = {2,2}; contains(c.r, c.o.x., c.o.y, p.x, p.y); // function call contains(c, p);
```

```
int contains (struct circle c, struct point p)
{ return sqr(c.o.x-p.x) + sqr(c.o.y-p.y) > sqr(c.r) ? 0 : 1 ; }
```





## Structures as Function Arg. (2)

2) Pass the complete structure by simply providing the name of the structure var as the argument in the function call

Unlike array names, structure names are NOT pointers, and hence, they're passed-by-value

- When a struct name is provided as argument, entire struct is copied to the called function (and changes are **not** reflected to calling func)
- If there is an array in the struct, it is also copied!

## Structures as Function Arg. (3)

```
3) Pass a pointer to the structure variable as the
argument in the function call
int contains (struct circle *c, struct point *p)
\{ return sqr(c.o.x-p.x) + sqr(c.o.y-py) > sqr(c.r) ? 0 : 1; \}
{ return sqr(c \rightarrow o.x - p \rightarrow x) + sqr(c \rightarrow o.y - p \rightarrow y) > sqr(c \rightarrow r)
                                                           ?0:1;}
In main:
struct circle c = \{2, \{1, 1\}\}; struct point p = \{2, 2\};
contains(c.r, c.o.x., c.o.y, p.x, p.y); // function call
contains(c, p);
contains(&c, &p); // changes to args made in func are
                        visible after returning to caller
                                                                 45
```

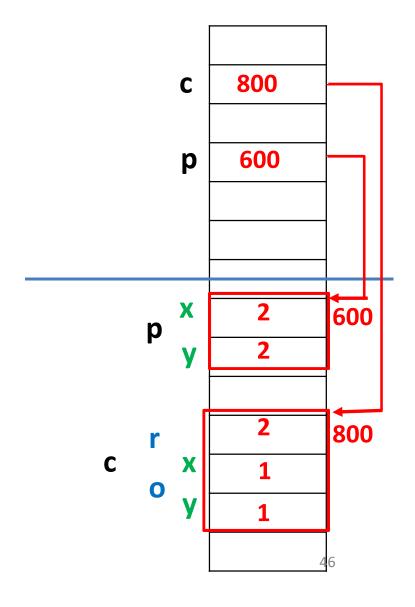
int contains (struct circle \*c, struct point \*p)

{ return  $sqr(c \rightarrow o.x - p \rightarrow x) + sqr(c \rightarrow o.y - p \rightarrow y) > sqr(c \rightarrow r)$ 

? **0:1**; }

#### int main(void) {

struct circle  $c = \{2, \{1, 1\}\};$ struct point  $p = \{2, 2\};$ contains(&c, &p);}



#### Structures as Func. Values

Structures may be returned as function values

```
struct rectangular { float x, y; };
struct polar { float r, theta; };
struct polar convert (struct rectangular rec)
{ struct polar pol;
 if (rec.x == 0 \&\& rec.y == 0)
    pol.r = pol.theta = 0;
                                    In main:
 else
                                    struct rectangular r={2 ,1};
 { pol.r= sqrt(rec.x * rec.x +...);
                                    struct polar p;
   pol.theta = ...; }
  return pol;
                                                              47
```

```
struct polar convert (struct rectangular rec)
{ struct polar pol;
  if (rec.x == 0 \&\& rec.y == 0)
    pol.r = pol.theta = 0;
                                                      rec
  else
  { pol.r= sqrt(rec.x * rec.x +...);
                                                                2.23
                                                  pol
                                                                0.46
   pol.theta = ...; }
                                                      theta
  return pol; }
int main(void) {
struct rectangular r={2,1};
                                                               2.23
struct polar p;
                                                               0.46
                                                      theta
p = <del>convert(r)</del>; }
    Substitute with the returned value
```

#### Structures as Func. Values

A func may return a pointer to the structure

```
struct rectangular { float x, y; };
struct polar { float r, theta; };
struct polar * convert (struct rectangular rec)
{ struct polar *p;
  p = (struct polar *) malloc(sizeof(struct polar));
 if (p)
                                      In main:
 \{ \text{ if (rec.x == 0 \&\& rec.y == 0)} \}
                                       struct rectangular r={2,1};
      p \rightarrow r = p \rightarrow theta = 0;
                                       struct polar *polp;
   else ... }
                                       polp = convert(r);
  return p;
```

```
struct polar * convert (struct rectangular rec)
{ struct polar *p;
  p = (struct polar *) malloc(sizeof(struct polar));
                                                               rec
  if (p)
  \{ \text{ if (rec.x == 0 \&\& rec.y == 0)} \}
                                                                    p
     p \rightarrow r = p \rightarrow theta = 0;
   else ... }
  return p; }
                                                                polp
int main(void) {
struct rectangular r={2,1};
struct polar *polp;
                                                          HEAP
polp = convert(r); }
                                                                theta
```

Substitute with the returned value

2.23

400

400

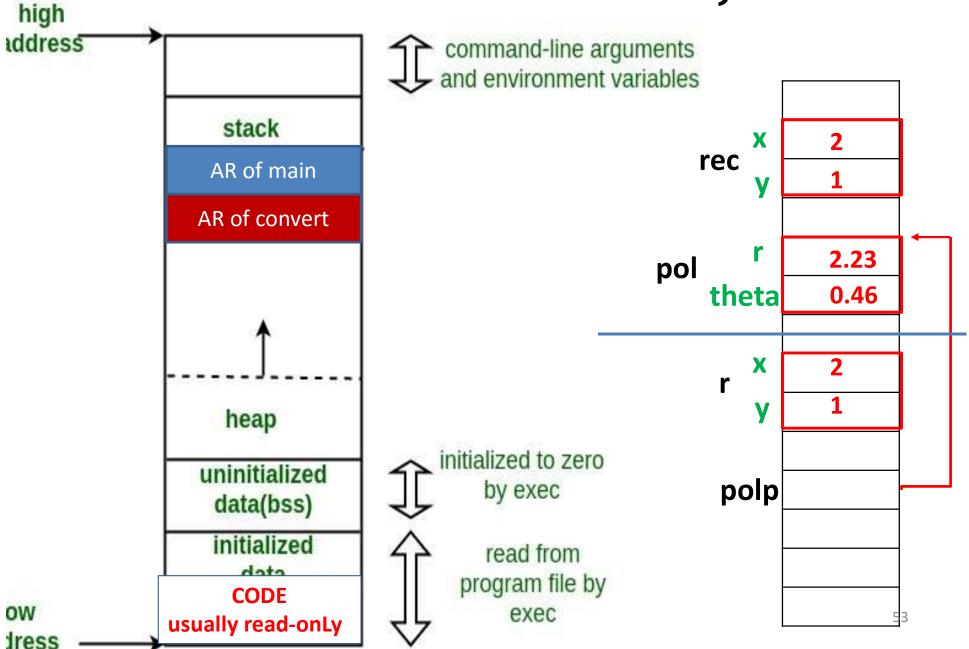
400

#### How about this version?

```
WRONG!!!!!
struct rectangular { float x, y; };
struct polar { float r, theta; };
struct polar * convert (struct rectangular rec)
{ struct polar pol;
 if (rec.x == 0 \&\& rec.y == 0)
    pol.r = pol.theta = 0;
                                   In main:
 else
                                   struct rectangular r={2 ,1};
 { pol.r= sqrt(rec.x * rec.x +...);
                                   struct polar *polp;
   pol.theta = ...; }
                                   polp = convert(r);
 return &pol;
                                                             51
```

```
struct polar * convert (struct rectangular rec)
{ struct polar pol;
  if (rec.x == 0 \&\& rec.y == 0)
    pol.r = pol.theta = 0;
                                                     rec
  else
  { pol.r= sqrt(rec.x * rec.x +...);
                                                               2.23
                                                 pol
                                                               0.46
   pol.theta = ...; }
                                                      theta
  return &pol; }
int main(void) {
struct rectangular r={2,1};
                                                      polp
struct polar *polp;
polp = <del>convert(r)</del>; }
    Substitute with the returned value
```

## Memory Layout of C Programs



#### Arrays & Structures

- Arrays and sturctures can be freely intermixed to create:
  - Arrays of structures
  - Structures containing arrays
  - Arrays of structures containing arrays...

## Arrays of structures

 Used when a large no of similar records are required to be processed together

```
struct date { int d, m, y; };
struct date bdays[3] = {{1, 1, 2012}, {1,9,1980}, {1,3,1995}};
```

## Arrays of structures

```
struct date { int d, m, y; };
                                               bday
                                                        412
struct date bdays[3] = {{1, 1, 2012},
  {1,9,1980}, {1,3,1995}};
                                                                  40b
                                                  d
                                                           1
struct date *bday;
                                          bdays[0]
                                                         2012
(*(bdays+1)).y
                                                           1
                                                  d
bdays[1].y = 1990;
                                                           9
                                          bdays[1]
bday = &bdays[1];
                                                         2000
                                                  V
bday \rightarrow y = 2000;
                                                  d
                                          bdays[2]
                                                         1995
                                                  y
                                                                   56
```

 A struct can contain an array name[0] 'a' struct person name[1] 'i' name[2] student1 { char name[5]; **'\0'** name[3] int id; name[4] } student1 = {"ali" , 100}; id 100 //or {{'a', 'l', 'i', '\0'}, 100};

57

A struct can contain an array

```
struct person
{ char name[5];
 int id;
} student1 = {"ali", 100}; //or {{'a', 'l', 'i', '\0'}, 100};
// btw, is this string modifiable or not?
// how do we assign name to "veli" ?
strcpy(student1.name, "veli");
//OR student1.name[0]='v'; student1.name[1]='e'; ...
printf("%c", student1.name[0]);
```

When a struct containing an array is passed as an argument to a function, the member array is passed-by-value! (even when it is the only member)
 struct time { int val[3]: } noon = {12, 0, 0}:

```
struct time { int val[3]; } noon = {12, 0, 0};
void advanceTime(struct time t)
{ int i;
 for (i=0; i<3; i++)
     t.val[i] += 5; }
int main
{ advanceTime(noon);
 printf("%d", noon.val[0]);}
```

```
struct time { int val[3]; } noon = {12, 0, 0};
void advanceTime(struct time t)
                                               val[0]
                                                         17
                                               val[1]
                                                         5
{ int i;
                                                val[2]
 for (i=0; i<3; i++)
     t.val[i] += 5; }
int main
{ advanceTime(noon);
 printf("%d", noon.val[0]);}
                                                val[0]
                                                         12
                                          noon val[1]
                                                          0
                                                val[2]
                                                          0
```

### Array of structs with arrays...

#### Array of structs with arrays...

```
name[0]
                                                        'a'
struct student
                                            name[1]
{ char name[5];
                                            name[2]
                                                         'i'
                                students[0]
                                                        '\0'
  int grades[3];
                                            name[3]
                                            name[4]
} students[3]={{"ali", {100, 80, 90}}, grades[0]
                                                        100
                 {"veli", {60, 50, 20}},
                                           grades[1]
                                                         80
                 {"jo", {10, 40, 25}}};
                                                         90
                                           grades[2]
                                            name[0]
                                                        'v'
                                                        'e'
                                            name[1]
                                            name[2]
                                                         4
                                            name[3]
                                                         'i'
                                students[1]
                                            name[4]
                                                        '\0'
                                            grades[0]
                                                        60
                                            grades[1]
                                                        50
                                                         20
                                           grades[2]
```

#### students[2]

name[0]	<b>'j'</b>
name[1]	'o'
name[2]	<b>'</b> \0'
name[3]	
name[4]	
grades[0]	10
grades[1]	40
grades[2]	25

# Array of structs with arrays...

```
sp
 struct student
 { char name[5];
                                                                             'a'
                                                                 name[0]
    int grades[3];
                                                                             4
                                                                 name[1]
                                                                             'i'
                                                                 name[2]
                                                     students[0]
 } students[3]={...};
                                                                             '\0'
                                                                 name[3]
students[0].grades[1] = 85; // 80 becomes 85
                                                                 name[4]
struct student *sp;
                                                                 grades[0]
                                                                             100
sp = &students[0]; // equivalent to?
                                                                 grades[1]
                                                                              85
                                                                              90
// print first stu name's first char
                                                                 grades[2]
                                                                 name[0]
printf("%c", sp\rightarrow name[0]
                                                     students[1]
// print his full name and second grade,
                                                                grades [2]
printf("%s %d", sp\rightarrow name, sp\rightarrow grades[1]);
                                                                 name[0]
                               *(sp\rightarrow grades + 1) students[2]
                                                                grades [2]
```

## A "ptr to a struct" is the parameter

```
void g(struct student *p)
struct student
{ int grade;
                                \{ p \rightarrow grade = 0; \}
                                  strcpy(p \rightarrow name, "TONY");
   char name[5]; };
                                                   A-75 Jane
int main()
                                                   B- 0 Tony
                                                   C- Compile error
{ struct student *sp;
                                                   D- Run time error
 sp = (struct student *) malloc (sizeof(struct student));
 sp -> grade = 75;
 \Rightarrow name, "JANE");
 printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?
 g(sp);
printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?}
```

```
void g(struct student *p)
                                                      5000
\{ p \rightarrow grade = 0; \}
                                                 p
  strcpy(p \rightarrow name, "TONY");
int main()
                                                 sp
                                                       5000
                                                                 乃
                                                                      5000
{ struct student *sp;
                                                                 'T'
 sp = (struct student *)
                                                                 'M'
                                                                 'n
              malloc (sizeof(struct student));
                                                                  'Y'
 sp -> grade = 75;
                                                                 4\0"
 \Rightarrow name, "JANE");
 printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?
 g(sp);
printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?}
```

## A "ptr to a struct" is the parameter

```
void g(struct student *p)
struct student
                          { p = (struct student *) malloc (...));
{ int grade;
                             p \rightarrow grade = 0;
   char name[5]; };
                            strcpy(p → name, "TONY", Jane
int main()
                                                       C- Compile error
{ struct student *sp;
                                                       D- Run time erro
 sp = (struct student *) malloc (sizeof(struct student));
 \Rightarrow grade = 75;
 strcpy(sp \rightarrow name, "JANE");
 printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?
 g(sp);
printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?
```

```
void g(struct student *p)
                                                     5000
                                                                      200
                                                 p
                                                                  'T'
{ p = (struct student *) malloc (...));
                                                                  'O'
  p \rightarrow grade = 0;
                                                                  'N'
                                                                  Υ'
  strcpy(p \rightarrow name, "TONY");
                                                                  '\0'
int main()
{ struct student *sp;
                                                      5000
                                                sp
                                                                 75
                                                                     5000
 sp = (struct student *)
                                                                 Ά'
              malloc (sizeof(struct student));
                                                                 'N'
 sp -> grade = 75;
                                                                 'E'
                                                                 '\0'
 strcpy(sp → name, "JANE");
 printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?
 g(sp);
printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?}
```

## A "ptr to ptr to a struct" is parameter

```
void f(struct student **p)
struct student
                             \{ (*p) \rightarrow \text{grade} = 0;
{ int grade;
                               strcpy((*p) \rightarrow name, "TONY");}
  char name[5]; };
                                                  A-75 Jane
int main()
                                                  B- 0 Tony
                                                  C- Compile error
{ struct student *sp;
                                                  D- Run time error
 sp = (struct student *) malloc (sizeof(struct student));
 sp -> grade = 75;
 \Rightarrow name, "JANE");
 printf("see %s %d\n", sp -> name, sp -> grade); // Output?
 f(&sp);
printf("see %s %d\n", sp -> name, sp -> grade); // Output? }
```

```
void g(struct student **p)
                                                      600
                                                                  (*p)
\{(*p) \rightarrow grade = 0;
                                                 p
  strcpy((*p) \rightarrow name, "TONY");}
int main()
                                                600
                                                sp
                                                      5000
                                                                75
                                                                     5000
{ struct student *sp;
 sp = (struct student *)
                                                                '''
                                                                N
              malloc (sizeof(struct student));
                                                                 4
 sp -> grade = 75;
                                                                4\0°
 \Rightarrow name, "JANE");
 printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?
 g(&sp);
printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?}
```

## A "ptr to ptr to a struct" is parameter

```
void f(struct student **p)
struct student
                          { struct student *fp;
{ int grade;
                           fp = (struct student *) malloc (...));
  char name[5]; };
                           fp \rightarrow grade = 0;
                            strcpy(fp \rightarrow name, "TONY");
int main()
                            *p = fp; }
{ struct student *sp;
 sp = (struct student *) malloc (sizeof(struct student));
 sp -> grade = 75;
                                                  B- 0 Tony
                                                  C- Compile error
 \Rightarrow name, "JANE");
                                                  D- Run time error
 printf("see %s %d\n", sp -> name, sp -> grade); // Output?
 f(&sp);
printf("see %s %d\n", sp -> name, sp -> grade); // Output? }
```

```
void g(struct student **p)
                                                         fp
                                                                                 800
                                                               800
                                                                             0
{ struct student *fp;
                                                                            'T'
 fp = (struct student *) malloc (...));
                                                                            'O'
 fp \rightarrow grade = 0;
                                                              600
                                                                            'N'
                                                         p
  strcpy( fp → name, "TONY");
                                                                            'Y'
                                                                            '\0'
*p = fp; }
                                                        600
int main()
                                                              50000
                                                        sp
                                                                           75
                                                                                5000
{ struct student *sp;
                                                                           ľ
 sp = (struct student *)
                                                                           Ά'
                                                                           'N'
              malloc (sizeof(struct student));
                                                                           'E'
 sp -> grade = 75;
                                                                           '\0'
 strcpy(sp \rightarrow name, "JANE");
 printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?
 g(&sp);
 printf("see %s %d\n", sp \rightarrow name,sp \rightarrow grade); // Output?}
                                                                               71
```

#### Structs & Pointers

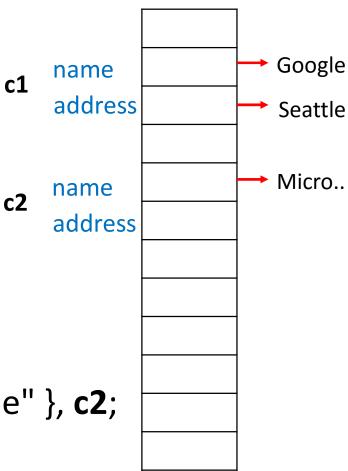
A struct can contain pointers as member variables

```
struct company
```

```
{ char *name;
 char *address;}
```

```
struct company c1 = {"Google", "Seattle" }, c2;
c2.name = "Microsoft";
```

// Are these modifiable? How can we make them modifiable?



#### Structs & Pointers

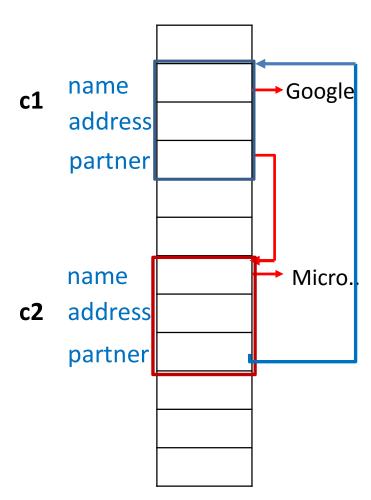
A struct can **not** be nested within itself, but may contain **pointers** to **structs** of their **own type**!

```
struct company
{ char *name;
  char *address;
  struct company *partner;}
struct company c1, c2;
c1.name = "Google"; c1.partner = &c2;
c2.name = "Microsoft"; c2.partner = &c1;
```

#### **Structs & Pointers**

#### struct company

```
{ char *name;
  char *address;
  struct company *partner;}
```



#### struct company c1, c2;

```
c1.name = "Google"; c2.name = "Microsoft";
c1.partner = &c2;
c2.partner = &c1;
```